XICE Debugger Supplement for Motorola 68000, 68EC000, 68HC000 and 68302 Processors and the EL1600 Emulator

May 1993
P/N 922-17320-02
Copyright © 1993 Applied Microsystems Corporation. All rights reserved.
IBM XT and IBM AT are trademarks of IBM Corporation
Microsoft and MS-DOS™ are trademarks of Microsoft Corporation.
Microtec is a registered trademark of Microtec Research, Inc.
Sun, Sun-4, NFS, and PC-NFS are trademarks of Sun Microsystems, Inc.
UNIX is a registered trademark of AT&T.
VALIDATE is a registered trademark of Applied Microsystems Corporation
Chapter 1
XICE for 68000/EC000/HC000 and 68302

Overview .............................................................. S1-2
Conventions ......................................................... S1-3
Support services .................................................... S1-4
XICE screens ......................................................... S1-4
Command considerations ....................................... S1-7
Register support .................................................. S1-9
Chip select registers (68302 only) ......................... S1-12
Using trace ............................................................ S1-13
  Disassembled trace display .................................... S1-17
Operations during run .......................................... S1-18
Software performance analysis ............................ S1-19
File formats and converters .................................. S1-19

Chapter 2
XICE Command Supplement

ASM — single line assembler ................................. S2-2
BPSPACE — specify breakpoint space .................... S2-4
BREAKACCESS — sets an access breakpoint .............. S2-5
BREAKCOMPLEX — ties a macro to an event break ... S2-10
BRÉAKINSTRUCTION — sets an instruction break .... S2-12
BREAKREAD — sets a read breakpoint .................... S2-16
BREAKWRITE — Sets a write Breakpoint ................. S2-20
BROWSE — display class inheritance information .......... S2-24
BTE — enable or disable emulator bus timeout ............... S2-25
BUS — show state of processor bus signals ................... S2-27
BUSTIME — capture bus timing in trace buffer ............ S2-29
CAS — continuous address strobe ................................ S2-30
CRC — calculate a CRC for a range of memory ............ S2-31
DBP — disable bus error on peek/poke ....................... S2-33
DIAG 0 — simple target ram test .............................. S2-34
DIAG 1 — complex target ram test ............................. S2-36
DIAG 2 — continuous read from target memory ............ S2-38
DIAG 3 — continuous write to target memory ............. S2-40
DIAG 4 — write alternating pattern to target location ... S2-42
DIAG 5 — write rotated to target memory .................. S2-44
DIAG 6 — write then read target memory .................. S2-46
DIAG 7 — continuous read from target memory range ... S2-48
DIAG 8 — write incremental value to target memory ..... S2-50
DIAG 9 — continuous stream of reset pulses ............... S2-52
DNL — download hex file to target .......................... S2-53
DNLFMT — specify download format .......................... S2-54
DNL_GAP — specify maximum bytes between blocks ....... S2-55
DOWN (DOW) — move current scope .......................... S2-56
DRT — display raw trace ........................................... S2-57
DRTMR — enable refresh of memory (68302 only) ........ S2-59
DRUN — start dynamic run mode .............................. S2-60
DSTOP — stop dynamic run mode .............................. S2-62
DT — display trace ................................................ S2-63
DTB — display disassembled trace backwards ............ S2-65
DTF — display disassembled trace forward ............... S2-67
DUPDATE — specify polling frequency ..................... S2-69
EMUVAR S — display emulator variable values .......... S2-70
EV — define an event.................................................. S2-71
EVTARM — enable or disable automatic trigger arming S2-76
EVTCLR — clear event system ....................................... S2-77
EVTGRP — specify event group ..................................... S2-78
EVTVARS — display internal debugger variable values S2-80
EXPLAIN — explain error message.................................. S2-81
EXVE C — software breakpoint execution trap number .... S2-82
FAST — fast interrupt emulation control........................ S2-84
FRZ — freeze peripheral activity (68302 only)............ S2-86
FTO — fast bus timeout............................................... S2-87
GROUP — display active event group ............................ S2-88
HWCONFIG — display hardware name and version ....... S2-89
MEMVARS — display memory access variable values .... S2-90
MWARN — control address out-of-range warnings ....... S2-90
NETERR — specify timeout warning delay.................... S2-92
NETFAIL — specify download abort timeout................... S2-93
NULL_TGT — enable null target mode ............................ S2-94
OVE — overlay memory spaces...................................... S2-96
OVS — set emulator overlay speed ............................... S2-98
PERFACT — enable PA data gathering .......................... S2-100
PERFCLR — remove PA data ......................................... S2-101
PERFDATA — display PA symbol data .......................... S2-102
PERFDEPTH — maximum lines of PA output .................. S2-103
PERFDISP — display PA information ............................. S2-104
PERFEX — exclude addresses from PA ......................... S2-106
PERFEXCLR — clear PA exclusions ............................... S2-108
PERFFORMAT — format of PA display .......................... S2-109
PERFIN T — specify PA time interval ............................ S2-111
PERFMODE — control PA data display ..................... S2-112
PERFTOL — specify symbol search distance ............... S2-113
PPT — peek/poke trace ........................................ S2-114
RAMACCESS — locate a range of RAM memory......... S2-115
RESET — reset processor and target to initial state ... S2-116
RFS — control software refresh............................. S2-117
RFSADR — refresh software addresses.................... S2-119
RFSASP — refresh software address space............... S2-120
RFSMSK — refresh software mask.......................... S2-122
RIRR — control 302 register restoration on reset ...... S2-123
ROMACCESS — locate a range of ROM memory ....... S2-124
RUN_POLL — set number of polls during run........... S2-126
RUN_TIME — set maximum run time ...................... S2-127
SCRATCH — breakpoint scratch area address......... S2-128
SIA — special interrupt vector ............................. S2-129
SIZE — set the size for memory accesses............. S2-130
SLO — slow interrupt emulation control................ S2-132
SPACE — set the space for memory accesses.......... S2-134
STI — enable or disable step-through interrupts.... S2-136
TAD — control tri-state of address bus............... S2-137
TCEBRK — control tracing of breakpoints............... S2-138
TED — control trace/overlay for external DMA ...... S2-139
TID — control trace/overlay for internal DMA ...... S2-141
TRCCLR — clear trace buffer.............................. S2-142
TRCFRAME — trace cycle number........................ S2-143
TRCINT — trace interval.................................. S2-144
TRCMODE — trace mode ..................................... S2-145
TRIG — set status trigger ................................ S2-146
TSRCH — search trace memory for patterns .......... S2-151
Chapter 3

Overview ................................................................. S3-1
User-entered commands........................................ S3-1
Tutorial program.................................................. S3-2
Embedded systems considerations........................ S3-3
Preparing code for debugging............................. S3-3
Using the XICE debugger user interface.............. S3-9
Environment variables........................................... S3-9
Debugger configuration file - xice.cfg................. S3-9
Debugger invocation............................................... S3-11
Include files - introduction................................ S3-12
Using Help............................................................ S3-15
Additional error message information................. S3-17
Navigating XICE windows (viewports).................. S3-17
Modifying and saving debugger start-up options.... S3-18
Recording a debug session.................................... S3-20
Convenience features............................................ S3-20
Using debugger functions ................................... S3-22
Getting oriented with the code............................ S3-22

Contents
Checking the state of the debugger and emulator .... S3-24
Checking the state of the target ......................... S3-27
Controlling the Emulator and CPU .................... S3-28
Memory control ................................................. S3-29
Using overlay memory ...................................... S3-31
Basic breakpoints ............................................. S3-34
Program execution and related commands .......... S3-35
Capturing and displaying execution trace history .... S3-37
Executing XICE commands in dynamic run mode .... S3-41
Logic state and timestamp options .......... S3-42
Debugger macros ............................................. S3-42
Using the event system ................................... S3-45
Scope loops and diagnostics ............................ S3-49

Chapter 4
Using Breakpoints and the Event System

Overview .......................................................... S4-1
Emulator and simulator versions of the debugger .... S4-1
Breakpoint and event system commands .............. S4-2
Working with basic breakpoints .......................... S4-3
Setting basic breakpoints .................................. S4-3
Displaying breakpoints ..................................... S4-3
Clearing breakpoints ....................................... S4-4
Instruction breakpoints (BI, GO_instruction) ....... S4-4
Access breakpoints (BA, BR, BW) ....................... S4-9
Breakpoint latency ........................................... S4-13
Working with the event system ......................... S4-14
Event and trigger statement groups ................. S4-16
Events ........................................................... S4-18
Triggers .......................................................... S4-20
Event system breaks and trace ............................ S4-24
Event system limitations ................................... S4-24

Chapter 5
Using Performance Analysis

Event system setup ............................................ S5-3
Sample include file .......................................... S5-3
Special considerations ....................................... S5-5
Limitations of statistical performance analysis ...... S5-5
Exclusion of address ranges ................................ S5-6

Chapter 6
Using the Time Stamp Module

Overview .......................................................... S6-2
Possible measurements ....................................... S6-2
Installation ....................................................... S6-4
Hardware installation ........................................ S6-4
Using the Time Stamp module .............................. S6-6
Getting started .................................................. S6-6
Making a measurement ....................................... S6-8
Collecting time stamp information in a file .......... S6-15

Chapter 7
Simulated I/O

Using simulated character input with XICE .......... S7-1
Using simulated character output with XICE ........ S7-3
Using simulated character input with XRAY ........ S7-5
Using simulated character output with XRAY ........ S7-6
XICE is Applied Microsystems' integrated debugger for use with Applied in-circuit emulators. It is a part of a complete embedded development toolchain. Other toolchain components include the XRAY simulator, XRAY monitor, MCC and CCC cross-compilers, and ASM cross-assembler, linker, and object module librarian.

This manual provides information that is specific to using XICE with Applied emulators for the Motorola 68000/68HC000/68EC000 and 68302 processors. It supplements the descriptions of core debugger commands found in the XRAY68K Documentation Set.

Two emulator hardware configurations support the 68000 processor. One consists of an emulation board, probe module and probe tip; the other consists of an emulation board and probe tip. The emulation board and probe tip configuration is also used by the 68HC000 and 68EC000 emulators. See Chapter 1 of your Hardware Setup and Reference Guide to identify the configuration of your 68000. The two configurations have minor differences in operation. Unless otherwise noted, all references to 68000 in this supplement refer to both versions of 68000 and to 68HC000 and 68EC000.

This version of XICE is based on the latest version of Microtec Research XRAY68K. It fully supports the output of the 4.3 MCC 68K compiler and 6.9 ASM68K assembler. Although it may function properly with earlier versions of MCC and ASM, Applied Microsystems does not guarantee full backwards compatibility.
Overview

This supplement provides the information you need to use XICE with an EL 1600 emulator for a Motorola 68000/68HC000/68EC000 or 68302 microprocessor.

- Chapter 1 highlights several key concepts, including register support, using trace, operations during run, and software performance analysis. The "Command considerations" section describes exceptions and any XRAY command that is not fully supported in XICE.
- Chapter 2 provides an alphabetical reference of XICE commands.
- Chapter 3 provides a set of mini-tutorials on code preparation, emulation, and XICE setup and use.
- Chapter 4 covers using the standard breakpoints and the optional advanced event system.
- Chapter 5 explains the performance analysis capabilities of XICE.
- Chapter 6 explains installation and use of the Timestamp module.
- Chapter 7 describes simulated I/O functions.

For detailed information on running under X windows on Sun workstations see Appendix A of your XRAY68K User's Guide.

For detailed information on starting the debugger see the XICE Installation Guide.
This manual uses the following conventions:

<table>
<thead>
<tr>
<th>When you see</th>
<th>This means</th>
</tr>
</thead>
<tbody>
<tr>
<td>bold type</td>
<td>The name of a control software configuration or executable file, a keyword or command, or a key that you should press.</td>
</tr>
<tr>
<td>italics</td>
<td>A variable, or a file name. Sometimes italics are used for emphasis the first time a key word or concept is introduced.</td>
</tr>
<tr>
<td>&lt;F7&gt;</td>
<td>Press the F7 function key.</td>
</tr>
<tr>
<td>[option]</td>
<td>Optional item. You do not have to select the option. You do not enter the brackets.</td>
</tr>
<tr>
<td>{this</td>
<td>that}</td>
</tr>
<tr>
<td>{item}...</td>
<td>You may select one or more of the items.</td>
</tr>
<tr>
<td>run</td>
<td>Screen output or example code.</td>
</tr>
</tbody>
</table>
Support services

Applied Microsystems provides a full range of support services. New software is covered by a 90-day warranty that includes full applications phone support. Additional support agreements are available to extend the initial warranty and to provide additional services.

If you encounter trouble installing or using your software, consult your manuals to verify that you are using appropriate procedures. See also the Appendix B, Troubleshooting, in the XICE Installation Guide. It covers the most frequently encountered operational problems.

If the problem persists, call Customer Support at 1-800-ASK-4AMC or 1-206-882-2000 (in Washington State). Customers outside the United States should contact their sales representative or local Applied Microsystems office.

Have your software serial number available. This number displays during boot-up and is also included on the label of your master media.

XICE screens

Figure 1-1 and Figure 1-2 show C-source-level and assembly-level XICE screens.
```c
extern struct com_port com;

if ((com.write_data & STATUS) == CLEAR)
{
    com.write_data = (c | READY);  /* put data with status = ready */
    return c;                      /* return success */
}
else
{
    return 0;                     /* not clear */
}

//***************************************************************
** printcom(p) - Print ascii strings out com port
**
** This function outputs ascii character strings until terminated by
** a null character. An initialized pointer to the ascii text should
** be passed as a character pointer. The actual characters are output
** using putcom() library call.
**
printcom(p)
c = *p;

while ((c = *p++) != '\0')  /* do until null */
{
    while (!putcom(c));         /* write to EMUALTOR (monitor) */
    com.write_data = (c | READY);
    return c;
    return success;
}
```

**Figure 1-1 XICE source-level screen**
Figure 1-2 XICE assembly-level screen
Command considerations

Chapter 3 of the reference manual in the XRAY Documentation Set describes the core set of commands for both XRAY and XICE. The information that follows covers any XICE exceptions to the information in the reference manual.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAKACCESS</td>
<td>Cannot be used when event system triggers are armed. Number limited when BREAKINSTRUCTION also used. See “Basic breakpoints” in Chapter 3 for an overview.</td>
</tr>
<tr>
<td>BREAK-COMPLEX</td>
<td>Variable EVTMODE in xice.cfg must be ON. Restricts event system to one armed trigger. See Chapter 2.</td>
</tr>
<tr>
<td>BREAK-INSTRUCTION</td>
<td>Requires assignment of trap vector and scratch space. See EXVEC and SCRATCH in Chapter 2 and “Instruction Breakpoints” in Chapter 4 for an explanation.</td>
</tr>
<tr>
<td>BREAKREAD</td>
<td>Cannot be used when event system triggers are armed. Number limited when BREAKINSTRUCTION also used. See “Basic breakpoints” in Chapter 3 for an overview.</td>
</tr>
<tr>
<td>BREAKWRITE</td>
<td>Cannot be used when event system triggers are armed. Number limited when BREAKINSTRUCTION also used. See “Basic breakpoints” in Chapter 3 for an overview.</td>
</tr>
<tr>
<td>CLOCK</td>
<td>Not supported in XICE.</td>
</tr>
<tr>
<td>CPU</td>
<td>Not supported in XICE.</td>
</tr>
<tr>
<td>HOST</td>
<td>Causes erratic behavior on some PCs.</td>
</tr>
<tr>
<td>ICE</td>
<td>Not supported in XICE.</td>
</tr>
<tr>
<td>INPORT</td>
<td>Chapter 7 describes the use of INPORT for simulated I/O.</td>
</tr>
<tr>
<td>INTERRUPT</td>
<td>Not supported in XICE.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NOICE</td>
<td>Not supported in XICE.</td>
</tr>
<tr>
<td>NOINTERRUPT</td>
<td>Not supported in XICE.</td>
</tr>
<tr>
<td>NOME</td>
<td>Memory is mapped in blocks of 2K. If you start or end your mapping request at points that are not at multiples of 2K, XICE adjusts the request to meet the 2K requirement.</td>
</tr>
<tr>
<td>OPTION</td>
<td>The options CPU and SPEED are not supported. The option VPCOLOR is supported only for IBM-PCs and compatibles.</td>
</tr>
<tr>
<td>OUTPORT</td>
<td>Chapter 7 describes the use of OUTPORT for simulated I/O.</td>
</tr>
<tr>
<td>OVERLAY</td>
<td>Not supported by the 68000 or 68302 emulator.</td>
</tr>
<tr>
<td>RAM</td>
<td>Memory is mapped in blocks of 2K. If you start or end your mapping request at points that are not at multiples of 2K, XICE adjusts the request to meet the 2K requirement.</td>
</tr>
<tr>
<td>RESTORE</td>
<td>Not supported in XICE.</td>
</tr>
<tr>
<td>ROM</td>
<td>Memory is mapped in blocks of 2K. If you start or end your mapping request at points that are not at multiples of 2K, XICE adjusts the request to meet the 2K requirement.</td>
</tr>
<tr>
<td>SAVE</td>
<td>Not supported in XICE.</td>
</tr>
<tr>
<td>SEARCH</td>
<td>Not supported on XICE.</td>
</tr>
<tr>
<td>SETSTATUS</td>
<td>The following identifiers are not supported in XICE: QUALIFY and TRACE.</td>
</tr>
<tr>
<td>STATUS</td>
<td>The following identifiers are not supported in XICE: QUALIFY and TRACE.</td>
</tr>
<tr>
<td>TRACE</td>
<td>Not supported in XICE.</td>
</tr>
</tbody>
</table>
The following are the 68000 and 68302 registers that are supported by XICE. You can enter them in either upper case or lower case, but they are displayed in the case shown.

<table>
<thead>
<tr>
<th>Register</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AO-A7</td>
<td>Address Registers</td>
</tr>
<tr>
<td>D0-D7</td>
<td>Data Registers</td>
</tr>
<tr>
<td>FP</td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>Program Counter</td>
</tr>
<tr>
<td>SP</td>
<td>Supervisor Stack Pointer</td>
</tr>
<tr>
<td>SSP</td>
<td>Status Register</td>
</tr>
<tr>
<td>USP</td>
<td>User Stack Pointer</td>
</tr>
</tbody>
</table>

The following additional registers, which are related to just the 68302, are also supported by XICE:

<table>
<thead>
<tr>
<th>Register</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAR</td>
<td>Base Address Register</td>
</tr>
<tr>
<td>BCR</td>
<td>Byte Count Register</td>
</tr>
<tr>
<td>BR0</td>
<td>Base Register 0</td>
</tr>
<tr>
<td>BR1</td>
<td>Base Register 1</td>
</tr>
<tr>
<td>BR2</td>
<td>Base Register 2</td>
</tr>
<tr>
<td>BR3</td>
<td>Base Register 3</td>
</tr>
<tr>
<td>CMR</td>
<td>Channel Mode Register</td>
</tr>
<tr>
<td>CR</td>
<td>Command Register</td>
</tr>
<tr>
<td>Register</td>
<td>Name</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>CSR</td>
<td>Channel Status Register</td>
</tr>
<tr>
<td>DAPR</td>
<td>Destination Address Pointer Register</td>
</tr>
<tr>
<td>DSR1</td>
<td>SCC1 Data Sync Register</td>
</tr>
<tr>
<td>DSR2</td>
<td>SCC2 Data Sync Register</td>
</tr>
<tr>
<td>DSR3</td>
<td>SCC3 Data Sync Register</td>
</tr>
<tr>
<td>FCR</td>
<td>Function Code Register</td>
</tr>
<tr>
<td>GIMR</td>
<td>Global Interrupt Mode Register</td>
</tr>
<tr>
<td>IMR</td>
<td>Interrupt Mask Register</td>
</tr>
<tr>
<td>IPR</td>
<td>Interrupt Pending Register</td>
</tr>
<tr>
<td>ISR</td>
<td>Interrupt In-Service Register</td>
</tr>
<tr>
<td>OR0</td>
<td>Option Register 0</td>
</tr>
<tr>
<td>OR1</td>
<td>Option Register 1</td>
</tr>
<tr>
<td>OR2</td>
<td>Option Register 2</td>
</tr>
<tr>
<td>OR3</td>
<td>Option Register 3</td>
</tr>
<tr>
<td>PACNT</td>
<td>Port A Control Register</td>
</tr>
<tr>
<td>PADAT</td>
<td>Port A Data Register</td>
</tr>
<tr>
<td>PADDR</td>
<td>Port A Data Direction Register</td>
</tr>
<tr>
<td>PBCNT</td>
<td>Port B Control Register</td>
</tr>
<tr>
<td>PBDAT</td>
<td>Port B Data Register</td>
</tr>
<tr>
<td>PBDDDR</td>
<td>Port B Data Direction Register</td>
</tr>
<tr>
<td>SAPR</td>
<td>Source Address Pointer Register</td>
</tr>
<tr>
<td>SCCE1</td>
<td>SCC1 Event Register</td>
</tr>
<tr>
<td>SCCE2</td>
<td>SCC2 Event Register</td>
</tr>
<tr>
<td>SCCE3</td>
<td>SCC3 Event Register</td>
</tr>
<tr>
<td>Register</td>
<td>Name</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>SCCM1</td>
<td>SCC1 Mask Register</td>
</tr>
<tr>
<td>SCCM2</td>
<td>SCC2 Mask Register</td>
</tr>
<tr>
<td>SCCM3</td>
<td>SCC3 Mask Register</td>
</tr>
<tr>
<td>SCCS1</td>
<td>SCC1 Status Register</td>
</tr>
<tr>
<td>SCCS2</td>
<td>SCC2 Status Register</td>
</tr>
<tr>
<td>SCCS3</td>
<td>SCC3 Status Register</td>
</tr>
<tr>
<td>SCM1</td>
<td>SCC1 Mode Register</td>
</tr>
<tr>
<td>SCM2</td>
<td>SCC2 Mode Register</td>
</tr>
<tr>
<td>SCM3</td>
<td>SCC3 Mode Register</td>
</tr>
<tr>
<td>SCON1</td>
<td>SCC1 Configuration Register</td>
</tr>
<tr>
<td>SCON2</td>
<td>SCC2 Configuration Register</td>
</tr>
<tr>
<td>SCON3</td>
<td>SCC3 Configuration Register</td>
</tr>
<tr>
<td>SCR</td>
<td>System Control Register</td>
</tr>
<tr>
<td>SIMASK</td>
<td>Serial Interface Mask Register</td>
</tr>
<tr>
<td>SIMODE</td>
<td>Serial Interface Mode Register</td>
</tr>
<tr>
<td>SPMD</td>
<td>SCP, SMC Mode and Clock Control</td>
</tr>
<tr>
<td>TCN1</td>
<td>Timer Counter Register</td>
</tr>
<tr>
<td>TCN2</td>
<td>Timer Counter Register</td>
</tr>
<tr>
<td>TCR1</td>
<td>Timer Capture Register</td>
</tr>
<tr>
<td>TCR2</td>
<td>Timer Capture Register</td>
</tr>
<tr>
<td>TER1</td>
<td>Timer Event Register</td>
</tr>
<tr>
<td>TER2</td>
<td>Timer Event Register</td>
</tr>
<tr>
<td>TMR1</td>
<td>Timer Mode Register</td>
</tr>
<tr>
<td>Register</td>
<td>Name</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>TMR2</td>
<td>Timer Mode Register</td>
</tr>
<tr>
<td>TRR1</td>
<td>Timer Reference Register</td>
</tr>
<tr>
<td>TRR2</td>
<td>Timer Reference Register</td>
</tr>
<tr>
<td>WCN</td>
<td>Watchdog Counter</td>
</tr>
<tr>
<td>WRR</td>
<td>Watchdog Reference Register</td>
</tr>
</tbody>
</table>

## Chip select registers (68302 only)

You must configure the 68302 chip select registers to match the target system before starting up XICE. The EL 1600 requires that the chip select registers be programmed to respond in all function code spaces to permit target memory operations to work correctly. Memory operations (also referred to as peeks and pokes) include actions such as displaying memory, downloading code and data, loading the reset vectors, and fill and block moves.

You can configure the chip select registers in the `xice.cfg` file. The *XICE Installation Guide* describes how to preset registers using `xice.cfg`. The *El 1600 Hardware Setup and Reference Guide* also provides information on setting the chip select registers.

It is important to set up the chip select registers correctly; otherwise, it is possible to inadvertently program a DTACK over the memory location of emulator internal memory. Then, if the processor supplies a DTACK for emulator internal memory, an emulator crash can occur.

Once the chip select registers are loaded initially, the switches RIRR and UIR determine when they are restored or updated. RIRR set to ON restores the registers after an emulator reset. UIR set to ON updates the registers after an emulation break occurs or whenever you make a change to any of the register values.
If you do not set RIRR to ON, the emulator will not reload the register values after a reset.

If you do not leave UIR set to ON, the emulator’s copy of the internal registers will not be updated until the next RUN to PAUSE transition.

---

**Using trace**

The XICE trace capability allows you to view either raw bus cycles or disassembled source code interleaved with the high-level code.

In addition to the softswitches, the following commands set up trace:

- **PPT** controls the tracing of peeks and pokes
- **TCEBRK** controls the tracing of breakpoints
- **TED** controls whether external or internal DMA is traced
- **TID** controls whether external or internal DMA is traced
- **TRCFRAME** sets the trace cycle numbers
- **TRCINT** specifies how traced timestamp information is displayed
- **TRCMODE** sets trace mode (assembly and source, assembly only, or source only)
- **TSTAMP** controls whether timestamp or LSA is traced
- **TUNITS** sets the timestamp units
The commands to use trace are as follows:

- **DRT**: display raw trace
- **DT**: display trace, both assembly and disassembled source level
- **DTB**: display trace backwards, both assembly and disassembled source level
- **DTF**: display trace forwards, both assembly and disassembled source level
- **TRCCLR**: clear trace buffer
- **TRS Rach**: search for specified pattern in trace

The commands to set up and use trace are described in Chapter 2 of this supplement. When you use a trace command, the trace information is displayed in the command viewport. Raw trace is formatted as shown in Figure 1-3.

![Figure 1-3 Raw trace format](image-url)
**Raw trace display**

The raw trace display columns shown in Figure 1-3 are:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAME</td>
<td>The decimal count of the line in the trace buffer. Line 0 corresponds to the most recently traced cycle.</td>
</tr>
<tr>
<td>ADDRESS</td>
<td>The hex value of the address bus.</td>
</tr>
<tr>
<td>DATA</td>
<td>The hex value of the data bus.</td>
</tr>
<tr>
<td>IPL</td>
<td>This column lists the bustime information if the BUSTIME is set to ON. If BUSTIME is set to OFF, this column lists the current interrupt level. The range for the bustime information is 3 to 9. Raw trace displays a plus mark (+) if bustime is outside of this range.</td>
</tr>
<tr>
<td>FCn</td>
<td>The memory space accessed:</td>
</tr>
<tr>
<td></td>
<td>UD  user data space</td>
</tr>
<tr>
<td></td>
<td>UP  user program space</td>
</tr>
<tr>
<td></td>
<td>SD  supervisor data space</td>
</tr>
<tr>
<td></td>
<td>SP  supervisor program space</td>
</tr>
<tr>
<td></td>
<td>PU  CPU space</td>
</tr>
<tr>
<td></td>
<td>000 reserved memory space</td>
</tr>
<tr>
<td></td>
<td>011 reserved memory space</td>
</tr>
<tr>
<td></td>
<td>100 reserved memory space</td>
</tr>
<tr>
<td>MEM</td>
<td>The memory type accessed, its boundary, and its location:</td>
</tr>
<tr>
<td></td>
<td>R read  B byte  T target</td>
</tr>
<tr>
<td></td>
<td>W write  W word  O overlay</td>
</tr>
<tr>
<td></td>
<td>For example, WBT indicates a byte wide write to target memory and RWO means a word wide read from overlay.</td>
</tr>
<tr>
<td>VPA(68000)</td>
<td>State of valid peripheral access pins.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DMA (68302)</td>
<td>A direct memory access. For the 68302, this information indicates internal DMA cycles if TID is set to ON, or external DMA cycles if TED is set to ON. (TED and TID relate to the 68302 only.) If both TID and TED are set to ON, trace will show the DMA cycles for both TED and TID but there will be no way to determine which was a result of internal DMA and which was a result of external DMA.</td>
</tr>
<tr>
<td>VMA(68000)</td>
<td>State of valid memory access pins.</td>
</tr>
<tr>
<td>IAC (68302)</td>
<td>CPU internal access pin.</td>
</tr>
<tr>
<td>FLAGS</td>
<td>Flags set: X cycles for which the emulator break bit is asserted B bus error M illegal memory access</td>
</tr>
<tr>
<td>LSA BITS</td>
<td>LSA BITS displays the state of each pin of the LSA during that bus cycle. This column does not appear if you set TSTAMP to ON but is replaced by timestamp information.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>The timestamp information is recorded as the interval between successive bus cycles, if TRCINT is set to INTERVAL, or relative to the bus cycle number specified by the command TRCFRAME, if TRCINT is set to OFFSET. This column does not appear if you set TSTAMP to OFF but is replaced by LSA information.</td>
</tr>
</tbody>
</table>
Disassembled trace display

Figure 1-4 shows the disassembled trace format. The information that follows this figure defines the information in each field.

```
if (putcom(led_port[i] )) i++;
0113 0056D0 4E71 NOP
0111 0056D2 58BF ADDQ.L #4,A7
0109 0056D4 4A80 TST.L D0
0107 0056D6 671E BEQ.S $0056F6
 >> while ( coarse--) /* with above string */
0103 0056F6 2002 MOVE.L D2.D0
0101 0056F8 5382 SUBQ.L #1,D2
0099 0056FA 4A80 TST.L D0
0097 0056FC 6A96 BNE.S $0056A4
 >> if (! (i == 0) && (i < 17))
0093 0056A4 4AB90000 TST.L 1
      5E18 0056E1B 00000000
0087 0056A9 6D34 BLT.S $0056E0
0081 0056AC 7011 MOVEQ.L #11,D0
0079 0056AE B8B90000 CMP.L 1,D0
      5E18 0056E1B 00000000
0073 0056B4 6F2A BLE.S $0056E0
 >> if (putcom(led_port[i] )) i++;
0067 0056B6 30390000 MOVE.W i+2,D0
      5E1A 0056E1A 00000000
0061 0056BC 287C0000 MOVE.L #0000604E,A0
      604E 0056E1A 00000000
0053 0056C2 10300000 MOVE.B $0(A0,D0,.W),D0 00604E>7C
0049 0056C6 4880 EXT.W D0
0046 0056CA 48C0 EXT.L D0
0044 0056CC 2F00 MOVE.L D0,-(A7) 007FB0<00000007C
0042 0056CE 4EEA0004 JSR putcom
0034 putcom:
```

Figure 1-4 Disassembled trace format

C source

If trcmode is set to BOTH, C source is interleaved into disassembly.

Cycle number

An index of the bus cycle in the trace buffer. The most recently traced cycle is 1. This number corresponds to the frame number in the raw trace display.

Address

Address of instruction in memory.
Object code
Numeric representation of assembly code.

Instruction
Assembly language instruction in text form.

Symbols
English text representation of addresses, operands and data.

Data movements
The data cycles that occurred as a result of the instruction:

• address < data data written to address
• data > address data read from address

Operations during run

In normal operating mode, XICE does not permit additional operations while the emulator is running. The DRUN, DSTOP, and DUPDATE commands provide this additional functionality.

The DRUN (dynamic run) command executes the target program and continues execution until it is stopped by DSTOP, a breakpoint, an error, or a halt instruction. The purpose of this mode is to allow you to interact with the emulator and debugger dynamically, while the emulator is running. In DRUN you can examine and qualify trace, set and change events and breakpoints, examine and change memory and perform most other interactive functions with the emulator.

The DUPDATE command allows you to specify how frequently the emulator is polled and the viewports updated during DRUN (dynamic run) mode. However, dynamic commands are no longer accepted.

Chapter 2 describes each command in detail. Chapter 3 includes a short tutorial.
Software performance analysis

The performance analysis features of XICE for the EL1600 68000 and 68302 development system allow you to:

- determine which areas of a program use the most CPU time
- identify bottlenecks in time critical applications
- monitor the effects of programming changes made to improve throughput.

Using statistical performance analysis, these features sample instruction fetch bus cycles at regular intervals using the event system; determine what function was active when a sample was taken; keep a tally of the number of samples falling within each function; and report the sample information. The report is in a user-specifiable format.

Chapter 5 provides an overview of performance analysis.

File formats and converters

Although XICE requires IEEE695 object format to enable symbolic debugging, support for other formats is built into XICE, and additional converters are available. See the descriptions in Chapter 2 for UPL, DNL, UPLFMT, and DNLFMT for supported non-IEEE695 formats, procedures, and limitations. Contact your Applied Microsystems representative for information about additional converters.
Core debugger commands are covered in the *XRAY Reference Manual*. This chapter lists commands that are specific to XICE and are not listed in the *XRAY Reference Manual* or that are XICE-specific implementations of core XRAY commands. These commands are entered in the command viewport at the XICE prompt. Any combination of upper-case and lower-case letters can be used in commands.
ASM — single line assembler

Works with

   □ EL 1600   □ CodeTAP

Syntax

   ASM [address]

Description

The ASM command allows you to enter assembly code. If an address is included, assembly will begin at that point. If no address is included, assembly will begin at the last assembly address. A carriage return with no assembly instruction terminates the assembler.

If loaded, symbols can be used in instructions, for example

call main

The prompt for the single line assembler is the address followed by a colon, e.g.,

   00000000:

Notes

The line assembler does not accept symbols that are not already in the symbol table.

In single-line assembler mode, you can change assembly locations with the ORG command. For example,

   org 100

will change the assembly location to 100.
Example

```
ASM 0X100
00000100: move.l (A7), A0
00000102: add.l #6, D0
00000108: move.l #0, (A0)
0000010e:
```
BPSPACE— specify breakpoint space

Works with

- EL 1600
- CodeTAP

Syntax

BPSPACE [USER|SUPERVISOR|ANY]

Description

This command allows you to explicitly specify the space for breakpoints. The choices are USER, SUPERVISOR, or ANY. The default is ANY. If no argument is given, the state of the switch is displayed.

Notes

You can also specify a default for BPSPACE in XICE.CFG with SW_BPSPACE.

Example

BPSPACE ANY

Related Commands

XICEVARS
## BREAKACCESS—sets an access breakpoint

### Works with
- **EL 1600**
- **CodeTAP**

### Abbreviation
- **BA**

### Syntax

```
BREAKACCESS [address | address_range] [;macro_name()]
```

### Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>Specifies the address of the breakpoint.</td>
</tr>
<tr>
<td>address_range</td>
<td>Specifies the address range for the breakpoint. A breakpoint will be set at each statement within the address range.</td>
</tr>
<tr>
<td>macro_name()</td>
<td>Specifies a macro to be executed when the breakpoint is encountered. Note that the macro name must include a set of parentheses: macro1(). When a macro is executed, the macro controls whether program execution stops or continues.</td>
</tr>
</tbody>
</table>
The **BREAKACCESS** command sets an access breakpoint at the specified memory location(s). An access breakpoint halts program execution each time the target program attempts to read from or write to the specified memory location(s). Memory locations can contain code or data.

Use an ampersand (\&) to reference symbolic addresses rather than just the symbol names. Using a symbol name alone returns its value, not the address.

The debugger performs the following functions when an access breakpoint is encountered:

1. Completes the execution of the instruction at that location.
2.Suspends program execution.
3. Executes a macro (if one was specified when the breakpoint was set). Depending on the macro, the debugger will do one of the following:
   **4. If the macro return value is true (nonzero), the debugger resumes execution at the instruction immediately after the break.**
   **5. If the macro return value is false (zero), the debugger returns to command mode and displays break information.**
   **6. If a macro was not specified, the debugger returns to command mode and displays break information.**

**XICE** automatically assigns a breakpoint number to each breakpoint; this number is used to reference or clear the breakpoint. The Break viewport displays breakpoint numbers, breakpoint locations, breakpoint types, and breakpoint command arguments. In high-level mode, the line number and module name containing the breakpoint are also displayed. The breakpoint type for the **BREAKACCESS** command is **ACCESS**.

Breakpoints can be removed with the **CLEAR** command.

If the **BREAKACCESS** command is specified without parameters, the debugger displays the Break viewport.
### Notes

Access breakpoints are set to break on a read (br), a write (bw), or a read or write (ba) of the breakpoint address. These breakpoints are implemented using emulator hardware and may be set in RAM or ROM.

You cannot set an access breakpoint when any event system triggers are armed. Disarming the triggers will allow you to set access breakpoints. Likewise, setting access breakpoints causes the event system to be disabled. Clearing the access breakpoints will allow you to arm the event system triggers.

With instruction breakpoints, the break occurs before the instruction at the specified address is executed. With access breakpoints, the break begins on the cycle in which the access occurs but may continue or “skid” several cycles after access of the breakpoint location.

Cannot be used when event system triggers are armed. Number limited when BREAKINSTRUCTION also used. See “Basic breakpoints” in Chapter 3 for an overview.

### Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA @sieve\flags</td>
<td>Sets a breakpoint at the address of the variable array flags in the root named @sieve.</td>
</tr>
<tr>
<td>BA flags..flags+10</td>
<td>Sets breakpoints starting at the address of the array flags and ending 10 bytes after the address of flags.</td>
</tr>
<tr>
<td>BA &amp;flags[0]</td>
<td>Sets a breakpoint at the address of the array element flags[0].</td>
</tr>
</tbody>
</table>
Sets a breakpoint at line 17 and executes the macro readprime when the breakpoint is encountered.

Sets a breakpoint at the address referred to by the value in variable prime.

This command is correct if prime is a pointer. The breakpoint is set at the location specified by the variable prime. For example, if the value of prime is 0x0123, a breakpoint is set at the address 0x0123.

This command may not be correct if prime is a scalar, since the value in prime is treated as an address and the breakpoint is set at that address rather than at the address of the variable prime.

Sets a breakpoint at the address of the variable prime regardless of its type.

This command is correct if prime is a scalar; it sets a breakpoint at the address of the variable prime.
If `prime` is a pointer, the breakpoint is set at the address of the pointer rather than at the address it is pointing to (i.e., `prime`).

- **BA 20**
  - Sets a breakpoint at address 20.

- **BA flags..flags+9;when (*flags == 1)**
  - Sets breakpoints starting at the address of the array `flags` and ending 9 bytes after the address of `flags`, but the predefined `when` macro stops debugger execution when the first element of `flags` is equal to 1.

**Related Commands**

- `BREAKINSTRUCTION`, `BREAKREAD`, `BREAKWRITE`, `CLEAR`, `GO`, `GOSTEP`, `STEP`, `STEPOVER`
BREAKCOMPLEX — ties a macro to an event system break

Works with

- EL 1600
- CodeTAP

Syntax

```
BC trig(n)[;macro_name]
```

Description

BC is used to associate a macro with an event system breakpoint. Complex breakpoints are used to halt program execution and then execute the specified macro. The argument `trig(n)` refers to a trigger that you must set using the command TRIG following setting up the BC command. If a macro is specified, it is executed each time a break is encountered. Execution continues if the macro returns non-zero.

BC (with an attached macro) works the same way as the other breakpoint commands (BREAKACCESS, BREAKREAD, BREAKWRITE, and BREAKINSTRUCTION).

The debugger performs the following functions when a complex breakpoint is encountered:

1. Completes the execution of the instruction at that location.
2. Suspends program execution.
3. Executes a macro (if one was specified when the breakpoint was set). Depending on the macro, the debugger will do one of the following:
   4. If the macro return value is true (non-zero), the debugger resumes execution at the instruction immediately after the break.
   5. If the macro return value is false (zero), the debugger returns to command mode and displays break information.
6. If a macro was not specified, the debugger returns to command mode and displays break information.

All breakpoints are automatically assigned a breakpoint number by the debugger, which is used to reference or clear the breakpoint. The break viewport displays breakpoint numbers, breakpoint locations, breakpoint types, and breakpoint command arguments. In high-level mode, the line number and module name containing the breakpoint are also displayed.

Breakpoints are removed with the CLEAR command.

Notes

You must set the variable EVTMODE to ON to use BC.

While EVTMODE is set to ON, XICE is restricted to only one event system breakpoint armed at a time even though STAT TRIG may display other triggers as armed.

Examples

```
bc trig(2); done()
```

Executes the macro done each time trigger 2 is true. Breaks execution when done returns a zero.

Related Commands

BREAKACCESS, BREAKINSTRUCTION, BREAKREAD, BREAKWRITE, CLEAR, EV, EVTMODE, TRIG
BREAKINSTRUCTION — sets an instruction breakpoint

Works with

- EL 1600
- CodeTAP

Abbreviation

B

Syntax

BREAKINSTRUCTION [address | address_range] [,macro_name]

Description

- **address**: Specifies the address of the breakpoint.
- **address_range**: Specifies the address range for the breakpoint. If you specify a range of instruction breakpoints, they will be set on the first byte of each instruction or (for high-level code) the first instruction of each line.
- **macro_name**: Specifies a macro to be executed when the breakpoint is encountered. Note that the macro name must include a set of parentheses: macro1(). When a macro is executed, the macro controls whether program execution stops or continues.
The **BREAKINSTRUCTION** command sets an instruction breakpoint at the specified memory location(s). An instruction breakpoint halts program execution each time the target program attempts to execute an instruction at the specified memory location(s).

The debugger performs the following functions when an instruction breakpoint is encountered:

1.Suspends program execution before the instruction at the breakpoint address is executed.

2. Executes a macro (if one was specified when the breakpoint was set). Depending on the macro, the debugger will do one of the following:

3. If the macro return value is true (nonzero), the debugger resumes execution starting at the instruction where the break occurred and displays break information.

4. If the macro return value is false (zero), the debugger returns to command mode without executing the instruction where the break occurred.

5. If a macro was not specified, the debugger returns to command mode without executing the instruction where the breakpoint was encountered.

**XICE** automatically assigns a breakpoint number to each breakpoint; this number is used to reference or clear the breakpoint. The Break viewport displays breakpoint numbers, breakpoint locations, breakpoint types, and breakpoint command arguments. In high-level mode, the line number and module name containing the breakpoint are also displayed.

The breakpoint type for the **BREAKINSTRUCTION** command is **INST**.

Breakpoints can be removed with the **CLEAR** command.

If the **BREAKINSTRUCTION** command is specified without parameters, the debugger displays the Break viewport.
Notes

See Chapter 4 for more detailed explanation of breakpoint use.

Execution breakpoints consume event system resources and affect what is possible using other features. In general, the emulator manages these resources and warns you when it makes adjustments and presents an error when resources are exhausted or when you attempt something that creates a conflict. So you need not concern yourself with more than the following general guidelines.

- If you use them in addition to the event system, note that instruction breakpoints consume an address and a status resource in each event group. Limit event system address/status resource consumption to no more than one address and one status comparator in each group. Set up the event and trigger statements, but leave them disarmed until you are ready to use them.

- You can specify an address range. XICE handles ranges by breaking them into multiple individual single-point breakpoints. Thus, if you specify that a breakpoint should be for a range of 20 addresses, you may set only 12 additional breakpoints.

- If you use them with access breakpoints, note the limitations explained in the BREAKACCESS command description.

- When XICE is invoked, it performs a read of the area designated for SCRATCH if SW_SCRATCH is specified in xice.cfg. If SCRATCH is set to an area of memory that does not return a DTACK at the end of the read, it will hang XICE. In such a case, comment out the default address for SCRATCH (0x9ff0) in the xice.cfg file. Then specify the address for SCRATCH before setting any breakpoints by using the SCRATCH command or preferably in an include file when you invoke XICE.
## Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>B #20</td>
<td>Sets a breakpoint at line number 20.</td>
</tr>
<tr>
<td>B 2210h..2216h</td>
<td>Sets breakpoints starting at address 2210 and ending at address 2216 (hexadecimal), assembly-level mode only.</td>
</tr>
<tr>
<td>BREAK #1..#4</td>
<td>Sets breakpoints starting at line number 1 and ending at line number 4.</td>
</tr>
<tr>
<td>B #15..#18;FOO()</td>
<td>Sets breakpoints starting at line number 15 and ending at line number 18. Executes macro FOO after each line.</td>
</tr>
<tr>
<td>B SIEVE#28</td>
<td>Sets a breakpoint at line number 28 in the module SIEVE.</td>
</tr>
<tr>
<td>B #10;when(i==3)</td>
<td>Sets a breakpoint at line number 10 and stops only if variable i is equal to 3.</td>
</tr>
<tr>
<td>B 0x93</td>
<td>Sets a breakpoint at address 93 (hexadecimal), assembly-level mode only.</td>
</tr>
</tbody>
</table>

## Related Commands

BREAKACCESS, BREAKREAD, BREAKWRITE, CLEAR, GO, GOSTEP, STEP, STEPOVER
BREAKREAD — sets a read breakpoint

Works with

- EL 1600
- CodeTAP

Abbreviation

BR

Syntax

BREAKREAD [address | address_range] [;macro_name]

Description

- **address**: Specifies the address of the breakpoint.

- **address_range**: Specifies the address range for the breakpoint. A breakpoint will be set at each statement within the address range.

- **macro_name**: Specifies a macro to be executed when the breakpoint is encountered. Note that the macro name must include a set of parentheses: macro1(). When a macro is executed, the macro controls whether program execution stops or continues.

The **BREAKREAD** command sets a read breakpoint at the specified memory location(s). A read breakpoint halts program execution each time the target program attempts to read data from the specified memory location(s).
Use an ampersand (&) to reference symbolic addresses rather than just the symbol names. Using a symbol name alone returns its value, not the address.

The debugger performs the following functions when a read breakpoint is encountered:

1. Completes the execution of the instruction at that location.
2.Suspends program execution.
3. Executes a macro (if one was specified when the breakpoint was set). Depending on the macro, the debugger will do one of the following:
4. If the macro return value is true (nonzero), the debugger resumes execution at the instruction immediately after the breakpoint.
5. If the macro return value is false (zero), the debugger returns to command mode and displays breakpoint information.
6. If a macro was not specified, the debugger returns to command mode and displays updated breakpoint information.

XICE automatically assigns a breakpoint number to each breakpoint; this number is used to reference or clear the breakpoint. The Break viewport displays breakpoint numbers, breakpoint locations, breakpoint types, and breakpoint command arguments. In high-level mode, the line number and module name containing the breakpoint are also displayed. The breakpoint type for the **BREAKREAD** command is **READ**.

Breakpoints can be removed with the **CLEAR** command.

If the **BREAKREAD** command is specified without parameters, the debugger displays the Break viewport.

**Notes**

See **BREAKACCESS** for restrictions.
## Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR 0x300</td>
<td>Sets a breakpoint at address 300 (hexadecimal).</td>
</tr>
<tr>
<td>BR @sieve\flags</td>
<td>Sets a breakpoint at the address of the variable array flags in the root named @sieve.</td>
</tr>
<tr>
<td>BR flags..flags+10</td>
<td>Sets breakpoints starting at the address of the array flags and ending 10 bytes after the address of flags.</td>
</tr>
<tr>
<td>BR 20h..30h;FOO()</td>
<td>Sets breakpoints from address 20h (hexadecimal) to 30h and executes the macro FOO on every breakpoint between these addresses.</td>
</tr>
<tr>
<td>BR &amp;flags[0]</td>
<td>Sets a breakpoint at the address of array element flags[0].</td>
</tr>
<tr>
<td>BR prime</td>
<td>Sets a breakpoint at the address referred to by the value in variable prime.</td>
</tr>
</tbody>
</table>

This command is correct if prime is a pointer. The breakpoint is set at the location of the variable prime. For example, if the value of prime is 0x0123, a breakpoint is set at the address 0x0123.
This command may not be correct if `prime` is a scalar, since the value in `prime` is treated as an address and the breakpoint is set at that address rather than at the address of the variable `prime`.

Sets a breakpoint at the address of the variable `prime` regardless of its type.

This command is correct if `prime` is a scalar; it sets a breakpoint at the address of the variable `prime`.

If `prime` is a pointer, the breakpoint is set at the address of the pointer rather than at the address it is pointing to (i.e., `prime`).

Sets a breakpoint at the address of `count` and only stops when the value of `k` is less than 30.

Related Commands

BREAKACCESS, BREAKINSTRUCTION, BREAKWRITE, CLEAR, GO, GOSTEP, STEP, STEPOVER
BREAKWRITE — Sets a Write Breakpoint

Works with

- EL 1600
- CodeTAP

Abbreviation

BW

Syntax

BREAKWRITE [address | address_range] [:macro_name]

Description

- **address**: Specifies the address of the breakpoint.

- **address_range**: Specifies the address range for the breakpoint. A breakpoint will be set at each statement within the address range.

- **macro_name**: Specifies a macro to be executed when the breakpoint is encountered. Note that the macro name must include a set of parentheses: macro1(). When a macro is executed, the macro controls whether program execution stops or continues.

The **BREAKWRITE** command sets a write breakpoint at the specified memory location(s). A write breakpoint halts program execution each time the target program attempts to write data to the specified memory location(s).
The debugger performs the functions listed below when a write breakpoint is encountered:

1. Completes the execution of the instruction at that location.
2. Suspends program execution.
3. Executes a macro if one was specified when the breakpoint was set. Depending on the macro, the debugger will do one of the following:
   4. If the macro return value is true (nonzero), the debugger resumes execution at the instruction immediately after the breakpoint.
   5. If the macro return value is false (zero), the debugger returns to command mode and displays breakpoint information.
   6. If a macro was not specified, the debugger returns to command mode and displays viewport information and breakpoint information.

XICE automatically assigns a breakpoint number to each breakpoint; this number is used to reference or clear the breakpoint. The Break viewport displays breakpoint numbers, breakpoint locations, breakpoint types, and breakpoint command arguments. In high-level mode, the line number and module containing the breakpoint are also displayed. The breakpoint type for the BREAKWRITE command is WRITE.

Breakpoints can be removed with the CLEAR command.

If the BREAKWRITE command is specified without parameters, the debugger displays the Break viewport.

Notes

See BREAKACCESS for restrictions.

Related Commands

BREAKACCESS, BREAKINSTRUCTION, BREAKREAD, CLEAR, GO, GOSTEP, STEP, STEPOVER
## Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BW @sieve\flags</td>
<td>Sets a breakpoint at the address of the variable array flags in the root named @sieve.</td>
</tr>
<tr>
<td>BW flags..flags+10</td>
<td>Sets breakpoints starting at the address of the array flags and ending 10 bytes after the address of flags.</td>
</tr>
<tr>
<td>BW 0x100;FOO()</td>
<td>Sets a breakpoint at address 0x100 and executes the macro FOO on the break.</td>
</tr>
<tr>
<td>BW &amp;flags[0]</td>
<td>Sets a breakpoint at the address of array element flags[0].</td>
</tr>
<tr>
<td>BW prime</td>
<td>Sets a breakpoint at the address referred to by the value in variable prime.</td>
</tr>
</tbody>
</table>

This command is correct if prime is a pointer. The breakpoint is set at the location specified by the variable prime. For example, if the value of prime is 0x0123, a breakpoint is set at the address 0x0123.

This command may not be correct if prime is a scalar, since the value in prime is treated as an address and the breakpoint is set at that address rather than at the address of the variable prime.
**BW &prime**

Sets a breakpoint at the address of the variable `prime` regardless of its type.

This command is correct if `prime` is a scalar; it sets a breakpoint at the address of the variable `prime`.

If `prime` is a pointer, the breakpoint is set at the address of the pointer rather than at the address it is pointing to (i.e., `prime`).

**BREAKW 100h**

Sets a breakpoint at address `100h` (hexadecimal).
BROWSE — display class inheritance information

Works with

- EL 1600
- CodeTAP

Syntax

BROWSE SYMBOL_NAME

Description

This command displays the inheritance relationships of a C++ class. It shows the base classes (parents) and the derived classes (children) of the given class.

The symbol name that you specify may be the name of a class, object, or class member.

Notes


Example

BROWSE COUT
BTE — enable or disable emulator bus timeout

Works with

- EL 1600  □ CodeTAP

Syntax

BTE [ON|OFF]

Description

This switch controls the emulator bus timeout. The valid arguments are ON and OFF.

When set to ON, during RUN mode the EL 1600 will automatically time out in the length of time specified by the switch FTO if the target fails to supply a DTACK signal. It performs the timeout by sending a DTACK signal to the CPU and then executing a break. This ensures that the emulator will not hang after an invalid memory location has been accessed.

When set to OFF, the EL 1600 waits for the target to supply a DTACK signal, and may wait indefinitely.

The default for BTE is OFF. If no argument is given, the state of the switch is displayed.

Notes

BTE is only valid during RUN mode.

See FTO for the length of time until the timeout.

You can also enable or disable the emulator bus timeout in XICE.CFG with SW_BTE.
Example

BTE OFF

Related Commands

FTO
BUS — show state of processor bus signals

Works with

- EL 1600  
- CodeTAP

Syntax

BUS

Description

This command displays information about the processor, and the state of the processor’s pins. The signals are displayed as 0 or 1.

0  The signal is inactive.
1  The signal is active.

Notes

The information for internal emulator memory (EIM) is displayed for the 68302 only. See the EL 1600 Hardware Setup and Reference Manual for further information on EIM.

Example

> bus

PINS:

    BERR= 0
    VCC= 1
    TRST= 0
    PRST= 0
IPL0 = 0
IPL1 = 0
IPL2 = 0
BGT = 0
DBF = 0
HLT = 0
CLK = 1
8 Bit Mode = 0
EIM switch = 0xff0000

FSI count: 0
BUSTIME — capture bus timing in trace buffer

Works with

- EL 1600  
- CodeTAP

Syntax

BUSTIME [ON|OFF]

Description

This switch specifies whether bus timing information or interrupt level information should be recorded in trace. If BUSTIME is set to ON, bus timing information is recorded in the trace under the heading IPL. If BUSTIME is set to OFF, the current interrupt level information is recorded in trace rather than the bus timing information.

The default for BUSTIME is OFF. If no argument is given, the state of the switch is displayed.

Notes

You can also set BUSTIME in XICE.CFG with SW_BUSTIME:ON and SW_BUSTIME:OFF.

Example

BUSTIME ON
CAS — continuous address strobe

Works with

- EL 1600
- CodeTAP

Syntax

CAS [ON|OFF]

Description

This switch specifies whether the target sees address strobes while the emulator is paused. If CAS is set to ON, the target sees address strobes while the emulator is paused. If CAS is set to OFF, the target does not see address strobes when the emulator is paused.

The default for CAS is OFF. If no argument is given, the state of the switch is displayed.

Notes

You can also set CAS in XICE.CFG with SW_CAS:ON and SW_CAS:OFF.

Example

CAS OFF
CRC — calculate a CRC for a range of memory

Works with

- EL 1600
- CodeTAP

Syntax

`CRC address_range [/address_space]`

Description

This command performs a CRC over the specified range, where `range` is of the form `start..end`. `Address_space` specifies the memory space from which the target reads are to be performed. If no memory space is specified, it defaults to the memory read space specified by the command SPACE.

Notes

Memory space is processor-specific. The valid values for 68000 family processors are as follows:

<table>
<thead>
<tr>
<th>Address Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SD or SC5</td>
<td>Supervisor data space</td>
</tr>
<tr>
<td>SP or SC6</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>CPU or SC7</td>
<td>CPU space</td>
</tr>
</tbody>
</table>
Example

CRC 0..0xFFFF /SD 
Calculates a CRC in supervisor data space from 0 to 0xFFFF
DBP — disable bus error on peek/poke

Works with

- EL 1600
- CodeTAP

Syntax

```
DBP [ON|OFF]
```

Description

This command controls whether bus errors are reported on peeks and pokes. If DBP is set to ON, the bus error signals detected from the target system are displayed. If DBP is set to OFF, target system bus errors are ignored during peeks and pokes.

The default for DBP is ON. If no argument is given, the state of the switch is displayed.

Notes

You can also set DBP in XICE.CFG with SW_DBP:ON and SW_DBP:OFF.

Example

```
DBP ON
```
DIAG 0 — simple target ram test

Works with

- EL 1600
- CodeTAP

Syntax

```
DIAG 0, address_range[#count][;memory_space][/access_size]
```

Description

Target diagnostic 0 performs a simple target RAM test on the range of target memory you specify.

The optional parameter `#count` indicates the number of passes to make. The valid choices for `#count` are #0 and #1. #0 will cause the test to continue until you enter ctrl-C. #1 will cause one iteration of the test. If you do not set `#count`, it defaults to #1.

The optional parameter `;memory_space` specifies the memory space in which the test is performed. Memory space codes are the same as for the CRC command. If you do not set `;memory_space`, it defaults to the value set with the SPACE DIAG command.

Memory space is processor-specific. The valid values for 68000 family processors are as follows:

<table>
<thead>
<tr>
<th>Code Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
</tbody>
</table>
The optional parameter `/access_size` specifies the size in which memory is accessed. It may be a /1 (byte), /2 (word), or /4 (longword). If you do not set `/access_size`, it defaults to the value set with the SIZE DIAG command. For access size /4, the emulator actually uses two word accesses since the 68000 and 68302 processors cannot directly access memory using longwords.

**Example**

```
DIAG 0, 0..0x0fff #6
```
DIAG 1 — complex target ram test

Works with

■ EL 1600  □ CodeTAP

Syntax

DIAG 1, address_range[#count][;memory_space][/access_size]

Description

Target diagnostic 1 performs a complex target RAM test on the range of target memory you specify.

The optional parameter #count indicates the number of passes to make. The valid choices for #count are #0 and #1. #0 will cause the test to continue until you enter ^C. #1 will cause one iteration of the test. If you do not set #count, it defaults to 1.

The optional parameter ;memory_space specifies the memory space in which the test is performed. Memory space codes are the same as for the CRC command. If you do not set ;memory_space, it defaults to the value set with the SPACE DIAG command.

Memory space is processor-specific. The valid values for 68000 family processors are as follows:

<table>
<thead>
<tr>
<th>Code Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>Code Space</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SD or SC5</td>
<td>Supervisor data space</td>
</tr>
<tr>
<td>SP or SC6</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>CPU or SC7</td>
<td>CPU space</td>
</tr>
</tbody>
</table>

The optional parameter /access_size specifies the size in which memory is accessed. It may be a /1 (byte), /2 (word), or /4 (longword). If you do not set /access_size, it defaults to the value set with the SIZE DIAG command. For access size /4, the emulator actually uses two word accesses since the 68000 and 68302 processors cannot directly access memory using longwords.

**Example**

```plaintext
DIAG 1, 0..0xffff #0
```
DIAG 2 — continuous read from target memory

Works with

- EL 1600
- CodeTAP

Syntax

```plaintext
DIAG 2, address[; memory_space][/access_size]
```

Description

Target diagnostic 2 performs a continuous read from the target memory address you specify. This test continues until you enter ctrl-C.

The optional parameter `;memory_space` specifies the memory space in which the test is performed. Memory space codes are the same as for the CRC command. If you do not set `;memory_space`, it defaults to the value set with the SPACE DIAG command.

Memory space is processor-specific. The valid values for 68000 family processors are as follows:

<table>
<thead>
<tr>
<th>Code Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>Code Space</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>SD or SC5</td>
<td>Supervisor data space</td>
</tr>
<tr>
<td>SP or SC6</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>CPU or SC7</td>
<td>CPU space</td>
</tr>
</tbody>
</table>

The optional parameter `/access_size` specifies the size in which memory is accessed. It may be a /1 (byte), /2 (word), or /4 (longword). If you do not set `/access_size`, it defaults to the value set with the SIZE DIAG command. For access size /4, the emulator actually uses two word accesses since the 68000 and 68302 processors cannot directly access memory using longwords.

**Notes**

None

**Example**

```
DIAG 2, 0x0 /2
```
DIAG 3 — continuous write to target memory

Works with

■ EL 1600 □ CodeTAP

Syntax

DIAG 3, address=data[,memory_space][/access_size]

Description

Target diagnostic 3 performs a continuous write of data to the target memory address you specify. This test continues until you enter ctrl-C.

The optional parameter ;memory_space specifies the memory space in which the test is performed. Memory space codes are the same as for the CRC command. If you do not set ;memory_space, it defaults to the value set with the SPACE DIAG command.

Memory space is processor-specific. The valid values for 68000 family processors are as follows:

<table>
<thead>
<tr>
<th>Code Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>Code Space</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>SD or SC5</td>
<td>Supervisor data space</td>
</tr>
<tr>
<td>SP or SC6</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>CPU or SC7</td>
<td>CPU space</td>
</tr>
</tbody>
</table>

The optional parameter `/access_size` specifies the size in which memory is accessed. It may be a /1 (byte), /2 (word), or /4 (longword). If you do not set `/access_size`, it defaults to the value set with the SIZE DIAG command. For access size /4, the emulator actually uses two word accesses since the 68000 and 68302 processors cannot directly access memory using longwords.

**Example**

```
DIAG 3, 0x1000 = 0xcf2617 /2
```
DIAG 4 — write alternating pattern to target location

Works with

- EL 1600
- CodeTAP

Syntax

DIAG 4, address=data[:memory_space][/access_size]

Description

Target diagnostic 4 writes an alternating pattern of data to the target memory address you specify. It uses the data given as one pattern and the data given, inverted bit-wise, as the alternate pattern. This test continues until you enter ctrl-C.

The optional parameter ;memory_space specifies the memory space in which the test is performed. Memory space codes are the same as for the CRC command. If you do not set ;memory_space, it defaults to the value set with the SPACE DIAG command.

Memory space is processor-specific. The valid values for 68000 family processors are as follows:

<table>
<thead>
<tr>
<th>Code Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
</tbody>
</table>
The optional parameter /access_size specifies the size in which memory is accessed. It may be a /1 (byte), /2 (word), or /4 (longword). If you do not set /access_size, it defaults to the value set with the SIZE DIAG command. For access size /4, the emulator actually uses two word accesses since the 68000 and 68302 processors cannot directly access memory using longwords.

Example

```
DIAG 4, 0x10 = 0x5555 /2
```
DIAG 5 — write rotated to target memory

Works with

- EL 1600
- CodeTAP

Syntax

```
DIAG 5, address=data[:memory_space] [access_size]
```

Description

Target diagnostic 5 performs a continuous write of data to the target memory address you specify, and after each write the data value is rotated left by one bit. For example with data=01, one complete rotation is as follows:

```
01
02
04
08
10
20
40
80
01
```

This test continues until you enter ctrl-C.

The optional parameter ;memory_space specifies the memory space in which the test is performed. Memory space codes are the same as for the CRC command. If you do not set ;memory_space, it defaults to the value set with the SPACE DIAG command.
Memory space is processor-specific. The valid values for 68000 family processors are as follows:

<table>
<thead>
<tr>
<th>Code Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SD or SC5</td>
<td>Supervisor data space</td>
</tr>
<tr>
<td>SP or SC6</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>CPU or SC7</td>
<td>CPU space</td>
</tr>
</tbody>
</table>

The optional parameter `/access_size` specifies the size in which memory is accessed. It may be a /1 (byte), /2 (word), or /4 (longword). If you do not set `/access_size`, it defaults to the value set with the SIZE DIAG command. For access size /4, the emulator actually uses two word accesses since the 68000 and 68302 processors cannot directly access memory using longwords.

