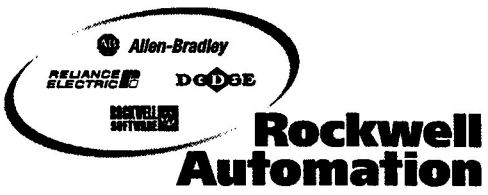
RELIANCE ELECTRIC

# AUTOMAX
# Pocket Reference




Bringing Together Leading Brands in Industrial Automation

The information in this user's manual is subject to change without notice.

**DANGER**

**THIS MATERIAL IS NOT INTENDED TO PROVIDE OPERATIONAL INSTRUCTIONS. QUALIFIED ELECTRICAL PERSONNEL MUST READ AND UNDERSTAND THE APPLICABLE INSTRUCTION MANUALS IN THEIR ENTIRETY PRIOR TO INSTALLING, ADJUSTING, OPERATING, AND SERVICING THIS EQUIPMENT. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN SEVERE BODILY INJURY OR LOSS OF LIFE.**

# Table of Contents

# Introduction

The equipment described in this document is manufactured by Reliance Electric Industrial Company.

This document is designed to serve as a quick reference for AutoMax systems. It includes information about programming, hardware module port and register descriptions, and a list of status and error codes. This document does not take the place of instruction manuals that describe the individual hardware modules, the programming languages, and the AutoMax Executive software. For specific information, you must refer to the following instruction manuals:

- J-3675 AutoMax ENHANCED BASIC LANGUAGE INSTRUCTION MANUAL
- J-3676 AutoMax CONTROL BLOCK LANGUAGE INSTRUCTION MANUAL
- J-3677 AutoMax LADDER LOGIC LANGUAGE INSTRUCTION MANUAL
- J-3616 KERMIT COMMUNICATIONS SOFTWARE INSTRUCTION MANUAL
- J-3618 NORTON EDITOR INSTRUCTION MANUAL
- J-3630 ReSource AutoMax PROGRAMMING EXECUTIVE INSTRUCTION MANUAL VERSION 1.0
- J-3684 ReSOURCE AutoMax PROGRAMMING EXECUTIVE INSTRUCTION MANUAL VERSION 2.0
- J-3636 COMMON MEMORY MODULE INSTRUCTION MANUAL
- J-3649 AutoMax CONFIGURATION TASK INSTRUCTION MANUAL
- J-3650 AutoMax PROCESSOR MODULE INSTRUCTION MANUAL
- J-3661 ReSource AutoMax SOFTWARE LOADING INSTRUCTIONS
- J-3750 ReSource AutoMax PROGRAMMING EXECUTIVE MANUAL VERSION 3.0
- S-3006 DISTRIBUTED POWER DC DRIVE CONFIGURATION AND PROGRAMMING
- IEEE 518 GUIDE FOR THE INSTALLATION OF ELECTRICAL EQUIPMENT TO MINIMIZE ELECTRICAL NOISE INPUTS TO CONTROLLERS
- Your personal computer and DOS operating system manual(s)
- Other instruction manuals applicable to your hardware configuration

# Organization of this Reference Manual

This reference manual is divided into four sections as follows:

1. Status and error codes organized by where they appear.
2. Hardware module reference.
3. Programming reference (BASIC, Control Block, PC/Ladder Logic, and Norton Editor).
4. Appendices:
   - A — ASCII conversion chart
   - B — Decoding hexadecimal addresses of bus errors
   - C — Summary of common DOS commands
   - D — Summary of Windows commands

# Section I
# Status and Error Codes

# Status and Error Codes

Status and error codes can be used to diagnose the state of the system. These codes indicate conditions or symptoms only, not necessarily the root cause of any problem. Status codes are those that indicate a condition that does not necessarily signify an error. For example, code "LO" means that the runbase, or operating system, needs to be loaded onto the rack. Error codes indicate a hardware or software problem in the system. Error code 7.2. displayed on the Processor module, for example, means that the system received a spurious interrupt during runbase booting. Depending upon its severity, an error can have three different effects on the system when it is running:

a) "ERROR" is listed in the task status field on the ON LINE menu of the AutoMax Programming Executive software. The task that caused the error, as well as any other tasks that are running, continue to run. In most cases, the error log for the task will show an error code and a line number indicating where the error occured.

Example: Error code 759 in the error log; 0 is being used as a divisor in an application task.

b) All tasks in the rack are stopped. The Processor(s) can still respond to commands from the personal computer.

Example: Error code 17 on Processor LEDs; a task has attempted to read or write to an invalid address.

c) All tasks in the rack are stopped and all Processors in the rack are shut down. The Processor(s) cannot respond to commands from the personal computer.

Example: Error code F9 on Processor LEDs; the Processor runbase, or operating system, is not functioning correctly.

**Location of Status and Error Codes**

Status and error codes can appear in three places. Hardware status and error codes are usually displayed on the seven-segment LEDs on faceplates. Codes found on module faceplates are read top to bottom. Note carefully that some of the codes include decimal points, which are easy to overlook. Codes "3.0." and "30", for example, are not the same.

Application software errors that occur while tasks are running are displayed in the error log maintained for each task by the AutoMax Executive software. The error log is accessed through the INFO LOG option from the ON LINE menu. Where applicable, the error log will display the line number of the statement that caused the error, or the hexadecimal bus address where the error occurred. To decode the bus error address, see Appendix B.

Errors that occur when tasks are compiled, loaded to the Processor, or saved from the Processor are written to .LOG files if the LOG option is selected. They are also displayed on the screen during these operations unless the NOSCREEN option is selected.

Another status indicator is the single green LED labeled "OK" found on some modules, such as the AutoMax Processor and the Common Memory module. This LED is either on or off and signifies whether the module is functioning properly when power is applied to the system. Note that in the case of the Common Memory module, this LED will also be off if the module is not in slot zero of the rack.

**Organization of Status and Error Codes**

In this reference manual, status and error codes are listed together, organized in numerical and, where applicable, alphabetical order according to where they appear. Codes have been assigned in logical categories, e.g., BASIC Run Time Errors, and are so labeled. Specific troubleshooting procedures are usually included at the end of each section. A description of general troubleshooting procedures precedes the list of status and error codes.

# General Troubleshooting Procedures

**DANGER**

**ONLY QUALIFIED ELECTRICAL PERSONNEL WHO ARE FAMILIAR WITH THE CONSTRUCTION AND OPERATION OF THE EQUIPMENT AND HAZARDS INVOLVED SHOULD INSTALL, ADJUST, OPERATE, AND/OR SERVICE THE EQUIPMENT. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN SEVERE BODILY INJURY OR LOSS OF LIFE.**

**WARNING**

**INSERTING OR REMOVING HARDWARE OR ITS CONNECTING CABLES MAY RESULT IN UNEXPECTED MACHINE MOTION. POWER TO THE MACHINE SHOULD BE TURNED OFF BEFORE INSERTING OR REMOVING HARDWARE OR ITS CONNECTING CABLES. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY.**

The following are intended to be general guidelines on troubleshooting. For specific information, turn to the individual instruction manuals for AutoMax hardware and software.

1. Check for error codes on hardware module faceplates and the error log for all tasks. If the information given is specific enough, e.g., the error log lists the line number where the error occurred, correct the problem indicated.

2. With power off, try removing and then re-inserting any suspect module.

3. With power off, confirm that each module with cable connections has the proper connector attached and that the connections are tight.

4. Check the voltage coming into the power supply through the rack connections labeled "188" and "189". It should be 115VAC.

5. Check the LEDs on I/O modules. Make certain their state (off or on) corresponds to the state of the device to which they are connected for input modules. For output modules, make certain the state of the LED corresponds to the state of the variable used to reference the point.

# General Troubleshooting (continued)

6. Check the voltage on each terminal strip for wiring problems.

7. Try to read inputs or write to outputs on suspect modules through the I/O Monitor in the AutoMax Executive software. If the problem is in a remote rack, try to read/write to the Remote I/O Communications module, M/N 57C416. If you can communicate with the Remote I/O Communications module, try accessing other modules in the remote rack.

8. Systematically replace modules, always replacing the most recently removed module before going on to the next. The last module removed before the problem was corrected is the one that caused the problem.

Basic order of replacement for local racks:

suspect module
Processor(s)
power supply
backplane

Basic order of replacement for remote racks:

suspect module in remote rack
Remote I/O Communications module in remote rack
Remote I/O Communications module in local rack
Processor(s) in local rack

9. Verify that the physical configuration of the system is reflected correctly in the configuration for the rack. Verify that application tasks reference the correct I/O points.

Note that one condition always signifies that a module is malfunctioning and should be replaced. If power to the rack is on and both the single green LED labeled "OK" and the seven-segment LEDs are off on any of the following modules, the module is malfunctioning and should be replaced:

AutoMax Processor module
DCS 5000 Processor module (M/N 57C407)
Network module (M/N 57C404)
Remote I/O module (M/N 57C416)
Modbus Interface module (M/N 57C414)
AutoMate Interface module (M/N 57C417)
Allen-Bradley Interface module (M/N 57C418)
Toledo Scale Interface module (M/N 57C428)

# Error Codes Displayed on AutoMax Processor Module (M/N 57C430, M/N 57C431 or M/N 57C435)

**Processor Overload**

| | |
|---|---|
| 00 | CPU Overload |

Corrective action: move one or more application tasks to other Processor modules in the rack.

**Power-Up Diagnostics**

The following error codes are displayed while the Processor module performs power-up diagnostics.

| | |
|---|---|
| 0.0. | EPROM failed |
| 0.1.–0.3. | Bad CPU |
| 0.4. | Internal bus error test failure |
| 0.5. | Parity test failure |
| 0.6. | External bus error test failure |
| 0.7. | Processor in the wrong slot |
| 1.0.–1.6. | RAM failure |
| 2.0. | I/O protection failure |
| 2.1. | PIO failed |
| 2.2. | PC accelerator failed |
| 2.3. | 8253 timer/counter failed |
| 2.4. | SIO failure |
| 2.5. | Communications interrupt failed |
| 2.6. | SIO interrupt failed |
| 2.7. | 8253 timer/counter interrupt failed |
| 2.8. | Local watchdog failed |
| 3.0. | Bad backplane |
| 3.1. | Multibus parity test failure |
| 4.0.–4.5. | Common memory RAM failure |
| 4.6. | Common memory system watchdog failure |
| 5.0. | Processors with incompatible EPROMs in rack |

Corrective action: replace the Processor, or replace the Common Memory module if error codes 4.0.–4.6. remain on.

**Runtime Errors**

| | |
|---|---|
| 02 | Invalid task or configuration checksum |

Corrective action: replace the Processor module.

# AutoMax Processor Error Codes (continued)

**Runbase Booting**

The following status/error codes may be displayed while you load the runbase, i.e., operating system, onto the Processor module(s). All of the following codes except 6.5. apply to the top port of the Processor module, labeled "Programmer/Port B".

5.1. Incompatible runbase downloaded
6.0. Unexpected interrupt on upper port of Processor
6.1. Parity error
6.2. Receiver overrun
6.3. Framing error
6.4. Serial port fatal error
6.5. Illegal interrupt on lower port of Processor
6.6. Transmit interrupt error
6.7. Runbase integrity lost
6.8. Bad runbase checksum
6.9. Transmit buffer error
7.0. Multi-Processor runbase download in progress
7.1. Disconnect time-out during download
7.2. Spurious interrupt received

Corrective action: 6.3. may be caused by attempting AutoMax ON-LINE functions before the runbase is loaded onto the Processor module(s) in the rack. In this case, exit the ON-LINE menu and download the runbase. 7.0. is a status message only. For all other error codes, cycle power and try to load the runbase again.

**Loading the Runbase Over the Network**

8.0. Bad message length specified for network message
8.1. Bad destination drop
8.2. Transmitting drop inactive
8.3. Destination port unallocated
8.4. Destination port busy
8.5. Did not receive expected response
8.6. Spurious network interrupt received
8.7. Network message is being transmitted

Corrective action: 8.0. and 8.1. are caused by a failed Processor in the left most slot. For 8.2., check the coax cable; then try replacing the Network module. For 8.3. – 8.5., check the destination Network module, then the leftmost Processor in the destination rack. For 8.6. and 8.7., cycle power and try to load the runbase again.

**Miscellaneous Processor Errors**

8.8. Processor failure

Corrective action: replace Processor module.

# AutoMax Processor Error Codes (continued)

**STOP ALL Error Codes**

The following hardware and software error codes cause all tasks running in the rack to stop.

| | |
|---|---|
| 10 | Event count underflow<br>- too many WAITs (max. 32768)<br>- not enough SETs (BASIC tasks) |
| 11 | Event count overflow<br>- too many SETs (max. 32767)<br>- not enough WAITs (BASIC tasks) |
| 12 | Hardware event time-out<br>- interrupt time exceeded programmed time-out limit in a Control Block task |
| 13 | Runbase boot error<br>- a check on the runbase failed |
| 14 | Processor overlap limit exceeded<br>- ran out of processing capacity (time) |
| 15 | External watchdog time-out detected<br>- green LED out on another Processor |
| 17 | Address error detected<br>- caused by a read/write to an invalid address |
| 18 | Spurious interrupt or hardware failure |
| 19 | Power failure detected |
| 1A | Watchdog on this Processor failed |
| 1b | Hardware event count limit exceeded<br>- too many interrupts set without being acknowledged<br>- program too long<br>- collective scans too fast |
| 1C | Illegal instruction detected<br>- runbase software fault<br>- bad Processor module<br>- bad EPROMs |
| 1d | Privilege violation detected<br>- runbase software fault<br>- bad Processor module |
| 1E | Un-implemented instruction detected<br>- runbase software fault<br>- bad Processor module |
| 1F | Illegal interrupt detected<br>- runbase software fault<br>- bad Processor module |
| 31 | Bus error<br>- attempt to access invalid address |
| 32 | Define channel error<br>- problem in application software |
| 33 | Define scan error<br>- hardware fault |

# AutoMax Processor Error Codes (continued)

| | |
|---|---|
| 34 | Memory integrity lost<br>- hardware fault |
| 35 | D-C drive CML block initialization error |
| 36 | Communication between drive Processor and I/O controller lost |
| 37 | D-C drive I/O controller run-time board error<br>- hardware fault |
| 38 | UDC module generated a STOP ALL |
| 39 | UDC module interrupt allocation failed |

Corrective action: correct the problem in application software. Try to reset by cycling power and re-loading configuration and application tasks. Replace the Processor module. For error codes 31 and 37, see Appendix B in this manual. For 38, examine the error logs for all UDC tasks in the rack. For 39, cycle power to the rack and re-load configuration and application tasks.

**BASIC STOP ALL Error Codes**

The following error codes are caused by problems in BASIC tasks and cause all tasks running in the rack to stop.

| | |
|---|---|
| 40 | Too many RETURNs from GOSUBs (or RETURN without GOSUB) |
| 41 | Illegal jump into a FOR loop |
| 42 | NEXT statement does not match current FOR |
| 43 | Invalid START EVERY statement |
| 44 | Invalid EVENT statement |
| 45 | STOP statement executed in application software (causes a STOP ALL and clears all I/O) |
| 46 | SET or WAIT attempted with no event definition |
| 47 | Task stack overflow |
| 48 | GOSUBs not balanced at END statement |
| 49 | Insufficient space for channel buffer |
| 4A | Attempted to execute undefined opcode |
| 4B | Attempted to execute non-executable opcode |
| 4C | Attempted to execute illegal opcode |
| 4D | RESTORE to non-DATA statement line number |
| 4E | Attempted to take square root of a negative number |
| 4F | Attempted RESUME without being in an ON ERROR handler |

Corrective action: correct the problem in application software; for 47, check for PUT on a closed part; for 4A, check use of Ethernet functions in standard operating system.

# AutoMax Processor Error Codes (continued)

**Multibus™ and Processor Bus STOP ALL Error Codes**

| | |
|---|---|
| 50 | On-board parity error |
| 51–54 | On-board bus error or access violation |
| 55 | Multibus parity error during read access |
| 56–58 | Multibus access violation or bus error |
| 60 | Network interrupt allocation failed |
| 61 | Network receive queue overflow |
| 62 | Network transmit queue underflow |

Corrective action: reset by cycling power and re-loading configuration and application tasks. If the small green LED labeled "OK" on the Processor module faceplate is off, replace the Processor module. For error 58, check for incorrect IOWRITE statement. Correct any incorrect accesses in application software. Systematically replace hardware modules. For error codes 50-57, if none of the above correct the problem, try replacing the Rack/Backplane assembly. Also see Appendix B in this manual.

**AutoMax Drive-Related Error Codes**

The following error codes indicate a power circuit or external drive system fault. After correcting the problem, reset the Processor module by cycling power and re-loading the configuration task and application tasks to clear the error code. Note that these error codes also appear in the Error Log for the Processor.

| | |
|---|---|
| 80 | Instantaneous overcurrent fault<br>– armature current exceeded IOC_THRESH value in CML task |
| 81 | Line sync loss fault |
| 82 | Tach loss fault<br>– 40% armature phase angle with less than 5% tach feedback |
| 83 | Overspeed/overvoltage fault<br>– CML task OSV_FDBK exceeded OSV_THRESH number |
| 84 | Hardware overspeed fault<br>– Drive Analog module potentiometer setting exceeded by input voltage |
| 85 | External IET fault<br>– external fault input triggered |
| 86 | Phase rotation fault<br>– incorrect phasing |
| 87 | Shorted SCR detected in power module |

Corrective action: troubleshoot power circuit and external drive system.

# AutoMax Processor Error Codes (continued)

**Configuration Error Codes**

The following error codes usually indicate a discrepancy between the actual hardware configuration and the I/O definitions in the configuration for the rack.

| | |
|---|---|
| E0 | TASK specified in configuration uninstalled, at wrong priority, of wrong type, on wrong Processor module; wrong spelling of TASK |
| E1 | Invalid configuration, configuration not successfully downloaded |
| E2 | I/O referenced in configuration is missing. |
| E3 | I/O referenced in configuration is missing. Invalid configuration, configuration not successfully downloaded. |
| E4 | Error building task, insufficient memory in Processor module. Invalid configuration, configuration not successfully downloaded. |
| E5 | Error building task, insufficient memory in Processor module. Invalid configuration, configuration not successfully downloaded. |
| E6 | I/O referenced in configuration is missing. Error building task, insufficient memory in Processor module. |
| E7 | Invalid configuration, configuration not successfully downloaded. I/O referenced in configuration is missing. Invalid configuration, configuration not successfully downloaded. |
| E8 | Error installing application task, common symbol could not be resolved, insufficent memory in Processor module. |
| E9 | Error installing application task, common symbol could not be resolved, insufficient memory in Processor module. Invalid configuration, configuration not successfully downloaded. |
| EA | Error installing application task, common symbol could not be resolved, insufficent memory in Processor module. I/O referenced in configuration is missing. |
| Eb | Error installing application task, common symbol could not be resolved, insufficient memory in Processor module. I/O referenced in configuration is misisng. Invalid configuration, configuration not successfully downloaded |

# AutoMax Processor Error Codes (continued)

| | |
|---|---|
| EC | Error building task; error installing application task, common symbol could not be resolved, insufficient memory in Processor module. |
| Ed | Error building task; error installing application task, common symbol could not be resolved, insuffient memory in Processor module. Invalid configuration not successfully downloaded. |
| EE | Error building task; error installing application task, common symbol could not be resolved, insuffient memory in Processor module. I/O referenced in configuration is missing. |
| EF | Common variable forced by another Processor module |

Corrective action: verify that the configuration correctly describes the physical configuration of the system and the tasks installed on the Processor module(s). Reset by cycling power and re-loading the configuration and application tasks. For error code EF, un-force the variable and do a STOP ALL CLEAR from the AutoMax ON-LINE menu.

**Fatal Errors**

The following error codes usually indicate that the runbase is not functioning correctly. If any of these error codes appear, the configuration task and all application tasks are deleted from the Processor module.

| | |
|---|---|
| F0–F9 | Fatal error |
| FA-FF | Fatal error |

Corrective action: cycle power. Re-load the configuration task and all application tasks. Replace the Processor module.

**Informational Messages**

The following codes signify a particular condition, not necessarily an error.

| | |
|---|---|
| dd | This Processor module has successfully completed power-up diagnostics and is waiting for other Processor modules to complete their diagnostics |
| LO | The runbase needs to be loaded onto the rack |
| b0 | Rack configuration is being validated |
| d0 | Application task installation in progress |

# AutoMax Processor Error Codes (continued)

| | |
|---|---|
| d1 | Waiting on synchronizing event (in a rack with multiple Processors) |
| d2 | Waiting on mutual exclusion lock (in a rack with multiple Processors) |

Corrective action for b0 and d0 that do not change or disappear: re-load configuration and application tasks.

# Error Codes Displayed on Network (M/N 57C404) and Remote I/O (M/N 57C416) Communication Modules

0 CPU failed power-up diagnostic
1 EPROM failed power-up diagnostic
2 RAM failed power-up diagnostic
3 CTC failed power-up diagnostic
4 SIO failed power-up diagnostic
5 DMA runtime failure; message transmit timeout
6 Dual Port memory failed power-up diagnostic
7 Memory management unit failed power-up diagnostic
8 or .8 Bad EPROMs
9 PIO port failed power-up diagnostic
A Invalid drop number. This only occurs if the drop number on the thumbwheel switches is greater than 55 on a Network Module or greater than 7 on a Remote I/O Module.
b Watchdog failed power-up diagnostic
C Board not communicating. If board is a Master drop, no other drops are functional on the Network. If board is a Slave drop, it is not receiving any messages from the Master. This fault code is reset whenever the line goes active.
d System (backplane) watchdog failure; Processor module(s) went down; module is operational but will not transmit or receive data until the watchdog is reset.
E Power failure. This code is normally present from the time that a low voltage is detected until power is completely lost.

Corrective action (0 through 9, b): replace Network or Remote I/O Module.

## Error Codes Displayed on AutoMax Drive Controller Module (B/M 57406)

0 CPU diagnostic
1 EPROM diagnostic
2 RAM diagnostic
3 PIO (8255) diagnostic
4 Counter (8253) diagnostic
5 Watchdog timer diagnostic
6 Interrupt structure diagnostic
7 Analog board diagnostic
8 or .8 Bad EPROMs
C Communication not active with processor (normal code when CML task stopped)
d Spurious interrupt received — fatal
E Power fail interrupt received — fatal
F Watchdog timeout detected — fatal

Corrective action: replace Drive Controller module. Replace Drive Analog module. For code 7, check power supply voltage. For E, check power supply or backplane. For C, start the CML task.

## Error Codes Displayed on AutoMate® (M/N 57C417) Modbus™ (M/N 57C414), Allen-Bradley™ (M/N 57C418), and Toledo Scale™ (M/N 57C428) Interface Modules

0 CPU failed power-up diagnostic
1 EPROM failed power-up diagnostic
2 RAM failed power-up diagnostic
3 CTC failed power-up diagnostic
4 SIO failed power-up diagnostic
5 DMA failed
6 Dual Port memory failed power-up diagnostic
7 Memory management unit failed power-up diagnostic
8 or .8 Bad EPROMs
9 PIO port failed power-up diagnostic
A Invalid device number. This only occurs if the device number on the thumbwheel switches is 00.
b Watchdog failed power-up diagnostic

C Communication line status. Displayed only if the link has not been configured by the application program.
d System (backplane) watchdog failed; a Processor module(s) went down. Module is operational but will not transmit or receive data until the watchdog is reset.
E Power failure. This code is normally present from the time that a low voltage is detected until power is completely lost.

Corrective action (0 through 9, b): replace the module. For M/N 57C417, C may indicate that the R-NET Gateway is not responding.

## Error Codes Displayed on Remote I/O Head (M/N 45C33 or 57C330)

0 CPU failed power-up diagnostic
1 EPROM failed power-up diagnostic
2 RAM failed power-up diagnostic
3 CTC failed power-up diagnostic
3. CTC runtime failure
4 SIO failed power-up diagnostic
5 DMA failed
6 I/O device interface failed
8 or .8 Bad EPROMs
9 PIO port failed power-up diagnostic
A Invalid device number. This only occurs if the device number on the thumbwheel switches is not between 1 and 7 inclusive.
b Watchdog failed power-up diagnostic
C Communication line status. This means that the Head is not receiving message from the Master. This fault is reset whenever the line goes active.
E Power failure. This code is normally present from the time that a low voltage is detected until power is lost.

Corrective action (0 through 9, b): replace the Remote I/O Head.

# Error Codes Displayed on MaxPak III High Speed Link (M/N 57C424)

0 CPU failed power-up diagnostic
1 EPROM failed power-up diagnostic
2 RAM failed power-up diagnostic
3 CTC failed power-up diagnostic
5 DMA failed
6 Dual Port memory failed power-up diagnostic
7 Memory management unit failed power-up diagnostic
9 Parallel I/O port failed power-up diagnostic
b Watchdog failed power-up diagnostic
C Communication line status. Displayed only when no messages are received from the MaxPak III.
d System (backplane) watchdog failed. Board is operational but will not transmit or receive data until the watchdog is reset.
E Power failure. This code is normally present from the time that a power failure is detected until power is lost.
.3 CTC run time failure.
.4 SIO run time failure.
.r Incompatible software revisions (MaxPak III – 57C424)

Corrective action (0 through 9, b): replace the MaxPak III High Speed Link.

# Error Codes Displayed on R-Net Processor Module (M/N 57C429)

(Code listed below is the "COM ER" LED pattern read top to bottom)

0001 EPROM checksum error
0010 Scratch pad RAM failure
0011 Dual port memory failure
0100 Node number witch setting >250
0101 Communication circuit failure
0111 Local watchdog interrupt
1010 Software test (negative buffer length detected)
1011 Software test (Message Length = 0 detected)
1100 Unused interrupt detected
1101 Attempted word access on odd byte
1110 System watchdog interrupt
1111 Bus error

Corrective action: for errors 1-3, and 5, replace the R-Net Processor module; for error 4, correct switch setting on module; for errors 9-15, cycle power to attempt to clear the error.

# Compile, Load and Save Error Codes Displayed in .LOG Files and on the Screen

The error codes in this section are generated when tasks are compiled, saved from the Processor, or loaded onto the Processor with the LOG option selected. They are displayed in the corresponding .LOG files and on the screen unless the NOSCREEN option is selected.