**Example**

```
DIAG 5, 0 = 0x1212 ;UD
```
DIAG 6 — write then read target memory

Works with

- EL 1600
- CodeTAP

Syntax

```plaintext
DIAG 6, address=data[;memory_space][/access_size]
```

Description

Target diagnostic 6 performs a continuous write of *data* to the target memory address you specify, and after each write then reads the same location. This test continues until you enter ctrl-C.

The optional parameter `;memory_space` specifies the memory space in which the test is performed. Memory space codes are the same as for the CRC command. If you do not set `;memory_space`, it defaults to the value set with the SPACE DIAG command.

Memory space is processor-specific. The valid values for 68000 family processors are as follows:

<table>
<thead>
<tr>
<th>Code Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>Code Space</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>SD or SC5</td>
<td>Supervisor data space</td>
</tr>
<tr>
<td>SP or SC6</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>CPU or SC7</td>
<td>CPU space</td>
</tr>
</tbody>
</table>

The optional parameter `/access_size` specifies the size in which memory is accessed. It may be a `/1` (byte), `/2` (word), or `/4` (longword). If you do not set `/access_size`, it defaults to the value set with the SIZE DIAG command. For access size `/4`, the emulator actually uses two word accesses since the 68000 and 68302 processors cannot directly access memory using longwords.

**Example**

```
DIAG 6,0x1000 = 0x24 /1
```
DIAG 7 — continuous read from target memory range

Works with

- EL 1600
- CodeTAP

Syntax

```
DIAG 7, address_range[;memory_space][/access_size]
```

Description

Target diagnostic 7 performs a continuous read from the target memory range you specify. This test continues until you enter ctrl-C.

The optional parameter `memory_space` specifies the memory space in which the test is performed. Memory space codes are the same as for the CRC command. If you do not set `memory_space`, it defaults to the value set with the SPACE DIAG command.

Memory space is processor-specific. The valid values for 68000 family processors are as follows:

<table>
<thead>
<tr>
<th>Code Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
</tbody>
</table>
### Code Space | Description
--- | ---
SD or SC5 | Supervisor data space
SP or SC6 | Supervisor program space
CPU or SC7 | CPU space

The optional parameter `/access_size` specifies the size in which memory is accessed. It may be a `/1` (byte), `/2` (word), or `/4` (longword). If you do not set `/access_size`, it defaults to the value set with the SIZE DIAG command. For access size `/4`, the emulator actually uses two word accesses since the 68000 and 68302 processors cannot directly access memory using longwords.

**Example**

```
DIAG 7, 0..0x1f /2
```
DIAG 8 — write incremental value to target memory

Works with

■ EL 1600  □ CodeTAP

Syntax

DIAG 8, address[;memory_space][/access_size]

Description

Target diagnostic 8 writes an incrementing value to the target memory address you specify. This test continues until you enter ctrl-C.

The optional parameter ;memory_space specifies the memory space in which the test is performed. Memory space codes are the same as for the CRC command. If you do not set ;memory_space, it defaults to the value set with the SPACE DIAG command.

Memory space is processor-specific. The valid values for 68000 family processors are as follows:

<table>
<thead>
<tr>
<th>Code Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
</tbody>
</table>
## Code Space Description

<table>
<thead>
<tr>
<th>Code Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD or SC5</td>
<td>Supervisor data space</td>
</tr>
<tr>
<td>SP or SC6</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>CPU or SC7</td>
<td>CPU space</td>
</tr>
</tbody>
</table>

The optional parameter `/access_size` specifies the size in which memory is accessed. It may be a `/1` (byte), `/2` (word), or `/4` (longword). If you do not set `/access_size`, it defaults to the value set with the SIZE DIAG command. For access size `/4`, the emulator actually uses two word accesses since the 68000 and 68302 processors cannot directly access memory using longwords.

### Example

```
DIAG 8, 0x1000 ;SD /1
```
DIAG 9 — continuous stream of reset pulses

Works with

- EL 1600
- CodeTAP

Syntax

DIAG 9

Description

Target diagnostic 9 sends a continuous stream of reset pulses to the target CPU. This test continues until you enter ctrl-C.

Example

DIAG 9
DNL — download hex file to target

Works with

- EL 1600
- CodeTAP

Syntax

DNL "filename" [,offset]

Description

DNL allows you to download a hex file from the host to the target in the format specified by the DNLFMT command. The contents of the file will be downloaded to the memory locations specified in the file. If you wish to download the file to memory locations different from those specified in the file, enter a value for offset. The offset value will be added to the address of each record to determine the actual download address. The default value for offset is 0.

The MAP, OVERLAY, SPACE, and SIZE commands affect how memory is accessed by DNL. Memory read-after-write verification is controlled by the setting of the VERIFY switch.

Notes

Quotation marks are optional if the file name consists of alphanumeric characters or a period. File names that contain a leading slash must be in double quotation marks (e.g., "/root"). File names that contain a leading backslash must be in single quotation marks (e.g., '\root').

Example

DNL my.file ,0x1000
DNLFMT — specify download format

Works with

- EL 1600  
- CodeTAP

Syntax

DNL format

Description

DNLFMT is used to specify the format for hex file downloads using the DNL command. Recognized formats are:

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEL</td>
<td>Intel hex format. Extended segment address records and extended linear address records are supported.</td>
</tr>
<tr>
<td>SREC</td>
<td>Motorola S1, S2, S3-records with Microtec extensions.</td>
</tr>
<tr>
<td>XTEK</td>
<td>Extended Tektronics hex format.</td>
</tr>
</tbody>
</table>

The default format of DNLFMT is SREC. The command XICEVARS displays the status of this variable as well as all the XICE variables.

Notes

Symbols are not supported for these formats.

Example

DNLFMT SREC

Related Commands

DNL, UPL, UPLFMT
**DNL_GAP — specify maximum bytes between blocks**

**Works with**
- EL 1600
- CodeTAP

**Syntax**

DNL_GAP [1-1024]

**Description**

DNL_GAP specifies the maximum number of bytes allowed between two cached download blocks before they are considered discontiguous. Download speed is faster for contiguous blocks than for discontiguous blocks. Specifying a larger number will improve download speed, but may cause some locations to be overwritten if there are discontiguous blocks that are smaller than the value of DNL_GAP. A lower value will avoid this. If no argument is given, the current value is displayed.

The default value of DNL_GAP is 1. The command XICEVARS displays the status of this variable as well as all the XICE variables.

**Example**

DNL_GAP 4

**Related commands**

DNL, DNLFMT
DOWN (DOW) — move current scope

Works with

- EL 1609
- CodeTAP

Syntax

```
DOWN [number_of_levels]
```

Description

The UP and DOWN commands allow you to move the current scope up or down the runtime stack. This is especially helpful when debugging recursive functions. It is not a good idea to go down farther than you have gone up.

Example

```
DOW 5
```

Related commands

UP
DRT — display raw trace

Works with

- EL 1600
- CodeTAP

Syntax

DRT [start|start..end]

Description

This command displays raw trace information. If you wish to limit the display, you may specify a range of bus cycles.

DRT by itself displays the last page of trace. DRT with a range displays trace for the specified range. DRT with a start number displays trace from the specified frame forward.

Notes

The columns DMA and IAC relate only to the 68302. See section 1 of this supplement for a description of the information in each of the columns in the trace display.
### Example

<table>
<thead>
<tr>
<th>FRAME</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>IPL</th>
<th>FCn</th>
<th>MEM</th>
<th>DMA IAC</th>
<th>FLAGS: X-break</th>
<th>B-BERR</th>
<th>M-illegal mem access</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>005732</td>
<td>22</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>005733</td>
<td>.2F</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>007FA4</td>
<td>00</td>
<td>0000</td>
<td>SD</td>
<td>WBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>007FA5</td>
<td>.00</td>
<td>0000</td>
<td>SD</td>
<td>WBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>007FA6</td>
<td>56</td>
<td>0000</td>
<td>SD</td>
<td>WBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>007FA7</td>
<td>.D0</td>
<td>0000</td>
<td>SD</td>
<td>WBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>005734</td>
<td>00</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>005735</td>
<td>.04</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>005736</td>
<td>10</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>005737</td>
<td>.39</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>007FA8</td>
<td>00</td>
<td>0000</td>
<td>SD</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>007FA9</td>
<td>.00</td>
<td>0000</td>
<td>SD</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>007FAA</td>
<td>00</td>
<td>0000</td>
<td>SD</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>007FAB</td>
<td>.7C</td>
<td>0000</td>
<td>SD</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>00573B</td>
<td>00</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>005739</td>
<td>.00</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>00573A</td>
<td>60</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>00573B</td>
<td>.4C</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>00573C</td>
<td>.02</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>00573D</td>
<td>.80</td>
<td>0000</td>
<td>SP</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>00604C</td>
<td>80</td>
<td>0000</td>
<td>SD</td>
<td>RBO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Related Commands

DT, DTB, DTF
DRTMR — enable dynamic refresh of memory (68302 only)

Works with

- EL 1600
- CodeTAP

Syntax

DRTMR [NONE | TMR1 | TMR2 | SCC1 | SCC2 | SCC3]

Description

This switch controls the dynamic refresh of memory. The following arguments are allowed for DRTMR:

- NONE: do not allow a DRAM refresh
- TMR1: use TMR1 to trigger a DRAM refresh
- TMR2: use TMR2 to trigger a DRAM refresh
- SCC1: use SCC1 to trigger a DRAM refresh
- SCC2: use SCC2 to trigger a DRAM refresh
- SCC3: use SCC3 to trigger a DRAM refresh

The default for DRTMR is NONE.

Notes

This command is not used by XICE for the 68000.

You can also set DRTMR in XICE.CFG with SW_DRTMR:ON or SW_DRTMR:OFF.

Example

DRTMR NONE
DRUN — start dynamic run mode

Works with

■ EL 1600  □ CodeTAP

Syntax

DRUN

Description

The DRUN (dynamic run) command executes the target program and continues execution until it is stopped by DSTOP, a breakpoint, an error, or a halt instruction. The purpose of this mode is to allow the user to interact with the emulator and debugger dynamically, while the emulator is running. In DRUN the user can examine and qualify trace, set and change events and breakpoints, examine and change memory and perform most other interactive functions with the emulator.

During DRUN the breakpoint and event systems are active but the emulator is not polled regularly for status. This can result in the emulator breaking execution with no notification to the user. Because most commands force polling of the emulator, unless there is no user interaction, the emulator status will be made known to XICE. When a break in execution is detected, the user will be notified and DRUN will be exited. The DUPATE command can be used to force regular polling of the emulator.

During DRUN the XICE version number on the XICE status line is replaced by the word DRUN.
Notes

When trace is requested from the emulator, the trace system is disabled during the time trace is uploaded to the host. This has an unavoidable side effect of also disabling the break system during the same period.

Example

DRUN

Related Commands

DSTOP, DUPDATE
DSTOP — stop dynamic run mode

Works with

- EL 1600
- CodeTAP

Syntax

DSTOP

Description

The DSTOP command stops the DRUN (dynamic run) command and breaks program execution.

Notes

None

Example

DSTOP

Related Commands

DRUN, DUPDATE
DT — display trace

Works with

- EL 1600
- CodeTAP

Syntax

DT [start..end]

Description

This command displays disassembled trace (bus cycles), showing either assembly instructions, source lines or both depending on the value you set for TRCMODE. You may specify the start..end range of bus cycles anywhere from 0 to 8K.

TRCMODE values are as follows:

- **ASM** causes an assembly instruction only display
- **SRC** causes a source line only display
- **BOTH** causes an interleaved high level source and assembly display

If no argument is given, DT shows the last instruction executed.

Notes

The emulator displays the message UNATTACHED BUS DATA when there is data on the bus that does not match up to an instruction. Unattached bus data may be caused by an external device putting data on the bus or, for CPUs that have cache, running with cache which allows data movements that do not match fetched instructions.
The disassemblers require a continuous trace stream to be able to disassemble correctly. For this reason, if you have PPT set to ON, or if you use the event system to qualify trace, DT will not be able to function correctly. Failures will range from incorrect information being displayed to crashing XICE.

Example

```
DT 10..50
```

Disassembles trace cycles 10 to 50.

Related commands

DTB, DTF, TRCMODE
DTB — display disassembled trace backwards

Works with

- EL 1600
- CodeTAP

Syntax

DTB

Description

This command displays disassembled trace backwards (away from cycle 0), one page at a time starting from the most recent cycle count. It shows either assembly instructions, source lines or both depending on the value you set for TRCMODE.

TRCMODE values are as follows:

- ASM causes an assembly instruction only display
- SRC causes a source line only display
- BOTH causes an interleaved high level source and assembly display

Notes

The emulator displays the message UNATTACHED BUS DATA when there is data on the bus that does not match up to an instruction. Unattached bus data may be caused by an external device putting data on the bus or, for CPUs that have cache, running with cache which allows data movements that do not match fetched instructions.

The disassembler requires a continuous stream of at least 128 lines of raw trace for a good disassembly. For this reason, if you have PPT set to ON, if you use the event system to qualify trace, or if you attempt using DTB with fewer than 128 lines of
raw trace you will not be able to accurately disassemble the trace. The results will range from an error message being displayed to incorrect information being displayed to XICE crashing.

Example

```
[DT]  
} } }  WORKING ...} ...]

0048 004050 4EB30000 JSR main
        408EQ
0045 main:
   >> initial();            /* Initialize Variables */
0045 00408E 4EB0018 JSR initial
0041 initial:
   >> pattern = ONE_ON;
0041 0040A8 37FO00FZ MOVE.L #SPZ, pattern
       00000F44EQ
   >> speed = MEDIUM;
0035 0040B0 23F00000 MOVE.L #5000 F4F4, speed
       014000005E5P
   >> direct = left;
0029 0040BA 7001 MOVEQ.L $D1.D0L
0026 0040BC 23C00000 MOVE.L D0, direct
       5FC0EQ
>>}]
0023 0040C2 4275 RTS
   >> initial();            /* Initialize Variables */
0017 004092 4271 NOP
   >> step();                /* Single Step Loop */
0016 004094 4EB002E JSR step
0014 step:
   >>
0014 0040C4 2702 MOVE.L D2.-(A7)EQ
   >> for (loops = 5; loops != 0; loops--)/ repeat output 5 times */
0011 0040C8 7405 MOVEQ.L #S5.D2EQ
   >> outled(0xFF0F);        /* 0111 1111 */
0010 0040CE 487800FF PEA.L $0000FF
0006 0040CC 4EB50000 JSR outled
        407c
```

Related Commands

DT, DTF, TRCMODE
DTF — display disassembled trace forward

Works with

- EL 1600
- CodeTAP

Syntax

DTF

Description

This command displays disassembled trace forward (toward cycle 0), one page at a time starting from the most recent cycle count. It shows either assembly instructions, source lines or both depending on the value you set for TRCMODE.

TRCMODE values are as follows:

- ASM causes an assembly instruction only display
- SRC causes a source line only display
- BOTH causes an interleaved high level source and assembly display

Notes

The emulator displays the message UNATTACHED BUS DATA when there is data on the bus that does not match up to an instruction. Unattached bus data may be caused by an external device putting data on the bus or, for CPUs that have cache, running with cache which allows data movements that do not match fetched instructions.
The disassembler requires a continuous stream of at least 128 lines of raw trace for a good disassembly. For this reason, if you have PPT set to ON, if you use the event system to qualify trace, or if you attempt using DTF with fewer than 128 lines of raw trace you will not be able to accurately disassemble the trace. The results will range from an error message being displayed to incorrect information being displayed to XICE crashing.

Example

```
\>`
\` WORKING \`
0054 START: 004044 2E7C0000 MOVE.L #$00008000, A\+$00008000, A\+$
0050 00404A 2C7C0000 MOVE.L #$00000000, A\+$00000000, A\+$
```

Related commands

DT, DTB, TRCMODE
DUPDATE — specify polling frequency in dynamic run mode

Works with

- EL 1600
- CodeTAP

Syntax

DUPDATE [1...]

Description

The DUPDATE command allows you to specify how frequently the emulator is polled during DRUN (dynamic run) mode. The value entered represents the number of polls per minute. Whenever the emulator is polled the screen viewports are updated and the user is notified if emulation has broken.

DUPDATE is entered while in DRUN mode; in DUPDATE mode, commands from the user are no longer accepted. To exit DUPDATE and return to DRUN, enter Ctrl-C.

If no argument is given, DUPDATE defaults to 20. Above 100 polls per minute, increasing the polling rate will have no increasing effect.

Notes

None

Example

DUPDATE 30

Related Commands

DRUN, DSTOP
EMUVARS — display emulator variable values

Works with

- EL 1600
- CodeTAP

Syntax

EMUVARS

Description

This command displays the current values and descriptions for all the emulator softswitch variables.

Example

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BTE</td>
<td>OFF</td>
<td>Enable (ON)* vs. disable (OFF) bus timeout</td>
</tr>
<tr>
<td>BUSTIME</td>
<td>OFF</td>
<td>Capture bus timing (ON) vs. interrupt level (OFF)* in trace</td>
</tr>
<tr>
<td>CAS</td>
<td>OFF</td>
<td>Address strobe active during run (OFF)* vs. run and pause (ON)</td>
</tr>
<tr>
<td>DBP</td>
<td>ON</td>
<td>Enable (ON)* vs. disable (OFF) bus error detect on peek/poke</td>
</tr>
<tr>
<td>DRTMR</td>
<td>NONE</td>
<td>Use TMR1, TMR2, SCC1, SCC2, SCC3. NONE* to trigger DRAM refresh</td>
</tr>
<tr>
<td>EXVEC</td>
<td>1S</td>
<td>Soft breakpoint execution trap number (0-15*)</td>
</tr>
<tr>
<td>FAST</td>
<td>ON</td>
<td>Enable on emulation (ON)* vs. disable (OFF) fast interrupts</td>
</tr>
<tr>
<td>FRZ</td>
<td>OFF</td>
<td>Assert (ON) vs. do not assert (OFF)* FRZ while paused</td>
</tr>
<tr>
<td>FTO</td>
<td>OFF</td>
<td>Enable (ON) vs. disable (OFF)* fast bus timeout</td>
</tr>
<tr>
<td>MWARN</td>
<td>ON</td>
<td>Warn user (ON)* vs. ignore (OFF) out of range memory accesses</td>
</tr>
<tr>
<td>OVE</td>
<td>0x66</td>
<td>Memory spaces overlay will respond to (0-0xFF). Default: 0x66</td>
</tr>
<tr>
<td>OVS</td>
<td>1</td>
<td>Overlay speed (0-7). Default: 0</td>
</tr>
<tr>
<td>PPT</td>
<td>OFF</td>
<td>Enable (ON) vs. disable (OFF)* tracing of peek/poke cycles</td>
</tr>
<tr>
<td>RFS</td>
<td>OFF</td>
<td>Enable (ON) vs. disable (OFF)* software refresh</td>
</tr>
<tr>
<td>RFSADR</td>
<td>0</td>
<td>Software refresh address. Default: 0</td>
</tr>
<tr>
<td>RFSASP</td>
<td>5</td>
<td>Address space for software refresh (0-7). Default: 5</td>
</tr>
<tr>
<td>RFSMSK</td>
<td>0</td>
<td>Don't care mask for software refresh. Default: 0</td>
</tr>
<tr>
<td>RIRR</td>
<td>ON</td>
<td>Restore (ON)* vs. do not restore (OFF) registers on reset</td>
</tr>
<tr>
<td>SCRATCH</td>
<td>0xffff0</td>
<td>Start address of breakpoint scratch area. Default: 0</td>
</tr>
<tr>
<td>SIA</td>
<td>0</td>
<td>Special interrupt vector address. Default: 0</td>
</tr>
<tr>
<td>SLO</td>
<td>OFF</td>
<td>Wait then enable (ON) vs. disable (OFF)* slow interrupts</td>
</tr>
<tr>
<td>TAD</td>
<td>OFF</td>
<td>Tri-state (ON) vs. do not tri-state (OFF)* address bus in pause</td>
</tr>
<tr>
<td>TCEBRK</td>
<td>OFF</td>
<td>Trace (ON) vs. do not trace (OFF)* execution break cycles</td>
</tr>
<tr>
<td>TED</td>
<td>OFF</td>
<td>Enable (ON) vs. disable (OFF)* trace/overlay for external DMA</td>
</tr>
<tr>
<td>TID</td>
<td>OFF</td>
<td>Enable (ON) vs. disable (OFF)* trace/overlay for internal DMA</td>
</tr>
<tr>
<td>UIR</td>
<td>ON</td>
<td>Enable (ON)* vs. disable (OFF) auto-update of chip select reg</td>
</tr>
</tbody>
</table>

Related commands

BUS, EVT vars, MEM vars, XICE vars

S2-70

XICE Supplement for 68000/68HC000/68EC000 and 68302
EV — define an event

Works with

- EL 1600
- CodeTAP

Syntax

\[ EV(n) = \{ \text{event-definition} | \text{CLEAR} \} \]

Description

The EV command supports the EL 1600 event system capability. EV sets up an event definition and TRIG defines the action(s) to take place, once the trigger is armed, each time the event definition is met. For example,

\[
\text{trig}(5) = \text{ev}(1), \text{break}
\]

Sets trigger 5 to cause a break when event 1 is true.

\[
\text{trig}(5) = \text{arm}
\]

Arms trigger 5.

An event name is shown as ev\((n)\) where \(n\) is the number identifying this event (the curly braces are required punctuation). The event number \(n\) must be between 1 and 32.

An event definition is the specification of a possible state of the trace frame (the address, data, and status buses) along with the state of other event resources such as counters, during that trace frame’s bus cycle. An event is true when all of the terms within the event are true at the same time (i.e., the same single bus cycle).

The following general rules relate to setting up the event definition:

- The logical operators for equality (==), or inequality (!=) are used to set the values. E.g., ev\((1) = \text{stat}==\text{rd}\)
Event terms can be used only once in any one event definition.

Each testable condition must be separated from the next by a comma.
E.g., ev(1) = addr==0x1, data==0x2, stat==word

Addresses can be specified as ranges that are denoted by (..).
E.g., ev(1) = addr==0x0000..0xffff. Note, however, that you cannot have two comparators of the same type in a single event statement.
E.g., ev(1) = addr==0x1, addr==0x2 will not work.

Don't care masks can be used to exclude parts of data bus information.
E.g., ev(1) = data==0x0034 &=0x00ff defines an event that would be valid whenever the 8 least significant bits of the data bus are 0x34.

The counters start at 0 each time you GO. They require specific values. You may not use ranges or don't care masks with the counter.

LSA bits may be set using don't care masks.
E.g., ev(31) = LSA==0x2&=0x3

The information that follows lists the elements available for setting up an event definition for the 68000 and 68302.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>The value that appears on the address bus.</td>
</tr>
<tr>
<td>COUNT</td>
<td>The counter value.</td>
</tr>
<tr>
<td>DATA</td>
<td>The value that appears on the data bus.</td>
</tr>
<tr>
<td>STATUS</td>
<td>The type of bus activity (e.g. instruction fetch, read, write, interrupt acknowledge).</td>
</tr>
<tr>
<td>LSA</td>
<td>The value of the LSA bits.</td>
</tr>
</tbody>
</table>
The information that follows lists the valid 68000 and 68302 STATUS mnemonics that can be used in an event definition either in their positive form as listed below or in their negative form by prepending a NOT_.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERR</td>
<td>Bus error</td>
</tr>
<tr>
<td>BRK</td>
<td>Break signal asserted. This status is useful for determining the skid, i.e., the number of bus cycles between the time the break occurs and emulation stops.</td>
</tr>
<tr>
<td>BYTE</td>
<td>Byte access</td>
</tr>
<tr>
<td>DMA</td>
<td>Pod DMA signal state (68302 only)</td>
</tr>
<tr>
<td>IAC</td>
<td>CPU IAC pin state (68302 only)</td>
</tr>
<tr>
<td>OVL</td>
<td>Overlay access</td>
</tr>
<tr>
<td>RD</td>
<td>CPU read</td>
</tr>
<tr>
<td>TAR</td>
<td>Target access</td>
</tr>
<tr>
<td>VIO</td>
<td>Violation error</td>
</tr>
<tr>
<td>WORD</td>
<td>Word access</td>
</tr>
<tr>
<td>WR</td>
<td>CPU write</td>
</tr>
</tbody>
</table>
The information that follows lists additional valid 68000 and 68302 STATUS mnemonics that can be used in an event definition. However, these mnemonics may only be used as shown below, in other words, unlike the list of mnemonics above, they do not have a negative form.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Access to CPU space</td>
</tr>
<tr>
<td>INTR0</td>
<td>An interrupt 0 is pending</td>
</tr>
<tr>
<td>INTR1</td>
<td>IP0-IP2 is set to 1 (active low)</td>
</tr>
<tr>
<td>INTR2</td>
<td>IP0-IP2 is set to 2 (active low)</td>
</tr>
<tr>
<td>INTR3</td>
<td>IP0-IP2 is set to 3 (active low)</td>
</tr>
<tr>
<td>INTR4</td>
<td>IP0-IP2 is set to 4 (active low)</td>
</tr>
<tr>
<td>INTR5</td>
<td>IP0-IP2 is set to 5 (active low)</td>
</tr>
<tr>
<td>INTR6</td>
<td>IP0-IP2 is set to 6 (active low)</td>
</tr>
<tr>
<td>INTR7</td>
<td>IP0-IP2 is set to 7 (active low)</td>
</tr>
<tr>
<td>SD</td>
<td>Access to supervisory data space</td>
</tr>
<tr>
<td>SP</td>
<td>Access to supervisory program space</td>
</tr>
<tr>
<td>UD</td>
<td>Access to user data space</td>
</tr>
<tr>
<td>UP</td>
<td>Access to user program space</td>
</tr>
</tbody>
</table>

**Examples:**

\[ ev(1) = \text{addr==0x13400..0x134FF, stat==up|rd} \]

In the example above, the event expression \( ev(1) \) is true if any address in the 256 byte block 0x13400..0x134FF is read in user fetches.

\[ ev(1) = \text{addr==0x13400, count==4} \]
The expression \( ev(1) \) is true for all accesses of address 0x13400 when the counter has reached the count of 4.

\[ ev(1) = \text{clear} \]

Clears the event definition for \( ev(1) \) for reuse.

**Related commands**

EVTMODE, EVTGRP, EVTCLR, EVTARM, GROUP, TRIG
EVTARM — enable or disable automatic trigger arming

Works with

- EL 1600
- CodeTAP

Syntax

EVTARM [ON|OFF]

Description

This switch specifies whether triggers are automatically armed when they are defined using the TRIG command. If EVTARM is set to ON, triggers are automatically armed when defined. If EVTARM is set to OFF, triggers are not automatically armed when defined.

The default is ON. If no argument is given, the state of the switch is displayed.

Notes

Triggers are only active when they have been armed.

You can also set EVTARM in XICE.CFG with SW_EVTARM:ON or SW_EVTARM:OFF.

Example

EVTARM ON

Related Commands

EV, EVTMODE, EVTCLR, TRIG, XICEVARS
## EVTCLR — clear event system

### Works with

- EL 1600
- CodeTAP

### Syntax

\[
\text{EVTCLR}
\]

### Description

This command clears all events and triggers set up in the event system and resets the event state variables to their initial values.

### Example

\[
\text{EVTCLR}
\]

### Related Commands

EV, EVTMODE, EVTARM, TRIG, XICEVARS
EVTGRP — specify event group

Works with

- EL 1600
- CodeTAP

Syntax

`EVTGRP [1|2|3|4]`

Description

This command specifies the event group for an event when arming triggers. Whenever a trigger is armed, either automatically (when EVTARM is set to ON) or explicitly with a TRIG command, it is armed in a particular event group. This group is specified by the EVTGRP variable.

Event group 1 is the default. If no argument is given, the state of the switch is displayed.

The emulator allows four event groups. For any one group there may not be more than the following comparators:

- 2 address comparators (specifying an address range counts as a single address comparator)
- 2 data comparators
- 2 status comparators
- 1 LSA comparator
- 1 counter

Notes

You can also set EVTGRP in XICE.CFG with SW_EVTGRP:1, SW_EVTGRP:2, SW_EVTGRP:3, or SW_EVTGRP:4.

Example

`EVTGRP 2`
Related commands

GROUP, EV, EVTCLR, EVTARM, TRIG, EVTVARS
EVT VARS — display internal debugger variable values

Works with

- EL 1600
- CodeTAP

Syntax

EVT VARS

Description

This command displays the current value and description for the emulator event state variable GROUP. It indicates the event group that the emulator is currently in and is a read-only value. It cannot be changed.

Example

```
  EVT VARS
              WORKING ... ... 
  EVENT STATE VARIABLES
  GROUP   1  Event group that the emulator is currently in. <READ-ONLY>
```

Related Commands

EMUVARS, MEMVARS, XICEVARS
EXPLAIN — explain error message

Works with

- EL 1600
- CodeTAP

Syntax

EXPLAIN

Description

This command provides additional information about the last emulator-related error message reported.

Notes

This command only supports emulator-related error messages. If no emulator-related error messages have been generated, EXPLAIN states that the error is not emulator related.

Example

EXPLAIN
# EXVEC — software breakpoint execution trap number

## Works with

| EL 1600 | CodeTAP |

## Syntax

```
EXVEC [number]
```

## Description

The EXVEC switch specifies the software execution breakpoint trap number, where `number` may be from 0 to 15. The number that you enter for EXVEC should be the same as the trap number in your Vector Table.

The default for EXVEC is 15. If no argument is given, EXVEC shows the current setting for the switch.

## Notes

The trap number that you specify for EXVEC must be dedicated to XICE exclusively. This is because XICE uses that trap to implement software breakpoints. If your program also uses that trap, XICE will report a spurious break every time your program executes that trap.

XICE sets the vector for the specified trap to point to code that it has placed in the scratch area to handle software breakpoints. The target system may not change the vector of the specified trap once XICE has initialized it, nor may the target system modify the code XICE has placed in the scratch area.

To install a software breakpoint, XICE must be able to modify the opcode at the desired break address. If that address is in ROM, you will have to map that section of code to emulator...
overlay RAM. You may map the overlay in read-only mode, which prevents the target system from writing into the area but still allows XICE to modify the opcode.

You can also set EXVEC in XICE.CFG with SW_EXVEC: *number*.

**Example**

```
EXVEC 15
```

**Related commands**

BREAKINSTRTUTION, GOSTEP
FAST — fast interrupt emulation control

Works with

- EL 1600
- CodeTAP

Syntax

FAST [OFF|ON]

Description

The FAST switch controls the fast interrupt enable. When FAST is set to ON, interrupts are enabled immediately upon entering RUN mode. When FAST is set to OFF, interrupts are disabled.

The default is OFF. If no argument is given, FAST shows the current setting for the switch.

Notes

You can also set FAST in XICE.CFG with SW_FAST:ON or SW_FAST:OFF.

If you set both FAST and SLO to ON, FAST has precedence over SLO. The following table shows the results for the possible switch setting combinations for FAST and SLO. This table applies to target-generated interrupts passed to the target processor when the emulator is running.
<table>
<thead>
<tr>
<th>SLO</th>
<th>FAST</th>
<th>Result While in RUN Mode</th>
<th>Result While in PAUSE Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>ON</td>
<td>Interrupts immediately enabled.</td>
<td>Interrupts immediately enabled upon return to RUN mode.</td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td>Interrupts enabled after approximately 160 clock cycles.</td>
<td>Interrupts enabled after approximately 160 clock cycles after return to RUN mode.</td>
</tr>
<tr>
<td>OFF</td>
<td>ON</td>
<td>Interrupts immediately enabled.</td>
<td>Interrupts immediately enabled upon return to RUN mode.</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>Interrupts generated by the target system will be inhibited from reaching the emulator.</td>
<td>Interrupts generated by the target system will be inhibited from reaching the emulator.</td>
</tr>
</tbody>
</table>

**Example**

```
FAST ON
```

**Related commands**

```
SLO
```
FRZ — freeze peripheral activity (68302 only)

Works with

- EL 1600
- CodeTAP

Syntax

FRZ [ON|OFF]

Description

The FRZ switch enables or disables peripheral activity during PAUSE mode. When FRZ is set to ON, the FRZ pin is asserted in PAUSE mode, which disables any peripheral activity. When FRZ is set to OFF, peripheral activity is not blocked during PAUSE.

The default is OFF, which allows peripheral activity during PAUSE mode. If no argument is given, FRZ shows the current setting for the switch.

Notes

You can also set FRZ in XICE.CFG with SW_FRZ:ON or SW_FRZ:OFF.

This command is not used by XICE for the 68000.

Example

FRZ ON
**FTO — Fast Bus Timeout**

**Works with**

- EL 1600
- CodeTAP

**Syntax**

```
FTO [ON|OFF]
```

**Description**

The FTO switch controls the length of time for the bus timeout. If FTO is set to ON, a bus timeout occurs in 112 clock cycles. If FTO is set to OFF, a bus timeout requires 28,672 clock cycles, which is approximately 2 milliseconds.

The default for FTO is OFF. If no argument is given, FTO shows the current setting for the switch.

**Notes**

If BTE is set to OFF, this switch has no effect regardless of its setting.

You can also set FTO in XICE.CFG with SW_FTO:ON or SW_FTO:OFF.

**Example**

```
FTO OFF
```

**Related Commands**

- BTE
GROUP — display active event group

Works with

- EL 1600
- CodeTAP

Syntax

GROUP

Description

This command displays which of the four event groups was active at the last refresh or break.

Example

> GROUP
  Current setting is 2

Related Commands

EVTGRP, EV, EVTVARS
HWCONFIG — display hardware name and version

Works with

- EL 1600
- CodeTAP

Syntax

HWCONFIG

Description

This command displays the name and version of all hardware and software being used by the emulator.

Example

HWCONFIG

Current emulator configuration is:
- EL1600 Ethernet controller, version 1.01
- EL1600 1M Overlay, version 0.01
- EL1600 Dynamic T & B Board, version 0.01
- EL1600 68000 SCSI Shell(00), version 1.02

Related Commands

BUS
MEMVARS — display memory access variable values

Works with

- EL 1600
- CodeTAP

Syntax

MEMVARS

Description

This command displays the current values and descriptions for all the memory access variables.

Notes

The possible values for SPACE, SIZE, and OVERLAY are provided in the descriptions of each of these commands.

Example

```
memvars
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Space</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE:</td>
<td>SP</td>
<td>1</td>
</tr>
<tr>
<td>COMP1:</td>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>COMP2:</td>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>COPYFROM:</td>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>COPYTO:</td>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>DIAG:</td>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>FILL:</td>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>READ:</td>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>SEARCH:</td>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>STACK:</td>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>TEST:</td>
<td>SD</td>
<td>1</td>
</tr>
<tr>
<td>WRITE:</td>
<td>SD</td>
<td>2</td>
</tr>
</tbody>
</table>

Related commands

EMUVARS, EVTVARS, XICEVARS
MWARN — control address out-of-range warnings

Works with

- EL 1600
- CodeTAP

Syntax

MWARN [ON|OFF]

Description

This switch is used to protect your target hardware from unwanted accesses that may be used by the emulator during target writes due to the inability of the 68000 and 68302 to access target memory with byte accesses. For example, if you request a word write to an odd address such as 0x1, the emulator will read the word at 0x0, OR in the new data value for address 0x1 and write it back. Then it will read the word at 0x2, or in the data value at 0x2, and write it back. If you wish to be informed when these types of accesses occur at the upper and lower boundaries, set this switch ON.

The default for MWARN is ON. If no argument is given, MWARN shows the current setting for the switch.

Notes

You can also set MWARN in XICE.CFG with SW_MWARN:ON or SW_MWARN:OFF.

Example

MWARN ON
NETERR — specify timeout warning delay

Works with

- EL 1600
- CodeTAP

Syntax

NETERR [seconds]

Description

NETERR is the approximate amount of time, in seconds, that XICE waits for a response from the emulator before issuing a warning message after a code packet has been sent during download.

Example

NETERR 90 delays 90 seconds before issuing a warning message.

See also

- NETFAIL
NETFAIL — specify download abort timeout

Works with

- EL 1600
- CodeTAP

Syntax

NETFAIL [seconds]

Description

NETFAIL is the approximate amount of time, in seconds, that XICE waits for a response from the emulator before generating a timeout error and aborting the download process after a NETERR warning message has been displayed.

Example

NETFAIL 90  delays 90 seconds before aborting downloads

See also

NETERR
NULL_TGT — enable null target mode (68000/HC000/EC000)

Works with

- EL 1600
- CodeTAP

Syntax

NULL_TGT [ON|OFF|AUTO]

Description

This switch enables and disables null target mode. In null target mode probe tip signals to the target are disconnected and an internal clock is used. This mode allows you to operate the emulator without a target. The arguments cause the following actions:

- **ON**: Enables null target mode.
- **OFF**: Disables null target mode.
- **AUTO**: Null target mode is selected automatically when emulator detects absence of target power.

The default for NULL_TGT is AUTO. If no argument is given, NULL_TGT shows the current setting for the switch.

Notes

Null target mode is available only for 68000/HC000/EC000 emulators which use a probe tip configuration, with no probe module. It is not available for the 68302 or for 68000's which use a probe tip/probe module configuration. See your EL 1600 Hardware Setup and Reference Guide to identify the configuration of your 68000.
If you invoke XICE with no target connected to the emulator the SW_NULL_TGT switch in the XICE.CFG should be set to ON or AUTO. XICE invocation will fail if you have no target or no target power and are not in null target mode.

When running code in null target mode, the overlay memory board must be set up to return DTACK. Use OVS to specify this operation.

You can also set NULL_TGT in XICE.CFG with SW_NULL_TGT:AUTO or SW_NULL_TGT:ON or SW_NULL_TGT:OFF.

Example

```
NULL_TGT ON
```

Related Commands

OVS
OVE — overlay memory spaces

Works with

- EL 1600  - CodeTAP

Syntax

OVE [0..0xFF]

Description

This switch specifies which memory spaces overlay responds to. Overlay can respond to multiple spaces. To specify multiple spaces, OR the masks given below to create a number that is between 0 and 0xFF.

Memory address space is processor-specific. The valid values for the 68000 and 68302 processors are as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Address Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>0x02</td>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>0x04</td>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>0x08</td>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>0x10</td>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>0x20</td>
<td>SD or SC5</td>
<td>Supervisor data space</td>
</tr>
<tr>
<td>0x40</td>
<td>SP or SC6</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>0x80</td>
<td>CPU or SC7</td>
<td>CPU space</td>
</tr>
</tbody>
</table>
The default for OVE is 0x66, which represents the address spaces for Supervisor Program Space (0x40), Supervisor Data Space (0x20), User Data Space (0x02), and User Program Space (0x04) all ORed together (0x66).

If no argument is given, OVE shows the current setting for the switch.

**Notes**

You can also set OVE in XICE.CFG with SW_OVE:value.

**Example**

OVE 0x66
OVS — set emulator overlay speed

Works with

- EL 1600
- CodeTAP

Syntax

OVS [number]

Description

This switch specifies the number of wait states (from 0 to 7) to be inserted before the overlay memory supplies a DTACK signal to terminate the cycle.

The valid OVS settings are as follows:

- 0  DTACK supplied by target memory
- 1  No delay, address strobe returned to the processor as DTACK (DSACK)
- 2  +1 cycle delay
- 3  +2 cycles delay
- 4  +3 cycles delay
- 5  +4 cycles delay
- 6  +5 cycles delay
- 7  +6 cycles delay

The default for OVS is 0. If no argument is given, the state of the switch is displayed.

Notes

The CPU accepts the first DTACK it receives, either from the target or from the internally generated DTACK using the OVS setting.
The 68000 and the 68302 may not be able to run out of overlay memory without wait states since overlay RAM may not respond as quickly as target RAM. If you observe erratic overlay operation, set OVS to 2 or more to insert a delay.

The 20 MHz 68302 requires that OVS be set at 2 or greater. Otherwise, XICE may report illegal switches on initialization.

If you are using the emulator with a null target, you must set OVS to a non-zero value for proper operation with overlay memory.

You can also set the emulator overlay speed in XICE.CFG with SW_OVS:n, for example SW_OVS:1.

Example

```
OVS 1
```

Related Commands

MEMVARS
PERFACT — enable performance analysis data gathering

Works with

- EL 1600
- CodeTAP

Syntax

PERFACT [STATISTICAL|OFF]

Description

PERFACT turns statistical performance analysis data gathering on and off. If PERFACT is on, the emulator will periodically upload data during run, and process the data for display. The interval between data uploads is determined by the PERFINT switch.

Notes

For more information refer to Chapter 5, Using Performance Analysis.

Example

PERFACT STATISTICAL

Related Commands

PERFINT
PERFCLR — remove performance analysis data

Works with

- EL 1600
- CodeTAP

Syntax

PERFCLR

Description

PERFCLR purges all accumulated performance analysis data from the system. The event system and address exclusion setups are not disturbed.

Accumulated performance analysis data is automatically cleared when a LOAD is executed.

Notes

For more information refer to Chapter 5, Using Performance Analysis.

Example

PERFCLR

Related Commands
PERFDATA — display performance analysis symbol data

Works with

■ EL 1600  □ CodeTAP

Syntax

PERFDATA [symbol\string]

Description

PERFDATA displays the address range and number of samples for symbols which appear in the accumulated performance analysis data.

Note that the address range is derived from the trace data and is typically a subrange of the actual addresses for that function (as shown by PRINTSYMBOL, for example).

Notes

For more information refer to Chapter 5, Using Performance Analysis.

Example

PERFDATA main
main:
  Address range: 0x0000566E..0x00005700
  Hits: 5789

Related Commands
PERFDEPTH — maximum number of lines of PA output

Works with

□ EL 1600  □ CodeTAP

Syntax

PERFDEPTH [0...]

Description

When running Performance Analysis on a large program, the high number of symbols encountered can cause the PA display to exceed the depth of the screen. Quite often, many of the last symbols displayed are of little interest because they did not occur often. PERFDEPTH can be used to limit the display to only the more frequently encountered symbols.

The default is zero; all available lines display. Setting PERFDEPTH to any non-zero number limits the display to the specified number of lines.

Notes

You can also set the symbol display line limit in xice.cfg using the softswitch SW_PERFDEPTH:number.

For more information refer to Chapter 5, Using Performance Analysis.

Example

PERFDEPTH 20
PERFDISP — display performance analysis information

Works with

- EL 1600
- CodeTAP

Syntax

PERFDISP

Description

PERFDISP displays performance analysis information. The format of the display is specified by PERFFORMAT. If it is set to display all data, the display is in the format: symbol: percentage of samples in function: number of samples in function: histogram, where symbol is a function or subroutine in your program. Any of percentage, samples, or histogram may be left out, although at least one will always be present. For more information refer to Chapter 5, Using Performance Analysis.

Example

PERFDISP

Hits used: 40928 (40928 total, 0 excluded)

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>PERCENT</th>
<th>HITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>func9:</td>
<td>30.8</td>
<td>12605</td>
</tr>
<tr>
<td>func8:</td>
<td>17.1</td>
<td>6999</td>
</tr>
<tr>
<td>main:</td>
<td>16.6</td>
<td>6794</td>
</tr>
<tr>
<td>func7:</td>
<td>12.4</td>
<td>5075</td>
</tr>
<tr>
<td>func6:</td>
<td>11.1</td>
<td>4543</td>
</tr>
<tr>
<td>func5:</td>
<td>6.2</td>
<td>2538</td>
</tr>
<tr>
<td>func4:</td>
<td>3.1</td>
<td>1269</td>
</tr>
<tr>
<td>func3:</td>
<td>1.6</td>
<td>655</td>
</tr>
<tr>
<td>func2:</td>
<td>0.8</td>
<td>327</td>
</tr>
<tr>
<td>func1:</td>
<td>0.3</td>
<td>123</td>
</tr>
</tbody>
</table>

XICE Supplement for 68000/68HC000/68EC000 and 68302
Related commands

PERFMODE, PERFFORMAT
PERFEX — exclude addresses from performance analysis

Works with

- EL 1600  □ CodeTAP

Syntax

PERFEX [address|address range|symbol[#distance]]

Description

PERFEX excludes certain addresses from the performance analysis data. If an address or address range is specified, those addresses are excluded. If a symbol name is given, PERFEX searches forward for the next symbol and excludes up to that symbol. This can be used to exclude a function. Normally the limit for this search is given by PERFTOL, but it may be overridden with #distance. Exclusion ranges are automatically merged when they overlap or are contiguous.

If no arguments are given, PERFEX displays all exclusions in effect and the names of symbols within those exclusions.

For recommendations on using PERFEX effectively, see the chapter on using performance analysis in this supplement.

Notes

For more information refer to Chapter 5, Using Performance Analysis.
Example

PERFEX 0x5800..0x5900

PERFEX wait
Symbol getcom found at 0x00005704. Excluding from 0x0000566E to 0x00005703.

PERFEX
Current address exclusion ranges are:
   0x0000566E..0x00005703 wait
   0x00005800..0x00005900 sort, shuffle

Related Commands

PERFEXCLR, PERFTOL
PERFEXCLR — clear performance analysis exclusions

Works with

■ EL 1600   □ CodeTAP

Syntax

PERFEXCLR [address|address range|symbol[#distance]]

Description

PERFEXCLR clears address range exclusions set with PERFEX. If an address or address range is specified, any exclusion of those addresses is cleared. If a symbol name is given, PERFEXCLR searches forward for the next symbol and clears any exclusions up to that symbol. This can be used to clear the exclusion of a function. Normally the limit for this search is given by PERFTOL, but it may be overridden with #distance.

If no arguments are given, PERFEXCLR clears all exclusions in effect. All address range exclusions are automatically cleared when a LOAD is executed.

For more information refer to Chapter 5, Using Performance Analysis.

Example

PERFEXCLR 0x5800..0x5900

PERFEXCLR wait
Symbol getcom found at 0x00005704.Clearing from 0x0000566E to 0x00005703.

Related Commands

PERFEX, PERFTOL
PERFFORMAT — format of performance analysis display

Works with

- EL 1600
- CodeTAP

Syntax

PERFFORMAT [STANDARD|PERCENT|HITS|BAR|PH|PB|HB|-PHB|ALL]

Description

PERFFORMAT governs the display of performance analysis data according to the chart below.

<table>
<thead>
<tr>
<th></th>
<th>percent</th>
<th>hits (samples)</th>
<th>histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERCENT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HITS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PH</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STANDARD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes

For more information refer to Chapter 5, *Using Performance Analysis*. 
Example

PERFFORMAT ALL

Related commands

PERFDISP, SW_PERFFMT_STAT
PERFINT — specify performance analysis time interval

Works with

- EL 1600
- CodeTAP

Syntax

PERFINT [1 - 120]

Description

PERFINT controls the time in seconds between uploads of performance analysis data from the emulator.

The default for PERFINT is 3.

Notes

You can also set the performance analysis data gathering time interval in XICE.CFG with SW_PERFINT: number.

For more information refer to Chapter 5, Using Performance Analysis.

Example

PERFINT 3
PERFMODE — control performance analysis data display

Works with

- EL 1600
- CodeTAP

Syntax

PERFMODE [ALWAYS | DEMAND]

Description

PERFMODE controls whether performance analysis data is displayed every time it is uploaded from the emulator (ALWAYS), or stored for display at a later time (DEMAND), using PERFDISP.

The default for PERFMODE is DEMAND.

Notes

You can also set the performance analysis display mode in XICE.CFG with SW_PERFMODE:option.

For more information refer to Chapter 5, Using Performance Analysis.

Example

PERFMODE D

Related Commands

PERFDISP
PERFTOL — specify symbol search distance

Works with

■ EL 1600  □ CodeTAP

Syntax

PERFTOL [ 1... ]

Description

Because most addresses do not fall exactly on the beginning of a symbol, it is necessary to search backward to determine to which symbol a traced address belongs. PERFTOL specifies the maximum distance to search before giving up and labelling the address NO_SYMBOL.

PERFTOL also controls how far forward PERFEX will search when trying to exclude a symbol.

The default for PERFTOL is 2000. If no argument is given, the state of the switch is displayed.

Notes

You can also set the symbol search distance in XICE.CFG with SW_PERFTOL:NUMBER.

For more information refer to Chapter 5, Using Performance Analysis.

Example

PERFTOL 2000

Related Commands

PERFEX
PPT — peek/poke trace

Works with

EL 1600

CodeTAP

Syntax

PPT [OFF | ON]

Description

This switch controls the tracing of emulator peek/poke cycles made while in PAUSE mode. If PPT is set to ON, peek/poke cycles while in PAUSE are traced. If PPT is set to OFF, peek/poke cycles while in PAUSE are not traced.

When PPT is ON, XICE may be unable to perform trace disassembly in certain circumstances. This switch can, however, be useful for capturing cycles generated during a DIAG test.

The default for PPT is OFF. If no argument is given, the state of the switch is displayed.

Notes

You can also set PPT in XICE.CFG with SW_PPT:ON or SW_PPT:OFF.

If this switch is ON, the DT, DTB, and DTF commands could give erroneous information.

Example

PPT OFF

Related Commands

DTB, DTF
RAMACCESS — locate a range of RAM memory

Works with

- EL 1600
- CodeTAP

Syntax

```plaintext
RAMACCESS [address|address_range] [{, | =}
{COPY|MAP|TARGET|UNKNOWN} [=bank_range]
```

Description

This command specifies a range of memory locations that can be accessed during execution of the target program. If no parameters are specified, the memory map is displayed in the command viewport. The MAP and COPY options map the specified range to the emulator's overlay memory. TARGET and UNKNOWN map memory to target or as unknown.

Mappings have a 2K minimum granularity. If the beginning and end of an attempted mapping do not fall on 2K boundaries, the emulator automatically adjusts the start and end addresses and returns a warning that the mapping has been adjusted.

Example

```plaintext
ram 0x1000..0x1fff=target  # maps the range to target memory
ram 0x2000..0x2fff=copy    # copies contents of the range from target memory to emulator overlay memory
```

Related Commands

ROMACCESS
RESET — reset processor and target to initial state

Works with

- EL 1600
- CodeTAP

Syntax

RESET

Description

The XICE RESET command asserts both HALT and RESET simultaneously to the microprocessor in the probe module or on the probetip. Once in pause mode, the processor executes a RESET instruction that resets the external target hardware.

If you want to reset the external target hardware without resetting the processor, you must execute a RESET instruction in code. Or you can install a reset button on your target for this purpose.

Since memory is not re-initialized upon use of the RESET command, variables are not reset to their original values.

Notes

None

Example

RESET

Related Commands

RESTART
RFS — control software refresh

Works with

- EL 1600
- CodeTAP

Syntax

RFS [ON|OFF]

Description

This switch enables memory refreshes during PAUSE mode. If RFS is set to ON, memory is refreshed during PAUSE mode. If RFS is set to OFF, memory is not refreshed during PAUSE mode.

If you set RFS to ON, you must also specify the memory area to be refreshed using the following commands:

- RFSADR specifies the memory area to be refreshed
- RFSMSK specifies any mask to be applied to the memory area to be refreshed
- RFSASP specifies the memory space to be refreshed

If you change the values for any of the above switches, the change takes effect immediately.

The default for RFS is OFF. If no argument is given, the state of the switch is displayed.

Notes

The switch PPT specifies tracing peek/poke cycles during PAUSE mode. Setting both RFS and PPT to ON will result in incorrect trace information.
You can also set RFS in XICE.CFG with SW_RFS:ON or SW_RFS:OFF.

Example

RFS OFF

Related Commands

RFSADR, RFSASP
RFSADR — refresh software addresses

Works with

- EL 1600
- CodeTAP

Syntax

RFSADR [address]

Description

This switch specifies the address for memory refreshing during PAUSE mode. It is only active if RFS is set to ON. The related command RFSASP specifies the address space for memory refreshes and RFSMSK specifies any masks to be applied.

The default address for RFSADR is 0x0. If no argument is given, the state of the switch is displayed.

Notes

You can also set RFSADR in XICE.CFG with SW_RFSADR:address.

Example

RFSADR 0x0

Related Commands

RFSASP, RFSMSK
RFSASP — refresh software address space

Works with

- EL 1600  
- CodeTAP

Syntax

RFSASP [space]

Description

This switch specifies the address space for memory refreshing during PAUSE mode. It is only active if RFS is set to ON. The related command RFSADR specifies the address for refreshes and the command RFSMSK specifies any mask on the refresh.

The 68000 and 68302 address spaces are as follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Address Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>1</td>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>2</td>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>3</td>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>4</td>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>5</td>
<td>SD or SC5</td>
<td>Supervisor data space</td>
</tr>
<tr>
<td>6</td>
<td>SP or SC6</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>7</td>
<td>CPU or SC7</td>
<td>CPU space</td>
</tr>
</tbody>
</table>

The default space for RFSASP is 5. If no argument is given, the state of the switch is displayed.
Notes

You can also set RFSASP in XICE.CFG with SW_RFSASP: [space].

Example

RFSASP 5

Related commands

RFSADR, RFSMSK
RFSMSK — refresh software mask

Works with

- EL 1600
- CodeTAP

Syntax

RFSMSK [address_mask]

Description

This switch specifies the address mask for memory refreshing during PAUSE mode. It is only active if RFS is set to ON. The related command RFSASP specifies the address space for refreshes and RFSADR specifies the address for refreshes.

The default mask for RFSMSK is 0. If no argument is given, the state of the switch is displayed.

Notes

You can also set RFSMSK in XICE.CFG with SW_RFSMSK:address_mask.

Example

RFSMSK 0x0

Related Commands

RFS, RFSADR, RFSASP
RIRR — control register restoration on reset (68302 only)

Works with

- EL 1600
- CodeTAP

Syntax

RIRR [ON|OFF]

Description

This switch controls the restoration of the CPU's chip select and control registers (SCR, BAR, WRR, BR0, OR0, BR1, OR1, BR2, OR2, BR3, and OR3). If RIRR is set to ON, the emulator's internal copy of the chip select and control registers is written to the CPU whenever you use the command RESET to reset the emulator. If RIRR is set to OFF, internal registers are not restored to the CPU on RESET.

The default for RIRR is ON. If no argument is given, the state of the switch is displayed.

Notes

- Timer and serial communication controller registers may be saved depending on the setting you use for the switch DRTMR.

You can also set RIRR in XICE.CFG with SW_RIRR:ON or SW_RIRR:OFF.

This command is not used by XICE for the 68000.

Example

RIRR OFF

Related Commands

DRTMR, UIR
ROMACCESS — locate a range of ROM memory

Works with

- EL 1600
- CodeTAP

Syntax

ROMACCESS [address|address_range] [{, l =} {COPY|MAP|TARGET|UNKNOWN} [=bank_range]

Abbreviation

ROM

Description

This command specifies a range of memory locations that cannot be written to during execution of the target program. If no parameters are specified, the memory map is displayed in the command viewport. The MAP and COPY options map the specified range to the emulator's overlay memory. TARGET and UNKNOWN map memory to target or as unknown.

Mappings have a 2K minimum granularity. If the beginning and end of an attempted mapping do not fall on 2K boundaries, the emulator automatically adjusts the start and end addresses and returns a warning that the mapping has been adjusted.

Example

rom 0x1000..0x1fff=target maps the range to target memory
rom 0x2000..0x2fff=copy copies contents of the range from target memory to emulator overlay memory
Related Commands

RAMACCESS
RUN_POLL — set number of polls per second during run

Works with

- EL 1600
- CodeTAP

Syntax

RUN_POLL [n]

Description

This command controls how many times per second the emulator is polled while in RUN mode. Valid values are 1-20.

The default for RUN_POLL is 5. If no arguments are given, the current value is displayed.

Notes

A lower number will slightly reduce response time, but will also reduce network traffic and CPU load. You can set polls per second in XICE.CFG with SW_RUN_POLL:num.

Example

RUN_POLL 10 causes 10 polls per second

Related Commands

XICEVARS
RUN_TIME — set maximum run time

Works with

- EL 1600
- CodeTAP

Syntax

RUN_TIME \[n\]

Description

This command controls the maximum time (in seconds) the emulator will be allowed to run before emulation is broken.

The default is 0, which will allow the emulator to stay in run indefinitely. If no arguments are given, the current value is displayed.

Notes

RUN_TIME controls the time the emulator is actually running. When PERFACT is set to STATISTICAL, the emulator is not always in run due to the overhead associated with performance analysis. Therefore the elapsed time may be several seconds longer than the actual run time.

Example

RUN_TIME 3 causes emulator to run for 3 seconds before breaking
SCRATCH — breakpoint scratch area address

Works with

■ EL 1600  □ CodeTAP

Syntax

SCRATCH [address]

Description

This command specifies the starting address in RAM for the 8 bytes of scratch memory in the supervisor program space needed for software breakpoints. This area must be specified in order to use execution breakpoints and it must be an area that is unused by the program being debugged. If no argument is given, the current address is displayed.

If you do not have any spare RAM in the target system, you may set the scratch space in unused memory and map overlay memory to that area.

When XICE is invoked, it performs a read of the area designated for SCRATCH if SCRATCH is specified. If SCRATCH is set to an area of memory that does not return a DTACK at the end of the read, it will hang XICE. In such a case, comment out the default address for SCRATCH (0x9ff0) in the xice.cfg file. Then specify the address for SCRATCH using this command before setting any breakpoints, or create an include file or start-up file for invoking XICE.

Notes

You can also set SCRATCH in XICE.CFG with SW_SCRATCH.

Example

SCRATCH 0x0

S2-128
SIA — special interrupt vector

Works with

- EL 1600
- CodeTAP

Syntax

SIA [address]

Description

This command specifies the address of the special interrupt vector for forced special interrupts, which are one of the actions that an event system trigger may use.

The default address for SIA is 0. If no argument is given, the current address is displayed.

Notes

You can also set the special interrupt vector in XICE.CFG with SW_SIA:address.

Example

SIA 0xFFFFF00
SIZE — set the size for memory accesses

Works with

- EL 1600
- CodeTAP

Syntax

SIZE [memory_access_type=][124]

Description

This command allows you to examine and set the size to be used for memory accesses. The memory access type you specify must be one of the following:

- CODE: All accesses of code, including fetches
- COMP1: Memory for first argument of COMPARE
- COMP2: Memory for second argument of COMPARE
- COPYFROM: Source memory for a COPY
- COPYTO: Destination memory for a COPY
- DIAG: Memory to use with DIAG
- FILL: Memory for use with FILL
- READ: Memory for generic reads (DUMP, CRC, etc.)
- SEARCH: Memory for use with SEARCH
- STACK: Memory accesses for the stack
- TEST: Memory for use with TEST
- WRITE: Memory for generic writes (SETMEM, etc.)
The code for the sizes are as follows:

1  1 byte
2  2 bytes (word)
4  4 bytes (long)

If you do not specify a size argument, the current state of SIZE for the specified memory access type is displayed. If you do not specify a memory access type, the current state of SIZE for all memory access types is displayed.

Notes

You can also set SIZE in XICE.CFG with SW_SIZE:'[<memory_access_type>] [1|2|4]'$. For example, SW_SIZE:'CODE 2' would set the size for all accesses of code, including fetches, to 2 bytes (word) long. Note that you must enclose the two arguments in double quotes, e.g., SW_SIZE:'CODE 2'.

Example

SIZE CODE 4

Related Commands

MEMVARS
SLO — slow interrupt emulation control

Works with

- EL 1600
- CodeTAP

Syntax

SLO [ON|OFF]

Description

This switch enables inserting a delay before allowing an interrupt when entering RUN mode. If SLO is set to ON, XICE will insert a 160 clock cycle delay before allowing an interrupt. If you set both FAST and SLO to ON, FAST takes precedence, which means that interrupts are enabled immediately upon entering RUN mode. If SLO is set to OFF, the setting for FAST determines whether interrupts are enabled.

The default for SLO is OFF. If no argument is given, the current address is displayed.

Notes

You can also set SLO in XICE.CFG with SW_SLO:ON and SW_SLO_OFF.

If you set both FAST and SLO to ON, FAST has precedence over SLO. The following table shows the results for the possible switch setting combinations for FAST and SLO. This table applies to target-generated interrupts passed to the target processor when the emulator is running.
<table>
<thead>
<tr>
<th>SLO</th>
<th>FAST</th>
<th>Result While in RUN Mode</th>
<th>Result While in PAUSE Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>ON</td>
<td>Interrupts immediately enabled.</td>
<td>Interrupts immediately enabled upon return to RUN mode.</td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td>Interrupts enabled after approximately 160 clock cycles.</td>
<td>Interrupts enabled after approximately 160 clock cycles after return to RUN mode.</td>
</tr>
<tr>
<td>OFF</td>
<td>ON</td>
<td>Interrupts immediately enabled.</td>
<td>Interrupts immediately enabled upon return to RUN mode.</td>
</tr>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>Interrupts generated by the target system will be inhibited from reaching the emulator.</td>
<td>Interrupts generated by the target system will be inhibited from reaching the emulator.</td>
</tr>
</tbody>
</table>

**Example**

SLO ON

**Related commands**

FAST
SPACE — set the space for memory accesses

Works with

- EL 1600
- CodeTAP

Syntax

```
SPACE [<memory_access_type>][address_space]
```

Description

This command allows you to examine and set the space to be used for memory accesses. The memory access type you specify must be one of the following:

- **CODE**: All accesses of code, including fetches
- **COMP1**: Memory for first argument of COMPARE
- **COMP2**: Memory for second argument of COMPARE
- **COPYFROM**: Source memory for a COPY
- **COPYTO**: Destination memory for a COPY
- **DIAG**: Memory to use with DIAG
- **FILL**: Memory for use with FILL
- **READ**: Memory for generic reads (MEMGET, CRC, etc.)
- **SEARCH**: Memory for use with SEARCH
- **STACK**: Memory accesses for the stack
- **TEST**: Memory for use with TEST
- **WRITE**: Memory for generic writes (MEMSET, etc.)
If you do not specify an address space, the current state of SPACE for the specified memory access type is displayed. If you do not specify a memory access type, the current state of SPACE for all memory access types is displayed.

**Notes**

Memory address space is processor-specific. The valid values for the 68000 and 68302 processors are as follows:

<table>
<thead>
<tr>
<th>Address Space</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC0</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>UD or SC1</td>
<td>User data space</td>
</tr>
<tr>
<td>UP or SC2</td>
<td>User program space</td>
</tr>
<tr>
<td>SC3</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SC4</td>
<td>Reserved memory space</td>
</tr>
<tr>
<td>SD or SC5</td>
<td>Supervisor data space</td>
</tr>
<tr>
<td>SP or SC6</td>
<td>Supervisor program space</td>
</tr>
<tr>
<td>CPU or SC7</td>
<td>CPU space</td>
</tr>
</tbody>
</table>

You can also set SPACE in XICE.CFG with SW_SPACE:” [<memory_access_type>] [address_space]”. For example, SW_SPACE:”CODE UP” would set the size for all accesses of code, including fetches, to 2 bytes (word) long. Note that you must enclose the two arguments in double quotes, e.g., SW_SIZE:”CODE SC2”.

**Example**

```
SPACE READ UP
```

**Related commands**

MEMVARS
STI — enable or disable step-through interrupts

Works with

- EL 1600
- CodeTAP

Syntax

STI [ON|OFF]

Description

This switch enables or disables step-through interrupts. If STI is ON, the emulator will recognize an interrupt during a STEP operation and STEP through the interrupt service routine. If STI is OFF, the emulator will ignore interrupts during a STEP operation.

The default for STI is OFF. If no argument is given, the state of the switch is displayed.

Notes

You can also set STI in XICE.CFG with SW_STI:ON or SW_STI:OFF.

Example

STI OFF
TAD — control tri-state of address bus

Works with

- EL 1600
- CodeTAP

Syntax

```
TAD [ON|OFF]
```

Description

This switch specifies whether the address bus is tri-stated while in PAUSE mode. If TAD is set to ON, the address bus is tri-stated while the emulator is PAUSED and during peeks and pokes. If TAD is set to OFF, addresses generated during PAUSE mode are output to the target system.

The default for TAD is OFF. If no argument is given, the state of the switch is displayed.

Notes

You can also set TAD in XICE.CFG with SW_TAD:ON or SW_TAD:OFF.

Example

```
TAD ON
```
TCEBRK — control tracing of breakpoints

Works with

- EL 1600
- CodeTAP

Syntax

TCEBRK [ON|OFF]

Description

This switch controls the capture of software instruction breakpoint execution in the trace buffer. If TCEBRK is set to ON, the breakpoint cycles are recorded in the trace and are marked with an X in the column FLAGS. If TCEBRK is set to OFF, the breakpoint cycles are not recorded in the trace.

The default for TCEBRK is OFF. If no argument is given, the state of the switch is displayed.

Notes

You can also set TCEBRK in XICE.CFG with SW_TCEBRK:ON or SW_TCEBRK:OFF.

Example

TCEBRK OFF
TED — control trace/overlay for external DMA (68302 only)

Works with

| EL 1600 | CodeTAP |

Syntax

```
TED [ON|OFF]
```

Description

This switch enables or disables tracing external DMA cycles (which are generated by the target hardware following the BR, BGF, or BGACK protocol) and enables or disables external DMA accesses to overlay. If TED is ON, external DMA cycles are traced and external DMA can access overlay. If TED is OFF, external DMA cycles are not traced and external DMA cannot access overlay. Also, if TED is OFF external DMA cycles cannot access internal registers or dual-port locations.

The default for TED is OFF. If no argument is given, the state of the switch is displayed.

Notes

The switch TID enables tracing of internal DMA cycles. External DMA can run while the emulator is in PAUSE mode; however, these cycles will not be traced, nor may they access overlay, regardless of the TED setting.

You can also set TED in XICE.CFG with SW_TED.

Example

```
TED OFF
```
Related commands

TID
TID — control trace/overlay for internal DMA (68302 only)

Works with

☐ EL 1600  ☑ CodeTAP

Syntax

TID [ON|OFF]

Description

This switch enables or disables tracing internal DMA cycles and enables or disables internal DMA access to overlay. If TID is ON, internal DMA cycles are traced and internal DMA can access overlay. If TID is OFF, internal DMA cycles are not traced and internal DMA cannot access overlay.

The default for TID is OFF. If no argument is given, the state of the switch is displayed.

Notes

The switch TED enables tracing of internal DMA cycles.

You can also set TID in XICE.CFG with SW_TID:ON or SW_TID:OFF.

This command is not used by XICE for the 68000.

Example

TID OFF

Related commands

TED
TRCCLR — clear trace buffer

Works with

- EL 1600
- CodeTAP

Syntax

TRCCLR

Description

This command deletes the information in the trace buffer.

Example

TRCCLR
TRCFRAME — trace cycle number

Works with

- EL 1600
- CodeTAP

Syntax

TRCFRAME [cycle_number]

Description

This command specifies the trace cycle number to be the time 0 for offset timestamps in a raw trace display.

The default cycle is cycle 0. If no argument is given, the state of the switch is displayed.

Notes

You can also set TRCFRAME in XICE.CFG with SW_TRCFRAME: cycle_number.

Example

TRCFRAME 125

Cycle 125 will be time 0 in the next raw trace display.

Related commands

XICEVARS
TRCINT — trace interval

Works with

- EL 1600
- CodeTAP

Syntax

TRCINT [INTERVAL | OFFSET]

Description

This command specifies how timestamps are displayed in raw trace. If TRCINT is set to INTERVAL, timestamps are displayed as the time interval between successive bus cycles. If TRCINT is set to OFFSET, timestamps are the time relative to the bus cycle number specified by the command TRCFRAME.

The default for TRCINT is OFFSET. If no argument is given, the state of the switch is displayed.

Notes

You can also set TRCINT in XICE.CFG with SW_TRCINT:INTERVAL or SW_TRCINT:OFFSET.

Example

TRCINT INTERVAL

Related Commands

XICEVARS
TRCMODE — trace mode

Works with

| EL 1600 | CodeTAP |

Syntax

TRCMODE [ASM|SRC|BOTH]

Description

This command specifies the type of information displayed by
the commands DT, DTB, and DFT. The default is BOTH. The
valid arguments are:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>displays assembly instructions only</td>
</tr>
<tr>
<td>SRC</td>
<td>displays source level instructions only</td>
</tr>
<tr>
<td>BOTH</td>
<td>displays source and assembly instructions interleaved together</td>
</tr>
</tbody>
</table>

Notes

You can also set TRCMODE in XICE.CFG with
SW_TRCMODE:ASM, SW_TRCMODE:SRC, or
SW_TRCMODE:BOTH.

Example

TRCMODE ASM

Related Commands

DT, DTB, DTF, XICEVARS
TRIG — set status trigger

Works with

- EL 1600
- CodeTAP

Syntax

trig\(n\) = [event\(n\), action|CLEAR|ARM|DISARM]

Description

The TRIGGER command defines the action for XICE to take after an event or a number of events is true. A trigger may be used to control event system resources (timers, counters, etc.), tracing, breaking, or other analyzers. A trigger must be armed for any action to be taken.

The following general rules relate to setting a trigger:

- A trigger may list up to 8 events separated by vertical bars (|’s).
  
  **E.g.,** trig\(1\) = ev\(1\) | ev\(32\), grp2 sets a trigger

  so that any time either event 1 or event 32 is true, XICE will switch its monitoring to only group 2 events.

- Triggers are active only when they have been armed, either explicitly by using the command trig\(n\) = arm, or automatically when they are defined if the XICE switch EVTARM is set to ON.

- If arming a trigger fails (for example, if there are not enough resources to fulfill the request), the trigger will be defined but not armed.

- While armed, you cannot change the definition of the trigger or the event(s) specified by the trigger.
In any one event group, you may not arm triggers using more than the following comparators:
- 2 address comparators (specifying an address range counts as a single address comparator)
- 2 data comparators
- 2 status comparators
- 1 LSA comparator
- 1 counter

You cannot arm triggers for an event breakpoint if you are already using BA breakpoints in your code.

A given trigger \( \text{trig} \ (n) \) can be active (armed) in only one group at a time. However, an event can be used in more than one trigger and more than one group at a time.

By not specifying a trigger number for an arm or disarm command, all defined triggers can be armed or disarmed at once. E.g., \( \text{trig} = \text{arm} \) will arm all the triggers that have been set.

You cannot use the TRIG command to set up a break followed by a macro. However, you can set the switch SW_EVTMODE in XICE.CFG to ON and use the command BREAKCOMPLEX (BC) to do so. When EVTMODE is ON, however, you will not be able to use break as an action in TRIG commands.

To change the group that triggers will be armed in, use the EVTGRP command. To list the status of one or more triggers, use the STATUS command. To clear all the triggers at once, use the EVTCLR command.
Possible actions that can be specified using the TRIG command are:

<table>
<thead>
<tr>
<th>Action</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAK</td>
<td>Breaks emulation as a result of specified conditions, or by default if no action is requested.</td>
</tr>
<tr>
<td>CNT</td>
<td>Count only qualified cycles.</td>
</tr>
<tr>
<td>FSI</td>
<td>Force a special interrupt. FSI provides a way to jump to a specified address when a specific event is detected. It can allow you to patch your code fast and can also allow you to write soft shutdown routines for machinery that cannot be halted using a simple breakpoint. You must set up the SIA (special interrupt address) switch prior to using an FSI action. You may also see some unusual cycles in the trace memory at the address where the FSI occurred. These are internal cycles that are traced as the execution address is changed. These internal cycles are not purged from the trace memory. The FSI routine residing at the SIA address should terminate with a return from exception (RET) instruction. Execution resumes at the address immediately following the instruction that caused the FSI. If this is a soft shutdown, you will probably define a breakpoint at the RET instruction.</td>
</tr>
<tr>
<td>GRP1</td>
<td>Switch to group 1 events.</td>
</tr>
<tr>
<td>GRP2</td>
<td>Switch to group 2 events.</td>
</tr>
<tr>
<td>GRP3</td>
<td>Switch to group 3 events.</td>
</tr>
<tr>
<td>GRP4</td>
<td>Switch to group 4 events.</td>
</tr>
<tr>
<td>Action</td>
<td>Definitions</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>RCT</td>
<td>Resets the counter.</td>
</tr>
<tr>
<td>TGR</td>
<td>The trigger signal is an output that is available from the BNC connector on the back panel of the emulator chassis and from pin 19 of the optional LSA pod. When a trigger event is detected, the trigger signal is asserted and remains so for the duration of the specified bus cycle. If a trigger event is specified for more than one consecutive bus cycle, the signal stays high for the duration of the consecutive bus cycles. You can use the trigger signal as a pulse for triggering other diagnostic equipment. You can also use it in conjunction with a counter or timer for timing subroutines or use it with the optional timestamp pod for timing subroutines. See the command TUNITS for further information on using the optional timestamp capability.</td>
</tr>
<tr>
<td>TOC</td>
<td>Toggle the current setting for the counter. No count until first TOC.</td>
</tr>
<tr>
<td>TOT</td>
<td>Toggle the current setting for tracing or not tracing. Trace uses the following rules: If you do not specify TOT, trace is on. If you specify TOT, when you change event groups, trace goes off and when it encounters the next TOT, it goes ON. If you specify TOT, every time the emulator goes into RUN, trace is OFF until the first TOT is encountered.</td>
</tr>
<tr>
<td>TRC</td>
<td>Trace this bus cycle.</td>
</tr>
</tbody>
</table>
Examples

TRIGGER(1) = ev(4) | ev(5), TOT
TRIGGER(2) = ev(6), TOT

The above command for trigger 1 toggles the current setting for trace when either event(4) or event(5) is true. The command for trigger 2 toggles the setting for trace when event(6) occurs.

Related commands

BC, EV, EVTARM, EVTCLR, EVTGRP, EVTMODE, STATUS
TSRCH — search trace memory for patterns

Works with

- EL 1600  
- CodeTAP

Abbreviation

TS

Syntax

TSRCH [trace range], [addr=value], [data=value], [stat=value], [lsa=value]

Description

Lets you search trace memory for a specified pattern. The syntax is similar to that for the “EV” command, with an optional starting point, or search range. Output is in “DRT” format.

The “value” can be a simple value (0x1000), a range (0x1000..0x2000), or a value with a “care” mask (0x1000&=0xf000).

For the status comparator, values can also be entered mnemonically using the mnemonics recognized by the event system. Mnemonics can be logically combined. See the EV command description on page 2-71 for valid status mnemonics.
Examples

$\begin{align*}
\text{ts} & \ \text{addr}=0x1000 & \text{Search for all occurrences of address } 0x1000 \\
\text{ts} & \ 200,\text{addr}=0x1000..0x2000,\text{data}=0x300 & \text{Start at cycle 200, search for cycles with address range of 1000 to 2000, and data } = 0x300 \\
\text{ts} & \ 200..300,\text{stat}=sd|wr & \text{Search traces cycles 200 to 300 for writes to supervisor data space} \\
\text{ts} & \ \text{lsa}=0x100&=0x300 & \text{Search for bit 8 } = 1, \text{bit } 9 = 0 \text{ in the LSA field, ignore other bits}
\end{align*}$

Related Commands

DRT, EV
## TSTAMP — show timestamp or LSA in trace

### Works with

- **EL 1600**
- **CodeTAP**

### Syntax

```
TSTAMP [ON|OFF]
```

### Description

This switch controls whether the raw trace display reports timestamp or LSA data. If you set TSTAMP to **ON**, the DRT command shows timestamp information. If you set TSTAMP to **OFF**, the DRT command shows LSA information.

If you have a timestamp unit attached to your emulator, you should set TSTAMP to **ON**; otherwise, it should be set to **OFF**.

The default for TSTAMP is **OFF**. If no argument is given, the state of the switch is displayed.

### Notes

You can also set TSTAMP in XICE.CFG with

```
SW_TSTAMP:ON or SW_TSTAMP:OFF.
```

For more information about using time stamp, see Chapter 6.

### Example

```
TSTAMP OFF
```

### Related Commands

- DRT, XICEVARS
**TUNITS — timestamp units**

**Works with**

- EL 1600
- CodeTAP

**Syntax**

```
TUNITS [0x0..0xFF]
```

**Description**

This switch determines the timestamp units displayed in the raw trace display. You should set this switch if you have a timestamp module attached to your emulator and you have set TSTAMP to ON. Otherwise, you should leave it set to 0.

The valid values for TUNITS are as follows:

<table>
<thead>
<tr>
<th>Unit</th>
<th>Time Base</th>
<th>Effect of TGR on Timestamp Counter</th>
<th>Useful Measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>.1 µs</td>
<td>Any TGR high causes the timestamp counter to be reset to 0. No manual reset is required in this mode for either absolute or relative timestamping.</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>0x1</td>
<td>1 µs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2</td>
<td>.01 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3</td>
<td>.1 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x4</td>
<td>1 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x5</td>
<td>.1 µs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6</td>
<td>1 µs</td>
<td>While the TGR is held high by the Event Monitor System, the timestamp counter counts. Manual reset is required in this mode for absolute timestamping, but not for relative timestamping.</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>0x7</td>
<td>.01 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8</td>
<td>.1 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x9</td>
<td>1 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unit</td>
<td>Time Base</td>
<td>Effect of TGR on Timestamp Counter</td>
<td>Useful Measurements</td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>-----------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>0xA</td>
<td>.1 µs</td>
<td>In this mode, a long TGR signal from the Event Monitor System resets the counter. After that, successive short TGR signals turn the counter on and off. Manual reset stops the counter and sets it to zero</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>0xB</td>
<td>1 µs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC</td>
<td>.01 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xD</td>
<td>.1 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xE</td>
<td>1 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xF</td>
<td>NA</td>
<td>This setting is used to count occurrences. Each time the TGR signal goes high, the timestamp counter is incremented. Manual reset is required</td>
<td>Count occurrences</td>
</tr>
</tbody>
</table>

The default for TUNITS is 0. If no argument is given, the state of the switch is displayed.

**Notes**

The setting that you use for TUNITS must match the settings you use for the physical switches on the timestamp module itself.

You can also set TUNITS in XICE.CFG with SW_TUNITS:units.

For more information about using time stamp, see Chapter 6.

**Example**

```
TUNITS 0x0
```

**Related Commands**

```
XICEVARS
```
UIR — update internal chip select registers (68302 only)

Works with

☐ EL 1600  ☑ CodeTAP

Syntax

UIR [ON|OFF]

Description

This switch controls whether the emulator’s copy of the CPU’s internal chip select and control registers are automatically updated. Each time the emulator makes a transition from RUN to PAUSE mode, the internal registers SCR, WRR, BAR, BR0, OR0, BR1, OR1, BR2, OR2, BR3, and OR3 are automatically updated. If UIR is set to OFF, the EL 1600’s internal copy of the chip select registers are not automatically updated.

If you modify the chip select registers while the emulator is paused, you must have UIR set to ON to have the modifications update the internal copy of the registers. If UIR is ON, the modified version of the internal chip select copy will be loaded into the 68302 chip select registers after each RESET command until either they are modified during a RUN, in which case these modifications will be saved during the RUN to PAUSE transition, or until you change the setting for UIR to OFF.

The default for UIR is ON. If no argument is given, the state of the switch is displayed.

Notes

We recommend leaving UIR set to ON.

Timer and serial communication controller registers may be saved depending on the setting you use for the switch DRTMR.
See also the information in section 1 on configuring the chip select registers and the command description for RIRR. RIRR controls resetting the internal chip select registers after a reset.

You can also set UIR in XICE.CFG with SW_UIR:ON or SW_UIR:OFF.

This command is not used by XICE for the 68000.

Example

UIR ON

Related commands

DRTMR, RIRR
UP — move the current scope

Works with

- EL 1600
- CodeTAP

Syntax

UP \{number_of_levels\}

Description

The UP and DOWN commands allow you to move the current scope up or down the runtime stack. This is especially helpful when debugging recursive functions. It is not a good idea to go down farther than you have gone up.

Example

UP 5

Related commands

DOWN
UPL — upload hex data to host

Works with

- EL 1600
- CodeTAP

Syntax

```
UPL "filename", address_range
```

Description

UPL is used to upload data from the target to a host file in the format specified by UPLFMT. The address range is the address range of the data to be uploaded.

The MAP, OVERLAY, SPACE, and SIZE commands affect how memory is accessed by UPL.

Notes

Quotation marks are optional if the file name consists of alphanumeric characters or a period. File names that contain a leading slash must be in double quotation marks (e.g., "/root"). File names that contain a leading backslash must be in single quotation marks (e.g., '\root').

Example

```
UPL my.file ,0x8000..+0x3fff
```

Related Commands

DNL, DNLFMT, UPLFMT, MAP, OVERLAY, SIZE, SPACE, VERIFY
UPLFMT — specify upload format

Works with

- EL 1600
- CodeTAP

Syntax

```
UPLFMT format
```

Description

UPLFMT is used to specify the format for hex file uploads using the UPL command. Recognized formats are:

- **INTEL**: Intel hex format. Extended segment address records and extended linear address records are supported.
- **SREC**: Motorola S3-records with Microtec extensions.
- **XTEK**: Extended Tektronics hex format.

Notes

The default format is SREC. The command XICEVARS displays the status of UPLFMT and all other XICE variables. Symbols are not supported for these formats.

Example

```
UPLFMT AMC
```

Related Commands

DNL, DNLFMT, UPL
VERIFY — memory read-after-write verify switch

Works with

- EL 1600
- CodeTAP

Syntax

VERIFY [ON|OFF]

Description

This switch enables or disables memory read-after-write verification. If VERIFY is set to ON, memory is verified after being written. If VERIFY is set to OFF, memory is not verified after being written.

The default for VERIFY is ON. If no argument is given, the state of the switch is displayed.

Notes

You can also set VERIFY in XICE.CFG with SW_VERIFY:ON or SW_VERIFY:OFF.

Example

VERIFY ON

Related commands

XICEVARS
XICEVARS — display internal debugger variable values

Works with

☐ EL 1600 □ CodeTAP

Syntax

XICEVARS

Description

This command displays the current values and descriptions for all the internal debugger variables.

Example

XICE INTERNAL VARIABLES

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPSPACE ANY</td>
<td>Set processor space in which inst. breakpoints will be set</td>
</tr>
<tr>
<td>DNLFMT SREC</td>
<td>External file format for hex/binary downloads</td>
</tr>
<tr>
<td>DNL_GAP 1</td>
<td>Number of bytes to allow between cached download blocks</td>
</tr>
<tr>
<td>UPLFMT SREC</td>
<td>External file format for hex/binary uploads</td>
</tr>
<tr>
<td>EVTARM ON</td>
<td>Arm (ON)* vs. do not arm (OFF) triggers automatically</td>
</tr>
<tr>
<td>EVTGRP 1</td>
<td>Event group to use when arming triggers. Default: 1</td>
</tr>
<tr>
<td>RUN_POLL 5</td>
<td>Frequency of host polling of emulator (1-20). Default: 5</td>
</tr>
<tr>
<td>RUN_TIME 0</td>
<td>Maximum time to stay running (0: forever)</td>
</tr>
<tr>
<td>TRCFRAME 0</td>
<td>Cycle number for time 0 timestamp alignment. Default: 0</td>
</tr>
<tr>
<td>TRCIINT OFFSET</td>
<td>Display of raw trace timestamps: (INTERVAL) vs. (OFFSET)*</td>
</tr>
<tr>
<td>TRCMODE BOTH</td>
<td>Trace display is assembly (ASM), source (SRC), or both (BOTH)*</td>
</tr>
<tr>
<td>VERIFY ON</td>
<td>Memory verification enabled (ON) vs. disabled (OFF)*</td>
</tr>
<tr>
<td>PERFORMATOFF</td>
<td>If ON, P,A data will be collected and processed.</td>
</tr>
<tr>
<td>PERFMODE DEMAND</td>
<td>Format of P,A. data</td>
</tr>
<tr>
<td>PERFDEPTH0</td>
<td>Display P,A. data whenever uploaded or display on demand</td>
</tr>
<tr>
<td>TSTAMP OFF</td>
<td>How many lines of P,A. output to display</td>
</tr>
<tr>
<td>TUNITS 0</td>
<td>Interpret raw trace field as timestamp (ON) or LSA (OFF)*</td>
</tr>
<tr>
<td>Timestamp units for raw trace display (default of 0)</td>
<td></td>
</tr>
</tbody>
</table>

Related commands

EMUVARS, EVTVARS, MEMVARS
Chapter 3
XICE Tutorial

Overview

This chapter introduces the XICE debugger. It covers the basics needed to prepare the sample code for an embedded system application, introduces the new user to the XICE debugger interface, and demonstrates the use of many commands commonly used in a debug session.

User-entered commands

Throughout the tutorial, commands that you should enter are prefaced by a “>” prompt. Examples, headed by For example only, should not be entered by the user during the tutorial.

Example of a command for you to enter

>context

XICE allows most commands to be abbreviated. The abbreviated command is used whenever possible.

An example of the abbreviated form of the “context (con)” command

>con

All the XICE commands used in the tutorial are covered in depth in the XICE Supplement or the XRAY Documentation Set for 68xxx Family.
Tutorial program

The program *cdemon.x* (SUN version) or *cdemon.abs* (PC version) is used throughout the tutorial. Further references to the tutorial code will use the SUN version, *cdemon.x*. If you are using a PC, substitute *cdemon.abs* for *cdemon.x* references.

The *cdemon.x* program, located in the *demo* directory, is compiled and ready to be loaded by the debugger, allowing you to skip the tools section and go directly to the XICE section if you wish to do so.
Embedded systems considerations

Developing code for the embedded system environment, as opposed to the native operating system environment, is in some ways analogous to leaving home for the first time. Many of the chores that may have been taken care of by parents, like laundry or dishes, must now be done without such support. Likewise, in the embedded systems environment, many of the functions that were performed by the operating system (locating code in memory, communicating with I/O) must now be taken care of by the programmer. These, and other, embedded systems considerations are treated in depth in the “Embedded Environments” chapter of the MCC compiler manual and the “Software Development Cycle” section of the ASM68K manual.

Preparing code for debugging

Environment variables
Before invoking the compiler, assembler, and linker ensure the path and environment variables have been set up. To check the path and environment variables you can type `env` if your are on a UNIX workstation or type `set` if you are using a PC. If they are not set up or you are not sure they are correct, please refer to the XICE Installation Guide and set them up at this time.

Makefile for cdemon demonstration code
The Sun makefile, named `makefile`, for `cdemon.x` is located in the `demo` directory. It shows the flags (options) necessary to produce the symbolic information used by XICE to display source, evaluate expressions and symbolic references, and to display type information.

The makefile also generates the linker command file, `ieee.cmd`, which includes flags for placing symbolic information in the object module, designating output file format, and commands for locating code and data.
A batch file performing functions similar to the makefile is used for the PC version of cdemon.

**Makefile line to generate a linker command file line**

```bash
echo FORMAT IEEE \: \: ieee.cmd
```

The assembler, compiler, and linker invocations are assigned the variable names AS, CC, and LD respectively. The variable names are substituted for the invocations later on in the makefile, to cut down on typing time and errors and to improve readability.

**Makefile line associating a variable with the compiler invocation**

```bash
CC=mcc68k
```

**Makefile line associating a variable with the compiler options**

```bash
CCFLG= -g -nOc -nOl -nOR -c -nQ
```
Embedded systems considerations

Figure 3-1 CDEmon makefile
Compiler flags for symbolic debugging
The following MCC68K compiler command line flags produce linkable object modules containing symbolic information for the debugger.

```
mcc68k -g -nOc -nOl -nOR -c -nQ
-g (generate line number and tracing info) (not default)
```

The remaining data options select other functions.

- `nOc` (disable stack pop optimizations)
- `nOl` (disables local optimizations such as code hoisting)
- `nOR` (disables use of registers for variables)
- `-c` (make object file but don’t link it to make an executable file)
- `-nQ` (display any informational messages)

**Linker command file for cdemon**

In our example the linker command file, `ieee.cmd`, is created by the makefile lines that “echo” the actual linker commands into the command file. This takes advantage of UNIX and DOS’s ability to append (>>) echoed “statements” into a file. The resulting linker command file is shown below.

```
PUBLIC STACKTOP=8000h
ORDER startup,code,strings,zerovars,vars
SECT MMIO_LO=3000h
SECT MMIO_HI=0a000h
SECT VECTORS=0
SECT startup=4000h
LOAD alib
LOAD cdemon
LOAD clib
LOAD com
LOAD data
LOAD mcc68kab.lib
```

*Figure 3-2 Linker command file*
Locating target code and data

In embedded systems applications, code and data are usually located by the linker at compile time instead of by an operating system loader at run time. The linker command file *ieee.cmd* is a good place to examine how this may be accomplished. The lines of the linker command file pertinent to locating code and data are presented and explained below.

```plaintext
PUBLIC STACKTOP=8000h (defines value of the external definition of STACKTOP) 
ORDER startup,code,strings,zerovars,vars (overrides linker’s default ordering of assigning base addresses to segments) 
SECT MMIO_LO=3000h 
SECT MMIO_HI=0a000h 
SECT VECTORS=0 
SECT startup=4000h 
```

Linker switches for symbolic debugging

The following linker commands and flags produce an output file that XICE can symbolically debug. The linker is invoked in the command file mode.

```plaintext
lnk68k -c ieee.cmd -m -o cdemon.x -cdemon.map 
-c ieee.cmd (use ieee.cmd as the linker command file) 
-m (write a memory/symbol map to standard out) 
-cdemon.map (redirect memory/symbol map to file cdemon.map) 
-o cdemon.x (name the linker output file cdemon.x) 
FORMAT IEEE (produce MRI-extended IEEE-695 output file format) 
LIST d,x,s,t (list of linker flags) 
   d (external definition symbols in object) 
   x (external definition symbol table to listing) 
   s (local symbols in object) 
   t (local symbol table to listing) 
The remaining commands perform other functions. 
NAME cdemon (names final output module cdemon when generating ieee) 
```
Additional information
Additional information on subjects covered in this section is available in the compiler and assembler manuals.
Using the XICE debugger user interface

The XICE debugger is a “windowed” user interface. The windows in XICE are called “viewports”. The DOS version has scrollable, zoomable viewports and a keyboard command line interface. The Sun version is similar but offers mouse support primarily in the form of command buttons, point-and-click temporary breakpoints and variable evaluation, and “cut and paste” features. Both versions include on-line help.

Brief information about navigating the user interface, including function keys, mouse, viewports, line editing, and control keys, can be found in the on-line help. Detailed explanations of the features of user interface can be found in the XRAY User’s Guide.

Environment variables

Before invoking the debugger ensure the path and environment variables (XRAY and XRAYLIB) have been set up. To check the path and environment variables you can type env if you are on a UNIX workstation or type set if you are using a PC. If they are not set up or you are not sure they are correct, please refer to the XICE Installation Guide and set them up now.

Debugger configuration file - xice.cfg

For the tutorial, you need to modify two parameters, called softswitches, in the debugger’s configuration file, xice.cfg, before invoking the debugger. The modifications can be made using an editor like “vi” or “ed”. Appendix A in the XICE Installation Guide provides a detailed description of the xice.cfg file.
Before modifying \texttt{xice.cfg}, you should make a backup copy using the UNIX \texttt{"cp xice.cfg xice\_bak.cfg"} or the DOS \texttt{"copy xice.cfg xice\_bak.cfg"} command.

The tutorial assumes the emulator is plugged into the factory supplied “null” target, or that \texttt{NULL\_TGT} is enabled for 68000/HC000/EC000 probe-tip-only configuration emulators. This is referred to as “null” target mode. The tutorial program will be loaded into emulator memory configured (or mapped) to replace (or overlay) the desired address spaces of target memory. Memory can be mapped by commands in the \texttt{xice.cfg} file, in an include file, or at the XICE command line level. We will begin with the default condition of memory already mapped by the “MAP” command in the \texttt{xice.cfg} file. Modifying the memory map at the XICE command line is covered later.

Open the \texttt{xice.cfg} file using your text editor

At this time you should open the \texttt{xice.cfg} file using your text editor.

First \texttt{xice.cfg} modification - “scratch” memory softswitch

\begin{verbatim}
/* SW_SCRATCH:0x9ff0 */
\end{verbatim}

The \texttt{SW\_SCRATCH} statement sets aside 8 bytes of memory starting at address 0x9ff0 as instruction breakpoint “scratch” memory.

The debugger requires 8 bytes of “scratch” RAM located in supervisor program space to support the TRAP instruction used for BI’s (break instructions), temporary breakpoints, and high-level single stepping. The address of the “scratch” RAM is set by the \texttt{SW\_SCRATCH} softswitch in the \texttt{xice.cfg} file. If the \texttt{SW\_SCRATCH} line is “commented out” using C language comment field notation, “/\!*.. */”, you must uncomment this line for the tutorial.
Uncomment the SW_SCRATCH line:

```
/SW_SCRATCH:0x9ff0 /
```

The TRAP instruction number is determined by the SW_EXVEC softswitch in the xice.cfg file. The TRAP instruction selected is reserved for emulator use and should not be used in your code. The default setting for SW_EXVEC selects TRAP 15.

**Second xice.cfg modification - bus timeout softswitch**

The bus timeout switch, SW_BTE, determines whether or not emulation breaks and returns control to the debugger if it enters a hung condition, i.e. waiting for a DTACK signal to be returned from non-existent memory. The timeout is fixed at 1 second. To enable the timeout set SW_BTE:ON.

The softswitch should be turned on while debugging hardware or software.

**Modify the xice.cfg line for enabling bus timeout**

```
SW_BTE:ON \
```

Now you should close the xice.cfg file saving the changes.

**Close the xice.cfg file using your text editor**

**Debugger invocation**

The tutorial is meant to be run from the bin directory of the toolchain. If you are not in the bin directory of the toolchain, cd to there now.

The emulator control software, called shell code, is located in the file with the extension .shl. When starting a debug session, you should invoke the debugger using the “force reload of shell code” option, -e boot. This ensures the emulator control software is correct for the version of the debugger you are
using. If you update the debugger, be sure to install its corresponding updated shell code file. Using the debugger with mismatched shell code will have unexpected results.

**Invoke the debugger with forced reload of shell code**

```
executable_name -e boot
```

As the debugger comes up, you will see messages displayed sequentially on the XICE screen, “Initializing”, “NOTE: Downloading operating system to emulator, please wait”, and “Loading file: xxxxx.shl”. Once the debugger is up and running you may exit the debugger by typing an abbreviated version of quit followed by yes, “q y”.

**Exit the debugger**

```
-q y
```

### Include files - introduction

An include file is simply a file containing debugger commands that will be executed when the file is loaded by the debugger. It is similar to a UNIX shell script or a DOS batch file. For example, the supplied include file, `cdemon.inc`, simply loads the appropriate `cdemon` absolute file from the `demo` subdirectory.

You can invoke the debugger using the “include file” option, `-i filename.inc`. Those with UNIX systems may want to background the program in order to free up a shell or make it easier to “kill” the program if necessary.

**Invoke the debugger with the include file option**

```
executable_name -i cdemon.inc
```

As the debugger comes up you will again see the “Initializing” message. The “Loading file: xxxxx.shl” message will not appear since we did not choose to reload the shell code. You will see a “Reading Absolute File: none” message followed by the two include file commands being echoed in the debugger command window.
At this point, the tutorial code, `cdemon.x` or `cdemon.abs`, has been loaded, and the debugger is ready for commands.

**Required memory**
Although set up for the Applied Microsystems Demonstrator module, the tutorial can work with the “null” target shipped with every new system, or with NULL_TGT enabled, or with your own target system. It requires either overlay or target RAM at addresses 0x0..0xffff. The default `xice.cfg` automatically maps overlay RAM for `cdemon`. If overlay is not available, map target RAM as follows (mapping memory is described on page S3-32):

**Map 0x0 to 0xffff as target RAM:**

```
ram 0x0..0xffff=target
```

The specific memory map for `cdemon.x` is as follows:

<table>
<thead>
<tr>
<th>Section</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>startup</td>
<td>00004000..0000408D</td>
</tr>
<tr>
<td>MMIO_LO</td>
<td>00003000..00003001</td>
</tr>
<tr>
<td>MMIO_HI</td>
<td>0000A000..0000A001</td>
</tr>
<tr>
<td>VECTORS</td>
<td>00000000..00000013</td>
</tr>
<tr>
<td>vars</td>
<td>00004B8C..00004BA5</td>
</tr>
<tr>
<td>code</td>
<td>0000408E..000049C5</td>
</tr>
<tr>
<td>zerovars</td>
<td>000049FC..00004B8B</td>
</tr>
<tr>
<td>strings</td>
<td>000049C6..000049F9</td>
</tr>
</tbody>
</table>
Include file for 68302 setup

When you emulate while plugged into a 68302 target, it is extremely important to correctly setup the System Integration Block (SIB) to match your target’s memory map and DTACK requirements. An incorrectly programmed SIB can potentially cause the emulator to hang. The hardware portion of SIB setup is covered in the EL1600 emulator manual.

The 68302 tutorial depends on the default register settings and overlay memory mapping specified in the xice.cfg shipped with the software; these set up the SIB compatible with the default hardware settings. If you have not changed the default register setting in xice.cfg, you may proceed with the tutorial.

If you have changed xice.cfg, you may want to create an include file that sets up the CPU’s System Integration Block, speeds the download, suppress memory related warning messages, loads the program cdemon.x or cdemon.abs, and switches the debugger into source level display mode.
The following provides an example of such a file:

Note: Comment fields begin with a semi-colon (;). The actual commands appear here in bold type.

```
;cdemon.inc
;Set memory access size to word(2); increases speed of the "load" size write 2
; set "error" to ignore errors and continue include file execution error=continue
; The following are 68302 specific commands
; Other CPU's should ignore the error messages
; set BAR register to Base Address 0xe00000, CFC=0 to ignore FC's
setreg @BAR=0xe000
; set OR0 to internal 6 wait-state DTACK, R/W, any FC in 64k block
setreg @OR0=0xdf0e
; set BR0 enable chip select 0(CSO), any FC, Base Address 0
setreg @BR0=0xc001
; return "error" to its default value of abort include for errors error=abort
; "mwarn off" suppresses memory warning messages during the "load" mwarn off
load './demo/cdemon.x'
; Turn "mwarn" back on mwarn on
; Switch to "source-level" mode mode high
```

Figure 3-3 Sample UNIX include file for 68302 registers

Using Help

The on-line help information displayed by the debugger is in the ASCII file with the suffix .hlp.

Help is available in scrollable menu form or may be invoked with a subject argument.
Using the help menu

Typing "help" or "h" invokes the help information menu of XICE topics and commands. There are two levels of help, the top menu level of alphabetized subjects and the lower level of actual help information on each subject.

Invoke the help menu

`:help`

Scroll up and down through the menu by using the up and down arrow keys (depicted as "^" and "v" on the bottom of the help menu). A carriage return selects the subject highlighted by the cursor and displays the next level of help information.

Scroll down and up using the up/down arrow keys

`:v`
```
```

Scroll down to and select the command "DEFINE"

`:v
>Return

At this point you can scroll through the subjects at the "help information" level. A carriage return or down arrow (CR/v) goes to either the second page of information on the current subject or to the next subject if there's only one page. The up arrow scrolls back through the previous subjects. Pressing Escape (Esc) closes the help window and returns you to the Command viewport.

For example only - Invoking help on the step command

You can also invoke help for a specific command:

```
h step
```
Exiting help
An "Esc" exits help and returns you to the XICE command line. If you want to leave the help information displayed on the screen while you type in a command use the F7 key to exit help instead of Esc.

Exit help and return to the command line
Esc

Additional error message information
If an error message pops up for any reason, you may be prompted by the debugger to type `explain` to receive additional information about the error.

Navigating XICE windows (viewports)

Activating viewports (selecting XICE windows)
An active viewport is indicated by a shaded box surrounding the viewport. You can activate any XICE viewport in either of two ways. One way is to use the viewport activate command, `vactive (va)`, with the viewport’s number as the argument. The viewport’s number is located in the upper right hand corner of each viewport.

Another way, perhaps faster, is to simply scroll through the viewports using the F1 function key.

Activate the code viewport

```plaintext
> va 2
```

Zooming and unzooming viewports (enlarging and reducing XICE windows)
The `zoom (z)` command toggles viewports between their normal and enlarged sizes.
A viewport can be enlarged in either of two ways. You can enlarge and activate a viewport by using the zoom command similar to the way the vactive command is used. Or, you can use the F4 function key to enlarge an active viewport.

**Zoom** or **F4** the enlarged window again to return it to normal size.

**Zoom the code viewport**

```
:\z 2
```

**Return the code viewport to normal size**

```
:\z 2
```

**Scrolling viewports**

You can scroll up and down in an active viewport by using the up and down arrow keys. For a standard SUN or PC keyboard, use the Home and PgUp keys to go to the beginning or end of a viewport.

**Modifying and saving debugger start-up options and viewports**

A set of default debugger display and execution options is read in when XICE is invoked. The options control functions such as breakpoint alignment, default radix, viewport display and others. These options can be changed and saved.

In addition to the options, you can also save the size and position information for all the predefined and user defined viewports. The viewport information and options are saved to a file created by the **startup** command and restored when you invoke the debugger.

Detailed functional descriptions of all the options are in the **XRAY Documentation Set** under the **option** command.
Displaying debugger options
Use the `option (op)` command to display the current set of debugger options. It will zoom the command window to display the options.

Display the current debugger options and unzoom the command window

```plaintext
>op
>F4
```

Modifying debugger options
Use the `option` command with an “option name=value” argument to change an option.

We will not modify any options, but an example is shown below.

For example only - Modifying the debugger default radix

```plaintext
op radix=hex
```

Saving the options and viewports
Use the `startup` command to save options and viewport information to the start-up file. You can choose the name of the start-up file. This allows different users to save personal setups in their own unique start-up files. The `startup` command used with no argument saves the information to a file named `startup.xry`.

To use your saved options, use the `-s` debugger command line switch with your start-up file name as the argument when you invoke the debugger.

We will not save any options now. An example of saving the information to the default start-up file, `startup.xry`, is shown below.

For example only - Saving options and viewports to default file `startup.xry`

```plaintext
startup
```
Recording a debug session

Recording commands
Sometimes it may be useful to record the commands used during a debug session. The log command opens a file and saves the command line input into the named file. The "log" file can be used as an "include" file which can be loaded and executed by the debugger command include to recreate a debug session.

Recording commands and their output
The journal command records both the commands and their output into a file. This will be demonstrated later in the tutorial where we use the command to save the contents of the emulator's "bus cycle trace" memory.

Convenience features

Command history
XICE has a command “history” feature similar to that of the UNIX C Shell. You can display a list of executed commands or recall a specific command from the list by using the command history (hi).

For example only - Using history

hi (display list of executed commands)
hi step<ret> (recall step command if on history list)
hi step<ret><ret> (recall and execute step command)

Another way to re-execute a past command is simply to use the up/down arrow keys to scroll through past commands until you reach the desired one, then press <Return> to execute the command.

Command aliasing
XICE allows you to assign a different name to a debugger command. Use the alias command to do this. Preferred aliases can be placed in an “include” file and loaded at the start of your debug session
For example you may want to rename the `load` command to `dnl`.

**For example only - Renaming load to ld**

```bash
alias load=ld
```

Although XICE allows you to define any alias, you cannot successfully invoke an alias for a command that is itself an alias. Many XICE commands are aliases of XRAY simulator commands. To determine whether a command is already an alias, invoke help for the command. If it is an alias, the help screen says "alias of SS ...." If you create and invoke an alias for such a command, an "Unknown Command Entered" error is returned.
Using debugger functions

Getting oriented with the code

When starting a debug session you will want to get oriented with the code, particularly if the code is not your own. The following commands will help you do this.

**Displaying available modules**
A quick display of the names of the source modules available for debugging is a good place to start. Use the `printsymbol` (ps) command with the `/m` flag and `*` argument to display all module names. Of course with very large programs containing many modules, this may be impractical.

The command will zoom the command window and display the names of cdemon's seven modules along with "type" and address information for each module.

**Display module names and unzoom the command window**

```plaintext
>ps /m *
>F4
```

`Printsymbol` is an important and versatile command with many other options for displaying symbols and subsets of symbols.

**Current viewing (scope) and execution context**
The debugger is capable of viewing a module that is not the current execution module. The current execution module is the module that the program counter (PC) is focused on. If you were to execute a `step` command the debugger would execute the source line pointed to by the PC. Use the `context` (con) command to display the current “viewing” and “execution” modules.
Display context (current viewing and execution modules)

:con

Note the current viewing module line, CDEMON, will have "(view)" at the end of it, while the current execution module line, address 4044, will end in "(PC)".

Changing scope
The viewing context can be changed by using the scope (sc) command. This will cause the source for the module to be displayed in the CODE viewport. It also allows access to the module's symbols and line numbers without having to type the qualifying module or procedure name, saving a considerable amount of typing.

The scope command is case sensitive.

Change the current scope to the module DATA

>sc DATA

Display the current context

:con

You will see the current viewing module is now "DATA", while "4044" remains the current execution module address.

Return to scoping the current execution module CDEMON

>sc

Selecting assembly or high-level debugging mode
You might be debugging a small piece of code that controls some I/O device and decide you need to work with your code at the assembly level. Use the mode (m) command to quickly toggle between high-level and assembly-level debugging modes.
Change to assembly-level debugging mode

> m

The code viewport now displays assembly-level code and its corresponding high-level source code, if any.

Return to high-level debugging mode

> m

Other C source operations
There are other debugger commands that display source without changing scope (list), evaluate expressions (cexpression), find the next occurrence of a given string in the source (next), and display parameters passed to procedures (expand). These commands are covered in both the on-line help and the XRAY Documentation Set.

Checking the state of the debugger and emulator

When starting a debug session you should take a quick look at the state of the debugger and emulator. This is particularly true if someone else has been using the emulator between your sessions. Also, you should examine the state of the debugger and emulator any time you get unexpected results from breakpoints or event system setups.

The following commands will allow you to view and modify the parameters that control the state of the debugger and emulator.

Displaying debugger status
Use the status all command to display the debugger's current version, directory, log file, journal file, startup file, and loaded absolute file. Have this information at hand if you ever call for factory support.

Display debugger status

> stat all
The status information should be displayed in the VIEW(ALL) viewport located directly above the CODE viewport.

Displaying emulator hardware configuration
Use the `hwconfig` command to display the emulator's hardware configuration and the version of emulator control (shell) code. Have this information at hand if you ever call for factory support. The display will vary depending on your particular configuration.

Display the emulator's hardware configuration and shell code versions

```
-hwconfig
EL1600 Enhanced SCSI Controller, version 0.01
EL1600 1M Overlay, version 0.01
EL1600 Dynamic T & B Board, version 0.01
EL1600 68302 SCSI Shell(00), version 2.00
```

Softswitches, options, variables, and "double-argument" commands
Three “display variable” commands - `emuvars`, `xicevars`, and `memvars` - show the state of most of the parameters controlling debugger and emulator functionality. Display the remaining parameters with the `evtvars` and `option` commands.

These parameters are referred to as "softswitches" (short for software switches), options, variables, commands, and "double argument" commands. To help remember the names of the commands to display these parameters, you should think of them collectively as "variables".

The variables can all be modified at the command line or in the `xice.cfg` file, with the notable exception of `evtmode`, which can only be modified in the `xice.cfg` file. Also note the state of the variable `evtmode` is not accessible at the XICE command line. You must look in the `xice.cfg` file for this information.
The \texttt{xice.cfg} file contains a commented list of the variables along with a brief description of each. This list is a useful reference when you are modifying the \texttt{xice.cfg} file. You may want to print a copy of the \texttt{xice.cfg} file to use as a quick reference guide.

In the tutorial we will only display the variables. Detailed functional descriptions of the variables can be found in chapter 2 and in the \textit{XICE Installation Guide}.

\textbf{Activate the COMMAND viewport}

\texttt{>va 1}

You need to \textbf{zoom} the command viewport to see all the variables.

\texttt{Zoom the COMMAND viewport}

\texttt{>z 1}

\textbf{Displaying emulator variables}

The emulator softswitches control how the emulator treats CPU signals and functions like DMA, timers, chip selects, refresh, and interrupts.

\textbf{Display the emulator variables}

\texttt{>emuvars}

You should see a list of the emulator's softswitches showing their state and a brief functional description.

\textbf{Displaying XICE variables}

The XICE variables control tracing and event system functions.

\textbf{Display the XICE debugger variables}

\texttt{>xicevars}

You should see a list of XICE's internal variables showing their state and a brief functional description.
Displaying XICE memory operation variables
The variables controlling memory access operations are called "double argument" commands in the xice.cfg file.

Display the debugger memory operation variables

`-memvars`

You should see a 12x5 matrix showing the twelve memory operations and how each is configured for the four memory access variables.

The twelve memory operations are code, comp1, comp2, copyfrom, copyto, diag, fill, read, search, stack, test, and write.

The four memory access variables are shown with a brief description below.

- **space** `sp/sd/up/ud/cpu/sc0,3,4` (specify memory space)
- **size** 1 (specify access size 1 "byte", 2 "word", 4 "longword", ANY)
- **overlay** on/off (specify overlay(on) or target(off) memory accesses)
- **address** phys/logical (specify physical or logical addressing)

Checking the state of the target

Checking the CPU bus
Use the bus command to display the state of the CPU’s control lines as sampled at the probetip.

Display the CPU bus status

`-bus`
Return the COMMAND viewport to normal size

>F4

Controlling the Emulator and CPU

Changing the contents of a CPU register
While debugging your code, you may find a register holding a different value than what you expected. The setreg command allows you to directly modify the contents of a CPU register. This lets you replace the questionable value with the expected value and test the results.

Go to assembly mode and change the contents of the PC register to 0x100

>m
>:setreg @pc=0x100

Note the PC register value displayed in the REGISTERS viewport has changed to 0100. (You must zoom (Z 14) the REGISTERS viewport on the IBM-PC.)

Return the contents of the PC register to 0x4044 and return to high level mode

>:setreg @pc=0x4044
>m

Other 68000 and 68302 register controls
These are the softswitches that control the DMA (ted, tid), timer (drtmr), interrupt (fast, slo), refresh (drtmr), and chip select (rirr, uir) registers. These softswitches are covered in Chapter 2 of this supplement.
Memory control

Displaying memory addresses and variables
The **dump (du)** command displays the contents of memory at a given address or range of addresses in both hexadecimal and ASCII format.

As do most XICE commands, **dump** also accepts a symbol name as an address argument. This allows us to **dump** the contents of the tutorial’s “memory mapped” output port, “led_port”, without recalling the port’s numerical address. (Note: We could have found the address of led_port with the **printsymbol** command.)

**Dump the contents of "led_port"**

```bash
>du led_port
```

Another way of viewing the contents of “led_port” is to use the **printvalue (p)** command. The **printvalue** command displays the values of expressions according to their type. The **printsymbol (ps)** command will show “led_port” is an array of signed char, so **printvalue** will display character values found at “led_port”.

**Display symbol information about "led_port"**

```bash
>ps led_port
```

**Print the value at "led_port"**

```bash
>p led_port
```

You may want to keep a continuous display of a variable’s value on the screen. The **monitor (mon)** command opens the DATA viewport and displays the selected variable. The display is updated during every “run to pause” transition.

**Monitor the variable "led_port"**

```bash
>mon led_port
```
Modifying memory
Modify memory with the `setmem` command. `Setmem` has a switch for byte, word, and longword data arguments. We will use `setmem` with the longword switch, `/l`, to modify the contents of "led_port", and then view "led_port" with the `dump` and `printvalue` commands.

Set memory at "led_port", then display the new contents

```
>:setmem/l led_port=0x58494345
>:du/l led_port
:p led_port
```

Using the single line assembler
There may be times when you need to make a small change to an assembly module, perhaps just to try something out. Use the debugger's built in line assembler to make your patch and avoid a time consuming "exit debugger, edit code, assemble and link, download, and try-it-out" debugging cycle. The line assembler is invoked with the `asm` command.

Below we assemble a simple "nop" loop. To exit the assembler type a carriage return on an empty assembler line.

Assemble a "nop" loop beginning at address 0x9000

```
asm 0x9000
00009000:  nop
00009002:  nop
00009004:  jmp 9000
0000900a:  <Return>
```

Using the memory disassembler
The memory disassembler can only be used with XICE in the "assembly mode".

Disassemble the "nop" loop at 0x9000. The disassembled memory is displayed in the CODE viewport.
Go to assembly mode and disassemble memory at 0x9000

```
-m
-dis 0x9000
00009000 4E71 nop
00009002 4E71 nop
00009004 4EF9 0000 9000 jmp $009000
```

Return to high level mode

```
:m
```

Other memory operations
There are other memory commands available that fill memory with a given value (fill), copy the contents of one block of memory to another (copy), compare the contents of two memory blocks (compare), and search through memory for a pattern (search). These commands are covered in the XRAY Documentation Set.

Using overlay memory

Overlay memory is emulator memory that can replace target memory by overlaying it, or be used where target memory resources do not exist. Assigning overlay memory to address ranges and access types chosen by the user is called "mapping" overlay.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
</table>

Overlay has a minimum granularity of 2K. If a mapping does not begin and end on a 2K boundary, the emulator automatically adjusts the mapping in both directions to the next 2K boundary and issues a warning that it has adjusted the original mapping.

Displaying the memory map
Use the ramaccess (ram) command with no arguments to display the current overlay vs. target memory map.
Display current overlay vs. target memory map

>ram

You should see a display of the type of memory (ram, rom, or nomem), who owns it (emulator or target), its address range, and how much emulator memory remains.

Mapping overlay memory as RAM
Use the ramaccess command with a range argument to map overlay memory as read/write memory (RAM). Memory mapped as RAM is fully accessible to the executing program and to the user.

We will not alter the memory map in the tutorial, so an example is shown below.

For example only - Mapping overlay from 0x1000 to 0x2000 as RAM

ram 0x1000..0x2000

Mapping overlay memory as ROM
Use the romaccess (rom) command with a range argument to map overlay memory as read only memory (ROM).

Memory mapped as ROM will cause a “write violation” break if written to by the executing program. However, the user can still write to this memory using any debugger memory write command such as setmem or fill.

An example is shown below.

For example only - Mapping overlay from 0 to 0xffff as ROM

rom 0x0..0xffff

Mapping memory as illegal access space
Memory can be mapped as illegal or non-existent by the nomemaccess (nomem) command.
Memory mapped as "nomem" causes an "illegal access" break when the executing program tries to access it for any reason. This is useful when implemented as a built-in bounds checker for the executing program. If there are areas in the memory map that nothing should access, map them as nomem.

Memory mapped as "nomem" is also unavailable to the user, even through debugger memory commands such as setmem or fill.

An example is shown below.

For example only - Mapping 0x1000 to 0x2000 as illegal memory

nomem 0x1000..0x1fff

Mapping overlay memory back to target memory
Use the ramaccess command with the =target argument to reassign overlayed memory to the target.

An example is shown below.

For example only - Returning overlay memory 0x1000 to 0x2000 to target

ram 0x1000..0x1fff=target

Copying target memory contents to overlay memory
The ramaccess command, when used with =copy, first maps overlay memory over the range argument and then copies the contents of target memory into that range of overlay memory.

Use this command when you need to copy the contents of your target ROM or PROM into overlay memory for patching, to avoid having to burn a new ROM.

An example is shown below.
Basic breakpoints

Basic breakpoints include access and instruction breakpoints. Simple to set up and use, they let you stop emulation at a predetermined location in the program.

Setting access breakpoints

Access breakpoints are set to break on a read status (br), a write status (bw), or don’t-cared for a read or write (ba) of the breakpoint address. These breakpoints are implemented using emulator hardware and may be set in RAM or ROM.

You cannot set an access breakpoint when any event system triggers are armed. Disarming the triggers will allow you to set access breakpoints. Likewise, setting access breakpoints causes the event system to be disabled. Clearing the access breakpoints will allow you to arm the event system triggers.

Macros can be attached to access breakpoints. You may have multiple ba, br, and, bw breakpoints set, each with its own macro attached. Macros are illustrated on page S3-42.

Set a write access breakpoint at "led_port"

->bw led_port

Setting instruction breakpoints

Instruction breakpoints (bi) are software implemented breakpoints. They use the TRAP instruction to temporarily replace the instruction at the breakpoint address. These breakpoints can only be set in RAM. There must also be RAM located at the “scratch” address to accommodate the TRAP instruction’s interrupt service routine.
A bi uses one each of the emulator's two address and status comparators. The remaining address and status comparators are available for access breakpoints. This means if you set a bi you can additionally set 3 single-address access breakpoints of the same type (all br's, for example) or one range access breakpoint.

Set an instruction breakpoint at outled

```
>bi outled
```

Clearing breakpoints
Use the clear (cl) command to clear all breakpoints. Use clear with a number or range argument to clear a particular breakpoint or group of breakpoints.

Clear breakpoints 1 and 2

```
>cl 1..2
```

Program execution and related commands
The following commands control resetting the CPU, restoring the program start address, and executing the program in real-time or in steps at a time.

Resetting the processor
Use the reset (rese) command to restore the processor to its initial reset state.

Reset the CPU

```
>rese
```

Restoring the program start address
Use the restart (rest) command to reset the program counter to the program's starting address. For cdemon.x this returns us to "start:" in the ALIB module, address 0x4044.

Restore the program start address

```
>rest
```
Starting and stopping program execution

Use the \texttt{go (g)} command to start or continue program execution. The program will execute until a breakpoint is reached, an error occurs, or the user stops emulation with a \texttt{CTRL-C}.

Use the \texttt{go} command with an address and a passcount to execute until the address is seen “passcount” number of times. The command sets a temporary breakpoint at the address and counts each occurrence of the breakpoint.

\textbf{Execute until "outled" is seen four times}

\begin{verbatim}
>g outled%%4
\end{verbatim}

After a few moments, the emulator will break and display the PC value at the time execution stopped.

Stepping through the program

Stepping refers to executing code a number of lines at a time. Single stepping executes either one source line or one assembly line of code at a time, depending on which mode you are in. To single step use the \texttt{step (s)} command without a number argument.

\textbf{For example only - Executing one line of code and five lines of code}

\begin{verbatim}
s
s 5
\end{verbatim}

Use the \texttt{stepover (so)} command if you want to single step but do not want to step through called routines. This command will execute the entire called routine then stop.

Use the \texttt{gostep (gos)} command if you want to step continuously until a specific condition is met. The condition is defined by a macro you attach to the \texttt{gostep} instruction. For instance, \texttt{gostep} can be used to step until a register holds a particular value.
For example only - Stepping until a condition defined in "my_macro" is met

gos my_macro()
trace information can assist you in diagnosing general memory problems or memory errors that may have shown up in one of XICE's ram diagnostic tests (diag).

Also, with ppt on, cycles generated by XICE memory commands or by downloading code with the load command are included in trace memory. The load command cycles can be a valuable source of troubleshooting information when a download fails for some reason. You can examine the last cycle in trace memory and determine if the download went to valid RAM memory, nonexistent memory, or ROM, for example.

To collect trace information we will restart and then go until we reach the function “outled”.

**Restart, then go to "outled"**

`rest`  
`g outled`

**Displaying raw trace history**

Use dt for displaying raw bus cycle information and optional logic state and timestamp information.

**Display the captured raw trace information**

`>drt`

`Zoom the COMMAND viewport and examine the raw trace. You can scroll up and down in the viewport with the up/down arrow keys to view trace that may have scrolled off the display.**

**Zoom the COMMAND viewport**

`>z 1`

The FRAME numbers on the far left of the trace are used to reference when in trace history the information occurred. The lower line numbers are the last cycles captured prior to a “break” in emulation. The “break” at trace line 1 occurred when we reached the function “outled”.
The other raw trace columns show the address (ADDRESS), the data (DATA), Interrupt Priority Level (IPL), Function Code (FCn), type of memory cycle (MEM), flags set (FLAGS), and logic state information (LSA BITS) when each bus cycle was captured.

Additionally, trace for the 68302 shows a column of information on DMA cycles (DMA), and on the CPU internal access pin (IAC). For the 68000, these columns are replaced by information on the state of valid peripheral access pins (VPA), and the state of the memory access pins (VMA).

**Searching trace history for a pattern (emulator only)**
To search trace memory for patterns, use the `tsrch (ts)` command. The command syntax is similar to the `ev` command. You may qualify the search with combinations of address, data, and lsa patterns, and status. You may also specify a starting line number in trace history.

**Search trace history for an occurrence of a write to address 0xfe**

```
:ts addr=0xfe,stat=wr
```

**Displaying disassembled trace history**
Use `dtb` (display trace backwards) for displaying the trace buffer information formatted in assembly or high-level mode, or as an interleaving of both modes. The `dtf` (display trace forwards) command performs the same trace display function, but in a different direction. Use the `dt` command with a start address to begin disassembling at a particular line in trace.

The `trcmode` XICE variable controls the disassembled trace display mode. The variable’s default (both) causes an interleaving of assembly and source.

**Display the trace information in disassembled format**

```
>dtb
```
The numbers on the far left of the disassembled trace correspond directly to the FRAME numbers on the far left of the raw trace display. They are useful when correlating a line of disassembled trace to its bus cycle equivalent line in raw trace.

Observe the call to the function “outled”, JSR outled, and the source line for the function “outled”, outled(0xFE).

**Saving trace to a file**

You may need a hardcopy of trace or a copy of trace on disk for later referencing. Or, you may have a problem that requires factory support. The Applications department might request a hardcopy of trace memory to assist in solving the problem.

Earlier we discussed the **journal** (jou) command, which records both the commands and their output into a file. You can use the **journal** command to save a partial or entire trace disassembly into a file. The example below shows how to save a trace memory display to a file.

**For example only - Saving part of a raw trace to a file named trace.raw**

```
jou on="trace.raw"
drt 0..42
jou off
```

**jou on="trace.raw"** creates a file named trace.raw as the journal file. **drt 0..42** displays raw traces lines 0 through 42. This display goes to both the COMMAND viewport and the journal file. **jou off** causes journaling to cease and closes the journal file.

Using the /a option with jou allows you reopen and append to an existing file.
Executing XICE commands in run mode (dynamic run mode-drun)

The **drun** command lets you use XICE commands without stopping program execution (run mode).

For instance, you may want to examine trace history (**drt** or **dtb**) while executing your program. If you enter run mode using **drun** instead of **go**, you can use the **drt** command to display the trace history, gathered up to the point where you entered the **drt** command, while the target program continues to run.

To exit dynamic run mode use the **dstop** command.

**Restart, enter dynamic run mode, then display raw trace history**

>`rest`
>`drun`
>`drt`

**Zoom command window, view raw trace, then unzoom**

>`z 1`
>`z 1`

Use the **dupdate** command in dynamic run mode to cause the code, register, stack/trace, and data viewports to be updated at the interval given in the command's argument. The default argument value is 20 polls per minute.

Note that you cannot enter commands while in this mode, and that real time operation is sacrificed during the command's polling process.

**Switch to assembly level, enter dupdate mode, and then observe the stack and register viewports being updated (60 times per minute)**

>`m`
>`dupdate 60`
Exit dupate mode (control-c), exit dynamic run mode (dstop), then switch back to high-level mode

- Control-c
- dstop
- m

Logic state and timestamp options

Logic state probe and timestamp probe options are available from Applied Microsystems. If you have either of these options, see Chapter 6 of this supplement for information on how to use the timestamp option and the following XICE variables. These variables control capturing and displaying logic state and timestamp information.

XICE variables for logic state and timestamp options

tstamp (display lsa or timestamp info on/off)
trcframe (cycle number for time 0 timestamp in display)
trcint (display interval or offset timestamp)
tunits (timestamp units for raw trace 0x0)

Debugger macros

Macros provide an efficient means of executing repetitive tasks or generalizing a task that originally acted on only a specific item. XICE uses the same C-like sequence of expressions, statements, and debugger commands as XRAY to define and invoke macros. Chapter 4 of the XRAY Reference Manual in the XRAY Documentation Set is devoted to explaining how to generate your own macros and to use the predefined macros that come with XICE. The following section demonstrates briefly how to create a macro and then save it into an “include” file that can be executed by the debugger.

Creating a macro

Use the define (def) command to create a macro. This puts XICE in the macro define mode. Notice the prompt will change from a greater-than sign (>) to a colon (:) when you enter this mode. A period (.) in the first column of the line exits macro definition mode.
Most of the commands found in this supplement are Applied’s emulator-specific aliases of “set status (ss)” commands. When these commands are used in a macro you must precede and follow the command “$,” preface the command with “ss,” and usually follow the command with a comma to accommodate the XICE command line parser. To see whether a command is an alias of a “set status” command, invoke on-line help for the command in question. Then press <return>. There will be an “alias of SS command_name” message in the upper right corner of the help screen, i.e. for drt you will see “alias of SS DRT”. The macro we will create demonstrates using the aliased command drt.

**Define a macro named dmp_trc**

```
def dmp_trc()
 :{
 :$
 :ss drt, 
 :$
 ;
 :}
 :
 :
```

**Displaying a macro**

Use the show command to display a macro.

**Display the macro dmp_trc**

```
>sh dmp_trc
```

**Deleting a macro**

Use the delete command to delete a macro. An example follows.

**For example only - deleting a macro**

```
del big_mac
```
Saving a macro to a file
After you determine that your macro works, you may want to save it to a file for later use. The procedure to do this consists of XICE commands to open a file and assign it a viewport number (fopen), display the macro (show) into the file, and close the file (vclose). The resulting file can be used as an include file that recreates the macro.

Save a macro to a file

```plaintext
>fopen 50, "dmp_trc.inc"
>show dmp_trc,50
>vclose 50
```

Fopen should have created a file named dmp_trc.inc in the current directory. The file contains the commands necessary to create the macro dmp_trc, placed there by the show command.

Assigning a macro to a breakpoint
A macro can be assigned to a breakpoint by setting a breakpoint and following it with ";your_macro()". The macro is executed when the breakpoint occurs.

Assign macro “dmp_trc” to a write access breakpoint at “led_port”