**Control Block Error Codes**

257 Bad Control Block statement format
258 Unrecognized name for Control Block
259 Missing END statement in Control Block task
261 Variable used in Control Block not defined
262 Bad literal value for KI, KP, or KD
263 Bad WLD * KP / C value
264 Bad literal value for DEAD_BAND, MAX_CHANGE, or LOOP_TIME
265 Invalid data type for literal in Control Block
266 Incomplete input pairs or input/output pairs in a Control Block
267 Bad SCALE, REQUIRED_SAMPLES, or MAX_COLUMNS value
268 Bad specification for array in Control Block
269 Control Block not the only statement for that line number
270 CML specified literal field out of range
271 SCAN_LOOP block not allowed with CML block
272 Integer literal field too large
273 Invalid parameter keyword in Control Block
274 Calculated K value out of range
275 Literal symbol too long
276 Required Control Block field missing
277 Required Control Block literal missing
278 Control Block field must be literal
279 Control Block field must be variable
280 Non-contiguous inputs, input pairs or input/output pairs in Control Block
281 Missing SCAN_LOOP block in Control Block task
282 Signed boolean or numeric literal not allowed
283 WLD value out of range
284 Invalid value for Lead Lag W
285 Invalid value for WM
286 Invalid value for WLD
287 Word size out of range
288 Array specified has too many subscripts
289 Integer literal > 24 bits (can't be accurately converted to real)
290 Invalid value for MAX_INPUT

291 More than 1 CALL SCAN_LOOP in a Control Block task
292 Fast floating point overflow
293 Fast floating point underflow
294 Fast floating point divided by 0
295 Meaningless tangent argument
296 Minimum number of inputs or outputs not programmed
297 Invalid data type for variable in Control Block
298 Parameter keyword previously defined in Control Block
299 Data structure symbol name too long
300 Data Structure requires more than maximum storage
301 Number of inputs/outputs greater than data structure definition
302 Duplicate definition or incorrect data structure type
303 Invalid Control Block Mode specified

**IODEF, RIODEF, NETDEF, RNETDEF, MODDEF Error Codes (Configuration Errors)**

306 Bad IODEF statement format
307 IODEF address must not be odd
308 Bad IODEF variable type
309 IODEF hex address too large
310 Invalid bit number specification in RIO/NET DEF
311 Invalid literal in RIO/NET DEF
312 Missing master slot specification in RIODEF
313 Bad literal in IODEF
314 Missing bit field specification
315 Missing slot specification
316 Bad RIO/NET DEF statement format
317 Missing drop specification
318 Bad MODDEF statement format
319 Bad GATEWAY register specification
320 Bad RNETDEF statement format
321 Bad RNETDEF register specification
322 Invalid variable data type in GATEWAY definition
323 Bad ABDEF statement format
324 Bad file specification in ABDEF statement
325 Bad boolean literal specification

**Function Call Error Codes**

336 Invalid function call format
337 Incorrect number of parameters in function call
338 Bad parameter data type in function call or bad array subscript
339 Parameter symbol not defined
340 Variable must be simple (not array variable)
341 Invalid function parameter
342 Invalid function expression
343 Bad function variable
344 Bad array; must be 1 dimension (integer)
345 Bad BLOCK_MOVE variable
346 Variable in function call not defined as COMMON

**Insufficient Memory Error Codes**

356 Insufficient memory to compile array
357 Insufficient memory to compile FOR statement
358 Insufficient memory to build symbol table
359 Insufficient symbol table memory
360 Object code buffer overflow
361 Opcode position overflow; statement too long
362 No more user stack
363 No more program stack
364 No more type stack; expression too long
365 No more operator stack; expression too long
366 No more memory to link object code buffer

**FOR-NEXT Error Codes**

376 FOR control variable cannot be a tunable variable
377 NEXT control variable does not match FOR control variable
378 Control variable must be simple (not array) variable
379 Invalid data type on control variable in FOR statement
380 Bad FOR statement format
381 Invalid statement type following THEN in IF statement
382 Missing expected THEN
383 Invalid data type for expression in FOR statement
384 Missing corresponding FOR statement
385 FOR loops nested too deep

**OPEN, CLOSE, INPUT, PRINT Error Codes**

396 Bad device name — PORTA or PORTB only
397 Bad logical file number specification
398 Bad device name for OPEN statement
399 Bad baud rate in OPEN SETUP parameter
400 Invalid device specification
401 Bad OPEN statement format
402 Duplicate logical file number
403 Invalid CLOSE statement format
404 Invalid device name
405 Missing expected print field
406 Specified file has not been defined (no OPEN)
407 Device must be accessed by OPEN first
408 Invalid data type for PRINT USING format
409 Bad PRINT USING format
410 Specified format field width too wide
411 Cannot have PRINT USING with channel
412 Bad GET statement format
413 Bad PUT statement format
414 Bad INPUT statement format
415 Cannot close a channel
416 Cannot GET from a channel
417 Cannot PUT to a channel
418 Bad SETUP specification in OPEN device
419 Open device attempted on a channel

**START, WAIT, DELAY, EVENT Error Codes**

426 Invalid time units specification
427 Missing DELAY expression
428 Bad START EVERY statement
429 Bad WAIT statement format
430 Invalid event name
431 Bad EVENT statement format
432 Bad time units in START statement
433 Delay time units must be integer
434 Duplicate event name
435 Missing start interval
436 Missing event definition

**Channel I/O Error Codes**

446 Missing DEPTH parameter on OPEN CHANNEL FOR INPUT
447 Bad OPEN CHANNEL format
448 Bad channel template in OPEN statement
449 Invalid DEPTH specification for OPEN CHANNEL
450 INPUT/PRINT reference does not match channel template
451 Not assigned
452 Channel template too large
453 Channel packet too large
454 Channel was opened for input but output was attempted
455 Channel was opened for output but input was attempted

**Array Error Codes**

466 Array requires more than maximum storage
467 Bad array subscript
468 Number of subscripts does not match definition
469 Not assigned
470 Missing array dimension
471 Too many array subscripts

**Miscellaneous Compiler Error Codes**

486 Missing delimiter
487 Missing equal sign "="
488 Missing left parenthesis "("
489 Missing right parenthesis ")"
490 Missing expected comma "," or semicolon ";"
491 Missing line number
492 Invalid line number
493 Line number out of range (must be 1 to 32767)
494 Invalid data type mixing in expression
495 Invalid variable type
496 Variable name same as reserved symbol
497 Variable name too long
498 Missing variable name
499 Variable name too long
500 Invalid subscripted variable
501 Invalid variable specified in READ statement
502 Missing variable definition

503 Invalid statement terminator; expecting EOS
504 Task must be a CONFIGURATION task
505 Missing operand (symbol or literal)
506 Missing arithmetic/relational operator
507 Not a valid statement for this task type
508 Invalid integer expression for ON GOTO
509 Invalid ON GOTO statement format
510 Missing expected TO
511 Expected expression not found
512 Missing expected line number
513 Invalid boolean expression
514 Invalid tunable statement definition
515 Symbol already defined; duplicate definition
516 Invalid data type for a tunable variable
517 Tunable variable ranges are inconsistent
518 Undefined variable or statement not permitted in this type of task
519 Invalid tunable variable definition format
520 Tunable cannot be array or left side of equal
521 Missing expected variable
522 DATA statement not first statement for this line number
523 Not assigned
524 Overflow in ASCII to binary integer conversion
525 Numeric literal too large
526 Real literal too large
527 Null buffer overflow; statement too large
528 Object buffer overflow; statement too large
529 Expression evaluator; stack integrity lost; expression too long and/or too complex
530 Compiler integrity lost
531 Illegal symbol in REM statement
532 CALL statement not first statement for this line number
533 Task not of type BASIC, Control or Configuration
534 Invalid task statement format
535 Invalid task priority
536 Invalid task name
537 Invalid slot specification
538 Missing string variable in GET statement
539 Illegal on board I/O address specified
540 Bad IOWRITE format
541 Bad IOWRITE option expression
542 Bad IOWRITE value expression
543 Bad IOWRITE address expression
544 REM statement not the first statement on the line
545 Bad ON ERROR statement format
546 Fatal expression evaluation error; no opcode match
547 String literal too large
548 Too many total elements for an array
549 Array variable was referenced as a simple variable
550 Illegal state in expression evaluation; integrity lost
551 Bad expression in SET_MAGNITUDE statement

552 Bad SET_MAGNITUDE statement format
553 Bad variable type in SET_MAGNITUDE statement
554 Invalid TIMEOUT expression in EVENT statement
555 Symbol > 255 characters long; statement too long
556 Bad IF statement transfer line number
557 Invalid characters after the ampersand continuator
558 Remark statement too long
559 Line number out of range
560 Must be 1st statement on the line
561 Symbol is not a variable name
562 Loss of precision in converting real number

**Resolution Error Codes**

656 Line used in RESTORE is not a DATA statement
657 FOR and NEXT variables do not match
658 Insufficient memory to compress object code
659 Object code larger than 32K
660 Stack requirements too large
661 Data structures too large
662 Symbol table integrity lost
663 Insufficient memory for post-compile resolution
664 Line number not resolved
665 Too many symbols
666 No TASK statement in configuration task
667 No symbols in configuration task
668 Duplicate data pointers with same data type; assigned two different variables of the same type to the same register or bit
669 Symbol table too large (too many symbols)
670 Invalid condition; integer literal in BASIC task symbol table
671 Unable to allocate enough space for symbol table
672 Symbol table integrity lost
673 Too many COMMON integers, double integers, booleans used
674 Unable to allocate space for the BASIC runtime structure header
675 Too many LOCAL integers, double integers, booleans used
676 Too many LOCAL integers, double integers, booleans literals used
677 Too many COMMON reals, strings, arrays used
678 Too many LOCAL reals, strings, arrays used
679 Too many OPEN CHANNEL statements
680 Too many arrays used
681 Too many FOR loops used
682 Too many real literals used
683 Too many real tunable variables defined
684 Invalid condition; literal in Configuration task
685 Invalid condition; string literal type in symbol table

686 Offset to real literal in Control Block task > 16 bits
687 Invalid condition; LOCAL variable in CONFIGURATION task
688 Invalid condition; relative symbol number not resolvable
689 Task too large
690 Error opening the object output file
691 Error writing to object output file
692 Task with READ statements but no DATA statements
693 Too many LOCAL integers, double integers, boolean variables used
694 Unable to allocate enough space for object code
695 Undefined Control Block data structure found
696 Error closing source file (disk may be full)
697 Error closing log file (disk may be full)
699 Error attempting to load time/date into object file
700 Object size > 32767 in Control Block task
701 Symbol & data size > 32767 in Control Block task

Corrective action: correct problem in the application software.

# Run Time Error Codes Displayed in the Processor or UDC Error Log

The following error codes are displayed in the error log maintained for each Processor and UDC module by the AutoMax Executive software.

756 Arithmetic integer overflow code
757 Arithmetic real overflow code
758 String concatenate overflow
759 Divide by zero
760 Integer multiply overflow
761 Integer assign overflow
762 Single integer conversion overflow in real to single integer
763 Double integer conversion overflow in real to double integer
764 Real to double conversion yields number > 24 bits
765 String overflow
766 Precision lost in real to integer array element conversion
767 Precision lost in real to double integer array element conversion
768 Precision lost in real to single integer conversion

769 Array subscript out of bounds
770 Requested substring > string
771 DATA type in READ statement does not match DATA statement
772 No more DATA statements
773 Bad line number for RESTORE
774 Overflow in conversion of real to integer of FOR loop control variable
775 Overflow in conversion of real to integer of FOR statement TO value
776 Overflow in conversion of real to integer of FOR statement STEP value
777 Integer > 24 bits in STEP value integer to real conversion
778 Bad IOWRITE
779 Integer control variable overflow in FOR statement
780 Double integer control variable overflow in FOR statement
781 Real control variable overflow in FOR statement
782 Negative delay
783 Delay value too large (0 to 32767)
784 Negative start interval
785 Delay value too large (0 to 32767)
786 Not assigned
787 Hardware event # ticks < 0
788 Hardware event ticks overflow
789 Print buffer overflow; print field too long
790 Device not open properly
791 OPEN with bad device address
792 Device not open for write
793 No stack space for print
794 Device not allocated
795 No buffer for print operation; insufficient memory
796 Fatal print error
797 Device already open
798 Device OPENed different from intended
799 Bad allocate
800 Bad default OPEN
801 Device already closed
802 Device opened as a channel
803 Bad device close; no address
804 Default device not allocated
805 Channel not open
806 Print integer channel overflow
807 Message overflow
808 Unsuccessful channel open
809 Integer > 24 bits in real conversion
810 Real to integer overflow
811 No buffer for GET operation
812 No print buffer
813 Device closed on GET
814 GET attempted to un-opened device; GET not open for read
815 Bad GET operation
816 No buffer for PUT operation
817 No print buffer
818 Device closed on PUT statement

819 PUT attempted on un-opened device; PUT not open for write
820 Unsuccessful PUT operation
821 Device should be open
822 Invalid baud rate
823 Bad SETUP re-configuration
824 Precision out of range
825 Print width too long; printing integer field in PRINT
826 Print width too long; printing integer field in PRINT USING with L/R/C/Z format
827 Negative decimal places
828 Number of decimal points > max precision
829 Width less than zero
830 Field width overflow
831 Requested substring width < zero
832 Requested substring width > maximum
833 No space for requested PRINT USING field
834 String > field width
835 Bad channel depth
836 Device not open
837 Attempted negative square root
838 First substring position specification > string length
839 Not assigned
840 Not assigned
841 Wrong data type input for boolean
842 Another error occurred during execution of ON ERROR routine
843 Could not allocate for write
844 Wrong data type input for string
845 Last substring position < first substring position
846 First substring position specification <= 0
847 Last substring position specification <= 0
848 Rotate count > 31
849 Overflow on absolute value function
850 Not assigned
851 Device open at END
852 Channel not open on input
853 Wrong type for integer
854 Next character after field not legal
855 Bad next character
856 No input channel I/O buffer
857 Not allocated for re-configuration
858 Bad BCD digit
859 Channel already open
860 Wrong token for comma
861 Not open for read
862 No comma between fields
863 Wrong data type input for real
864 No buffer space can be allocated for I/O
865 Not assigned
866 Invalid re-configuration
867 Missing line number
868 Bad device on input
869 Wrong type for double integer
870 No device address
871 Number > 24 bits

872 Not open for write
873 No device address
874 Attempt to execute a null opcode
875 Unbalanced GOSUB-RETURN
876 NEXT does not match loop variable in FOR statement
877 NEXT does not match FOR
878 Bad START statement format
879 Bad hardware event call
880 Undefined opcode
881 Stack overflow
882 No channel buffer space
883 STOP executed
884 Opcode not assigned
885 No event address defined
886 GOSUBs not balanced
887 Bad VAL function conversion
888 BCD output number > 9999
889 Bad bit number in function call
890 Bad option number in function call
891 Invalid GATEWAY transfer call
892 BLOCK_MOVE invalid source parameter
893 BLOCK_MOVE invalid destination parameter
894 BLOCK_MOVE invalid transfer size parameter
895 RESUME without executing ON ERROR statement
956 UDC task tick rates do not match

Corrective action: correct problem in application software. For error code 891, check returned status variable of GATEWAY_CMD_OK@; decode status as follows:

| Decimal | |
|---|---|
| 01 | Illegal function code |
| 02 | Illegal starting register |
| 03 | Illegal data |
| 04 | PC aborted |
| 05 | Not assigned |
| 06 | PC busy |
| 07 | Not assigned |
| 08 | Illegal data in response message |
| 09 | Response timeout error |
| 20 | Dual port address error |
| 21 | Gateway card not found or not accessible |
| 22 | No available Gateway channel |
| 23 | Illegal register number |
| 24 | Illegal number of registers |
| 25 | Illegal command number |
| 26 | Illegal command number/register set |
| 27 | Illegal register number/number of registers |
| 28 | Illegal device address |

**UDC Drive Fault Register Error Codes**

1000 SCR fault
1001 M-Contactor fault
1002 Not used
1003 Sync loss (A-C line voltage)
1004 Instantaneous overcurrent fault
1005 Conduction time out
1006 Field loss fault
1007 Tach loss fault
1008 Broken wire in resolver
1009 Not used
1010 Over-speed trip
1011 Power technology module fault
1012 Not used
1013 Not used
1014 Not used
1015 Fiber optic link com. fault

Corrective action: these errors reflect the status of the drive fault register (A=202, B=1202) on the UDC module. See S-3006 for more information.

**Serial I/O Error Codes**

1064 EIA control (Carrier Detect lost)
1065 Parity error (when enabled)
1066 Overrun error
1067 Framing error

Corrective action: check for noise on the RS-232 cable, problems with the device connected to the Processor, and problems with the Processor itself.

**AutoMax DC Drive (DC_DRIVE_CML INIT) Errors**

| | | legal range |
|---|---|---|
| 2000 | IO_CNTL_SLOT error | 0 to 15 |
| 2001 | DIG_IO_SLOT error | 0 to 15 |
| 2002 | REF_LAG input out of range | 1 to 500 |
| 2003 | PI_W_LEAD input out of range | 10 to 500 |
| 2004 | AG_FACTOR input out of range | 1 to 300 |
| 2005 | OSV_THRESH input out of range | 0 to 32767 |
| 2006 | LIM_BAR input out of range | 100 to 400 |
| 2007 | IOC_THRESH out of range | 100 to 400 |
| 2008 | IOC calculation error (IOC_THRESH>1.8 LIM_BAR) | |
| 2009 | MAX_M_DROPOUT input out of range | .1 to 1.0 |

2010 "Allocate Vector for Regulator" error. (Possible if using 4 Processor Modules and all 4 were using hardware interrupts. All 4 of the bus interrupt lines would be in use, leaving none for the CML task.)

**AutoMax DC Drive Run Time Errors**

2020 REF_RATE input out of range 0<=32767
2021 KP input out of range 0 to 4
2022 TEST_ALPHA input out of range 0 to 180
2023 CC_THRESH input out of range 0 to 32767
2030 M-status would not go TRUE during start
2031 Current would not go to zero within MAX_M_DROPOUT during stop
2032 M-status would not go FALSE during stop
2050 Internal value > 32 bits; input numbers too large
2051 Computed OUTPUT value > 16 bits; input numbers too large, divide by zero
2054 WLG input value out of range
2055 WLD and/or WLG input value out of range
2056 KI input value out of range
2057 KP and/or WLD input value out of range
2058 COLUMN number out of range
2059 COLUMN can not currently be accessed

Corrective action: correct problem in application software.

# Section II
# Hardware Module Reference

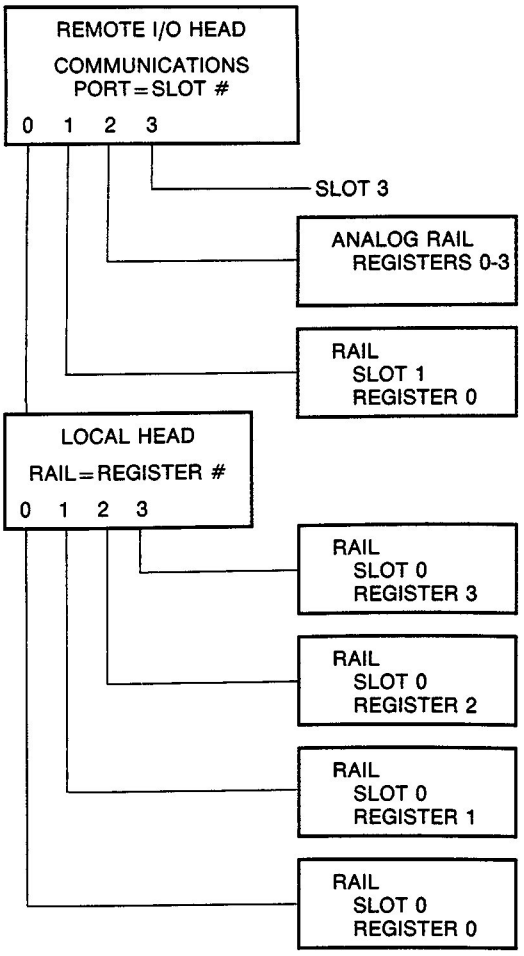
Note the following conventions in register and bit descriptions:

R = read only; must NEVER be written to by the application program
R/W = read/write; can be read or written to by the application program
X = not used

# Remote I/O Head M/N 45C33/57C330

The communications port on the 45C33/57C330 can connect to a single rail or to a Local Head, which can then connect to 1, 2, 3 or 4 rails. Note that input cards and output cards cannot be mixed on the same I/O rail; electronic input cards cannot be used.

**Sample Connection**



# M/N 45C33/57C330 (continued)

**LEDs**

| Color | LED | Description |
|---|---|---|
| yellow | RUN | Remote Head CPU in run |
| green | POWER | Power Supply OK |
| yellow | CPU READY | Remote Head CPU ready to run |
| | | If Rail Fault Indicator is lit: |
| yellow | FAULT MSB | - - ->00 = Rail 0 01 = Rail 1 |
| yellow | FAULT LSB | - - - - -^ 10 = Rail 2 11 = Rail 3 |
| red | RAIL FAULT | Rail Fault Indicator |

**Programming terminal interrogation:**

Command "R" can be entered to read the rails at a communication port with the coax connected and the module operating.

Command "S" for status can be entered with the coax connected and the module operating. The "Last Line Failure" codes displayed are as follows:

'LE' restart due to excessive LINE ERRORS of any kind (RECEIVE TIMEOUT, CRC, OVERRUN, or ABORT).

'UI' restart due to UNEXPECTED INIT request message.

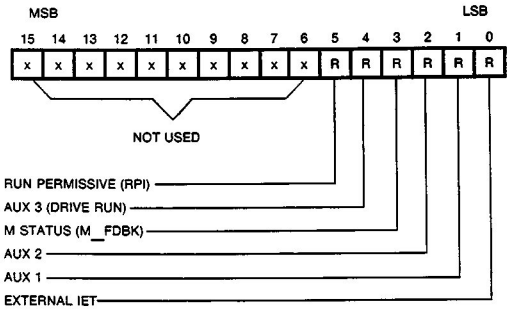
'PF' restart due to a POWER FAIL interrupt which occurs while the line is active. Input power has to remain valid for this code to be stored and displayed.

'Rn' restart due to a port (rail) fault detected during the most recent port I/O update cycle.

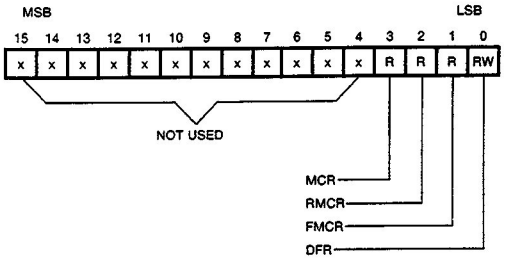
n port number (0-3) which experienced the fault.

# Drive Digital I/O Module
# B/M 57401

**Register 0 (READ Only)**




**Register 1 (WRITE Bit 0 Only)**




**LEDs**

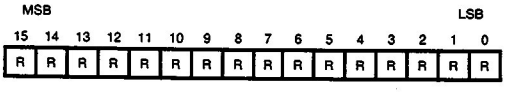
| Color | LED | Description |
|---|---|---|
| yellow | IET | Reg 0, Bit 0 – 12 V IET input |
| yellow | | Not used |
| yellow | AUX1 | Reg 0, Bit 1 – Aux.1 Input |
| yellow | AUX2 | Reg 0, Bit 2 – Aux.2 Input |
| yellow | MFBK | Reg 0, Bit 3 – "M" Cont. Input |
| yellow | DRIVE RUN | Reg 0, Bit 4 – Aux.3 Input |
| yellow | RPI | Reg 0, Bit 5 – Run Perm. Input not used |
| yellow | DFR | Reg 1, Bit 0 – Drive Fault Out |
| yellow | FMCR | Reg 1, Bit 1 – For. MCR |
| yellow | RMCR | Reg 1, Bit 2 – Rev. MCR |
| yellow | MCR | Reg 1, Bit 3 – Main Cntrl Relay (light only) |

# B/M 57401 (continued)

| Term | Signal | Description | Reg. |
|---|---|---|---|
| 1 | 0 V (Common) | IET Common | |
| 2 | 0 to 12 V DC | IET Input | Reg 0, Bit 0 |
| 3 | 0 V (Neutral) | Aux1 Neutral | |
| 4 | 0 to 115 V AC | Aux1 Input | Reg 0, Bit 1 |
| 5 | 0 V (Neutral) | Aux2 Neutral | |
| 6 | 0 to 115 V AC | Aux2 Input | Reg 0, Bit 2 |
| 7 | 0 V (Neutral) | M Fdbk Neutral | |
| 8 | 0 to 115 VAC | M Fdbk Input | Reg 0, Bit 3 |
| 9 | 0 V (Neutral) | Aux3 Neutral | |
| 10 | 0 to 115 V AC | Aux3 Input | Reg 0, Bit 4 |
| 11 | 0 V (Neutral) | RPI Neutral | |
| 12 | 0 to 115 V AC | RPI Input | Reg 0, Bit 5 |
| 13 | N.O. | DFR Contact | Reg 1, Bit 0 |
| 14 | Wiper | DFR Contact | |
| 15 | N.C. | FMCR Contact | Reg 1, Bit 1 |
| 16 | N.O. | FMCR Contact | |
| 17 | Wiper | FMCR Contact | |
| 18 | N.C. | RMCR Contact | Reg 1, Bit 2 |
| 19 | N.O. | RMCR Contact | |
| 20 | Wiper | RMCR Contact | |

# 115 V AC/DC Input Module M/N 57C400

**Register 0**



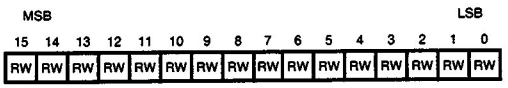
# M/N 57C400 (continued)

| Term. | Signal | Description | Reg. |
|---|---|---|---|
| 1 | 0 to 115 V AC | Signal Input | Reg 0, Bit 0 |
| 2 | 0 to 115 V AC | Signal Input | Reg 0, Bit 1 |
| 3 | 0 to 115 V AC | Signal Input | Reg 0, Bit 2 |
| 4 | 0 to 115 V AC | Signal Input | Reg 0, Bit 3 |
| 5 | 0 V AC | Isolated Common for bits 0 through 3 | |
| 6 | 0 to 115 V AC | Signal Input | Reg 0, Bit 4 |
| 7 | 0 to 115 V AC | Signal Input | Reg 0, Bit 5 |
| 8 | 0 to 115 V AC | Signal Input | Reg 0, Bit 6 |
| 9 | 0 to 115 V AC | Signal Input | Reg 0, Bit 7 |
| 10 | 0 V AC | Isolated Common for bits 4 through 7 | |
| 11 | 0 to 115 V AC | Signal Input | Reg 0, Bit 8 |
| 12 | 0 to 115 V AC | Signal Input | Reg 0, Bit 9 |
| 13 | 0 to 115 V AC | Signal Input | Reg 0, Bit 10 |
| 14 | 0 to 115 V AC | Signal Input | Reg 0, Bit 11 |
| 15 | 0 V AC | Isolated Common for bits 8 through 11 | |
| 16 | 0 to 115 V AC | Signal Input | Reg 0, Bit 12 |
| 17 | 0 to 115 V AC | Signal Input | Reg 0, Bit 13 |
| 18 | 0 to 115 V AC | Signal Input | Reg 0, Bit 14 |
| 19 | 0 to 115 V AC | Signal Input | Reg 0, Bit 15 |
| 20 | 0 V AC | Isolated Common for bits 12 through 15 | |

Output data is contained in register 0.

# 24-115 V AC/DC Output Module (M/N 57C402) and 115 V AC Output Module (M/N 57C403 and 61C503)

**Register 0**



# M/N 57C402, 57C403 and 61C503 (continued)

| Term. | Signal | Description | Register |
|---|---|---|---|
| 1 | 0 to MAX | Signal Output | Reg 0, Bit 0 |
| 2 | 0 to MAX | Signal Output | Reg 0, Bit 1 |
| 3 | 0 to MAX | Signal Output | Reg 0, Bit 2 |
| 4 | 0 to MAX | Signal Output | Reg 0, Bit 3 |
| 5 | MAX V AC | Isolated Common for bits 0 through 3 | |
| 6 | 0 to MAX | Signal Output | Reg 0, Bit 4 |
| 7 | 0 to MAX | Signal Output | Reg 0, Bit 5 |
| 8 | 0 to MAX | Signal Output | Reg 0, Bit 6 |
| 9 | 0 to MAX | Signal Output | Reg 0, Bit 7 |
| 10 | MAX V AC | Isolated Common for bits 4 through 7 | |
| 11 | 0 to MAX | Signal Output | Reg 0, Bit 8 |
| 12 | 0 to MAX | Signal Output | Reg 0, Bit 9 |
| 13 | 0 to MAX | Signal Output | Reg 0, Bit 10 |
| 14 | 0 to MAX | Signal Output | Reg 0, Bit 11 |
| 15 | MAX V AC | Isolated Common for bits 8 through 11 | |
| 16 | 0 to MAX | Signal Output | Reg 0, Bit 12 |
| 17 | 0 to MAX | Signal Output | Reg 0, Bit 13 |
| 18 | 0 to MAX | Signal Output | Reg 0, Bit 14 |
| 19 | 0 to MAX | Signal Output | Reg 0, Bit 15 |
| 20 | 0 V AC | Isolated Common for bits 12 through 15 | |

MAX = 24 V AC for 57C402 and 115 V AC for 57C403

# Network Communications Module M/N 57C404

The Network module must have the DROP DEPTH set in the application software before data can be read or written.

REGISTER 20 = DROP DEPTH (decimal) must be greater than 0 (56 minus the thumbwheel setting).

# M/N 57C404 (continued)

register 20, bit 14 = error bit, set to ON by the Network module if existing DROP DEPTH is invalid.

register 20, bit 15 = processing complete bit, set to ON when the Network module has finished processing the DROP DEPTH register.

DROP register formats:

DROP 0 register 4-7 = status for drops 1-55
registers 32-39 = broadcast registers

DROP 1-55 registers 0-31 = OUTPUTS (from SLAVE) (read/write)
registers 32-63 = INPUTS (to SLAVE) (read only)

**Programming terminal interrogation:** Command "s" for status can be entered with the coax connected and the module operating.

**Drop 0 (Area) Status and Control Registers**

| | |
|---|---|
| Registers 0-3 | System use only; do not use |
| Register 4 | Drop 0 through Drop 15 Status in bits 0 to 15 |
| Register 5 | Drop 16 through Drop 31 Status in bits 0 to 15 |
| Register 6 | Drop 32 through Drop 47 Status in bits 0 to 15 |
| Register 7 | Drop 48 through Drop 55 Status in bits 0 to 7 |
| Registers 8-11 | System use only; do not use |
| Register 12 | Drop Number |
| Register 13 | Keyswitch Mode (1 = protect, 2 = set-up, 3 = program) |
| Register 14 | Messages Received |
| Register 15 | Receive Timeouts |
| Register 16 | CRC/Parity Errors |
| Register 17 | Overrun Errors |
| Register 18 | Abort Errors |
| Register 19 | Messages Transmitted |
| Register 20 | Drop Depth |
| Registers 21-31 | System use only; do not use |
| Register 32 | Broadcast Data — transmitted every 2.6 msec |
| Register 33 | Broadcast Data — transmitted every 2.6 msec |
| Register 34 | Broadcast Data — transmitted every 2.6 msec |

## M/N 57C404 (continued)

| | |
|---|---|
| Register 35 | Broadcast Data — transmitted every 2.6 msec |
| Register 36 | Broadcast Data — transmitted every 2.6 msec |
| Register 37 | Broadcast Data — transmitted every 2.6 msec |
| Register 38 | Broadcast Data — transmitted every 2.6 msec |
| Register 39 | Broadcast Data — transmitted every 2.6 msec |
| Registers 40-63 | System use only; do not use |

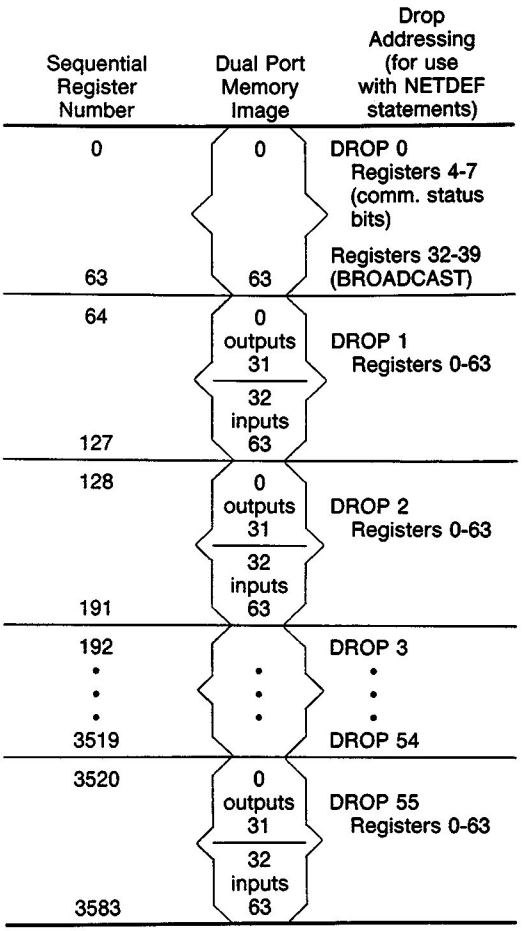
In the master module, all registers are read only, with the exception of 32-39. Register assignments on slave modules are the same except that register 20 is read/write and all other registers are read only.

# M/N 57C404 (continued)

**Memory Map**

To find the sequential register number for a particular drop, use the following:

Sequential Register Number = Drop Number * 64 + Register Offset (0-63) in Desired Drop



# Drive Controller Module (B/M 57406) Inputs and Drive Analog I/O Module (B/M 57405) Outputs

The Drive Analog I/O Module has three outputs: one non-changeable output for current minor loop feedback and two D/A outputs, which are "steerable" from the programming terminal. Drive Controller module inputs are READ only and must not be written to by the application program.

| Function | Terminal | Nominal Scaling | 57C406 Register # |
|---|---|---|---|
| Sterable D/A #1 | 1(−), 2(+) | ±4095 = ±10 V DC (× 1 Gain) | Not Available |
| Sterable D/A #2 | 3(−), 4(+) | ±4095 = ±10 V DC (× 1 Gain) | Not Available |
| Minor Loop | 5(−), 6(+) | ±4095 = ±3.3 V DC | Not Available |
| Analog Ref. | 7, 8 | ±4095 = ±10 V DC | 4096 |
| Analog Tach Feedback | 9, 10 | ±3072 = ±5 V DC | 4097 |
| Armature Voltage Feedback | *PMI (Scaling Circuit) through Drive Analog | ±8192 = ±10 V DC | 4098 |
| Current Major Loop Feedback (filtered current feedback) MAJOR_I_FDBK% | PMI (Burden Resistor) through Drive Analog | ±4095 = ±10 V DC | 4099 |
| AC Line Voltage | PMI (Line Sync Transformer) through Drive Analog | 0-8130 = 0-10 V DC 4866 = 460 or 230 V AC | 4100 |
| Current Minor Loop Feedback (unfiltered current feedback) MINOR_i_FDBK% | PMI (Burden Resistor) through Drive Analog | ±4095 = ±10 V DC | 4101 |

*Available for use when 57408-1 is installed for fast bridge change.