```plaintext
>bw led_port;dmp_trc()
```

Restart, then go till the breakpoint is reached

```plaintext
>rest
:q
```

When the breakpoint at “led_port” occurred, emulation stopped and raw trace information was immediately displayed in the COMMAND viewport.

Return the COMMAND viewport to normal size

```plaintext
>F4
```
Access breakpoints and event system breakpoints are mutually exclusive. We need to clear the write access breakpoint before continuing on to the event system.

Clear write breakpoint number 1

```
-rcl 1
```

**Using the event system**

Sometimes running to a basic breakpoint and examining trace history does not provide information specific enough to debug your target's code or hardware. Also, you may want the emulator to perform some action other than breaking when the conditions become true. You may need to trigger an oscilloscope after a complex set of CPU bus cycle conditions become true, or to trace only certain types of bus cycle information under certain conditions. For example, the conditions might be the fifth write that a specific subroutine makes to a certain I/O location.

The event system supplies the mechanism to define conditions and take actions by creating event statements composed of logically combined conditions and binding these to trigger statements that perform the actions. This mechanism allows the emulator to perform various actions based on events of complexity far surpassing that of simple breakpoints.

This section will help you get started using the event system. Comprehensive user information and descriptions of all available conditions and actions are in Chapter 4, "Using Breakpoints and the Event System."

**General information**

The event system is implemented with emulator hardware and can be used in both RAM and ROM regions.

Setting access breakpoints disables the event system. Clearing the access breakpoints will allow you to arm event system triggers.
**Step 1: Setting up event statements**

The first step in setting up an event statement is deciding what condition(s) you need to include. For most simple address and status conditions you probably need only an access breakpoint. We will start out here with those conditions however, to keep the first event statement simple.

**Define an event statement for a write to "led_port"**

\[
ev(1) = \text{addr} = \text{led}_\text{port}, \text{stat} = \text{wr}
\]

\( \text{ev}(1) \) begins the definition of event number 1.

\( \text{addr} = \text{led}_\text{port} \) defines “\text{led}_\text{port}” as the address of interest.

\( \text{stat} = \text{wr} \) defines the access to “\text{led}_\text{port}” as a write.

**Displaying event status**

Use \text{stat ev} to display the event in the VIEW(EV) viewport.

\[
\text{> stat ev}
\]

The first line in the VIEW(EV) viewport indicates there were 32 events available. We used one event, leaving 31 currently available.

The second and third lines display the event we set up, \text{EV}(1). Notice that the address is displayed in both numerical form and by its symbolic name, “\text{led}_\text{port}”.

The remaining lines indicate that no other events are defined.

If we had defined many events you would need to zoom the viewport to display them. The following demonstrates zooming and scrolling the VIEW(EV) viewport.

**Zoom the VIEW(EV) viewport, scroll around, then unzoom**

\[
\text{>z 24}
\]

\[
\text{><down arrow>}
\]

\[
\text{<-up arrow>}
\]

\[
\text{>z 24}
\]
Step 2: Setting up trigger statements (assigning events to actions)

Next, you choose what action(s) are taken when the conditions defined in the event statement(s) occur. The event statement is tied to the action(s) by forming a trigger statement. When the event becomes true the actions happen.

We'll bind the simple action of stopping emulation to our event statement by defining trigger statement 1.

Define a trigger binding the break action to ev(1)

\[ \text{::trig(1)}=\text{ev(1)}, \text{break} \]

Arming triggers

Trigger statements must be “armed” before they are considered active. The easiest way to arm triggers is to leave the “auto-arm” variable evtarm at its default state of on. With evtarm on the triggers are automatically armed as soon as they are defined.

Triggers are armed for a particular "group". The evtgrp command determines what group a trigger statement is armed for when the trigger statement is defined. If you don't need to use an additional "group" of triggers leave evtgrp set to 1. In this tutorial we leave evtgrp at its default value of 1. This means the trigger we set up is armed for group 1.

Displaying trigger status

Use stat trig to display the trigger statement in the VIEW(TRIG) viewport.

Display trigger status and zoom the VIEW(TRIG) viewport

\[ \text{::stat trig} \]
\[ \text{::z 24} \]

The first line in the VIEW/TRIG) viewport indicates there were 16 triggers available. We used one, leaving 15 currently available.
The second line shows trigger statement 1 (TRIG {1}) is associated with ( = ) event statement 1 (EV{1}).

The third line indicates trigger 1 is armed for group 1.

The fourth line shows the action to take.

The fifth line shows the actual trigger statement definition.

The remaining lines indicate the other triggers are undefined, unarmed, and available (CLEAR).

**Unzoom the VIEW(TRIG) viewport**

```
>z 24
```

**Step 3: Using the trigger statement**
The trigger statement we set up is armed and ready for use. As soon as we go into "run" mode, the event system will unobtrusively monitor the executing environment until the event statement conditions are met. When this happens the actions in the trigger statement occur.

**Restart, then run the program until the event systems takes over**

```
>rest
>g
```

The emulator will "break" and display the current PC value. In the CODE viewport you should see a highlighted source line indicating where the break occurred. The symbol "led_port" should also be on this line.

**Assigning a macro to an event system “break” action (breakcomplex)**
Use the **breakcomplex (bc)** command to tie a macro to an event system “break” action. Since it severely limits the event system when in effect (only one armed trigger allowed), this command should only be used when you absolutely need to tie a macro to an event system “break” action. Use a macro tied to a basic breakpoint whenever possible.
To use bc the evtmode variable must be on. This variable's default is off. Unlike most other XICE variables, evtmode can only be changed by exiting the debugger, modifying the SW_EVTMODE command in the xice.cfg file, and then re-invoking the debugger.

You must set up for a bc in a carefully prescribed order.

1. Set up the event statement.
2. Tie the macro to the trigger statement and the bc.
3. Define the trigger statement with an event statement but no action. (No action is used in the trigger statement since the "break" action is built into the bc command.)

An example demonstrating the exact order of commands necessary to tie a macro to an event system "break" trigger follows. Assume evtmode is on.

For example only - Attaching the macro “dmp_trc” to an event system trigger

```plaintext
ev{1}=addr==led_port,stat==wr
bc trig(1); dmp_trc()
trig{1}=ev{1}
```

Additional event system features
In addition to the simple conditions and actions illustrated here, the event system possesses many advanced features such as groups, counters, timers, flags, "soft shutdown", conditional tracing, trigger generation, and others. These features are covered in Chapter 4 of this supplement.

Scope loops and diagnostics

Built in scope loops and memory diagnostic programs are included with the debugger in the form of diag commands. These programs save you from writing your own routines to test memory or to stimulate memory for “scoping” or logic analysis.
Another diagnostic, named **crc**, calculates the CRC-16 (cyclic redundancy check) over the desired range.

**Memory and IO read/write scope loops**
Diagnostic numbers 2 through 8 are used to perform reads and/or writes of selected memory with patterns chosen by the user.

**For example only - scope loop, continuous read of address 0x4000**

`diag 2,0x4000`

**Memory diagnostics, simple and complex**
Diagnostic numbers 0 and 1 perform simple and complex diagnostics on the selected memory.

**For example only - complex memory test of address range 0x2000 to 0x2400**

`diag 0,0x2000..0x2400`

**Cyclic Redundancy Check**
Use the **crc** command with a range argument to perform a CRC-16 of the specified range. The command will return a hex value for the CRC.

**For example only - cyclic redundancy check of address range 0x2000 to 0x2400**

`crc 0x2000..0x2400`
Chapter 4
Using Breakpoints and the Event System

Overview

This chapter covers the breakpoint and conditional control features of XICE when used with the EL 1600 event system.

Basic breakpoints are tools for interrupting emulation or simulation in order to inspect trace for insight into code execution. Typically, they take two forms: access breakpoints and execution breakpoints.

The EL 1600 event system provides XICE with additional flexibility both in what can cause the emulator to intervene in code execution and in what actions can occur. These features provide powerful capabilities for debugging and integration.

Because all break capabilities and event conditionals are implemented using event system resources, the following sections explain both in terms of event system resource use. The initial sections explain the automatic invocation of event resources to provide basic access and execution breakpoints. Later sections explore using the full power of the event system to control emulation using sets of conditionals that you define.

Emulator and simulator versions of the debugger

This chapter deals only with XICE in-circuit debugger extensions to the XRAY simulated event system. The commands mentioned here are documented in the alphabetical command reference, Chapter 2, of this supplement. The XICE version uses emulator hardware resources to exercise and respond to actual processor and other target resources. For information about the XRAY simulator's event system, refer to the SET STATUS command description in the XRAY Reference Manual.
Breakpoint and event system commands

The following XICE commands comprise those for basic breakpoints and the event system.

<table>
<thead>
<tr>
<th>Basic Breakpoint Command Names</th>
<th>Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAKACCESS</td>
<td>BA</td>
<td>Set a breakpoint on access to specified address(es)</td>
</tr>
<tr>
<td>BREAKINSTRUCTION</td>
<td>BI or B</td>
<td>Set a breakpoint on specified instruction</td>
</tr>
<tr>
<td>BREAKREAD</td>
<td>BR</td>
<td>Set a breakpoint on a read at specified address</td>
</tr>
<tr>
<td>BREAKWRITE</td>
<td>BW</td>
<td>Set a breakpoint on a write at specified address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event System Command Names</th>
<th>Mnemonic</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAKCOMPLEX</td>
<td>BC</td>
<td>Set a breakpoint that calls a named macro.</td>
</tr>
<tr>
<td>EV</td>
<td>EV</td>
<td>Define event statement</td>
</tr>
<tr>
<td>EVTARM</td>
<td>EVTARM</td>
<td>Enable/disable automatic trigger arming</td>
</tr>
<tr>
<td>EVTCLR</td>
<td>EVTCLR</td>
<td>Clear event system</td>
</tr>
<tr>
<td>EVTGRP</td>
<td>EVTGRP</td>
<td>Specify event group</td>
</tr>
<tr>
<td>EVT VARS</td>
<td>EVT VARS</td>
<td>Display all event state variables</td>
</tr>
<tr>
<td>STATUS EVENT</td>
<td>STAT EV</td>
<td>Display event statement(s)</td>
</tr>
<tr>
<td>STATUS TRIGGER</td>
<td>STAT TRIG</td>
<td>Display trigger statement(s)</td>
</tr>
<tr>
<td>TRIG</td>
<td>TRIG</td>
<td>Define trigger statement</td>
</tr>
</tbody>
</table>
Working with basic breakpoints

You use a breakpoint to examine behavior of the target under certain controlled conditions. Typically you set breakpoints either on accesses to specific memory locations or on specific instructions in code. This is very helpful in isolating bugs when troubleshooting hardware and software in the target environment.

Setting basic breakpoints

There are two categories of basic breakpoints: access and instruction breakpoints. You use an access breakpoint (BA) to break on reads and/or writes to data and program locations. By specifying the type of access, you can limit the access break to break exclusively on read (BR) or write (BW) accesses. When you want to stop program execution on a particular instruction in your code, you use an instruction breakpoint (BI) or a temporary “go-until” breakpoint.

You set up basic breakpoints by typing the appropriate breakpoint command at the prompt. The debugger manages the event resources, assigns a number to each breakpoint for reference, and displays them in the breakpoint viewport (va 25).

Displaying breakpoints

Breakpoints are displayed in viewport 25. Using the VACTIVE command (VA 25), you can see the current breakpoints. This viewport also opens when you set a breakpoint, issue a breakpoint command without an argument, or use the command OPTION BREAK ON. (OPTION BREAK SWAP returns the window to normal mode.)
Clearing breakpoints

Breakpoints may be cleared using the CLEAR command. You can clear an individual breakpoint by giving its number or all breakpoints by not specifying a number.

Instruction breakpoints (BI, GO_instruction)

XICE uses a software execution breakpoint mechanism to provide up to 32 instruction execution breakpoints (BI), as well as temporary breakpoints (GO_instruction) and high-level single stepping. The break occurs immediately before the instruction actually executes.

Use the following syntax:

<table>
<thead>
<tr>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI #20</td>
<td>Sets a breakpoint at line number 20. May require module name.</td>
</tr>
<tr>
<td>BI SIEVE#28</td>
<td>Sets a breakpoint at line number 28 in the module SIEVE.</td>
</tr>
<tr>
<td>BI 0x2210..0x2216</td>
<td>Sets breakpoints starting at address 2210 and ending at address 2216</td>
</tr>
<tr>
<td></td>
<td>(hexadecimal), assembly-level mode only.</td>
</tr>
<tr>
<td>BREAKI #1..#4</td>
<td>Sets breakpoints starting at line number 1 and ending at line number 4.</td>
</tr>
<tr>
<td>BI #15..#18;FOO()</td>
<td>Sets breakpoints starting at line number 15 and ending at line number 18. Executes macro FOO after each line.</td>
</tr>
</tbody>
</table>
Bl #10; when(i==3)  
Sets a breakpoint at line number 10 and stops only if variable i is equal to 3.

Bl 0x93  
Sets a breakpoint at address 93 (hexadecimal), assembly-level mode only.

Bl step  
Sets a breakpoint at the address of step.

Ranges
You can specify an address range when defining a basic breakpoint (BI). XICE handles ranges by breaking them into multiple individual single-point breakpoints. Thus, if you specify that a breakpoint should be for a range of 20 addresses, you may set only 12 additional breakpoints.

Setting one-time breakpoints
A temporary or one-time breakpoint is an execution breakpoint attached to the current XICE GO instruction. Temporary breakpoints are commonly used to skip over a section of code or a subroutine. You can either click on an instruction in the code window (platforms with mouse support only) or enter the address or line number on which to break as a suffix to the GO command.

go 0x1234  
Tells the emulator to run until it sees address 0x1234 on the bus

The same software execution breakpoint mechanism is used; the emulator runs to it and clears the breakpoint after breaking. If a macro or other operator is attached, it acts just as it would if attached to permanent breakpoints.

Trace display
When you view disassembled trace after an instruction breakpoint is reached, the display shows the instruction break at the point you defined. However, the raw trace display may
show extra pre-fetches that result from filling the CPU pipeline. The break has occurred at the specified point, as is shown by the CPU program counter (PC).

**Initial setup**

The execution breakpoint mechanism replaces the actual instruction in memory with a TRAP \( n \) instruction and sets an access breakpoint to stop execution on instruction fetches in the interrupt service routine. Consequently, these breakpoints will use target or overlay resources and can only be used in writable memory. So some setup is necessary before the initial use of execution breakpoints and temporary breakpoints.

In most cases, setup is virtually transparent, even for code that resides in PROM, if you use overlay memory and the defaults in xice.cfg:

- **Dedicate a TRAP for XICE to use.**

  The default set in xice.cfg is TRAP 15; if your code makes no use of this trap, skip to the next bullet.

  You must allocate exclusive use of one of the 16 TRAP vectors to XICE. Use the EXVEC command, or set a default with the SW_EXVEC softswitch in xice.cfg.

  The assigned trap in the vector table must be in writable memory so that it can be modified to point to the scratch area. This memory may be target RAM or emulator overlay ROM or RAM. If your vector table resides in target ROM or PROM, use the emulator's overlay memory to map the vector area as ROM; then the vector can be modified by XICE, but the area is protected from target writes during program execution. For example, if the vector table resides at 0x0000..0x0013 in target PROM, enter the following to map it to overlay ROM:

  \[
  \text{rom 0x0000..0x0013}
  \]

- **Define scratch space in writable memory.**
The default 8-byte scratch space set in xice.cfg starts at 0x9ff0; if your program never makes use of this area, skip to the next bullet.

XICE needs 8 bytes of writable memory for its breakpoint routine. This SCRATCH memory area may be either target RAM or emulator overlay RAM or ROM. If you do not have any unused target RAM, pick an area of target memory outside the boundaries of your memory map, and map overlay to that area.

Set the SW_SCRATCH softswitch in xice.cfg to the scratch area's starting address or allocate scratch space using the SCRATCH command before using a BI or temporary breakpoint. If the area you select does not return DTACK, see the guidelines in “Working within the limits”.

☐ Ensure that the breakpoint falls on an instruction boundary.

If you use an instruction's label, XICE always places the breakpoint in the correct place. If you use a hex address in assembly mode or a line number in source mode, XICE does not check for alignment. In some instances, code corruption can result if you do not place the breakpoint on the start of the instruction.

☐ Ensure that the instruction resides in writable memory.

The memory location on which you are placing the instruction breakpoint must be in writable memory so that the actual instruction can be replaced by the trap instruction. This may be either target RAM or emulator overlay memory. If the code resides in PROM, use the emulator's overlay memory to map the area as ROM; the code space can then be modified by XICE, but the area is protected from target writes during program execution.

For example, to overlay the wait instruction from target PROM to ROM overlay, enter:

```
rom wait=copy
```
For example, this is what takes place if you set an execution breakpoint on \textit{wait} in the \texttt{cdemon} program using the defaults for scratch RAM and exception vectors:

\textbf{bi wait}

1. An address comparator and status comparator are set up to monitor execution of the code in the SCRATCH memory area at 0x9ff.
2. The \textit{wait} instruction is replaced by the trap instruction specified by TRAP 15. The original \textit{wait} instruction is saved to be restored immediately following any breakpoint.
3. During program execution, the trap executes.
4. The trap instruction causes an interrupt through the EXVEC trap vector. Code execution then begins in the SCRATCH memory area.
5. The event system detects program execution in the SCRATCH memory area and triggers the breakpoint.
6. The break is reported to XICE, and the wait instruction is restored in code.

\textbf{Working within the limits}

Execution breakpoints consume event system resources and affect what is possible using other features. In general, the emulator manages these resources and warns you when it makes adjustments and presents an error when resources are exhausted or when you attempt something that creates a conflict. So you need not concern yourself with more than the following general guidelines.

- If you use them in addition to the event system, note that instruction breakpoints consume an address and a status resource in each event group. Limit event system address/status resource consumption to no more than one address and one status comparator in each group. Set up the event and trigger statements, but leave them disarmed until you are ready to use them.
You can specify an address range. XICE handles ranges by breaking them into multiple individual single-point breakpoints. Thus, if you specify that a breakpoint should be for a range of 20 addresses, you may set only 12 additional breakpoints.

If you use them with access breakpoints, note the limitations explained in the “Access breakpoints” section that follows.

When XICE is invoked, it performs a read of the area designated for SCRATCH if SW_SCRATCH is specified in xice.cfg. If SCRATCH is set to an area of memory that does not return a DTACK at the end of the read, it will hang XICE. In such a case, comment out the default address for SCRATCH (0x9ff0) in the xice.cfg file. Then specify the address for SCRATCH before setting any breakpoints by using the SCRATCH command or preferably in an include file when you invoke XICE.

Access breakpoints (BA, BR, BW)

An access breakpoint sets a read, write, or read/write breakpoint at the specified memory location or range. An access breakpoint halts program execution each time the target program attempts the specified type of access at the specified location(s).

Use an ampersand (&) to reference symbolic addresses rather than just the symbol names. Using a symbol name alone returns its value, not the address.

You have up to 6 access breakpoints of the same type (e.g., all reads) or up to 3 each of two different types. They are set using the syntax:

```
[ ba | br | bw ] [ address | address_range ] [ ;macro_name ]
```

- `ba` for read or write access
- `br` for read access
- `bw` for write access
<table>
<thead>
<tr>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR 0x300</td>
<td>Sets a read access breakpoint at address 300 (hexadecimal).</td>
</tr>
<tr>
<td>BW @sieve\flags</td>
<td>Sets a write access breakpoint at the address of the variable array flags in the root named @sieve.</td>
</tr>
<tr>
<td>BA flags..flags+10</td>
<td>Sets read/write access breakpoint starting at the address of the array flags and ending 10 bytes after the address of flags.</td>
</tr>
<tr>
<td>BR 20h..30h;FOO()</td>
<td>Sets read access breakpoints from address 20h (hexadecimal) to 30h and executes the macro FOO on every breakpoint between these addresses.</td>
</tr>
<tr>
<td>BR &amp;flags[0]</td>
<td>Sets a read access breakpoint at the address of array element flags[0].</td>
</tr>
<tr>
<td>BA prime</td>
<td>Sets a read/write access breakpoint at the address referred to by the value in variable prime.</td>
</tr>
</tbody>
</table>

This command is correct if prime is a pointer. The breakpoint is set at the location of the variable prime. For example, if the value of prime is 0x0123, a breakpoint is set at the address 0x0123.
This command may not be correct if prime is a scalar, since the value in prime is treated as an address and the breakpoint is set at that address rather than at the address of the variable prime.

**BW &prime**

Sets a write access breakpoint at the address of the variable prime regardless of its type.

This command is correct if prime is a scalar; it sets a breakpoint at the address of the variable prime.

If prime is a pointer, the breakpoint is set at the address of the pointer rather than at the address it is pointing to (i.e., prime).

**BA &count; when (k<30)**

Sets a read/write access breakpoint at the address of count and only stops when the value of k is less than 30.

Access breakpoints begin breaking on the cycle in which the access occurs but may continue or “skid” several cycles after access to the breakpoint location.

XICE performs the following functions when it encounters an access breakpoint:
1. Completes the execution of the instruction at that location.
2. Suspends program execution.
3. Executes a macro (if one was specified when the breakpoint was set). Depending on the macro, the debugger will do one of the following:
   - If the macro return value is true (nonzero), the debugger resumes execution at the instruction immediately after the breakpoint.
   - If the macro return value is false (zero), the debugger returns to command mode and displays breakpoint information.
4. If a macro was not specified, XICE returns to command mode and displays updated breakpoint information.

**Working within the limits**

Access breakpoints consume event system resources and affect what is possible using other features. In general, the emulator manages these resources and warns you when it makes adjustments and presents an error when resources are exhausted or when you attempt something that creates a conflict. So you need not concern yourself with more than the following general guidelines.

- Up to 6 access breakpoints are possible when you are not using BI execution breakpoints. You can have 6 breakpoints of one access type (read/write/either) or two groups of 3 with different access types.
- Up to 3 access breaks of the same type (ba, br, or bw) are possible when you do use BI execution breakpoints. The first BI uses one each of the two available address and status comparators, leaving one of each for use with access breakpoints. Each address comparator may have three address points or one range associated with it; hence the three possible access breakpoints.
- Currently, an XRAY68K core limitation causes an error when you attempt to set an access breakpoint of one access type within a previously defined range of another access type. For example, if you set a read access break over the
range 0x4000..0x47ff, you cannot set a write access break for an address within that range.

- Setting an access breakpoint disables the event system. Likewise, arming an event trigger prevents use of access breakpoints.

**Breakpoint latency**

Instruction breakpoints have no execution latency associated with them. The break always occurs immediately prior to the execution of the instruction in question. Since the execution break mechanism does not generate any external events, there is no time-latency either.

The execution delay associated with an access breakpoint is up to six clock cycles, plus any instructions currently in progress at the time the request is completed.
Working with the event system

The event system provides emulator, target, and execution break control, allowing you to monitor any predefined series of conditions in realtime and then perform emulator actions based on those conditions.

The event system covered in this chapter is only available in XICE, the emulator version of the debugger. It monitors target information at the bus cycle level, including every read or write cycle that the microprocessor executes. The event system “sees” every signal that can affect the target system. It can take a variety of actions based on conditions that you predefine.

You can think of events (conditions) as inputs to the event system and triggers as statements that tie inputs to outputs. Figure 4-1 shows the structure of the event system. The possible events are listed on the left, and the possible triggers are listed on the right.

Basic concepts are introduced in Figure 4-1 and outlined in “Working with the Event System.” More complete coverage of each procedure can be found under “Events” and “Triggers” later in this section. The tutorial in Chapter 3 also provides a practical exercise in event system use.
To use the event system, you first define conditions (event statements) using the **EV** command. Then, using the **TRIG** command, you define actions (trigger statements) that are to take place under those conditions. Up to 32 event statements and 16 trigger statements can be defined. Typically, you will modify the construction of the event system during the course of a debugging session. You can employ different events and triggers from among those defined as you go.
In order to be in effect, an event statement must:

- be defined
- be referred to in an active trigger statement

In order to be active, a trigger statement must:

- contain a reference to one or more event statements that are in effect
- be armed
- be in the group that is current

If you set EVTARM to ON, triggers are armed by default when defined. The event system hardware resources may limit the number of triggers that you can have armed at the same time. Disarming and rearming triggers with `TRIG` saves you from retyping them every time they are armed.

**Example**

```
ev{1}=data==0x1234  # Event 1 is active when 1234 appears on data bus.
trig{1}=ev{1}, break  # Event 1 triggers trig 1, which is a break.
```

**Note**

Break is the default action. The trigger in the example above could have been stated: `trig {1}=ev{1}`

**Event and trigger statement groups**

A group is a set of events and triggers that can be activated and deactivated all together. There are four groups in the event system (1, 2, 3, and 4), one of which is current at any given
time. This gives the event system a state-machine capability for debugging difficult problems. The emulator always starts in group 1 each time you enter the command GO.

When setting up events and triggers, you always associate them with a group. The group a particular event or trigger is associated with is the one current at the time you set it up.

The GROUP command shows the group that was current when the emulator last broke, and the EVTGRP command shows the group to which new event statements will be added.

Example

```
GROUP displays the number of the group current at the break.

EVTGRP displays the group new events and triggers will be added to.

EVTGRP 2 Makes “group 2” the group new event or trigger statements will be added to.

GROUP Response not updated until next break.

EVTGRP Responds “2”.
```

Displaying event system constructs

Event and trigger statements can be displayed using the STATUS command. By issuing the command STAT EV you can see the current event statements. By issuing the command STAT TRIG you can see the current trigger statements.

Clearing event system constructs

Event statements may be cleared using the EV or TRIG “clear” option. When a trigger is cleared, it is disarmed and cleared from XICE. When an event is cleared, it is removed from XICE as long as it is not currently referred to in an armed trigger. If
it is used by an armed trigger, then it is set in the emulator and cannot be cleared until all armed triggers that use it are either disarmed or cleared.

The **EVTCLR** command is provided to clear the entire event system. It clears the event and trigger statements in from XICE and resets the emulator’s event hardware.

**Logging event system constructs**

If you wish to use a set of event and trigger statements for future debugging, you can use the **LOG** command to capture statements as you write them. You can then edit the log file with a text editor to create an include file that reproduces your event system setup.

**Events**

An event statement is constructed of one or more events and defines a portion of a potential bus cycle. The definition can include the state of the processor bus (the address, data, and status busses) and/or the state of other event resources (counters and LSA). You define an event to be valid when the portion of interest of the current cycle matches the value specified in the event statement. Parts of the cycle not defined in the event statement are ignored. The event system watches the busses and other event resources for valid events and matches them with actions specified in armed triggers.

You can define up to 32 event statements in XICE. A defined event can be used in any or all of the four event system groups.

**Examples**

\[ \text{ev(1)=addr=0x1234, status!=word} \]

Event 1 is active if 0x1234 appears on the address bus and data is not a word.
In an event statement, the comma (,) is the AND operator for the whole statement and the (!=) is the NOT operator for a whole event within the statement.

\[ ev(1) = \text{status} != \text{not_word} | \text{wr} \]

Event 1 is active if this is not a word operation and the data bus is in write mode.

The string “NOT_” is the NOT operator within the status event definition and the pipe (|) is the AND operator within the status event definition. Not all status conditions accept the “NOT_” construction. See the SETSTATUS EV description in Chapter 2.

The description of the EV command in Chapter 2 of this manual covers all the elements that can be included in an event definition. The following descriptions give detail about selected event topics that require more information.

**Symbolic references**

Symbol table values can be used throughout event system commands. The correct translations are performed for valid symbols to get actual symbol addresses, which can be used in defining events.

**Ranges**

You can specify an address range when defining a basic breakpoint (BI) or an event system event (EV). For EV commands, address ranges use a single address comparator. For BI commands, XICE handles ranges by breaking them into multiple individual single-point breakpoints. Thus, if you specify that a breakpoint should be for a range of 20 addresses, you may set only 12 additional breakpoints.
The syntax to specify a range is:

\[ \text{first_value..last_value} \]

**Example**

```c
ev(1)=addr==\text{0x03fa..0x03ff}
```

Event 1 is active when any address between 0x03FA and 0x03FF (inclusive) appears on the bus.

**Don't-care masks**

With the event system, you can also use don't care masks with data and LSA to further qualify a portion of a potential bus cycle for an event definition. The mask is a hexadecimal representation showing which bit positions are relevant and which are not relevant. You specify the don’t care mask value immediately following the LSA or data. Bits masked with a 0 will be ignored.

The syntax is:

\[ \text{value &=mask_value} \]

**Example**

```c
ev(1)\text{= data==0x3E13 &=0xFFFE}
```

Event 1 is active when 0x3E13 or 0x3E12 is seen on the data bus.

**Triggers**

A trigger is a list of up to 8 ORed event statement references, followed by one or more action definitions. If no action is specified, the default action, `break`, is assumed.

Triggers are associated with one of the four event system groups, as determined by which group is current when the trigger is defined. When the event system encounters
conditions specified in an event statement, it initiates the actions called for in armed triggers for the current group that refer to that event definition.

Multiple events in a trigger are ORed; the OR operator in this construction is the pipe (|). Multiple actions are ANDed; the AND operator is the comma (,). A comma (,) is also used in the trigger definition to separate the events from the actions.

You may store up to 16 trigger statements at any given time although the number that may be armed may be smaller depending on the demand for event resources.

**Example**

```
trig {1}=ev{1}|ev{32},rct,grp2
```

When event 1 is valid or event 32 is valid, reset the counter, and change to group 2.

The description of the TRIG command in Chapter 2 of this manual covers all the elements that can be included in a trigger.

**Disarming and rearming triggers**

Triggers are armed (active) when they are defined if the EVTARM switch is ON (the default) and there are sufficient event system resources available. The purpose of the ability to disarm triggers is to allow you to keep triggers that you are not using or that would cause the emulator to run out of event system resources. You can then reuse them without having to retype them.

You can turn this switch on and off with the EVTARM command. You can also set EVTARM by including the statement SW_EVTARM:ON or SW_EVTARM:OFF in your XICE.CFG file.

At the time you define the trigger, XICE checks the EVTARM switch. If it is on, XICE attempts to arm the trigger by setting its events and then the trigger itself in the emulator. If arming fails—if, for example there are not enough resources
available—the trigger will be defined but disarmed. See “Event system limitations” on page 3-24 for a summary of event system resource limits.

Triggers may be explicitly disarmed and rearmed using TRIG.

Example

trig(10)=disarm

Using groups
Changing groups activates alternate sets of events and triggers. When setting up triggers, change the current group manually using the EVTGRP command if your next trigger statement is to apply to another group.

Note
The GROUP command shows the group that the emulator last broke in, and the EVTGRP command shows the group that new event statements will be added to.

As an example of the common use of groups, you may wish to trace a subroutine after it has been called by module A or module B, but not if it has been called from modules C, D, or E. In this case, you would define a set of event statements to the address ranges of modules A and B. When either of these modules is encountered, switch to group 2 and look for the subroutine. After tracing the subroutine, switch back to group 1.

Event system breaks
The event system can use either basic or complex breakpoints. However, you must choose which type to use before invoking XICE. The switch SW_EVTMODE should be set to ON or OFF in the XICE.CFG file. The default is OFF.
When the SW_EVTMODE switch is OFF in the XICE.CFG file, you can set as many triggers with BREAK as an action as can be supported by hardware. However, because the emulator cannot determine which of these caused an event system break, macros attached to event system breakpoints will not work.

**Attaching macros to event system breakpoints**

A BREAKCOMPLEX (BC) breakpoint is an event system breakpoint that can initiate a macro.

When the SW_EVTMODE variable is ON, only one event system breakpoint may be armed at a time. In this case you may use a complex breakpoint (BC) in order to call a macro. The BC command will bind the trigger specified to the sole complex breakpoint and will appear in the XICE breakpoint window. The complex breakpoint is defined using the BC command to tie the break to a trigger.

---

**Example**

```plaintext
ev{1}=addr==0x0
trig{1}=ev{1}
bc trig{1};foo()
```

Event 1 is valid when 0 shows up on the address bus.

Trigger 1 is set for event 1.

Breaks and executes the macro “foo”

---

**Note**

It is illegal to specify BREAK directly in a trigger statement when SW_EVTMODE is ON.

---

**Note**

SW_EVTMODE must be ON; trigger 1 must be armed.
Event system breaks and trace

The event system provides a variety of switches that affect how the system captures trace. These are listed in table 3-2 and covered in the alphabetical command reference, Chapter 2. Each switch can be set in XICE.CFG or read directly from the command prompt.

<table>
<thead>
<tr>
<th>Switch</th>
<th>Description</th>
<th>Manual Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPT</td>
<td>Enable/Disable tracing of emulator peeks and pokes</td>
<td>S2-114</td>
</tr>
<tr>
<td>TED</td>
<td>Controls whether external DMA is traced</td>
<td>S2-139</td>
</tr>
<tr>
<td>TID</td>
<td>Controls whether internal DMA is traced</td>
<td>S2-141</td>
</tr>
<tr>
<td>TRCBRK</td>
<td>Controls the tracing of breakpoints</td>
<td>S2-138</td>
</tr>
<tr>
<td>TRCFRAME</td>
<td>Establish baseline traceframe number</td>
<td>S2-143</td>
</tr>
<tr>
<td>TRCINT</td>
<td>Display timestamps as INTERVAL/OFFSET</td>
<td>S2-144</td>
</tr>
<tr>
<td>TRCMODE</td>
<td>Display disassembled trace as ASM, SRC or BOTH</td>
<td>S2-145</td>
</tr>
<tr>
<td>TSTAMP</td>
<td>Controls whether timestamp or LSA is traced.</td>
<td>S2-153</td>
</tr>
<tr>
<td>TUNITS</td>
<td>Sets the timestamp units</td>
<td>S2-154</td>
</tr>
</tbody>
</table>

Event system limitations

XICE can keep track of up to 32 event statements and up to 16 trigger statements at the same time. Depending on the events and actions specified, different numbers of hardware resources are required by each trigger statement. In general, the emulator manages these resources and warns you when it makes adjustments and presents an error when resources are exhausted or when you attempt something that creates a conflict. So you need not concern yourself with more than the following general guidelines.
Statement limits
You can mix events (e.g. status, data, count...) in a single event statement. However, only one status, address, data, or LSA definition can be given in a single event statement. When you write the event statement, you can improve the flexibility by compounding resource references within the status, address, or data definition. Multiple status states can be ANDed within a status definition. Addresses and data can be given as ranges or qualified with don't care masks, but not both at the same time.

Example

```
ev(1)=addr==0x1234, data==0x0 & 0x0ff, status==byte | rd
```

Event 1 is valid when the address is 0x1234, and the least significant 8 bits of the data bus are 0x00, and read is asserted and access is by byte.

Trigger statement limits
Up to 8 event statements can be referred to in a trigger statement. (Multiple event statement references are ORed together.)

For a given XICE session, event system breaks (caused by a break action in a trigger definition) must be exclusively basic or exclusively complex. This is specified by the SW_EVTMODE switch in the XICE.CFG file. Basic breaks cannot be attached to macros, but you can specify a larger number of them. A complex break initiates a macro, but you are restricted to only one being armed at a time.

Hardware resource limits
You can free up event resources by disarming the triggers that refer to the event statements that employ those resources or by switching groups.

Events that are used by armed triggers use comparator hardware in the emulator.
The system keeps track of available resources and prompts you when you exceed the maximum. So you need not remember the specific number of event resources you have used. When you see “Limitation Error” or “Comparator Unavailable,” you have reached the maximum. You may choose to eliminate unneeded events or use the LOG command to create and save separate event system setups for use in separate sessions.

For BI, XICE handles ranges by breaking them into multiple individual single-point breakpoints. Thus, if you specify that an instruction breakpoint should be for a range of 20 addresses, you may set only 12 additional execution breakpoints.

**Single-stepping emulation**
The event system is not active during single-stepping in assembly-level mode. Trace will accumulate as you single step.
Chapter 5
Using Performance Analysis

The performance analysis features of XICE for the EL1600 Motorola 68000 and 68302 development system allow you to:

- determine which areas of a program use the most CPU time
- identify bottlenecks in time critical applications
- monitor the effects of programming changes made to improve throughput.

The performance analysis features are implemented using statistical performance analysis, which involves: sampling instruction fetch bus cycles at regular intervals using the event system; determining what function was active when a sample was taken; keeping a tally of the number of samples falling within each function; and reporting the sample information.

The report is in a user-specifiable format. The default format is:

- Function name
- Percentage of samples falling within that function
- A horizontal histogram showing this percentage in relation to other functions.

A sample report is shown below.

Hits used: 8186 (8186 total, 0 excluded)
FUNCTION       PERCENT
---------------------------
func9:      30.8 **************
func8:       17.1 ******
main:       16.6 ******
func7:       12.4 *****
func6:       11.1 ****
func5:        6.2 ***
func4:        3.1 *
func3:        1.6 *
func2:         0.8 *
func1:         0.3 *
Commands relating to performance analysis are listed in the table below. These commands are explained in more detail in Chapter 2 of this supplement.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERFFACT</td>
<td>Off: no performance analysis data is gathered. Statistical: trace is uploaded periodically from the emulator and processed. This flag is tested while in run.</td>
</tr>
<tr>
<td>PERFCLR</td>
<td>Removes all accumulated performance analysis data.</td>
</tr>
<tr>
<td>PERFDATA</td>
<td>Displays address range and number of hits associated with a symbol in the performance analysis display.</td>
</tr>
<tr>
<td>PERFDEPTH</td>
<td>Specifies maximum number of lines of performance analysis display output.</td>
</tr>
<tr>
<td>PERF DISP</td>
<td>Displays performance analysis information.</td>
</tr>
<tr>
<td>PERFEX</td>
<td>Allows convenient exclusion of address ranges and functions from performance analysis data (e.g. delay loops or functions).</td>
</tr>
<tr>
<td>PERF EX CLR</td>
<td>Clears exclusion(s) set with PERFEX.</td>
</tr>
<tr>
<td>PERF FORMAT</td>
<td>Determines display format of performance analysis information from the command line.</td>
</tr>
<tr>
<td>PERF FINT</td>
<td>Time interval in seconds between trace uploads from the emulator.</td>
</tr>
</tbody>
</table>
### Event system setup

To use performance analysis, you must set up the event system to wait a number of bus cycles, then capture the next instruction fetch cycle. An include file to set up the event system is supplied; it is also shown below. To use it, follow these steps:

1. Load the program file.
2. Include the file for event system setup (perf.inc in your bin directory).
3. Give the following commands:
   
   ```
   >go
   ```

   Approximately every three seconds, captured trace will be uploaded from the emulator, processed and displayed.

### Sample include file

Below is an include file suitable to set up the event system for performance analysis:

<table>
<thead>
<tr>
<th>PERFMODE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[Always</td>
<td>Demand]</td>
</tr>
<tr>
<td>----------------</td>
<td></td>
</tr>
</tbody>
</table>
| **Always**: Information is displayed whenever it is uploaded from the emulator.  
| **Demand**: Information is only displayed on command (PERFDISP). |

<table>
<thead>
<tr>
<th>PERFTOL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[distance]</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>Specifies maximum distance when searching for symbols.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sw_perffmt_stat:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[Standard</td>
<td>Percent</td>
</tr>
<tr>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>Determines display format of performance analysis information from XICE.CFG.</td>
<td></td>
</tr>
</tbody>
</table>
This file can be used to set up the EL-1600 68000 or 68302 event system for statistical performance analysis.

;remove any residual event system setup
evtclr

evtgrp 1

ev(1)=stat==0xffffffff&=0xffffffff
trig(1)=ev(1),trc

ev(2)=addr==0..0xffffffff
trig(2)=ev(2),cnt

; This timer value is the number of bus cycles that are allowed to elapse between grabs by the trace system. You will probably want to change it a few times during a P.A. run to assure that the number is not in sync with your program. If the period of this timer matches a periodicity in your program, very inaccurate results might occur.

ev(3)=count==1000
trig(3)=ev(3),rct,grp2

evtgrp 2

ev(4)=stat==proglrd
trig(4)=ev(4),trc,grp1

; Gather P.A. info during run.
perfact statistical

; data collection interval
perfint 3

; display P.A. data every time trace is uploaded.
perfmode always
Special considerations

XICE must halt the processor when uploading trace data. It does not take very long, but it could interfere with the operation of your program if you have timing constraints.

Performance analysis relies on all instruction fetches appearing on the bus. If you are using the processor’s cache, instructions may be fetched from the cache and performance analysis reports will be unreliable at best. For example, a tight loop which consumes 90% of all CPU cycles might appear on the bus only when it is entered (if it fits entirely in the cache), which would produce a very misleading performance analysis report.

Prefetch information is included as bus cycles are captured. This can add inaccuracies, since prefetch data often goes past the boundaries of a function.

Performance analysis uses the majority of event system resources, which means that you cannot do performance analysis and use a complicated event system setup at the same time. You might be able to share resources with the performance analysis setup, but you should check carefully to ensure that your additions do not interfere with the needs of the performance analysis setup.

Performance data and address range exclusions are automatically cleared when a load command is executed.

Limitations of statistical performance analysis

Statistical performance analysis can be inaccurate if the interval between samples matches a period within the program. For example, suppose you have the event system set up to sample every 1000 bus cycles (as in the include file above). Further suppose that there is a small routine in your program which is executed from a loop, and the length of the loop is roughly 1000 bus cycles. It could easily occur that almost
every sample would come from that same small routine, and hence that routine would appear to consume most of the CPU cycles even though it actually consumes much less than that.

The uploading of trace offers another opportunity for inaccuracy. For example, suppose you have XICE set up to upload trace data every three seconds (as in the include file above). Further suppose that you have a routine which is executed roughly every three seconds. That routine might never be sampled, if it always is executed in the period between when the trace buffer fills and when trace is uploaded. While a routine that is executed only every several seconds is unlikely to be a performance problem, the picture presented in the performance analysis reports will still be inaccurate.

For maximum accuracy, you should run performance analysis on your program several times, with different intervals between samples and uploads. Note that if you greatly increase the interval between samples, you may wish to increase the interval between uploads as well so you get a full trace buffer each time, although it will not cause problems if you do not. If you notice a drop in the number of samples collected between reports, this is what is happening.

**Exclusion of address ranges**

The ability to exclude address ranges allows you to discard data from functions and areas which are not of interest. For example, if you have a delay function, it may consume the majority of CPU cycles. This information might be interesting if your program should not be spending that much time in the delay function, but if you are interested in the performance of the rest of the program, it simply gets in the way.

The performance analysis system allows you to exclude address ranges in two different ways. The exclusion may be performed on the host or on the emulator. Each has advantages and disadvantages, and both may be used simultaneously to solve problems for which each is best suited.
The first way is via the `perfex` command. Its primary advantage is that any number of exclusions may be specified, covering arbitrary address ranges. Its primary disadvantage is that the exclusion is done on the host. For example, if you use `perfex` to exclude a delay function which consumes 90% of the CPU cycles, then roughly 90% of uploaded trace data will be discarded. This is highly wasteful. If additional exclusions are in force, all trace data in a given upload might be discarded.

The second way is by appending an event negating the range to `ev(10)` in the supplied event system setup include file (see above). The primary advantage of this method is that the exclusion is done on the emulator and resources are not wasted uploading trace data which falls within the excluded range. The primary disadvantage is that only one address range can be excluded in this manner. Note that if you use this method, you may wish to increase the interval between trace data uploads, since if (for example) you are excluding 90% of the potentially-sampled cycles, then it will take ten times as long to fill up the trace buffer.

You may use both methods at once. Suppose you have a delay function which consumes most of the CPU cycles, but wish to exclude other functions or ranges at the same time. You can use the event system to exclude the delay function and `perfex` to exclude the other functions. This setup means that resources are not wasted uploading thousands of samples which would be excluded, but at the same time you can exclude any number of functions or address ranges.
Chapter 6
Using the Time Stamp Module

This chapter describes the Time Stamp module and how to install and use it with the XICE debugger and the EL1600 emulator. Complete examples are provided for using the module to do each possible type of measurement. The sample raw trace display screens are for a Motorola 68302 processor; however, they are directly applicable to Motorola 68000 and Intel 80C18X family processors.
Overview

The Time Stamp feature is used for measuring time and for counting occurrences of events.

<table>
<thead>
<tr>
<th>Commands Used to Set Up Time Stamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>tstamp on/off</td>
</tr>
<tr>
<td>trcint offset/interval</td>
</tr>
<tr>
<td>tunits 0x0-0xf</td>
</tr>
<tr>
<td>trcframe n</td>
</tr>
<tr>
<td>ev(n)</td>
</tr>
<tr>
<td>trig(n)</td>
</tr>
</tbody>
</table>

Possible measurements

There are eight distinct measurements that can be made using the Time Stamp module. They are categorized into the two general groups shown below:

**Elapsed time measurements**
- Measure time spent in a module
- Measure time spent between modules
- Measure duration of time when memory is accessed (opcode or data)
- Measure duration of time when code is accessed (opcode only)
- Measure interrupt response time directly

**Count occurrences**
- Count number of times address or range of memory is accessed (opcode or data)
- Count number of times code is accessed (opcode only)
- Count module linkage activity (the number of times one module calls another)

Each time measurement is based on one of five scales: .1uS, 1uS, .01mS, .1mS or 1mS, which you specify. This allows you to collect your data using the appropriate time scale. The maximum number of counts for any time base is 65,535 so you have a maximum period of 65 seconds without overflow.

Time can be measured on an offset time frame, or on an interval time frame. When you use the offset time frame, the measurement is from the time the counter is reset and is centered around the raw trace frame selected by the XICE trace cycle number variable, `trcframe`.

When you use the interval time frame, the measurement is from one traced cycle to the next traced cycle. For example, if you were measuring the elapsed time for entering and exiting a module, the time displays would show as follows:

<table>
<thead>
<tr>
<th></th>
<th>Offset</th>
<th>Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>enter</td>
<td>0ns</td>
<td>700ns</td>
</tr>
<tr>
<td>exit</td>
<td>-700ns</td>
<td>700ns</td>
</tr>
<tr>
<td>enter</td>
<td>0ns</td>
<td>700ns</td>
</tr>
<tr>
<td>exit</td>
<td>-700ns</td>
<td>700ns</td>
</tr>
<tr>
<td>enter</td>
<td>0ns</td>
<td>700ns</td>
</tr>
<tr>
<td>exit</td>
<td>-700ns</td>
<td>700ns</td>
</tr>
</tbody>
</table>
Installation

Hardware installation

The Time Stamp module consists of the module itself and the cable to connect it to the emulator.

There are three steps to hardware installation:

1. Turn the emulator off.
2. Remove the front cover of the emulator.
3. Connect the module to the 40-pin connector on the trace/break board, as shown in the following illustration. Note that you cannot use the Logic State Analysis probe and the Time Stamp module at the same time.
Before plugging in the cable, turn off power to the EL1600 emulator to prevent damage to the cable and module. Do not plug in or unplug the Time Stamp module with emulator power turned on.
Using the Time Stamp module

This section defines the labels, buttons, switches and LEDs on the Time Stamp module, and provides information on how the unit works.

Figure 6-2 Time Stamp module

Getting started

Figure 6-3 shows the end of your Time Stamp module including the locations of the trigger inputs, reset button, switch and overflow indicator LED.
The TGR input measures interrupt latency directly. The TGR input connects directly to the interrupt line in your target circuit to avoid any logic delays due to use of the Advanced Event System. It is designed for processors that pull lines low for interrupts. (Motorola) (see page 22)

The TGR input measures interrupt latency directly. The TGR input connects directly to the interrupt line in your target circuit to avoid any logic delays due to use of the Advanced Event System. It is designed for processors that pull lines high for interrupts. (Intel processors) (see page 22)

The reset button resets the time stamp counter to 0.

The switch determines the time base and counting type. (see page 10)

The overflow LED indicates when the counter overflows the 65,535 limit.
The examples that follow give information on when to use the manual reset button, TGR and TGR, and how to use the switch to choose the time stamp mode and time base.

---

**Warning**

Do not plug in or unplug the Time Stamp module when power is turned on to the emulator.

---

**Making a measurement**

The basic steps to make a measurement are as follows:

1. Set up the XICE variables for time stamp. Set the XICE variable `tstamp` on to display timestamp information in raw trace. Set the XICE `trcint` and `trcframe` variables to the appropriate values for the measurement you want to make.

2. Choose a switch setting on the Time Stamp module and set the XICE variable `tunits` to the same value as the switch.

3. Set up the trigger inputs.

4. Set up the Advanced Event System to trigger the Time Stamp module at the appropriate program states.

5. Run your program.

6. View the time stamp information.

7. Interpret the time stamp information.

Each step is described in detail below.
Step 1: Set up XICE timestamp variables

The XICE variables tstamp, trcint, tunits, and trcframe control the raw trace display of information coming in on the LSA/time stamp port. Settings tstamp on, trcint offset and interval, tunits, and trcframe are used with the Time Stamp option. Setting tstamp off is used with the LSA pod.

- **tstamp off**
  Default: LSA value shown as 16 bits

- **tstamp on, trcint offset**
  Display the offset time value

- **tstamp on, trcint interval**
  Display the interval time value

- **tunits 0x0-0xf**
  Select timestamp time base (use same value as switch)

- **trcframe n**
  Center timestamp display around trace frame number n

Offset time values are used when you want to measure the total amount of time spent or the number of occurrences. Interval time values are used when you are interested in the time spent between points A and B in your code, but are not interested in how long it takes to get to point A.

You may select values by editing the xice.cfg file or by using the following commands within XICE:

To select timestamp display in raw trace, enter:

```
:tstamp on.
```

To select display mode offset or interval, enter:

```
:trcint x (where x is offset or interval)
```

To select raw trace frame number for centering timestamp display in interval mode:

```
:trcframe n (where n is the raw trace frame number)
```
Step 2. Set up Time Stamp module switch
The time base for the timestamp is controlled by setting a switch on the Time Stamp module and setting the same value for the XICE timestamp units variable, tunits.

Choose a switch setting on your Time Stamp module based on your measurement type and preferred time base. We recommend starting with the slowest time frame: 1 mS. The table below shows the maximum measurable time period for each switch setting.

<table>
<thead>
<tr>
<th>Time Base</th>
<th>Maximum Measurable Time Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 uS</td>
<td>6.5 milliseconds</td>
</tr>
<tr>
<td>1.0 uS</td>
<td>65 milliseconds</td>
</tr>
<tr>
<td>.01 mS</td>
<td>.65 second</td>
</tr>
<tr>
<td>0.1 mS</td>
<td>6.5 seconds</td>
</tr>
<tr>
<td>1.0 mS</td>
<td>65 seconds</td>
</tr>
</tbody>
</table>

If the counter overflows, the yellow overflow LED will be lit. Check to see if you are using the correct time base for the duration of your measurements. When the counter overflows the 65,355 limit, it starts again at 0.

When the emulator is paused, no TGR is generated by the Advanced Event System in positions 0-4, so the counter is not reset and is likely to overflow. This is not a problem.
For example, the **drt** display might be as follows.

<table>
<thead>
<tr>
<th>FRAME</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>IPL</th>
<th>FCn</th>
<th>MEM</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>#20</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>700ns</td>
</tr>
<tr>
<td>#19</td>
<td>001100</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>700ns</td>
</tr>
<tr>
<td>#18</td>
<td>001124</td>
<td>754B</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>700ns</td>
</tr>
<tr>
<td>#17</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>700ns</td>
</tr>
</tbody>
</table>

The following table summarizes the switch positions.

The trigger to start and stop the counter in the Time Stamp module is either the TGR signal from the Advanced Event System (Step 4), or the TGR or TGR direct input from your target interrupt line (Step 3).

<table>
<thead>
<tr>
<th>Position</th>
<th>Time Base</th>
<th>Effect of TGR on Time Stamp Counter</th>
<th>Useful Measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.1 uS</td>
<td>Any TGR high causes the time stamp</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>1</td>
<td>1 uS</td>
<td>counter to be reset to 0. No manual</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>.01 mS</td>
<td>reset is required in this mode for either</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.1 mS</td>
<td>Offset or interval time stamping.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1 mS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.1 uS</td>
<td>While the TGR is held high by the</td>
<td>Elapsed time</td>
</tr>
<tr>
<td>6</td>
<td>1 uS</td>
<td>Advanced Event System, the time stamp</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>.01 mS</td>
<td>counter counts. Manual reset is required</td>
<td></td>
</tr>
<tr>
<td>No.</td>
<td>Time (μs)</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>.1 mS</td>
<td>In this mode for offset time stamping,</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1 mS</td>
<td>But not for interval time stamping.</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>.1 uS</td>
<td>In this mode, a long TGR signal(^1) from advanced event system resets the</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1 uS</td>
<td>The Advanced Event System resets the counter.</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>.01 mS</td>
<td>Count occurrences. The counter resets manually.</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>.1 mS</td>
<td>Reset stops the counter and sets it to zero.</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1 mS</td>
<td>This setting is used to count occurrences.</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) A long TGR is defined as being longer than 1.6 uS. This is the only mode where the length of the TGR matters. The following diagram shows what happens to the counter depending on the TGR signal.
Figure 6-4 Positions A-E: Effects of multiple TGR signals

**Step 3. Set up TGR input**

The counter in the Time Stamp module can be controlled in one of three ways:

1. The Advanced Event System TGR action.
2. The TGR input.
3. The TGR input.

The default is the Advanced Event System trigger input. No additional wires are necessary.

To use the TGR and TGR lines to measure interrupt latency, you must connect one of these lines to an interrupt line on your target. Use of the TGR and TGR external inputs is described fully in the example on page 22.

**Step 4. Set up the Advanced Event System**

Set up the Advanced Event System to selectively trace the memory, program activity, or modules you are interested in time stamping. You can set up the Advanced Event System through the XICE command line or by using an XICE “include” command file.
The three steps to set up the Advanced Event System follow:

1. Decide what condition you want to look at, and what actions to take when that condition is reached.
2. Set up the event statements to isolate that condition.
3. Set up trigger statements using the appropriate events and actions.

For more information on using the Advanced Event System, please see Chapter 4 of this manual. The examples beginning on page 15 provide examples of using the Advanced Event System to specify conditions appropriate for time stamping.

**Step 5. Run your program**
Run XICE using the `go` command.

**Step 6. View Time Stamp information**
Displaying the time stamp information follows:

First enter XICE command line mode by either:

- stopping emulation with the `<Control-c>` command
- using the Advanced Event System to break emulation

Then view the trace, using the `drt` command.

The last column of the raw trace display shows either offset or interval time stamp, depending on the setting you specified with the `trcint` command.

In offset mode, use the `trcframe n` variable to select a raw trace frame number to center the timestamp display around.

**Step 7. Interpret Time Stamp information**
The time stamp information is displayed in either offset mode or interval mode depending on the setting of the XICE `trcint` variable.

In offset mode, timestamps are the time relative to the raw trace frame number specified by the `trcframe` command.
In the interval mode, timestamps are displayed as the time interval between successive bus cycles.

Collecting time stamp information in a file

After setting up your Advanced Event System and Time Stamp module to provide just the information you need, you can use the XICE journal command to save the specific drt displays to an ASCII file. You can use a spreadsheet or data base management program to analyze the data stored in the file.

Below is an example demonstrating saving 42 frames of raw trace to a file named trace.raw.

1. jou on="trace.raw"
2. drt 0..42
3. jou off

Examples

The two basic measurement modes are "elapsed time" and "counting occurrences". The examples are organized as follows:

Measuring elapsed time
- measure the time it takes to go from event A to event B
- measure the time the program is in the specified range
- measure the time between an interrupt and interrupt servicing

Counting occurrences
- count the number of times the program transitions from event A to event B
- count the number of accesses to a memory location or range
Measuring elapsed time
The elapsed time measurement is used to measure in-module time, out-of-module time, inter-module time, and memory and program access time. These measurements use switch positions 0 to E. You must set the XICE variable tunits to be the same value as selected with the Time Stamp module switch.

Conceptually, there are three types of elapsed time measurements:

1. Measuring the time from event A to event B
   - used for measuring program time, out-of-module execution time, and inter-module execution time
2. Measuring the time spent in an address range
   - used for measuring memory time and program time (excluding calls to other modules)
3. Measuring the time between an interrupt and interrupt servicing
   - used for measuring interrupt latency

A to B mode
To measure the time it takes a program to get from event A to event B, set up the Advanced Event System so only event B appears in the trace display.

Step 1. Set LSA display type
   >trcint offset (Set display format to offset time stamp)

Step 2. Select Time Stamp module switch setting
Use positions 0-4, depending on your preferred time base. In positions 0-4, the TGR from the Advanced Event System resets the time stamp counter to 0.
Using the Time Stamp module

If you're not sure which time base to use, use position 4 for the slowest. If the counter overflows, the yellow overflow LED will light. See “Step 2. Set up Time Stamp module switch” on page 10 for a chart of maximum time periods per setting.

>`tunite 0xN (Set N to same value as the switch)

**Step 3. Set up the trigger input**

To measure elapsed time, use the Event System trigger input.

**Step 4. Set up the Advanced Event System**

```plaintext
ev{1}=addr=="A" (Specify event 1 in group 1 to be event A)

ev{2}=addr=="B" (Specify event 2 in group 1 to be event B)

trig{1}=ev{1},tgr,trc (Reset the time stamp counter to 0 and trace one cycle at event A)

trig{2}=ev{2},trc (Trace event B)
```

**Step 5. Run your program**

>`go Run program`

**Step 6. Stop emulation**

`<Control-c>` Stop emulation

**Step 7. View time stamp data**

```plaintext
drt Display the trace
```
Step 8. Interpret time stamp information

The last column of the trace display shows the offset time stamp information. Note that if event A and B are called more than once, you will get the time between events for each occurrence.

The following screen shows the raw trace display. The Time Stamp module switch was set to position #0 (.1 uSec). The time to go from A to B is shown to be 700 nSec.
Table showing data for FRAME, ADDRESS, DATA, IPL, FCn, MEM, DMA, IAC, FLAGS, and TIMESTAMP:

<table>
<thead>
<tr>
<th>FRAME</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>IPL</th>
<th>FCn</th>
<th>MEM</th>
<th>DMA</th>
<th>IAC</th>
<th>FLAGS</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>40</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>39</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>38</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>37</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>36</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>35</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>34</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>33</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>32</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>31</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>30</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>29</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>28</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>27</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>26</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>25</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>24</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>23</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>22</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>21</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>20</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>19</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>18</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>17</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>16</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>15</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>14</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>13</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>12</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>11</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>10</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>9</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>8</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>7</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>6</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>5</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>4</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>3</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>700ns</td>
</tr>
<tr>
<td>2</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0ns</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6-5 Sample DRT screen for measuring time from A to B
Range Mode
In range mode, the trace display shows the amount of time the program is in the specified range.

Press the module's manual reset button prior to performing this measurement.

Step 1. Set LSA display type

\[
\text{trcint offset} \quad \text{Set display format to offset time stamp}
\]

Step 2. Select Time Stamp module switch setting

Use positions 5-9, depending on your preferred time base. In these positions, the Advanced Event System TGR enables the counter.

If you're not sure which time base to use, use position 9 for the slowest. If the counter overflows, the yellow overflow LED will light. See page 10 for a chart of maximum time periods per setting.

\[
\text{tunits 0xN} \quad \text{Set N to same value as the switch}
\]

Step 3. Set up the trigger input

To measure elapsed time, use the Event System Trigger input.

Step 4. Set up the Advanced Event System

\[
\text{ev(1)=addr=="range"} \quad \text{Set event 1 to be the specified address range}
\]
\[
\text{ev(2)=addr!="range"} \quad \text{Set event 2 to be outside the specified address range}
\]
\[
\text{trig(1)=ev(1),tgr,grp2} \quad \text{When range is accessed, enable counter and go to group 2}
\]
Using the Time Stamp module

Step 5. Run your program

go

Run program

Step 6. Stop emulation

<Control-c>

Stop emulation

Step 7. View time stamp data

drt

Display the trace

Step 8. Interpret time stamp information

The last column of the trace display shows the amount of time accumulated while the program was in the specified range.

The following screen shows the raw trace display, for the above example using a range of 0x1100 to 0x1124. The Time Stamp module switch was set to position #5 (0.1 uSec). The time spent in this range was 5.4 uSec.
<table>
<thead>
<tr>
<th>FRAME</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>IPL</th>
<th>FCn</th>
<th>MEM</th>
<th>DMA</th>
<th>IAC</th>
<th>FLAGS</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>BEGINNING OF TRACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>BREAK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>00100A</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>00100C</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>00100E</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>001010</td>
<td>1000</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>001008</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>001002</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>001004</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>001006</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>001008</td>
<td>1100</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>001102</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.300us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>001104</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-5.000us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>001106</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-4.700us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>001108</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-4.400us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>00110A</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-4.100us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>00110C</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-3.800us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>00110E</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-3.500us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>001110</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-3.100us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>001112</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-2.800us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>001114</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-2.500us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>001116</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-2.200us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>001118</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-1.900us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00111A</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-1.600us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>00111C</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-1.300us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>00111E</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-1.000us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>001120</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-600ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>001122</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>-300ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>0ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>001126</td>
<td>100A</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>X 300ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>00100A</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>X 700ns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>00100C</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td>X 1.000us</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>BREAK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6-6 Sample DRT screen for measuring time in range

**Interrupt latency**

To measure the amount of time between the detection of an interrupt and when it is serviced, connect your target interrupt line directly to the TGR or TGR lines on the Time Stamp module. As shown in Figure 6-7, these lines perform exactly
the same function as the Advanced Event System TGR signal, but the direct trigger bypasses the delays inherent in going through the additional Advanced Event System logic.

Figure 6-7 Trigger input logic

Figure 6-8 shows the trigger pattern for the TGR and TGR inputs.
Step 1. Set LSA display type

trcint offset                Set display format to offset time stamp

Step 2. Select Time Stamp module switch setting

Use positions 0-4, depending on your preferred time base. In positions 0-4, the TGR from the external TGR, external TGR or Advanced Event System TGR resets the time stamp counter to 0.
If you’re not sure which time base to use, use position 4 for the slowest. If the counter overflows, the yellow overflow LED will light. See page 10 for a chart of maximum time periods per setting.

\[ \text{tunits } 0xN \]
\[
\text{Set N to same value as the switch}
\]

**Step 3. Set up the trigger input**

Connect either the TGR or TGR input on the Time Stamp module to the interrupt line on your target that you want to check. For example, to check the interrupt latency for interrupt INT0 on the 80C186, use the setup shown in Figure 6-9.

Figure 6-9 Target setup for measuring interrupt latency
Step 4. Set up the Advanced Event System

\[ ev(1) = \text{addr} == \text{intservice\_start} \]
Specify event 1 in group 1 to be the start of the interrupt service routine

\[ \text{trig}\{1\} = ev\{1\}, \text{trc} \]
Start tracing at the beginning of the interrupt service routine

Step 5. Run your program

\text{go}
Run program

Step 6. Stop emulation

\text{<Control-c>}
Stop emulation

Step 7. View time stamp data

\text{drt}
Display the trace

Step 8. Interpret time stamp information

The Advanced Event System traces the first cycle of the interrupt service routine. The last column of the trace display shows the amount of time elapsed between the start of the interrupt service routine and the actual interrupt processing.

**Counting occurrences**
The number of occurrences measurement is used to measure memory and program activity, module linkage activity and program flow activity. Use switch position F (count TGR pulses) for all counting measurements.
Using the Time Stamp module

Two types of counting occurrences measurements follow:

1. Counting the number of times the program transitions from event A to event B
   - used for measuring module linkage activity

2. Counting the number of accesses to some memory location(s).
   - used for measuring memory program activity

**A to B Mode**
This mode records the number of times the transition from event A to event B occurs. Trace is only recorded on exit from module B. Press the module's manual reset button prior to performing this measurement.

**Step 1. Set LSA display type**

- trcint offset
  - Set display format to offset timestamp

**Step 2. Select Time Stamp module switch setting**

Use timestamp module switch position F. For counting occurrences, the time base is irrelevant. In position F, when the TGR from the Advanced Event System goes high, the time stamp counter increments.

- tunits 0xf
  - Set timestamp to count occurrences

**Step 3. Set up the trigger input**

To count occurrences, use the Event System Trigger input.
Step 4. Set up the Advanced Event System

\[ ev(1) = \text{addr} == \text{start}_a \]
Specify event 1 in group 1 to be the start of module A

\[ ev(2) = \text{addr} == \text{start}_b \]
Specify event 1 in group 2 to be the start of module B

\[ ev(3) = \text{addr} == \text{end}_b \]
Specify event 2 in group 2 to be the end of module B

\[ \text{trig}(1) = \text{ev}(1), \text{grp}2 \]
Go to group 2 when module A is accessed

\[ \text{evtgrp} 2 \]
Begin arming triggers for group 2

\[ \text{trig}(2) = \text{ev}(2), \text{tgr} \]
Increment counter when entering module B from module A

\[ \text{trig}(3) = \text{ev}(3), \text{trc,grp}1 \]
Exit module B, record count in trace memory

Step 5. Run your program

\[ \text{go} \]
Run program

Step 6. Stop emulation

\[ <\text{Control-c}> \]
Stop emulation

Step 7. View time stamp data

\[ \text{trcframe} 2048 \]
Set trace frame variable to end of trace

\[ \text{drt} \]
Display the trace
Step 8. Interpret time stamp information

The last column shows the number of times module B was entered from module A. Note that only the locations `start_a` (1000) and `end_b` (1124) are traced. In the following screen we see that module B is called once each time from module A. The total number of calls is 0x3A31.
<table>
<thead>
<tr>
<th>Frame</th>
<th>Address</th>
<th>Data</th>
<th>IPL</th>
<th>FCn</th>
<th>MEM</th>
<th>DMA</th>
<th>IAC</th>
<th>Flags</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A1E</td>
</tr>
<tr>
<td>40</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A1F</td>
</tr>
<tr>
<td>39</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A1F</td>
</tr>
<tr>
<td>38</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A20</td>
</tr>
<tr>
<td>37</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A20</td>
</tr>
<tr>
<td>36</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A20</td>
</tr>
<tr>
<td>35</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A21</td>
</tr>
<tr>
<td>34</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A21</td>
</tr>
<tr>
<td>33</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A22</td>
</tr>
<tr>
<td>32</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A22</td>
</tr>
<tr>
<td>31</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A23</td>
</tr>
<tr>
<td>30</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A23</td>
</tr>
<tr>
<td>29</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A24</td>
</tr>
<tr>
<td>28</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A24</td>
</tr>
<tr>
<td>27</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A25</td>
</tr>
<tr>
<td>26</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A25</td>
</tr>
<tr>
<td>25</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A25</td>
</tr>
<tr>
<td>24</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A26</td>
</tr>
<tr>
<td>23</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A27</td>
</tr>
<tr>
<td>22</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A27</td>
</tr>
<tr>
<td>21</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A28</td>
</tr>
<tr>
<td>20</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A28</td>
</tr>
<tr>
<td>19</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A29</td>
</tr>
<tr>
<td>18</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A29</td>
</tr>
<tr>
<td>17</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2A</td>
</tr>
<tr>
<td>16</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2A</td>
</tr>
<tr>
<td>15</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2B</td>
</tr>
<tr>
<td>14</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2B</td>
</tr>
<tr>
<td>13</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2C</td>
</tr>
<tr>
<td>12</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2C</td>
</tr>
<tr>
<td>11</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2D</td>
</tr>
<tr>
<td>10</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2D</td>
</tr>
<tr>
<td>9</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2E</td>
</tr>
<tr>
<td>8</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2E</td>
</tr>
<tr>
<td>7</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2F</td>
</tr>
<tr>
<td>6</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A2F</td>
</tr>
<tr>
<td>5</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A30</td>
</tr>
<tr>
<td>4</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A30</td>
</tr>
<tr>
<td>3</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A31</td>
</tr>
<tr>
<td>2</td>
<td>001000</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>3A31</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BREAK</td>
</tr>
</tbody>
</table>

Figure 6-10 Sample DRT screen for counting occurrences
Range Mode
This mode records the number of accesses to some memory location(s). Trace is always recorded. The last trace cycles recorded show the accumulated access counts. Press the module's manual reset button prior to performing this measurement.

Step 1. Set LSA display type

trcint offset
Set display format to offset timestamp

Step 2. Select Time Stamp module switch setting

Use timestamp module switch position F. For counting occurrences, the time base is irrelevant. In this position, when the TGR from the Advanced Event System goes high, the time stamp counter increments.

tunits 0xf
Set timestamp to count occurrences

Step 3. Set up the trigger input

To count accesses, use the Event System Trigger input.

Step 4. Set up the Advanced Event System

ev{1}=addr=="range"
Specify the range to be monitored

trig(1)=ev{1},tgr
Increment counter whenever range is accessed

Step 5. Run your program

go
Run program
Step 6. Stop emulation

<Control-c> Stop emulation

Step 7. View time stamp data

trcframe 2048 Set trace frame variable to end of trace
drt Display the trace

Step 8. Interpret time stamp information

The last column of the last line of the trace display gives you the number of times the range was accessed. In the following sample screen, the range is set from 0x1100 to 0x1124. The range was accessed 4 times.
<table>
<thead>
<tr>
<th>FRAME</th>
<th>ADDRESS</th>
<th>DATA</th>
<th>IPL</th>
<th>FCn</th>
<th>MEM</th>
<th>DMA</th>
<th>IAC</th>
<th>FLAGS</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>001122</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0002</td>
</tr>
<tr>
<td>40</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0002</td>
</tr>
<tr>
<td>39</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>38</td>
<td>001102</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>37</td>
<td>001104</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>36</td>
<td>001106</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>35</td>
<td>001108</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>34</td>
<td>00110A</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>33</td>
<td>00110C</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>32</td>
<td>00110E</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>31</td>
<td>001110</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>30</td>
<td>001112</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>29</td>
<td>001114</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>28</td>
<td>001116</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>27</td>
<td>001118</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>26</td>
<td>00111A</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>25</td>
<td>00111C</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>24</td>
<td>00111E</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>23</td>
<td>001120</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>22</td>
<td>001122</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>21</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0003</td>
</tr>
<tr>
<td>20</td>
<td>001100</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>19</td>
<td>001102</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>18</td>
<td>001104</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>17</td>
<td>001106</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>16</td>
<td>001108</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>15</td>
<td>00110A</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>14</td>
<td>00110C</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>13</td>
<td>00110E</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>12</td>
<td>001110</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>11</td>
<td>001112</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>10</td>
<td>001114</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>9</td>
<td>001116</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>8</td>
<td>001118</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>7</td>
<td>00111A</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>6</td>
<td>00111C</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>5</td>
<td>00111E</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>4</td>
<td>001120</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>3</td>
<td>001122</td>
<td>4E71</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>2</td>
<td>001124</td>
<td>4EF8</td>
<td>000</td>
<td>SP</td>
<td>RWO</td>
<td></td>
<td></td>
<td></td>
<td>0004</td>
</tr>
<tr>
<td>1</td>
<td>Break</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 6-11** Sample DRT screen counting occurrences in a range
Chapter 7

Simulated I/O

During the development of a system, it is often desirable to simulate input functions with canned input from a file or interactive input from the keyboard. Likewise, it is useful to be able to capture output functions and put the results in a file or display them on a screen.

This chapter explains how to use simulated I/O with XICE for character I/O. It also covers using simulated I/O with XRAY.

See your XRAY manual for documentation of the INPORT and OUTPORT commands.

Using simulated character input with XICE

Simulated character input is available for the EL 1600. The three steps required to do simulated character input are described below.

1. Start the XICE program. Activate an input trap by setting a breakpoint:

   In low-level mode, set a breakpoint immediately before the read of the location _simulated_input in the INCHRW routine inside the inchrw.s or inchrw.src file. The command to do so is as follows:

   bi INCHRW\inchrw;inport(_simulated_input,1)
2. Define the stream to be used for input.
   You may use the keyboard or set up a file as the input stream for the simulated input.
   
The command to set up input from the keyboard is as follows:
   
   INPORT &_simulated_input,std

   The command to set up input from a file is as follows:
   INPORT &_simulated_input,f=file_name

   You may need to enter other INPORT commands if you wish additional input streams.

3. Type go to begin processing your file. XICE performs the following tasks as it processes your file:
   
   □ XICE instructs the emulator to select a suitable hardware or software breakpoint type.
   □ The code emulation begins.
   □ When the break instruction point is reached, the emulator stops.
   □ XICE then waits for one byte to be available from the input channel you specified using the INPORT command.
   □ Next, XICE internally transfers the byte read to the location _simulated_input.
   □ The emulator single-steps over the original program code word (restored for a software breakpoint, if necessary).
   □ The code emulation continues.
Using simulated character output with XICE

Simulated character output is available for EL 1600. There are three steps required to do simulated character output which are described below.

1. Start the XICE program. Activate an output trap by setting a breakpoint:

   In low-level mode, set a breakpoint immediately after the write to the location _simulated_output in the OUTCHR routine inside the outchr.c file. The command to do so is as follows:

   \texttt{bi OUTCHR\_sim_out_brk;outport(&\_simulated_output, 1,\_simulated_output)}

2. Define the stream to be used for output.

   You may use a viewport or set up a file as the destination for the simulated output.

   The command to set up output to the standard I/O window is as follows:

   \texttt{OUTPORT &\_simulated_output,std}

   The command to set up output to a file is as follows:

   \texttt{OUTPORT &\_simulated_output,f=file\_name}

   You may need to enter other \texttt{OUTPORT} commands if you wish additional destinations.

3. Type \texttt{go} to begin processing your file. XICE performs the following tasks as it processes your file:

   - XICE instructs the emulator to select a suitable hardware or software breakpoint type.
   - The code emulation begins.
   - When the break-instruction point is reached (_\texttt{sim_out_brk}), the emulator stops.
- The emulator automatically restores the original program code word if you used a software breakpoint.
- Next, XICE internally transfers the byte that was written to `simulated_output` to the port assigned via the command `OUTPORT`.
- The emulator then single-steps over the original program code word (restored for a software breakpoint, if necessary).
- The code emulation continues.
Using simulated character input with XRAY

The two steps required to do simulated character input using XRAY are described below.

1. Start the XRAY program. Define the stream to be used for input. You may use either the keyboard or set up a file as the input stream for the simulated input.

   The command to set up input from the keyboard is as follows:
   
   `INPORT &_simulated_input, std`

   The command to set up input from a file is as follows:
   
   `INPORT &_simulated_input, f=file_name`

   You may need to enter other INPORT commands if you wish additional streams for the input.

2. Type `go` to begin processing your file. When the access break occurs due to a read from the `_simulated_input` address, XRAY reads the simulated input from either the keyboard or from the file you specified, depending upon what you specified in the INPORT command.

   XRAY can interrupt an instruction execution and redirect accesses in mid-instruction. For example, if the instruction

   `MOVE.B _simulated_input,D0`

   appears in the instruction flow, XRAY detects an access to `_simulated_input` and replaces it with the data from the keyboard or file depending on which you specified in the INPORT command. XRAY then substitutes the byte from the simulated input into the register D0, and continues to run.
Using simulated character output with XRAY

The two steps required to do simulated character output using XRAY are described below.

1. Start the XRAY program. Define the viewport or file to be used for output.

   You may use any legal XRAY viewport for simulated output. You may also set up a file as the destination for the simulated output.

   The command to set up a viewport for simulated output is as follows:
   
   ```
   OUTPORT &_simulated_output, std
   ```

   The command to set up output to a file is as follows:
   
   ```
   OUTPORT &_simulated_output, f=file_name
   ```

   You may need to enter other outport commands if you wish to direct the simulated output to other places as well.

2. Type go to begin processing your file. When the access break occurs due to a read from the _simulated_output address, XRAY detects this special instance and writes instead the simulated output to either the viewport or the file you specified.

   XRAY can interrupt an instruction execution and redirect accesses in mid-instruction. For example, if the instruction
   
   ```
   MOVE.B D0, _simulated_output
   ```

   appears in the instruction flow, XRAY detects an access to _simulated_output and writes the data to the viewport or file depending on which you specified in the OUTPORT command. XRAY then continues to run.
The Master Index contains the indexes for the full documentation set related to the Applied toolchain and page numbering has been set up for this purpose. Numbers beginning with an R refer to reference manuals. Numbers beginning with a U refer to user guides. Numbers beginning with an I refer to installation guides. Lettered sections refer to appendices within manuals. For example, RA- refers to appendix A of a reference manual. The document set to which the manual belongs appears in parenthesis following the page number reference. The following abbreviations are used:

- ASM - ASM Documentation Set
- MCC - MCC Documentation Set
- CCC - CCC Documentation Set
- XRAY - XRAY Documentation Set
- SUP - XICE Supplement for the 68020, 68030, 68EC030 and the EL 3200
- FLEX - Flexible License Manager Documentation Set

68000, 68EC000, 68HC000, 68302 Documentation Set Master Index

- notation R2-12(XRAY)
- unary operator R3-8(MCC)
- unary operator R3-8(MCC)

Symbols
# operator R4-17(MCC)
# unary operator R3-8(MCC)
# unary operator R3-8(MCC)
- command line continuation character U2-21(ASM)
#line directive R4-1(MCC)
#line_number column R2-12(XRAY)
#line_number statement R2-12(XRAY)
#pragma options R6-37(MCC)
$ R2-10(ASM), R2-12(ASM)
% R2-12(ASM)
%(line continuation) R3-5(XRAY)
% line continuation U2-9(XRAY)
% operator R4-17(MCC)
& operator R3-3(MCC), R4-17(MCC)
& unary operator R3-8(MCC)
&& operator R4-17(MCC)
( ) R13-1(ASM)
(misaligned) U3-11(XRAY)
* R5-11(ASM), R13-1(ASM), R13-2(ASM)
* operator R4-17(MCC)
* unary operator R3-8(MCC)
+ R7-14(ASM), R13-2(ASM)
+ operator R4-17(MCC)
+ unary operator R3-8(MCC)
++ unary operator R3-8(MCC)
+e option, use of R3-18(CCC)
+ne option, use of R3-18(CCC)
, R13-1(ASM)
- operator R4-17(MCC)
-. operator R2-17(CCC), R2-19(CCC), R2-21(CCC), R2-25(CCC)
.* operator R2-18(CCC), R2-25(CCC)
.login U1-1(FLEX)
.login file
(see UNIX start-up file)
.profile U1-1(FLEX)
(see UNIX start-up file)
.Xdefaults file UA-6(XRAY)
/ operator R4-17(MCC)
/optimize=noglobalflow option U3-26(MCC)
/optimize=noinline option U3-26(MCC)
/optimize=nolocal option U3-26(MCC)
/optimize=nostablemem option U3-27(MCC)
/optimize=register option U3-26(MCC)
/optimize=reorder option U3-27(MCC)
/optimize=singleret option U3-27(MCC)
/optimize=size option U3-28(MCC)
/optimize:stablemem option U3-27(MCC)
/transform:reorder option U3-27(MCC)
/transform:singleret option U3-27(MCC)
/transform:size option U3-28(MCC)
/transform:stablemem option U3-27(MCC)
/transform:time option U3-28(MCC)
/pagelength option U3-28(MCC)
/pp option U3-29(MCC)
/prepend=dot option U3-12(MCC)
/prepend=underscore option U3-12(MCC)
/preserve_comments option U3-29(MCC)
/print_options option U3-30(MCC)
/quit option U3-29(MCC)
/rename option U3-30(MCC), U3-31(MCC)
/reserve option U3-14(MCC)
/show=include option U3-28(MCC)
/show=noinclude option U3-28(MCC)
/show=nosource option U3-29(MCC)
/show=source option U3-29(MCC)
/spath option U3-32(MCC)
/ssmultiple option R6-27(MCC), R6-31(MCC), R6-35(MCC), U3-11(MCC)
/strings_addr_as=code option U3-16(MCC)
/strings_addr_as=data option U3-16(MCC)
/suppress=error option U3-30(MCC)
/suppress=informational option U3-30(MCC)
/suppress=summary option U3-30(MCC)
/syntax_only option U3-33(MCC)
/title option U3-29(MCC)
/tmp U2-3(FLEX), UA-2(FLEX)
/truncate_identifiers option U3-12(MCC)
/truncate_identifiers option (VMS) R4-3(MCC)
/undefine option U3-33(MCC)
/unsignedchar option U3-22(MCC)
/unsignedchar option (VMS) R6-10(MCC)
/usr/local/flexlm/licenses/license.dat (see License file)
/usr/local/licenses.dat IC-1(FLEX)
/usr/mri 11-10(FLEX), UA-2(FLEX) (see also install_dir)
/usr/mri/bin U1-1(FLEX)
adding to path I2-1(FLEX)
creating scripts in I2-2(FLEX)
mixed architecture installation IA-1(FLEX)
/usr/mri/sun3 IA-1(FLEX)
/usr/mri/sun4 IA-1(FLEX)
/usr/tmp l2-9(FLEX)
/usr/tmp/license.log (see Log file)
error messages UC-1(FLEX)
/weakextern=common option U3-34(MCC)
/weakextern=initzero option U3-34(MCC)
/weakextern=public option U3-34(MCC)
: macro prompt U2-26(XRAY)
:: operator R2-25(CCC)
::* operator R2-18(CCC)
; R13-1(ASM), R13-2(ASM)
< operator R4-17(MCC)
<= operator R4-17(MCC)
=> operator R2-18(CCC)
=>* operator R2-18(CCC)
>= operator R4-17(MCC)
-> (macro notation) R6-3(ASM)
==(exists) R6-3(ASM)
== operator R4-17(MCC)
@ (nesting) R2-23(XRAY), U3-26(XRAY), U4-4(XRAY)
@ (path) U2-3(XRAY), U4-10(XRAY)

Master Index-3
@ (register)  R3-158(XRAY)
@ (root)  R2-20(XRAY)
@ (stack level)  R2-2(XRAY), R3-60(XRAY)
@ (symbol)  R2-7(XRAY)
  nesting  R2-23(XRAY), U3-26(XRAY), U4-4(XRAY)
  path  U2-3(XRAY), U4-10(XRAY)
  register  R3-158(XRAY)
  reserved words  R2-9(XRAY)
@ (symbol)  R2-7(XRAY)
  nesting  R2-23(XRAY), U3-26(XRAY), U4-4(XRAY)
  path  U2-3(XRAY), U4-10(XRAY)
  register  R3-158(XRAY)
  reserved words  R2-9(XRAY)
@ (symbol)  R2-7(XRAY)
  nesting  R2-23(XRAY), U3-26(XRAY), U4-4(XRAY)
  path  U2-3(XRAY), U4-10(XRAY)
  register  R3-158(XRAY)
  reserved words  R2-9(XRAY)
@ addr pseudo-register  RA-1(XRAY), RF-1(XRAY), RF-4(XRAY)
@ as pseudo-register  RA-1(XRAY), RF-1(XRAY)
@ chip pseudo-register  RA-1(XRAY), RF-1(XRAY)
@ cycles pseudo-register  RA-1(XRAY), RF-1(XRAY)
@ entry pseudo-register  RA-1(XRAY), RF-1(XRAY)
@ exc pseudo-register  RA-1(XRAY), RF-2(XRAY)
@ file pseudo-register  RA-1(XRAY), RF-2(XRAY)
@ fpf pseudo-register  RA-1(XRAY), RF-2(XRAY)

@ fpu pseudo-register  RA-1(XRAY), RF-2(XRAY), RF-5(XRAY), RF-10(XRAY)
@ hlpc pseudo-register  R2-21(XRAY), RA-1(XRAY), RF-3(XRAY)
@ line_range pseudo-register  RA-1(XRAY), RF-3(XRAY)
@ module pseudo-register  R2-21(XRAY), RA-1(XRAY), RF-3(XRAY)
@ pi pseudo-register  RA-1(XRAY), RF-3(XRAY)
@ pisize pseudo-register  RA-1(XRAY), RF-3(XRAY)
@ port_addr pseudo-register  R3-81(XRAY), R3-111(XRAY), RA-1(XRAY), RF-3(XRAY)
@ port_size pseudo-register  RA-1(XRAY), RF-3(XRAY)
@ port_value pseudo-register  R3-81(XRAY), R3-111(XRAY), RA-2(XRAY), RF-4(XRAY)
@ procedure pseudo-register  R2-21(XRAY), RA-2(XRAY), RF-4(XRAY)
@ root pseudo-register  R2-20(XRAY), RA-2(XRAY), RF-4(XRAY)
@ wait state pseudo-register  RA-2(XRAY), RF-4(XRAY)

[] indirect addressing  R2-9(XRAY)
\ R6-2(ASM)
\@ R2-8(ASM)
^ operator  R4-17(MCC)
  __OPTION_AVAIL macro  R4-4(MCC)
  __STDC__ preprocessor symbol  U2-43(MCC), U3-36(MCC)
  __STR_CASE_CMP macro  R4-5(MCC)
  __STR_CMP macro  R4-5(MCC)
  __DATE__ preprocessor symbol  R4-1(MCC)
  __FILE__ preprocessor symbol  R4-1(MCC)
  __LINE__ preprocessor symbol  R4-1(MCC)
  __STDC__ preprocessor symbol  R4-1(MCC), R4-3(MCC), U2-8(MCC), U2-16(MCC), U3-11(MCC), U2-12(CCC)
  __TIME__ preprocessor symbol  R4-2(MCC)
  __OPTION_utn preprocessor symbol  R4-3(MCC)
  __cplusplus  R5-2(CCC), RB-2(CCC)
  __68000 preprocessor symbol  R4-4(MCC)
  __68008 preprocessor symbol  R4-4(MCC)
  __68010 preprocessor symbol  R4-4(MCC)
  __68020 preprocessor symbol  R4-4(MCC)
  __68030 preprocessor symbol  R4-4(MCC)
  __68040 preprocessor symbol  R4-4(MCC)
  __68050 preprocessor symbol  R4-4(MCC)
  __68060 preprocessor symbol  R4-4(MCC)
  __68070 preprocessor symbol  R4-4(MCC)
  __68080 preprocessor symbol  R4-4(MCC)
  __68090 preprocessor symbol  R4-4(MCC)
  __680A0 preprocessor symbol  R4-4(MCC)
  __680B0 preprocessor symbol  R4-4(MCC)
  __680C0 preprocessor symbol  R4-4(MCC)
  __680D0 preprocessor symbol  R4-4(MCC)
  __680E0 preprocessor symbol  R4-4(MCC)
  __68100 preprocessor symbol  R4-4(MCC)
  __68110 preprocessor symbol  R4-4(MCC)
  __68120 preprocessor symbol  R4-4(MCC)
  __68130 preprocessor symbol  R4-4(MCC)
  __68140 preprocessor symbol  R4-4(MCC)
  __68150 preprocessor symbol  R4-4(MCC)
  __68160 preprocessor symbol  R4-4(MCC)
  __68170 preprocessor symbol  R4-4(MCC)
  __68180 preprocessor symbol  R4-4(MCC)
  __68190 preprocessor symbol  R4-4(MCC)
  __681A0 preprocessor symbol  R4-4(MCC)
  __681B0 preprocessor symbol  R4-4(MCC)
  __681C0 preprocessor symbol  R4-4(MCC)
  __681D0 preprocessor symbol  R4-4(MCC)
  __681E0 preprocessor symbol  R4-4(MCC)
  __681F0 preprocessor symbol  R4-4(MCC)
  __68200 preprocessor symbol  R4-4(MCC)
  __68210 preprocessor symbol  R4-4(MCC)
  __68220 preprocessor symbol  R4-4(MCC)
  __68230 preprocessor symbol  R4-4(MCC)
  __68240 preprocessor symbol  R4-4(MCC)
  __68250 preprocessor symbol  R4-4(MCC)
  __68260 preprocessor symbol  R4-4(MCC)
  __68270 preprocessor symbol  R4-4(MCC)
  __68280 preprocessor symbol  R4-4(MCC)
  __68290 preprocessor symbol  R4-4(MCC)
  __682A0 preprocessor symbol  R4-4(MCC)
  __682B0 preprocessor symbol  R4-4(MCC)
  __682C0 preprocessor symbol  R4-4(MCC)
  __682D0 preprocessor symbol  R4-4(MCC)
  __682E0 preprocessor symbol  R4-4(MCC)
  __682F0 preprocessor symbol  R4-4(MCC)
  __68300 preprocessor symbol  R4-4(MCC)
  __68310 preprocessor symbol  R4-4(MCC)
  __68320 preprocessor symbol  R4-4(MCC)
  __68330 preprocessor symbol  R4-4(MCC)
  __68340 preprocessor symbol  R4-4(MCC)
  __68EC000 preprocessor symbol  R4-4(MCC)
Numerics
16-bit bus R6-29(MCC)
16-bit displacement
(see -MI option) U2-28(MCC), U2-25(CCC)
/address=short option U3-13(MCC)
16-bit extension on stack
/address=long option U3-13(MCC)
-MI option U2-28(MCC), U2-25(CCC)
32-bit displacement
/address=long option U3-13(MCC)
-MI option U2-28(MCC), U2-25(CCC)
32-bit extension on stack
/address=long option U3-13(MCC)
-K2 option U2-24(MCC), U2-21(CCC)
32-bit extension on stack
/address=long option U3-13(MCC)
-K4 option U2-24(MCC), U2-21(CCC)

68000 S1-1(SUP)
68000/68020 structure alignment
R6-36(MCC)
68030 support RF-9(XRAY)
68040 support RF-10(XRAY)
68881 support RF-11(XRAY)
68EC000 S1-1(SUP)
68EC030 support RF-10(XRAY)
68EC040 support RF-11(XRAY)
68HC000 S1-1(SUP)
9-track tape
VMS I2-1(XRAY)

A
-A compiler option RB-2(CCC)
-a librarian command line option
U2-21(ASM)
-A option U2-8(MCC), U2-16(MCC), U2-12(CCC)
-A option (UNIX/DOS) R4-3(MCC)
a.out U2-24(ASM), U3-22(ASM)
A2-A5 relative addressing R3-24(ASM)–R3-33(ASM)
accessing dynamically allocated areas R3-27(ASM)
accessing statically allocated areas R3-25(ASM)
advantages R3-24(ASM)

Abbreviations R3-14(XRAY)
abort function R5-22(MCC)
relationship to signal R5-155(MCC)
Aborting installation I1-5(FLEX)
abs function R5-23(MCC)
relationship to labs R5-97(MCC)
Absolute addressing R9-25(MCC)
/code_addresses=absolute U3-14(MCC)
/data_addresses=absolute option U3-14(MCC)
-Mca option U2-27(MCC), U2-24(CCC)
-Mda option U2-27(MCC), U2-24(CCC)
Absolute expression R3-24(ASM)
Absolute expressions R4-8(ASM)
Absolute files, errors U4-11(XRAY)
ABSOLUTE linker command R10-7(ASM)–R10-8(ASM)
ABSOLUTE linker command line option U3-13(ASM)
Absolute object file format IEEE-695 U2-2(XRAY)
Absolute sections R9-3(ASM)
Absolute symbols R2-9(ASM), R4-7(ASM)
abspcadd assembler command line flag U2-6(ASM), U3-6(ASM)
Abstract class R3-15(CCC), RC-1(CCC), UA-1(CCC)
illegal use examples R3-17(CCC)
use example R3-15(CCC)
-acc option U2-17(MCC), U2-13(CCC)
Access breakpoints S3-34(SUP)
Access protection R2-2(CCC)
Access, user I3-6(FLEX)
Accessing
data members R3-3(CCC)
data structures
implicit class fields R5-8(CCC)
keywords R4-5(CCC), R4-6(CCC)
specifiers
default R2-3(CCC)
limited R2-3(CCC)
private R2-3(CCC)
protected R2-3(CCC)
structures R2-17(CCC)
Accessing memory
  NOMEMACCESS command
  R3-100(XRAY)
-acd option  U2-17(MCC), U2-13(CCC)

Accessing files in nondefault directories
  DOS  I1-5(MCC), I1-5(CCC), I1-6(CCC)
  UNIX System V/386  I3-2(MCC)
  VMS  I2-3(MCC)

Accessing the license file through the
daemon  I3-7(FLEX)
acos function  R5-24(MCC), U2-39(MCC), U3-25(MCC), U2-37(CCC)

Actions  S4-14(SUP)
Activate
  screen
    VSCREEN command  R3-210(XRAY)
  viewport
    VACTIVE command  R3-203(XRAY)
Active procedure (definition)  RD-1(XRAY)
Active symbols, verifying  R4-20(XRAY)
Active viewport  U3-33(XRAY)
ADD command  R3-18(XRAY), U2-26(XRAY)
Additional documentation  IP-2(MCC), UP1(-MCC), IP-2(XRAY), UP-1(CCC), IP-1(CCC)
ADDLIB librarian command  R13-4(ASM)
ADDMOD librarian command  U2-21(ASM), R13-5(ASM)

ADDMOD librarian command line option  U3-19(ASM)
@addr pseudo-register  RA-1(XRAY), RF-1(XRAY)
Address (definition)  R3-3(XRAY)
Address bus  S2-137(SUP)
Address modes  R3-7(ASM)-R3-24(ASM)
  absolute long  R3-12(ASM)
  absolute short  R3-11(ASM)
  address register direct  R3-9(ASM)
  address register indirect with displacement  R3-24(ASM)

code references
  absolute
    /code_addresses=absolute
      option  U3-14(MCC)
    -Mca option  U2-27(MCC), U2-24(CCC)
  PC-relative
    (see also Position-independent code)
    /code_addresses=pcrelative
      option  U3-14(MCC)
    -Mcp option  U2-27(MCC), U2-24(CCC)
  const section
    /const_addr_as options  U3-16(MCC)
    -ac options  U2-17(MCC), U2-13(CCC)
  data references
    absolute
      /data_addresses=absolute
        option  U3-14(MCC)
      -Mda option  U2-27(MCC), U2-24(CCC)
    PC-relative
      (see also Position-independent data)
      /data_addresses=pcrelative
        option  U3-14(MCC)
      -Mdp option  U2-28(MCC), U2-25(CCC)
    register-relative
      (see also Position-independent data)
      /data_addresses=anrelative  U3-14(MCC)
      -Md options  U2-27(MCC), U2-24(CCC)
  data register direct  R3-9(ASM)
  floating-point  R3-14(ASM)
  immediate  R3-13(ASM)
  initialized data section
    /initvars_addr_as options  U3-16(MCC)
-ai options  U2-17(MCC), U2-13(CCC)
literals section
 /literals_addr_as options  U3-16(MCC)
al options  U2-17(MCC), U2-14(CCC)
memory indirect post-indexed  R3-10(ASM)
memory indirect pre-indexed  R3-11(ASM)
program counter memory indirect
post-indexed  R3-12(ASM)
pre-indexed  R3-13(ASM)
program counter relative  R3-8(ASM)
program counter with base
displacement and index  R3-12(ASM)
program counter with displacement  R3-12(ASM)
register direct  R3-9(ASM)
register indirect  R3-9(ASM)
  8-bit displacement and index  R3-10(ASM)
  base displacement and index  R3-10(ASM)
displacement  R3-9(ASM)
postincrement  R3-9(ASM)
predecrement  R3-9(ASM)
selection  R3-20(ASM)–R3-22(ASM)
strings section
 /strings_addr_as options  U3-16(MCC)
syntax  R3-15(ASM)–R3-17(ASM)
user control  R3-22(ASM)–R3-24(ASM)
vars section
 /initvars_addr_as options  U3-16(MCC)
"address of" operator  R3-3(MCC)
Address strobes  S2-30(SUP)
Address_range (definition)  R3-3(XRAY)
Addresses
 byte value  R4-14(XRAY)
displaying
 NOSYMBOLS command  R3-108(XRAY)
indirect  R2-9(XRAY)
line numbers  R2-10(XRAY)
long value  R4-15(XRAY)
ranges  R2-10(XRAY)
word value  R4-38(XRAY)
Addressing
 absolute  R9-25(MCC)
changing the default  R8-2(MCC)
direct memory  R9-9(MCC)
PC-relative  R9-27(MCC)
register-relative  R9-25(MCC)
Addressing modes
 (see Address modes)
Addressing of processors
 odd-address restricted  R6-23(MCC)
odd-address unrestricted  R6-23(MCC)
Administration  i3-1(FLEX)
Administrative commands  U3-4(FLEX)
Advanced Event System  S3-45(SUP)
Advanced event system  S4-14(SUP)
structure  S4-14(SUP)
Advanced feature commands
 performance analysis
 PRINTPROFILE  R3-13(XRAY)
 PROFILE  R3-13(XRAY)
test coverage
 ANALYZE  R3-13(XRAY)
 PRINTANALYSIS  R3-13(XRAY)
trace
 SETSTATUS EVENT  R3-13(XRAY)
 SETSTATUS QUALIFY  R3-13(XRAY)
 SETSTATUS TRACE  R3-13(XRAY)
 STATUS BUFFER  R3-13(XRAY)
 STATUS EVENT  R3-13(XRAY)
 STATUS QUALIFY  R3-13(XRAY)
 STATUS TRACE  R3-13(XRAY)
 STATUS TRIGGER  R3-13(XRAY)
 TRACE  R3-13(XRAY)

Aggregates \(\text{R6-26(MCC)}\)
-aic option \(\text{U2-17(MCC), U2-13(CCC)}\)
-aid option \(\text{U2-17(MCC), U2-13(CCC)}\)
AIX
  host-specific information \(\text{UE-1(XRAY)}\)
-alc option \(\text{U2-17(MCC), U2-14(CCC)}\)
-ald option \(\text{U2-17(MCC), U2-14(CCC)}\)
Algebraic simplification \(\text{R101(MCC), R5-6(XRAY)}\)
ALIAS command \(\text{R3-20(XRAY)}\)
alias command (Linker) \(\text{U2-30(MCC), U2-26(CCC)}\)
ALIAS linker command \(\text{R10-9(ASM)--R10-10(ASM)}\)
Aliased references
  /optimize=stablemem option \(\text{U3-27(MCC)}\)
  -Ob option \(\text{U2-30(MCC), U2-27(CCC)}\)
ALIGN assembler directive \(\text{R5-4(ASM)}\)
ALIGN linker command \(\text{R10-11(ASM)--R10-12(ASM)}\)
Alignment
  68000/68020 structures \(\text{R6-36(MCC)}\)
  aggregates \(\text{R6-26(MCC)}\)
  array \(\text{R6-26(MCC)}\)
  bit fields \(\text{R6-24(MCC), R6-29(MCC)}\)
  data types \(\text{R6-30(MCC)}\)
  instructions \(\text{RF-4(XRAY)}\)
  low-level breakpoints
    ALIGN option \(\text{R3-104(XRAY)}\)
  padding bytes \(\text{R6-31(MCC), R6-34(MCC)}\)
  problems
    /align_check option \(\text{U3-11(MCC)}\)
    -KT option \(\text{U2-26(MCC)}\)
  structure members \(\text{R6-36(MCC)}\)
    /align option \(\text{U3-11(MCC)}\)
    /ssmultiple option \(\text{U3-11(MCC)}\)
    -Z options \(\text{U2-40(MCC), U2-37(CCC)}\)
  structures \(\text{R6-26(MCC), R6-29(MCC)}\)
  trace \(\text{R3-171(XRAY), R3-200(XRAY)}\)
  trailer bytes \(\text{R6-31(MCC), R6-34(MCC)}\)
  unions \(\text{R6-26(MCC)}\)
    -Zm option \(\text{U2-40(MCC), U2-38(CCC)}\)
Alignment of struct/union in parameter area \(\text{R7-4(MCC)}\)
Alignment, sections \(\text{R9-5(ASM)}\)
ALIGNMOD linker command \(\text{R10-13(ASM)}\)
Allocating
  data space \(\text{R5-35(MCC), R5-111(MCC), R5-203(MCC)}\)
  data types \(\text{R3-1(MCC), R6-3(MCC)}\)
  dynamic storage
    keywords \(\text{R4-4(CCC)}\)
    memory space \(\text{R5-146(MCC)}\)
Alphanumeric character, testing for \(\text{R5-83(MCC)}\)
Alternate locations
  UNIX
    executables \(\text{U2-4(MCC)}\)
    libraries \(\text{U2-4(MCC)}\)
    standard include files \(\text{U2-4(MCC)}\)
    temporary files \(\text{U2-4(MCC)}\)
  VMS
    libraries \(\text{U3-3(MCC)}\)
    standard include files \(\text{U3-3(MCC)}\)
    temporary files \(\text{U3-4(MCC)}\)
AMCTOOLS
  DOS \(\text{I1-1(MCC), I1-1(XRAY), I1-1(CCC)}\)
Anachronism \(\text{RC-1(CCC), UA-1(CCC)}\)
Anachronistic constructs, forbidden
  +p option \(\text{U2-32(CCC)}\)
ANALYZE command \(\text{R3-22(XRAY)}\)
Analyze test coverage data
  PRINTANALYSIS command \(\text{R3-115(XRAY)}\)
ANSIC
  contrast with C++ \(\text{R4-1(CCC)}\)
  function declarations \(\text{R4-12(CCC)}\)
ANSI C syntax \(\text{RB-1(MCC)}\)
ANSI-compliant mode, setting
  /ansi option \(\text{U3-11(MCC)}\)
  -A option \(\text{U2-8(MCC), U2-16(MCC), U2-12(CCC)}\)
Apollo
  host-specific information \(\text{UB-1(XRAY)}\)
Apollo installation
  DOMAIN/OS version required IB-1(FLEX)
  reading distribution I1-7(FLEX)
  TCP/IP IB-1(FLEX)
  version required IB-1(FLEX)
Apollo mouse support
  (see Mouse support)
  _APOLLO preprocessor symbol R4-3(MCC)
Apollo rbak command I1-7(FLEX)
Apollo support
  escape key UB-3(XRAY)
  Line Del key UB-3(XRAY)
  MOVE TO BOTTOM control key UB-2(XRAY)
Arc cosine of a number, computing R5-24(MCC)
Arc sine of a number, computing R5-26(MCC)
Arc tangent of a number, computing R5-28(MCC)
Arc tangent of d1/d2, computing R5-29(MCC)
arch command (Sun) IA-1(FLEX)
Architectures
  mixed installation IA-2(FLEX)
Argument
  passing R5-3(CCC)
  promotion R5-3(CCC)
Argument list, variable
  allowing access R5-196(MCC)
  returning arguments R5-193(MCC)
  terminating access R5-195(MCC)
Arithmetic functions
  (see Mathematical functions)
Arithmetic operators R3-10(MCC), RC-1(XRAY)
Arithmetic plus operation R2-23(CCC)
Arming triggers S2-76(SUP), S2-78(SUP)
Array operator synthesis
  optimization R106(MCC)
Arrays R6-4(MCC), R6-15(MCC)
  initialization R4-16(CCC)
Arrow (->) operator R2-17(CCC), R2-25(CCC)
Arrow-star (->*) operator R2-18(CCC)
  @as pseudo-register RA-1(XRAY), RF-1(XRAY)
-asc option U2-17(MCC), U2-14(CCC)
ASCII character code RA-1(ASM)-RA-2(ASM)
ASCII character set RA-1(MCC), UA-1(MCC), RB-1(XRAY)
ASCII character, testing for R5-85(MCC)
ASCII format
  conversion from byte R5-186(MCC)
ASCII string
  conversion from floating-point number R5-75(MCC)
  conversion from integer R5-95(MCC)
  conversion from long integer R5-109(MCC)
  conversion from unsigned integer to R5-96(MCC)
  conversion from unsigned long integer R5-110(MCC)
  conversion to double R5-177(MCC)
  conversion to floating-point number R5-31(MCC)
  conversion to integer R5-32(MCC)
  conversion to long integer R5-33(MCC), R5-179(MCC)
asctime function R5-20(MCC), R5-25(MCC)
-asd option U2-17(MCC), U2-14(CCC)
asin function R5-26(MCC), U2-39(MCC), U3-25(MCC), U2-37(CCC)
ASM (Single line assembler) S2-2(SUP)
ASM command S3-30(SUP)
asm keyword R4-2(CCC)
ASM pseudofunction
  (see asm pseudofunction)
asm pseudofunction R9-3(MCC)
asm pseudofunction, disabling
  /nomri_extensions option U3-25(MCC)
  -nx option U2-40(MCC), U2-37(CCC)
asm pseudofunction, enabling
  /mri_extensions option U3-24(MCC)
  -x option U2-39(MCC), U2-36(CCC)
asm support R2-26(CCC)
Assembler

- absolute expressions R3-24(ASM), R4-8(ASM)
- addressing modes R3-1(ASM)
- asmb_sym assembler symbol files R4-4(ASM)
- assembly time relative addressing R2-5(ASM)
- attributes
  - common vs. noncommon R4-2(ASM)
  - section alignment R4-3(ASM)
  - section type R4-4(ASM)
  - short vs. long R4-2(ASM)
- character set R2-6(ASM)
- constants R2-11(ASM)–R2-15(ASM)
  - character R2-13(ASM)–R2-15(ASM)
  - floating-point R2-12(ASM)–R2-13(ASM)
- integer R2-11(ASM)–R2-12(ASM)
- cross-reference table format R8-3(ASM)
- description U1-1(MCC), U1-1(ASM), U1-1(CCC)
- directives R5-1(ASM)–R5-74(ASM), R6-5(ASM)–R6-11(ASM), R7-5(ASM)–R7-13(ASM)
- error messages RB-1(ASM)–RB-14(ASM)
- floating-point R3-6(ASM)
- HP 64000 files R4-4(ASM)
- instructions R3-1(ASM)
- introduction R1-1(ASM), R2-1(ASM)
- invoking U3-45(MCC)
- link_sym linker symbol files R4-4(ASM)