# B/M 57406 and 57405 (continued)

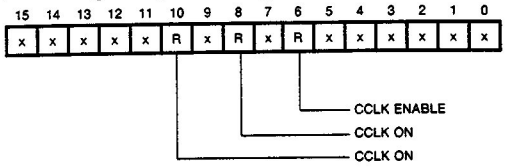
| Function | Units | Default Value | 57C406 Register # |
|---|---|---|---|
| Tach Loss Threshold | deg.α | 109 | 4136 |
| PLL Max Phase Change | µsec | 2 | 4137 |
| PLL Max Period Change | µsec | 4 | 4138 |
| Bit 0 Enables Fast Bridge Change | N/A | 0 (disabled) | 4146 |

# 2-Channel Analog Input Module M/N 57C409

**Register Assignments**

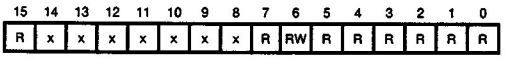
| Register Number | Description | Channel | Access |
|---|---|---|---|
| 0 | Data | 0 | R |
| 1 | Data | 1 | R |
| 2 | Current Count | 0 | R |
| 3 | Current Count | 1 | R |
| 4 | Status Register | | R |
| 5 | Interrupt Status and Control | 0 | R/W |
| 6 | Interrupt Status and Control | 1 | R/W |
| 7 | Period Count | 0 | R/W |
| 8 | Period Count | 1 | R/W |
| 9 | Low Pass Filter Selection | 0 | R/W |
| 10 | Low Pass Filter Selection | 1 | R/W |

**Status Register (READ Only) 4**

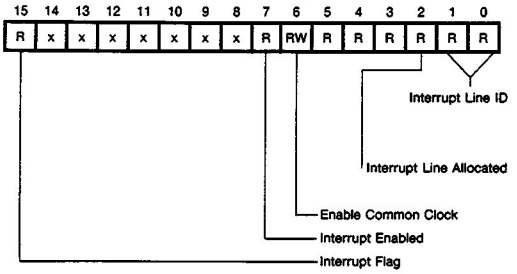


# M/N 57C409 (continued)

**Register 5 (ISCR for Channel 0)**




**Register 6 (ISCR for Channel 1)**




**Low Pass Filter Selection**

For register 9 (channel 0) and register 10 (channel 1) bits 0 and 1:

| Data | Filter Corner Frequency ±11% |
|---|---|
| 00 | 300 rad/sec |
| 01 | 145 rad/sec |
| 10 | 79 rad/sec |
| 11 | 21 rad/sec |

**LEDs**

| Color | LED | Description |
|---|---|---|
| yellow | CCLK ON | CCLK running on backplane |
| yellow | CCLK Enable | This card driving CCLK |
| yellow | CSD | N/A |
| yellow | IOCK | N/A |

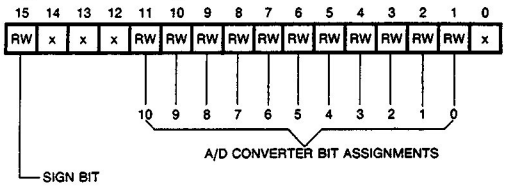
| Term. | Signal | Description | Reg. |
|---|---|---|---|
| 1 | ±10 V = ±4095 | Channel 0 Input | 0 |
| 2 | ±1 V = ±4095 | Channel 0 Input | 0 |
| 3 | | Channel 0 Common | |
| 4 | ±10 V = ±4095 | Channel 1 Input | 1 |
| 5 | ±1 V = ±4095 | Channel 1 Input | 1 |
| 6 | | Channel 1 Common | |
| 7 | | N/C | |
| 8 | +15 V | +25 ma power supply (for external use) | |
| 9 | 0 V | Common | |
| 10 | −15 V | −25 ma power supply | |

# Analog Output Module
# M/N 57C410

**Register Assignments**

| Register Number | Description | Access |
|---|---|---|
| 0 | Channel 0 | RW |
| 1 | Channel 1 | RW |
| 2 | Channel 2 | RW |
| 3 | Channel 3 | RW |

**Register Bit Arrangement**




**LEDs**

| Color | LED | Description |
|---|---|---|
| green | ISOL Power 0 | Ch. 0 Isolated Power Supply OK |
| green | ISOL Power 1 | Ch. 1 Isolated Power Supply OK |
| green | ISOL Power 2 | Ch. 2 Isolated Power Supply OK |
| green | ISOL Power 3 | Ch. 3 Isolated Power Supply OK |

# M/N 57C410 (continued)

| Term. | Signal | Voltage | | Reg. |
|---|---|---|---|---|
| 1 | | ±5 V | Ch. 0 Jumper | 0 |
| 2 | ±4095 | ±10 V | Ch. 0 Volt. Output | |
| 3 | | ±8 V | Ch. 0 Jumper | |
| 4 | ±4095 | 4-20 ma | Ch. 0 Curr. Output | |
| 5 | | 0 V | Ch. 0 Common | |
| 6 | | ±5 V | Ch. 1 Jumper | 1 |
| 7 | ±4095 | ±10 V | Ch. 1 Volt. Output | |
| 8 | | ±8 V | Ch. 1 Jumper | |
| 9 | ±4095 | 4-20 ma | Ch. 1 Curr. Output | |
| 10 | | 0 V | Ch. 1 Common | |
| 11 | | ±5 V | Ch. 2 Jumper | 2 |
| 12 | ±4095 | ±10 V | Ch. 2 Volt. Output | |
| 13 | | ±8 V | Ch. 2 Jumper | |
| 14 | ±4095 | 4-20 ma | Ch. 2 Curr. Output | |
| 15 | | 0 V | Ch. 2 Common | |
| 16 | | ±5 V | Ch. 3 Jumper | 3 |
| 17 | ±4095 | ±10 V | Ch. 3 Volt. Output | |
| 18 | | ±8 V | Ch. 3 Jumper | |
| 19 | ±4095 | 4-20 ma | Ch. 3 Curr. Output | |
| 20 | | 0 V | Ch. 3 Common | |

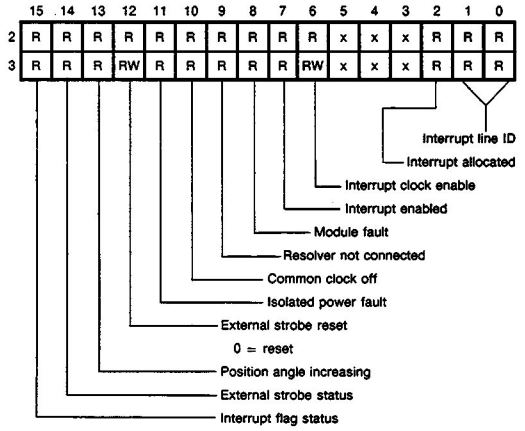
Note: Install jumpers for indicated voltage range (±10 V DC without jumpers).

# Resolver Input Module
# M/N 57C411

**Register Assignments**

| Register Number | Description | Access |
|---|---|---|
| 0 | Resolver Data | R |
| 1 | External Latch Data | R |
| 2 | Interrupt Status | R |
| 3 | Interrupt Status and Control | R/W |
| 4 | Update Period | R/W |

**Interrupt Status and Control Registers**
**Registers 2 and 3**




**LEDs**

| Color | LED | Description |
|---|---|---|
| yellow | DIRECTION | Forward or Reverse Rotation |
| yellow | FDBK OK | Resolver Electrically Connected |
| yellow | CCLK OK | CCLK running on backplane |
| yellow | IPS OK | Isolated Power Supply OK |
| green | OK | Board OK |

| Term. | Approx. Signal Level | Description |
|---|---|---|
| 1 | 26 V AC | Reference Output |
| 2 | 16 V AC | Reference Output |
| 3 | 11.8 V AC | Reference Input |
| 4 | 11.8 V AC | Reference Input |
| 5 | 11.8 V AC | Sine Input |
| 6 | 11.8 V AC | Sine Input |
| 7 | 11.8 V AC | Cosine Input |
| 8 | 11.8 V AC | Cosine Input |
| 9 | | Input for Strobe (Contact Closure Req'd) |
| 10 | | Input for Strobe (Contact Closure Req'd) |

# Field Controller Module B/M 57412

**Register Assignments**

| Register Number | Description | Range | Access |
|---|---|---|---|
| 0 | Field ON (bit 0) | 1 or 0 | R/W |
| 1 | Voltage Stability | 1 to 255 | R/W |
| 2 | Current Stability | 1 to 255 | R/W |
| 3 | Voltage Reference | 0 to 255 | R/W |
| 4 | Current Reference | ±255 | R/W |
| 5 | Current Feedback | ±127 | R |

# Modbus Interface Module M/N 57C414

**Status and Control Registers**

| | |
|---|---|
| Register 0 | Status and Control Register 1 |
| Register 1 | Status and Control Register 2 |
| Register 2 | Status and Control Register 3 |
| Register 3 | Status and Control Register 4 |
| Register 4 | Device Status (bit 0) |
| Registers 5-11 | Not used |
| Register 12 | Device Number |
| Register 13 | Keyswitch Mode (1 = protect), 2 = set-up, 3 = program) |
| Register 14 | Messages Received |
| Register 15 | Receive Timeouts |
| Register 16 | CRC/Parity Errors |
| Register 17 | Overrun Errors |
| Register 18 | Framing Errors |
| Register 19 | Messages Transmitted |
| Register 20 | Configuration/Update Request |
| Register 21 | Baud Rate (1200, 2400, 4800, 9600, 19200) |
| Register 22 | Response Timeout (seconds) |
| Register 23 | Number of Retries |
| Register 24 | Response Turn-around Delay (milliseconds) |
| Registers 25-49 | Not used |
| Registers 50-54 | Used for Debug Mode |
| Registers 55-61 | Not used |
| Register 62 | Module identification |
| Register 63 | (ASCII 'GTWY') |

# M/N 57C414 (continued)

**Memory Map**

This address map shows the relationship of the Modbus address to the decimal register number and the hex bus address number.

| Multibus Access | Modbus Register Number | Modbus Access |
|---|---|---|
| R | 00001 - 04096 single bits | R/W |
| R/W | 10001 - 14096 single bits | R |
| R/W | 30001 - 31024 16-bit registers | R |
| R/W | 40001 - 41024 16-bit registers | R/W |

| Modbus Register Number | Decimal Register Number | Hex Bus Address | Modbus Register Designations |
|---|---|---|---|
| 00001 | 64 | 2s0080 | |
| bit addressable | | | COIL REGISTERS |
| 04096 | 319 | 2s027E | |
| 10001 | 320 | 2s0280 | |
| bit addressable | | | INPUT REGISTERS |
| 14096 | 575 | 2s047E | |
| 30001 | 576 | 2s0480 | |
| word addressable | | | INPUT REGISTERS |
| 31024 | 1599 | 2s0C7E | |
| 40001 | 1600 | 2s0C80 | |
| word addressable | | | HOLDING REGISTERS |
| 41024 | 2623 | 2s147E | |

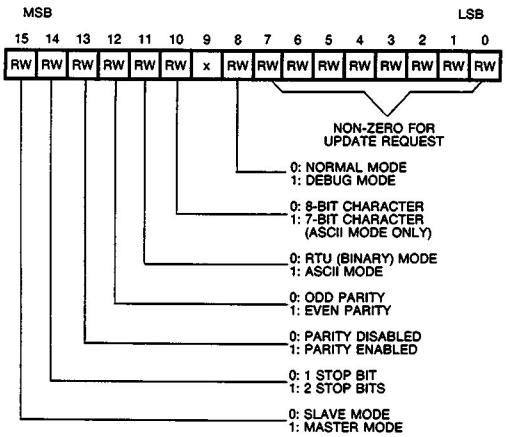
S=slot

# M/N 57C414
# (continued)

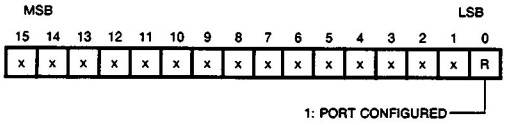
Register 21 must be set up before register 20.

REGISTER 21 = baud rate
(1200,2400,4800,9600,19200)

REGISTER 20 = configuration (see below)




**Register 4** **Link Status**



# M/N 57C414 (continued)

To convert Modbus Interface registers to hexadecimal addresses which can then be monitored using the DCS 5000 or AutoMax Executive software, perform the following calculations.

X = the register you want to monitor

1. For registers 00001 - 04096:
   X = ([{Modbus register # − 1}/16] + 64)
   For registers 10001 - 14096:
   X = ([{Modbus register # −1}/16] + 320)
   For registers 30001 - 31024:
   X = (Modbus register # − 29425)
   For registers 40001 - 41024:
   X = (Modbus register − 38401)
2. First, drop off any remainder from your result (X). Then multiply that result by 2.
3. Convert the result of #2 above to hexadecimal format.
4. To the result from #3 above add the following:
   2s0000H
   where s = slot number of the Interface module in hexadecimal.
5. The result is the DCS 5000 and AutoMax equivalent of the Modbus register.

Note: Registers 00001 to 14096 actually refer to bits within a Multibus word.

# 24V AC/DC Input Module M/N 57C415

**Register 0**

# M/N 57C415 (continued)

| Term. | Signal | Description | Register |
|---|---|---|---|
| 1 | 0 to 24 V AC | Signal Input | Reg 0, Bit 0 |
| 2 | 0 to 24 V AC | Signal Input | Reg 0, Bit 1 |
| 3 | 0 to 24 V AC | Signal Input | Reg 0, Bit 2 |
| 4 | 0 to 24 V AC | Signal Input | Reg 0, Bit 3 |
| 5 | 0 V AC | Isolated Common for bits 0 through 3 | |
| 6 | 0 to 24 V AC | Signal Input | Reg 0, Bit 4 |
| 7 | 0 to 24 V AC | Signal Input | Reg 0, Bit 5 |
| 8 | 0 to 24 V AC | Signal Input | Reg 0, Bit 6 |
| 9 | 0 to 24 V AC | Signal Input | Reg 0, Bit 7 |
| 10 | 0 V AC | Isolated Common for bits 4 through 7 | |
| 11 | 0 to 24 V AC | Signal Input | Reg 0, Bit 8 |
| 12 | 0 to 24 V AC | Signal Input | Reg 0, Bit 9 |
| 13 | 0 to 24 V AC | Signal Input | Reg 0, Bit 10 |
| 14 | 0 to 24 V AC | Signal Input | Reg 0, Bit 11 |
| 15 | 0 V AC | Isolated Common for bits 8 through 11 | |
| 16 | 0 to 24 V AC | Signal Input | Reg 0, Bit 12 |
| 17 | 0 to 24 V AC | Signal Input | Reg 0, Bit 13 |
| 18 | 0 to 24 V AC | Signal Input | Reg 0, Bit 14 |
| 19 | 0 to 24 V AC | Signal Input | Reg 0, Bit 15 |
| 20 | 0 V AC | Isolated Common for bits 12 through 15 | |

Input data is contained in register 0.

# Remote I/O Communications Module M/N 57C416

**Drop 0 (Master) Status and Control Registers**

| | |
|---|---|
| Registers 0-3 | Not used |
| Register 4 | Drop 1 through Drop 7 Status in bits 1 to 7 |
| Registers 5-11 | Not used |
| Register 12 | Drop Number |
| Register 13 | Keyswitch Mode (1 = protect, 2 = set-up, 3 = program) |
| Register 14 | Messages Received |
| Register 15 | Receive Timeouts |
| Register 16 | CRC/Parity Errors |
| Register 17 | Overrun Errors |
| Register 18 | Abort Errors |
| Register 19 | Messages Transmitted |
| Registers 20-511 | Not used |

All registers are read only.

**Programming terminal interrogation:**

Command "R" can be used to read the registers in a module in the slave drop with the coax connected and the module operating. Command "S" can be used to read the status of the module.

# AutoMate Interface Module M/N 57C417

### Status and Control Registers

| | |
|---|---|
| Register 0 | Status and Control Register 1 |
| Register 1 | Status and Control Register 2 |
| Register 2 | Status and Control Register 3 |
| Register 3 | Status and Control Register 4 |
| Register 4 | Device Status (bit 0-3) |
| Registers 5-11 | Not used |
| Register 12 | Drop Number |
| Register 13 | Keyswitch Mode (1 = protect, 2 = set-up, 3 = program) |
| Register 14 | Messages Received |
| Register 15 | Receive Timeouts |
| Register 16 | Checksum/Parity Errors |
| Register 17 | Overrun Errors |
| Register 18 | Framing Errors |
| Register 19 | Messages Transmitted |
| Register 20 | Link Configuration/Update Request |
| Register 21 | Link Baud Rate (1200, 2400, 4800, 9600, 19200) |
| Register 22 | Response Timeout (seconds) |
| Register 23 | Number of Nodes |
| Registers 24-49 | Not used |
| Registers 50-59 | Used for Debug Mode |
| Registers 60, 61 | Not used |
| Register 62 | Module Identification |
| Register 63 | (ASCII 'GTWY') |

### Memory Map

This address map shows the relationship of the AutoMate address to the decimal register number and the hex bus address number.

| Multibus Access | AutoMate Register Number | AutoMate Access |
|---|---|---|
| R | 0000.00 - 0377.17 single bits | R/W |
| R/W | 0400.00 - 0777.17 single bits | R |
| R/W | 2000 - 3777 16-bit registers | R |
| R/W | 4000 - 5777 16-bit registers | R/W |

# M/N 57C417 (continued)

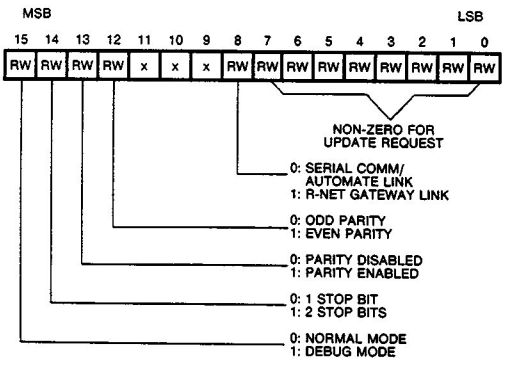
| AutoMate Register Number (Octal) | Decimal Register Number | Hex Bus Address | AutoMate Register Designations |
|---|---|---|---|
| 0000.00 | 64 | 2s0080 | DISCRETE OUTPUTS (from AUTOMATE) |
| bit addressable | | | |
| 0377.17 | 319 | 2s027E | |
| 0400.00 | 320 | 2s0280 | DISCRETE INPUTS (to AUTOMATE) |
| bit addressable | | | |
| 0777.17 | 575 | 2s047E | |
| 2000 | 576 | 2s0480 | INPUT REGISTERS (to AUTOMATE) |
| word addressable | | | |
| 3777 | 1599 | 2s0C7E | |
| 4000 | 1600 | 2s0C80 | OUTPUT REGISTERS (from AUTOMATE) |
| word addressable | | | |
| 5777 | 2623 | 2s147E | |

"s" is the slot number where the module is located.

Register 21 must be set up before register 20.

REGISTER 21 = Link baud rate (1200,2400,4800,9600,19200)

REGISTER 20 = Link configuration, update request (see below)



# M/N 57C417 (continued)

To convert AutoMate Interface registers to decimal register numbers which can then be monitored using the DCS 5000 or AutoMax Executive software, perform the following calculations.

X = the register you want to monitor

1. For octal registers 0000.00 – 0377.17 and 0400.00 – 0777.17:
   X = (AutoMate register # + 64)
   For registers 2000 – 3777 and 4000 – 5777:
   X = (AutoMate register # – 448)
2. The result is the DCS 5000 and AutoMax equivalent of the register.

# Allen-Bradley Interface Module M/N 57C418

**Status and Control Registers**

| | |
|---|---|
| Register 0 | Status and Control Register 1 |
| Register 1 | Status and Control Register 2 |
| Register 2 | Status and Control Register 3 |
| Register 3 | Status and Control Register 4 |
| Register 4 | Device Status (bit 0) |
| Registers 5-11 | Not used |
| Register 12 | Station Number |
| Register 13 | Keyswitch Mode (1 = protect, 2 = set-up, 3 = program) |
| Register 14 | Messages Received |
| Register 15 | Receive Timeouts |
| Register 16 | Checksum/Parity Errors |
| Register 17 | Overrun Errors |
| Register 18 | Framing Errors |
| Register 19 | Messages Transmitted |
| Register 20 | Link Configuration |
| Register 21 | Baud Rate (1200, 2400, 4800, 9600, 19200) |
| Register 22 | Response Timeout (seconds) |
| Register 23 | Number of Retries |
| Register 24 | ACK/NAK Timeout (25 ms. increments) |
| Registers 25-49 | Not used |
| Registers 50-59 | Used for Debug Mode |

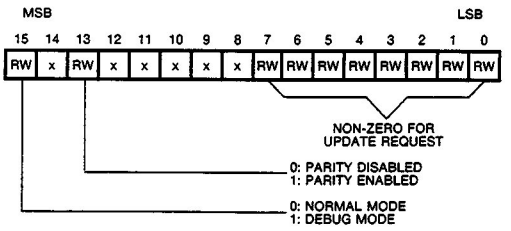
# M/N 57C418 (continued)

| | |
|---|---|
| Register 60 | Software Part Number |
| Register 61 | Revision Level (ASCII) |
| Register 62 | Module Identification |
| Register 63 | (ASCII 'GTWY') |

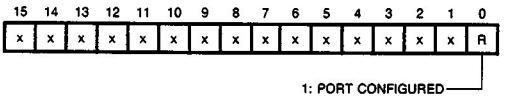
Registers 21-24 must be set up before register 20.

REGISTER 21 = Link baud rate (1200,2400,4800,9600,19200)

REGISTER 20 = configuration (see below)




**Register 4** **Link Status**




**Memory Map**

This address map shows the relationship of the A-B Interface address to the decimal register number and the hex bus address number.

| Multibus Access | A-B Interface File Register Number | A-B Interface Access |
|---|---|---|
| R | File B0, registers 0 to 255 single bits | R/W |
| R/W | File B1, registers 0 to 255 single bits | R |
| R/W | File N0, registers 0 to 1023 16-bit registers | R |
| R/W | File N1, registers 0 to 1023 16-bit registers | R/W |

# M/N 57C418 (continued)

| A-B Interface File Register Number | Decimal Register Number | Hex Bus Address | A-B Interface File Designations |
|---|---|---|---|
| 0 | 64 | 2s0080 | |
| bit addressable | | | B0 |
| 255 | 319 | 2s027E | |
| 0 | 320 | 2s0280 | |
| bit addressable | | | B1 |
| 255 | 575 | 2s047E | |
| 0 | 576 | 2s0480 | |
| word addressable | | | N0 |
| 1023 | 1599 | 2s0C7E | |
| 0 | 1600 | 2s0C80 | |
| word addressable | | | N1 |
| 1023 | 2623 | 2s147E | |

To convert Allen-Bradley Interface registers to decimal register numbers which can then be monitored using the DCS 5000 or AutoMax Executive software, perform the following calculations.

X = the register you want to monitor

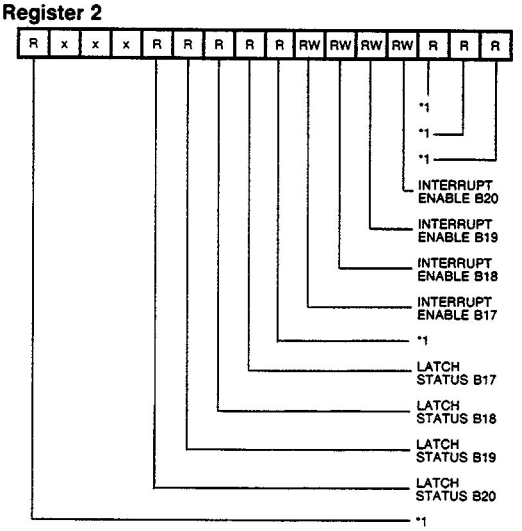
1. For file N0 registers:
   X = (file N0 register # + 576)
   For file N1 registers:
   X = (file N1 register # + 1600)
   For file B0 registers:
   X = (file B0 register # + 64)
   For file B1 registers:
   X = (file B1 register # + 320)
2. The result is the DCS 5000 and AutoMax equivalent of the register.

# 5-24 V DC Input Module
# M/N 57C419

## Registers 0 and 1

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |
| 1 | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R | R |

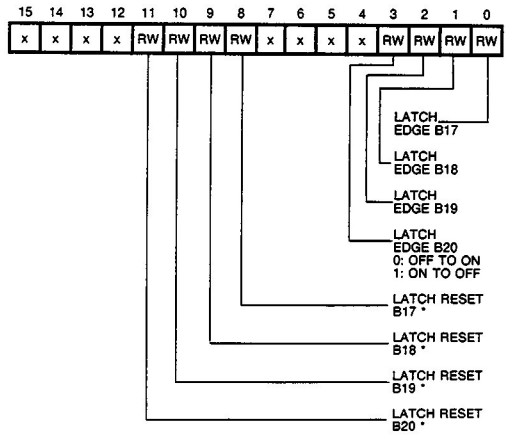
## Register 2




*1 These bits are controlled by the operating system and must not be set by the user.

B17-B20 refer to field connections.

## Register 3




* A Latch must be acknowledged by writing a 0 to the proper bit after the latch has occurred.

B17-B20 refer to field connections.

# M/N 57C419 (continued)

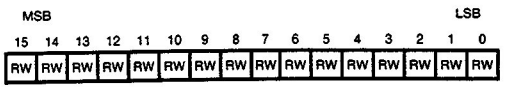
| Term. | Signal | Description |
|---|---|---|
| 1 | 5 or 24 V DC | Isolated Voltage for bits 0 to 3 |
| 2 | 0 to V supply | Signal input for bit 0 |
| 3 | 0 to V supply | Signal input for bit 1 |
| 4 | 0 to V supply | Signal input for bit 2 |
| 5 | 0 to V supply | Signal input for bit 3 |
| 6 | 5 or 24 V DC | Isolated Voltage for bits 4 to 7 |
| 7 | 0 to V supply | Signal input for bit 4 |
| 8 | 0 to V supply | Signal input for bit 5 |
| 9 | 0 to V supply | Signal input for bit 6 |
| 10 | 0 to V supply | Signal input for bit 7 |
| 11 | 5 or 24 V DC | Isolated Voltage for bits 8 to 11 |
| 12 | 0 to V supply | Signal input for bit 8 |
| 13 | 0 to V supply | Signal input for bit 9 |
| 14 | 0 to V supply | Signal input for bit 10 |
| 15 | 0 to V supply | Signal input for bit 11 |
| 16 | 5 or 24 V DC | Isolated Voltage for bits 12 to 15 |
| 17 | 0 to V supply | Signal input for bit 12* |
| 18 | 0 to V supply | Signal input for bit 13* |
| 19 | 0 to V supply | Signal input for bit 14* |
| 20 | 0 to V supply | Signal input for bit 15* |

*Reg. 1 only, these 4 inputs are also the latch inputs.

Terminal strips A (Reg. 0) and B (Reg. 1) share the same layout.

# 5-24 V DC Output Module M/N 57C420

**Registers 0 and 1**

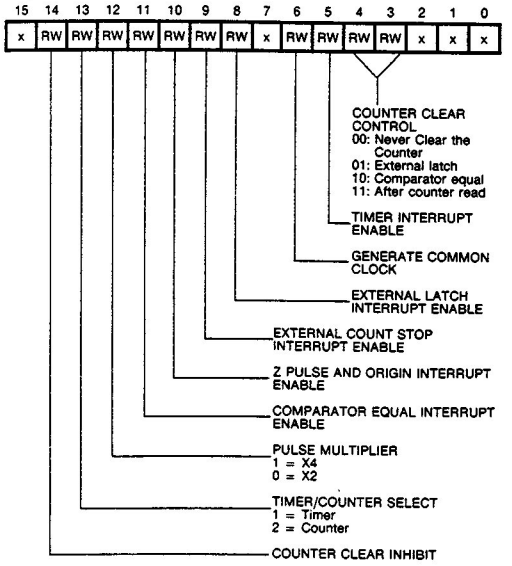


# Pulsetach Input Module
# M/N 57C421B

**Register and Bit Assignments**

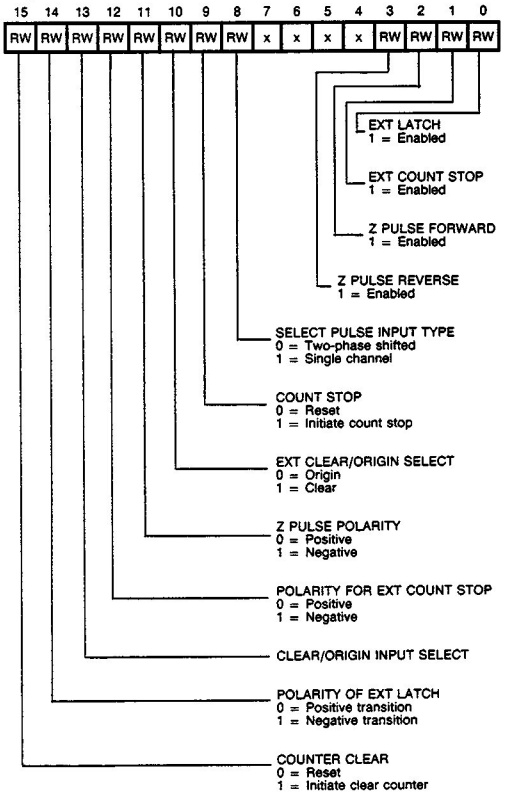
| Register Number | Bit Assignment 15 ... 8 | 7 ... 0 | Access |
|---|---|---|---|
| 0 | extended sign | MSB | R |
| 1 | 24-bit counter | LSB | R |
| 2 | MSB 16-bit timer | LSB | R/W |
| 3 | 11111111 | MSB | R/W |
| 4 | 24-bit comparator | LSB | R/W |
| 5 | Interrupt status and control register MSB | LSB | R/W |
| 6 | MSB Control register | LSB | R/W |
| 7 | MSB Control register | LSB | R/W |

**Register 5 (ISCR)**



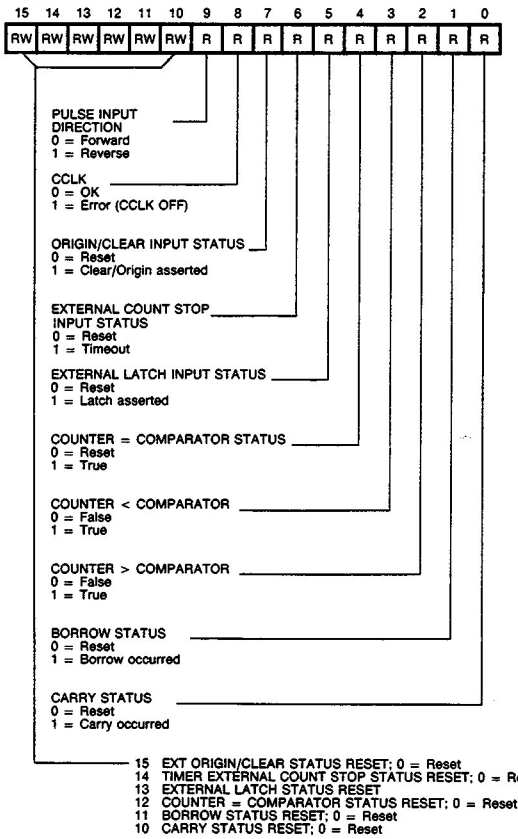
# M/N 57C421B (continued)

**Register 6**



# M/N 57C421B (continued)

**Register 7**




**LEDs**

| Color | LED | Description |
|---|---|---|
| yellow | COUNT STOP | External Count Stop Input |
| yellow | LATCH | External Latch Input |
| yellow | CLEAR | External Clear Input |
| yellow | FORWARD | Forward rotation sensed |
| yellow | REVERSE | Reverse rotation sensed |
| green | CCLK OK | CCLK running on backplane |

# M/N 57C421B (continued)

| Term. | Signal | Description |
|---|---|---|
| 1 | | N/C |
| 2 | 0 to +12 V DC | Channel +A 12 V Input |
| 3 | 0 to +12 V DC | Channel +B 12 V Input |
| 4 | 0 to +12 V DC | Channel +Z 12 V Input |
| 5 | | N/C |
| 6 | 0 to +5 V DC | Origin Clear 5 V Input |
| 7 | 0 V | Origin Clear Common |
| 8 | 0 to +5 V DC | External Latch 5 V Input |
| 9 | 0 V | External Latch Common |
| 10 | 0 to +5 V DC | External Count Stop 5 V Input |
| 11 | 0 V | External Count Stop Common |
| 12 | 0 to +5 V DC | Channel +A 5 V Input |
| 13 | | N/C |
| 14 | 0 to +5 or +12 | Channel A Common or -A |
| 15 | 0 to +5 V DC | Channel +B 5 V Input |
| 16 | | N/C |
| 17 | 0 to +5 or +12 | Channel B Common or -B |
| 18 | 0 to +5 V DC | Channel +Z 5 V Input |
| 19 | | N/C |
| 20 | 0 to +5 or +12 | Channel Z Common or -Z |

N/C=no connection.

# 2-Axis Servo Module
# M/N 57C422B

**Register Assignments**

| | |
|---|---|
| Register 0 | Encoder set-up |
| Register 1 | Drive set-up |
| Register 2 | Switch set-up |
| Register 3 | Proportional gain |
| Register 4 | Integral gain |
| Register 5 | Velocity loop gain |
| Register 6 | Feedforward gain |
| Register 7 | Deadband compensation |
| Register 8 | Maximum position error |
| Register 9 | Maximum velocity error |
| Register 10 | In-position tolerance |
| Registers 11, 12 | Positive overtravel limit |
| Registers 13, 14 | Negative overtravel limit |
| Registers 15, 16 | Low speed homing reference |
| Registers 17, 18 | Command position |
| Registers 19, 20 | Command velocity |
| Registers 21, 22 | Command acceleration |
| Registers 23, 24 | Command deceleration |
| Registers 25, 26 | Gear ratio |
| Register 27 | User LEDs register |
| Register 28 | Direct drive reference command |
| Registers 29, 30 | Sync position |
| Register 31 | Maximum voltage reference |
| Register 32 | Positive linearization constant |
| Register 33 | Negative linearization constant |
| Registers 34, 35 | Feedback unwind constant |
| Registers 36, 37 | Gearing unwind constant |
| Register 38 | Gearing modes |
| Registers 39, 40 | Backlash Compensation |
| Register 64 | Interrupt Reset (for both X and Y) |
| Register 65 | Mode |
| Register 66 | Command |
| Register 67 | Command |
| Registers 68, 69 | Interrupt enable |
| Register 72 | Status |
| Register 73 | Fault |
| Registers 74, 75 | Current feedback position |
| Registers 76, 77 | Current gearing position |
| Registers 78, 79 | Current velocity |
| Register 80 | Following error |
| Register 81 | Velocity error |
| Register 82 | Digital input status |
| Register 83 | Current velocity status update period |
| Registers 84, 85 | Feedback registration position |
| Registers 86, 87 | Gearing registration position |
| Registers 99-103 | Software version number |
| Registers 104, 105 | Interrupt status |
| Registers 124, 125 | Master axis position increment |
| Registers 126, 127 | Master Reference position |

# M/N 57C422B (continued)

| | |
|---|---|
| Register 128 | Number of points in CAM table |
| Register 129 | Time between points in CAM table |
| Register 130 | CAM Mode |
| Register 131 | Loop back point in Time CAM table |
| Registers 132-2046 | CAM data table |
| Register 4095 | Interrupt status and control (for both X and Y) |

Register assignments for X axis. For Y axis, add 2048 to the X axis register numbers (except registers 64 and 4095).

**LEDs**

| Color | LED | Description |
|---|---|---|
| yellow | XHOME | ↑ Home Limit Input |
| yellow | X+OTRAV | X Positive Overtravel Input |
| yellow | X−OTRAV | I Negative Overtravel Input |
| yellow | XFAULT | A Fault Input |
| yellow | XF.REG | x Feedback Registration Input |
| yellow | XG.REG | i Gearing Registration Input |
| yellow | XENABLE | s Enable Input |
| yellow | XSP | ↓ Spare |
| yellow | STAT3 | CPU Status LEDs |
| yellow | STAT2 | CPU Status LEDs |
| yellow | STAT1 | CPU Status LEDs |
| yellow | DIAG | CPU Status LEDs |
| green | OK | Board OK |
| yellow | YHOME | ↑ Home Limit Input |
| yellow | Y+OTRAV | Y Positive Overtravel Input |
| yellow | Y−OTRAV | I Negative Overtravel Input |
| yellow | YFAULT | A Fault Input |
| yellow | YF.REG | x Feedback Registration Input |
| yellow | YG.REG | i Gearing Registration Input |
| yellow | YENABLE | s Enable Input |
| yellow | YSP | ↓ Spare |

# M/N 57C422B (continued)

| Term. | Signal | Description |
|---|---|---|
| 1 | 0 V (Common) | from Encoder Power Supply |
| 2 | +5 V | from Encoder Power Supply |
| 3 | 0 V | to Feedback Encoder |
| 4 | +5 | to Feedback Encoder |
| 5 | 0 to +5 | +A from Feedback Encoder |
| 6 | 0 to +5 | −A from Feedback Encoder |
| 7 | 0 to +5 | +B from Feedback Encoder |
| 8 | 0 to +5 | −B from Feedback Encoder |
| 9 | 0 to +5 | +Z from Feedback Encoder |
| 10 | 0 to +5 | −Z from Feedback Encoder |
| 11 | 0 V | to Gearing Encoder |
| 12 | +5 | to Gearing Encoder |
| 13 | 0 to +5 | +A from Gearing Encoder |
| 14 | 0 to +5 | −A from Gearing Encoder |
| 15 | 0 to +5 | +B from Gearing Encoder |
| 16 | 0 to +5 | −B from Gearing Encoder |
| 17 | 0 to +5 | +Z from Gearing Encoder |
| 18 | 0 to +5 | −Z from Gearing Encoder |
| 19 | | N/C |
| 20 | | N/C |
| 21 | +12/15 V DC | from Drive Reference Power Supply |
| 22 | −12/15 V DC | from Drive Reference Power Supply |
| 23 | 0 V (Common) | to Drive Reference & From Power Supply |
| 24 | ±12/15 V DC | to Drive Reference |
| 25 | | Watchdog Contact Output |
| 26 | | Watchdog Contact Output |
| 27 | 0 to 24 V DC/AC | Home Limit Switch Input |
| 28 | 0 V (Common) | Home Limit Switch Common |
| 29 | 0 to 24 V DC/AC | Positive Overtravel Limit Switch Input |
| 30 | 0 V (Common) | Positive Overtravel Limit Switch Common |
| 31 | 0 to 24 V DC/AC | Negative Overtravel Limit Switch Input |
| 32 | 0 V (Common) | Negative Overtravel Limit Switch Common |
| 33 | 0 to 24 V DC/AC | Drive Fault Input |
| 34 | 0 V (Common) | Drive Fault Input |
| 35 | 0 to 24 V DC | Feedback Registration Input |
| 36 | 0 V (Common) | Feedback Registration Common |
| 37 | 0 to 24 V DC | Gearing Registration Input |
| 38 | 0 V (Common) | Gearing Registration Common |
| 39 | | N/C |
| 40 | | N/C |

Note: Both the X and Y axis have the same terminal layout. The uppermost connector on the module is for the X axis.

# MaxPak III
# High Speed Link
# Module M/N 57C424

**Dual Port Register Assignments**

| | |
|---|---|
| Register 0 | Status and control |
| Register 4 | Link active |
| Register 12 | Device number |
| Register 13 | Keyswitch state |
| Register 14 | Messages received |
| Register 15 | Receive timeouts |
| Register 16 | Checksum errors |
| Register 17 | Overrun errors |
| Register 18 | Framing errors |
| Register 19 | Messages transmitted |
| Register 20 | Parity errors |
| Register 26 | Comm. error flags |
| Register 27 | Configuration error flags |
| Register 28 | MaxPak III status byte |
| Register 29 | Transmit active |
| Register 30 | Command/status change |
| Register 31 | Comm. error reset |
| Register 40 | Total input on-line registers |
| Register 41 | Total input off-line registers |
| Register 42 | Total input on-line bits |
| Register 43 | Total input off-line bits |
| Register 44 | Total output registers |
| Register 45 | Total output bits |
| Register 60 | Fatal error # |
| Register 62 | Module ASCII ID "HS" |
| Register 63 | Module ASCII ID"L" |
| Register 64 | Module version number |
| Register 70 | # of registers to send to MaxPak III |
| Register 71 | # of bits to send to MaxPak III |
| Register 75 | # of registers to receive from MaxPak III |
| Register 76 | # of bits to receive from MaxPak III |
| Register 80 | Max. receive timeout (in msec.) |
| Register 81 | Speed loop time period (in ticks) |
| Register 100 | Data receive register 0 |
| Register 101 | Data receive register 1 |
| Register 102 | Data receive register 0 |
| Register : | |
| Register 354 | Data receive register 254 |
| Register 355 | Data receive register 255 |
| Register 400 | Data receive packed bits reg. 0 |
| Register 401 | Data receive packedbits reg. 1 |
| Register : | |

| | |
|---|---|
| Register 415 | Data receive packed bits reg. 15 |
| Register 1100 | Data transmit register 0 |
| Register 1101 | Data transmit register 1 |
| Register 1102 | Data transmit register 2 |
| Register : | |
| Register 1354 | Data transmit register 254 |
| Register 1355 | Data transmit register 255 |
| Register 1400 | Data transmit packed bits reg. 0 |
| Register 1401 | Data transmit register 1 |
| Register : | |
| Register 1415 | Data transmit packed bits reg. 15 |

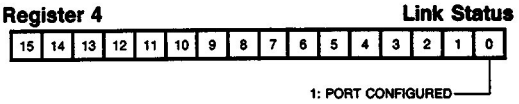
# Toledo Scale Interface Module M/N 57C428

Register 21 must be set up before register 20.

REGISTER 21 = baud rate
(1200,2400,4800,9600,19200)

REGISTER 20 = Update request
(non-zero for update request)

**Register 4** **Link Status**




**Register Assignments**

| | |
|---|---|
| Register 4 | Link status |
| Register 14 | Number of good messages received |
| Register 15 | X |
| Register 16 | Number of checksum or parity errors |
| Register 17 | Number of overrun errors |
| Register 18 | Number of framing errors |
| Register 20 | Configuration/update request |
| Register 21 | Baud rate |
| Register 22 | Link timeout |
| Register 64 | Request status |
| Register 65 | Message counter |
| Register 66 | Status byte "A" |
| Register 67 | Status byte "B" |
| Register 68 | Status byte "C" |
| Registers 69, 70 | Indicated weight |
| Registers 71, 72 | Tare weight |

**LEDs**

| Color | LED | Description |
|---|---|---|
| green | OK | Module is operational |

# AutoMax R-Net Processor Module M/N 57C429

**Register Organization**

| Register | Description | R/W |
|---|---|---|
| Register 0-63 | Status and Control | R/W |
| Register 64-2623 | AutoMate Image | R/W |
| Register 2624-3583 | Not used | X |
| Register 3584-4095 | Command Buffer | R |

**Status and Control Registers**

| | |
|---|---|
| Register 0 | System use only |
| Register 1 | System use only |
| Register 2 | System use only |
| Register 3 | System use only |
| Register 4 | Drops on line status (15-0) |
| Register 5 | Drops on line status (31-16) |
| Register 6 | Drops on line status (47-32) |
| Register 7 | Drops on line status (63-48) |
| Register 8 | System use only |
| Register 9 | System use only |
| Register 10 | System use only |
| Register 11 | System use only |
| Register 12 | Node number |
| Register 13 | Slot number |
| Register 14 | Messages received count |
| Register 15 | Receive timeouts count |
| Register 16 | CRC error count |
| Register 17 | Overrun error count |
| Register 18 | Illegal message count |
| Register 19 | Messages transmitted count |
| Register 20 | Lost token count |
| Register 21 | Max. node number |
| Register 22 | Response timeout (in seconds) |
| Register 23 | Current token time |
| Register 24 | Max. token time |
| Register 25 | Reserved for system use |
| Register : | |
| Register 49 | Reserved for system use |
| Register 50 | Transmit global data enabling reg. for AutoMate image reg. 71 |
| Register 51 | Transmit global data enabling reg. for AutoMate image reg. 72 |
| Register 52 | Transmit global data enabling reg. for AutoMate image reg. 73 |
| Register 53 | Transmit global data enabling reg. for AutoMate image reg. 74 |
| Register 54 | Reserved for system use |
| Register : | |
| Register 59 | Reserved for system use |
| Register 60 | LED error code |
| Register 61 | Reserved for system use |
| Register 62 | Reserved for system use |
| Register 63 | Reserved for system use |

**AutoMate Image Registers**

| | |
|---|---|
| Registers 64-575 | AutoMate Registers 0000.00-0777.17 (octal) |
| Registers 576-2623 | AutoMate Registers 2000-5777 (octal) |

# AutoMax Processor Module M/N 57C430, M/N 57C431 and M/N 57C435

### Connection Information

All Processor module ports except for the port labeled "PROGRAMMER/PORT B" on the leftmost Processor in the rack are available to the user to connect to an external device which will be controlled by application tasks running on the Processor. Refer to the Enhanced BASIC Language Instruction Manual (J-3675) for more information. Note that with AutoMax Processor modules, you can use the statements OPEN "PORTA" or OPEN "PORTB".

Note: If you do not enable bit 15 (hardware handshaking) in the SETUP parameter of the OPEN statement, only pins 2, 3 and 7 of the port you OPEN will be meaningful.

| Pin # | I/O | Function |
|---|---|---|
| 2 | O | This signal contains transmitted data. |
| 3 | I | This signal contains received data. |
| 4 | O | Transmit status. This signal is true whenever the transmitter is sending characters. It is used to "bracket" a character transmission. It can be used to enable/disable any type of external equipment, such as a tri-state transmit modem, which requires an enable signal to output characters. This signal is meaningful only if hardware handshaking has been enabled. |
| 5 | I | This signal enables the transmitter. It must be true for the tansmitter to send a character. This signal is typically used for hardware flow control. It is meaningful only if hardware handshaking has been enabled. |
| 6 | I | This signal enables the receiver. It must be true in order for the receiver to accept characters. If the signal becomes false while a message is being received, any characters being received will be deleted and an error will be reported to the application software. This signal is meaningful only if hardware handshaking has been enabled. |
| 7 | | Signal common. |

# M/N 57C430, 57C431, and 57C435 (continued)

| Pin # | I/O | Function |
|---|---|---|
| 10 | O | This signal is an isolated +12 Volt which can be used as an enable or equipment ready indicator. The signal is always on whenever power is applied to the Processor. |
| 20 | O | This signal indicates receiver status. The signal is true whenever the receiver can accept characters, i.e., when the receive buffer is not full. When the receive buffer fills to within a specified limit, the signal is turned off. The signal can be used to disable another transmitter. It is meaningful only when hardware handshaking has been enabled. |

**LEDs**

| Color | LED | Description |
|---|---|---|
| green | BAT.OK | On-board battery status |
| green | OK | Module is opertional |

# Power Supply Module M/N 57C491 and M/N 57C493

**57C491**

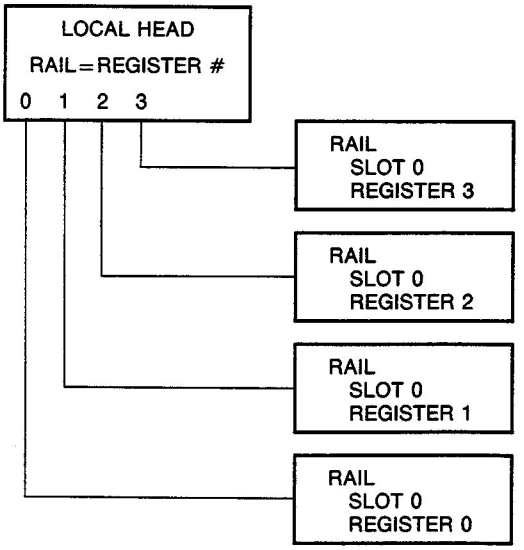
| Color | LED | Description |
|---|---|---|
| green | POWER | Power applied indicator |
| yellow | P/S READY | All voltages present |
| yellow | SYSTEM READY | All AutoMax Processors OK |
| red | BLOWN FUSE | Line fuse indicator (ON = open) |

**57C493**

| Color | LED | Description |
|---|---|---|
| green | POWER | Power applied indicator |
| red | FAULT | AutoMax Processor watchdog is not present or output voltage is out of range |

# Local I/O Head
# M/N 61C22 and 61C23

The ports on the 61C22 can connect to four Digital Rails. Note that input cards and output cards cannot be mixed on the same Rail. Electronic input cards cannot be used with the 61C22.

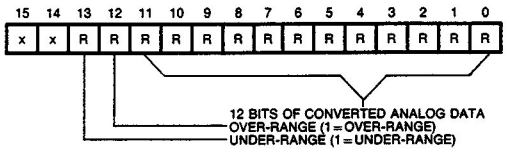



**LEDs**

| Color | LED | Description |
|---|---|---|
| green | POWER | Power ON indicator |
| green | COMM | Comm. w/host OK |
| red | RAIL FAULT 0 | Rail 0 fault |
| red | RAIL FAULT 1 | Rail 1 fault |
| red | RAIL FAULT 2 | Rail 2 fault |
| red | RAIL FAULT 3 | Rail 3 fault |

# 4-Input
# 4-20 mA Analog Rail Module
# M/N 61C345

**Registers 0, 1, 2 and 3 (READ only)**




**LEDs**

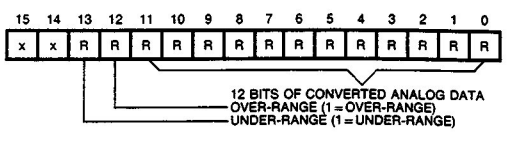
| Color | LED | Description |
|---|---|---|
| green | POWER OK | All req'd power present |
| green | COMM OK | Communication w/host OK |

Note that the MODE switch must always select local head mode for operation with the DCS/AutoMax Remote Head.

| Term. | Signal | Description | Reg. |
|---|---|---|---|
| 1 | 0 to 20 mA | Input Channel 0 | 0 |
| 2 | 0 | Common Channel 0 | |
| 3 | | N/C | |
| 4 | 0 to 20 mA | Input Channel 1 | 1 |
| 5 | 0 | Common Channel 1 | |
| 6 | | N/C | |
| 7 | 0 to 20 mA | Input Channel 2 | 2 |
| 8 | 0 | Common Channel 2 | |
| 9 | | N/C | |
| 10 | 0 to 20 mA | Input Channel 3 | 3 |
| 11 | 0 | Common Channel 3 | |
| 12 | | N/C | |

# 4-Input
# 0-10 V Analog Rail Module
# M/N 61C346

**Registers 0, 1, 2 and 3 (READ only)**



# M/N 61C346 (continued)

**LEDs**

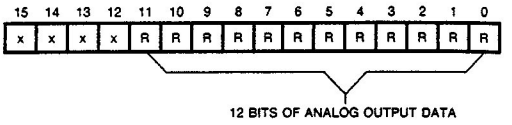
| Color | LED | Description |
|---|---|---|
| green | POWER OK | All req'd power present |
| green | COMM OK | Communication w/host OK |

Note that the MODE switch must always select local head mode for operation with the DCS/AutoMax Remote Head.

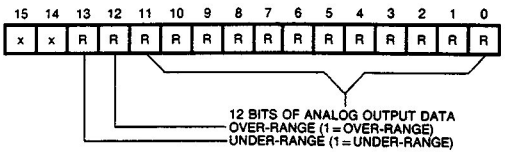
| Term. | Signal | Description | Reg. |
|---|---|---|---|
| 1 | 0 to 10 V | Input Channel 0 | 0 |
| 2 | 0 | Common Channel 0 | |
| 3 | | N/C | |
| 4 | 0 to 10 V | Input Channel 1 | 1 |
| 5 | 0 | Common Channel 1 | |
| 6 | | N/C | |
| 7 | 0 to 10 V | Input Channel 2 | 2 |
| 8 | 0 | Common Channel 2 | |
| 9 | | N/C | |
| 10 | 0 to 10 V | Input Channel 3 | 3 |
| 11 | 0 | Common Channel 3 | |
| 12 | | N/C | |

# 2-In/2-Out 0-10 V Analog Rail Module M/N 61C350

**Registers 0 and 1**




**Registers 2 and 3 (READ only)**



# M/N 61C350 (continued)

**LEDs**

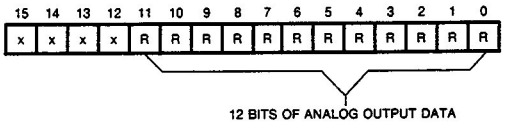
| Color | LED | Description |
|---|---|---|
| green | POWER OK | All req'd power present |
| green | COMM OK | Communication w/host OK |

Note that the MODE switch must always select local head mode for operation with the DCS/AutoMax Remote Head.

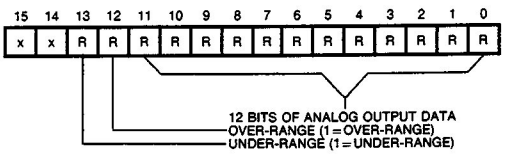
| Term. | Signal | Description | Reg. |
|---|---|---|---|
| 1 | 0 to 10 V | Output Channel 0 | 0 |
| 2 | 0 | Common Channel 0 | |
| 3 | | N/C | |
| 4 | 0 to 10 V | Output Channel 1 | 1 |
| 5 | 0 | Common Channel 1 | |
| 6 | | N/C | |
| 7 | 0 to 10 V | Output Channel 2 | 2 |
| 8 | 0 | Common Channel 2 | |
| 9 | | N/C | |
| 10 | 0 to 10 V | Output Channel 3 | 3 |
| 11 | 0 | Common Channel 3 | |
| 12 | | N/C | |

# 2-In/2-Out 4-20 mA Analog Rail Module M/N 61C351

**Registers 0 and 1**




**Registers 2 and 3 (READ only)**



# M/N 61C351 (continued)

**LEDs**

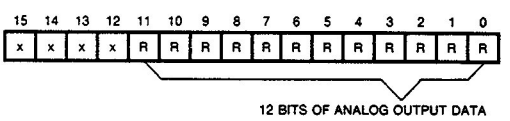
| Color | LED | Description |
|---|---|---|
| green | POWER OK | All req'd power present |
| green | COMM OK | Communication w/host OK |

Note that the MODE switch must always select local head mode for operation with the DCS/AutoMax Remote Head.

| Term. | Signal | Description | Reg. |
|---|---|---|---|
| 1 | 4 to 20 mA | Output Channel 0 | 0 |
| 2 | 0 | Common Channel 0 | |
| 3 | | N/C | |
| 4 | 4 to 20 mA | Output Channel 1 | 1 |
| 5 | 0 | Common Channel 1 | |
| 6 | | N/C | |
| 7 | 4 to 20 mA | Output Channel 2 | 2 |
| 8 | 0 | Common Channel 2 | |
| 9 | | N/C | |
| 10 | 4 to 20 mA | Output Channel 3 | 3 |
| 11 | 0 | Common Channel 3 | |
| 12 | | N/C | |

# 4 Output 4-20 mA Analog Rail Module M/N 61C365

**Registers 0, 1, 2 and 3 (READ only)**




**LEDs**

| Color | LED | Description |
|---|---|---|
| green | POWER OK | All req'd power present |
| green | COMM OK | Communication w/host OK |

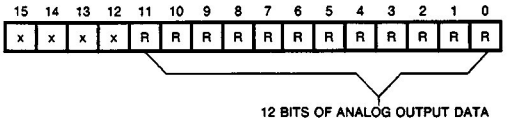
Note that the MODE switch must always select local head mode for operation with the DCS/AutoMax Remote Head.

# M/N 61C365 (continued)

| Term. | Signal | Description | Reg. |
|---|---|---|---|
| 1 | 4 to 20 mA | Output Channel 0 | 0 |
| 2 | 0 | Common Channel 0 | |
| 3 | | N/C | |
| 4 | 4 to 20 mA | Output Channel 1 | 1 |
| 5 | 0 | Common Channel 1 | |
| 6 | | N/C | |
| 7 | 4 to 20 mA | Output Channel 2 | 2 |
| 8 | 0 | Common Channel 2 | |
| 9 | | N/C | |
| 10 | 4 to 20 mA | Output Channel 3 | 3 |
| 11 | 0 | Common Channel 3 | |
| 12 | | N/C | |

# 4 Output 0-10 V Analog Rail Module M/N 61C366

**Registers 0, 1, 2 and 3 (READ only)**




**LEDs**

| Color | LED | Description |
|---|---|---|
| green | POWER OK | All req'd power present |
| green | COMM OK | Communication w/host OK |

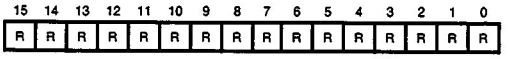
Note that the MODE switch must always select local head mode for operation with the DCS/AutoMax Remote Head.

# M/N 61C366 (continued)

| Term. | Signal | Description | Reg. |
|---|---|---|---|
| 1 | 0 to 10 V | Output Channel 0 | 0 |
| 2 | 0 | Common Channel 0 | |
| 3 | | N/C | |
| 4 | 0 to 10 V | Output Channel 1 | 1 |
| 5 | 0 | Common Channel 1 | |
| 6 | | N/C | |
| 7 | 0 to 10 V | Output Channel 2 | 2 |
| 8 | 0 | Common Channel 2 | |
| 9 | | N/C | |
| 10 | 0 to 10 V | Output Channel 3 | 3 |
| 11 | 0 | Common Channel 3 | |
| 12 | | N/C | |

# 115 V AC Input Module M/N 61C500

**Register 0 (READ only)**

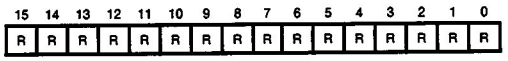



| Term. | Signal | Description | Reg. |
|---|---|---|---|
| 1 | 0 to 115 V AC | Signal Input | Reg 0, Bit 0 |
| 2 | 0 to 115 V AC | Signal Input | Reg 0, Bit 1 |
| 3 | 0 to 115 V AC | Signal Input | Reg 0, Bit 2 |
| 4 | 0 to 115 V AC | Signal Input | Reg 0, Bit 3 |
| 5 | 0 V AC | Isolated Common for bits 0 through 3 | |
| 6 | 0 to 115 V AC | Signal Input | Reg 0, Bit 4 |
| 7 | 0 to 115 V AC | Signal Input | Reg 0, Bit 5 |
| 8 | 0 to 115 V AC | Signal Input | Reg 0, Bit 6 |
| 9 | 0 to 115 V AC | Signal Input | Reg 0, Bit 7 |
| 10 | 0 V AC | Isolated Common for bits 4 through 7 | |
| 11 | 0 to 115 V AC | Signal Input | Reg 0, Bit 8 |
| 12 | 0 to 115 V AC | Signal Input | Reg 0, Bit 9 |
| 13 | 0 to 115 V AC | Signal Input | Reg 0, Bit 10 |
| 14 | 0 to 115 V AC | Signal Input | Reg 0, Bit 11 |
| 15 | 0 V AC | Isolated Common for bits 8 through 11 | |
| 16 | 0 to 115 V AC | Signal Input | Reg 0, Bit 12 |
| 17 | 0 to 115 V AC | Signal Input | Reg 0, Bit 13 |
| 18 | 0 to 115 V AC | Signal Input | Reg 0, Bit 14 |
| 19 | 0 to 115 V AC | Signal Input | Reg 0, Bit 15 |
| 20 | 0 V AC | Isolated Common for bits 12 through 15 | |

Input data is contained in Register 0.

# 24 V AC/DC Input Module
# M/N 61C515

**Register 0 (READ only)**




| Term. | Signal | Description | Reg. |
|---|---|---|---|
| 1 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 0 |
| 2 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 1 |
| 3 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 2 |
| 4 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 3 |
| 5 | 0 V AC | Isolated Common for bits 0 through 3 | |
| 6 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 4 |
| 7 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 5 |
| 8 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 6 |
| 9 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 7 |
| 10 | 0 V AC | Isolated Common for bits 4 through 7 | |
| 11 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 8 |
| 12 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 9 |
| 13 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 10 |
| 14 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 11 |
| 15 | 0 V AC | Isolated Common for bits 8 through 11 | |
| 16 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 12 |
| 17 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 13 |
| 18 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 14 |
| 19 | 0 to 24 V AC/DC | Signal Input | Reg 0, Bit 15 |
| 20 | 0 V AC | Isolated Common for bits 12 through 15 | |

Input data is contained in Register 0.