- listing, sample program R8-4(ASM)–R8-6(ASM)
- macros R6-1(ASM)–R6-11(ASM)
- name demangling RG-2(ASM)
- name mangling RG-2(ASM)
- object module R8-7(ASM)
- operation R8-1(ASM)
- options, passing directly
  - -Wa option U2-38(MCC), U2-35(CCC)
- overview R1-1(ASM)
- program counter R2-10(ASM)

relative addressing R3-24(ASM)–R3-33(ASM)
relocatable expressions R3-24(ASM), R4-7(ASM)–R4-8(ASM)
section attributes
  - assigning R4-5(ASM)
source file
  (see Assembler source file)
statement R2-1(ASM)–R2-4(ASM)
  - field R2-1(ASM)–R2-2(ASM)
  - comment R2-2(ASM)
  - label R2-1(ASM)
  - operand R2-2(ASM)
  - operation R2-2(ASM)
  - type R2-2(ASM)–R2-4(ASM)
  - directive R2-3(ASM)
  - instruction R2-2(ASM)
  - macro R2-3(ASM)
structure control directives R7-5(ASM)–R7-13(ASM)
symbolic addressing R2-4(ASM)–R2-6(ASM)
symbols R2-7(ASM)–R2-10(ASM)
  - invalid R2-8(ASM)
  - relocatable versus absolute R4-7(ASM)
  - reserved R2-8(ASM)–R2-9(ASM)
  - valid R2-8(ASM)
syntax R2-6(ASM)–R2-17(ASM)
terminator record RF-4(ASM)
UNIX/DOS U2-2(ASM)–U2-12(ASM)
file name defaults U2-4(ASM)
flags U2-5(ASM)–U2-11(ASM)
  - abspcadd U2-6(ASM)
  - brb U2-6(ASM)
  - brl U2-6(ASM)
  - hrs U2-6(ASM)
  - brw U2-6(ASM)
  - case U2-7(ASM)
  - cex U2-7(ASM)
  - cl U2-7(ASM)
  - cre U2-7(ASM)
  - d U2-7(ASM)
e U2-7(ASM)
frel U2-7(ASM)
g U2-7(ASM)
i U2-7(ASM)
llen U2-8(ASM)
mc U2-8(ASM)
md U2-8(ASM)
mex U2-8(ASM)
nest U2-8(ASM)
o U2-8(ASM)
old U2-8(ASM)
op U2-8(ASM)
opnop U2-9(ASM)
p U2-9(ASM)
pco U2-9(ASM)
pcr U2-10(ASM)
pcs U2-10(ASM)
quick U2-10(ASM)
r U2-10(ASM)
rel32 U2-11(ASM)
s U2-11(ASM)
t U2-11(ASM)
w U2-11(ASM)
x U2-11(ASM)

Invocation examples U2-12(ASM)
Invocation syntax U2-2(ASM)
Options U2-2(ASM)-U2-4(ASM)
  -b U2-2(ASM)
  -D U2-2(ASM)
  -f U2-3(ASM)
  -l U2-3(ASM)
  -L U2-3(ASM)
  -o U2-3(ASM)
  -V U2-3(ASM)

VAX/VMS U3-2(ASM)-U3-12(ASM)
File name defaults U3-4(ASM)
Flags U3-4(ASM)-U3-11(ASM)
  abs cp add U3-6(ASM)
  brb U3-6(ASM)
  brl U3-6(ASM)
  brs U3-6(ASM)
  brw U3-6(ASM)
  case U3-7(ASM)
  cex U3-7(ASM)
  cl U3-7(ASM)
  cre U3-7(ASM)
  d U3-7(ASM)
  e U3-7(ASM)
  frel U3-7(ASM)
  frs U3-7(ASM)
  g U3-7(ASM)
  i U3-7(ASM)
  llen U3-8(ASM)
  mc U3-8(ASM)
  md U3-8(ASM)
  mex U3-8(ASM)
  nest U3-8(ASM)
  o U3-8(ASM)
  old U3-8(ASM)
  op U3-8(ASM)
  opnop U3-9(ASM)
  p U3-9(ASM)
  pco U3-9(ASM)
  pcr U3-10(ASM)
  pcs U3-10(ASM)
  quick U3-10(ASM)
  r U3-10(ASM)
  rel32 U3-11(ASM)
  s U3-11(ASM)
  t U3-11(ASM)
  w U3-11(ASM)
  x U3-11(ASM)

Invocation examples U3-12(ASM)
Invocation syntax U3-2(ASM)
Options U3-3(ASM)-U3-4(ASM)
  DEFINE U3-3(ASM)
  FLAGS U3-3(ASM)
  IPATH U3-3(ASM)
  LIST U3-3(ASM)
  NOLIST U3-3(ASM)
  NOOBJECT U3-3(ASM)
  OBJECT U3-3(ASM)
  VERSION U3-3(ASM)

Assembler directives
  (see Directives, assembler)

Assembler in-line R9-3(MCC), R2-26(CCC)
  Considerations R9-8(MCC)
Assembler relocation flags  R6-2(ASM)
Assembler source file  R11-1(MCC)
  #include files expanded
    /show=include option  U3-28(MCC)
    -Fsi option  U2-20(MCC)
advantages to producing  R11-1(MCC)
contents  R11-3(MCC)
generating
    /asm=filename option  U3-13(MCC)
    -S option  U2-36(MCC), U2-33(CCC)
high-level source code, including as comments
    /show=source option  U3-29(MCC)
    -Fsm option  U2-20(MCC), U2-17(CCC)
line numbers  R11-2(MCC)
naming
    -o option  U2-33(MCC), U2-30(CCC)
variable names  R11-1(MCC)
Assembly and high-level code
    /show=source option  U3-29(MCC)
    -Fsm option  U2-20(MCC), U2-17(CCC)
Assembly code
  insertion specification
    keywords  R4-2(CCC)
Assembly code intermixed with source code
    LINES option  R3-107(XRAY)
Assembly code, optimizing by hand  R11-1(MCC)
Assembly file
  saving
    -H option  U2-22(MCC), U2-19(CCC)
Assembly language
  example of a routine  R7-9(MCC)
interface  R7-1(MCC), R7-8(MCC)
Assembly-level mode
  MODE command  R3-92(XRAY)
Assembly-level mode debugging  U1-2(XRAY), U2-20(XRAY)
  CLEAR command  U2-20(XRAY), U2-24(XRAY)
  GO command  U2-22(XRAY), U2-24(XRAY)
  MODE command  U2-20(XRAY)
PRINTVALUE command  U2-22(XRAY)
QUIT command  U2-24(XRAY)
RESTART command  U2-20(XRAY)
STEP command  U2-22(XRAY)
tutorial  U2-20(XRAY)
VSCREEN command  U2-24(XRAY)
Assembly-level screen  S1-6(SUP), U3-2(XRAY)
assert macro  R5-2(MCC), R5-27(MCC)
assert.h include file  R5-2(MCC)
Assignments
  form of  R3-9(MCC)
  operators  R3-12(MCC)
at sign (@)  R2-9(XRAY)
atan function  R5-28(MCC), U2-39(MCC), U3-25(MCC), U2-37(CCC)
atan2 function  R5-29(MCC)
atanh function  U2-40(MCC), U3-25(MCC), U2-37(CCC)
atexit function  R5-20(MCC), R5-30(MCC)
  relationship to exit  R5-47(MCC)
atof function  R5-31(MCC)
atoi function  R5-32(MCC)
atol function  R5-33(MCC)
Auto
  keyword  R3-5(MCC)
  storage class  R3-5(MCC)
  variables  R3-5(MCC)
AUTOEXEC.BAT
  DOS  I1-3(MCC), I1-3(XRAY), I1-4(CCC)
B
  -b assembler command line option  U2-2(ASM)
b.out  U2-24(ASM), U3-22(ASM)
Backspace
  disable for preprocessor
    -Es option  U2-19(MCC), U2-16(CCC)
    -Ps option  U2-33(MCC), U2-30(CCC)
Bandwidth, data bus  R6-23(MCC), R6-29(MCC)
Banner
  /banner option  U3-13(MCC)
-Vb option U2-37(MCC), U2-34(CCC)
Base address R9-6(ASM), R10-14(ASM)
Base class RC-1(CCC), UA-1(CCC)
definition R3-1(CCC)
packed R4-4(CCC)
specifier R3-1(CCC)
private R2-3(CCC), R3-1(CCC)
public R2-3(CCC), R3-1(CCC)
virtual R2-3(CCC), R3-1(CCC)
unpacked R4-4(CCC)

BASE linker command R10-14(ASM)–R10-15(ASM)
Base pointer R5-10(CCC)
Basic breakpoints S3-34(SUP)
Batch commands U4-8(XRAY)
Batch mode
C++ inspector U2-52(CCC)
Batch mode support U4-8(XRAY)
_BCS preprocessor symbol R4-3(MCC)
Before you install I-2(FLEX)
Beginning of a file, pointing to R5-145(MCC)
BIG_ENDIAN R6-1(MCC)
Binary expressions R3-9(MCC)
Binary search, performing R5-34(MCC)
Bit fields R6-20(MCC)
alignment R6-24(MCC), R6-29(MCC)
packing R6-23(MCC)

Bound pointer RA-6(CCC)
BPSPACE (specify breakpoint space) S2-4(SUP)
Branch instruction labels
local (??) R7-4(ASM)
Branch tail merging R1011(MCC)
Branch tail optimization R5-12(XRAY)
brb assembler command line flag U2-6(ASM), U3-6(ASM)
BREAK assembler directive R7-6(ASM)
break button
SunView support UH-12(XRAY)
brk statement R3-19(MCC)
BREAK statement in macros R4-4(XRAY)
Break viewport U3-7(XRAY)
# breakpoint number field U3-7(XRAY)
Address field U3-7(XRAY)

assembly-level mode U2-21(XRAY)
Command Argument field U3-8(XRAY)
description of fields U3-7(XRAY)
high-level mode U2-18(XRAY)
Line field U3-7(XRAY)
Mod/Fnct field U3-7(XRAY)
OPTION BREAK = ON command U3-7(XRAY)
OPTION SWAP = ON command U3-7(XRAY)

Type field U3-8(XRAY)

Break viewport, display of
BREAK option R3-104(XRAY)
BREAKACCESS S1-7(SUP)
BREAKACCESS command S2-5(SUP), R3-25(XRAY), U4-5(XRAY)
BREAKCOMPLEX S1-7(SUP), S2-10(SUP)
BREAKCOMPLEX command R3-28(XRAY)
BREAKINSTRUCTION S1-7(SUP)
BREAKINSTRUCTION button
X Window support UA-12(XRAY)
BREAKINSTRUCTION command S2-12(SUP), R3-30(XRAY), U2-30(XRAY), U4-5(XRAY)
high-level mode U2-18(XRAY)

Breakpoint scratch area address S2-128(SUP)

Breakpoints U4-5(XRAY)
access S3-34(SUP)
associating with macro R4-2(XRAY)
basic S3-34(SUP)
break when expression is true R4-36(XRAY), R4-37(XRAY)
breakcomplex S2-10(SUP)
clearing S3-35(SUP), U2-20(XRAY)
CLEAR command R3-40(XRAY)
complex S3-48(SUP), S4-23(SUP)
encountering during execution U2-23(XRAY)
event system S3-46(SUP)
in C++ RG-13(XRAY)
instruction S3-34(SUP), RF-7(XRAY), U2-17(XRAY), U2-21(XRAY), U2-30(XRAY)
recursive functions R2-24(XRAY)
setting S3-34(SUP)
access
    BREAKACCESS command R3-25(XRAY)
complex
    BREAKCOMPLEX command R3-28(XRAY)
instruction
    BREAKINSTRUCTION command S2-12(SUP), R3-30(XRAY)
read
    BREAKREAD command S2-16(SUP), R3-32(XRAY)
write
    BREAKWRITE command S2-20(SUP), R3-35(XRAY)
simple S4-3(SUP)
temporary S4-5(SUP), U2-28(XRAY), U4-6(XRAY)
Breakpoints,trace control S2-138(SUP)
BREAKREAD command S2-16(SUP), R3-32(XRAY), U4-5(XRAY)
BREAKWRITE S1-7(SUP)
BREAKWRITE command S2-20(SUP), R3-35(XRAY), U4-5(XRAY), U4-7(XRAY)
Brknst button
    X Window support UA-12(XRAY)
brl assembler command line flag U2-6(ASM), U3-6(ASM)
    BROWSE S2-24(SUP)
BROWSE command (C++) RG-12(XRAY)
brs assembler command line flag U2-6(ASM), U3-6(ASM)
brw assembler command line flag U2-6(ASM), U3-6(ASM)
bsearch function R5-34(MCC)
BTE (emulator bus timeout) S2-25(SUP)
Buffer
    altering mode R5-154(MCC)
    altering size R5-154(MCC)
    association with I/O file R5-150(MCC)
Buffer, trace
    SETSTATUS QUALIFY command R3-167(XRAY)
    STATUS BUFFER command R3-183(XRAY)
    STATUS QUALIFY command R3-188(XRAY)
Buffered data
    flushing to a file R5-54(MCC)
BUFSIZ R5-14(MCC)
Bus S2-137(SUP), S3-27(SUP)
Bus bandwidth R6-23(MCC), R6-29(MCC)
BUS command S2-27(SUP)
Bus errors S2-33(SUP)
Bus timeout, fast S2-87(SUP)
Bus timing information
    in trace S2-29(SUP)
Buttons
    arguments for
        SunView support UH-10(XRAY)
        X Window support UA-10(XRAY)
    constructing commands
        SunView support UH-10(XRAY)
        X Window support UA-10(XRAY)
    defining
        SunView support UH-14(XRAY)
        X Window support UA-13(XRAY)
    standard
        SunView support UH-11(XRAY)
SunView UH-9(XRAY)–UH-14(XRAY)
    break UH-12(XRAY)
    clear UH-12(XRAY)
    context UH-12(XRAY)
    go UH-12(XRAY)
    go to UH-12(XRAY)
    help UH-12(XRAY)
    load UH-12(XRAY)
    mode UH-12(XRAY)
    monitor UH-12(XRAY)
    prevcmd UH-12(XRAY)
    print UH-12(XRAY)
    print * UH-13(XRAY)
    scope UH-13(XRAY)
    screen UH-13(XRAY)
status UH-13(XRAY)
step UH-13(XRAY)
stepo UH-13(XRAY)
stop UH-13(XRAY)
user-defined
SunView support UH-14(XRAY)
X Window support UA-13(XRAY)

X Window
BrkInst UA-12(XRAY)
Clear UA-12(XRAY)
Context UA-12(XRAY)
Go UA-12(XRAY)
Go To UA-12(XRAY)
Help UA-12(XRAY)
Load UA-12(XRAY)
Mode UA-12(XRAY)
Monitor UA-12(XRAY)
Prevcmd UA-12(XRAY)
Print UA-12(XRAY)
Print * UA-13(XRAY)
Scope UA-13(XRAY)
Screen UA-13(XRAY)
Start UA-13(XRAY)
Status UA-13(XRAY)
Step UA-13(XRAY)
StepOvr UA-13(XRAY)

By-argument qualification RG-9(XRAY)
byte macro R4-14(XRAY)

Bytes
  clearing memory bytes R5-118(MCC)
  conversion to ASCII format R5-186(MCC)
  multibyte character R5-113(MCC)
  reading from a file R5-142(MCC)
  swapping odd and even bytes R5-182(MCC)
  writing to a file R5-202(MCC)

C
C calling conventions R7-1(MCC)
C comments, saving in preprocessor output
  /preserve_comments option U3-29(MCC)
  -C option U2-17(MCC), U2-14(CCC)

C compiler
  description U1-1(CCC)
C file, intermediate
    -Fc option U2-17(CCC)
C language elements R2-1(MCC)
-C linker command line option U2-13(ASM)
-c linker command line option U2-13(ASM)
C operators RC-1(XRAY)
-C option U2-17(MCC), U2-14(CCC)
-c option U2-18(MCC), U2-14(CCC)
  example use U2-41(MCC)

C++ implementation issues
  +p compiler option RB-3(CCC)
  -A compiler option RB-2(CCC)
  character constants RB-1(CCC)
  floating-point constants RB-2(CCC)
  identifiers RB-1(CCC)
  keywords RB-1(CCC)
  -nA compiler option RB-3(CCC)
  nonmember functions RB-2(CCC)
  nonstatic data members RB-2(CCC)
  nonvirtual base classes RB-2(CCC)
  operators RB-2(CCC)
  predefined macros RB-2(CCC)
  keywords R4-1(CCC)

C++ compiler
  command line options U2-6(CCC)
  command line syntax U2-1(CCC)
  default file name extensions U2-2(CCC)
  description U1-7(XRAY), U1-1(CCC)
  environment variables U2-4(CCC)
  file locations U2-3(CCC)

C++ compiler package
  assembler U1-1(CCC)
  C compiler U1-5(XRAY), U1-1(CCC)
  C++ compiler U1-7(XRAY), U1-1(CCC)
  component descriptions U1-1(CCC)
  data flow U1-4(CCC)
  librarian U1-2(CCC)
  XRAY Debugger U1-3(CCC)

C++ file suffix
  +z option U2-38(CCC)

C++ inspector U2-46(CCC)
batch mode  U2-52(CCC)
  help command  U2-53(CCC)
  run command  U2-52(CCC)
  srun command U2-52(CCC)
command line examples U2-53(CCC)
command line syntax  U2-47(CCC)
example files
  ASCII file containing mangled names U2-56(CCC)
  C++ source file U2-55(CCC)
interactive mode U2-48(CCC)
  Help button U2-51(CCC)
  Look button U2-50(CCC)
  Quit button U2-52(CCC)
  Run button U2-49(CCC)

C++ intermediate file
  +i option U2-20(CCC)

C++ language
  contrast with C R4-1(CCC)
  function declarations R4-12(CCC)
  grammar rules R4-11(CCC)
  keywords R4-1(CCC)
  parentheses R4-11(CCC)
  wrapper R5-6(CCC)

C++ support RG-1(XRAY)
  access protection RG-5(XRAY)
  breakpoints RG-13(XRAY)
    class RG-14(XRAY)
    instance RG-13(XRAY)
    object RG-13(XRAY)
    overloaded RG-15(XRAY)
  BROWSE command RG-12(XRAY)
  CEXPRESSION command RG-7(XRAY)
  class RG-2(XRAY)
  comments RG-16(XRAY)
  data members RG-3(XRAY)
  disable features RG-1(XRAY)
  DISASSEMBLE command RG-7(XRAY)
  inheritance RG-10(XRAY)
    BROWSE command RG-12(XRAY)
    member functions RG-3(XRAY)
    MONITOR command RG-15(XRAY)
    operator functions RG-6(XRAY)
    overloaded functions RG-6(XRAY), RG-
7(XRAY)
  by-argument qualification RG-9(XRAY)
  one-of qualification RG-7(XRAY), RG-9(XRAY)
  PRINTVALUE command RG-2(XRAY)
  reference variables RG-5(XRAY)
  run-time libraries RG-1(XRAY)
  static members RG-4(XRAY), RG-15(XRAY)
  struct RG-2(XRAY)
  union RG-2(XRAY)
  XRAY with C++ RG-1(XRAY)

c_plusplus RB-2(CCC)
c68k U3-3(MCC)

Calendar time
  breaking down to local time R5-82(MCC)
  conversion to local time R5-104(MCC)
  determining current time R5-185(MCC)

Calendar times, computing difference between R5-44(MCC)

Calling a C function R5-1(CCC)

Calling functions
  C from C++ R5-1(CCC)
  C++ from C R5-1(CCC), R5-4(CCC)--R5-
8(CCC)
    member functions R5-4(CCC)
    overloaded functions R5-4(CCC)
    member functions R5-4(CCC)

Calling technical support l1-9(FLEX), l1-
10(FLEX)

calloc function R5-20(MCC), R5-35(MCC), R9-12(MCC)
  relationship to free R5-67(MCC)
  relationship to realloc R5-143(MCC)

Cancel pending interrupts
  NOINTERRUPT command R3-
99(XRAY)

Cartridge tape
  VMS I2-1(XRAY)

CAS (continuous address strobe) S2-
30(SUP)

case assembler command line flag U2-
7(ASM), U3-7(ASM)
CASE linker command  R10-16(ASM)–R10-17(ASM)
catch keyword  R4-2(CCC), RB-1(CCC)
cc68ka5.lib  U2-42(CCC)
cc68ka5020.lib  U2-42(CCC)
cc68kab.lib  U2-42(CCC)
cc68kab020.lib  U2-42(CCC)
cc68kpc.lib  U2-42(CCC)
cc68kpc020.lib  U2-42(CCC)
cd command  It-6(FLEX)
ceil function  R5-36(MCC)
cex assembler command line flag  U2-7(ASM), U3-7(ASM)
CEXPRESSION command  R3-38(XRAY), U4-16(XRAY)
with C++ RG-7(XRAY)
Change memory contents
    SETMEM command  R3-156(XRAY)
Changing event groups  S4-22(SUP)
Changing flag states  S4-22(SUP)
Changing storage size  R5-143(MCC)
Char Del key (Apollo support)  UB-3(XRAY)
char type  R3-1(MCC), R6-3(MCC), R6-10(MCC)
Character
    alphanumeric, testing for  R5-83(MCC)
    comparing characters in memory  R5-119(MCC)
    converting to lowercase  R5-187(MCC), R5-188(MCC)
    converting to uppercase  R5-189(MCC), R5-190(MCC)
    copying characters from memory  R5-116(MCC), R5-120(MCC), R5-121(MCC)
    determining bytes in multibyte character  R5-113(MCC)
    first occurrence in string  R5-163(MCC)
    multibyte conversion to wide character  R5-115(MCC)
    reading character from a file  R5-55(MCC)
    reading from a file  R5-77(MCC)
    reading from standard input  R5-78(MCC)
    returned to file  R5-191(MCC)
    searches character in memory  R5-117(MCC)
    setting value in memory  R5-122(MCC)
    testing for ASCII character  R5-85(MCC)
    testing for control character  R5-86(MCC)
    testing for lower-case character  R5-84(MCC), R5-89(MCC)
    testing for upper-case character  R5-84(MCC), R5-93(MCC)
    writing to a file  R5-64(MCC), R5-134(MCC)
Character constants  R2-13(ASM)–R2-15(ASM)
Character handling functions  R5-3(MCC)
isalnum  R5-83(MCC)
isalpha  R5-84(MCC)
isctrl  R5-86(MCC)
digit  R5-87(MCC)
isgraph  R5-88(MCC)
islower  R5-89(MCC)
isprint  R5-90(MCC)
ispunct  R5-91(MCC)
isspace  R5-92(MCC)
isupper  R5-93(MCC)
isxdigit  R5-94(MCC)
tolower  R5-187(MCC)
toupper  R5-189(MCC)
Character set  RA-1(MCC)
    assembler  R2-6(ASM)
Character string constants  R2-4(MCC)
Characters
    ASCII character set  RA-1(MCC), UA-1(MCC), RB-1(XRAY)
    copying from memory  R4-25(XRAY)
    nonprintable  R2-5(MCC), R2-6(XRAY)
    searching for in memory  R4-23(XRAY)
    searching for in string  R4-30(XRAY)
    setting value in memory  R4-26(XRAY)
    strings  R2-5(XRAY)
Checking in licenses  U3-3(FLEX), U3-5(FLEX)
Checking out licenses U1-2(FLEX)
(see also Holding licenses)
after inactive period U1-2(FLEX)
automatically U1-2(FLEX), U3-5(FLEX)
determining which are held U1-4(FLEX), U3-2(FLEX)
expiration time
(see Expiration of licenses)
failure U2-5(FLEX)
first time U1-2(FLEX)
flow of operation U2-3(FLEX)
in advance U1-5(FLEX), U3-1(FLEX)
reserving licenses U2-11(FLEX)
length of time held
(see Expiration of licenses)
license server daemons not running U1-2(FLEX), U1-3(FLEX)
manually U3-1(FLEX)
MCC68K U1-2(FLEX)
operation flow U2-3(FLEX)
required number of licenses U2-1(FLEX)

Checking syntax only
/syntax_only option U3-33(MCC)
-y option U2-40(MCC), U2-37(CCC)

Checking, extra
/extra_checks option U3-23(MCC)
-v option U2-38(MCC), U2-35(CCC)

CHIP assembler directive R5-5(ASM)–R5-7(ASM)

Chip control registers, updating S2-156(SUP)
CHIP linker command R10-18(ASM)–R10-20(ASM)
@chip pseudo-register RA-1(XRAY), RF-1(XRAY)
cl assembler command line flag U2-7(ASM), U3-7(ASM)

Class R2-1(CCC)
access R2-2(CCC)
base RC-1(CCC), UA-1(CCC)
definition RC-1(CCC), UA-1(CCC)
derived RC-2(CCC), UA-2(CCC)
friend RC-2(CCC), UA-2(CCC)

member R2-2(CCC), RC-1(CCC), RC-2(CCC)

class keyword R4-2(CCC)

Class member UA-1(CCC), UA-2(CCC)

Classes
base R3-1(CCC)
declaring C++ R4-2(CCC), R4-10(CCC)
derived R3-1(CCC)
friends R2-4(CCC)
keyword R4-2(CCC)
nested R4-16(CCC)
operators R2-22(CCC)

Clear button
X Window support UA-12(XRAY)

clear button
SunView support UH-12(XRAY)
CLEAR command R3-40(XRAY), U2-20(XRAY), U2-24(XRAY)
Clear event system S2-77(SUP)
CLEAR librarian command R13-6(ASM)
clearerr function R5-12(MCC), R5-37(MCC)

Clearing
breakpoints
CLEAR command R3-40(XRAY)
memory R4-24(XRAY)
viewports
VCLEAR command R3-204(XRAY)

Clearing memory bytes R5-118(MCC)
Clock
not supported S1-7(SUP)
Clock frequency S2-27(SUP)
clock function R5-38(MCC)
CLOCKS_PER_SEC macro R5-38(MCC)
close function R5-39(MCC), R9-31(MCC), U2-52(MCC), U3-43(MCC), U2-44(CCC)
relationship to fclose R5-51(MCC)

Close viewport
VCLOSE command R3-205(XRAY)

Closing a file R5-39(MCC), R5-51(MCC)
CLR instruction, using
/cr option U3-13(MCC)
-Kc option U2-24(MCC), U2-21(CCC)
cmx68ka5.lib U2-42(CCC)
cmx68ka5020.lib U2-42(CCC)
cmx68k.lib U2-42(CCC)
cmx68kabol020.lib U2-42(CCC)
cmx68kpc.lib U2-42(CCC)
cmx68kpc020.lib U2-42(CCC)

Code
position-dependent R9-19(MCC)
position-independent R9-19(MCC)
/code_address option (VMS) R4-3(MCC)

Code and data section names,
specifying U2-28(MCC), U3-30(MCC), U2-25(CCC)

Code elimination, unreachable R102(MCC)

Code hoisting R1012(MCC), R5-13(XRAY)
/code option (VMS) R9-19(MCC), R9-20(MCC)

Code organization R8-1(MCC)
compiler-generated sections R8-2(MCC)

Code patching S3-30(SUP)

Code references
absolute
/code_addresses=absolute option U3-14(MCC)
-Mca option U2-27(MCC), U2-24(CCC)

PC-relative
/code_addresses=pcrelative U3-14(MCC)
-Mcp option U2-27(MCC), U2-24(CCC)

code section R8-4(MCC), U2-16(MCC), U3-15(MCC), U2-13(CCC)

naming
/rename option U3-30(MCC)
-NT option U2-29(MCC), U2-26(CCC)

Code viewport U3-1(XRAY), U3-10(XRAY)
assembly-level U3-2(XRAY)
C source code lines in U3-3(XRAY)
high-level U2-16(XRAY)

Code viewport display
HIGHLIGHT option R3-106(XRAY)

Colon-colon (::) operator R2-25(CCC)
Colon-colon-star (::*) operator R2-18(CCC)

Color
change display S1-8(SUP)

Color display, setting
COLOR option R3-105(XRAY)
VPCOLOR option R3-109(XRAY)

Colors, setting UA-3(XRAY)

Column number debugging information
/debug=multi_stmt option U3-19(MCC)
-Gm option U2-21(MCC)

COMLINE assembler directive R5-8(ASM)

Command
aliasing S3-20(SUP)
history S3-20(SUP)

Command definitions R3-17(XRAY)

Command file
-c linker option U2-13(ASM)

Command file, passing to linker
-e option U2-19(MCC), U2-16(CCC)

Command files
comments R3-4(XRAY)
processing
INCLUDE command R3-79(XRAY)

Command key commands R3-12(XRAY)

Command keys U1-3(XRAY), U3-35(XRAY)
(see also under specific name)

Apollo workstations UB-4(XRAY)
command line editing keys UB-5(XRAY)
DECstation terminals UC-3(XRAY)
HP support UD-6(XRAY)
IBM RS/6000 UE-3(XRAY)
Motorola Delta Series workstations UF-3(XRAY)
MOVE TO BOTTOM U3-36(XRAY)
MOVE TO TOP U3-36(XRAY)
PC/DOS UG-3(XRAY)
Sun Microsystems workstations UH-4(XRAY)
VT terminals UI-3(XRAY)

Command keys (trace mode)
Apollo workstations UB-5(XRAY)
DECstation terminals UC-5(XRAY)
HP terminals UD-7(XRAY)
IBM RS/6000 UE-4(XRAY)
Motorola Delta Series workstations UF-5(XRAY)
PC/DOS UG-5(XRAY)
Sun Microsystems workstations UH-5(XRAY)
VT terminals UI-5(XRAY)

Command line
syntax U2-1(CCC)

Command line continuation character
linker U3-21(ASM)

Command line flags
assembler
UNIX/DOS U2-5(ASM)–U2-11(ASM)
VAX/VMS U3-4(ASM)–U3-11(ASM)

Command line options
DOS U2-1(MCC)
-specifying in file
-d option U2-19(MCC), U2-15(CCC)
UNIX U2-1(MCC)
VMS U3-1(MCC)

COMMAND linker command line option U3-13(ASM)

Command position dependencies,
linker R10-3(ASM)

Command viewport U2-18(XRAY), U3-1(XRAY), U3-12(XRAY)
assembly-level U3-2(XRAY)

Commands U3-1(FLEX)
administrative U3-4(FLEX)
arch (Sun) IA-1(FLEX)
cd I1-6(FLEX)
xport (sh/ksh) I1-15(FLEX), I2-1(FLEX), U1-1(FLEX), U1-3(FLEX)
hostid I3-2(FLEX), I3-3(FLEX), U3-4(FLEX)
hostname I3-2(FLEX)
kll I2-8(FLEX)
librarian
(See Librarian commands)
linker
(See Linker commands)
Imdown I2-8(FLEX), U3-4(FLEX)

lmgrd I2-7(FLEX)
lmhostid U3-4(FLEX)
rmremove U3-5(FLEX)
rmreread U3-6(FLEX)
lmstat I2-4(FLEX)

ln I1-15(FLEX), IA-1(FLEX)
mcc68k U1-2(FLEX), U1-3(FLEX)
mkdir I1-6(FLEX)
mllicense
(see mllicense command)
mrcheckin IC-1(FLEX)
mrcheckout IC-1(FLEX)
mt I1-19(FLEX)
rbak (Apollos) I1-7(FLEX)
rcp U2-11(FLEX)
rsh I1-8(FLEX), I2-7(FLEX)
set (csh) I2-1(FLEX)
setenv (csh) I1-15(FLEX), I2-1(FLEX), U1-1(FLEX), U1-3(FLEX)
tail I2-4(FLEX)
tar I1-7(FLEX), I1-19(FLEX)
user U3-1(FLEX)

Comment indicators
librarian R13-1(ASM)
linker R10-21(ASM)

Comment linker command R10-21(ASM)

Comment statement R2-4(ASM)

Comments R2-8(MCC)
/* and */ tokens R4-7(CCC)
// token R4-7(CCC)
command files R3-4(XRAY)
examples R4-7(CCC)
include files R3-4(XRAY)
micro R4-3(XRAY)

Comments, representing R4-7(CCC)

Comments, saving in preprocessor output
/preserve_comments option U3-29(MCC)
-C option U2-17(MCC), U2-14(CCC)

COMMON assembler directive R5-9(ASM)–R5-10(ASM)

COMMON linker command R10-22(ASM)–R10-23(ASM)
Common subexpression optimization R5-6(XRAY)

Communications timeout S2-92(SUP)

COMPARE command R3-41(XRAY)
Comparing
  file contents to memory
  SETSTATUS VERIFY command R3-177(XRAY)
memory
  COMPARE command R3-41(XRAY)
strings R4-31(XRAY), R4-33(XRAY), R4-35(XRAY)

Comparing characters in memory R5-119(MCC)

Comparing two strings R5-171(MCC)

Comparing values R5-112(MCC), R5-123(MCC)

Compatibility
  between Flexible License Manager
  versions IC-1(FLEX), IC-2(FLEX)
  with other Highland Software
  vendors IC-5(FLEX)

Compatibility between C and C++ R4-1(CCC)

Compatibility with MRI toolchain S1-1(SUP)

Compiler
  command line U2-1(MCC), U3-1(MCC)
  description U1-1(MCC)
  input file extensions U2-2(MCC), U3-2(MCC)
invoking U2-1(MCC), U3-1(MCC)
syntax U2-1(MCC), U3-1(MCC)

Compiler constant
  _cplusplus R5-2(CCC), RB-2(CCC)
cplusplus RB-2(CCC)
NULL RB-2(CCC)

Compiler driver U2-1(CCC)

Compiler features R1-1(MCC), R1-2(CCC)

Compiler options
  UNIX/DOS
    -A R4-3(MCC)
    -e R9-24(MCC)
    -f R4-3(MCC)
    -g R4-3(MCC)
  -Kf R1016(MCC)
  -Kh R9-34(MCC)
  -Kr R9-45(MCC)
  -KT R4-3(MCC)
  -Ku R4-3(MCC), R6-10(MCC)
  -Mcp R4-3(MCC), R9-19(MCC), R9-20(MCC)
  -Md R4-3(MCC), R4-4(MCC)
  -Mdn R9-19(MCC), R9-20(MCC)
  -Mdp R9-19(MCC), R9-20(MCC)
  -nKu R4-3(MCC)
  -nOc R1017(MCC)
  -p R4-4(MCC)
  -utn R4-3(MCC)
  -x R4-3(MCC)

VMS
  /align_check R4-3(MCC)
  /ansi R4-3(MCC)
  /code R9-19(MCC), R9-20(MCC)
  /code_address R4-3(MCC)
  /cpu R4-4(MCC)
  /data R9-19(MCC), R9-20(MCC)
  /debug R4-3(MCC)
  /fpu R4-3(MCC)
  /frames R1016(MCC)
  /ireturn R9-45(MCC)
  /long R9-21(MCC), R9-23(MCC)
  /mri_extensions R4-3(MCC)
  /nounsignedchar R4-3(MCC)
  /optimize R1017(MCC)
  /reserve R9-34(MCC)
  /truncate_identifiers R4-3(MCC)
  /unsigned char R6-10(MCC)
  /unsignedchar R4-3(MCC)

Compiler output
  assembler source file R11-1(MCC)
  listing R11-3(MCC)

Compiler-generated literals section U2-16(MCC), U3-15(MCC), U2-13(CCC)

  naming
    /rename option U3-31(MCC)
    -NL option U2-29(MCC), U2-25(CCC)

specifying address mode
Compiler-generated tag data section U2-16(MCC), U3-15(MCC), U2-13(CCC)
/trace option U3-33(MCC)
-Kt option U2-26(MCC), U2-23(CCC)
Completeness testing I2-1(FLEX)
Complex breakpoints S2-10(SUP), S4-23(SUP)
Complex data types R6-4(MCC)
Complex expression R4-8(ASM)
Complex RAM test S2-36(SUP)
Complex relocatable expressions R4-8(ASM)
Complex straddling R6-24(MCC)
Components of the ASM68K package U1-1(ASM)
Compound statement R3-19(MCC)
Concatenating two strings R5-162(MCC), R5-170(MCC), R4-29(XRAY)
Conditional compilation directives R4-8(MCC)
  #elif R4-7(MCC), R4-13(MCC)
  #else R4-7(MCC), R4-14(MCC)
  #endif R4-7(MCC), R4-17(MCC)
  #ifdef R4-4(MCC), R4-19(MCC)
  ifndef R4-7(MCC), R4-20(MCC)
Conditional expressions R3-9(MCC)
Conditions S4-14(SUP)
CONFIG.SYS
DOS I1-3(MCC), I1-3(XRAY), I1-4(CCC)
Configurations, memory R9-15(MCC), R9-18(MCC), R9-42(MCC), R9-44(MCC)
Conflicting options, specifying U2-10(MCC), U2-6(CCC)
const
  avoiding confusing declarations R4-9(CCC)
const member function RC-1(CCC), UA-1(CCC)
const object RC-1(CCC), UA-1(CCC)

const section R8-4(MCC), U3-15(MCC)
  naming
    /rename option U3-30(MCC)
    -NC option U2-28(MCC), U2-25(CCC)
  specifying address mode
    /const_addr_as options U3-16(MCC)
    -ac options U2-17(MCC), U2-13(CCC)
const type R3-17(MCC)
Constant (definition) R3-3(XRAY)
Constant expressions R3-6(MCC), R3-17(MCC)
Constant folding optimization R109(MCC), R5-6(XRAY)
Constant literals section
  -al options U2-17(MCC), U2-14(CCC)
Constant variables section U3-15(MCC)
  naming
    /rename option U3-30(MCC)
    -NC option U2-28(MCC), U2-25(CCC)
  specifying address mode
    -ac options U2-17(MCC), U2-13(CCC)
Constants R2-1(MCC), R2-3(XRAY)
  (see Assembler constants)
    character strings R2-5(XRAY)
    floating-point R2-4(XRAY)
    hexadecimal R2-3(XRAY)
    integer R2-3(XRAY)
Constructor R2-11(CCC), RA-9(CCC), RA-13(CCC)
  definition RC-1(CCC), UA-1(CCC)
Context button
X Window support UA-12(XRAY)
context button
  SunView support UA-12(XRAY)
CONTEXT command R3-43(XRAY)
/F option button
  X Window support UA-12(XRAY)
Continuation character (#) U2-17(ASM), U3-17(ASM), R9-11(ASM), R10-49(ASM), R10-59(ASM)
Continuation character (%) R3-5(XRAY), U2-9(XRAY)
Continuation on command line minus sign U2-21(ASM), U3-18(ASM)
Continue program execution GO command R3-69(XRAY)
continue statement R3-19(MCC)
CONTINUE statement in macros R4-4(XRAY)
Control character functions U3-36(XRAY)
Control character, testing for R5-86(MCC)
Conversion operator () R4-11(CCC)
Conversion programs IEE2AOUT U2-24(ASM)–U2-28(ASM), U3-21(ASM)–U3-26(ASM)
COPY command R3-44(XRAY)
Copying characters from memory R4-25(XRAY)
memory block
COPY command R3-44(XRAY)
strings R4-32(XRAY)
Copying a string R5-172(MCC)
Copying characters from memory R5-116(MCC), R5-120(MCC), R5-121(MCC)
Copying one string to another string R5-166(MCC)
cos function R5-40(MCC), U2-40(MCC), U3-25(MCC), U2-37(CCC)
cosh function R5-41(MCC), U2-39(MCC), U3-25(MCC), U2-37(CCC)
Cosine of a number, computing R5-40(MCC)
Count occurrences S6-26(SUP)
Counted licenses U1-13(FLEX), I3-3(FLEX), U2-5(FLEX), U2-10(FLEX), UB-1(FLEX)
checking out U1-2(FLEX)
errors I2-6(FLEX)
start-up I2-3(FLEX)
testing I2-3(FLEX), I2-5(FLEX)
Counter overflow S6-7(SUP), S6-10(SUP)
CPAGE linker command R10-24(ASM)–R10-25(ASM)
CPU
bus S3-27(SUP)
not supported S1-7(SUP)
CPU simulation RF-1(XRAY)
68030 support RF-9(XRAY)
68851 instruction support RF-11(XRAY)
68881 floating-point support RF-11(XRAY)
BKPT instruction RF-7(XRAY)
CALLM and RTM instructions RF-8(XRAY)
CPU space references RF-7(XRAY)
cycle times RF-5(XRAY)
@cycles pseudo-register RF-5(XRAY)
@wait_state pseudo-register RF-5(XRAY), RF-6(XRAY)
exception handling RF-5(XRAY)
@exc pseudo-register RF-5(XRAY)
instruction alignment RF-4(XRAY)
memory initialization RF-6(XRAY)
trace bits RF-7(XRAY)
CPU space and coprocessor communication RF-7(XRAY)
CPU32 support RF-10(XRAY)
CRC (calculate a crc for a range of memory) S2-31(SUP)
create assembler command line flag U2-7(ASM), U3-7(ASM)
creat function R5-42(MCC), U2-52(MCC), U3-43(MCC), U2-44(CCC)
relationship to close R5-39(MCC)
relationship to read R5-142(MCC)
relationship to write R5-202(MCC)
CREATE librarian command R13-7(ASM)
Create screen
VOPEN command R3-208(XRAY)
Creating macros
DEFINE command R3-45(XRAY)
symbols
ADD command R3-18(XRAY)
viewports
VOPEN command R3-208(XRAY)

Creating a file R5-42(MCC)

Creating license file I1-9(FLEX)
features, specifying I1-12(FLEX), I1-14(FLEX)
servers, specifying I1-11(FLEX), I1-12(FLEX)

Creating operators R2-19(CCC)

Creating scripts for /usr/mri/bin 12-2(FLEX)

Cross-checking in C++ R5-1(CCC)

Cross-jump optimization R5-13(XRAY)

Cross-reference option R8-3(ASM)

ctime function R5-20(MCC), R5-43(MCC)

CTRL-L U3-35(XRAY)

ctype.h include file R5-3(MCC)

Current context
CONTEXT command R3-43(XRAY)

Current directory
SETSTATUS DIR command R3-159(XRAY)

Current module
SCOPE command R3-152(XRAY)

Current procedure (definition) RD-1(XRAY)

Cursor control keys
Apollo workstations UB-2(XRAY)
DECstation terminals UC-3(XRAY)
HP terminals UD-4(XRAY)
IBM RS/6000 UE-1(XRAY)
Motorola Delta Series workstations UF-2(XRAY)
PC/DOS UG-2(XRAY)
Sun Microsystems workstations UH-2(XRAY)
VT terminals UI-1(XRAY)

Cursor position in viewport
VSETC command R3-212(XRAY)

@cycles pseudo-register RA-1(XRAY), RF-1(XRAY)

-D assembler option U2-2(ASM)
-d librarian command line option U2-21(ASM)
-D option U2-18(MCC), U2-14(CCC)
+d option U2-15(CCC)
-d option U2-19(MCC), U2-15(CCC)

Daemon U2-3(FLEX)

DAEMON line
(see License file)

Daemon options file I3-3(FLEX), I3-4(FLEX), I3-5(FLEX), UB-1(FLEX)
comment line I3-5(FLEX)
EXCLUDE line I3-6(FLEX)
format I3-5(FLEX)
INCLUDE line I3-6(FLEX)
NOLOG line I3-6(FLEX)
RESERVE line I3-5(FLEX)

Daemons
license manager
(see lmgrd)
license server
(see License server daemons)
mlicense.daemon I2-3(FLEX)
vendor
(see MRI daemon)

Data
C++ structures
defining R5-10(CCC)
formats R6-1(MCC)
initialization R9-39(MCC)
position-dependent R9-19(MCC)
position-independent R9-19(MCC)
system R9-37(MCC)
types R3-1(MCC), R3-2(MCC), R6-3(MCC)
allocation R3-1(MCC), R6-3(MCC)
double R3-2(MCC)
float R3-2(MCC)
keyword R4-6(CCC)
pointer R3-2(MCC)
ranges R6-3(MCC)

Data bus bandwidth R6-23(MCC), R6-29(MCC)

Data flow diagram U1-4(MCC)
Data flow program diagram  U1-3(ASM)
Data member    R2-1(CCC), RC-1(CCC), UA-1(CCC)
/data option (VMS)  R9-19(MCC), R9-20(MCC)
Data record
   S1 format   RF-2(ASM)
Data references
   absolute
      /data_addresses=absolute option  U3-14(MCC)
      -Mda option  U2-27(MCC), U2-24(CCC)
   PC-relative
      /data_addresses=pcrelative
      option  U3-14(MCC)
      -Mdp option  U2-28(MCC), U2-25(CCC)
   register-relative
      /data_addresses=anrelative
      option  U3-14(MCC)
      -Md options  U2-27(MCC), U2-24(CCC)
Data space
   allocating  R5-35(MCC), R5-111(MCC), R5-203(MCC)
   deallocating  R5-67(MCC)
Data type alignment  R6-30(MCC)
Data type conversion  R2-17(XRAY)
Data types
   (see Symbolic references, data types)
Data viewport  U2-18(XRAY), U3-1(XRAY), U3-13(XRAY)
   assembly-level  U3-2(XRAY)
Date
   conversion to string  R5-25(MCC), R5-43(MCC)
__DATE__  preprocessor symbol  R4-1(MCC)
DC assembler directive  R5-11(ASM)??
DCB assembler directive  R5-14(ASM)
Dead (unreachable) code elimination  R5-2(XRAY)
Dead code elimination  R102(MCC)
Dealocating

data space  R5-67(MCC)
dynamic storage  R4-2(CCC)
/debug option (VMS)  R11-3(MCC)
DEBUG_SYMBOLS command  R10-26(ASM)
Debugger commands
   command file comments  R3-4(XRAY)
   command syntax  R3-2(XRAY)
   command parameters  R3-3(XRAY)
   command qualifiers  R3-2(XRAY)
   size qualifiers  R3-3(XRAY)
defining
   ALIAS command  R3-20(XRAY)
entering  R3-2(XRAY)
saving in file  R3-91(XRAY)
Debugger invocation  S1-2(SUP)
Debugger macros  R4-1(XRAY)
   assignment operators  RC-2(XRAY)
attached to viewport
   VMACRO command  R3-206(XRAY)
commands
   SHOW  R3-179(XRAY)
comments  R4-3(XRAY)
conditional statements
   BREAK  R4-4(XRAY)
   CONTINUE  R4-4(XRAY)
   DO-WHILE  R4-4(XRAY)
   FOR  R4-5(XRAY)
   IF  R4-6(XRAY)
   IF-ELSE  R4-7(XRAY)
   RETURN  R4-7(XRAY)
   WHILE  R4-7(XRAY)
displaying
   SHOW command  R3-179(XRAY)
execution after instruction
   GOSTEP command  R3-71(XRAY)
keywords as names  R4-10(XRAY)
macro definition  R4-8(XRAY)
   comments  R4-3(XRAY)
   file  R4-9(XRAY)
   INCLUDE command  R4-9(XRAY)
interactive  R4-8(XRAY)
local symbols  R4-8(XRAY)
macro body  R4-3(XRAY)
saving macros  R4-9(XRAY)
macro invocation R4-10(XRAY)
predefined macros R4-13(XRAY)
byte R4-14(XRAY)
dword R4-15(XRAY)
error R4-16(XRAY)
getc R4-17(XRAY)
import R4-18(XRAY)
isalive R4-20(XRAY)
key_get R4-21(XRAY)
key_stat R4-22(XRAY)
memcpy R4-25(XRAY)
memset R4-26(XRAY)
outport R4-27(XRAY)
strcat R4-29(XRAY)
strchr R4-30(XRAY)
strcmp R4-31(XRAY)
strcpy R4-32(XRAY)
stricmp R4-33(XRAY)
strlen R4-34(XRAY)
strncmp R4-35(XRAY)
until R4-36(XRAY)
when R4-37(XRAY)
word R4-38(XRAY)
properties R4-11(XRAY)
return values R4-10(XRAY)
saving R4-9(XRAY)
source patches R4-12(XRAY)
stop execution R4-10(XRAY)
use with breakpoints R4-2(XRAY)
use with viewports R4-2(XRAY)

Debugger symbols R3-18(XRAY)

Debugger variable values S2-80(SUP), S2-162(SUP)

Debugger, XRAY
description U1-3(CCC)

Debugging
assembly mode
-- notation R2-12(XRAY)
multiple statements R2-11(XRAY)

Debugging in assembly-level mode U1-2(XRAY), U2-20(XRAY)

BREAKINSTRUCTION command U2-21(XRAY)
CLEAR command U2-20(XRAY), U2-24(XRAY)
GO command U2-22(XRAY), U2-24(XRAY)
MODE command U2-20(XRAY)
OPTION LINES U2-20(XRAY)
PRINTVALUE command U2-22(XRAY)
QUIT command U2-24(XRAY)
RESTART command U2-20(XRAY)
STEP command U2-22(XRAY)
tutorial U2-20(XRAY)
VSCREEN command U2-24(XRAY)

Debugging in high-level mode U1-3(XRAY), U1-4(XRAY), U2-12(XRAY)
BREAKINSTRUCTION command U2-14(XRAY), U2-18(XRAY)
C expressions and statements U1-3(XRAY)
GO command U2-18(XRAY)
MONITOR command U2-18(XRAY)
PRINTVALUE command U2-18(XRAY)
screen U2-14(XRAY)
single-stepping U2-17(XRAY)
STEP command U1-4(XRAY), U2-16(XRAY)
STEP OVER command U1-4(XRAY), U2-19(XRAY)
tutorial U2-12(XRAY)

Debugging information
fully qualified path names
/debug=fullfilename option U3-19(MCC)
-Gf option U2-21(MCC), U2-18(CCC)
generating
/debug option U3-18(MCC)
-g option U2-22(MCC), U2-19(CCC)
line number information
/debug=lines option U3-19(MCC)
-Gl option U2-21(MCC), U2-18(CCC)
multiple statements on line
/debug=multi_stmt option U3-19(MCC)
-Gm option U2-21(MCC)
restricted
/debug=restricted option U3-20(MCC)
-Gr option U2-21(MCC), U2-18(CCC)
XRAY Source Explorer
-Gs option U2-22(MCC), U2-19(CCC)
Debugging multiple modules U2-10(XRAY)
Decimal interpretation
RADIX option R3-107(XRAY)
Decimal representation RA-1(MCC), RB-1(XRAY)
Declarations R3-16(MCC)
variables R4-8(CCC)
DECstation
host-specific information UC-1(XRAY)
Default command, example use and
description U2-41(MCC)
Default initialization R6-42(MCC)
default keyword R3-23(MCC)
Defaults, specifying in install.sh I1-5(FLEX)
DEFINE assembler command line option U3-3(ASM)
DEFINE command R3-45(XRAY), U2-26(XRAY)
#define directive R4-6(MCC), R4-8(MCC), R4-10(MCC), R9-6(MCC)
defined operator R4-13(MCC), R4-17(MCC)
Defining
debugger commands R3-20(XRAY)
input port address
INPORT command R3-80(XRAY)
micros
DEFINE command R3-45(XRAY)
output port address
OUTPORT command R3-110(XRAY)
performance analysis measurement
PROFILE command R3-134(XRAY)
Defining C++ operations R5-13(CCC)
Defining colors UA-3(XRAY)
Defining macros U2-26(XRAY)
interactive U2-27(XRAY)

Defining macros on command line
/define option U3-20(MCC)
-D option U2-18(MCC), U2-14(CCC)
Defining screens and viewports U3-32(XRAY)
FPRTTF command U1-5(XRAY), U3-32(XRAY)
VMACRO command U3-32(XRAY)
VOPEN command U3-32(XRAY), U3-34(XRAY)
VSSCREEN command U3-33(XRAY)
VSSCREEN command key U3-33(XRAY)
Definitions
address R3-3(XRAY)
address_range R3-3(XRAY)
constant R3-3(XRAY)
event R3-161(XRAY)
expression R3-3(XRAY)
expression_range R3-4(XRAY)
expression_string R3-4(XRAY)
frame R3-171(XRAY)
line_number R3-4(XRAY)
qualified bus cycle R3-167(XRAY)
symbol R3-4(XRAY)

Del key (IBM RS/6000 support) UE-2(XRAY)
Del key (Motorola Delta Series support) UF-2(XRAY)
Del key (PC support) UG-3(XRAY)
Del key (Sun support) UH-3(XRAY)
Del Line key (HP support) UD-5(XRAY)
Delete Char key (HP support) UD-5(XRAY)
DELETE command R3-48(XRAY)
Delete key (Apollo support) UB-3(XRAY)
delete keyword R4-2(CCC)
DELETE librarian command U2-21(ASM), U3-19(ASM), R13-8(ASM)
DELETE librarian command line option U3-19(ASM)
delete operator R2-21(CCC), RC-2(CCC), UA-1(CCC)
Deleting
symbols
DELETE command R3-48(XRAY)
viewports
VCLOSE command R3-205(XRAY)
Deleting operators R2-21(CCC)
Demand loading
DEMANDLOAD option R3-106(XRAY)
Demangling RG-1(ASM)—RG-4(ASM), RC-2(CCC), UA-2(CCC)
C++ names U2-46(CCC)
DEMO as host ID I3-3(FLEX)
DEMO licenses l1-1(FLEX), U2-2(FLEX)
host ID I3-3(FLEX)
installing I1-11(FLEX)
Demonstration licenses U2-2(FLEX)
Denying access to users I3-6(FLEX)
Dereferenced variable R2-22(XRAY)
Derived class RC-2(CCC), UA-2(CCC)
Derived classes
definition R3-1(CCC)
Description of documentation IP-2(FLEX), UP-1(FLEX)
Destructor R2-12(CCC)
definition RC-2(CCC), UA-2(CCC)
Device file for install
determining I1-18(FLEX)
icorrect I1-19(FLEX)
DIAG 0 (simple target ram test) S2-34(SUP)
DIAG 1 (complex target ram test) S2-36(SUP)
DIAG 2 (continuous read from target memory) S2-38(SUP)
DIAG 3 (continuous write to target memory) S2-40(SUP)
DIAG 4 (write alternating pattern to target location) S2-42(SUP)
DIAG 5 (continuous write to target memory) S2-44(SUP)
DIAG 6 (continuous write to target memory) S2-46(SUP)
DIAG 7 (continuous read from target memory) S2-48(SUP)
DIAG 8 (write increment value to target memory) S2-50(SUP)
DIAG 9 (continuous stream of reset pulses) S2-52(SUP)
Diagnostic macros R5-2(MCC)

assert R5-27(MCC)

diagtime function R5-44(MCC)

Digit
testing for a R5-87(MCC)
testing for hexadecimal R5-94(MCC)

DIN command R3-50(XRAY)

Direct memory addressing R9-9(MCC)
direction command key
DECstation trace support UC-6(XRAY)
HP trace support UD-8(XRAY)
IBM RS/6000 trace support UE-5(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC trace support UG-5(XRAY)
Sun trace support UH-5(XRAY)
VT terminal trace support UI-5(XRAY)
direction trace command key
Apollo trace support UB-6(XRAY)

Directive statement R2-3(ASM)

Directives, assembler
ALIGN R5-4(ASM)
CHIP R5-5(ASM)—R5-7(ASM)
COMLINE R5-8(ASM)
COMMON R5-9(ASM)—R5-10(ASM)
DC R5-11(ASM)—??
DCB R5-14(ASM)
DS R5-15(ASM)
ELSESEC R5-16(ASM)
END R5-17(ASM)—R5-18(ASM)
ENDC R5-19(ASM)
ENDR R5-20(ASM)
EQU R5-21(ASM)
FAIL R5-22(ASM)—R5-23(ASM)
FEQU R5-24(ASM)—R5-25(ASM)
FOPT R5-26(ASM)
DISASSEMBLE command  R3-52(XRAY), U3-11(XRAY), U4-2(XRAY)  
with C++  RG-7(XRAY)  
Disassembled code, displaying  U4-2(XRAY)  
Disassembled trace display  S1-17(SUP)  
Disassembled Trace, display forward  S2-67(SUP)  
Disassembling memory  
DISASSEMBLE command  R3-52(XRAY)  
Display commands  R3-7(XRAY)  
DISASSEMBLE  R3-7(XRAY), R3-52(XRAY)  
DUMP  R3-7(XRAY), R3-57(XRAY)  
EXPAND  R3-7(XRAY), R3-60(XRAY)  
FIND  R3-7(XRAY), R3-64(XRAY)  
FOPEN  R3-7(XRAY), R3-66(XRAY)  
PRINTF  R3-7(XRAY), R3-67(XRAY)  
LIST  R3-7(XRAY), R3-87(XRAY)  
MONITOR  R3-7(XRAY), R3-93(XRAY)  
NEXT  R3-8(XRAY), R3-97(XRAY)  
NOMONITOR  R3-8(XRAY), R3-102(XRAY)  
PRINTF  R3-8(XRAY), R3-118(XRAY)  
PRINTVALUE  R3-8(XRAY), R3-131(XRAY)  
Display device  U2-1(FLEX), UB-1(FLEX)  
specifying other than host  U3-1(FLEX)  
Display emulator variable values command  S2-70(SUP)  
DISPLAY environment variable  UH-2(XRAY)  
Display raw trace  S2-57(SUP)  
Display time stamp  
-Vt option  U2-38(MCC), U2-35(CCC)  
Display Trace Backwards command  S2-65(SUP)  
Display trace backwards command  S2-65(SUP)  
Display trace command  S2-59(SUP), S2-60(SUP)  
Displaying  
addresses  
NOSYMBOLS command  R3-108(XRAY)  
Break viewport  R3-104(XRAY)  
input port buffer  
DIN command  R3-50(XRAY)  
local variables  
EXPAND command  R3-60(XRAY)  
macro contents  
SHOW command  R3-179(XRAY)  
memory contents  
DUMP command  R3-57(XRAY)  
output port buffer  
DOUT command  R3-54(XRAY)  
results of performance analysis  
PRINTPROFILE command  R3-123(XRAY)  
source code  
LIST command  R3-87(XRAY)  
symbols  
NOSYMBOLS commands  R3-108(XRAY)  
PRINTSYMBOLS command  R3-127(XRAY)  
variables  
PRINTVALUE command  R3-131(XRAY)  
View viewport  R3-108(XRAY)  
Distribution  
contents of  I1-2(FLEX)  
reading  I1-7(FLEX), I1-8(FLEX), I1-16(FLEX)  
specifying programs not to read  I1-17(FLEX)  
serial number  I1-3(FLEX), I1-10(FLEX), U1-5(FLEX)  
Distribution format  
DOS  I1-1(CCC)  
div function  R5-45(MCC)  
Division  
computing quotient  R5-45(MCC), R5-99(MCC)
computing remainder R5-45(MCC), R5-99(MCC)

**DMA**
controlling trace/overlay S2-139(SUP), S2-141(SUP)

**DNL(download hex file to target)** S2-53(SUP)
**DNL_GAP** - specify maximum bytes between blocks S2-55(SUP)
**DNLFMT** (specify download format) S2-54(SUP)

**Documentation description** IP-2(FLEX), UP-1(FLEX)

**DOMAIN/OS**
host-specific information UB-1(XRAY)
mouse support
(see Mouse support)

**DOS**
(see UNIX/DOS)
accessing files in nondefault directories I1-5(MCC), I1-5(CCC), I1-6(CCC)
command line U2-1(MCC)
command line examples U2-41(MCC)
command line syntax U2-1(CCC)
compiler invocation U2-1(CCC)
compiler syntax U2-1(MCC)
compiler use U2-1(MCC), U2-1(CCC)
compiling a program U2-53(MCC), U2-45(CCC)
defaults I1-1(MCC), I1-1(XRAY), I1-1(CCC)
distribution format I1-1(MCC), I1-1(XRAY), I1-1(CCC)
environment variables U2-3(MCC)
file locations U2-3(MCC), U2-3(CCC)
file name defaults U2-2(MCC)
hard disk I1-1(MCC), I1-1(XRAY), I1-1(CCC)
host-specific information UG-1(XRAY)
installation diskettes I1-1(CCC)
installation procedure I1-2(MCC), I1-2(CCC)
invoking compiler U2-1(MCC)
libraries provided U2-50(MCC)

memory considerations U2-7(MCC)
option form, positive and negative U2-10(MCC)
option, command line descriptions of U2-10(MCC)

**DOS installation**
installation procedure I1-2(XRAY)
using XRAY I1-6(XRAY)

**Dot (.) operator** R2-17(CCC), R2-19(CCC), R2-21(CCC), R2-25(CCC)
**Dot-star (.* ) operator** R2-18(CCC), R2-25(CCC)

**Double**
conversion from ASCII string R5-177(MCC)
double type R3-1(MCC), R6-3(MCC)
Double, computing integer ceiling of R5-36(MCC)
**Double-precision format** R6-7(MCC)
**DOUT command** R3-54(XRAY)
do-while statement R3-20(MCC)
**DO-WHILE statement in macros** R4-4(XRAY)
**DOWN command** R3-56(XRAY)

**Download** S2-53(SUP)

timeout S2-92(SUP), S2-93(SUP)

**Download format** S2-54(SUP)

**Driver** U2-1(MCC)

**Driver, compilation** U2-1(CCC)
**DRT command** S2-57(SUP)
**DRUN (start dynamic run mode)** S2-60(SUP)
**DS assembler directive** R5-15(ASM)
**DSTOP (stop dynamic run mode)** S2-62(SUP), S2-69(SUP)
**DT S2-59(SUP), S2-60(SUP), S2-65(SUP)**

**DTACK signal**
bus timeout switch S2-25(SUP)
**DTB S2-65(SUP)**

**DTF command** S2-67(SUP)
**DUMP command** R3-57(XRAY)
size qualifiers R3-3(XRAY)

**Duplicate function names** R5-1(CCC)
dword macro R4-15(XRAY)
dynamic operations S1-18(SUP)
Dynamic run
  drun S2-60(SUP)
  dstop S2-62(SUP)
Dynamic storage
  allocating
    keywords R4-4(CCC)
  deallocating
    keywords R4-2(CCC)

E
  e assembler command line flag U2-7(ASM), U3-7(ASM)
  -e librarian command line option U2-21(ASM)
  +E option U2-16(CCC)
  -E option U2-19(MCC), U2-16(CCC)
  +e option U2-16(CCC)
  -e option U2-19(MCC), U2-16(CCC)
  -e option (UNIX/DOS) R9-24(MCC)
  +e option, use of R3-18(CCC)
  EBCDIC character code RA-1(ASM)-RA-2(ASM)
  EBSIG R5-155(MCC)
  Echo include files
    INCECHO option U4-12(XRAY)
  Echo macros
    INCECHO option U4-12(XRAY)
  EDOM error macro R5-4(MCC)
  EET R3-124(XRAY)
  Effective address syntax R3-14(ASM)
  Elapsed time S6-16(SUP)
    A to B S6-16(SUP)
    in range S6-16(SUP), S6-20(SUP)
    inter-module S6-16(SUP)
    measurement S6-2(SUP)
    out-of-module S6-16(SUP)
  Elapsed time measurement S6-16(SUP)
  #elif directive R4-7(MCC), R4-13(MCC)
  #else directive R4-7(MCC), R4-8(MCC), R4-14(MCC)
  ELSESEC assembler directive R5-16(ASM)
  Embedded systems
    code organization R8-1(MCC)
    considerations R9-1(MCC)
    user-modified routines R9-29(MCC)
Embedded systems considerations S3-3(SUP)
Emulator
  communication
    ICE command R3-77(XRAY)
    NOICE command R3-98(XRAY)
  passing information to
    EMULATOR option R3-106(XRAY)
Emulator variables S2-70(SUP)
  EMUVARS S2-70(SUP)
END assembler directive R5-17(ASM)-R5-18(ASM)
END librarian command R13-10(ASM)
END linker command R10-27(ASM)
ENDC assembler directive R5-19(ASM)
#ENDIF directive R4-7(MCC), R4-8(MCC), R4-15(MCC)
Ending XRAY session
  QUIT command R3-138(XRAY)
ENDM assembler directive R6-6(ASM)
end-of-line character R4-7(CCC)
ENDR assembler directive R5-20(ASM)
Entering host environment from XRAY
  HOST command R3-75(XRAY)
Entering input to install.sh l1-5(FLEX)
ENTRY pseudo-register RA-1(XRAY), RF-1(XRAY)
enum type R3-2(MCC), R6-3(MCC), R6-8(MCC)
Enumerated data types
  (See also enum type)
    packing R6-40(MCC)
Enumeration
  and scope R4-17(CCC)
  declaring within structures R4-17(CCC)
  types R4-15(CCC)
    reference by value R4-15(CCC)
Enumeration types R6-8(MCC)
Enumerator types, packed
  /define option U3-21(MCC)
  -D option U2-18(MCC), U2-15(CCC)
Environment variables

U2-3(MCC), I2-1(FLEX), I2-2(FLEX), U1-1(FLEX), U1-1(FLEX), U2-4(ASM)
DISPLAY UH-2(XRAY)
DOS I1-4(MCC), I1-5(XRAY), I1-5(CCC)
MR1_68K_BIN U2-4(MCC), U2-6(MCC), U2-5(CCC)
MR1_68K_INC U2-4(MCC), U2-7(MCC), U2-6(CCC)
MR1_68K_LIB U2-4(MCC), U2-7(MCC), U2-5(CCC)
MR1_68K_TMP U2-4(MCC), U2-7(MCC), U2-6(CCC)
USR_MRI U2-4(MCC)
GETLICENSE_CONNECT_INTERVAL U2-3(FLEX), UA-1(FLEX)
GETLICENSE_CONNECT_RETRIES U2-3(FLEX), UA-1(FLEX)
GETLICENSE_OUTPUT U2-5(FLEX), U3-3(FLEX), UA-1(FLEX)
GETLICENSE_VERBOSE UA-1(FLEX)

HP support

TERM UD-2(XRAY)
LICENSE_LOG_FILE UA-1(FLEX)
LM_LICENSE_FILE I1-15(FLEX), I2-1(FLEX), I2-6(FLEX), U1-1(FLEX), UA-2(FLEX)
multiple vendors IC-5(FLEX)
MLICENSE_DAEMON_OUTPUT U2-8(FLEX), U3-1(FLEX), UA-2(FLEX)
MLICENSE_DAEMON_TMP U2-3(FLEX), UA-2(FLEX)
MLICENSE_DAEMON_VERBOSE U2-8(FLEX), UA-2(FLEX)
MLICENSE_HOURS U1-3(FLEX), U3-2(FLEX), UA-2(FLEX)
MR1_68K_BIN U2-4(CCC)
MR1_68K_INC U2-4(CCC)
MR1_68K_LIB U2-15(ASM), U2-4(CCC)
MR1_68K_TMP U2-4(CCC)
OWL U2-46(CCC)
PATH I2-1(FLEX), U1-1(FLEX)
TERM UC-2(XRAY), UD-3(XRAY)
TERMINFO UD-4(XRAY)
TMP U2-5(ASM), U2-16(ASM)
UNIX

MRI_68K_BIN U2-4(MCC), U2-6(MCC), U2-5(CCC)
MRI_68K_INC U2-4(MCC), U2-7(MCC), U2-6(CCC)
MRI_68K_LIB U2-4(MCC), U2-7(MCC), U2-5(CCC)
MRI_68K_TMP U2-4(MCC), U2-7(MCC), U2-6(CCC)
USR_MRI U2-4(MCC)

VMS

(see Logical Names) U3-3(MCC)

X Window support

XOR_CHANGE UA-5(XRAY)
XRAY_BG_COLOR UA-3(XRAY)
XRAY_DISPLAY UA-4(XRAY)
XRAY_FONT_COLOR UA-3(XRAY)
XRAY_FONT_PATH environment variable UA-3(XRAY)
XRAY_FONTB UA-1(XRAY)
XRAY_FONTN UA-1(XRAY)

XHS68KLIB U2-5(XRAY)
XOS_OFF RG-1(XRAY)
XRAY R3-1(XRAY), R3-160(XRAY), U2-3(XRAY), U4-10(XRAY)
XRAY_CUR_COLOR UA-3(XRAY)
XRAY_DISPLAY UH-2(XRAY)
XRAYFON R3-105(XRAY), U2-7(XRAY)
XRAYLIB R3-1(XRAY), U2-4(XRAY), U4-11(XRAY)
XRAYTMP U2-7(XRAY), U4-11(XRAY)

EOF R5-14(MCC)

Epilogue R1016(MCC)

function

generating code for

epilogue R1016(MCC)
Epilogue, function R7-8(MCC), R4-2(CCC)
eprintf function R5-20(MCC), R5-46(MCC)
EQU assembler directive R5-21(ASM)
ERANGE error macro R5-4(MCC)
errno error macro R5-4(MCC)
errno.h include file R5-4(MCC)
ERROR command R3-59(XRAY)
#error directive R4-16(MCC)
    relation to #pragma error R4-25(MCC)
Error handling I2-6(FLEX), UC-1(FLEX)
    -15 message I2-3(FLEX)
    calling technical support I1-9(FLEX), I1-10(FLEX)
    cancelling a distribution read I1-17(FLEX)
    cannot find license file message I2-6(FLEX)
    could not connect to mllicense.daemon I2-3(FLEX)
    device drive incorrect I1-19(FLEX)
    features incorrect I1-12(FLEX), I1-14(FLEX)
    message in log file I2-4(FLEX), I2-8(FLEX)
    in install.sh I1-5(FLEX)
    indefinite delay U2-6(FLEX)
    license file
        cannot find I2-6(FLEX)
        no SERVER lines I2-6(FLEX)
        SERVER host name wrong I2-8(FLEX)
    license server daemons not running U1-2(FLEX), U1-3(FLEX)
    license server failure I1-14(FLEX), U2-10(FLEX)
    lmgrd missing I2-6(FLEX)
    log file, handling errors in I2-4(FLEX), I2-7(FLEX), I2-8(FLEX)
    MRI daemon not starting I2-4(FLEX)
    no SERVER lines message I2-6(FLEX)
    servers incorrect I1-12(FLEX), I2-4(FLEX), I2-8(FLEX)
    trying to connect error I3-2(FLEX)

Error handling for include files
    ERROR command R3-59(XRAY)
Error indicator
    checking the file I/O R5-53(MCC)
    clearing the file I/O R5-37(MCC)
ERROR linker command R10-28(ASM)