# Current Input Module
# M/N 61C540

| Register Number | Description | Access |
|---|---|---|
| 0 | Channel 0 A/D Data | R |
| 1 | Channel 1 A/D Data | R |
| 2 | Channel 2 A/D Data | R |
| 3 | Channel 3 A/D Data | R |
| 4 | Channel 4 A/D Data | R |
| 5 | Channel 5 A/D Data | R |
| 6 | Channel 6 A/D Data | R |
| 7 | Channel 7 A/D Data | R |
| 8 | Channel 8 A/D Data | R |
| 9 | Channel 9 A/D Data | R |
| 10 | Channel 10 A/D Data | R |
| 11 | Channel 11 A/D Data | R |
| 12 | Channel 12 A/D Data | R |
| 13 | Channel 13 A/D Data | R |
| 14 | Channel 14 A/D Data | R |
| 15 | Channel 15 A/D Data | R |
| 16 | High High Alarm Status | R |
| 17 | High Alarm Status | R |
| 18 | Low Alarm Status | R |
| 19 | Low Low Alarm Status | R |
| 20 | Out of Range Status | R |
| 21 | Channel Configuration Status | R |
| 22 | Configuration Status | R |
| 23 | Channel Number | R/W |
| 24 | Maximum Scaling Value | R/W |
| 25 | Minimum Scaling Value | R/W |
| 26 | Number of Samples | R/W |
| 27 | High High Alarm | R/W |
| 28 | High Alarm | R/W |
| 29 | Low Alarm | R/W |
| 30 | Low Low Alarm | R/W |
| 31 | Configuration Command | R/W |

# Current Input Module M/N 61C540 (continued)

| Terminal Block Lable | Current Input Function |
|---|---|
| V1 | Primary Power Source for Current Loops |
| G1 | Primary Power Source Return |
| 0A | Current Loop 0: Power |
| 0B | Current Loop 0: 4-20 mA Input |
| S | Current Loop 0: Cable Shield |
| S | Current Loop 1: Cable Shield |
| 1A | Current Loop 1: Power |
| 1B | Current Loop 1: 4-20 mA Input |
| 2A | Current Loop 2: Power |
| 2B | Current Loop 2: 4-20 mA Input |
| S | Current Loop 2: Cable Shield |
| S | Current Loop 3: Cable Shield |
| 3A | Current Loop 3: Power |
| 3B | Current Loop 3: 4-20 mA Input |
| 4A | Current Loop 4: Power |
| 4B | Current Loop 4: 4-20 mA Input |
| S | Current Loop 4: Cable Shield |
| S | Current Loop 5: Cable Shield |
| 5A | Current Loop 5: Power |
| 5B | Current Loop 5: 4-20 mA Input |
| 6A | Current Loop 6: Power |
| 6B | Current Loop 6: 4-20 mA Input |
| S | Current Loop 6: Cable Shield |
| S | Current Loop 7: Cable Shield |
| 7A | Current Loop 7: Power |
| 7B | Current Loop 7: 4-20 mA Input |
| 8A | Current Loop 8: Power |
| 8B | Current Loop 8: 4-20 mA Input |
| S | Current Loop 8: Cable Shield |
| S | Current Loop 9: Cable Shield |
| 9A | Current Loop 9: Power |
| 9B | Current Loop 9: 4-20 mA Input |
| 10A | Current Loop 10: Power |
| 10B | Current Loop 10: 4-20 mA Input |
| S | Current Loop 10: Cable Shield |
| S | Current Loop 11: Cable Shield |
| 11A | Current Loop 11: Power |
| 11B | Current Loop 11: 4-20 mA Input |
| 12A | Current Loop 12: Power |
| 12B | Current Loop 12: 4-20 mA Input |
| S | Current Loop 12: Cable Shield |
| S | Current Loop 13: Cable Shield |
| 13A | Current Loop 13: Power |
| 13B | Current Loop 13: 4-20 mA Input |
| 14A | Current Loop 14: Power |
| 14B | Current Loop 14: 4-20 mA Input |
| S | Current Loop 14: Cable Shield |
| S | Current Loop 15: Cable Shield |
| 15A | Current Loop 15: Power |
| 15B | Current Loop 15: 4-20 mA Input |
| V2 | Back-up Power Source for Current Loops |
| G2 | Back-up Power Source Return |

# Voltage Input Module
# M/N 61C542

| Register Number | Description | Access |
|---|---|---|
| 0 | Channel 0 A/D Data | R |
| 1 | Channel 1 A/D Data | R |
| 2 | Channel 2 A/D Data | R |
| 3 | Channel 3 A/D Data | R |
| 4 | Channel 4 A/D Data | R |
| 5 | Channel 5 A/D Data | R |
| 6 | Channel 6 A/D Data | R |
| 7 | Channel 7 A/D Data | R |
| 8 | Channel 8 A/D Data | R |
| 9 | Channel 9 A/D Data | R |
| 10 | Channel 10 A/D Data | R |
| 11 | Channel 11 A/D Data | R |
| 12 | Channel 12 A/D Data | R |
| 13 | Channel 13 A/D Data | R |
| 14 | Channel 14 A/D Data | R |
| 15 | Channel 15 A/D Data | R |
| 16 | High High Alarm Status | R |
| 17 | High Alarm Status | R |
| 18 | Low Alarm Status | R |
| 19 | Low Low Alarm Status | R |
| 20 | Out of Range Status | R |
| 21 | Channel Configuration Status | R |
| 22 | Configuration Status | R |
| 23 | Channel Number | R/W |
| 24 | Maximum Scaling Value | R/W |
| 25 | Minimum Scaling Value | R/W |
| 26 | Number of Samples | R/W |
| 27 | High High Alarm | R/W |
| 28 | High Alarm | R/W |
| 29 | Low Alarm | R/W |
| 30 | Low Low Alarm | R/W |
| 31 | Configuration Command | R/W |

# Voltage Input Module M/N 61C542 (continued)

| Terminal Block Lable | Voltage Input Function |
|---|---|
| 0A | Voltage Circuit 0: Analog Common |
| 0B | Voltage Circuit 0: ±10V Input |
| S | Voltage Circuit 0: Cable Shield |
| S | Voltage Circuit 1: Cable Shield |
| 1A | Voltage Circuit 1: Analog Common |
| 1B | Voltage Circuit 1: ±10V Input |
| 2A | Voltage Circuit 2: Analog Common |
| 2B | Voltage Circuit 2: ±10V Input |
| S | Voltage Circuit 2: Cable Shield |
| S | Voltage Circuit 3: Cable Shield |
| 3A | Voltage Circuit 3: Analog Common |
| 3B | Voltage Circuit 3: ±10V Input |
| 4A | Voltage Circuit 4: Analog Common |
| 4B | Voltage Circuit 4: ±10V Input |
| S | Voltage Circuit 4: Cable Shield |
| S | Voltage Circuit 5: Cable Shield |
| 5A | Voltage Circuit 5: Analog Common |
| 5B | Voltage Circuit 5: ±10V Input |
| 6A | Voltage Circuit 6: Analog Common |
| 6B | Voltage Circuit 6: ±10V Input |
| S | Voltage Circuit 6: Cable Shield |
| S | Voltage Circuit 7: Cable Shield |
| 7A | Voltage Circuit 7: Analog Common |
| 7B | Voltage Circuit 7: ±10V Input |
| 8A | Voltage Circuit 8: Analog Common |
| 8B | Voltage Circuit 8: ±10V Input |
| S | Voltage Circuit 8: Cable Shield |
| S | Voltage Circuit 9: Cable Shield |
| 9A | Voltage Circuit 9: Analog Common |
| 9B | Voltage Circuit 9: ±10V Input |
| 10A | Voltage Circuit 10: Analog Common |
| 10B | Voltage Circuit 10: ±10V Input |
| S | Voltage Circuit 10: Cable Shield |
| S | Voltage Circuit 11: Cable Shield |
| 11A | Voltage Circuit 11: Analog Common |
| 11B | Voltage Circuit 11: ±10V Input |
| 12A | Voltage Circuit 12: Analog Common |
| 12B | Voltage Circuit 12: ±10V Input |
| S | Voltage Circuit 12: Cable Shield |
| S | Voltage Circuit 13: Cable Shield |
| 13A | Voltage Circuit 13: Analog Common |
| 13B | Voltage Circuit 13: ±10V Input |
| 14A | Voltage Circuit 14: Analog Common |
| 14B | Voltage Circuit 14: ±10V Input |
| S | Voltage Circuit 14: Cable Shield |
| S | Voltage Circuit 15: Cable Shield |
| 15A | Voltage Circuit 15: Analog Common |
| 15B | Voltage Circuit 15: ±10V Input |

# RTD Module
# M/N 61C544

| Register Number | Description | Access |
|---|---|---|
| 0 | Channel 0 A/D Data | R |
| 1 | Channel 1 A/D Data | R |
| 2 | Channel 2 A/D Data | R |
| 3 | Channel 3 A/D Data | R |
| 4 | Channel 4 A/D Data | R |
| 5 | Channel 5 A/D Data | R |
| 6 | Channel 6 A/D Data | R |
| 7 | Channel 7 A/D Data | R |
| 8 | Channel 8 A/D Data | R |
| 9 | Channel 9 A/D Data | R |
| 10 | Channel 10 A/D Data | R |
| 11 | Channel 11 A/D Data | R |
| 12 | Channel 12 A/D Data | R |
| 13 | Channel 13 A/D Data | R |
| 14 | Channel 14 A/D Data | R |
| 15 | Channel 15 A/D Data | R |
| 16 | High High Alarm Status | R |
| 17 | High Alarm Status | R |
| 18 | Low Alarm Status | R |
| 19 | Low Low Alarm Status | R |
| 20 | Out of Range Status | R |
| 21 | Channel Configuration Status | R |
| 22 | Configuration Status | R |
| 23 | Channel Number | R/W |
| 24 | Reserved | |
| 25 | Reserved | |
| 26 | Number of Samples | R/W |
| 27 | High High Alarm | R/W |
| 28 | High Alarm | R/W |
| 29 | Low Alarm | R/W |
| 30 | Low Low Alarm | R/W |
| 31 | Configuration Command | R/W |

# RTD Module
# M/N 61C544 (continued)

| Term. Block Pin Number | RTD Function Ch. 0-7 D-Shell | RTD Function Ch. 8-15 D-Shell |
|---|---|---|
| 1 | Current Out Ch. 0 | Current Out Ch. 8 |
| 2 | Current Return Ch. 0 | Current Return Ch. 8 |
| 3 | RTD Voltage (+) Ch. 0 | RTD Voltage (+) Ch. 8 |
| 4 | RTD Voltage (−) Ch. 0 | RTD Voltage (−) Ch. 8 |
| 5 | Shield | Shield |
| 6 | Shield | Shield |
| 7 | Current Out Ch. 1 | Current Out Ch. 9 |
| 8 | Current Return Ch. 1 | Current Return Ch. 9 |
| 9 | RTD Voltage (+) Ch. 1 | RTD Voltage (+) Ch. 9 |
| 10 | RTD Voltage (−) Ch. 1 | RTD Voltage (−) Ch. 9 |
| 11 | Current Out Ch. 2 | Current Out Ch. 10 |
| 12 | Current Return Ch. 2 | Current Return Ch. 10 |
| 13 | RTD Voltage (+) Ch. 2 | RTD Voltage (+) Ch. 10 |
| 14 | RTD Voltage (−) Ch. 2 | RTD Voltage (−) Ch. 10 |
| 15 | Shield | Shield |
| 16 | Shield | Shield |
| 17 | Current Out Ch. 3 | Current Out Ch. 11 |
| 18 | Current Return Ch. 3 | Current Return Ch. 11 |
| 19 | RTD Voltage (+) Ch. 3 | RTD Voltage (+) Ch. 11 |
| 20 | RTD Voltage (−) Ch. 3 | RTD Voltage (−) Ch. 11 |
| 21 | Current Out Ch. 4 | Current Out Ch. 12 |
| 22 | Current Return Ch. 4 | Current Return Ch. 12 |
| 23 | RTD Voltage (+) Ch. 4 | RTD Voltage (+) Ch. 12 |
| 24 | RTD Voltage (−) Ch. 4 | RTD Voltage (−) Ch. 12 |
| 25 | Shield | Shield |
| 26 | Shield | Shield |
| 27 | Current Out Ch. 5 | Current Out Ch. 13 |
| 28 | Current Return Ch. 5 | Current Return Ch. 13 |
| 29 | RTD Voltage (+) Ch. 5 | RTD Voltage (+) Ch. 13 |
| 30 | RTD Voltage (−) Ch. 5 | RTD Voltage (−) Ch. 13 |
| 31 | Current Out Ch. 6 | Current Out Ch. 14 |
| 32 | Current Return Ch. 6 | Current Return Ch. 14 |
| 33 | RTD Voltage (+) Ch. 6 | RTD Voltage (+) Ch. 14 |
| 34 | RTD Voltage (−) Ch. 6 | RTD Voltage (−) Ch. 14 |
| 35 | Shield | Shield |
| 36 | Shield | Shield |
| 37 | Current Out Ch. 7 | Current Out Ch. 15 |
| 38 | Current Return Ch. 7 | Current Return Ch. 15 |
| 39 | RTD Voltage (+) Ch. 7 | RTD Voltage (+) Ch. 15 |
| 40 | RTD Voltage (−) Ch. 7 | RTD Voltage (−) Ch. 15 |

# 8-Channel Isolated Thermocouple and Low Level Input Module M/N 61C605

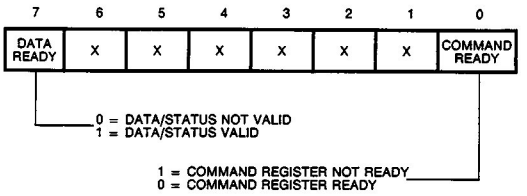
### Memory Map

| Hex Bus Address | Function | Register Name | Description |
|---|---|---|---|
| 2S0000<br>2S0000 | Write<br>Read | Data ready | Interrupt enable<br>Read data ready status |
| 2S0001 | Write | Command | Write analog data command |
| 2S0001 | Read | Status | Read analog data status |
| 2S0002 | Read | A/D data (low) | Read analog data low byte |
| 2S0003 | Read | A/D data (high) | Read analog data high byte |

Note: The 61C605 is programmed using BASIC IOWRITE statements and IOREAD% functions.

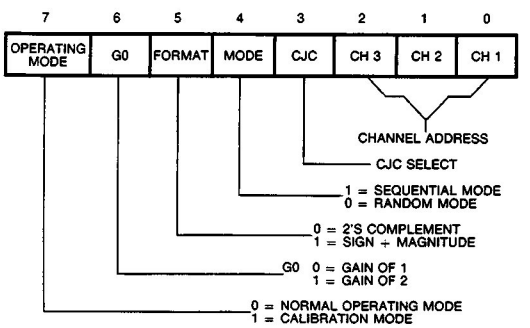
### Data Ready Register

LOCATION: BASE + 0




### Command Register

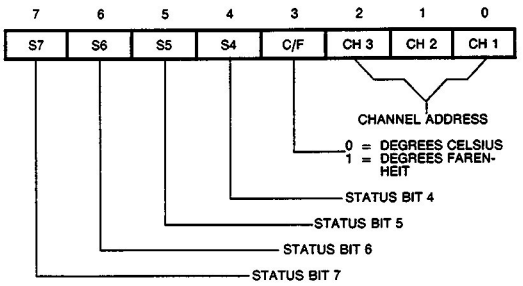
LOCATION: BASE + 1



# M/N 61C605
# (continued)

**Status Register**

LOCATION: BASE + 1




| Term. | Description |
|---|---|
| 1 | N/C |
| 2 | N/C |
| 3 | N/C |
| 4 | Channel 0 High |
| 5 | Channel 0 Low |
| 6 | −12 V DC |
| 7 | Channel 1 High |
| 8 | Channel 1 Low |
| 9 | −12 V DC |
| 10 | Channel 2 High |
| 11 | Channel 2 Low |
| 12 | −12 V DC |
| 13 | Channel 3 High |
| 14 | Channel 3 Low |
| 15 | −12 V DC |
| 16 | N/C |
| 17 | N/C |
| 18 | N/C |
| 19 | N/C |
| 20 | Ground |
| 21 | Ground |
| 22 | N/C |
| 23 | Channel 4 High |
| 24 | Channel 4 Low |
| 25 | −12 V DC |
| 26 | Channel 5 High |
| 27 | Channel 5 Low |
| 28 | −12 V DC |
| 29 | Channel 6 High |
| 30 | Channel 6 Low |
| 31 | −12 V DC |
| 32 | Channel 7 High |
| 33 | Channel 7 Low |
| 34 | −12 V DC |
| 35 | N/C |
| 36 | N/C |

N/C = no connection.

# 16-Channel Analog Input Module M/N 61C613

**Register Assignments**

| | |
|---|---|
| Location (M+0) | Status/error from module; command from software |
| Location (M+1) | Command from software |
| Location (M+2) | Return value from module or software |
| Location (M+3) | Not ready/ready from module |

M = base address in Multibus (byte) format.

Note that these registers cannot be monitored.

| Term. | Signal | Description |
|---|---|---|
| 1 | | N/C |
| 2 | + Signal | Channel 0 |
| 3 | – Signal | Channel 0 |
| 4 | | N/C |
| 5 | | N/C |
| 6 | | N/C |
| 7 | | N/C |
| 8 | + Signal | Channel 1 |
| 9 | – Signal | Channel 1 |
| 10 | | N/C |
| 11 | | N/C |
| 12 | | N/C |
| 13 | | N/C |
| 14 | + Signal | Channel 2 |
| 15 | – Signal | Channel 2 |
| 16 | | N/C |
| 17 | | N/C |
| 18 | | N/C |
| 19 | | N/C |
| 20 | + Signal | Channel 3 |
| 21 | – Signal | Channel 3 |
| 22 | | N/C |
| 23 | | N/C |
| 24 | | N/C |
| 25 | | N/C |
| 26 | + Signal | Channel 4 |
| 27 | – Signal | Channel 4 |
| 28 | | N/C |
| 29 | | N/C |
| 30 | | N/C |
| 31 | | N/C |
| 32 | + Signal | Channel 5 |
| 33 | – Signal | Channel 5 |
| 34 | | N/C |
| 35 | | N/C |
| 36 | | N/C |

# M/N 61C613 (continued)

| Term. | Signal | Description |
|---|---|---|
| 37 | | N/C |
| 38 | + Signal | Channel 6 |
| 39 | – Signal | Channel 6 |
| 40 | | N/C |
| 41 | | N/C |
| 42 | | N/C |
| 43 | | N/C |
| 44 | + Signal | Channel 7 |
| 45 | – Signal | Channel 7 |
| 46 | | N/C |
| 47 | | N/C |
| 48 | | N/C |
| 49 | + Signal | Temp Sensor |
| 50 | – Signal | Temp Sensor |

Terminal layout is shown for second connector; third connector has same layout (add 8 to Channel designation numbers above). Note that the first connector is not used.

# Section III
# Programming
# Reference

# BASIC
# Language Quick Reference

This section is an alphabetical listing of statements and functions in Reliance® Enhanced BASIC Language. The format of each listing is organized as follows. Each listing begins with the keyword. The line following the keyword begins with a letter designation indicating the type of instruction:

- (s) = BASIC statement (may be allowed in CONTROL BLOCK also)
- (f) = BASIC function (may be allowed in CONTROL BLOCK also)
- (c) = BASIC configuration task statement (for AutoMax versions up to and including 2.1)

After the letter designation comes a description of the listing and the format of the listing. Fields in capital letters must be entered as shown. This includes any special characters attached to fields, e.g., "%". Fields shown in lower-case represent specific information the programmer must enter in upper-case. Note that the underscore character "_" is a valid alphanumeric character used for clarification purposes only, i.e., to divide words at logical points.

Example:

The function

```
BLOCK_MOVE@(source%, dest%, size)
```

could be implemented in an IF-THEN-ELSE statement in the following manner:

```
1000 IF BLOCK_MOVE@(NETW_1_32%, NETW_2_32%, 32) THEN
        STOP
     ELSE
        11000
     END_IF
```

Following the listing format is a description of the fields the programmer must enter and the variable types permitted. At the very end of each listing is an example.

For detailed information about the BASIC language, refer to J-3675.

**Variable Types**

Five types of variables are used in BASIC:

1. Single integer (16-bit) variables (value range −32768 — +32767)
   Terminating character: %
2. Double integer (32-bit) variables (value range −2147483648 — +2147483648)
   Terminating character: !

3. Real variables (value range $9.2233717 \times 10^{18}$ to $5.4210107 \times 10^{-20}$ and $-9.2233717 \times 10^{18}$ to $-2.7105054 \times 10^{-20}$)
   Terminating character: none
4. Boolean (1 bit) variables [value range TRUE (ON) or FALSE (OFF)]
   Terminating character: @
5. String variables (value range 1 — 255 characters, must begin with letter or underscore)
   Terminating character: $

BASIC also permits array variables of up to four dimensions. Array subscripts must be positive integer variables or expressions in the range of 0 — 32767.

# BASIC Language Listing

### ABDEF

(c) Defines a variable for the Allen-Bradley® Interface module (M/N 57C418).

```
ABDEF var_name[SLOT=slot_number, FILE=file,                    &
                REGISTER=register_number, BIT=bit_number]
var_name = integer, real or boolean variable name for register or bit; integers and reals limited to integer files; booleans limited to binary files
slot_number = slot number of the interface module
file = file designation for the dual port interface area; must be enclosed in single or double quotes:
```

| File | Register Number Range |
|---|---|
| B0=Binary file 0 | 0<=255 |
| B1=Binary file 1 | 0<=255 |
| N0=Integer file 0 | 0<=1023 |
| N1=Integer file 1 | 0<=1023 |

```
register_number = register number within the file
bit_number = bit number (0 — 15) within the register; specified for booleans only
```

Example:

```
1000 ABDEF RETURN%[SLOT=8, FILE=N1, REGISTER=975]
```

### ABS

(f) Returns absolute value of an expression in real or integer format, depending upon input format.

```
ABS(expression)

expression = numeric (integer or real)
```

Example:

```
1000 VALUE1=ABS(VALUE%*2)
```

### ASC%

(f) Returns decimal ASCII value of single character string.

```
ASC%(string)

string = string variable or expression
```

Example:

```
1000 NUMBER%=ASC%(STRING1$)
```

### ATN

(f) Returns value (in real format) equal to arctangent of the input.

```
ATN(expression)
```

expression = numeric (integer or real) representing radians

Example:

```
1000 RADIANS=ATN(ANGLE)
```

### BCD_IN%

(f) Returns decimal value of BCD input.

```
BCD_IN%(expression)
```

expression = single or double variable or expression; value range ≤9999 hex

Example:

```
1000 SWITCH_VALUE%=BCD_IN%(INPUT_CARD%)
```

### BCD_OUT%

(f) Returns BCD value of a decimal number.

```
BCD_OUT%(expression)
```

expression = single or double variable or expression; value range ≤9999 hex

Example:

```
1000 IN_VALS=BCD_OUT%(SWITCH_VALS%)
```

### BINARY$

(f) Returns the binary form of the input as a string of 1s and 0s.

```
BINARY$(expression)
```

expression = integer or integer expression

Example:

```
1000 BITS$ = BINARY$(NUMBER_1%)
```

### BIT_CLR@

(f) Tests if bit is OFF (FALSE) at specified bit position.

```
BIT_CLR@(variable, bit_number)
```

variable = single or double integer
bit_number = bit within the variable to test; 0 — 15 for single integer, 0 — 31 for double integer

Example (in IF-THEN-ELSE statement):

```
1000 IF BIT_CLR@(BIT_VALS%,2) THEN
        2500
     ELSE
        3000
     END_IF
```

### BIT_MODIFY@

(f) Sets or clears the specified bit in the specified variable based on the selected option. Function is TRUE if bit change operation is completed. Function is FALSE if bit change operation is not completed. FALSE can occur if the specified variable is forced or if the bit is already in the correct state.

```
BIT_MODIFY@(variable, bit_number, option)

variable = single or double integer variable
bit_number = bit within the variable to test; 0 — 15 for single integer, 0 —
             31 for double integer
option = defines the change to be made to the bit; integer or boolean expres-
         sion

    Option 0 = unconditionally set bit to 0
           1 = unconditionally set bit to 1
           2 = if bit is 0, then set to 1
           3 = if bit is 1, then set to 0
```

Example (in IF-THEN-ELSE statement):

```
1000 IF BIT_MODIFY@(BIT_VALS%,2,SYSTEM_RUNNING@) THEN
        3000
     ELSE
        4000
     END_IF
```

### BIT_SET@

(f) Tests if specified bit is ON (TRUE).

```
BIT_SET@(variable,bit_number)

variable = single or double integer variable
bit_number = bit within the variable to test; range 0 — 15 for single integer,
             0 — 31 for double integer
```

Example (in IF-THEN-ELSE statement):

```
1000 IF BIT_SET@(BIT_VALS%,1) THEN
        2000
     ELSE
        3000
     END_IF
```

### BLOCK_MOVE@

(f) Moves a block of registers to and from the network.

```
BLOCK_MOVE@(source, dest, size)

source = boolean, integer, double integer, or real variable or one-dimensional
         integer array specifying starting point of registers to move
dest = integer variable or one-dimensional array specifying starting point of
       area to which registers are to be moved
size = number of registers (16-bit words) to move
```

Example (in IF-THEN-ELSE statement):

```
1000 IF BLOCK_MOVE@(NETW_1_32%, NETW_2_32%, 32) THEN
        STOP
     ELSE
        11000
     END_IF
```

### CHR$

(f) Returns a string character equal to decimal ASCII value of the input expression.

CHR$(expression)

expression = integer (hexadecimal equivalent permitted; must be terminated with H)

Example:

```
1000 A$=CHR$(41H)
```

### CLOSE

(s) De-allocates a channel or port.

CLOSE #device_number

device_number = logically assigned (through OPEN statement) device number; range 1 — 255

Example:

```
1000 CLOSE #1
```

### CLR_ERRLOG

(f) Clears the entire error log for the task, regardless of the number of logged errors.

CLR_ERRLOG

Example:

```
1000 CLR_ERRLOG
```

### COMMON

(s) Defines a common variable accessible to all tasks in the system.

COMMON variable

variable = simple or subscripted variable of any type; more than one variable is permitted in the statement if separated by commas; string variable length can be specified by adding the following immediately after the variable, or directly between the variable and the array specification:

:n

where n is the maximum string length (range 1 — 255); if not specified, default maximum is 31.

Example:

```
1000 COMMON COIL@, STRING5:15(100), NUM2!
```

### CONVERT%

(f) Converts data formats

CONVERT% (src_variable, src_subscript, dest_variable, dest_subscript, & num_of_words, mode)

scr_variable = the variable that selects where to get data from. This parameter may be a scalar or an array of any data type. If src_variable is an array, it sould be the base name and any data type character only.

src_subscript = only used if the src_variable is an array. If determines where in the array to begin reading. If not an array, the value should be 0.

dest_variable = the variable that selects where to move the data. This parameter may be a scalar or an array of any data type. If dest_variable is an array, it should only be the base name and any data type character.

dest_subscript = only used if destination_variable is an array. It determines where in the array to begin writing. If not an array, the value should be 0.

num_of_words = selects the number of words to move.
mode = determines the mode of operation.

| Value | Function |
|---|---|
| 0 | Move data with no change in format |
| 1 | Convert from Motorola Floating Point to IEEE format |
| 2 | Convert from IEEE Floating Point to Motorola format |
| 4 | Word swap (0102H to 0201H) |
| 8 | Long word swap (01020304H to 04030201H) |
| 9 | Motorola to IEEE followed by long word swap |
| 10 | Long word swap followed by IEEE to Motorola |

Example:

```
STATUS@ = CONVERT%(SRC_ARRAY, 10, DST_ARRAY, 20, 60, 9)
```

### COS

(f) Returns value (in real format) equal to cosine of input.

```
COS(expression)
```

expression = numeric (integer or real) representing radians

Example:

```
1000 COS(ANGLE)
```

### DATA

(s) See READ-DATA.

### DELAY

(s) Delays program execution by specified time period.

```
DELAY n units
```

n = arithmetic expression or constant that is evaluated to an integer result
units = type of unit to delay program execution; available units are HOURS, MINUTES, SECONDS, and TICKS (1 TICK = 5.5 milliseconds)

Example:

```
1000 DELAY 5 TICKS
```

### END

(s) (c) Denotes physical end of program; required in Control Block programs.

```
END
```

Example:

```
20000 END
```

### EVENT NAME (software)

(s) Defines a software event; used with SET and WAIT ON.

```
EVENT NAME = event_name
```

event_name = symbolic name for an event

Example:

```
2000 EVENT_NAME = SW_EVENT_1
```

### EVENT NAME (hardware)

(s) Defines a hardware event; used with SET and WAIT.

```
EVENT NAME = event_name
    INTERRUPT_STATUS = I/O_variable
    TIMEOUT = timeout_count
```

event_name = symbolic name given to particular event
I/O_variable = name of a common symbol that references the address of an interrupt register on a module that handles interrupts
timeout_count = specifies the maximum amount of time that can pass before a hardware event occurs; range 1 — 32767 ticks (1 tick = 5.5 milliseconds)

Example:

```
1000 EVENT NAME = MARKER_PULSE
        INTERRUPT_STATUS = RESOLVER_INTREG%
        TIMEOUT = 100
```

### EXP

(f) Returns a real value equal to (e ** expression), where e = 2.71828.

```
EXP(expression)
```

expression = numeric (integer or real) expression

Example:

```
1000 NUMBER = EXP(2*3)
```

### FINDVAR!

(f) Accepts a variable name as a string expression and returns a pointer to that variable.

```
FINDVAR!(varname$)
```

varname$ = a string expression for the name of the variable to find.