error macro R4-16(XRAY)
Error macros R5-4(MCC)
    EDOM R5-4(MCC)
    ERANGE R5-4(MCC)
    errno R5-4(MCC)
Error messages S2-81(SUP), RC-1(MCC), RE-1(XRAY)
    (see also Messages)
    abstract class R3-17(CCC)
    assembler RB-1(ASM)--RB-14(ASM)
    librarian RD-1(ASM)--RD-6(ASM)
    linker RC-1(ASM)--RC-11(ASM)
    macros R4-16(XRAY)
    pointing to error message R5-168(MCC)
    sending to standard error R5-128(MCC)
    suppression
        /suppress=error option U3-30(MCC)
        -Qe option U2-36(MCC), U2-33(CCC)
Error viewport U3-14(XRAY)
    PREV CMD command key U3-14(XRAY)
Errors reading absolute files U4-11(XRAY)
    -Es option U2-19(MCC), U2-16(CCC)
Escape key
    Apollo support UB-3(XRAY)
Escape sequences R2-5(MCC)
EV S2-71(SUP)
Event (definition) R3-161(XRAY), RD-1(XRAY)
Event Group S2-88(SUP)
Event group S2-88(SUP)
Event groups
    changing S4-22(SUP)
Event sequence R3-173(XRAY)
    (definition) RD-1(XRAY)
Event System
    setup S6-13(SUP)
Event system S4-14(SUP)
and performance analysis S4-3(SUP)
change event groups S4-22(SUP)
flags

Event system
groups S4-22(SUP)
Event system tutorial S4-3(SUP)
Event variable values S2-80(SUP)
Event, define an S2-71(SUP)

Examples
accessing C++ data structures from C R5-9(CCC)
arrays R4-16(CCC)
calling C functions from C++ R5-2(CCC)–
R5-3(CCC)
calling C++ member functions from C R5-5(CCC)
comment tokens R4-7(CCC)
conversion operator R4-11(CCC)
declaring const for C++ R4-9(CCC)
defining C++ data structures R5-10(CCC)
enumeration types R4-15(CCC)
function declarations R4-12(CCC)
global and local declarations R4-8(CCC)
interrupt keyword R4-3(CCC)–R4-4(CCC)
overloading function names R5-1(CCC)
packed keyword R4-4(CCC), R4-5(CCC)
processing of enumerators R4-17(CCC)
prototype declarations R4-13(CCC)
unpacked keyword R4-4(CCC)–R4-5(CCC)
void pointers R4-7(CCC)

@exc pseudo-register RA-1(XRAY), RF-2(XRAY)

Exception handling RF-2(XRAY), RF-5(XRAY)
keywords R4-2(CCC), R4-6(CCC)

EXCLUDE line I3-6(FLEX)
Excluding users I3-6(FLEX)
Exclusive Execution Time R3-124(XRAY)

Executable file
suppressing
-c option U2-18(MCC), U2-14(CCC)

Executable program
generation U2-53(MCC), U3-45(MCC),
U2-45(CCC)
running on a target system U2-53(MCC),
U3-46(MCC), U2-45(CCC)

Executables
UNIX
alternate locations U2-4(MCC)

Execute program
GO command R3-69(XRAY)

Executing a program U1-2(FLEX)

Executing install.sh I1-8(FLEX)

Executing multiple commands U4-8(XRAY)

Executing only preprocessor
/pp option U3-29(MCC)
-E option U2-19(MCC), U2-16(CCC)
-P option U2-33(MCC), U2-30(CCC)

Executing program U2-18(XRAY)
(see also Single-stepping, GO command)

Executing test scripts I2-2(FLEX)

Executing XRAY U2-9(XRAY)

Execution and breakpoint commands R3-7(XRAY)

BREAKACCESS S2-5(SUP), R3-7(XRAY), R3-25(XRAY)
BREAKINSTRUCTION S2-12(SUP), R3-7(XRAY), R3-30(XRAY)
BREAKREAD S2-16(SUP), R3-7(XRAY), R3-32(XRAY)
BREAKWRITE S2-20(SUP), R3-7(XRAY), R3-35(XRAY)
CLEAR R3-7(XRAY), R3-40(XRAY)
GO R3-7(XRAY), R3-69(XRAY)
EXECUTION breakpoints
  software  S4-6(SUP)
exit function  R6-3(ASM)
_exit function  R5-48(MCC), U2-52(MCC), U3-43(MCC), U2-44(CCC)
exit function  R5-47(MCC)
  relationship to _exit  R5-48(MCC)
  relationship to signal  R5-155(MCC)
EXIT linker command  R10-29(ASM)
exp function  R5-49(MCC), U2-39(MCC), U3-25(MCC), U2-37(CCC)
EXPAND command  R3-60(XRAY), U3-26(XRAY), U4-4(XRAY)
Expand viewport
  ZOOM command  R3-213(XRAY)
Expiration date of features  I3-3(FLEX)
Expiration of licenses  U1-3(FLEX), UA-2(FLEX)
  at specific date  U2-2(FLEX)
  delaying  U1-3(FLEX), U3-2(FLEX)
  speeding up  U3-3(FLEX)
EXPLAIN command  S2-81(SUP)
Explorer, generating code for
  -Gs option  U2-22(MCC), U2-19(CCC)
Exponent from floating-point number  R5-70(MCC)
Exponential value, computing  R5-49(MCC)
export command (sh/ksh)  U1-15(FLEX), I2-1(FLEX), U1-1(FLEX), U1-3(FLEX)
Expression (definition)  R3-3(XRAY)
Expression calculation
  CEXPRESSION command  R3-38(XRAY)
Expression elements  R2-1(XRAY)
  addresses  R2-9(XRAY)
  address ranges  R2-10(XRAY)
  indirect address  R2-9(XRAY)
  line numbers  R2-10(XRAY)
  constants  R2-3(XRAY)
  character strings  R2-5(XRAY)
  floating-point  R2-4(XRAY)
  integer  R2-3(XRAY)
operators  R2-1(XRAY)
  C language  R2-2(XRAY), RC-1(XRAY)
symbols  R2-7(XRAY)
  debugger  R2-8(XRAY)
  keywords  R2-9(XRAY)
  macro  R2-8(XRAY)
  program  R2-7(XRAY)
  reserved  R2-8(XRAY)
Expression string  R3-62(XRAY), R3-154(XRAY), R3-156(XRAY), R3-197(XRAY)
Expression_range (definition)  R3-4(XRAY)
Expression_string
  (definition)  R3-4(XRAY)
Expressions  R3-6(MCC), R2-15(ASM)–R2-17(ASM)
  absolute  R3-24(ASM), R4-8(ASM)
  assembly language  R2-12(XRAY)
  complex  R4-8(ASM)
  expression string  R2-13(XRAY)
  monitoring
    MONITOR command  R3-93(XRAY)
    NOMONITOR command  R3-102(XRAY)
  relocatable  R3-24(ASM), R4-7(ASM)–R4-8(ASM)
  complex  R4-8(ASM)
  simple  R4-8(ASM)
  source language  R2-12(XRAY)
  statement  R3-19(MCC)
  testing program expressions  R5-27(MCC)
Extensions
  file name  U2-2(MCC), U3-2(MCC)
Microtec Research
  (see Microtec Research extensions)
Extensions to C
  mri_extensions option  U3-24(MCC)
  -x option  U2-39(MCC), U2-36(CCC)
extern "C" declaration  R4-14(CCC), R5-2(CCC)
extern declaration  R4-8(CCC)
EXTERN linker command U2-15(ASM), R10-30(ASM)
Extern storage class R3-4(MCC)
keyword R3-5(MCC)
External DMA S2-139(SUP)
External symbols R4-6(ASM)
External variable names R11-1(MCC)
EXTRACT librarian command U2-21(ASM), U3-19(ASM), R13-11(ASM)
EXTRACT librarian command line option U3-19(ASM)
EXVEC command S2-82(SUP)

F
-f assembler command line option U2-3(ASM)
-F linker command line option U2-13(ASM)
-f option U2-20(MCC), U2-18(CCC)
-f option (UNIX/DOS) R4-3(MCC)
F0 key
Apollo support UB-4(XRAY), UC-4(XRAY)
Apollo trace support UB-5(XRAY)
F1 key
Apollo support UB-4(XRAY)
Apollo trace support UB-5(XRAY)
DECstation support UC-4(XRAY)
HP support UD-6(XRAY)
HP trace support UD-7(XRAY)
IBM RS/6000 support UE-3(XRAY)
IBM RS/6000 trace support UE-4(XRAY)
Motorola Delta Series support UF-3(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC support UG-4(XRAY)
PC trace support UG-6(XRAY)
Sun support UH-4(XRAY)
Sun trace support UH-5(XRAY)
F2 key
Apollo support UB-4(XRAY)
Apollo trace support UB-6(XRAY)
DECstation support UC-4(XRAY)
HP support UD-6(XRAY)
HP trace support UD-7(XRAY)
IBM RS/6000 support UE-3(XRAY)
IBM RS/6000 trace support UE-4(XRAY)
Motorola Delta Series support UF-3(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC support UG-3(XRAY)
PC trace support UG-5(XRAY)
Sun support UH-4(XRAY)
Sun trace support UH-5(XRAY)
F3 key
Apollo support UB-4(XRAY)
Apollo trace support UB-6(XRAY)
DECstation support UC-4(XRAY)
HP support UD-6(XRAY)
HP trace support UD-8(XRAY)
IBM RS/6000 support UE-3(XRAY)
IBM RS/6000 trace support UE-5(XRAY)
Motorola Delta Series support UF-3(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC support UG-3(XRAY)
PC trace support UG-5(XRAY)
Sun support UH-4(XRAY)
Sun trace support UH-5(XRAY)
F4 key
Apollo support UB-4(XRAY)
Apollo trace support UB-6(XRAY)
DECstation support UC-4(XRAY)
HP support UD-6(XRAY)
HP trace support UD-8(XRAY)
IBM RS/6000 support  UE-3(XRAY)
IBM RS/6000 trace support UE-5(XRAY)
Motorola Delta Series support  UF-4(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC support  UG-4(XRAY)
PC trace support  UG-5(XRAY)
Sun support  UH-4(XRAY)
Sun trace support  UH-5(XRAY)

**F5 key**
Apollo support  UB-4(XRAY)
Apollo trace support UB-6(XRAY)
DECstation support  UC-4(XRAY)
HP support  UD-6(XRAY)
HP trace support  UD-8(XRAY)
IBM RS/6000 support UE-3(XRAY)
IBM RS/6000 trace support UE-5(XRAY)
Motorola Delta Series support  UF-4(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC support  UG-4(XRAY)
PC trace support  UG-5(XRAY)
Sun support  UH-4(XRAY)
Sun trace support  UH-5(XRAY)

**F6 key**
Apollo support  UB-4(XRAY)
Apollo trace support UB-6(XRAY)
DECstation support  UC-4(XRAY)
HP support  UD-6(XRAY)
HP trace support  UD-8(XRAY)
IBM RS/6000 support UE-3(XRAY)
IBM RS/6000 trace support UE-5(XRAY)
Motorola Delta Series support  UF-4(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC support  UG-4(XRAY)
PC trace support  UG-5(XRAY)
Sun support  UH-4(XRAY)
Sun trace support  UH-5(XRAY)

**F7 key**
Apollo support  UB-5(XRAY)
Apollo trace support UB-6(XRAY)
HP support  UD-6(XRAY)
HP trace support  UD-8(XRAY)
IBM RS/6000 support UE-3(XRAY)
IBM RS/6000 trace support UE-5(XRAY)
Motorola Delta Series support  UF-4(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC support  UG-4(XRAY)
PC trace support  UG-5(XRAY)
Sun support  UH-4(XRAY)
Sun trace support  UH-5(XRAY)

**F8 key**
HP support  UD-7(XRAY)
Sun support  UH-4(XRAY)
Sun trace support  UH-6(XRAY)

**F9 key**
Apollo support  UB-5(XRAY)
Apollo trace support UB-6(XRAY)
DECstation support  UC-4(XRAY)
IBM RS/6000 support UE-4(XRAY)
IBM RS/6000 trace support UE-5(XRAY)
Motorola Delta Series support  UF-4(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC support  UG-4(XRAY)
PC trace support  UG-6(XRAY)
Sun support  UH-4(XRAY)
Sun trace support  UH-6(XRAY)

**fabs function**  R5-50(MCC), U2-39(MCC), U3-25(MCC), U2-37(CCC)
**Factorization optimization**  R5-3(XRAY)
**FAIL assembler directive**  R5-22(ASM)–R5-23(ASM)
**FAST command**  S2-84(SUP)
**Fast interrupts**  S2-84(SUP)
-Fc option  U2-17(CCC)
**fclose function**  R5-51(MCC)
relationship to fflush  R5-54(MCC)
Feature UB-1(FLEX)
  checking out license for
  (see Checking out licenses)
  names
  specifying in mllicense command  U3-3(FLEX)
FEATURE line  U2-4(FLEX), U2-9(FLEX)
  (see License file)
Features
  assembler  R1-1(ASM)--R1-2(ASM)
  expiration date  I3-3(FLEX)
  features versus programs  I1-2(FLEX)
  installation script  I1-5(FLEX)
  librarian  R12-1(ASM)
  linker  R9-2(ASM)
  specifying during install  I1-12(FLEX),
  I1-14(FLEX)
  version number  I3-3(FLEX)
Features of compiler  R1-1(MCC)
Features, XRAY  R1-2(XRAY)
- Fee option  U2-19(MCC)
- Fee option  U2-20(MCC)
f.eof function  R5-12(MCC), R5-52(MCC)
FEQU assembler directive  R5-24(ASM)--R5-25(ASM)
f.error function  R5-12(MCC), R5-53(MCC)
f.fflush function  R5-54(MCC)
f.fgetc function  R5-55(MCC)
  relationship to getc  R5-77(MCC)
  relationship to getchar  R5-78(MCC)
f.fgetc macro  R4-17(XRAY)
f.fgetpos function  R5-56(MCC)
  relationship to fsetpos  R5-73(MCC)
f.fgets function  R5-57(MCC)
File
  associating stream to a different file  R5-68(MCC)
closure  R5-39(MCC), R5-51(MCC)
creation  R5-42(MCC)
current location  R5-72(MCC), R5-73(MCC), R5-74(MCC), R5-108(MCC)
current location in file  R5-56(MCC)
  formatted output  R5-63(MCC), R5-130(MCC), R5-197(MCC), R5-198(MCC)
  formatted read  R5-147(MCC)
  making file inaccessible  R5-144(MCC), R5-192(MCC)
  opening  R5-61(MCC), R5-126(MCC)
  pointing to beginning  R5-145(MCC)
  reading  R5-71(MCC)
  reading a character  R5-55(MCC), R5-77(MCC)
  reading a string  R5-57(MCC)
  reading from a file  R5-66(MCC)
  receiving a character  R5-191(MCC)
  unlinking a file name  R5-192(MCC)
  writing  R5-76(MCC)
  writing a character  R5-134(MCC)
  writing a character to a file  R5-64(MCC)
  writing a string  R5-65(MCC)
File contents compared to memory
  SETSTATUS VERIFY command  R3-177(XRAY)
File descriptor, getting  R5-58(MCC)
File formats
  a.out  U2-24(ASM), U3-22(ASM)
  b.out  U2-24(ASM), U3-22(ASM)
  S-record  U2-13(ASM)
File locations  U2-3(CCC)
File name defaults
  assembler
    UNIX/DOS  U2-4(ASM)
    VAX/VMS  U3-4(ASM)
  librarian
    UNIX/DOS  U2-22(ASM)
    VAX/VMS  U3-20(ASM)
  linker
    UNIX/DOS  U2-16(ASM)
    VAX/VMS  U3-16(ASM)
File name size  R3-1(XRAY)
__FILE__ preprocessor symbol  R4-1(MCC), R4-23(MCC)
@file pseudo-register  RA-1(XRAY), RF-2(XRAY)
File read into memory  
   SETSTATUS READ command R3-169(XRAY)

FILE structure type  R5-14(MCC)

FILE.LIS  
   DOS I1-3(MCC), I1-3(XRAY), I1-3(CCC)
   FILENAME_MAX R5-15(MCC)
   fileno macro R5-58(MCC)

Files  
   #include (see #include files)
   absolute U4-11(XRAY)
   command R3-4(XRAY)
   format for XRAY R3-90(XRAY)
   help U4-11(XRAY)
   include U2-10(XRAY), U2-26(XRAY),
            U3-17(XRAY), U4-8(XRAY)
   input
      extensions U2-2(MCC), U3-2(MCC)
      locations U2-3(MCC), U3-3(MCC)
      journal U2-10(XRAY), U3-17(XRAY), U4-7(XRAY), U4-8(XRAY)
   linker command U2-19(MCC), U2-16(CCC)
   listing
      (see Listing files)
   loading U4-12(XRAY)
   log U2-10(XRAY), U3-17(XRAY)
   output
      extensions U2-3(MCC), U3-2(MCC)
      locations U2-3(MCC), U3-3(MCC)
      source U4-2(XRAY), U4-10(XRAY)
      startup.xry U2-10(XRAY)
      temporary U4-11(XRAY)

FILL command R3-62(XRAY), U4-16(XRAY)
   size qualifiers R3-3(XRAY)

Fill memory with pattern  
   FILL command R3-62(XRAY)

FIND command R3-64(XRAY)

Find specific string  
   FIND command R3-64(XRAY)

Flags S4-22(SUP)

FLAGS assembler command line option U3-3(ASM)

Flexible License Manager operation U2-1(FLEX)

Flexible Licensing functions, performing U3-1(FLEX)
   -Fli option U2-20(MCC), U2-17(CCC)
   float type R3-2(MCC), R6-3(MCC), R6-5(MCC)
   float.h include file R5-4(MCC)
   Floating licenses I1-1(FLEX), U2-1(FLEX),
                UB-1(FLEX)
      checking out U2-4(FLEX)
      mlicense.daemon operation U2-3(FLEX)
      restarting daemons U1-5(FLEX)
      specifying in license file I3-4(FLEX)
      testing installation of I2-3(FLEX)

Floating-point

addressing modes R3-14(ASM)
   constants R2-3(MCC), R2-12(ASM)–R2-13(ASM)
   coprocessor R3-6(ASM)
   double-precision R5-12(ASM)
   format R6-5(MCC)
   number
      calculating exponent R5-70(MCC)
      calculating mantissa R5-70(MCC)
      computing absolute value R5-50(MCC)
      computing product of floating-point
      number and power of two R5-98(MCC)
      conversion from ASCII string R5-31(MCC)
      conversion to ASCII string R5-75(MCC)
      rounding to nearest integer R5-59(MCC)
      remainder, computing R5-60(MCC)
      removing unneeded support R9-33(MCC)
      representation R3-2(MCC), R6-4(MCC)
      return values R7-4(MCC)
      single-precision R5-12(ASM)
      type macros R5-4(MCC)

Floating-point coprocessor instructions, generating
   /fpu option U3-23(MCC)
-f option U2-20(MCC), U2-18(CCC)
floor function R5-59(MCC)
-Flp option U2-20(MCC), U2-17(CCC)
-Flp0 option U2-20(MCC), U2-17(CCC)
-Flt option U2-20(MCC), U2-17(CCC)
Flushing buffered data to a file R5-54(MCC)
fmt function R5-60(MCC)
Font files
  fonts.dir UA-3(XRAY)
fonts.dir file UA-3(XRAY)
FOPEN command R3-66(XRAY), R4-17(XRAY), U2-33(XRAY)
fopen function R5-61(MCC)
FOPEN_MAX R5-15(MCC)
FOPT assembler directive R5-26(ASM)
FOR . . . ENDF loop R7-7(ASM)–R7-8(ASM)
for statement R3-20(MCC)
FOR statement in macros R4-5(XRAY)
Formal parameter R6-2(ASM)
FORMAT assembler directive R5-27(ASM)
Format conversion functions R5-17(MCC)
  atof R5-31(MCC)
atoi R5-32(MCC)
atol R5-33(MCC)
mblen R5-113(MCC)
bstowcs R5-114(MCC)
bstowc R5-115(MCC)
strtol R5-179(MCC)
strtol R5-180(MCC)
westombs R5-200(MCC)
wctomb R5-201(MCC)
FORMAT linker command R10-31(ASM)
FORMAT linker command line option U3-13(ASM)
Format of XRAY files R3-90(XRAY)
Format, absolute file U2-3(XRAY)
Format, distribution
  DOS I1-1(MCC), I1-1(XRAY), I1-1(CCC)
  UNIX System V/386 I3-1(MCC)
  VMS I2-1(MCC)
Formats
  data record (S1) RF-2(ASM)
  header record (S0) RF-2(ASM)
  Motorola S-record RF-1(ASM)–RF-7(ASM)
  record count record (S5) RF-4(ASM)
  symbol record RF-1(ASM)
Formatted I/O functions R5-14(MCC)
  fprintf R5-63(MCC)
fscanf R5-71(MCC)
printf R5-130(MCC)
scanf R5-147(MCC)
sprintf R5-158(MCC)
sscanf R5-161(MCC)
vfprintf R5-197(MCC)
vprintf R5-198(MCC)
vstrupr R5-199(MCC)
Formatted output
  FPRINTF command R3-67(XRAY)
  PRINTF command R3-118(XRAY)
Formatted output to a file R5-63(MCC), R5-197(MCC)
Formatted output to standard error R5-46(MCC)
Formatted output to standard output file R5-130(MCC), R5-198(MCC)
Formated read from standard input R5-147(MCC)
Formated string conversion R5-161(MCC)
Formatting a string R5-158(MCC), R5-199(MCC)
Formatting information for numeric quantities R5-100(MCC)
@fpu pseudo-register RA-1(XRAY), RF-2(XRAY)
fpos_t type
  fgetpos function R5-56(MCC)
  fsetpos function R5-73(MCC)
PRINTF command R3-67(XRAY), U3-32(XRAY), U4-16(XRAY)
printf function R5-63(MCC)
  relationship to vfprintf R5-197(MCC)
_FPU preprocessor symbol U2-20(MCC), U2-18(CCC)
@fpu pseudo-register RA-1(XRAY), RF-2(XRAY)
fputc function R5-64(MCC)
relationship to putchar R5-135(MCC)
relationship to putc R5-134(MCC)
fputs function R5-65(MCC)
Fractional and integer parts of a number, determining R5-124(MCC)
Frame (definition) R3-171(XRAY), RD-1(XRAY)
Frame pointer R7-5(MCC)
/frames option (VMS) R7-6(MCC), R1016(MCC)
fread function R5-66(MCC)
free function R5-20(MCC), R5-67(MCC), R9-12(MCC)
freopen function R5-68(MCC)
frexp function R5-70(MCC)
Friend
    class R2-4(CCC)
    declaring R4-2(CCC)
    functions
    calling from C R5-4(CCC)
        declaring R4-2(CCC)
    member function R2-4(CCC)
friend class R2-3(CCC), RC-2(CCC), UA-2(CCC)
friend function R2-3(CCC), RC-2(CCC), UA-2(CCC)
friend keyword R4-2(CCC)
Friendship RA-16(CCC)
frl assembler command line flag U2-7(ASM), U3-7(ASM)
frs assembler command line flag U2-7(ASM), U3-7(ASM)
FRZ command S2-86(SUP)
fscanf function R5-71(MCC)
fseek function R5-72(MCC)
fsetpos function R5-73(MCC)
-Fsm option U2-20(MCC), U2-17(CCC)
ftell function R5-74(MCC), U2-52(MCC), U3-44(MCC), U2-44(CCC)
    relationship to fgetpos R5-56(MCC)
FTO command S2-87(SUP)
ftoa function R5-75(MCC)

FULLDIR librarian command U2-21(ASM), U3-19(ASM), R13-12(ASM)—R13-13(ASM)
FULLDIR librarian command option U3-19(ASM)
Fully qualified names, generating for input files
    /debug=fullfilename option U3-19(MCC)
    -Gf option U2-21(MCC), U2-18(CCC)
Function declaration R4-12(CCC)
Function in-lining R1015(MCC)
Function key commands
HELP R3-12(XRAY), R3-72(XRAY)
MODE R3-12(XRAY), R3-92(XRAY)
PREV CMD R3-12(XRAY)
STEP R3-12(XRAY), R3-193(XRAY)
STEP OVER R3-12(XRAY), R3-195(XRAY)
VACTIVE+1 R3-12(XRAY), R3-203(XRAY)
VACTIVE-1 R3-12(XRAY), R3-203(XRAY)
VSCREEN R3-12(XRAY), R3-210(XRAY)
ZOOM R3-12(XRAY), R3-213(XRAY)
Function names, truncating
    /truncate_identifiers U3-12(MCC)
    -ut option U2-8(MCC)
Function signature RB-6(CCC), RC-2(CCC), UA-2(CCC)
Function, friend RC-2(CCC), UA-2(CCC)
Functions R3-24(MCC), R4-12(CCC)
calling at an absolute address R9-10(MCC)
declaration R4-12(CCC)
duplicate names R5-1(CCC)
epilogue R7-8(MCC), R4-2(CCC)
friend R2-3(CCC), R2-4(CCC)
include libraries R3-29(MCC)
library
    (See Library functions)
member R2-4(CCC)
non-reentrant R5-20(MCC)
operator R2-25(CCC)
performing Flexible Licensing U3-
1(FLEX)
prologue R7-7(MCC), R4-2(CCC)
  local variables in prologue R7-7(MCC)
prototype arguments R3-25(MCC)
prototype declaration R4-13(CCC)
return values
  floating-point R7-4(MCC)
  integer R7-4(MCC)
  structure R7-4(MCC)
  union R7-4(MCC)
returning from
  /optimize=singleret option U3-27(MCC)
  -Oe option U2-31(MCC), U2-28(CCC)
signature R5-1(CCC)
system R9-31(MCC)
tagging entry and exit points
  /trace option U3-33(MCC)
  -Kt option U2-26(MCC), U2-23(CCC)
truncating names
  /truncate_identifiers option U3-12(MCC)
  -ut option U2-37(MCC), U2-34(CCC)
undeclared R4-13(CCC)
virtual R3-12(CCC)
pure R3-14(CCC)
Functions, target R2-14(XRAY)
fwrite function R5-76(MCC)
div R5-45(MCC)
exit R5-47(MCC)
free R5-67(MCC)
labs R5-97(MCC)
ldiv R5-99(MCC)
malloc R5-111(MCC)
qsort R5-139(MCC)
rand R5-140(MCC)
realloc R5-143(MCC)
srand R5-160(MCC)

Generating floating-point processor instructions
  /fpu option U3-23(MCC)
  -f option U2-20(MCC), U2-18(CCC)
get_license I2-3(FLEX), U2-3(FLEX)
  connecting to mllicense.daemon U2-3(FLEX), UA-1(FLEX), UC-12(FLEX)
  continuous error messages UC-17(FLEX)
  license server daemons not running U1-2(FLEX)
  messages written to UA-1(FLEX)
  verbose mode toggle UA-1(FLEX)
getc function R5-12(MCC), R5-77(MCC)
  relationship to fgetc R5-55(MCC)
getchar function R5-12(MCC), R5-78(MCC)
get function R5-79(MCC)
GETLICENSE_CONNECT_INTERVAL U2-3(FLEX), UA-1(FLEX)
GETLICENSE_CONNECT_RETRIES U2-3(FLEX), UA-1(FLEX)
GETLICENSE_OUTPUT U2-5(FLEX), U3-3(FLEX), UA-1(FLEX)
GETLICENSE_VERBOSE UA-1(FLEX)
getsw function R5-80(MCC)
Getting licenses
  (see Checking out licenses)
getw function R5-81(MCC)
  -Gf option U2-21(MCC), U2-18(CCC)
  -Gl option U2-21(MCC), U2-18(CCC)
  -Gl option (UNIX/DOS) R11-3(MCC)
Global
  constants R4-9(CCC)
  redefinition R4-8(CCC)
Global constant propagation
  optimization R104(MCC), R5-3(XRAY)
Global copy propagation R105(MCC)
Global copy propagation optimization R5-3(XRAY)
Global declarations R3-4(MCC), R4-8(CCC)
Global optimizations R102(MCC) – R109(MCC), R5-2(XRAY)
Global variable propagation R105(MCC)
Global variable optimization
  /optimize=stablemem option U3-27(MCC)
  -Ob option U2-30(MCC), U2-27(CCC)
Global variables, referencing R2-15(XRAY)
Global-flow optimizer
  /optimize=globalflow option U3-26(MCC)
  -Og option U2-31(MCC), U2-28(CCC)
Glossary UB-1(FLEX), RE-1(ASM) – RE-2(ASM), RD-1(XRAY), RC-1(CCC), UA-1(CCC)
  -Gm option U2-21(MCC)
gmtime function R5-21(MCC), R5-82(MCC)
Go button
  X Window support UA-12(XRAY)
go button
  SunView support UH-12(XRAY)
GO command R2-24(XRAY), R3-69(XRAY), U4-4(XRAY), U4-6(XRAY)
  assembly-level mode U2-22(XRAY), U2-24(XRAY)
  breakpoint setting U2-28(XRAY)
  high-level mode U2-18(XRAY)
  X Window support UA-12(XRAY)
Go To button
  X Window support UA-12(XRAY)
go to button
  SunView support UH-12(XRAY)
GOSTEP command R3-71(XRAY), U4-6(XRAY)
goto statement R3-21(MCC)
  -Gr option U2-21(MCC), U2-18(CCC)
Granting access to users I3-6(FLEX)
GROUP S2-88(SUP)

Grouping stack adjust instructions
  optimization R1017(MCC)
Groups S4-22(SUP)
  -Gs option U2-22(MCC), U2-19(CCC)
Guide, description IP-1(CCC)

H
  -H option U2-22(MCC), U2-19(CCC)
Halt XRAY session
  QUIT command R3-138(XRAY)
  temporarily
  PAUSE command R3-114(XRAY)
Hardware information display S2-89(SUP)
Header files
  defining R5-15(CCC)
Header record (S0) RF-2(ASM)
Help U4-1(XRAY)
Help button U2-51(CCC)
  X Window support UA-12(XRAY)
help button
  SunView support UH-12(XRAY)
HELP command R3-72(XRAY), U2-14(XRAY), U4-1(XRAY)
help command U2-53(CCC)
HELP command key
  Apollo support UB-4(XRAY)
  Apollo trace support UB-6(XRAY)
  DECstation support UC-4(XRAY)
  DECstation trace support UC-6(XRAY)
  HP support UD-6(XRAY)
  HP trace support UD-8(XRAY)
  IBM RS/6000 support UE-3(XRAY)
  IBM RS/6000 trace support UE-5(XRAY)
  Motorola Delta Series support UF-4(XRAY)
  Motorola Delta Series trace support UF-5(XRAY)
  PC support UG-4(XRAY)
  PC trace support UG-5(XRAY)
  Sun support UH-4(XRAY)
  Sun trace support UH-5(XRAY)
  VT terminal support UI-3(XRAY)
  VT terminal trace support UI-5(XRAY)
Help facility U1-4(XRAY)
Help file U4-11(XRAY)
location U4-11(XRAY)
HELP librarian command R13-14(ASM)
Help menu
  detailed help topic U3-16(XRAY)
  HELP command R3-72(XRAY), U2-14(XRAY)
  HELP command key U2-14(XRAY)
Help messages, printing U2-33(XRAY)
Help viewport U3-15(XRAY)
  HELP command U1-4(XRAY), U3-15(XRAY)
  HELP command key U1-4(XRAY), U3-15(XRAY)
Hexadecimal digit, testing for R5-94(MCC)
Hexadecimal interpretation
  RADIX option R3-107(XRAY)
Hexadecimal representation RA-1(MCC), RB-1(XRAY)
Highland Software I1-1(FLEX), IC-5(FLEX)
High-level and assembly code
  /show=source option U3-29(MCC)
high-level and assembly code
  -Fsm option U2-20(MCC), U2-17(CCC)
high-level command key
  Apollo trace support UB-5(XRAY)
  DECstation trace support UC-5(XRAY)
  HP trace support UD-8(XRAY)
  IBM RS/6000 trace support UE-5(XRAY)
  Motorola Delta Series trace support UF-6(XRAY)
  PC trace support UG-6(XRAY)
  Sun trace support UH-6(XRAY)
  VT terminal trace support UI-5(XRAY)
High-level mode
  MODE command R3-92(XRAY)

High-level mode debugging U1-3(XRAY), U1-4(XRAY), U2-12(XRAY)
  BREAKINSTRUCTION command U2-18(XRAY)
  C expressions and statements U1-3(XRAY)
  GO command U2-18(XRAY)
  MONITOR command U2-18(XRAY)
  PRINTVALUE command U2-18(XRAY)
screen U2-14(XRAY)
  single-stepping U2-17(XRAY)
  STEP command U1-4(XRAY), U2-16(XRAY)
  STEPOVER command U1-4(XRAY), U2-19(XRAY)
tutorial U2-12(XRAY)
High-level screen S1-5(SUP), U3-1(XRAY)
HISTORY command R3-74(XRAY)
  @hlpc pseudo-register R2-21(XRAY), RA-1(XRAY), RF-3(XRAY)
Holding licenses U1-3(FLEX)
  (see also Checking out licenses)
  determining which are held U1-4(FLEX), U3-2(FLEX)
  expiration time
  (see Expiration of licenses)
HOST S1-7(SUP)
HOST command R3-75(XRAY)
Host environment
  HOST command R3-75(XRAY)
Host ID
  alternatives to lmhostid U3-8(FLEX)
  displaying U3-4(FLEX)
hostid command I3-2(FLEX), I3-3(FLEX), U3-4(FLEX)
hostname command I3-2(FLEX)
Hosts
  Apollo support R3-105(XRAY)
  different file systems U2-11(FLEX)
  DOS support R3-105(XRAY)
  specifying U2-1(FLEX)
  valid U2-11(FLEX)
Hosts, valid IP-1(FLEX), I1-1(FLEX)
HP
host-specific information UD-1(XRAY)

HP 9000 Series
(see HP support)

HP support
command keys UD-6(XRAY)
MOVE TO BOTTOM control key UD-5(XRAY)
MOVE TO TOP control key UB-2(XRAY), UD-5(XRAY)
TERM environment variable UD-2(XRAY)
X Window support UD-2(XRAY)

_HP9000_300 preprocessor symbol R4-3(MCC)

_HP9000_700 preprocessor symbol R4-3(MCC)

HP-UX
(see HP support)

HP-UX installation
terminal support
configuration parameters UD-3(XRAY)

HUGE_VAL macro R5-6(MCC), R5-7(MCC)

HWCONFIG S2-89(SUP)

HWCONFIG command S2-89(SUP)

Hyperbolic cosine of a number, computing R5-41(MCC)

Hyperbolic sine of a number, computing R5-157(MCC)

Hyperbolic tangent of a number, computing R5-184(MCC)

I/O buffering
buffered R6-3(CCC)
embedded environments R6-3(CCC)
native environments R6-3(CCC)
synchronized I/O with stdio buffering R6-4(CCC)

unbuffered R6-3(CCC)
unit buffered R6-4(CCC)

I/O device registers U2-31(MCC), U3-28(MCC), U2-28(CCC)

I/O file
association with buffer R5-150(MCC)

I/O functions R5-12(MCC)
clearerr R5-37(MCC)
fclose R5-51(MCC)
feof R5-52(MCC)
fflush R5-54(MCC)
ffget R5-55(MCC)
ffgetpos R5-56(MCC)
ffgets R5-57(MCC)
ffopen R5-61(MCC)

formatted R5-14(MCC)

fputc R5-64(MCC)

fputs R5-65(MCC)

fre R5-66(MCC)

fread R5-68(MCC)

fseek R5-72(MCC)

fsetpos R5-73(MCC)

ftell R5-74(MCC)

fwrite R5-76(MCC)

getc R5-77(MCC)

getchar R5-78(MCC)

gets R5-80(MCC)

defom R5-128(MCC)

dputc R5-134(MCC)

putchar R5-135(MCC)

puts R5-137(MCC)

remove R5-144(MCC)

rewind R5-145(MCC)

setbuf R5-150(MCC)

setvbuf R5-154(MCC)

ungetc R5-191(MCC)

I

i assembler command line flag U2-7(ASM), U3-7(ASM)

-I assembler command line option U2-3(ASM)

-I option U2-22(MCC), U2-19(CCC)
+I option U2-20(CCC)

I/O
simulated S7-1(SUP)
I/O screen  U2-25(XRAY), U3-4(XRAY)
I/O static initialization and termination  R7-6(CCC)
I/O, simulated  U4-9(XRAY)
IBM RS/6000
  host-specific information  UE-1(XRAY)
IBM RS/6000 support
  MOVE TO BOTTOM control key  UE-2(XRAY)
  MOVE TO TOP control key  UE-2(XRAY)
IBM-PC
  (see DOS installation)
  (see DOS)
  (see PC)
IBM-PC/DOS
  (see DOS)
 Ice
  not supported  S1-7(SUP)
ICE command  R3-77(XRAY)
Identifiers  R2-1(MCC)
  (see Symbols)
IDNT assembler directive  R5-28(ASM)
IEE2AOUT conversion utility  U2-24(ASM)–U2-28(ASM), U3-21(ASM)–U3-26(ASM)
IEEE floating-point format  R6-4(MCC)
IEEE-695 format  U2-2(XRAY), U2-3(XRAY)
IET  R3-124(XRAY)
IF . . . THEN . . . ELSE . . . ENDI assembler directive  R7-9(ASM)–R7-10(ASM)
#if directive  R4-7(MCC), R4-17(MCC)
if statement  R3-21(MCC)
IF statement in macros  R4-6(XRAY)
IFC assembler directive  R5-29(ASM)–R5-30(ASM)
IFDEF assembler directive  R5-31(ASM)
#ifndef directive  R4-7(MCC), R4-19(MCC)
if-else statement  R3-22(MCC)
IF-ELSE statement in macros  R4-7(XRAY)
IFEQ assembler directive  R5-32(ASM)–R5-33(ASM)
IFGE assembler directive  R5-34(ASM)
IFGT assembler directive  R5-35(ASM)
IFLE assembler directive  R5-36(ASM)
IFLT assembler directive  R5-37(ASM)
IFNC assembler directive  R5-38(ASM)–R5-39(ASM)
IFNDEF assembler directive  R5-40(ASM)
#define directive  R4-7(MCC), R4-20(MCC)
IFNE assembler directive  R5-41(ASM)
allegal access space  S3-32(SUP)
Implicit operand  R2-23(CCC)
In-circuit emulator commands  R3-12(XRAY)
  BREAKCOMPLEX  R3-12(XRAY), R3-28(XRAY)
  ICE  R3-12(XRAY), R3-77(XRAY)
  NOICE  R3-12(XRAY), R3-98(XRAY)
In-circuit emulator communication
  ICE command  R3-77(XRAY)
  NOICE command  R3-98(XRAY)
#include  R3-6(MCC)
INCLUDE assembler directive  R5-42(ASM)
INCLUDE command  R3-79(XRAY), U2-26(XRAY), U3-17(XRAY), U4-8(XRAY)
  defining macros  R4-9(XRAY)
#include directive  R4-6(MCC), R4-21(MCC), R5-1(MCC)
Include file  U2-10(XRAY), U2-26(XRAY), U3-17(XRAY), U4-8(XRAY)
  for performance analysis  S4-3(SUP)
Include file commands, displaying
  INCECHO option  R3-106(XRAY)
Include files
  (see #include files)
  comments  R3-4(XRAY)
  error handling
    ERROR command  R3-59(XRAY)
    INCLUDE command  R3-79(XRAY)
  no echo
    INCECHO option  U4-12(XRAY)
UNIX
  alternate locations  U2-4(MCC)
VMS
  alternate locations  U3-3(MCC)
#include files  R5-1(MCC)
  assert.h  R5-2(MCC)
  cdecl.h  R5-3(MCC)
  errno.h  R5-4(MCC)
expanding in assembler source file
  -Fsi option U2-20(MCC)
  /show=include option U3-28(MCC)
float.h R5-4(MCC)
limits.h R5-4(MCC)
locale.h R5-4(MCC)
math.h R5-5(MCC)
mathf.h R5-6(MCC)
mriext.h R5-7(MCC), R5-8(MCC)
search path, specifying
  -I option U2-22(MCC), U2-19(CCC),
  U2-22(MCC), U2-19(CCC)
  /ipath option U3-22(MCC), U2-19(CCC)
  /J option U2-23(MCC), U2-20(CCC),
  U2-23(MCC), U2-20(CCC)
  /spath option U3-32(MCC)
setjmp.h R5-9(MCC)
signal.h R5-10(MCC)
stdarg.h R5-10(MCC)
stddef.h R5-11(MCC)
stdio.h R5-2(MCC), R5-12(MCC),
R5-14(MCC), R5-20(MCC), R5-150(MCC), R5-193(MCC)
stdlib.h R5-2(MCC), R5-16(MCC),
R5-17(MCC)
string.h R5-17(MCC)
time.h R5-2(MCC), R5-19(MCC)

INCLUDE line l3-6(FLEX)
INCLUDE linker command R10-32(ASM)
Inclusive Execution Time R3-124(XRAY)
Incremental linking R9-1(ASM), R9-8(ASM)–
R9-9(ASM)
INDEX linker command R3-25(ASM), R3-27(ASM), R10-33(ASM)–R10-34(ASM)
Index simplification
  optimization R109(MCC), R5-11(XRAY)
Indexing array optimization R1018(MCC)
Indexing arrays optimization R5-18(XRAY)
Indirect addressing R2-9(XRAY)
Induction variable elimination R109(MCC)
#if directive R4-22(MCC)
  relation to #pragma info R4-26(MCC)

Informational messages
  (see also Messages)
suppression
    /suppress=informational option U3-30(MCC)
    -Qi option U2-36(MCC), U2-33(CCC)
Inheritance R3-1(CCC)–R3-18(CCC), RC-2(CCC), UA-2(CCC)
BROWSE command (C++) RG-12(XRAY)
  concepts
  access protection R3-1(CCC)
  derivation R3-1(CCC)
  fundamental elements R3-1(CCC)
  multiple R3-10(CCC), RC-2(CCC), UA-2(CCC)
  single R3-10(CCC), RC-2(CCC), UA-2(CCC)
INITDATA linker command R9-40(MCC),
R10-35(ASM)
INITDATA section R10-35(ASM)
initfini section R7-2(ASM), U2-13(CCC)
Initial setup
  user U1-1(FLEX)
Initialization R3-17(MCC), R9-39(MCC)
  compile-time R9-39(MCC)
  data R9-39(MCC)
  default R6-42(MCC)
  run-time R9-39(MCC)
  static constructor R7-1(CCC)
  static destructor R7-1(CCC)
  static objects R7-1(CCC)
  static variables R6-42(MCC)
Initialize local variables
  /init_locals option U3-22(MCC)
  -KI option U2-25(MCC), U2-22(CCC)
Initialized data section U2-16(MCC), U3-15(MCC), U2-13(CCC)
  naming
    /rename option U3-31(MCC)
    -NI option U2-29(MCC), U2-25(CCC)
  specifying address mode
    /initvars_addr_as options U3-16(MCC)
initvars section
  naming
    /rename option  U3-31(MCC)
In-line assembly  R9-2(MCC), R9-3(MCC)
In-line assembly code
  changing insert character
    /insert_char option  U3-12(MCC)
    -ui option  U2-37(MCC), U2-34(CCC)
  enabling
    /mri_extensions option  U3-24(MCC)
    -x option  U2-39(MCC), U2-36(CCC)
In-line expansion
  +d option  U2-15(CCC)
In-line function  R2-5(CCC)
  specification  R4-2(CCC)
inline keyword  R4-2(CCC)
In-line library function
  expansion  R1017(MCC)
In-line member function  R2-6(CCC)
In-lining
  /optimize=inline option  U3-26(MCC)
  library function calls, disabling
    -nx option  U2-40(MCC), U2-37(CCC)
    -Oi option  U2-31(MCC), U2-28(CCC)
In-lining assembly instructions  R2-26(CCC)
INPORT  S1-7(SUP), S7-1(SUP)
INPORT command  R3-80(XRAY), R4-18(XRAY), U2-11(XRAY), U4-9(XRAY)
size qualifiers  R3-3(XRAY)
import macro  R4-18(XRAY), U4-9(XRAY)
Import ports
  reading value from  R4-18(XRAY)
Input and output files, location  U2-3(CCC)
DOS  U2-3(MCC)
UNIX  U2-3(MCC)
VMS  U3-3(MCC)
Input files
  extensions  U2-2(MCC), U3-2(MCC)
generating fully qualified names
    /debug=fullfilename option  U3-19(MCC)
-Gf option  U2-21(MCC), U2-18(CCC)
locations  U2-3(MCC), U3-3(MCC)
Input files, rewinding
  RIN command  R3-146(XRAY)
Input for install.sh  I-5(FLEX)
Input port
  address
    INPORT command  R3-80(XRAY)
    buffer
      DIN command  R3-50(XRAY)
Input/Output functions
  (see I/O functions)
Ins key (IBM RS/6000 support)  UE-2(XRAY)
Ins key (Motorola Delta Series support)  UF-2(XRAY)
Ins key (PC support)  UG-3(XRAY)
Ins key (Sun support)  UH-3(XRAY)
Ins Line key (HP support)  UD-5(XRAY)
Insert Char key (HP support)  UD-5(XRAY)
Insert character, changing
    /insert_char option  U3-12(MCC)
    -ui option  U2-37(MCC), U2-34(CCC)
Inspection tool  U2-46(CCC)
Install directory
  (see install_dir, /usr/mri)
INSTALL program
DOS  I-2(MCC), I-2(XRAY), I-3(CCC)
install.sh  U2-9(FLEX)
  aborting  I-5(FLEX)
  architectures mixed  IA-1(FLEX)
  defaults, specifying  I-5(FLEX)
  device to use for installation  I-18(FLEX)
  drive to use for installation  I-19(FLEX)
  size qualifiers  R3-3(XRAY)
  error handling  I-5(FLEX)
  executing script  I-8(FLEX)
  features of  I-5(FLEX)
  features, specifying  I-12(FLEX), I-14(FLEX)
  mixed architectures  IA-1(FLEX)
  part A  I-9(FLEX)
  description  I-5(FLEX)
skipping Il-9(FLEX)
part B Il-16(FLEX)
description Il-5(FLEX)
going directly to Il-9(FLEX)
skipping Il-16(FLEX)
servers, specifying Il-11(FLEX), Il-12(FLEX)
exiting loop Il-11(FLEX)
skipping a part Il-5(FLEX)

install_dir Il-10(FLEX)
(see also /usr/mri)
/usr/mri/sun3 IA-1(FLEX)
/usr/mri/sun4 IA-1(FLEX)
default
(see /usr/mri)
nonstandard Il-10(FLEX), I2-2(FLEX)
install_dir order subdirectory Il-9(FLEX), Il-10(FLEX)
install_dir/bin, adding to path I2-1(FLEX)
install_dir/bin/MRI I3-2(FLEX)

Installation
DOS I1-1(XRAY)
time stamp module S6-4(SUP)
VMS I2-1(XRAY)

Installation diskettes
DOS I1-1(CCC)

Installation procedure
DOS I1-2(MCC), I1-2(CCC)
UNIX System V/386 I3-1(MCC)
VMS I2-1(MCC)

Installation script U2-9(FLEX)
(see install.sh)

Installing Flexible License Manager
(see also install.sh)
Apollo
(see Apollo installation)
architecture mixed IA-1(FLEX), IA-2(FLEX)
defaults, use of Il-5(FLEX)
DEMO licenses
(see DEMO licenses)
directory other than /usr/mri Il-10(FLEX), I2-2(FLEX)

license file with nonstandard name Il-15(FLEX), I2-1(FLEX), I2-7(FLEX)
license type, specifying Il-14(FLEX)
mixed architectures IA-1(FLEX), IA-2(FLEX)
node-locked licenses
(see Node-locked licenses)
nonstandard location Il-10(FLEX)
preinstallation steps Il-2(FLEX)
remote installation Il-8(FLEX)
skipping a part Il-5(FLEX)
steps for installing Il-6(FLEX)
who Il-1(FLEX)

Installing toolkit
(see Toolkit)

Instruction alignment RF-4(XRAY)

Instruction breakpoint
setting
(see BREAKINSTRUCTION command)

Instruction breakpoints S3-34(SUP)
Instruction operands R3-3(ASM)

Instruction scheduling
/optimize=reorder option U3-27(MCC)
-Or option U2-32(MCC), U2-29(CCC)

Instruction statement R2-2(ASM)
Instruction types, variants R3-2(ASM)
int type R3-2(MCC), R6-3(MCC), R6-12(MCC)

Integer
computing absolute value R5-23(MCC)
conversion from ASCII string R5-32(MCC)
conversion to ASCII string R5-95(MCC)

Integer constants R2-11(ASM)–R2-12(ASM)

Integer return values R7-4(MCC)
Intermediate C file saved
-Fc option U2-17(CCC)

Intermediate C++ file, saved
+i option U2-20(CCC)

Internal compiler errors RC-15(MCC)

Internal DMA S2-141(SUP)
Interrupt
keyword R4-2(CCC)
-Kr command line option R4-3(CCC)
not supported S1-7(SUP)
INTERRUPT command R3-84(XRAY)
Interrupt handlers, declaring R7-11(MCC), R9-45(MCC)
interrupt keyword R7-11(MCC), R9-45(MCC), R4-2(CCC)
interrupt keyword, disabling
-nx option U2-40(MCC), U2-37(CCC)
Interrupt latency S6-22(SUP)
Interrupt procedures
/ireturn=exception option U3-22(MCC)
/ireturn=subroutine option U3-22(MCC)
-Kr option U2-26(MCC), U2-23(CCC)
return with RTE instruction
/ireturn=exception option U3-22(MCC)
-nKr option U2-26(MCC), U2-23(CCC)
return with RTS instruction
/ireturn=subroutine option U3-22(MCC)
-Kr option U2-26(MCC), U2-23(CCC)
Interrupts S2-129(SUP), S2-132(SUP), S2-136(SUP)
cancel
NOINTERRUPT command R3-99(XRAY)
simulation
INTERRUPT command R3-84(XRAY)
Introduction R1-1(MCC), R1-1(CCC)
assembler R1-1(ASM), R2-1(ASM)
C++ compiler package U1-1(CCC)
librarian R12-1(ASM)
linker R9-1(ASM)
Introduction to compiler package U1-1(MCC)
Invocation S1-2(SUP)
assembler U3-45(MCC)
compiler U2-1(MCC), U3-1(MCC), U2-1(CCC)
linker U2-55(MCC), U3-45(MCC), U2-46(CCC)
Invocation examples
UNIX/DOS
assembler U2-12(ASM)
librarian U2-23(ASM)–U2-24(ASM)
command file U2-23(ASM)–U2-24(ASM)
command line U2-23(ASM)
interactive U2-23(ASM)
linker U2-17(ASM)–U2-20(ASM)
command file U2-17(ASM)
without command file U2-19(ASM)–U2-20(ASM)
VAX/VMS
assembler U3-12(ASM)
command line U3-12(ASM)
multiple flags U3-12(ASM)
librarian U3-20(ASM)–U3-21(ASM)
command file U3-21(ASM)
command line U3-20(ASM)
linker U3-16(ASM)–U3-17(ASM)
command file U3-16(ASM)
command line U3-17(ASM)
VMS
linker
command file U3-16(ASM)
Invocation methods, librarian
command file R12-3(ASM)
command line R12-3(ASM)
interactive R12-3(ASM)
Invocation syntax
assembler
UNIX/DOS U2-2(ASM)
VAX/VMS U3-2(ASM)
librarian
UNIX/DOS U2-21(ASM)
VAX/VMS U3-18(ASM)
linker
UNIX/DOS U2-13(ASM)
VAX/VMS U3-13(ASM)
Invoking XRAY U2-9(XRAY)
.inc file U2-10(XRAY)
.jou file U2-10(XRAY)
.log file  U2-10(XRAY)
DOS  U2-9(XRAY)
include file  U2-10(XRAY)
journal file  U2-10(XRAY)
log file  U2-10(XRAY)
startup.xry file  U2-10(XRAY)
UNIX  U2-9(XRAY)
VMS  U2-9(XRAY)
ios68ka5.lib  U2-42(CCC)
ios68ka5020.lib  U2-42(CCC)
ios68kab.lib  U2-42(CCC)
ios68kab020.lib  U2-42(CCC)
ios68kpc.lib  U2-42(CCC)
ios68kpc020.lib  U2-42(CCC)
IPATH assembler command line option  U3-3(ASM)
IRET instruction, using for interrupt procedures
    /ireturn=exception  U3-22(MCC)
    /ireturn option (VMS)  R9-45(MCC)
    /ireturn=subroutine option (VMS)  R7-11(MCC)
IRP assembler directive  R5-43(ASM)–R5-44(ASM)
IRPC assembler directive  R5-45(ASM)–R5-46(ASM)
isalive macro  R4-20(XRAY)
isalnum function  R5-83(MCC)
isalpha function  R5-84(MCC)
isascii function  R5-85(MCC)
iscntrl function  R5-86(MCC)
isdigit function  R5-87(MCC)
isgraph function  R5-88(MCC)
islower function  R5-89(MCC)
isprint function  R5-90(MCC)
ispunct function  R5-91(MCC)
isspace function  R5-92(MCC)
isupper function  R5-93(MCC)
isxdigit function  R5-94(MCC)
itoa function  R5-95(MCC)
itostr function  R5-96(MCC)

J
-J option  U2-23(MCC), U2-20(CCC)
jmp_buf type  R5-9(MCC)
JOURNAL command  R3-86(XRAY), U4-7(XRAY), U4-8(XRAY)
Journal file  U2-10(XRAY), U3-17(XRAY), U4-7(XRAY), U4-8(XRAY)
Journal viewport  U3-17(XRAY)
    JOURNAL command  U3-17(XRAY)
Jump optimizations  R1011(MCC)–R1015(MCC), R5-12(XRAY)
Jumping stack levels in execution  U4-4(XRAY)
Jumps outside of a function
    Function, jumps outside of a  R5-107(MCC)

K
-K2 option  U2-24(MCC), U2-21(CCC)
-K4 option  U2-24(MCC), U2-21(CCC)
Kanji characters  R5-16(MCC)
-Kc option  U2-24(MCC), U2-21(CCC)
key_get macro  R4-21(XRAY)
key_stat macro  R4-22(XRAY)
Keywords  R2-6(MCC)
    C++  R4-1(CCC)
    -Kf option  U2-25(MCC), U2-22(CCC)
    -Kf option (UNIX/DOS)  R7-6(MCC), R1016(MCC)
    -Kh option  U2-25(MCC), U2-22(CCC)
    -Kh option (UNIX/DOS)  R7-5(MCC), R9-34(MCC)
    -Kl option  U2-25(MCC), U2-22(CCC)
kill command  l2-8(FLEX)
KP , key (VT terminal support)  UI-2(XRAY)
KP + key (VT terminal support)  UI-2(XRAY)
KP0 key
    DECstation trace support  UC-5(XRAY)
    VT terminal support  UI-3(XRAY)
    VT terminal trace support  UI-5(XRAY)
KP1 key
    DECstation trace support  UC-5(XRAY)
    VT terminal support  UI-3(XRAY)
VT terminal trace support  UI-5(XRAY)

KP2 key
DECstation trace support  UC-5(XRAY)
VT terminal support  UI-3(XRAY)
VT terminal trace support  UI-5(XRAY)

KP3 key
DECstation trace support  UC-5(XRAY)
VT terminal support  UI-3(XRAY)
VT terminal trace support  UI-5(XRAY)

KP4 key
DECstation trace support  UC-5(XRAY)
VT terminal support  UI-3(XRAY)
VT terminal trace support  UI-5(XRAY)

KP5 key
DECstation trace support  UC-6(XRAY)
VT terminal support  UI-3(XRAY)
VT terminal trace support  UI-5(XRAY)

KP6 key
DECstation trace support  UC-6(XRAY)
VT terminal support  UI-4(XRAY)
VT terminal trace support  UI-5(XRAY)

KP7 key
DECstation trace support  UC-6(XRAY)
VT terminal support  UI-4(XRAY)
VT terminal trace support  UI-5(XRAY)

KP8 key
DECstation trace support  UC-6(XRAY)
VT terminal support  UI-4(XRAY)
VT terminal trace support  UI-5(XRAY)

KP9 key
VT terminal support  UI-4(XRAY)
VT terminal trace support  UI-6(XRAY)

-Kr command line option  R4-3(CCC)
-Kr option  U2-26(MCC), U2-23(CCC)
-Kr option (UNIX/DOS)  R7-11(MCC), R9-45(MCC)

-KT option  U2-26(MCC)
-Kt option  U2-26(MCC), U2-23(CCC)
-KT option (UNIX/DOS)  R4-3(MCC)
-Ku option  U2-27(MCC), U2-23(CCC)
-Ku option (UNIX/DOS)  R4-3(MCC), R6-10(MCC)

L
-L assembler command line option  U2-3(ASM)
-I assembler command line option  U2-3(ASM)
-I librarian command line option  U2-21(ASM)
-I linker command line option  U2-14(ASM)
-I option  U2-27(MCC), U2-24(CCC)
L_tmpnam  R5-15(MCC)
Labeled statement  R3-21(MCC)
Labels for line numbers
   /debug=lines option  U3-19(MCC)
   -Gl option  U2-21(MCC), U2-18(CCC)
   labs function  R5-97(MCC)
Latency
   access breakpoints  S4-13(SUP)
   execution breakpoints  S4-13(SUP)
Iconv structure  R5-100(MCC)
Idxexp function  R5-98(MCC)
Idiv function  R5-99(MCC)
Left shift operation  R2-24(CCC)
Legal expressions  R2-12(XRAY)
   examples  R2-13(XRAY)
   expression strings  R2-13(XRAY)
Length of initial string segment  R5-167(MCC), R5-175(MCC)
Length of line  R3-5(XRAY)
Length of program identifiers  U3-12(MCC)
Length of string  R5-169(MCC)
Librarian
   command characters
      asterisk  R13-1(ASM)
      blanks  R13-2(ASM)
      comma  R13-1(ASM)
      parentheses  R13-1(ASM)
      plus  R13-2(ASM)
      semicolon  R13-1(ASM)
   command file comments  R13-2(ASM)
   commands  R13-1(ASM)
      ADDLIB  R13-4(ASM)
      ADDMOD  R13-5(ASM)
      CLEAR  R13-6(ASM)
      CREATE  R13-7(ASM)
      DELETE  R13-8(ASM)
DIRECTORY R13-9(ASM)
END R13-10(ASM)
EXTRACT R13-11(ASM)
FULLDIR R13-12(ASM)–R13-13(ASM)
HELP R13-14(ASM)
OPEN R13-15(ASM)
QUIT R13-10(ASM)
REPLACE R13-16(ASM)
SAVE R13-17(ASM)
demangled name RG-4(ASM)
description U1-2(ASM), U1-2(CCC)
error messages RD-1(ASM)–RD-6(ASM)
features R12-1(ASM)
function R12-1(ASM)–R12-6(ASM)
introduction R12-1(ASM)
invocation methods R12-3(ASM)
listing of commands R13-14(ASM)
listings, sample R14-1(ASM)–R14-6(ASM)
unmangled name RG-4(ASM)
message severity levels RD-1(ASM)
overview R1-1(ASM)
return codes R12-6(ASM)
special characters R13-1(ASM)
syntax R13-1(ASM)–R13-2(ASM)
UNIX/DOS U2-21(ASM)–U2-24(ASM)
  file name defaults U2-22(ASM)
  invocation examples U2-23(ASM)–U2-24(ASM)
  options U2-21(ASM)–U2-22(ASM)
    -a U2-21(ASM)
    -d U2-21(ASM)
    -e U2-21(ASM)
    -l U2-21(ASM)
    -r U2-22(ASM)
    -V U2-22(ASM)
use of special characters R13-1(ASM)
VAX/VMS U3-18(ASM)–U3-21(ASM)
  file name defaults U3-20(ASM)
  invocation examples U3-20(ASM)–U3-21(ASM)
  invocation syntax U3-18(ASM)
  options U3-19(ASM)
ADDMOD U3-19(ASM)
DELETE U3-19(ASM)
EXTRACT U3-19(ASM)
FULLDIR U3-19(ASM)
OPTION U3-19(ASM)
OUTPUT U3-19(ASM)
REPLACE U3-19(ASM)
VERSION U3-19(ASM)

Librarian description U1-2(MCC)
Libraries R9-40(MCC), U2-50(MCC), U2-39(CCC)
  C++ I/O levels R6-1(CCC)
  C++ library R6-1(CCC)
  conditions for use U2-43(CCC)
  termination routine (exit) U2-45(CCC)
  UNIX
    alternate locations U2-4(MCC)
    UNIX level-1 functions U2-43(CCC)
    UNIX level-1+ elements R6-2(CCC)
    use U2-41(CCC)
  VMS
    alternate locations U3-3(MCC)

Library functions
(see under specific names)
close R9-31(MCC)
  Iseek R9-31(MCC)
  mathematical functions R5-5(MCC)
  non-reentrant R5-20(MCC)
open R9-31(MCC)
read R9-31(MCC)
unimplemented R5-2(MCC)
write R9-31(MCC)

Library include files
(see #include files)
LIBRARY linker command line option U3-14(ASM)

Library macros R5-2(MCC), R5-3(MCC), R5-4(MCC), R5-5(MCC), R5-6(MCC), R5-7(MCC), R5-8(MCC), R5-10(MCC), R5-11(MCC), R5-14(MCC)–R5-15(MCC)

License Authorization Form I1-2(FLEX), I1-4(FLEX), I1-5(FLEX), I1-11(FLEX), U2-9(FLEX), UB-1(FLEX)
License daemons
(see License server daemons)

License file  I1-2(FLEX), U2-9(FLEX), U2-10(FLEX), UB-2(FLEX)
cannot find message  I2-6(FLEX)
creating
(see Creating license file)
DAEMON line  I3-2(FLEX), I3-5(FLEX)
editable items  I3-2(FLEX)
editing  I3-2(FLEX)
example  I3-1(FLEX), I3-4(FLEX), U2-9(FLEX)
FEATURE line  I1-14(FLEX), U2-4(FLEX), U2-9(FLEX)
error in  I2-4(FLEX), I2-8(FLEX)
syntax  I3-3(FLEX)
floating licenses  I3-4(FLEX)
format  I3-1(FLEX)
function  U2-1(FLEX)
hosts with different file systems  U2-11(FLEX)
mllicense.daemon accessing  U2-4(FLEX)
modifying  I1-9(FLEX), I1-10(FLEX)
multiple vendors  IC-5(FLEX)
name  I1-14(FLEX), I2-1(FLEX), I2-7(FLEX)
name nonstandard  U1-1(FLEX), UA-2(FLEX)
node-locked licenses  I3-4(FLEX)
passwords  I3-1(FLEX)
rereading  U3-6(FLEX)
SERVER line  U2-9(FLEX)
error in  I2-4(FLEX), I2-8(FLEX)
host name incorrect  I2-8(FLEX)
how many  U2-10(FLEX)
how many  I2-6(FLEX)
syntax  I3-2(FLEX)

License holder  UB-2(FLEX)
(see mllicense.daemon)

License manager  UB-2(FLEX)
(see lmgrd)

License manager daemon
(see lmgrd)

License server  U2-5(FLEX), U2-10(FLEX),
holding
  (see Holding licenses, Checking out licenses)
information, obtaining  U1-4(FLEX), U3-2(FLEX), U3-6(FLEX)
limited number of users
  (see Counted licenses)
ode-locked
  (see Node-locked licenses)
  obtaining information U1-4(FLEX), U3-2(FLEX), U3-6(FLEX)
partial hosts
  (see Node-locked licenses)
reserving  U2-11(FLEX)
reserving for specific users  I3-5(FLEX)
restarting daemons  U1-5(FLEX)
status  U1-4(FLEX), U3-2(FLEX), U3-6(FLEX)
types of  I1-1(FLEX), U2-1(FLEX), U2-2(FLEX)
  specifying during install  I1-14(FLEX)
unlimited users
  (see node-locked licenses)
  (see Node-locked licenses, DEMO licenses)
version number  I3-3(FLEX)
  with expiration date  U2-2(FLEX)
Limits macros  R5-4(MCC)
limits.h include file  R5-4(MCC)
Line
  reading from standard input  R5-80(MCC)
Line continuation character (%)  R3-5(XRAY), U2-9(XRAY)
Line Del key
  Apollo support  UB-3(XRAY)
#line directive  R4-6(MCC), R4-23(MCC)
Line length  R3-5(XRAY)
Line number control directive (#line)  R4-6(MCC)
Line number labels, producing
  /debug=lines option  U3-19(MCC)
  -Gl option  U2-21(MCC), U2-18(CCC)

Line number, variable, and symbol information
  /debug option  U3-18(MCC)
  -g option  U2-22(MCC), U2-19(CCC)
Line numbers  R2-10(XRAY)
Line numbers in assembly language source file  R11-2(MCC)
__LINE__ preprocessor symbol  R4-1(MCC), R4-23(MCC)
Line_number (definition)  R3-4(XRAY)
@line_range pseudo-register  RA-1(XRAY), RF-3(XRAY)
Linker
  absolute output file  R11-9(ASM)
  alias command  U2-30(MCC), U2-26(CCC)
  command example
    HP 64000 system, IEEE  R9-45(MCC)
    IEEE  R9-41(MCC)
    ROM-based system, IEEE  R9-43(MCC)
  command file  R11-4(ASM)
    (see Linker command file)
  command line continuation
    character  U3-21(ASM)
  command position dependencies  R10-3(ASM)
  commands
    ABSOLUTE  R10-7(ASM)–R10-8(ASM)
    ALIAS  R10-9(ASM)–R10-10(ASM)
    ALIGN  R10-11(ASM)–R10-12(ASM)
    ALIGNMOD  R10-13(ASM)
    BASE  R10-14(ASM)–R10-15(ASM)
    CASE  R10-16(ASM)–R10-17(ASM)
    CHIP  R10-18(ASM)–R10-20(ASM)
    Comment  R10-21(ASM)
    COMMON  R10-22(ASM)–R10-23(ASM)
    CPAGE  R10-24(ASM)–R10-25(ASM)
    DEBUG_SYMBOLS  R10-26(ASM)
    END  R10-27(ASM)
    ERROR  R10-28(ASM)
EXIT R10-29(ASM)
EXTERN R10-30(ASM)
FORMAT R10-31(ASM)
INCLUDE R10-32(ASM)
INDEX R3-25(ASM), R3-27(ASM), R10-33(ASM)-R10-34(ASM)
INITDATA R10-35(ASM)
LISTABS R10-37(ASM)
LISTMAP R10-38(ASM)-R10-39(ASM)
LOAD R10-40(ASM)-R10-41(ASM)
LOAD_SYMBOLS R10-42(ASM)
LOWERCASE R10-43(ASM)-R10-44(ASM)
MERGE R10-45(ASM)-R10-46(ASM)
NAME R10-47(ASM)
NODEBUG_SYMBOLS R10-26(ASM)
NOERROR R10-28(ASM)
NOPAGE R10-50(ASM)-R10-51(ASM)
ORDER R10-48(ASM)-R10-49(ASM)
PAGE R10-50(ASM)-R10-51(ASM)
PUBLIC R10-52(ASM)-R10-53(ASM)
RESADD R10-54(ASM)
RESMEM R10-55(ASM)
SECT R10-56(ASM)
SECTSIZE R10-57(ASM)
SORDER R10-58(ASM)-R10-59(ASM)
START R10-60(ASM)
SYMTRAN R10-61(ASM)-R10-62(ASM)
UPPERCASE R10-63(ASM)
WARN R10-64(ASM)
error messages RC-1(ASM)-RC-11(ASM)
features R9-2(ASM)
header record RF-2(ASM)
incremental linking R9-1(ASM), R9-8(ASM)-R9-9(ASM)
invocation U2-46(ASM)
invoking U2-55(ASM), U3-45(ASM)
listing, sample R11-1(ASM)-R11-18(ASM)
map file R11-5(ASM)-R11-8(ASM)
memory space assignment R9-6(ASM)
message severity levels RC-1(ASM)
module record RF-1(ASM)
operation R11-1(ASM)
overview R1-1(ASM), R9-1(ASM)
passing options directly
-WI option U2-38(ASM), U2-35(ASM)
public symbol table R11-3(ASM)
record count record RF-4(ASM)
relocation types R9-8(ASM)
sections R9-2(ASM)-??
  absolute R9-3(ASM)
  common R9-4(ASM)
  long R9-5(ASM)
  noncommon R9-4(ASM)
  relocatable R9-3(ASM)
  section alignment R9-5(ASM)