Example:

```
POINTER! = FINDVAR!(VARIABLE_NAME$)
```

### FIX

(f) Returns the whole part of a real number.

```
FIX(expression)
```

expression = a real variable or expression

Example:

```
1000 WHOLE_PART = FIX(REAL_VALUE)
```

### FOR-NEXT

(s) Performs repetitive program looping based on parameters specified.

```
FOR variable = expression_1 TO expression_2,                    &
               STEP expression_3
...
...
NEXT variable
```

variable = simple numeric variable used as loop index
expression_1 = initial value of index; any numeric expression
expression_2 = terminating value of index; any numeric expression
expression_3 = optional; incremental value of index; any numeric expression; default is +1

Example:

```
1000 FOR M% = 1100 TO 1500 STEP 100
...
...
3000 NEXT M%
```

## GATEWAY_CMD_OK@

(f) Boolean function used for register transfers to interface modules. Used with VARPTR! function.

```
GATEWAY_CMD_OK@(status,cmd_code,slave_drop,                    &
                    slave_reg,master_var, num_regs)
```

status = integer variable name where gateway command is stored
cmd_code = variable name or expression representing command sent to interface module; must be type integer
slave_drop = variable name or expression of type integer; device address of the slave for which the command is intended
slave_reg = a string variable or expression describing the starting register in the slave that is to be read or written
master_var = variable name (using VARPTR! function) or expression representing the address of the first register in the master from or to which data is to be written; must be type double integer
num_regs = variable name or expression defining the number of registers to be transferred; must be of type integer

Example (including VARPTR! function, in IF-THEN-ELSE statement):

```
1500 MSTR_REG! = VARPTR!(MASTER)
2000 IF NOT GATEWAY_CMD_OK@(RET_STAT1%,MOD_REG%,          &
        SLVDROP%,SLVREG$,MSTR_REG!,XFER_SIZE%) THEN       &
           2500                                           &
        ELSE                                              &
           3000                                           &
        END_IF
```

## GBLDEF

(c) Defines common variables accessed through the Network Communications module (M/N 57C404 or M/N 57C404A). Position information is stored in a separate file with the extension .NET.

```
GBLDEF variable [SLOT=s], NETWORK="n"
[, NET_NAME=name]
```

variable = the name that will be used in this drop to reference the network variable; all variable types except string are supported; although reals and double integer variables can also be used, it is strongly recommended that they be avoided because of the possibility that all 32 bits will not transfer in one operation.

s = slot that contains the Network Communication module (M/N 57C404 or 57C404A) in this rack; range 0–15.

n = name of the .NET file from which to extract the rest of the definition for this variable; range is a single letter from A to Z in quotation marks; if not specified, default is A.

name = the name that will be used on the network to reference the network variable; this field is used only when you want the name of the network variable to be different on the network ("name" in the .NET file) than it is in this rack; the name must be unique within the network; if the field is not specified, the default is "variable".

Example:

```
60   GBLDEF REF_1% [SLOT=3, NETWORK="B", NET_NAME=SPEED%]
```

### GET

(s) Inputs a single character from a device into a string variable.

```
GET #n:EMPTY=m, string_variable
```

n = the logical number assigned to a device using an OPEN statement; if not entered, default is PORTA on the Processor module on which the task resides

m = :EMPTY=m is an optional parameter; m specifies line number to which to transfer control if the channel is empty

string_variable = a string variable which will be loaded with one character from the device

Example:

```
1000 GET #2:EMPTY=1100, A$
```

### GOSUB-RETURN

(s) GOSUB unconditionally transfers program control to the specified line number until RETURN statement. RETURN returns program control to first numbered program line following the GOSUB or ON GOSUB statement that caused program control to be diverted to a particular error-handling routine.

```
GOSUB n
...
...
RETURN
```

n = integer or integer expression representing line number

Example:

```
1000 GOSUB 2000
...
...
3000 RETURN
```

### GOTO

(s) Unconditionally transfers program control to specified line number.

```
GOTO n
```

n = integer or expression representing line number

Example:

```
1000 GOTO 2000
```

### HEX$

(f) Returns the hexadecimal value of the input as a string.

```
HEX$(expression)
```

expression = integer or integer expression

Example:

```
1000 HEX_VAL$ = HEX$(X%)
```

## INCLUDE

(s) Inserts a DOS file containing statements into the task as it is compiling.

```
INCLUDE "filename.INC"
```

filename = name of the DOS file containing the statements to be included; drive and subdirectory specification cannot be included; the file itself must have only one statement per line and must not have line numbers as these will be generated by the compiler in increments of 1, beginning with the INCLUDE statement line number; no INCLUDE statements are permitted in the file itself; the extension .INC is required in the filename

Example:

```
1000 INCLUDE "IODEFS.INC"
```

## IF-THEN-ELSE

(s) Conditionally executes statements following THEN or transfers program control based on the results of the relational expression. If the expression is true, the statement following THEN is executed. If the expression is false, the statement following ELSE is executed.

```
IF boolean_exp THEN
    statement
ELSE
    statement
END_IF
```

boolean_exp = boolean variable or relational expression; if this parameter must be extended to the second line [and the ampersand (&) added to the end of the line to indicate it continues], all of the following lines except the last must also include the & at the end

statement = either program line to be executed, or BASIC statement or series of statements separated by backslashes or colons

Example:

```
1000 IF (A > 3) THEN
    3000
ELSE
    A = 3
END_IF
```

## INPUT

(s) Reads data from a device or channel.

```
INPUT #n:EMPTY=m, input_list
```

n = optional; logical device number assigned using OPEN statement; range 1 — 255; default is Processor PORTA

m = parameter :EMPTY=m is optional and specified only for channels and not devices; specifies line number to which to transfer control if the channel is empty

input_list = list of variables to be read, separated by commas; simple or subscripted variables

Example:

```
10000 INPUT #1:EMPTY=10200, A%,B%
```

## IODEF (standard addressing)

(c) Defines I/O in local rack using standard DCS5000/AutoMax addressing.

```
IODEF variable_name[SLOT=slot_number,                                    &
      REGISTER=reg_number, BIT=bit_number]
```

variable_name = integer, double integer or boolean variable
slot_number = slot number of the I/O module in the local chassis; range 0 — 15
reg_number = register number or the I/O module; range 0 — 32767 16-bit registers; default is 0 if not specified
bit_number = bit number in the register; specified for booleans only; range 0 — 15

Example:

```
1000 IODEF DIGITAL_IN%[SLOT=3, REGISTER=1, BIT=4]
```

## IODEF (hexadecimal addressing)

(c) Defines I/O in a local rack using hexadecimal addressing.

```
IODEF variable_name[ADDRESS=hex_addressH, BIT=bit_number]
```

hex_address = specific hexadecimal address of the I/O location; must be a word (16 bits) or even address with the lower order bit equal to zero; the address always starts with 2, followed by the hexadecimal slot number and the four-digit hexadecimal byte number
variable_name = integer, double integer or boolean variable
bit_number = bit number in the register; specified for booleans only; range 0 — 15

Example:

```
1000 IODEF RELAY_1@[ADDRESS=260000H,BIT=1]
```

## IOREAD%

(f) Returns an integer value obtained by reading data from an I/O module.

```
IOREAD%(option,addressH)
```

option = integer variable or expression that defines the type of READ access to perform
- option 1 = byte read
- option 2 = double byte read:
  - address = LSB
  - address + 1 = MSB
  - (used for foreign I/O modules only)
- option 3 = double byte read:
  - address = MSB
  - address + 1 = LSB
- option 4 = 32-bit word read:
  - address = MSB
  - address + 1 = next byte
  - address + 2 = next byte
  - address + 3 = LSB

address = double integer variable or expression that contains the address data is to be read from; range ≥220000H

Example:

```
1000 X%=IOREAD%(2, 220000H)
```

## IOWRITE

(s) Outputs data to an I/O module.

```
IOWRITE%(option, data, addressH)

option = integer variable or expression that defines the type of READ access
         to perform
         option 1 = byte read
         option 2 = double byte read:
                    address = LSB
                    address + 1 = MSB
                    (used for foreign I/O modules only)
         option 3 = double byte read:
                    address = MSB
                    address + 1 = LSB
         option 4 = 32-bit word read:
                    address = MSB
                    address + 1 = next byte
                    address + 2 = next byte
                    address + 3 = LSB
data = integer variable name or expression defining data to output
address = double integer variable or expression that contains the address
          data is to be read from; range ≥220000H
```

Example:

```
1000 IOWRITE(3, DATA_1,230000H)
```

## LEFT$

(f) Returns a substring of specified length from the leftmost portion of a string.

```
LEFT$(string, string_length)

string = a string variable or expression; string variables must be surrounded
         by single or double quotes
string_length = the number of characters to take from the left side of the
                string
```

Example:

```
1000 SUB_STRING = LEFT$('ABCDEFG',4)
```

## LEN%

(f) Returns a value equal to the character length of the specified string variable or expression.

```
LEN%(string)

string = a string variable or expression
```

Example:

```
1000 LENGTH% = LEN%('STRING1')
```

## LET

(s) Assigns a value to a variable.

```
LET variable = expression

variable = simple or subscripted variable of any type
expression = constant or expression
```

Example:

```
1000 LET SPEED% = 25
```

## LN

(f) Returns a real value (in real format) equal to the natural log of the input.

```
LN(expression)

expression = numeric (integer or real) expression
```

Example:

```
1000 NATURAL_LOG = LN(REFERENCE!+GAIN%)
```

### LOCAL (simple or subscripted variable)

(s) Defines a variable used in the local application task only; opposite of COMMON.

```
LOCAL variable
```

variable = simple or subscripted variable of any type; more than one variable is permitted in the statement if separated by commas; string variable length can be specified by adding the following immediately after the variable, or directly between the variable and the array specification:

```
:n
```

where n is the maximum string length (range 1 — 255); if not specified, default maximum is 31.

Example:

```
1000 LOCAL ABC%, STRING1$:50, STRING2$:10(100)
```

### LOCAL (tunable variable)

(s) Defines a tunable variable used in the local application task only.

```
LOCAL simple_variable[CURRENT=value_1,HIGH=value_2,                &
                      LOW=value_3, STEP=value_4]
```

simple_variable = tunable variable of integer, double integer or real type
value_1 = initial current value
value_2 = the highest value the variable can achieve
value_3 = the lowest value the variable can achieve
value_4 = the amount that the operator can change the value by incrementing or decrementing it through the executive software

Example:

```
1000 LOCAL TENSION_GAIN%[CURRENT=25,HIGH=50,LOW=10,                &
                         STEP=5]
```

### MEMDEF

(c) Defines common variables which do not have physical I/O associated with them. These variables are cleared (set to 0, FALSE, or OFF) during a STOP ALL condition.

```
MEMDEF variable
```

variable = simple or subscripted variable of any type; multiple variables are permitted if separated by commas

Example:

```
1000 MEMDEF ERROR_MESSAGE$, SHEET_COUNT!
```

### MID$

(f) Returns a substring from a string, starting and ending with the specified character positions.

```
MID$(string, start, end)
```

string = string variable or expression; string variables must be surrounded by single or double quotation marks
start = starting character position of substring
end = ending character position of the substring

Example:

```
1000 SUBSTRING_1$ = MID$('ABCDEFG',2,3)
```

### MODDEF

(c) Defines I/O accessed through the Modbus Interface module (M/N 57C414).

```
MODDEF var_name[SLOT=slot, REGISTER=register]

var_name  = integer or boolean variable
slot      = slot number of the I/O module
register  = register number on the I/O module; register value range:
            integer variables: 30001 — 41024
            boolean variables: 1 — 14096
```

Example:

```
1000 MODDEF COILREF%[SLOT=7,REGISTER=30001]
```

### NETDEF

(c) Defines common variables accessed through the Network Communications module (M/N 57C404 or M/N 57C404A).

```
NETDEF var_name[SLOT=slot,DROP=drop,REGISTER=register,          &
                BIT=bit]

var_name  = name of register or bit; integer or boolean variable
slot      = slot number of Network module in the rack on which the task will run;
            range 0 — 15
drop      = drop number of the network drop where var_name is stored; range 0
            — 43
register  = register number on the Network module on which var_name is
            stored; range 0 — 63
bit       = bit number of the bit in the register number on the Network module;
            range 0 — 15
```

Example:

```
1000 NETDEF LINEREF%[SLOT=6,DROP=0,REGISTER=32,BIT=0]
```

### NEXT

See FOR-NEXT.

### NVMEMDEF

(c) Defines common variables not associated with physical I/O and that retain their values in the event of a power failure or STOP ALL condition.

```
NVMEMDEF variable

variable  = simple or subscripted variable of any data type; multiple variables
            are permitted if separated by commas
```

Example:

```
1000 NVMEMDEF REV_COUNT!, GEAR_RATIOS%(10,10)
```

### ON ERROR GOTO

(s) Unconditionally transfers program control to specified statement line if a non-fatal error occurs.

```
ON ERROR GOTO line_number

line_number  = line number where control should be transferred in the event
               of an error; typically where an error-handling routine begins
```

Example:

```
1000 ON ERROR GOTO 2000
```

## ON GOSUB

(s) Conditionally transfers program control to a subroutine at any one of the specified line numbers based on the result of an integer expression.

ON expression GOSUB line_number

expression = integer variable or arithmetic expression that results in an integer value
line_number = line number to which control is transferred depending upon the evaluated expression; multiple line numbers are separated by commas; see RETURN

Example:

```
1000 ON GAIN% GOSUB 2000,3000,4000
```

## ON GOTO

(s) Conditionally transfers program control to one of the specified line numbers based on the result of an integer expression.

ON expression GOTO line_number

expression = integer variable or arithmetic expression that results in an integer value; result determines which line number control is transferred to; if result = 1, control is transferred to first line number, etc.
line_number = line number to which control is transferred depending upon the evaluated expression; multiple line numbers are separated by commas

Example:

```
1000 ON A% GOTO 1100,1200,1300,1400
```

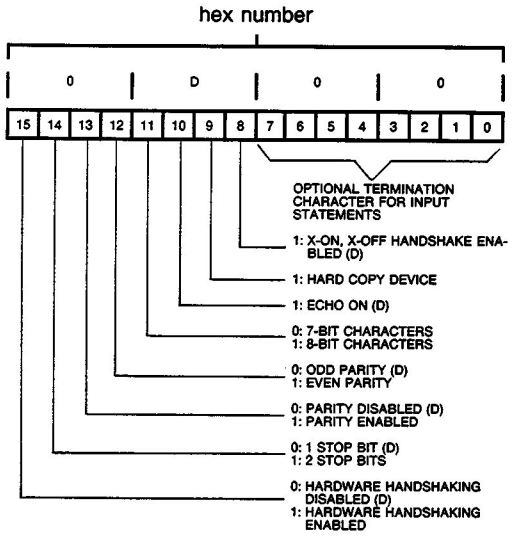
## OPEN

(s) Allocates a Processor port for exclusive use and equates a logical name with the port; used with INPUT and PRINT statements.

OPEN "device_name" AS FILE #n,SETUP=specs,baud_rate, &
ACCESS=status

device_name = pre-assigned character string that defines the name for the device; PORTA or PORTB
n = number assigned to the port; range 1 — 255
specs = a hexadecimal word constant or integer expression that describes a bit pattern defining various characteristics for the device; see below for more information on this parameter
baud_rate = baud rate of device
status = optional parameter; specifies whether this task has EXCLUSIVE or NON_EXCLUSIVE access to the device; if EXCLUSIVE, no other task can read or write to the device until this task closes it (with CLOSE); if this parameter is specified as EXCLUSIVE, specs and baud_rate must be enclosed in parentheses. If this parameter is specified as NON_EXCLUSIVE, specs and baud_rate cannot be specified in this statment. If not specified, the default is EXCLUSIVE

**Specifying the "specs" Parameter in OPEN Statements**




0D00 = Default Setting

## OPEN CHANNEL

(s) Equates a logical name with a data channel in the system, creating a data path between tasks. Allows tasks to communicate using the INPUT and PRINT statements. The OPEN CHANNEL statements in the tasks that are communicating must be exactly the same except for the depth parameter, which is specified only in tasks which want to input data from the channel.

```
OPEN CHANNEL FOR input_or_output AS FILE #n, TYPE=(type),          &
      DEPTH=depth
```

input_or_output = specifies whether the task is reading data from the channel (INPUT), or sending data through the channel (OUTPUT)

n = a logical number for that channel; range 1 - 255

type = list of different variable types that are to be passed through the channel; multiple types must be separated by commas:

- I = single integer
- D = double integer
- R = real
- S = string
- B = boolean

depth = how many messages a channel or data path can hold before it is considered full; integer constant or expression; specified only if the task is reading data from the channel, not writing

Example:

See J-3675 for more information.

## PRINT

(s) Outputs data to a device or channel; used with the OPEN statement.

```
PRINT #n:FULL=m, print_list
```

n = logical number assigned to device or channel using OPEN statement; range 1 — 255; if not specified, default is PORTA
m = :FULL=m is applicable only if n is a channel and not a device and is optional; m specifies the line number to which control is to be transferred if the channel is full
print_list = list of data items to be printed separated by commas or semi-colons

Example:

```
1000 PRINT #2, SPEED%, B%
```

## PRINT USING

(s) Outputs data to a device using the specified format.

```
PRINT USING #n, type_width:expression
```

n = logical number assigned to device or channel using OPEN statement; range 1 — 255; if not specified, default is PORTA
type_width = format type followed by field width with no spaces between; format types:

L = left justify
R = right justify
C = center
Z = load leading zeros in front
D = print in decimal format

field width: integer or integer expression in the range 1 — 132; D format field width =

integer1.integer2
integer1 = total field width; range 3 — 132
integer2 = number of characters after the decimal; range 0 — 26

expression = variable expression; multiple expressions must have the type_width parameter specified individually and must be separated by commas

Example:

```
1000 PRINT USING #1, L40:STRING1$, D3.1:SPEED
```

## PUT

(s) Outputs a single character to a device.

```
PUT #n, string_variable$
```

n = the logical number assigned to a device using an OPEN statement; if not specified, default is PORTA
string_variable = string variable

Example:

```
1000 PUT #2, A$
```

## READ-DATA

(s) Inputs data from a list of values, expressions, literals, or strings in the DATA statements; also used with RESTORE.

```
READ variable
DATA expression
```

variable = simple or subscripted variable of any type; multiple variables must be separated by commas
expression = expression loaded into the corresponding variable in the READ statement; the type must match the variable type in the READ statement; string constants must be enclosed in single or double quotes

Example:

```
1000 READ A%, C1!, Z$
...
...
2000 DATA 2, (83+19), "OVERSPEED"
```

### READVAR%

(f) Accepts a variable name as a string end returns the value in variable VALUE. Requires the Ethernet™ operating system in the rack.

READVAR%(vn$,value)

vn$ = a string expression for the name of the variable to read. It can be a boolean, integer, double integer, real or string, or an array of these types. Only one-dimensional arrays are allowed.
value = the variable where the value read is written.

Example:

```
STATUS% = READVAR%(VARIABLE_NAMES$,value)
```

### REM and !

(s) Either statement can be used to document the program with comments and explanations. ! statements are downloaded with the task onto the Processor module if the task was compiled with the /Reconstruct option; these statements require 50 µsec. to execute. The task can then be fully reconstructed and uploaded back to the personal computer at a later date. REM statements are stripped out when the task is compiled.

REM comment

or

! comment

comment = any text

Example:

```
1000 REM This is a non-reconstructible comment
2000 !This is a reconstructible comment
```

### RESTORE

(s) Restores internal data pointer to the first data item of the first DATA statement or to the first data item of the specified DATA statement line number.

RESTORE n

n = line number of DATA statement; integer or expression; if not specified, default is first DATA statement in the task

Example:

```
1000 RESTORE 5000
```

## RESUME

(s) Returns program control to first numbered program line following the ON ERROR statement that caused program control to be diverted to an error handling routine.

```
RESUME
```

Example:

```
1000 RESUME
```

## RETURN

(s) See GOSUB-RETURN.

## RIGHT$

(f) Returns a substring of specified length from the rightmost portion of a string.

```
RIGHT$(string, string_length)
```

string = string variable or expression; strings must be enclosed in single or double quotation marks
string_length = the number of characters to take from the right side of the string

Example:

```
1000 STRING1$ = RIGHT$("ABCDEF",4)
```

## RIODEF

(c) Defines I/O located in a remote rack.

```
RIODEF variable[MASTER_SLOT=master_slot,DROP=drop,                &
                SLOT=slot,REGISTER=register,BIT=bit]
```

variable = integer or boolean variable
master_slot = slot number of the remote I/O master module in the main, or local, rack; range 0 — 15
drop = drop number of the remote I/O rack; range 0 — 7
slot = slot number of the I/O module in the remote rack; range 0 — 15
register = register number on the remote I/O module; range 0 — 31; default is 0 if not specified
bit = bit number of the I/O point on the register; specified for boolean variables only; range 0 — 15

Example:

```
1000 RIODEF LEVEL%[MASTER_SLOT=15,DROP=3,SLOT=4,                  &
                   REGISTER=1]
```

## RNETDEF

(c) Defines I/O in the system connected via R-NET using the AutoMate Interface module (M/N 57C417).

```
RNETDEF variable[SLOT=slot,REGISTER=register]
```

variable = integer or boolean variable
slot = slot number of the 57C417 module; range 0 — 15
register = register number on the I/O module; format:

0000.BB

where 0000 is the register number in octal, and BB is the bit number in octal; bit number range 0 — 17

Registers reserved for integer variables:

| Register Number | Multibus Access | R-NET Access |
|---|---|---|
| 2000 — 3777 | r/w | read only |
| 4000 — 5777 | r/w | r/w |

Registers reserved for boolean variables:

| Register Number | Multibus Access | R-NET Access |
|---|---|---|
| 0000.00 — 0377.17 | read only | r/w |
| 400.00 — 777.17 | r/w | read only |

Example:

```
1000 RNETDEF GATEOK@[SLOT=6,REGISTER=400.00]
```

### ROTATEL%

(f) Returns an integer value equal to the integer expression that was input rotated the specific number of binary places to the left.

```
ROTATEL%(variable,count)
```

variable = a single or double integer variable or expression
count = number of bit positions to rotate the integer expression; bit 15 (or 31 for double integers) wraps around to bit 0

Example:

```
1000 MOVE1% = ROTATEL%(INPUT_CARD%, 4)
```

### ROTATER%

(f) Returns an integer value equal to the integer expression that was input rotated the specific number of binary places to the right.

```
ROTATER%(variable,count)
```

variable = a single or double integer variable or expression
count = number of bit positions to rotate the integer expression; bit 0 wraps around to bit 15 (or 31 for double integers)

Example:

```
1000 SWAP1% = ROTATER%(INPUT_CARD%, 3)
```

### SET_MAGNITUDE

(s) Assigns 16 or 32-bit hexadecimal values to variables.

```
SET_MAGNITUDE(variable,value)
```

variable = numeric simple variable of any type
value = numeric constant or expression

Example:

```
1000 SET_MAGNITUDE(A%,OFFFFH)
```

### SET-WAIT ON

(s) SET and WAIT ON allow synchronization between tasks based on the setting of an event. WAIT ON suspends a task until an event occurs in another task. SET makes the suspended task eligible to run. The software EVENT NAME statement must be used in both tasks to define the event used to synchronize the tasks.

```
SET event_name
WAIT ON event_name
```

event_name = name of event; must be defined previously in each task using software EVENT NAME; the variable must be a common

Example:

1st Task

```
1000 EVENT NAME = GAIN_OVER
...
...
2000 IF GAIN > MAX_GAIN THEN SET GAIN_OVER
```

2nd Task

```
1000 EVENT NAME = GAIN_OVER
...
...
2000 WAIT ON GAIN_OVER
```

### SHIFTL%

(f) Returns an integer value equal to the integer expression that was input shifted the specified number of the binary places to the left. Binary places vacated by the shift are filled with 0s.

SHIFTL%(variable,count)

variable = single or double variable or expression
count = number of bit positions to shift the integer or integer expression (shift begins at bit 15 for single integer and 31 for double integer)

Example:

```
1000 LSBITS_0_4% = SHIFTL%(INPUT_CARD%,12)
```

### SHIFTR%

(f) Returns an integer value equal to the integer expression that was input shifted the specified number of binary places to the right.

SHIFTR%(variable,count)

variable = single or double variable or expression
count = number of bit positions to shift the integer or integer expression (shift begins at bit 0)

Example:

```
1000 SHIFT1% = SHIFTR%(INPUT_CARD,4)
```

### SIN

(f) Returns the sine (in real format) of the input.

SIN(expression)

expression = numeric (integer or real) representing radians

Example:

```
1000 RADIANS = SIN(ANGLE)
```

### SQRT

(f) Returns a real value equal to the square root of the input and the same data type as the input.

SQRT(expression)

expression = integer or real variable or expression

Example:

```
1000 SQUARE_ROOT = SQRT(INPUT1)
```

### STR$

(f) Returns a string of characters from a numeric expression.

STRS$(expression)

expression = integer or real expression

Example:

```
STRING$ = STRS$(NUM1%*3)
```

### START EVERY

(s) Causes periodic re-start of the task.

START EVERY n units

n = any arithmetic expression or constant that evaluates to an integer result
units = units of time to delay re-start; units are SECONDS, MINUTES, HOURS and TICKS (1 TICK = 5.5 milliseconds); the plural form of the time unit must always be used, i.e., 1 TICKS, not 1 TICK

Example:

```
1000 START EVERY 20 TICKS
```

### STOP

(s) Stops program execution. Clears all I/O (sets to 0, FALSE or OFF) in the local and remote racks.

STOP

Example:

```
1000 STOP
```

### TAN

(f) Returns the tangent (in real format) of the input.

TAN(expression)

expression = numeric (integer or real) expression representing radians

Example:

```
1000 RADIANS = TAN(ANGLE)
```

### TASK

(c) Defines for each application task the task name, programming language, priority, Processor location, and whether the task is critical to the operation of the system.

TASK name[TYPE=language,PRIORITY=m,SLOT=n,CRITICAL=status]

name = name of the task, limited to 8 characters; first character must be a letter
language = programming language the task is written in; either BASIC, PC or CONTROL
m = priority of task execution on a scale of 4 (highest) to 11 (lowest)
n = slot number of the Processor on which the task will run; range 0 – 4
status = specifies whether the task is critical to the system, i.e., whether it can be stopped independently (FALSE) or only via a STOP ALL command from the executive software (TRUE)

Example:

```
1000 TASK SPD_REG[TYPE=CONTROL,PRIORITY=5,SLOT=2,          &
                  CRITICAL=FALSE]
```

### THEN

(s) See IF-THEN.

### VAL%

(f) Returns the integer value of a string in an integer format.

VAL%(string)

string = string variable or expression

Example:

```
1000 NUMBER% = VAL%(FIRST_WORD$)
```

### TST_ERRLOG@

(f) Tests the state of the error log and loads the number of errors into the specified variable.

TST_ERRLOG@(variable%)

variable = integer variable that will be loaded with the number of logged errors (1 to 3)

Example (in IF-THEN-ELSE statement):

```
1000 IF TST_ERRLOG@(NUM_ERRORS%) THEN
        2000
     ELSE
        3000
     END_IF
```

### VAL

(f) Returns the real value of a string in a real format.

VAL(string)

string = string variable or expression

Example:

```
1000 NUMBER = VAL(FIRST_WORD$)
```

### VARPTR!

(f) Returns the address of the variable which is the first register on the Modbus Interface module to which or from which data is to be transferred. Used with the GATEWAY_CMD_OK@ function.

VARPTR!(variable)

variable = common variable representing the physical address of the first register on the Modbus Interface module to which or from which data is to be transferred

Example:

```
1000 MSTR_REG! = VARPTR!(MASTER)
```

### WRITEVAR%

(f) Writes a value to a variable entered as a string. Requires the Ethernet™ operating system in the rack.

WRITEVAR%(vn$,value)

where

vn$ = a string expression for the name of the variable to write to. It can be a boolean, integer, double integer, real, or string, or an array of these types. Only one-dimensional arrays are allowed.

value = the variable that has the value to write; cannot be a literal.

Example:

```
STATUS% = WRITEVAR%(VARIABLE_NAME$, VALUE)
```

### WAIT ON

(s) See SET-WAIT ON.

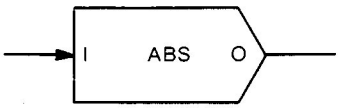
# Control Block Language Quick Reference

Note that all text must be entered in uppercase.

### ABSOLUTE_VALUE

Function

OUTPUT = | INPUT |




```
CALL ABSOLUTE_VALUE(INPUT=input%,                                    &
  OUTPUT=output%)
```

### ALARM

Function

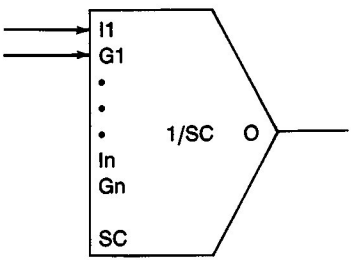
If INPUT is equal to or exceeds any of the alarm limits, the proper ALARM output(s) are set TRUE.


```
CALL ALARM(INPUT=input%,                                             &
  HIGH_LIMIT=high_limit%,                                            &
  LOW_LIMIT=low_limit%,                                              &
  HIGH_HIGH_LIMIT=high_high_limit%,                                  &
  LOW_LOW_LIMIT=low_low_limit%,                                      &
  ALARM_HIGH=alarm_high@,                                            &
  ALARM_LOW=alarm_low@,                                              &
  ALARM_HIGH_HIGH=alarm_high_high@,                                  &
  ALARM_LOW_LOW=alarm_low_low@)
```

### AMPLIFIER

Function

```
OUTPUT = (INPUT1*GAIN1+. . .INPUTn*GAINn)/                           &
    SCALE
```

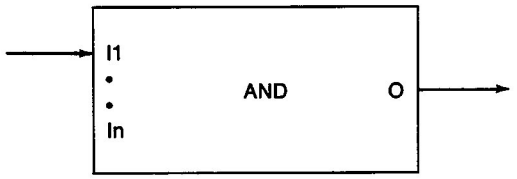



```
CALL AMPLIFIER(INPUT1=input1%,GAIN1=gain1%,. . .,                    &
  INPUTn=inputn%,GAINn=gainn%,SCALE=scale%,                          &
  OUTPUT=output%)
```

**AND**

Function

OUTPUT = INPUT1 and . . . INPUTn




Maximum "n" = 8

```
CALL AND(INPUT1=input1@,                                    &
  INPUTn=inputn@,                                           &
  OUTPUT=output@)
```

**BIT SELECT**

Function

If (INPUT−OFFSET) ≥0 and
(INPUT−OFFSET) ≤15
then OUTPUT(INPUT−OFFSET) is set TRUE

All other outputs(n) are set FALSE

else

All outputs(n) are set FALSE


```
CALL BIT_SELECT(INPUT=input%,OFFSET=offset%,                &
  OUTPUT0=output0@, . . .OUTPUTn=outputn@)
```

**COMPARE**

Function

Compares INPUT1 against INPUT2

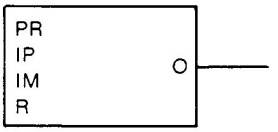
OUTPUT_GTR is set TRUE when INPUT1 is greater than INPUT2
OUTPUT_EQU is set TRUE when INPUT1 is equal to INPUT2
OUTPUT_LES is set TRUE when INPUT1 is less than INPUT2


```
CALL COMPARE(INPUT1=input1%,INPUT2=input2%,                 &
  OUTPUT_GTR=greater_than@,                                 &
  OUTPUT_EQU=equal@,                                        &
  OUTPUT_LES=less_than@)
```

## COUNTER

Function

OUTPUT = OUTPUT(n−1) + INPUT(+) − INPUT(−)

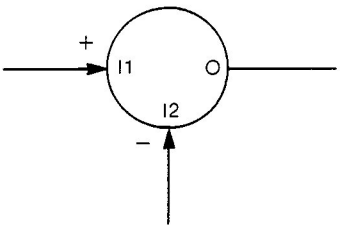



```
CALL COUNTER(RESET=reset@,PRESET=preset%),                    &
  INPUT_PLUS=input_plus%,                                     &
  INPUT_MINUS=input_minus%,                                   &
  OUTPUT=output%)
```

## DIFFERENCE

Function

OUTPUT = INPUT1 − INPUT2

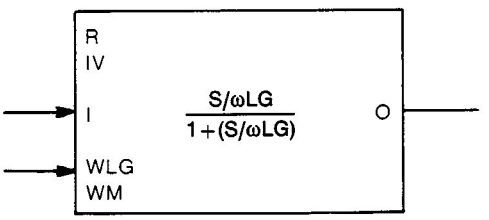



```
CALL DIFFERENCE(INPUT1=input1%,INPUT2=input2%,                &
  OUTPUT=output%)
```

## DIFF_LAG

Function

$$\text{LAPLACE TRANSFER FUNCTION} = \frac{s/\omega lg}{1+(s/\omega lg)}$$




```
CALL DIFF_LAG(INPUT=input%,ωlg=ωlg,WM=nnn.n,                  &
  INITIAL_VALUE=initial_value%,RESET=reset@,                  &
  OUTPUT=output%)
```

## FUNCTION BLOCK

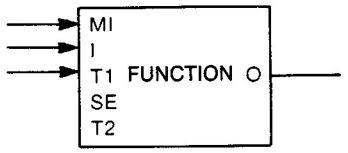
Function

$$\text{subscript.remainder} = \frac{\text{INPUT} * (\text{table\_size} - 1)}{(\text{max\_input} + 1)}$$

$$\text{OUTPUT} = \text{tN\%(subs)} + [\{\text{Tn\%(subs} + 1) - \text{Tn\%(subs)}\} * \text{REM}$$

where:
Tn = Table n
Subs = Subscript of above equation
REM = Remainder of above equation

(Note: OUTPUT is the result of linear interpolation between two points in the table.)

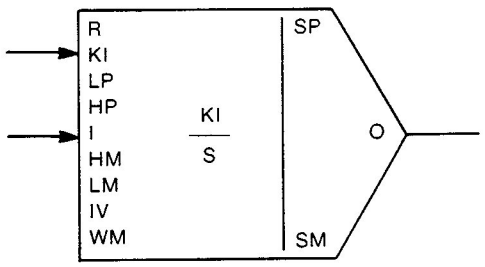



```
CALL FUNCTION(INPUT=input%,MAX_INPUT=max_input,                    &
  SELECT=select@,TABLE1=table1%,                                   &
  TABLE2=table2%,OUTPUT=output%)
```

## INTEGRATE

Function

$$\text{LAPLACE TRANSFER FUNCTION} = \frac{KI}{S}$$




```
CALL INTEGRATE(INPUT=input%,KI=ki,WM=nnn.n,                        &
  INITIAL_VALUE=initial_value%,                                    &
  LIMIT_PLUS=limit_plus%,                                          &
  LIMIT_MINUS=limit_minus%,RESET=reset@,                           &
  HOLD_PLUS=hold_plus@,                                            &
  HOLD_MINUS=hold_minus@,                                          &
  SATURATED_PLUS=saturated_plus@,                                  &
  SATURATED_MINUS=saturated_minus@,                                &
  OUTPUT=output%)
```

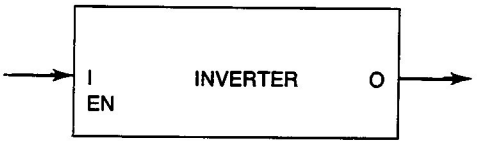
## INVERTER

Function

If ENABLE = TRUE then OUTPUT = −INPUT

else

OUTPUT = INPUT

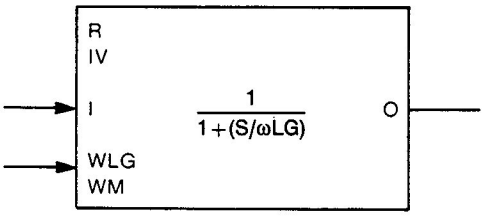



```
CALL INVERTER(ENABLE=enable@,INPUT=input%,                    &
  OUTPUT=output%)
```

**LAG**

Function

$$\text{LAPLACE TRANSFER FUNCTION} = \frac{1}{1+(s/\omega lg)}$$




```
CALL LAG(INPUT=input%,ωlg=ωlg,WM=nnn.n,                       &
  INITIAL_VALUE=initial_value%,                               &
  RESET=reset@,OUTPUT=output%)
```

**LATCH**

Function

If RESET is TRUE set OUTPUT FALSE

else

If RESET is FALSE and SET is TRUE set OUTPUT TRUE

else if

RESET and SET are FALSE and CLOCK is TRUE set OUTPUT to the state of INPUT

else

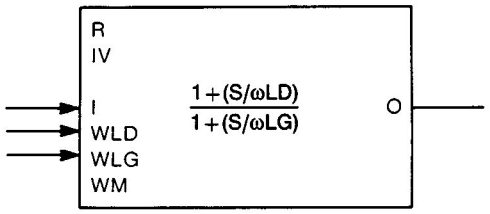
OUTPUT state is unchanged.


```
CALL LATCH(RESET=reset@,SET=set@,                             &
  CLOCK=clock@,INPUT=input@,                                  &
  OUTPUT=output@)
```

**LEAD_LAG**

Function

$$\text{LAPLACE TRANSFER FUNCTION} = \frac{1+(s/\omega ld)}{1+(s/\omega lg)}$$

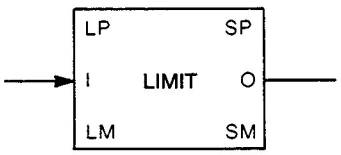



```
CALL LEAD_LAG(INPUT=input%,ωld=ωld,                    &
  ωlg=ωlg,WM=nnn.n,                                    &
  INITIAL_VALUE=initial_value%,                        &
  RESET=reset@,OUTPUT=output%)
```

**LIMIT**

Function

OUTPUT=INPUT within the range LIMIT(+) to LIMIT(−). If INPUT exceeds either limit, OUTPUT is held at that limit and the proper SATURATED output will be set.

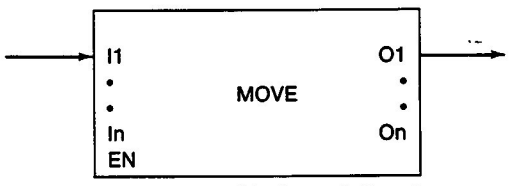



```
CALL LIMIT(INPUT=input%,LIMIT_PLUS=limit_plus%,        &
  LIMIT_MINUS=limit_minus%,                            &
  SATURATED_PLUS=saturated_plus@,                      &
  SATURATED_MINUS=saturated_minus@,                    &
  OUTPUT=output%)
```

**MOVE**

Function

If ENABLE=TRUE then
OUTPUT1=INPUT1
. .
. .
OUTPUTn=INPUTn



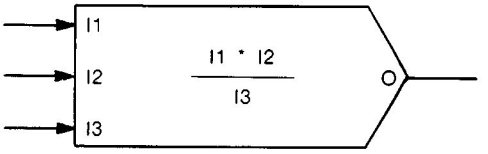

Maximum "n" = 8

```
CALL MOVE(ENABLE=enable@,                              &
  INPUT1=input1%,OUTPUT1=output1%, . . .               &
  INPUTn=inputn%,OUTPUTn=outputn%)
```

## MULTIPLY AND DIVIDE

Function

OUTPUT = INPUT * INPUT2/INPUT3




```
CALL MULTIPLY_DIVIDE(INPUT1 = input1%,                    &
  INPUT2 = input2%,INPUT3 = input3%,                      &
  OUTPUT = output%)
```

## NOTCH FILTER

Function

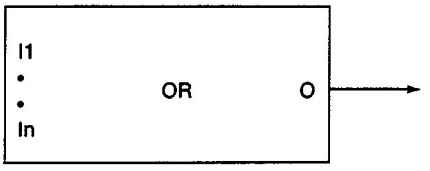
$$\text{LAPLACE TRANSFER FUCTION} = \frac{S^{**}2 + \omega n^{**}2}{S^{**}2 + \omega nS/Q + \omega n^{**}2}$$


```
CALL NOTCH(INPUT = input%,                                &
  Q_FACTOR = q_factor,                                    &
  WN — ωn,                                                &
  RESET = reset@,                                         &
  OUTPUT = output%)
```

## OR

Function

OUTPUT = INPUT1 or . . . INPUTn




Maximum "n" = 8

```
CALL OR(INPUT1 = input1@,                                 &
  INPUTn = inputn@,                                       &
  OUTPUT = output@)
```

## PACK BITS

Function

BITn in OUTPUT is set to the state of INPUTn. If INPUTn is not programmed, then BITn in OUTPUT is set FALSE.


Maximum "n" = 15

```
CALL PACK_BITS(INPUT0=input0@, . . . INPUTn=inputn@,                    &
  OUTPUT=output%)
```

**PID**

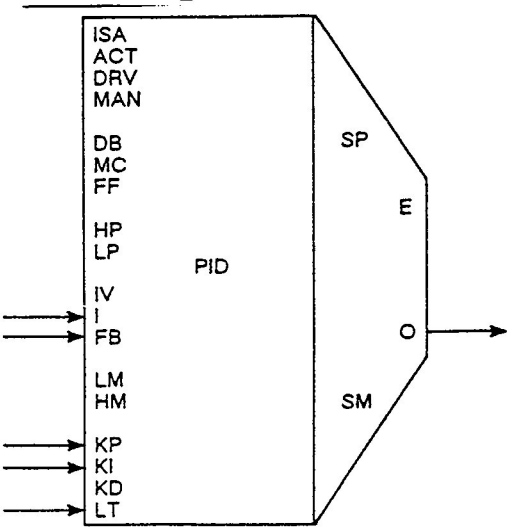
Function

If MANUAL is true then

```
ERROR=0
IOUT(n)=INITIAL_VALUE
If IOUT(n) > LIMIT_PLUS – FEED_FORWARD then
IOUT(n)=LIMIT_PLUS – FEED_FORWARD
SATURATED_PLUS=TRUE
If IOUT(n) < LIMIT_MINUS – FEED_FORWARD then
IOUT(n)=LIMIT_MINUS – FEED_FORWARD
SATURATED_MINUS=TRUE
OUTPUT=IOUT(n) + FEED_FORWARD
```

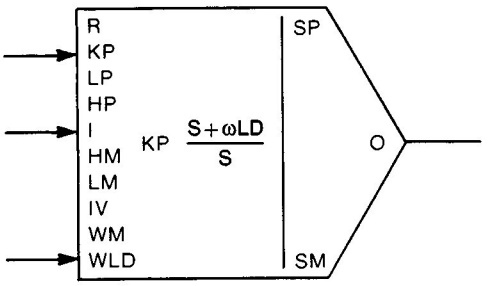
Else

```
ERROR=INPUT – FEEDBACK or FEEDBACK – INPUT
Incr(n)=Calculated delta change value for selected PID
algorithm (ISA or Independent)
If ABSERROR| < DEAD_BAND then incr(n)=0
If ABSIncr(n)| > MAX_CHANGE then
limit Incr(n) to MAX_CHANGE
IOUT(n)=Incr(n) + IOUT(n–1)
If HOLD_PLUS is TRUE and IOUT(n) > IOUT(n–1) then
IOUT(n)=IOUT(n–1)
If HOLD_MINUS is TRUE and IOUT(n) < IOUT(n–1) then
IOUT(n)=IOUT(n–1)
If IOUT(n) > LIMIT_PLUS – FEED_FORWARD then
IOUT(n)=LIMIT_PLUS – FEED_FORWARD
SATURATED_PLUS=TRUE
If IOUT(n) < LIMIT_MINUS – FEED_FORWARD then
IOUT(n)=LIMIT_MINUS – FEED_FORWARD
SATURATED_MINUS=TRUE
OUTPUT=IOUT(n) + FEED_FORWARD
```



```
CALL PID(ISA= ⟨ boolean literal ⟩,ACTION= ⟨ boolean literal ⟩,                &
  DERIVATIVE= ⟨ boolean literal ⟩,MANUAL=manual@,                            &
  KP=kp,KI=ki,KD=kd,LOOP_TIME=loop_time,                                    &
  INITIAL_VALUE=initial_value%,FEED_FORWARD=feed_forward%,                  &
  INPUT=input%,FEEDBACK=feedback,                                           &
  DEAD_BAND=dead_band%,MAX_CHANGE=max_change%,                              &
  LIMIT_PLUS=limit_plus%,LIMIT_MINUS=limit_minus%,                          &
  HOLD_PLUS=hold_plus@,HOLD_MINUS=hold_minus@,                              &
  SATURATED_PLUS=saturated_plus@,                                           &
  SATURATED_MINUS=saturated_minus@,                                         &
  ERROR=error%,OUTPUT=output%)
```

## PROP_INT




```
CALL PROP_INT(INPUT=input%,KP=kp,ωld=ωld,                                   &
  WM=nnn.n,INITIAL_VALUE=initial_value%,                                    &
  LIMIT_PLUS=limit_plus%,                                                   &
  LIMIT_MINUS=limit_minus%,RESET=reset@,                                    &
  HOLD_PLUS=hold_plus@,                                                     &
  HOLD_MINUS=hold_minus@,                                                   &
  SATURATED_PLUS=saturated_plus@,                                           &
  SATURATED_MINUS=saturated_minus@,                                         &
  OUTPUT=output%)
```

## PULSE_MULTIPLIER

Function

If WORD_SIZE > 0 (Relative mode) then

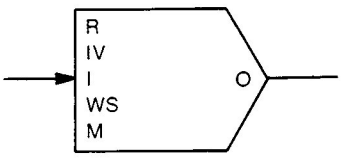
If falling edge of RESET then
  INPUT(n−1)=INITIAL_VALUE
ERROR=INPUT − INPUT(n−1)

else

ERROR=INPUT

$$OUTPUT = \left(\frac{ERROR * MULTIPLIER}{32767}\right) +$$

OUTPUT = REM(n−1)

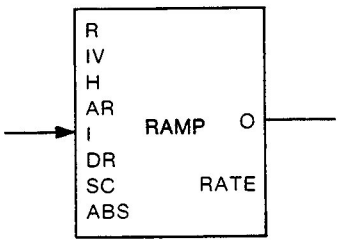



```
CALL PULSE_MULT(INPUT=input%,RESET=reset@,                                  &
  WORD_SIZE=nnn,INITIAL_VALUE=initial_value%,                               &
  MULTIPLIER=multiplier%,OUTPUT=output%)
```

**RAMP**

Function

For a change in INPUT, OUTPUT will ramp toward the new INPUT value. During steady state operation, OUTPUT=INPUT. Two types of RAMP generators are provided: a normal (algebraic) ramp and an absolute value ramp. For a normal ramp, the accel condition is defined by an input that is becoming more positive, and a decel condition is defined by an input that is becoming more negative. For an absolute value ramp, the accel condition is defined by an input moving away from zero, and a decel condition is defined by an input moving towards or through zero.

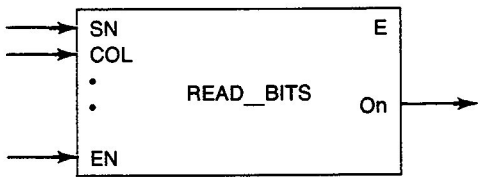



```
CALL RAMP(INPUT=input%,ABS_RAMP=TRUE/FALSE,                    &
  RESET=reset@,INITIAL_VALUE=initial_value%,                   &
  HOLD=hold@,ACCEL_RATE=accel_rate%,                           &
  DECEL_RATE=decel_rate%,SCALE=nnnnn,                          &
  OUTPUT=output%,RATE=rate%)
```

**READ BITS**

Function

This function reads data from a column in the specified BOOLEAN data structure.

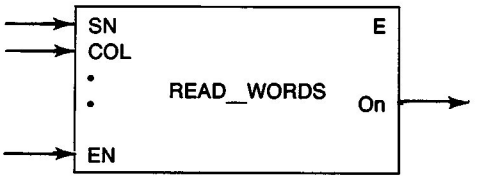



```
CALL READ_BITS(STRUCTURE_NAME=structure_name@,                 &
  COLUMN=column%,ENABLE=enable@,                               &
  ERROR=error@,                                                &
  OUTPUT1=output1%, . . . OUTPUTn=outputn%)
```

**READ WORDS**

Function

This function reads data from a column in the specified INTEGER data structure.



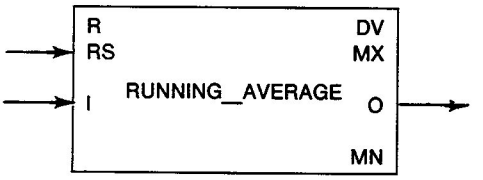

```
CALL READ_WORDS(STRUCTURE_NAME=structure_name%,                    &
  COLUMN=column%,ENABLE=enable@,                                   &
  ERROR=error@,                                                    &
  OUTPUT1=output1%, . . . OUTPUTn=outputn%)
```

## RUNNING AVERAGE

Function

$$OUTPUT = \frac{[INPUT(n) + \ldots INPUT(n+1)]}{(REQUIRED\_SAMPLES)}$$

The OUTPUT is updated each scan with the average of the samples read for INPUT over the last RS scans.



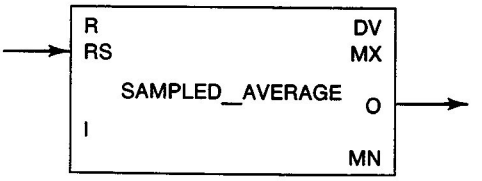

```
CALL RUNNING_AVERAGE(REQUIRED_SAMPLES=req_sam,                     &
  RESET=reset@,                                                    &
  INPUT=input%,                                                    &
  DATA_VALID=data_valid@,                                          &
  MAX_VALUE=max_value%,MIN_VALUE=min_value%,                       &
  OUTPUT=output%)
```

## SAMPLED AVERAGE

Function

$$OUTPUT = \frac{[INPUT(1) + \ldots INPUT(RS)]}{(REQUIRED\_SAMPLES)}$$

The OUTPUT is updated once every RS scans with the average of the samples read for INPUT during the last RS scans.



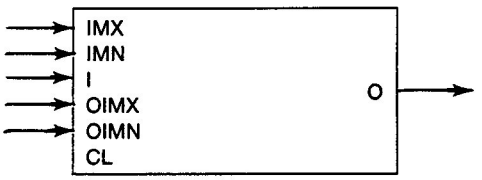

```
CALL SAMPLED_AVERAGE(REQUIRED_SAMPLES=req_sam,                     &
  INPUT=input%,                                                    &
  RESET=reset@,                                                    &
  DATA_VALID=data_valid@,                                          &
  MAX_VALUE=max_value%,MIN_VALUE=min_value%,                       &
  OUTPUT=output%)
```

## SCALE

Function

If CLAMP is TRUE and INPUT exceeds INPUT_MAX or INPUT_MIN, then the value of INPUT is clamped at the proper limit.

$$OUTPUT = \frac{(INPUT - INPUT\_MIN) * (OUTPUT\_MAX - OUTPUT\_IMIN)}{INPUT\_MAX - INPUT\_MIN} + OUTPUT\_IMIN$$




```
CALL SCALE(INPUT=input%,CLAMP=clamp@,                    &
  INPUT_MAX=input_max%,                                  &
  INPUT_MIN=input_min%,                                  &
  OUTPUT_IMAX=output_imax%,                              &
  OUTPUT_IMIN=output_imin%,                              &
  OUTPUT=output%)
```

## SCAN_LOOP

Time scanned loop (no event)

```
CALL SCAN_LOOP(TICKS=ticks%)
```

Or hardware event

```
EVENT NAME=START_TASK,                                   &
  INTERRUPT_STATUS=ISCR%,TIMEOUT=6
CALL SCAN_LOOP(TICKS=4,                                  &
  EVENT=START_TASK)
```

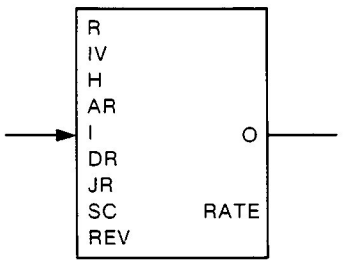
Or software event

```
EVENT NAME=BEGIN
CALL SCAN_LOOP(TICKS=4,EVENT=BEGIN)
```

## S_CURVE

Function

The S_CURVE block performs the same basic function as the RAMP block with a jerk rate added. If the REVERSE bit=TRUE, then the input ACCEL rate will become the DECEL rate and the input DECEL rate will become the ACCEL rate. This is used to provide the function similar with the "motor" type RAMP block (ABS_RAMP=TRUE). With the S_CURVE block, however, this function can be dynamically controlled by the application.



```
CALL S_CURVE(INPUT=input%,RESET=reset@,                    &
  HOLD=hold@,ACCEL_RATE=accel_rate%,                       &
  DECEL_RATE=decel_rate%,                                  &
  JERK_RATE=jerk_rate%,                                    &
  SCALE=nnnnn,REVERSE=reverse@,                            &
  INITIAL_VALUE=initial_value%,                            &
  OUTPUT=output%,RATE=rate%)
```

## SEARCH

Function

Compare INPUT against selected TABLE elements.

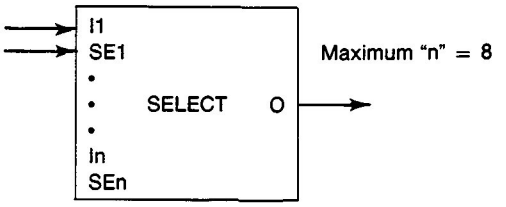
Search the selected table for a match according to the comparison options selected by the three BOOLEAN inputs COMPARE_GTR, COMPARE_EQU and COMPARE_LES. The search starts at the top of the table (array element 0) and tests INPUT against each element in the table until either a match is found or the end of the table is reached (last element in the array). If a match occurs, the search function is terminated and the index into the table where the match occurred is written to OUTPUT. FOUND is then set true. If no match occurred, the OUTPUT is set to a value of −1 and FOUND is set FALSE.


```
CALL SEARCH(COMPARE_GTR=〈boolean literal〉,                &
  COMPARE_EQU=〈boolean literal〉,                           &
  COMPARE_LES=〈boolean literal〉,                           &
  INPUT=input%,                                            &
  SELECT=select@,                                          &
  TABLE1=table1%,TABLE2=table2%,                           &
  FOUND=found@,OUTPUT=output%)
```

## SELECT

Function

OUTPUT will equal the sum of all selected inputs. If no SELECT is true, OUTPUT=0.

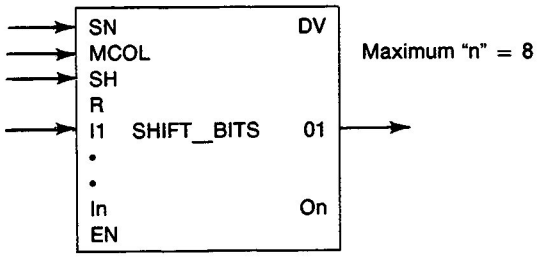



```
CALL SELECT(INPUT1=input1%,SELECT1=select1@,               &
  . . ., INPUTn=inputn%,SELECTn=selectn@,                  &
  OUTPUT=output%)
```

**SHIFT BITS**

Function

When RESET is FALSE and SHIFT is TRUE, shift the data in the specified BOOLEAN data structure towards the output(s), update the output(s) with the state(s) at column MCOL – 1 and if ENABLE is TRUE, shift the state(s) of the input(s) into column 0, else set the state(s) of column 0 FALSE.

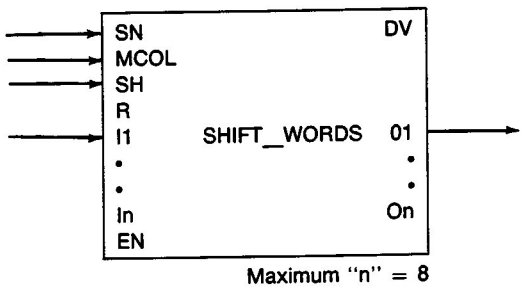



```
CALL SHIFT_BITS(STRUCTURE_NAME=struc_name@,                    &
  MAX_COLUMNS=max_columns%,                                    &
  RESET=reset@,SHIFT=shift@,                                   &
  INPUT1=input1@, . . . INPUTn=inputn@,                        &
  ENABLE=enable@,DATA_VALID=data_valid@,                       &
  OUTPUT1=output1@, . . . OUTPUTn=outputn@)
```

**SHIFT WORDS**

Function

When RESET is FALSE and SHIFT is TRUE, shift the data in the specified INTEGER data structure towards the output(s), update the output(s) with the value(s) at column MCOL – 1, and if ENABLE is TRUE, shift the data at the input(s) into column 0, else shift zeros into column 0.

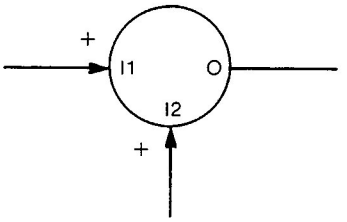



```
CALL SHIFT_WORD(STRUCTURE_NAME=struc_name%,                    &
  MAX_COLUMNS=max_columns%,                                    &
  RESET=reset@,SHIFT=shift@,                                   &
  INPUT1=input1%, . . . INPUTn=inputn%,                        &
  ENABLE=enable@,DATA_VALID=data_valid@,                       &
  OUTPUT1=output1%, . . . OUTPUTn=outputn%)
```

## SUMMER

Function

OUTPUT = INPUT1 + INPUT2




```
CALL SUMMER(INPUT1 = input1%,INPUT2 = input2%,                    &
  OUTPUT = output%)
```

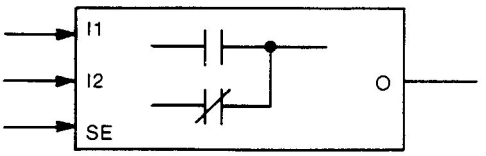
## SWITCH

Function

If SELECT = TRUE then
OUTPUT = INTPUT1

else

OUTPUT = INPUT2

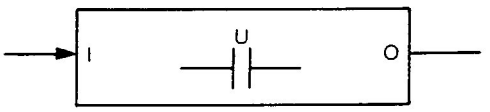



```
CALL SWITCH(INPUT1 = input1%,INPUT2 = input2%,                    &
  SELECT = select@,OUTPUT = output%)
```

## TRANSITION

Function

OUTPUT = TRUE when INPUT goes from off to on

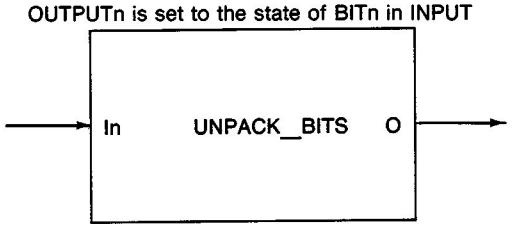



```
CALL TRANSITION(INPUT = input@,                                   &
  OUTPUT = output@)
```

### UNPACK BITS

Function

OUTPUTn is set to the state of BITn in INPUT

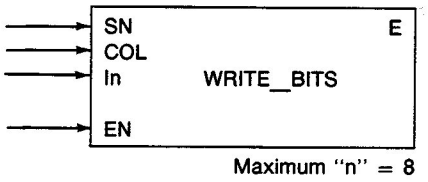



```
CALL UNPACK_BITS(INPUT=input%,                                    &
  OUTPUT0=output0@, . . . OUTPUTn=outputn@)
```

### WRITE BITS

Function

This function stores data into a column in the specified BOOLEAN data structure.

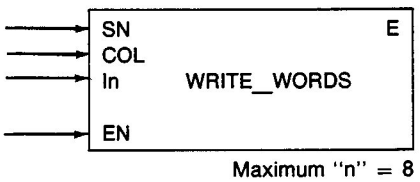



```
CALL WRITE_BITS(STRUCTURE_NAME=structure_name@,                   &
  COLUMN=column%,ENABLE=enable@,                                  &
  INPUT1=input1@, . . . INPUTn=inputn@,                           &
  ERROR=error@)
```

### WRITE WORDS

Function

This function stores data into a column in the specified INTEGER data structure.



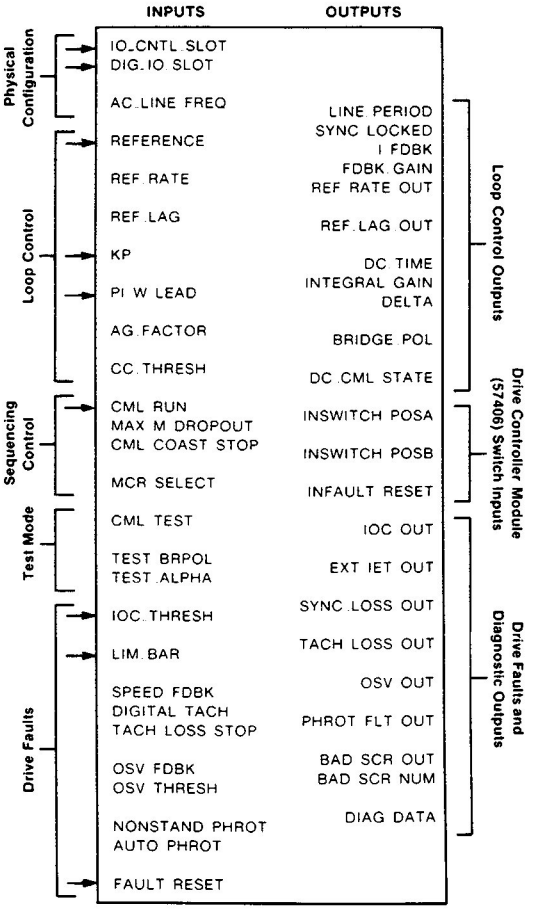

```
CALL WRITE_WORD(STRUCTURE_NAME=struc_name%,                       &
  COLUMN=column%,ENABLE=enable@,                                  &
  INPUT1=input1%, . . . INPUTn=inputn%,                           &
  ERROR=error@)
```

# Current Minor Loop

## DC_DRIVE_CML

Function

Performs the current minor loop regulation for the S6R DC Motor drive.



```
CALL DC_DRIVE_CML(IO_CNTL_SLOT=io_cntl_slot%,                    &
  DIG_IO_SLOT=dig_io_slot%,                                      &
  AC_LINE_FREQ=ac_line_freq%,                                    &
  *REFERENCE=reference%,*REF_RATE=ref_rate,                      &
  REF_LAG=ref_lag,*KP=kp,PI_W_LEAD=pi_w_lead,                    &
  AG_FACTOR=ag_factor%,                                          &
  *CC_THRESH=cc_thresh%,                                         &
  *CML_RUN=cml_run@,                                             &
  *CML_COAST_STOP=cml_coast_stop@,                               &
  *MCR_SELECT=mcr_select@,                                       &
  MAX_M_DROPOUT=max_m_dropout,                                   &
  *CML_TEST=cml_test@,                                           &
  *TEST_BRPOL=test_brpol@,                                       &
  *TEST_ALPHA=test_alpha%,                                       &
  IOC_THRESH=ioc_thresh%,                                        &
  LIM_BAR=lim_bar%,                                              &
  DIGITAL_TACH=digital_tach@,                                    &
  *SPEED_FDBK=speed_fdbk%,                                       &
  TACH_LOSS_STOP=tach_loss_stop@,                                &
  *OSV_FDBK=osv_fdbk%,                                           &
  OSV_THRESH=osv_thresh%,                                        &
  *NONSTAND_PHROT=nonstand_phrot@,                               &
  AUTO_PHROT=auto_phrot@,                                        &
  *FAULT_RESET=fault_reset%,                                     &
  DC_CML_STATE=dc_cml_state@,                                    &
  SYNC_LOCKED=sync_locked@,                                      &
  LINE_PERIOD=line_period%,                                      &
  INSWITCH_POSA=inswitch_posa@,                                  &
  INSWITCH_POSB=inswitch_posb@,                                  &
  INFAULT_RESET=infault_reset@,                                  &
  **DC_TIME=dc_time%,                                            &
  **BRIDGE_POL=bridge_pol@,                                      &
  **REF_RATE_OUT=ref_rate_out%,                                  &
  **REF_LAG_OUT=ref_lag_out%,                                    &
  **FDBK_GAIN=fdbk_gain%,**I_FDBK=i_fdbk%,                       &
  **INTEGRAL_GAIN=integral_gain%,                                &
  **DELTA=delta%,IOC_OUT=ioc_out@,                               &
  SYNC_LOSS_OUT=sync_loss_out@,                                  &
  TACH_LOSS_OUT=tach_loss_out@,                                  &
  OSV_OUT=osv_out@,                                              &
  EXT_IET_OUT=ext_iet_out@,                                      &
  PHROT_FLT_OUT=phrot_flt_out@,                                  &
  BAD_SCR_OUT=bad_scr_out@,                                      &
  BAD_SCR_NUM=bad_scr_num%,                                      &
  DIAG_DATA=diag_data%)
```

* Dynamically adjustable

** Latched on error

Note that the asterisks are not part of the CML statement.

# Control Block Execution Time Estimates

The execution time for a Control Block task can be determined by adding the execution time for each statement to the SCAN_LOOP/END time. Dividing by the scan time (TICKS × tick rate in seconds) gives the estimated CPU usage for the task.

| Block Name | 7010 Processor Maximum Time (μsec) | 6010/6011 Processor Maximum Time (μsec) |
|---|---|---|
| ABSOLUTE_VALUE | 9.16 | 41 |
| ALARM | 11.16 + n(2.48) | 44 + n(11) |
| AMPLIFIER | 13.16 + k(3.12) | 62 + k(13.8) |
| AND | 10.80 + j(1.88) | 41 + j(8.3) |
| BIT_SELECT | 26.04 + n(1.20) | 118 + n(6.5) |
| COMPARE | 11.24 + n(1.16) | 45 + n(5.5) |
| COUNTER | 15.64 | 61 |
| DIFFERENCE | 10.80 | 44 |
| DIFF_LAG | 33.04 | 154 |
| END | 13.82 + t(2.0) + i(1.6) + b(2.2) + d(2.5) | 60 + t(3.2) + i(2.8) + b(4.3) + d(4.9) |
| FUNCTION | 23.56 | 93 |
| INTEGRATE | 39.14 | 193 |
| INVERTER | 10.84 | 48 |
| LAG | 26.52 | 118 |
| LATCH | 15.04 | 68 |
| LEAD_LAG | 68.60 | 284 |
| LIMIT | 18.20 | 69 |
| MOVE | 10.40 + p(2.32) | 39 + p(11) |
| MULTIPLY_DIVIDE | 13.76 | 71 |
| NOTCH | 85.16 | 393 |
| OR | 10.24 + j(1.88) | 41 + j(8.3) |
| PACK_BITS | 24.28 + j(1.44) | 110 + j(5) |
| PID | 124.00 | 515 |
| PROP_INT | 40.32 | 193 |
| PULSE_MULT | 18.32 | 86 |
| RAMP | 27.72 | 122 |
| READ_BITS | 24.32 + n(1.12) | 106 + n(7.3) |
| READ_WORDS | 25.76 + n(1.28) | 111 + n(5.6) |
| RUNNING_AVERAGE | 28.64 | 125 |
| SAMPLED_AVERAGE | 29.00 | 125 |
| S_CURVE | 47.00 | 282 |
| SCALE | 22.48 | 98 |
| SCAN_LOOP | 80.00 + t(1.6) + i(2.5) + b(3.8) + d(4.1) | 326 + t(2.4) + i(4.4) + b(6.8) + d(7.2) |
| SEARCH | 19.80 + m(0.68) | 74 + m(3) |
| SELECT | 11.36 + j(2.52) | 44 + j(12.2) |
| SHIFT_BITS | 18.28 + p(2.04) | 92 + p(19.7) |
| SHIFT_WORDS | 17.80 + p(1.72) | 89 + p(13.3) |
| SUMMER | 11.28 | 44 |
| SWITCH | 11.52 | 47 |
| TRANSITION | 11.36 | 48 |
| UNPACK_BITS | 23.40 + n(1.44) | 106 + n(7.3) |
| WRITE_BITS | 25.56 + j(1.36) | 103 + j(6) |
| WRITE_WORDS | 25.52 + j(0.36) | 110 + j(1.9) |
| !<REM> | 9.88 | 40 |

b = number of common boolean variables referenced by the task
d = number of common double integer variables referenced by the task
i = number of common integer variables referenced by the task
j = total number of inputs programmed
k = total number of input pairs programmed
m = number of elements searched
n = total number of outputs programmed
p = total number of input/output pairs programmed
t = total number of common integer, boolean, and double integer variables referenced by the task

# AutoMax Off-Line PC Editor Quick Reference

F1 — Help

F2 — Initiate search for variable name/contact type
ALT-F2 — Next occurrence search variable

F3 — Commands
- E - Exit and update file
- Q - Quit
- R - Resequence
- S - Substitute
- P - Preset modify
- A - Add element description
- C - Change element description
- I - Program information
- M - Move sequence
- D - Delete multiple sequences
- N - Include sequences from file
- W - Wildcard substitution
- T - Text edit descriptions
- V - Variable scope change

F4 — Normally open contact
ALT-F4 — Up transition contact

F5 — Normally closed contact
ALT-F5 — Down transition contact

F6 — Horizontal line
ALT-F6 — Delete element

F7 — Vertical branch
ALT-F7 — Delete branch

F8 — Coil — ( )

F9 — Functions
- T - Timer (On-delay)
- O - Off-delay timer
- C - Counter
- S - Shift register
- R - Remarks
- E - Event coil

F10 — Find sequence number/coil name

Home — First sequence
End — Last sequence
PgUp — Previous sequence
PgDn — Next sequence
Del — Delete sequence
Ins — Insert sequence (after)

# PC/Ladder Language Format

**Normally Open Contact**

bitname

**Normally Closed Contact**

bitname

**Up Transition Contact**

bitname

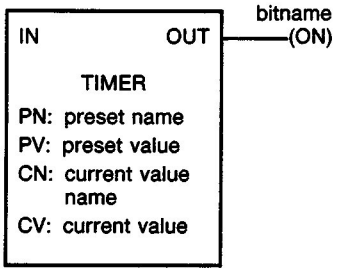
**Down Transition Contact**

bitname

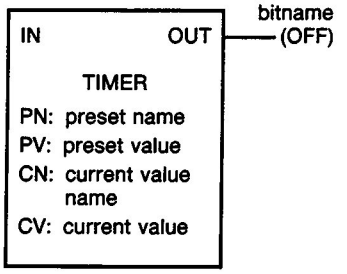
**Coil**

bitname

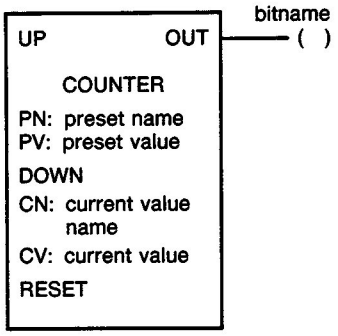
**On Delay Timer**



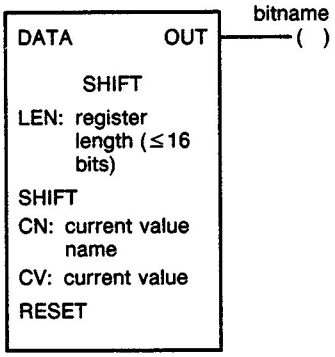

**Off Delay Timer**



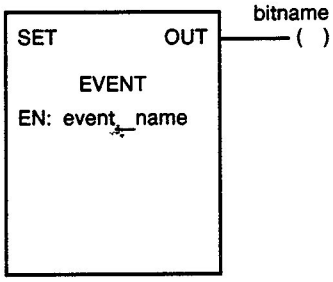
**Counter**




**Shift Register**




**Event Coil**



# PC/Ladder Logic Execution Time and Memory Usage Estimates

The execution time for a ladder logic task can be estimated using the following timing estimates. After the total execution time is computed, divide by the scan time to obtain an estimate of the CPU usage for the task. A maximum of 2047 different symbols may be used in a single task. Ladder logic tasks are limited to a maximum of 90K bytes for ladder sequences plus 45K for symbols.

| Ladder Logic Operation | 7010 Execution Time In µsec | 6010 Execution Time In µsec | Memory Usage in bytes |
|---|---|---|---|
| Normally Open Contact | .5 | 1.5 | 6 |
| Normally Closed Contact | .5 | 1.5 | 6 |
| Transition Contacts | | | |
| AND | 1.0 | 3.0 | 10 |
| OR | 1.2 | 3.5 | 12 |
| Coils | 7.1 | 26.0 | 8 |
| Timers | 14.7 | 50.0 | 24 |
| Counters | 14.0 | 50.0 | 24 |
| Shift Registers | 11.9 | 41.0 | 24 |
| Events | | | |
| Event Set | 152.0 | 272.0 | |
| Event Not Set | 9.6 | 18.0 | 24 |
| Remark | 3.5 | 11.0 | 24 |
| Variables | | | 8+ # of characters in name |
| System Overhead - Fixed | 100 | 314 | 3000 |
| System Overhead - Variable | 6c+3f | 18c+9f | c/n *10 + 10/sequence |

Key:

c = number of common variables
f = number of forced variables
n = packing density, a function of how the common booleans are stored in memory. The value can range from 1 (worst case) to 16 (ideal case). Use a value of 8 as an estimate.

# Norton™ Editor Command Summary

**F1 — Help**

**F2 — Status**

**F3 — File commands**

E — *Exit,* save the data and end the edit session
Q — *Quit,* discard the data and end the edit session
S — *Save,* write the data to disk and continue editing
X — *eXchange* windows, switch to other window
N — *New,* edit a new file
A — *Append* another file to end of edit data
W — *Write* part of the data to disk
L — *Load* more data from a large file
C — *Close* the output file, open new output

**F4 — Block commands**

S — *Set* a block marker
R — *Remove* block markers
D — *Delete* a block
C — *Copy* a block
W — *Window-to-window* block copy
M — *Move* a block
L — *Line,* mark an entire line as a block
E — *End-of-line,* mark to end-of-line as a block
F — *Find* block marker

**F5 — Screen format commands**

L — *Line length,* set line length
W — *Word-wrap,* toggle on and off
F — *Format* a paragraph
T — *Tab,* set tab spacing
I — *Indent,* toggle auto indent on and off
C — *Cursor,* set cursor type
D — *Display,* set display color
S — *Save* editor with new defaults set
K — *Key* define, change operation of *Tab* and *Ins* keys

**F6 — Miscellaneous commands**

G — *Goto* a line number
M — *Matching* punctuation, finds matching symbol
C — *Condensed* display mode
*Ins* — *Insert* mode cancel, switch to *replace* mode
T — *Test* windows for differences

**F7 — Printer commands**

P — *Print* entire edit buffer
B — *Block,* print marked block
E — *Eject* paper (form-feed)
S — *Size* in lines per page
M — *Margin,* set left margin for printing

**F9 — DOS Command Processor**

**Other commands:**

*Ins* — *Insert,* switch to insert mode
*^U* — *Undelete*
*^F* — *Find* or *find-and-replace*
*^C* — *Continue* find operation
*^P* — *Insert* control character
*^V* — *Vice versa,* flip upper- and lowercase

**Cursor Movement**

| | |
|---|---|
| *Right Arrow* | right one character |
| *Left Arrow* | left one character |
| *Up Arrow* | up one line |
| *Down Arrow* | down one line |
| *^Right Arrow* | right one word |
| *^Left Arrow* | left one word |
| *Home* | beginning of line |
| *End* | end of line |
| *PgUp* | top of page *or* previous page |
| *PgDn* | bottom of page *or* next page |
| *^Home* or *^PgUp* | beginning of the edit data |
| *^End* or *^PgDn* | end of the edit data |

**Delete Commands**

| Key | Deletes |
|---|---|
| *Backspace* | one character to the left, when in Insert mode |
| *Del* | the character under cursor |
| *^W* | the word to the left of the cursor |
| *Alt-W* | the word to the right of the cursor |
| *^L* | from the cursor to beginning of line |
| *Alt-L* | from cursor to end of line |
| *Alt-K* | the entire current line |
| *F4* D | the marked block |
| *^U* or *Alt-U* | undeletes any but a deleted block (until the cursor is moved) |

**Search Commands**

| | |
|---|---|
| Alt F | Find string forward |
| ^F | Find string reverse |
| Alt C | Continue find string forward |
| ^C | Continue find string reverse |
| F4 F | Find block marker forward |

To make the search case-insensitive, terminate the string with ESC rather than RETURN.

To search for a new line character, enter control RETURN in the search string.

**Search and Replace Commands**

Forward search and replace:
  ALT F
  enter search string
  ALT F
  enter replacement string

Reverse search and replace:
  ^F
  enter search string
  ^F
  enter replacement string

Responses to search and replace:

| | |
|---|---|
| Y | — Replace |
| N | — Doesn't replace |
| * | — Replace all |
| SPACE | — Quits search and replace |
| | |
| ALT C | — Continue search forward |
| ^C | — Continue search reverse |

# Section IV
# Appendices

# Appendix A
# ASCII
# Conversion Chart

| Hex | Dec | * | Char | Hex | Dec | Char | Hex | Dec | Char |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 0 | @ | NUL | 2B | 43 | + | 55 | 85 | U |
| 01 | 1 | A | SOH | 2C | 44 | , | 56 | 86 | V |
| 02 | 2 | B | STX | 2D | 45 | - | 57 | 87 | W |
| 03 | 3 | C | ETX | 2E | 46 | . | 58 | 88 | X |
| 04 | 4 | D | EOT | 2F | 47 | / | 59 | 89 | Y |
| 05 | 5 | E | ENQ | 30 | 48 | 0 | 5A | 90 | Z |
| 06 | 6 | F | ACK | 31 | 49 | 1 | 5B | 91 | [ |
| 07 | 7 | G | BEL | 32 | 50 | 2 | 5C | 92 | \ |
| 08 | 8 | H | BS | 33 | 51 | 3 | 5D | 93 | ] |
| 09 | 9 | I | HT | 34 | 52 | 4 | 5E | 94 | ^ |
| 0A | 10 | J | LF | 35 | 53 | 5 | 5F | 95 | _ |
| 0B | 11 | K | VT | 36 | 54 | 6 | 60 | 96 | ` |
| 0C | 12 | L | FF | 37 | 55 | 7 | 61 | 97 | a |
| 0D | 13 | M | CR | 38 | 56 | 8 | 62 | 98 | b |
| 0E | 14 | N | SO | 39 | 57 | 9 | 63 | 99 | c |
| 0F | 15 | O | SI | 3A | 58 | : | 64 | 100 | d |
| 10 | 16 | P | DLE | 3B | 59 | ; | 65 | 101 | e |
| 11 | 17 | Q | DC1 | 3C | 60 | < | 66 | 102 | f |
| 12 | 18 | R | DC2 | 3D | 61 | = | 67 | 103 | g |
| 13 | 19 | S | DC3 | 3E | 62 | > | 68 | 104 | h |
| 14 | 20 | T | DC4 | 3F | 63 | ? | 69 | 105 | i |
| 15 | 21 | U | NAK | 40 | 64 | @ | 6A | 106 | j |
| 16 | 22 | V | SYN | 41 | 65 | A | 6B | 107 | k |
| 17 | 23 | W | ETB | 42 | 66 | B | 6C | 108 | l |
| 18 | 24 | X | CAN | 43 | 67 | C | 6D | 109 | m |
| 19 | 25 | Y | EM | 44 | 68 | D | 6E | 110 | n |
| 1A | 26 | Z | SUB | 45 | 69 | E | 6F | 111 | o |
| 1B | 27 | [ | ESC | 46 | 70 | F | 70 | 112 | p |
| 1C | 28 | \ | FS | 47 | 71 | G | 71 | 113 | q |
| 1D | 29 | ] | GS | 48 | 72 | H | 72 | 114 | r |
| 1E | 30 | ^ | RS | 49 | 73 | I | 73 | 115 | s |
| 1F | 31 | - | US | 4A | 74 | J | 74 | 116 | t |
| 20 | 32 | | SP | 4B | 75 | K | 75 | 117 | u |
| 21 | 33 | | ! | 4C | 76 | L | 76 | 118 | v |
| 22 | 34 | | " | 4D | 77 | M | 77 | 119 | w |
| 23 | 35 | | # | 4E | 78 | N | 78 | 120 | x |
| 24 | 36 | | $ | 4F | 79 | O | 79 | 121 | y |
| 25 | 37 | | % | 50 | 80 | P | 7A | 122 | z |
| 26 | 38 | | & | 51 | 81 | Q | 7B | 123 | { |
| 27 | 39 | | ' | 52 | 82 | R | 7C | 124 | \| |
| 28 | 40 | | ( | 53 | 83 | S | 7D | 125 | } |
| 29 | 41 | | ) | 54 | 84 | T | 7E | 126 | ~ |
| 2A | 42 | | * | | | | 7F | 127 | DEL |

* Press 〈CTRL〉 at the same time as the indicated character

# Appendix B
# Decoding Bus Errors

Some bus errors that occur in the rack can be easily traced using the procedure described below. These errors result in a hexadecimal entry in the "Error Specific" field in the programming executive software error log.

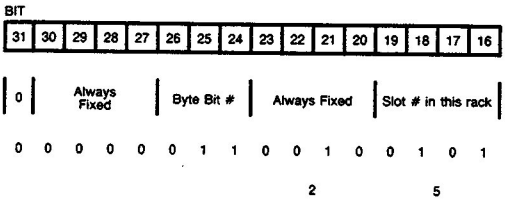
The procedure consists of converting the hexadecimal number given in the error log into a binary number, and then interpreting the most significant and least significant 16 bits as described below.

1. Convert the hexadecimal number found in the "Error Specific" field into a binary number. If the hexadecimal number consits of fewer than eight digits, pad the number with zeroes. For example, hexadecimal 3250184 would be padded with a zero first to become 03250184. The conversion to binary would look like this:

0 3 2 5 0 1 8 4

0000 0011 0010 0101 0000 0001 1000 0100

2. Interpret the resulting binary pattern in two stages. First, examine the most significant 16 bits using the pattern below. This step uses the sample value in step 1 above.

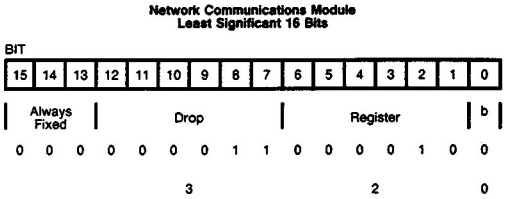



The first line under the figure above describes the meaning of the pattern. The second line shows the equivalent hexadecimal value. The third line shows the decimal value corresponding to the hexadecimal value in the second line. It is the decimal values that are used to decode the bus error slot location.

Step 2 tells you that the bus error occurred in slot 5 in the rack. Note that the "Byte Bit #" vaue will be used in step 4 to help pinpoint the bit at which the error occurred.

3. Interpret the least significant bits of the hex value according to the type of module found in the slot.

For this example, we will assume that the module in slot 5 is a Network Communications module. In this case, we will use the following bit pattern to decode the drop, register, and bit number where the error occurred.




The first line under the figure above describes the meaning of the pattern. The second line shows the equivalent hexadecimal value. The third line shows the decimal value corresponding to the hexadecimal value in the second line. It is the decimal values that are used to decode the bus error register location. In the case of Network or Remotel/O modules, this step also tells you the drop location where the error occurred.

Step 3 tells you that the error occurred in register 2 on drop 3. Note that the "b" value is used in the next step to detemine the bit.

4. In order to determine the bit number at which the error occurred, use the following formula:

$$\text{bit number} = \text{Byte Bit \#} + (8 \times \text{b})$$

To continue with our sample value, the equation would look like this:

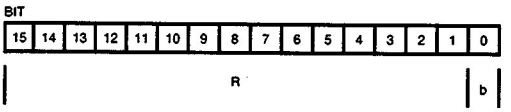
$$\text{bit number} = 3 + (8 \times 0)$$
$$\text{bit number} = 3 + 0$$
$$\text{bit number} = 3$$

Note that if "Byte Bit #" and 8 × b both equal 0, the address may have been accessed as either an integer (16 bits) or bit 0 of byte 0.

5. Use the figures below to determine the register and bit number when bus errors occur on the modules specified. The slot number and "Bit Byte #" are always decoded using the figure in step 2 above. The formula used to find the bit is the same as in step 4 above.

**Any Local I/O Module**
**Least Significant 16 Bits**

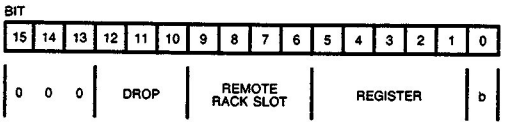

**A Modbus Interface Module M/N 57C414**
**Least Significant 16 Bits**



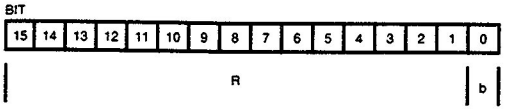

where R is a local register address, which may be used with the I/O monitor. Determine the Modbus register number from the following table:

| R (Decimal) | Modbus Register (Decimal) |
|---|---|
| 64 - 319 | [(R - 64)*16] + BYTE BIT# + 8*b + 1 |
| 320 - 575 | [(R - 320)*16] + BYTE BIT# + 8*b + 10001 |
| 576 - 1599 | (R - 576) + 30001 |
| 1600 - 2623 | (R - 1600) + 40001 |

**A Remote I/O Module M/N 57C416**
**Least Significant 16 Bits**



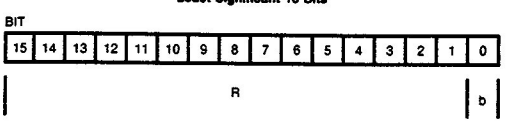

**An AutoMate Interface Module M/N 57C417**
**Least Significant 16 Bits**




where R is a local register address, which may be used with the I/O monitor. Determine the AutoMate register number from the following table:

| R (Decimal) | AutoMate Register (Octal) |
|---|---|
| 64 - 319 | 0000.00 - 0377.17 |
| 320 - 575 | 0400.00 - 0777.17 |
| 576 - 1599 | 2000 - 3777 |
| 1600 - 2623 | 4000 - 5777 |

**An Allen-Bradley Interface Module M/N 57C418**
**Least Significant 16 Bits**

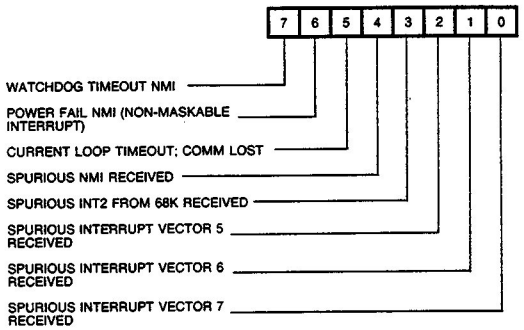



where R is a local register address, which may be used with the I/O monitor. Determine the A-B register number from the following table:

| R (Decimal) | A-B File | A-B Register (Decimal) |
|---|---|---|
| 64 - 319 | B0 | R - 64 |
| 320 - 575 | B1 | R - 320 |
| 576 - 1599 | N0 | R - 576 |
| 1600 - 2623 | N1 | R - 1600 |

# Decoding Error Code 37

When a 37 error code occurs on the Processor module, examine the Error Log for the module using the Programming Executive software. Decode the hexadecimal code found in the Error Log as follows.




Corrective action: Reset the Drive Controller module. If this does not correct the problem, systematically replace the Drive Controller module and then the Processor module.

# Appendix C
# Summary of
# Common DOS Commands

| | |
|---|---|
| CD | Change to specified directory |
| CD\ | Change to root directory |
| CD.. | Change to parent directory of current directory |
| CHKDSK | Checks and displays the condition of a specified disk or file. If no disk drive is specified the command operates on the default drive. |
| CHKDSK/V | Executes the CHKDSK command and displays all files |
| CHKDSK/F | Executes the CHKDSK command and fixes any problems found during the check |
| COPY | Copies a file(s) from one disk or directory to another specified disk or directory<br>e.g. copy (src drv):\(dir)\file.ext (dest drv):\(dir)\file.ext |
| DEL | Deletes specified file(s) |
| DISKCOPY | Copies contents of one disk to another |
| DIR | Lists all files in the default directory |
| DIR/W | Selects wide display for DIR command |
| DIR/P | Selects page mode for DIR command |
| FORMAT | Formats the disk in the specified drive (NOTE: This command erases all files on the disk) |
| MD | Makes (creates) a new directory |
| PATH | Sets or displays a command search path<br>e.g. PATH=C:\DCS |
| PATH; | Sets a null path (i.e. deletes existing path) |
| PROMPT | Changes the normal MS-DOS prompt<br>e.g. PROMPT=$p$g<br>With this command the prompt will now show the default directory |
| RD | Removes (deletes) specified directory (NOTE: All files within specified directory must first be deleted or copied before operation) |
| TYPE | Displays specified file<br>(NOTE: To display a file one screen at a time use the I MORE command)<br>e.g. TYPE filename.ext I MORE |

# Appendix D
# Windows Command Summary

| Access via: | Keyboard | Mouse |
|---|---|---|
| Control menu | ALT,SPACEBAR | Click on menu |
| Menus | ALT,RIGHT(orLEFT), ENTER<br>ALT,underlined letter | Click on menu |
| Commands | DOWN(orUP),ENTER<br>Type underlined letter | Click on command |
| Making a Selection | 1. DOWN (or UP) to locate item<br>2. SPACEBAR to select | Click on item |
| Making Multiple Selections | 1. CTRL+DOWN(orUP) to locate item<br>2. CTRL+SPACEBAR to select<br>3. Repeat for each item to be selected | CTRL+ click on each item to be selected. |
| Moving in a Dialog Box | 1. TAB to area<br>2. Use DIRECTION keys<br>ALT+ underlined letter<br>SPACEBAR toggle choices | Click on desired area,<br>Click on desired item<br>Click to toggle |
| Text Box | 1. Use DIRECTION keys to locate text<br>2. SHIFT+DIRECTION to select text | Drag to select text |
| List Box | 1. Use DIRECTION keys, HOME, END, PAGE UP, PAGE DOWN<br>2. SPACEBAR to select | Use scroll bar, then click on item |
| Moving a Window or Icon | 1. ALT+ESC<br>2. ALT+SPACEBAR<br>3. Press M<br>4. Use DIRECTION keys<br>5. ENTER | Window - Drag title bar<br>Icon - Drag icon |
| Changing Size of a Window | 1. ALT+ESC<br>2. ALT+SPACEBAR<br>3. Press S<br>4. Use DIRECTION keys to select and move border<br>5. ENTER | Drag border or corner |
| Enlarging a Window | 1. ALT+ESC<br>2. ALT+SPACEBAR<br>3. Press X | Click on Maximize box (upper-right corner) |
| Shrinking a Window | 1. ALT+ESC<br>2. ALT+SPACEBAR<br>3. Press N | Click on Minimize box (upper-right corner) |
| Restoring a Window | 1. ALT+ESC<br>2. ALT+SPACEBAR<br>3. Press R | Click on Restore box (upper-right corner) |

| | | |
|---|---|---|
| Restoring an Icon | ALT+TAB | Double-click on icon |
| Scrolling: | | |
| rows, columns | Use DIRECTION keys | Drag scroll box |
| Top of list | HOME | Drag scroll box to top of scroll bar |
| End of list | END | Drag scroll box to bottom of scroll bar |
| up 1 screen | PAGE UP | Click above scroll box |
| down 1 screen | PAGE DOWN | Click below scroll box |
| Cancel a Command | ESC | Click on empty space |
| End session | 1. ALT+first letter of left-most menu<br>2. X to select Exit Windows command<br>3. OK at dialog box | Double click on Control menu, then click on OK at dialog box |

**For additional information**
1 Allen-Bradley Drive
Mayfield Heights, Ohio 44124 USA
Tel: (800) 241-2886 or (440) 646-3599
http://www.reliance.com/automax

**www.rockwellautomation.com**

**Corporate Headquarters**
Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

**Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions**
Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

**Headquarters for Dodge and Reliance Electric Products**
Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433
Europe/Middle East/Africa: Rockwell Automation, Brühlstraße 22, D-74834 Elztal-Dallau, Germany, Tel: (49) 6261 9410, Fax: (49) 6261 17741
Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 6356-9077, Fax: (65) 6356-9011

Publication J-3669-1 - December 1998