  short R9-4(ASM)
  type R9-5(ASM)-R9-6(ASM)
start address R11-3(ASM)
suppressing call to U2-18(ASM), U2-14(ASM)
symbol name size RG-3(ASM)
symbol record RF-1(ASM)
syntax R10-1(ASM)-R10-4(ASM)
UNIX/DOS U2-13(ASM)-U2-20(ASM)
file name defaults U2-16(ASM)
invocation examples U2-17(ASM)-U2-20(ASM)
options U2-13(ASM)-U2-15(ASM)
  -C U2-13(ASM)
  -c U2-13(ASM)
  -F U2-13(ASM)
unresolved externals R11-2(ASM)
use R9-39(MCC)

VAX/VMS U3-12(ASM)--??
file name defaults U3-16(ASM)
invocation examples U3-16(ASM)--
U3-17(ASM)
invocation syntax U3-13(ASM)
options U3-13(ASM)--U3-15(ASM)
  ABSOLUTE U3-13(ASM)
  COMMAND U3-13(ASM)
  FORMAT U3-13(ASM)
  LIBRARY U3-14(ASM)
  MAP U3-14(ASM)
  NOABS U3-13(ASM)
  NOMAP U3-14(ASM)
  OBJECT U3-14(ASM)
  OPTION U3-14(ASM)
  REFERENCES U3-14(ASM)
  VERSION U3-15(ASM)

Linker command file U3-45(MCC), U2-45(CCC)
alternate U2-19(MCC), U2-16(CCC)
default U2-19(MCC), U2-16(CCC)
passing
  -e option U2-19(MCC), U2-16(CCC)

Linking R4-6(ASM)
LIST assembler command line option U3-3(ASM)
LIST assembler directive R5-47(ASM)
LIST command R3-87(XRAY), U4-2(XRAY)
LISTABS linker command R10-37(ASM)
Listing files format
  /show=include option U3-28(MCC)
  -Flp option U2-20(MCC), U2-17(CCC)
generating
  /list option U3-24(MCC)
  -1 option U2-27(MCC), U2-24(CCC)

omitting page header
  /nopage option U3-28(MCC)
  /pagelength option U3-28(MCC)
  -Flp0 option U2-20(MCC), U2-17(CCC)

page length, specifying
  /pagelength option U3-28(MCC)
  -Flp option U2-20(MCC), U2-17(CCC)
title
  /title option U3-29(MCC)
  -Flt option U2-20(MCC), U2-17(CCC)

Listing source code
  LIST command R3-87(XRAY)
LISTMAP linker command R10-38(ASM)--
R10-39(ASM)
literals section R8-4(MCC), U2-16(MCC), U3-
15(MCC), U2-13(CCC)
naming
  /rename options U3-31(MCC)
  -NL option U2-29(MCC), U2-25(CCC)
specifying address mode
  /literals_addr_as options U3-
16(MCC)
  -al options U2-17(MCC), U2-
14(CCC)

LITTLE R6-1(MCC)
llen assembler command line flag U2-
8(ASM), U3-8(ASM)
LLEN assembler directive R5-48(ASM)
LM_LICENSE_FILE 11-15(FLEX), I2-1(FLEX),
I2-6(FLEX), I3-7(FLEX), U1-1(FLEX),
UA-2(FLEX)
(see also License file)
multiple vendors IC-5(FLEX)
Imdown command I2-8(FLEX), U3-4(FLEX)
Imgrd U2-9(FLEX), UB-2(FLEX)
(see also License server daemons)
exit immediately I2-6(FLEX)
function U2-5(FLEX)
log file messages I2-4(FLEX)
missing I2-6(FLEX)
password computation I3-1(FLEX)
start-up I2-3(FLEX)
manual I2-7(FLEX)
Imgrd command I2-7(FLEX)
Imgrd daemon
loss of licenses when restarting U1-5(FLEX)
multiple copies UC-17(FLEX)
Imhostid command U3-4(FLEX)
Imhostid utility
working without U3-8(FLEX)
Imremove command U3-5(FLEX)
Imreread command U3-6(FLEX)
Imstat command I2-4(FLEX), U3-6(FLEX)
In command I1-15(FLEX), IA-1(FLEX)
Load address R9-6(ASM)
Load button
X Window support UA-12(XRAY)
load button
SunView support UH-12(XRAY)
LOAD command R3-89(XRAY)
/A option U4-12(XRAY)
/NS option U4-12(XRAY)
LOAD linker command R10-40(ASM)–R10-41(ASM)
LOAD_SYMBOLS linker command R10-42(ASM)
Loading
object module
LOAD command R3-89(XRAY)
program
RELOAD command R3-141(XRAY)
Loading files U4-12(XRAY)
application U4-12(XRAY)
ROM support routines U4-12(XRAY)
LOCAL assembler directive R6-7(ASM)–R6-8(ASM)
Local declarations R3-4(MCC), R4-8(CCC)
storage classes R4-8(CCC)
Local labels (??) R7-4(ASM)
Local optimizations R109(MCC)–R1011(MCC), R5-6(XRAY)
/optimize=local option U3-26(MCC)
-Ol option U2-32(MCC), U2-29(CCC)
Local symbols
in macro definition R4-8(XRAY)
referencing R2-16(XRAY)
register R2-16(XRAY)
Local variable initialization
/init_locals option U3-22(MCC)
-KI option U2-25(MCC), U2-22(CCC)
Local variables R6-42(MCC)
displaying
EXPAND command R3-60(XRAY)
Locale functions R5-4(MCC)
localeconv R5-100(MCC)
setlocale R5-152(MCC)
Locale information
querying R5-152(MCC)
setting R5-152(MCC)
locale.h include file R5-4(MCC)
localeconv function R5-100(MCC)
localtime function R5-21(MCC), R5-104(MCC)
relationship to gmtime R5-82(MCC)
Location
current in a file R5-56(MCC), R5-72(MCC), R5-73(MCC), R5-74(MCC), R5-108(MCC)
Locations of input and output files U2-3(CCC)
DOS U2-3(MCC)
UNIX U2-3(MCC)
VMS U3-3(MCC)
lockMRI I2-9(FLEX)
LOG command R3-91(XRAY), U3-17(XRAY)
Log file U2-10(XRAY), U3-17(XRAY)
displaying I2-4(FLEX)
errors in I2-4(FLEX), I2-7(FLEX), I2-8(FLEX)
trimming I3-6(FLEX)
Log file error messages UC-1(FLEX)
log function R5-105(MCC), U2-40(MCC), U3-25(MCC), U2-37(CCC)
Log viewport U3-17(XRAY)
   INCLUDE command U3-17(XRAY)
   LOG command U3-17(XRAY)
log10 function R5-106(MCC), U2-40(MCC),
   U3-25(MCC), U2-37(CCC)
Logarithm of a number, computing
   base 10   R5-106(MCC)
   natural   R5-105(MCC)
Logging commands U4-7(XRAY)
Logic State Analyzer (LSA) S6-4(SUP)
Logical AND operation R2-23(CCC)
Logical names (VMS) U3-3(MCC)
login U1-1(FLEX)
login file
   (see UNIX start-up file)
Long double type R3-1(MCC), R6-3(MCC)
   double-precision format R6-7(MCC)
Long integer
   computing absolute value R5-97(MCC)
   conversion from ASCII string R5-
      33(MCC), R5-179(MCC)
   conversion to ASCII string R5-109(MCC)
   reading from a stream R5-79(MCC)
   writing to stream R5-136(MCC)
/long option (VMS) R9-21(MCC), R9-23(MCC)
long type R3-2(MCC), R6-3(MCC), R6-
   14(MCC)
longjmp
   establishing a label R5-151(MCC)
longjmp function R5-107(MCC)
   relationship to setjmp R5-151(MCC)
   relationship to signal R5-155(MCC)
Look button U2-50(CCC)
Loop controls R7-4(ASM)
Loop invariant code optimization R5-
   10(XRAY)
Loop optimizations R106(MCC), R5-9(XRAY)
Loop rotation optimization R5-11(XRAY)
Lower-case characters
   converting to R5-187(MCC), R5-
      188(MCC)
   testing for R5-89(MCC)
LOWERCASE linker command R10-
   43(ASM)—R10-44(ASM)
Low-level mode
   MODE command R3-92(XRAY)
Low-level screen S1-6(SUP)
LSA S2-151(SUP), S2-153(SUP)
LSA port
   parameter setting S6-9(SUP)
lseek function R5-108(MCC), R9-31(MCC),
   U2-52(MCC), U3-44(MCC), U2-
   44(CCC)
   relationship to fseek R5-72(MCC)
   relationship to fsetpos R5-73(MCC)
   relationship to rewind R5-145(MCC)
ltoa function R5-109(MCC)
ltostr function R5-110(MCC)
lvalue R5-110(MCC)
M
   -M linker command line option U2-14(ASM)
   -m linker command line option U2-14(ASM)
   _M68 preprocessor symbol R4-3(MCC)
Machine-dependent
   optimizations R1016(MCC)—
      R1018(MCC), R5-17(XRAY)
Macro
   assigning to a breakpoint S3-44(SUP),
      S3-48(SUP)
   creating S3-42(SUP)
   deleting S3-43(SUP)
   displaying S3-43(SUP)
   saving to a file S3-43(SUP)
MACRO assembler directive R6-9(ASM)—R6-
   10(ASM)
Macro body R6-9(ASM), R3-45(XRAY)
Macro commands R3-11(XRAY)
   DEFINE R3-11(XRAY), R3-45(XRAY)
   MACRO R4-10(XRAY)
   SHOW R3-11(XRAY)
Macro definition directives R4-6(MCC)
   #define R4-6(MCC), R4-10(MCC)
   #undef R4-6(MCC), R4-31(MCC)
MACRO directive R6-1(ASM)
Macro directives
   (see Directives, assembler)
Macro keyword R4-10(XRAY)
Macro statement R2-3(ASM)
Macro terminator R6-6(ASM)
Macro text prompt (:) R3-46(XRAY)
Macros R6-1(ASM)--R6-11(ASM), U1-3(XRAY), U2-26(XRAY), U4-6(XRAY)
(see Debugger macros)
(see Library macros)
body R6-1(ASM)
call R6-2(ASM)--R6-4(ASM)
creating
  DEFINE command R3-45(XRAY)
defining on command line
    /define option U3-20(MCC)
    -D option U2-18(MCC), U2-14(CCC)
formal parameter R6-2(ASM)
heading R6-1(ASM)
import U4-9(XRAY)
interactive definition of U2-27(XRAY)
no echo
  INCECHO option U4-12(XRAY)
number of parameters
NARG R6-4(ASM)
output U4-9(XRAY)
parameter delimiter (<>) R6-3(ASM)
parameter exists (==) R6-3(ASM)
undefined
  /undefine option U3-33(MCC)
  -U option U2-36(MCC), U2-33(CCC)
until U4-6(XRAY)
when U4-6(XRAY), U4-7(XRAY)
Mail from restarting script I2-5(FLEX)
Makefile S3-3(SUP)
malloc function R5-21(MCC), R5-111(MCC), R9-12(MCC)
  relationship to free R5-67(MCC)
  relationship to realloc R5-143(MCC)
  relationship to setbuf R5-150(MCC)
Mangling RG-1(ASM)--RG-4(ASM), RC-2(CCC), UA-2(CCC)
extern "C" and R5-7(CCC)
function names R5-1(CCC)
overloaded functions and R5-4(CCC)
preventing for C function names R5-1(CCC)
Mantissa from floating-point number R5-70(MCC)
Manual, description RP-1(ASM)
Manually checking out licenses
  (see Checking out licenses)
MAP linker command line option U3-14(ASM)
Mapping overlay S3-32(SUP)
  illegal S3-32(SUP)
  RAM S3-32(SUP)
  ROM S3-32(SUP)
Mapping RAM S2-115(SUP)
Mapping ROM S2-124(SUP)
MASK2 assembler directive R5-49(ASM)
math.h include file R5-5(MCC), R9-12(MCC)
Mathematical functions R5-5(MCC)
  acos R5-24(MCC)
  acosh R5-6(MCC)
  asin R5-26(MCC)
  asinh R5-6(MCC)
  atan R5-28(MCC)
  atanh R5-29(MCC)
  atan2 R5-6(MCC)
  atanh R5-6(MCC)
  ceil R5-36(MCC)
  celi R5-6(MCC)
  cos R5-40(MCC)
  cosh R5-41(MCC)
  coshf R5-6(MCC)
  exp R5-49(MCC)
  expf R5-6(MCC)
  fabs R5-50(MCC)
  fabsf R5-6(MCC)
  floor R5-59(MCC)
  floorf R5-6(MCC)
  fmod R5-60(MCC)
  fmodf R5-6(MCC)
  frexp R5-70(MCC)
  frexpf R5-6(MCC)
  ldexp R5-98(MCC)
  ldexpf R5-6(MCC)
log R5-105(MCC)
log10 R5-106(MCC)
log10f R5-6(MCC)
logf R5-6(MCC)
mod R5-124(MCC)
modf R5-7(MCC)
pow R5-129(MCC)
powf R5-7(MCC)
sin R5-156(MCC)
sinf R5-7(MCC)
sinh R5-157(MCC)
sinhf R5-7(MCC)
sqrt R5-159(MCC)
sqrf R5-7(MCC)
tan R5-183(MCC)
tanf R5-7(MCC)
tanh R5-184(MCC)
tanhf R5-7(MCC)

mathf.h include file R5-6(MCC)
max macro R5-112(MCC)
mblen function R5-113(MCC)
mbstowcs function R5-114(MCC)
mbtowc function R5-115(MCC)
mc assembler command line flag U2-8(ASM), U3-8(ASM)
-Mca option U2-27(MCC), U2-24(CCC)
MCC68K U1-2(FLEX)
mcc68k command U1-2(FLEX), U1-3(FLEX)
mcc68k a5.lib U2-51(MCC), U3-42(MCC), U2-42(CCC)
mcc68ka5020.lib U2-51(MCC), U3-42(MCC), U2-42(CCC)
mcc68kab.lib U2-51(MCC), U3-42(MCC), U2-42(CCC)
mcc68kab020.lib U2-51(MCC), U3-42(MCC), U2-42(CCC)
mcc68kpc.lib U2-51(MCC), U3-42(MCC), U2-42(CCC)
mcc68kpc020.lib U2-51(MCC), U3-42(MCC), U2-42(CCC)

_MCC68K preprocessor symbol R4-2(MCC)
-Mcp option U2-27(MCC), U2-24(CCC)
-Mcp option (UNIX/DOS) R4-3(MCC), R9-19(MCC), R9-20(MCC)

md assembler command line flag U2-8(ASM), U3-8(ASM)
-Md option U2-27(MCC), U2-24(CCC)
-Md option (UNIX/DOS) R4-3(MCC), R4-4(MCC)
-Mda option U2-27(MCC), U2-24(CCC)
-Mdn option (UNIX/DOS) R9-19(MCC), R9-20(MCC)
-Mdp option U2-28(MCC), U2-25(CCC)
-Mdp option (UNIX/DOS) R9-19(MCC), R9-20(MCC)

Member
function R2-2(CCC), RC-2(CCC)
calling from C R5-4(CCC)
specification R4-2(CCC)

Member function UA-2(CCC)
memccpy function R5-116(MCC)
memchr function R5-117(MCC)
memchr macro R4-23(XRAY)
memcmp function R5-119(MCC)
memcpy function R5-120(MCC)
memcpy macro R4-25(XRAY)
memmove function R5-121(MCC)

Memory
access
RAMACCESS command R3-139(XRAY)
addresses, displaying
NOSYMBOLS command R3-108(XRAY)
clearing R4-24(XRAY)
comparison
COMPARE command R3-41(XRAY)
contents, changing
SETMEM command R3-156(XRAY)
copying
COPY command R3-44(XRAY)
copying characters R4-25(XRAY)
disassembling
DISASSEMBLE command R3-52(XRAY)
dump
  DUMP command R3-57(XRAY)
examination
  TEST command R3-197(XRAY)
filing
  FILL command R3-62(XRAY)
initialization RF-6(XRAY)
searching for characters R4-23(XRAY)
setting values R4-26(XRAY)
Memory access size S2-130(SUP)
Memory access variable values S2-90(SUP)
Memory accesses S2-134(SUP)
Memory commands R3-8(XRAY)
  COMPARE R3-8(XRAY), R3-41(XRAY)
  COPY R3-8(XRAY), R3-44(XRAY)
  FILL R3-8(XRAY), R3-62(XRAY)
  NOMEMACCESS R3-8(XRAY), R3-100(XRAY)
  RAMACCESS R3-8(XRAY), R3-139(XRAY)
  ROMACCESS R3-8(XRAY), R3-147(XRAY)
  SEARCH R3-8(XRAY), R3-154(XRAY)
  SETMEM R3-8(XRAY), R3-156(XRAY)
  SETREG R3-8(XRAY), R3-158(XRAY)
  TEST R3-8(XRAY), R3-197(XRAY)
Memory configurations R9-15(MCC), R9-18(MCC), R9-42(MCC), R9-44(MCC)
Memory considerations U2-7(MCC)
  avoiding prototypes U2-8(MCC)
  preprocessing U2-8(MCC)
Memory contents written to file
  SETSTATUS WRITE command R3-178(XRAY)
Memory control
  disassembling S3-30(SUP)
  displaying S3-29(SUP)
  modifying S3-30(SUP)
Memory functions R5-18(MCC)
  memchr R5-117(MCC)
  memcmp R5-119(MCC)
  memcpy R5-120(MCC)
  memmove R5-121(MCC)
  memset R5-122(MCC)
  compare R3-8(XRAY), R3-41(XRAY)
  copy R3-8(XRAY), R3-44(XRAY)
  fill R3-8(XRAY), R3-62(XRAY)
  initialize RF-6(XRAY)
  search R4-23(XRAY)
  set R4-26(XRAY)
  setmem R4-23(XRAY)
  setreg R4-26(XRAY)
Memory layout R6-42(MCC)
Memory space
  allocating R5-146(MCC)
Memory space assignment R9-6(ASM)–R9-8(ASM)
Memory spaces S2-96(SUP)
memset function R5-122(MCC)
memset macro R4-26(XRAY)
MEMVARS S2-90(SUP)
MERGE linker command R10-45(ASM)–R10-46(ASM)
Message severity levels RC-2(MCC)
  librarian RD-1(ASM)
  linker RC-1(ASM)
Messages UC-1(FLEX)
  check-in and check-out I3-6(FLEX)
  error
    (see Error handling)
    get_license UA-1(FLEX)
    log file UC-1(FLEX)
    mllicense or other programs UC-11(FLEX)
      where displayed U3-3(FLEX)
      mllicense.daemon U2-8(FLEX), UC-1(FLEX)
      changing where displayed U3-1(FLEX), UA-2(FLEX)
      where displayed U1-3(FLEX)
      specifying not to write to log I3-6(FLEX)
Messages, diagnostic
  displaying
    /nosuppress option U3-30(MCC)
    -nQ option U2-36(MCC), U2-33(CCC)
  suppressing
    /quit option U3-29(MCC)
    -Q option U2-36(MCC), U2-33(CCC)
  writing to stderr
    /diagnostics_to=stderr option U3-21(MCC)
    -Fee option U2-19(MCC)
  writing to stdout
    /diagnostics_to=stdout option U3-21(MCC)
Messages, error RC-1(MCC)
Messages, help
printing U2-33(XRAY)

Messenger symbols R9-40(MCC)
mex assembler command line flag U2-8(ASM), U3-8(ASM)
MEXIT assembler directive R6-11(ASM)

Microprocessor
(see Processor)

Microprocessor references RP-3(XRAY)

Microprocessor reset
RESET command R3-142(XRAY)

Microprocessor, specifying
CPU option R3-105(XRAY)

Microtec Research extensions U2-39(MCC), U2-36(CCC)
/mrei_extensions option U3-24(MCC)
functions R5-7(MCC)
   eprintf R5-46(MCC)
   ftoa R5-75(MCC)
   getl R5-79(MCC)
   getw R5-81(MCC)
   isascii R5-85(MCC)
   itoa R5-95(MCC)
   itostr R5-110(MCC)
   memccpy R5-116(MCC)
   memclr R5-118(MCC)
   putl R5-136(MCC)
   putw R5-138(MCC)
   swab R5-182(MCC)
   toascii R5-186(MCC)
   _tolower R5-188(MCC)
   _toupper R5-190(MCC)
   zalloc R5-203(MCC)
macros
   BLKSIZE R5-8(MCC)
   FALSE R5-8(MCC)
   fileno R5-58(MCC)
   isascii R5-85(MCC)
   max R5-112(MCC)
   min R5-123(MCC)
   NULLPTR R5-8(MCC)
   stdaux R5-8(MCC)
   stdprn R5-9(MCC)
   toascii R5-186(MCC)
   _tolower R5-188(MCC)
   _toupper R5-190(MCC)
   TRUE R5-9(MCC)
-x option U2-39(MCC), U2-36(CCC)

Microtec Research toolkit
(see Toolkit)
assembler U1-7(XRAY)
compiler U1-5(XRAY)
linker U1-7(XRAY)
object module librarian U1-7(XRAY)

Migrating from version 1 to version 2 IC-1(FLEX), IC-2(FLEX)

min macro R5-123(MCC)

misaligned error message U3-11(XRAY)
Mixed architecture installation IA-2(FLEX)
Mixed operands R3-14(MCC)

mkdir command l1-6(FLEX)

mkfontdir command UA-3(XRAY)

-Ml option U2-28(MCC), U2-25(CCC)
mlicense command U2-9(FLEX), U3-1(FLEX), UB-2(FLEX)
arguments for, specifying U3-3(FLEX)
-C option U2-8(FLEX), U3-1(FLEX)
-d option U3-1(FLEX)
error messages UC-11(FLEX)
examples U3-3(FLEX)
-f option U3-1(FLEX)
   example U3-3(FLEX)
-g option U1-5(FLEX), U3-1(FLEX)
   example U3-3(FLEX)
expiration time U3-2(FLEX)

-h option U1-5(FLEX), U3-2(FLEX)
   example U3-3(FLEX)
-I option U3-2(FLEX)
   output sample U1-4(FLEX)
-i option U3-2(FLEX)
   example U3-3(FLEX)
   output sample U1-4(FLEX)

options
   specifying in file U3-1(FLEX)

Master Index-65
-R option U3-2(FLEX), U3-5(FLEX)
-r option U3-2(FLEX), U3-5(FLEX)
  example U3-3(FLEX)
return code U3-3(FLEX)
status code U3-3(FLEX)
syntax U3-1(FLEX)
-V option U2-8(FLEX), U3-2(FLEX)
-v option U3-2(FLEX)
  example U3-3(FLEX)
verbose mode toggle U3-2(FLEX)
mlicense.daemon I2-3(FLEX), U2-3(FLEX),
  UB-2(FLEX)
  abort U2-3(FLEX)
  checking in licenses manually U3-5(FLEX)
communication with other processes U2-3(FLEX), UA-2(FLEX)
connecting to get_license U2-3(FLEX), UA-1(FLEX), UC-12(FLEX)
holding licenses U1-3(FLEX)
license file access U2-4(FLEX)
license server daemons start-up U2-3(FLEX), U2-6(FLEX)
messages U2-8(FLEX), UC-1(FLEX)
  changing where displayed U3-1(FLEX), UA-2(FLEX)
  where displayed U1-3(FLEX)
start-up U2-3(FLEX)
status code U2-8(FLEX)
  verbose mode toggle U2-8(FLEX), U3-2(FLEX), UA-2(FLEX)
MLICENSE_DAEMON_OUTPUT U2-8(FLEX), U3-1(FLEX), UA-2(FLEX)
MLICENSE_DAEMON_TMP U2-3(FLEX), UA-2(FLEX), UC-9(FLEX)
MLICENSE_DAEMON_VERBOSE U2-8(FLEX), UA-2(FLEX)
MLICENSE_HOURS U1-3(FLEX), U3-2(FLEX), UA-2(FLEX)
Mode button
  X Window support UA-12(XRAY)
mode button
  SunView support UH-12(XRAY)

MODE command R3-92(XRAY), U2-20(XRAY), U3-1(XRAY)
  ASSEMBLY option U4-2(XRAY)
  HIGH option U4-2(XRAY)

MODE command key
  Apollo support UB-4(XRAY)
  Apollo trace support UB-6(XRAY)
  DECstation support UC-4(XRAY)
  DECstation trace support UC-5(XRAY)
  HP support UD-6(XRAY)
  HP trace support UD-8(XRAY)
  IBM RS/6000 support UE-3(XRAY)
  IBM RS/6000 trace support UE-5(XRAY)
  Motorola Delta Series support UF-3(XRAY)
  Motorola Delta Series trace support UF-5(XRAY)
  PC support UG-3(XRAY)
  PC trace support UG-5(XRAY)
  Sun support UH-4(XRAY)
  Sun trace support UH-5(XRAY)
  VT terminal support UI-3(XRAY)
  VT terminal trace support UI-5(XRAY)

Mode selection for debugging
  MODE command R3-92(XRAY)

Modes
  address
    (see Address modes)
  ANSI-compliant
    /ansi option U3-11(MCC)
    -A option U2-8(MCC), U2-16(MCC),
      U2-12(CCC)
  processor
    /cpu option U3-16(MCC)
    -p option U2-34(MCC), U2-30(CCC)
  verbose
    -V options U2-37(MCC), U2-34(CCC)
  modf function R5-124(MCC)

Modifying license file l1-9(FLEX), l1-10(FLEX)
Module (@module) R2-21(XRAY)
Module (definition) RD-1(XRAY)
@module pseudo-register RA-1(XRAY), RF-3(XRAY)
Module record RF-1(ASM)
Module, naming
/rename option U3-31(MCC)
-NM option U2-29(MCC), U2-26(CCC)
Monitor button
X Window support UA-12(XRAY)
SunView support UH-12(XRAY)
MONITOR command R3-93(XRAY), U2-18(XRAY), U4-15(XRAY)
with C++ RG-15(XRAY)
Monitoring
expressions
MONITOR command R3-93(XRAY)
NOMONITOR command R3-102(XRAY)
memory location
NOMEMACCESS command R3-100(XRAY)
Monitoring variables U2-18(XRAY), U2-22(XRAY)
in Data viewport U2-19(XRAY)
Monochrome displays for PCs
COLOR option R3-105(XRAY)
MORE option U2-22(XRAY)
Motorola Delta Series
host-specific information UF-1(XRAY)
Motorola Delta Series support
MOVE TO BOTTOM control key UF-3(XRAY)
MOVE TO TOP control key UF-2(XRAY)
Motorola S-record
(see S-record format)
Mouse support
Apollo
(see DOMAIN/OS)
DOMAIN/OS
menu selections
control strings UB-10(XRAY)
default settings UB-9(XRAY)
user-defined settings UB-12(XRAY)
viewport operations UB-7(XRAY)
COPY key UB-7(XRAY)
highlighting text UB-7(XRAY)
PASTE key UB-7(XRAY)
SunOS
scratch area UH-15(XRAY)
display size UH-15(XRAY)
edit operations UH-15(XRAY)
selection buttons UH-9(XRAY)
constructing commands UH-10(XRAY)
default settings UH-11(XRAY)
user-defined settings UH-14(XRAY)
viewport operations UH-6(XRAY)
activate viewport UH-7(XRAY)
move viewport UH-7(XRAY)
resize viewport UH-7(XRAY)
scroll viewport UH-7(XRAY)
zoom viewport UH-7(XRAY)
X Window
selection buttons UA-10(XRAY)
constructing commands UA-10(XRAY)
default settings UA-12(XRAY)
user-defined settings UA-13(XRAY)
viewport operations
activate viewport UA-9(XRAY)
move viewport UA-9(XRAY)
resize viewport UA-9(XRAY)
scroll viewport UA-9(XRAY)
zoom viewport UA-9(XRAY)
MOVE TO BOTTOM command keys U3-36(XRAY)
MOVE TO BOTTOM control key
Apollo support UB-2(XRAY)
HP support UD-5(XRAY)
IBM RS/6000 support UE-2(XRAY)
Motorola Delta Series support UF-3(XRAY)
PC support UG-3(XRAY)
Sun support UH-3(XRAY)
VT terminal support UI-2(XRAY)
MOVE TO TOP command keys U3-36(XRAY)
MOVE TO TOP control key
HP support  UB-2(XRAY), UD-5(XRAY)
IBM RS/6000 support  UE-2(XRAY)
Motorola Delta Series support  UF-2(XRAY)
PC support  UG-2(XRAY)
Sun support  UH-3(XRAY)
VT terminal support  UI-2(XRAY)

MRI daemon  I3-2(FLEX), I3-3(FLEX), U2-5(FLEX), UB-2(FLEX), UB-3(FLEX)
(see also License server daemons)
log file messages  I2-4(FLEX)
loss of licenses when restarting  U1-5(FLEX)
not starting  I2-4(FLEX)
specifying in license file  U2-9(FLEX)
start-up  I2-3(FLEX)
status  U3-6(FLEX)
status, normal  I2-4(FLEX)

_MRI_EXTENSIONS preprocessor symbol  U3-25(MCC)
_MRI preprocessor symbol  R4-2(MCC)
MRI toolchain compatibility  S1-1(SUP)
_MRI_68K_BIN  U2-4(MCC), U2-6(MCC), U2-4(CCC), U2-5(CCC)
_DOS  I1-4(MCC), I1-4(XRAY), I1-5(CCC)
_MRI_68K_INC  U2-4(MCC), U2-7(MCC), U2-23(MCC), U2-4(CCC), U2-6(CCC), U2-20(CCC)
_DOS  I1-4(MCC), I1-4(XRAY), I1-5(CCC)
_MRI_68K_LIB  U2-4(MCC), U2-7(MCC), U2-4(CCC), U2-5(CCC)
_DOS  I1-4(MCC), I1-5(CCC)
_MRI_68K_LIB environment variable  U2-15(ASM)
_MRI_68K_TMP  U2-4(MCC), U2-7(MCC), U2-4(CCC), U2-6(CCC)
_DOS  I1-4(MCC), I1-5(CCC)
mrcheckin command  IC-1(FLEX)
mrcheckout command  IC-1(FLEX)
mriext.h include file  R5-7(MCC), R5-8(MCC)
MS-DOS
(see DOS installation)
(see DOS)
_MSdos preprocessor symbol  R4-3(MCC)
mstart_lmgrd  I2-3(FLEX), U1-2(FLEX), U1-3(FLEX)
mt command  I1-19(FLEX)
Multiple flags, entering
  assembler  U2-5(ASM), U3-12(ASM)
Multiple inheritance  R3-10(CCC), RC-2(CCC), UA-2(CCC)
Multiple jump optimization  R1013(MCC), R5-14(XRAY)
Multiple modules, debugging  U2-10(XRAY)
Multiple statement debugging  R2-11(XRAY)
Multistatement debugging  R2-11(XRAY)
  colon qualifier  R2-12(XRAY)
  dot qualifier  R2-12(XRAY)
Multitasking environment (MTE)  R9-12(MCC), R9-34(MCC)
Multi-threaded environments  R9-12(MCC)
MWARM command  S2-91(SUP), S2-94(SUP)
MWARN  S2-91(SUP)

N
-nA compiler option  R4-6(CCC), RB-3(CCC)
-nA option  U2-16(MCC), U2-12(CCC)
NAME assembler directive  R5-50(ASM)
Name demangling  RG-1(ASM)–RG-4(ASM),
  RB-6(CCC), RC-3(CCC), U2-46(CCC),
  UA-2(CHAR)
NAME linker command  R10-47(ASM)
Name mangling  RG-1(ASM)–RG-4(ASM),
  RB-6(CCC), RC-3(CCC), UA-2(CHAR)
Names, truncating
  /truncate_identifiers  U3-12(MCC)
  -ut option  U2-8(MCC), U2-37(MCC), U2-34(CCC)
Naming convention for symbols  U2-37(MCC), U3-13(MCC), U2-34(CCC)
Naming modules
  /rename option  U3-31(MCC)
  -NM option  U2-29(MCC), U2-26(CCC)
NARG reserved symbol  R2-9(ASM), R6-4(ASM), R8-3(ASM)
-NC option  U2-28(MCC), U2-25(CCC)
-nC option  U2-17(MCC), U2-14(CCC)
-nc option U2-18(MCC), U2-14(CCC)
+nd option U2-15(CCC)
NDEBUG macro R5-27(MCC)
-ne option U2-19(MCC), U2-16(CCC)
+n option U2-16(CCC)
+ne option, use of R3-18(CCC)
Negative option prefix U2-10(MCC), U3-5(MCC)
nest assembler command line flag U2-8(ASM), U3-8(ASM)
Nest class R4-16(CCC)
structures R4-16(CCC)
Nested procedure R2-23(XRAY)
Nesting (@) R2-23(XRAY)
NETFAIL S2-92(SUP)
NETERR S2-93(SUP)
Networks U2-11(FLEX), UB-2(FLEX)
multiple architectures U2-11(FLEX)
specifying U2-1(FLEX)
new keyword R4-4(CCC)
new operator R2-19(CCC), R2-21(CCC), RC-3(CCC), UA-3(CCC)
Newline
disable for preprocessor
-Es option U2-19(MCC), U2-16(CCC)
-Ps option U2-33(MCC), U2-30(CCC)
NEXT assembler directive R7-11(ASM)
NEXT command R3-97(XRAY)
Next key (HP support) UD-5(XRAY)
-nf option U2-20(MCC), U2-18(CCC)
-nFee option U2-19(MCC)
-nFio option U2-20(MCC)
-nFl option U2-20(MCC), U2-17(CCC)
-nFs option U2-20(MCC)
-nFsm option U2-20(MCC), U2-17(CCC)
-ng option U2-22(MCC), U2-19(CCC)
-nGf option U2-21(MCC), U2-18(CCC)
-nG options U2-21(MCC), U2-18(CCC)
-nGr option U2-21(MCC), U2-18(CCC)
-nGs option U2-22(MCC), U2-19(CCC)
-nH option U2-22(MCC), U2-19(CCC)
-NI option U2-29(MCC), U2-25(CCC)
+n option U2-20(CCC)
-nK2 option U2-24(MCC), U2-21(CCC)
-nK4 option U2-24(MCC), U2-21(CCC)
-nKc option U2-24(MCC), U2-21(CCC)
-nKf option U2-25(MCC), U2-22(CCC)
-nKl option U2-25(MCC), U2-22(CCC)
-nKr option U2-26(MCC), U2-23(CCC)
-nKT option U2-26(MCC), U2-23(CCC)
-nKt option U2-26(MCC), U2-23(CCC)
-nKu option U2-27(MCC), U2-23(CCC)
-nKw option (UNIX/DOS) R4-3(MCC)
-NL option U2-29(MCC), U2-25(CCC)
-nl option U2-27(MCC), U2-24(CCC)
-NM option U2-29(MCC), U2-26(CCC)
-no option U2-30(MCC), U2-27(CCC)
no option prefix
VMS U3-5(MCC)
NOABS linker command line option U3-13(ASM)
-nOb option U2-30(MCC), U2-27(CCC)
-nOc option U2-31(MCC), U2-28(CCC)
-nOc option (UNIX/DOS) R7-6(MCC), R1017(MCC)
NODEBUG_SYMBOLS linker command R10-26(ASM)
Node-locked licenses I1-1(FLEX), U2-2(FLEX), U2-3(FLEX), U2-10(FLEX), UB-2(FLEX)
checking out U2-4(FLEX)
installing only I1-11(FLEX), I2-6(FLEX)
mllicense.daemon operation U2-3(FLEX)
specifying in license file I3-4(FLEX), U2-9(FLEX)
Node-locking U2-2(FLEX)
-nOe option U2-31(MCC), U2-28(CCC)
NOERROR command R10-28(ASM)
NOFORMAT assembler directive R5-27(ASM)
-nOg option U2-31(MCC), U2-28(CCC)
-node option U2-31(MCC), U2-28(CCC)
Not supported S1-8(SUP)
NOICE command R3-98(XRAY)

Master Index-69
Nointerrupt  
not supported S1-8(SUP)  
NOINTERRUPT command R3-99(XRAY)  
-nOi option U2-32(MCC), U2-29(CCC)  
NOLIST assembler command line option U3-3(ASM)  
NOLIST assembler directive R5-47(ASM)  
NOLOG line 3-6(FLEX)  
NOMAP linker command line option U3-14(ASM)  
Nomem  
not supported S1-8(SUP)  
NOMEMACCESS command R3-100(XRAY), U2-11(XRAY)  
NOMONITOR command R3-102(XRAY)  
Nonlocal jump  
functions R5-9(MCC)  
  longjmp R5-107(MCC)  
  setjmp R5-151(MCC)  
  type  
    jmp_buf R5-9(MCC)  
Nonmember function  
calling from C R5-4(CCC)  
Nonprintable characters R2-5(MCC), R2-6(XRAY)  
Non-reentrant functions R5-20(MCC)  
  (see also Reentrant functions)  
Nonrewinding devices 11-18(FLEX)  
NOOBJ assembler directive R5-51(ASM)  
NOOBJECT assembler command line option U3-3(ASM)  
NOPAGE assembler directive R5-61(ASM)  
NOPAGE linker command R10-50(ASM)–R10-51(ASM)  
-nOR option U2-32(MCC), U2-29(CCC)  
-nOr option U2-32(MCC), U2-29(CCC)  
Normalized form R6-5(MCC)  
-nP option U2-33(MCC), U2-30(CCC)  
+np option U2-32(CCC)  
-null macro R5-11(MCC), R5-15(MCC), R5-17(MCC), RB-2(ASM)  
Null target mode S2-94(SUP)  
NULL_TGT (enable null target mode) S2-94(SUP)  
Number generator, pseudorandom R5-140(MCC)  
Numbers, specifying U4-17(XRAY)  
Numeric constants R2-1(MCC)  
Numeric interpretation  
  RADIX option R3-107(XRAY)  
-nV option U2-38(MCC), U2-35(CCC)  
-nv option U2-38(MCC), U2-35(CCC)  
+nw option U2-36(CCC)  
-nx option U2-39(MCC), U2-36(CCC)  
-ny option U2-40(MCC), U2-37(CCC)  
-NZ option U2-29(MCC), U2-26(CCC)  
O  
-o assembler command line flag U2-8(ASM), U3-8(ASM)  
-o assembler command line option U2-3(ASM)  
-o linker command line option U2-14(ASM)  
O opcode error R7-14(ASM)  
-O option U2-30(MCC), U2-27(CCC)  
-o option U2-33(MCC), U2-30(CCC)  
-Ob option U2-30(MCC), U2-27(CCC)  
OBJECT assembler command line option U3-3(ASM)  
Object file  
  -o assembler option U2-3(ASM)  
Object files  
naming  
  -o option U2-33(MCC), U2-30(CCC)  
producing only  
  -c option U2-18(MCC), U2-14(CCC)  
Object handle  
  keywords R4-6(CCC)
OBJECT linker command line option U3-14(ASM)
Object module, loading
LOAD command R3-89(XRAY)
-Oc option U2-31(MCC), U2-28(CCC)
Octal representation RA-1(MCC), RB-1(XRAY)
Odd-address restricted processors R6-23(MCC)
Odd-address unrestricted processors R6-23(MCC)
-Oe option U2-31(MCC), U2-28(CCC)
OFFSET assembler directive R5-52(ASM)–R5-53(ASM)
Offset of structure member, determining R5-125(MCC)
offsetof macro R5-125(MCC)
-Og option U2-31(MCC), U2-28(CCC)
-0i option U2-31(MCC), U2-28(CCC)
old assembler command line flag U2-8(ASM), U3-8(ASM)
One-of qualification RG-7(XRAY), RG-9(XRAY)
On-line help U1-4(XRAY), U2-14(XRAY)
On-line help menu
HELP command R3-72(XRAY)
op assembler command line flag U2-8(ASM), U3-8(ASM)
opcode error (O) R7-14(ASM)
Open file for writing
FOPEN command R3-66(XRAY)
open function R5-126(MCC), R9-31(MCC), U2-52(MCC), U3-44(MCC), U2-44(CCC)
rela relation ship to close R5-39(MCC)
rela relation ship to fopen R5-62(MCC)
rela relation ship to read R5-142(MCC)
rela relation ship to write R5-202(MCC)
OPEN librarian command R13-15(ASM)
Opening a file R5-61(MCC), R5-126(MCC)
OpenWindows operation UA-6(XRAY)
Operand syntax R3-17(ASM)–R3-20(ASM)
Operands
overloading R2-25(CCC)
precedence RC-3(XRAY)
sizes R5-11(MCC)

opnop assembler command line flag U2-9(ASM), U3-9(ASM)

OPT assembler directive R5-54(ASM)—R5-59(ASM)

Optimizations R101(MCC)
  algebraic simplification R101(MCC)
  array operator synthesis R106(MCC)
  branch tail R5-12(XRAY)
  branch tail merging R1011(MCC)
  code for epilogue R1016(MCC)
  code for prologue R1016(MCC)
  constant folding R109(MCC), R5-6(XRAY)
  cross-jump R1013(MCC), R5-13(XRAY)
  dead code elimination R5-2(XRAY)
  factoring R5-3(XRAY)
  general R101(MCC)
    algebraic simplification R5-6(XRAY)
    redundant code elimination R5-7(XRAY)
    strength reduction R5-5(XRAY)
  global R102(MCC)—R109(MCC), R5-2(XRAY)
  global constant propagation R104(MCC), R5-3(XRAY)
  global copy propagation R105(MCC), R5-3(XRAY)
  grouping stack adjust instructions R1017(MCC)
  indexing arrays R1018(MCC), R5-18(XRAY)
  in-line library function expansion R1017(MCC)
  instruction scheduling R1016(MCC)
  jump R1011(MCC)—R1015(MCC), R5-12(XRAY)
  local R109(MCC)—R1011(MCC), R5-6(XRAY)
  loop R106(MCC), R5-9(XRAY)
  loop invariant code optimization R5-10(XRAY)

loop rotation R5-11(XRAY)
machine dependent R1016(MCC)—R1018(MCC)
machine-dependent R5-17(XRAY)
multiple jump R1013(MCC), R5-14(XRAY)
redundant code elimination R102(MCC)
redundant jump elimination R1014(MCC), R5-14(XRAY)
redundant load and store elimination R5-8(XRAY)
register coloring R5-4(XRAY)
short circuit evaluation R5-8(XRAY)
short/displacement R5-15(XRAY)
short/long displacement R1015(MCC)
short-circuit evaluation R1011(MCC)
strength reduction R102(MCC)
strength reduction and index simplification R109(MCC), R5-11(XRAY)
subexpression elimination R5-6(XRAY)
time versus size U3-28(MCC)
unreachable code R5-5(XRAY)
unused definition elimination R5-5(XRAY)

/optimize option (VMS) R1017(MCC)
/optimize=nocombine_pops option (VMS) R7-6(MCC)

Optimizing code U2-30(MCC), U2-27(CCC)
  execution time
    /optimize=time option U3-28(MCC)
    -Ot option U2-33(MCC), U2-29(CCC)
    global-flow optimizer
      /optimize=globalflow option U3-26(MCC)
      -Og option U2-31(MCC), U2-28(CCC)
  in-lining
    /optimize=inline option U3-26(MCC)
    -Oi option U2-31(MCC), U2-28(CCC)
  local optimizations
    /optimize=local option U3-26(MCC)
    -Ol option U2-32(MCC), U2-29(CCC)
register coloring
   /optimize=register option  U3-26(MCC)
   -OR option U2-32(MCC), U2-29(CCC)

size
   /optimize=size option  U3-27(MCC)
   -Os option U2-32(MCC), U2-29(CCC)

OPTION S1-8(SUP)
OPTION command  R3-103(XRAY), U2-22(XRAY), U4-17(XRAY)
   ALIGN option U2-21(XRAY)
   BREAK option U3-7(XRAY)
   LINES option U3-3(XRAY), U4-3(XRAY)
   SYMBOLS option U4-3(XRAY)

Option descriptions
   DOS U2-11(MCC)
   UNIX U2-11(MCC)
   VMS U3-5(MCC)

Option form, positive and negative  U2-6(CCC)

OPTION librarian command line option  U3-19(ASM)

OPTION linker command line option  U3-14(ASM)

_OPTION_utn preprocessor symbol  R4-3(MCC)

Options
   command line  U2-10(MCC), U3-5(MCC)
   conflicting, specifying  U2-10(MCC), U3-5(MCC)
   debugger  S3-18(SUP)
   descriptions  U2-10(MCC), U3-5(MCC)
   file, specifying
      -d option  U2-19(MCC), U2-15(CCC)
   negative  U2-10(MCC), U3-5(MCC)
   passing to assembler
      -Wa option U2-38(MCC), U2-35(CCC)
   passing to linker
      -Wl option U2-38(MCC), U2-35(CCC)
   summary  U2-11(MCC), U3-5(MCC), U2-7(CCC)

VMS
   /code_addresses  R4-3(MCC)
   /data_addresses  R4-3(MCC)

Options active listing
   suppression
      /print_options option  U3-30(MCC)
      -Qo option U2-36(MCC), U2-33(CCC)

Options for XRAY saved
   STARTUP command  R3-180(XRAY)

Options, command line  U2-9(XRAY), U2-6(CCC)
   negative form  U2-6(CCC)
   specifying conflicting  U2-6(CCC)

   -OR option U2-32(MCC), U2-29(CCC)
   -Or option U2-32(MCC), U2-29(CCC)

Order directory  l1-9(FLEX), l1-10(FLEX)
ORDER linker command  R10-48(ASM)–R10-49(ASM)

Order number, purpose of entering  l1-9(FLEX), l1-10(FLEX)

Order of evaluation  R3-13(MCC)
ORG assembler directive  R5-60(ASM)
   -Os option U2-32(MCC), U2-29(CCC)
   osm68ka5.lib  U2-42(CCC)
   osm68ka5020.lib  U2-42(CCC)
   osm68kab.lib  U2-42(CCC)
   osm68kab020.lib  U2-42(CCC)
   osm68kpc.lib  U2-42(CCC)
   osm68kpc020.lib  U2-42(CCC)
   -Ot option U2-33(MCC), U2-29(CCC)

Out-of-range warnings  S2-91(SUP), S2-94(SUP)

OUTPORT S1-8(SUP), S7-1(SUP)
OUTPORT command  R3-110(XRAY), U2-11(XRAY), U4-9(XRAY)
   size qualifiers  R3-3(XRAY)
output macro  R4-27(XRAY), U4-9(XRAY)

Output
   assembler source file  R11-1(MCC)
   listing  R11-3(MCC)

Output file
   -o assembler option  U2-3(ASM)
Output files
- extensions U2-3(MCC), U3-2(MCC)
- locations U2-3(MCC), U3-3(MCC)
- naming
  -o option U2-33(MCC), U2-30(CCC)
- specifying format
  /show=include option U3-28(MCC)
  -Fli option U2-20(MCC), U2-17(CCC)

Output files, rewinding
- ROUT command R3-149(XRAY)
- OUTPUT librarian command line option U3-19(ASM)

Output port
- address
  OUTPUTPORT command R3-110(XRAY)
- buffer
  DOUT command R3-54(XRAY)

Output ports, writing value to
- R4-27(XRAY)

Output, multipage
- MORE option R3-107(XRAY)

OVE S2-96(SUP)
- Overflow counter S6-10(SUP)
- OVERLAY S1-8(SUP)

Overlay
- mapping S2-115(SUP), S2-124(SUP)

Overlay memory
- copying target to overlay S3-33(SUP)
- displaying S3-31(SUP)
- mapping S3-32(SUP)

Overlay memory spaces S2-96(SUP)
- Overlay speed S2-96(SUP), S2-98(SUP)

Overloaded function RC-3(CCC), UA-3(CCC)
- Overloaded operators R2-25(CCC), RC-3(CCC), UA-3(CCC)
- Overloading RA-15(CCC), RA-16(CCC), RC-3(CCC), UA-3(CCC)
- conversion operator R4-11(CCC)
- function
  - names R5-1(CCC)
  - operators R2-25(CCC)
  - delete RA-8(CCC)
  - keyword R4-4(CCC)
  - new RA-9(CCC)

Overview R1-1(XRAY)
- OVS (overlay speed) S2-98(SUP)
- OWL environment variable U2-46(CCC)

owl68k inspection tool
  (see C++ inspector)

P

p assembler command line flag U2-9(ASM), U3-9(ASM)

+p compiler option R3-3(CCC)

-P option U2-33(MCC), U2-30(CCC)

+p option U2-32(CCC)

-p option U2-34(MCC), U2-30(CCC)

-p option (UNIX/DOS) R4-4(MCC)

Packed enumerator type
- /define option U3-21(MCC)

-D option U2-18(MCC), U2-15(CCC)

packed keyword R6-27(MCC), R4-4(CCC)

packed keyword, disabling
- -nx option U2-40(MCC), U2-37(CCC)

Packed structures R6-27(MCC), R6-29(MCC)

packd keyword, disabling
- /define option U3-21(MCC)
- -D option U2-18(MCC), U2-15(CCC)

Packed structures R6-27(MCC), R6-29(MCC)

-tips R6-39(MCC)

packed type R6-38(MCC)

Packing
- bit fields R6-23(MCC)
- enumerated data types R6-40(MCC)

-tips R6-39(MCC)

Padding R6-17(MCC), R6-22(MCC)

-bytes R6-31(MCC), R6-34(MCC)

-structures R6-29(MCC)

PAGE assembler directive R5-61(ASM)

Page header, specifying for listing file
- /nopage option U3-28(MCC)
- /pagelength option U3-28(MCC)

-Flp0 option U2-20(MCC), U2-17(CCC)

PAGE linker command R10-50(ASM)–R10-51(ASM)

Parameters
- passing R7-1(MCC)
- popping R7-5(MCC)
- setting R7-1(MCC)
Parentheses
redundant R4-11(CCC)
usage in C++ R4-11(CCC)

Part A
(see install.sh)

Part B
(see install.sh)

Part number I1-9(FLEX), I1-10(FLEX)

Passing options
to assembler
-Wa option U2-38(MCC), U2-35(CCC)
to linker
-Wl option U2-38(MCC), U2-35(CCC)

Password U2-9(FLEX)
Passwords, encryption of I3-1(FLEX)

Patching code S3-30(SUP)
Patching source R4-12(XRAY)

PATH
DOS I1-4(MCC), I1-4(XRAY)
path (csh variable) I2-1(FLEX), U1-1(FLEX)
PATH (sh variable) I2-1(FLEX), U1-1(FLEX)
Path size R3-1(XRAY)
Path, search
SETSTATUS ENVIRONMENT command R3-160(XRAY)

Paths
relative U1-3(FLEX)
PAUSE command R3-114(XRAY)
Pause debugging session
PAUSE command R3-114(XRAY)

PC
host-specific information UG-1(XRAY)

PC monochrome displays
COLOR option R3-105(XRAY)

_PC preprocessor symbol R4-3(MCC)

PC support
MOVE TO BOTTOM control key UG-3(XRAY)
MOVE TO TOP control key UG-2(XRAY)

PC-DOS
(see DOS installation)
(see DOS)

pco assembler command line flag U2-9(ASM), U3-9(ASM)
pcc assembler command line flag U2-10(ASM), U3-10(ASM)

PC-relative address mode
code references
(see also Position-independent code)
/code_addresses=pcrelative
   option U3-14(MCC)
-Mcp option U2-27(MCC), U2-24(CCC)
data references
(see also Position-independent data)
/data_addresses=pcrelative
   option U3-14(MCC)
-Mdp option U2-28(MCC), U2-25(CCC)

PC-relative addressing R9-27(MCC)

pcs assembler command line flag U2-10(ASM), U3-10(ASM)

Peek/poke
bus errors S2-33(SUP)

Peek/poke trace S2-114(SUP), S2-116(SUP),
S2-117(SUP), S2-119(SUP), S2-120(SUP), S2-122(SUP), S2-123(SUP), S2-126(SUP), S2-128(SUP), S2-129(SUP), S2-130(SUP), S2-132(SUP), S2-134(SUP), S2-136(SUP), S2-137(SUP), S2-138(SUP), S2-139(SUP), S2-141(SUP), S2-142(SUP), S2-143(SUP), S2-144(SUP), S2-145(SUP), S2-146(SUP), S2-151(SUP), S2-153(SUP), S2-154(SUP), S2-156(SUP), S2-161(SUP), S2-162(SUP)

PERFACT (enable/disable) S2-100(SUP)
PERFCLR (remove data) S2-101(SUP)
PERFDATA (display symbol data) S2-102(SUP)
PERFDEPTH S2-103(SUP)
PERFDISP (display) S2-104(SUP)
PERFEX (exclude addresses) S2-106(SUP)
PERFEXCLR (clear excluded addresses)  S2-108(SUP)
PERFFORMAT (specify format)  S2-109(SUP)
PERFIN (specify time interval)  S2-111(SUP)
PERFMODE (control data display)  S2-112(SUP)
Performance analysis  S2-100(SUP), S2-101(SUP), S2-104(SUP), S2-106(SUP), S2-108(SUP), S2-109(SUP), S2-111(SUP), S2-112(SUP), S2-113(SUP), S4-1(SUP)–S4-3(SUP), ??–S4-4(SUP)
data collection  R6-8(XRAY)
   PROFILE  R3-135(XRAY), R6-8(XRAY)
PERFDEPTH  S2-103(SUP)
PRINTPROFILE command  R3-123(XRAY)
PROFILE command  R3-134(XRAY)
program_unit  R3-134(XRAY)
reporting results  R6-8(XRAY)
   PRINTPROFILE  R6-9(XRAY)
   PRINTPROFILE command  R3-124(XRAY)
tutorial  R6-11(XRAY)
Performance analysis results
   PRINTPROFILE command  R3-123(XRAY)
PERFTOL (symbol search distance)  S2-113(SUP)
Peripheral activity, Pause mode  S2-86(SUP)
error function  R5-128(MCC)
   PF1 key (VT terminal support)  UI-2(XRAY)
   PF2 key (VT terminal support)  UI-2(XRAY)
   PF3 key (VT terminal support)  UI-2(XRAY)
   PF4 key (VT terminal support)  UI-2(XRAY)
   Pg Dn key (PC support)  UG-2(XRAY)
   Pg Dn key (RS/6000 support)  UE-2(XRAY)
   Pg Up key (PC support)  UG-2(XRAY)
   Pg Up key (RS/6000 support)  UE-2(XRAY)
   @pi pseudo-register  RA-1(XRAY), RF-3(XRAY)
   _PIC preprocessor symbol  U2-27(MCC), U2-24(CCC)
   _PID preprocessor symbol  U2-27(MCC), U2-24(CCC)
   _PID_REG preprocessor symbol  U2-27(MCC), U2-24(CCC)
   @pizzare pseudo-register  RA-1(XRAY), RF-3(XRAY)
pixinit section  R7-5(CCC), U2-13(CCC)
PLEN assembler directive  R5-62(ASM)
Pointer
   base  R5-10(CCC)
   bound  RA-6(CCC)
   operators
      .  R2-17(CCC), R2-25(CCC)
      .*  R2-18(CCC), R2-25(CCC)
      ->  R2-17(CCC), R2-18(CCC), R2-25(CCC)
      -*  R2-18(CCC)
      new  R2-21(CCC)
      to member  R2-18(CCC)
      typing  R2-18(CCC)
      to object  R2-7(CCC)
      void*  R4-7(CCC)
Pointer operators
      .  R2-17(CCC), R2-19(CCC), R2-21(CCC)
      .*  R2-18(CCC)
      ::*  R2-18(CCC)
      ->  R2-17(CCC)
      -*  R2-18(CCC)
      data member  R2-19(CCC)
      delete  R2-21(CCC)
      member function  R2-19(CCC)
      new  R2-19(CCC)
Pointer to data member  R2-19(CCC)
Pointer to member  RC-4(CCC), UA-3(CCC)
Pointer to member function  R2-19(CCC)
Pointers  R3-2(MCC), R6-8(MCC)
   types  R3-2(MCC), R6-3(MCC)
Polling the emulator  S2-126(SUP)
Popping stack
   /optimize=combine_pops option  U3-25(MCC)
   -Of option  U2-31(MCC), U2-28(CCC)
Port address
input
  INPORT command R3-80(XRAY)
output
  OUTPORT command R3-110(XRAY)
Port buffers
displaying input
  DIN command R3-50(XRAY)
displaying output
  DOUT command R3-54(XRAY)
Port I/O and interrupt commands R3-9(XRAY)
  DIN R3-9(XRAY), R3-50(XRAY)
  DOUT R3-9(XRAY), R3-54(XRAY)
  INPORT R3-9(XRAY), R3-80(XRAY)
  INTERRUPT R3-9(XRAY), R3-84(XRAY)
  NOINTERRUPT R3-9(XRAY), R3-99(XRAY)
  OUTPORT R3-9(XRAY), R3-110(XRAY)
  RIN R3-9(XRAY)
  RIN command R3-146(XRAY)
  ROUT R3-9(XRAY), R3-149(XRAY)
port@host l3-7(FLEX)
@port_addr pseudo-register R3-81(XRAY), R3-111(XRAY), RA-1(XRAY), RF-3(XRAY)
@port_size pseudo-register RA-1(XRAY), RF-3(XRAY)
@port_value pseudo-register R3-81(XRAY), R3-111(XRAY), RA-2(XRAY), RF-4(XRAY)
Position-dependent
code R9-19(MCC)
data R9-19(MCC)
Position-independent
code R9-19(MCC)
data R9-19(MCC)
Position-independent code
  /code_addresses=pcrelative U3-14(MCC)
  -Mcp option U2-27(MCC), U2-24(CCC)

Position-independent data
  /data_addresses=anrelative option U3-14(MCC)
  /data_addresses=pcrelative option U3-14(MCC)
  -Md options U2-27(MCC), U2-24(CCC)
pow function R5-129(MCC)
PPT S2-96(SUP), S2-114(SUP), S2-116(SUP), S2-117(SUP), S2-119(SUP), S2-120(SUP), S2-122(SUP), S2-123(SUP), S2-126(SUP), S2-128(SUP), S2-129(SUP), S2-130(SUP), S2-132(SUP), S2-134(SUP), S2-136(SUP), S2-137(SUP), S2-138(SUP), S2-139(SUP), S2-141(SUP), S2-142(SUP), S2-143(SUP), S2-144(SUP), S2-145(SUP), S2-146(SUP), S2-151(SUP), S2-153(SUP), S2-154(SUP), S2-156(SUP), S2-161(SUP), S2-162(SUP)
  #pragma asm directive R4-24(MCC), R9-7(MCC)
  #pragma directives R4-8(MCC)
  #pragma endasm directive R4-24(MCC), R9-8(MCC)
  #pragma error directive R4-25(MCC)
  #pragma info directive R4-26(MCC)
  #pragma list directive R4-27(MCC)
  #pragma macro directive R4-28(MCC)
  #pragma option directive R4-29(MCC)
  #pragma options R6-37(MCC)
  #pragma warn directive R4-30(MCC)
Pre-ANSI C
  function declarations R4-12(CCC)
Precedence
  expressions R3-13(MCC)
Precedence order of operators R2-1(XRAY)
Predefined symbols R4-1(MCC)
  _DATE_ R4-1(MCC)
  _FILE_ R4-1(MCC)
  _LINE_ R4-1(MCC)
  _STDC_ R4-1(MCC)
  _TIME_ R4-2(MCC)
Prefetch (trace) R3-171(XRAY), R3-200(XRAY)

Preinstallation steps I1-2(FLEX)

Preparing programs for debugging U2-1(XRAY)

command files U1-3(XRAY)

compiler debug option U2-1(XRAY)

data types U2-2(XRAY)

line numbers U2-2(XRAY)

module and procedure U2-2(XRAY)

storage class U2-2(XRAY)

/prepend option (VMS) R9-7(MCC)

Preprocessor R4-1(MCC)

executing only

/p option U3-29(MCC)

-E option U2-19(MCC), U2-16(CCC)

-P option U2-33(MCC), U2-30(CCC)

macros

defining on command line

/define option U3-20(MCC)

-D option U2-18(MCC), U2-14(CCC)

undefining

/undefine option U3-33(MCC)

-U option U2-36(MCC), U2-33(CCC)
output
saving comments
/preserve_comments option U3-29(MCC)
-C option U2-17(MCC), U2-14(CCC)
sending to standard output
-E option U2-19(MCC), U2-16(CCC)
symbols
(see under specific name)
Preprocessor directives
#define R4-10(MCC)
#else R4-13(MCC)
#else R4-14(MCC)
#error R4-15(MCC)
#if R4-17(MCC)
#endif R4-15(MCC)
#include R4-21(MCC)
#include R4-22(MCC)
#include R4-23(MCC)
#include asm R4-24(MCC)
#pragma R4-25(MCC)
#include info R4-26(MCC)
#include list R4-27(MCC)
#include macro R4-28(MCC)
#include option R4-29(MCC)
#include warn R4-30(MCC)
#undef R4-31(MCC)
#warning R4-32(MCC)
types R4-6(MCC)
conditional compilation R4-8(MCC)
line number control R4-6(MCC)
macro definition R4-6(MCC)
#pragma R4-8(MCC)
source file include R4-6(MCC)

Preprocessor macros
__OPTION_AVAIL R4-4(MCC)
__STR_CASE_CMP R4-5(MCC)
__STR_CMP R4-5(MCC)
Preprerequisite software RP-3(XRAY)

PREV CMD command key U2-30(XRAY), U3-14(XRAY)
Apollo support UB-5(XRAY)
HP support UD-7(XRAY)
IBM RS/6000 support UE-3(XRAY)
Motorola Delta Series support UF-4(XRAY)
PC support UG-4(XRAY)
Sun support UH-4(XRAY)
VT terminal support UI-4(XRAY)
Prev key (HP support) UD-5(XRAY)
PrevCmd button
X Window support UA-12(XRAY)
prevcmd button
SunView support UH-12(XRAY)
Primary expressions R3-7(MCC)
Print * button
X Window support UA-13(XRAY)
print * button
SunView support UH-13(XRAY)
Print button
X Window support UA-12(XRAY)
print button
SunView support UH-12(XRAY)
Print formatted output
PRINTF command R3-67(XRAY)
Printable argument
testing for R5-88(MCC), R5-90(MCC)
PRINTANALYSIS command R3-115(XRAY)
PRINTF command R3-118(XRAY), U4-16(XRAY)
printf function R5-14(MCC), R5-130(MCC)
relationship to vprintf R5-198(MCC)
removing unneeded I/O support R9-32(MCC)
PRINTPROFILE command R3-123(XRAY)
Prints formatted output
to Command viewport
PRINTF command R3-118(XRAY)
PRINTSYMBOLS command R3-127(XRAY), U4-15(XRAY)
errors U4-14(XRAY)
options U4-13(XRAY)
PRINTTYPE command R3-130(XRAY)
PRINTVALUE command R3-131(XRAY), U4-15(XRAY)
   assembly-level mode U2-22(XRAY)
   high-level mode U2-18(XRAY)
   with C++ RG-2(XRAY)
private
   keyword R4-5(CCC)
prnhelp program U2-33(XRAY)

Problems
   (see Error handling)
Procedure (@procedure) R2-21(XRAY)
Procedure (definition) RD-1(XRAY)
@procedure pseudo-register RA-2(XRAY), RF-4(XRAY)
Procedure, nested R2-23(XRAY)
Processor modes
   /cpu option U3-16(MCC)
   -p option U2-34(MCC), U2-30(CCC)
Processor signals S2-27(SUP)
Processor, specifying
   /cpu option U3-16(MCC)
   -p option U2-34(MCC), U2-30(CCC)
profile U1-1(FLEX)
PROFILE command R3-134(XRAY)
profile file
   (see UNIX start-up file)
Program counter R2-10(ASM)
Program identifiers R9-10(ASM)–R9-11(ASM)
Program identifiers, truncating
   /truncate_identifiers U3-12(MCC)
   -ut option U2-8(MCC), U2-37(MCC), U2-34(CCC)
Program sections R4-1(ASM)–R4-4(ASM), R9-2(ASM)–R9-5(ASM)
Program stack references R2-23(XRAY)
Program symbols R3-18(XRAY)
Program termination R5-22(MCC), R5-47(MCC), R5-48(MCC)
Program trace R6-19(XRAY)
   trace control R6-23(XRAY)
   SETSTATUS QUALIFY R6-23(XRAY)
   SETSTATUS TRACE R6-23(XRAY)
setstatus trigger R6-23(XRAY)
trace display R6-23(XRAY)
STATUS BUFFER R6-23(XRAY)

Programs U2-1(FLEX), UB-2(FLEX)
directory installed in U1-1(FLEX), UA-2(FLEX)
   specifying in UNIX start-up file U1-1(FLEX)
   error messages UC-11(FLEX)
   executing U1-2(FLEX)
   expiration date I3-3(FLEX)
   features versus programs I1-2(FLEX)
   host located on U2-10(FLEX)
   hosts, specifying U2-1(FLEX)
   installing I1-6(FLEX)
   networks, specifying U2-1(FLEX)
   testing I2-2(FLEX)
   version number I3-3(FLEX)

Prologue R1016(MCC)
   function
   generating code for
   prologue R1016(MCC)
Prologue, function R7-7(MCC), R4-2(CCC)
Promotion R5-3(CCC)
Prompts in install.sh I1-5(FLEX)
   with (.) I1-12(FLEX)
Prompts, macro (;) U2-26(XRAY)
protected keyword R4-6(CCC)
Prototyped functions R4-13(CCC)–R4-14(CCC)
   checking R4-14(CCC)
   declaration R4-13(CCC)
Prototypes, function
   arguments R3-25(MCC)
-Ps option U2-33(MCC), U2-30(CCC)
Pseudo-Ops
   (see Directives)
Pseudo-registers RA-1(XRAY)
   @addr RA-1(XRAY), RF-1(XRAY), RF-4(XRAY)
   @as RA-1(XRAY), RF-1(XRAY)
   @chip RA-1(XRAY), RF-1(XRAY)
@cycles RA-1(XRAY), RF-1(XRAY), RF-5(XRAY)
@entry RA-1(XRAY), RF-1(XRAY)
@exc RA-1(XRAY), RF-2(XRAY), RF-5(XRAY)
@file RA-1(XRAY), RF-2(XRAY)
@fpf RA-1(XRAY), RF-2(XRAY)
@fpu RA-1(XRAY), RF-2(XRAY), RF-5(XRAY), RF-10(XRAY)
@hlpc R2-21(XRAY), RA-1(XRAY), RF-3(XRAY)
@line_range RA-1(XRAY), RF-3(XRAY)
@module R2-21(XRAY), RA-1(XRAY), RF-3(XRAY)
@pi RA-1(XRAY), RF-3(XRAY)
@pysize RA-1(XRAY), RF-3(XRAY)
@port_addr R3-81(XRAY), R3-111(XRAY), RA-1(XRAY), RF-3(XRAY)
@port_size RA-1(XRAY), RF-3(XRAY)
@port_value R3-81(XRAY), R3-111(XRAY), RA-2(XRAY), RF-4(XRAY)
@procedure R2-21(XRAY), RA-2(XRAY), RF-4(XRAY)
@root R2-20(XRAY), RA-2(XRAY), RF-4(XRAY)
@wait_state RA-2(XRAY), RF-4(XRAY), RF-6(XRAY)

Psuedorandom number generator R5-140(MCC)
ptrdiff_t type R5-11(MCC)
Public
  keyword R3-5(MCC)
  storage class R3-5(MCC)
  variable names R11-1(MCC)
public
  keyword R4-5(CCC)
PUBLIC linker command R10-52(ASM)–R10-53(ASM)
Publications, related RP-3(CCC), UP-2(CCC)
Punctuation mark, testing for R5-91(MCC)
Pure specifier R3-14(CCC)
Pure virtual function R3-14(CCC), RC-4(CCC), UA-3(CCC)

putc function R5-12(MCC), R5-134(MCC)
relationship to fputc R5-64(MCC)
putchar function R5-12(MCC), R5-135(MCC)
putl function R5-136(MCC)
relationship to printf R5-79(MCC)
puts function R5-137(MCC)
putw function R5-138(MCC)
relationship to getw R5-81(MCC)

Q
-Q option U2-36(MCC), U2-33(CCC)
-Qe option U2-36(MCC), U2-33(CCC)
-Qi option U2-36(MCC), U2-33(CCC)
-Qo option U2-36(MCC), U2-33(CCC)
-Qs option U2-36(MCC), U2-33(CCC)
qsort function R5-139(MCC)
Qualified bus cycle (definition) R3-167(XRAY)
Qualified reference R2-15(XRAY), R2-19(XRAY)
definition RD-1(XRAY)
Qualify trace buffer
  STATUS QUALIFY command R3-188(XRAY)
Qualify trace information
  SETSTATUS QUALIFY command R3-167(XRAY)
Question mark-colon (?:) operator R2-25(CCC)
Questions and answers
  learning to use XRAY U4-1(XRAY)
  managing XRAY files U4-10(XRAY)
  using XRAY variables U4-13(XRAY)
Queued functions at program
termination R5-30(MCC)
quick assembler command line flag U2-10(ASM), U3-10(ASM)
Quick sort algorithm R5-139(MCC)
Quit button U2-52(CCC)
QUIT command R3-138(XRAY), U2-24(XRAY)
QUIT librarian command R13-10(ASM)
Quitting XRAY session
QUIT command R3-138(XRAY)
Quotient, computing R5-45(MCC), R5-99(MCC)
-Qw option U2-36(MCC), U2-33(CCC)

R
r assembler command line flag U2-10(ASM), U3-10(ASM)
-r librarian command line option U2-22(ASM)
-r linker command line option U2-14(ASM)
R1 key (Sun support) UH-3(XRAY)
R10 key (Sun support) UH-2(XRAY)
R12 key (Sun support) UH-2(XRAY)
R13 key (Sun support) UH-3(XRAY)
R14 key (Sun support) UH-2(XRAY)
R15 key (Sun support) UH-3(XRAY)
R3 key (Sun support) UH-3(XRAY)
R7 key (Sun support) UH-3(XRAY)
R8 key (Sun support) UH-2(XRAY)
R9 key (Sun support) UH-3(XRAY)
raise function R5-141(MCC)
Raising d1 to the power d2 R5-129(MCC)
RAM S1-8(SUP), S2-115(SUP)
RAM access
RAMACCESS command R3-139(XRAY)
RAM test S2-34(SUP), S2-36(SUP)
RAMACCESS S2-115(SUP)
RAMACCESS command R3-139(XRAY), U2-11(XRAY)
rand function R5-21(MCC), R5-140(MCC)
Random number generator
setting the seed R5-160(MCC)
Range, addresses R2-10(XRAY)
Raw trace command S2-57(SUP)
Raw trace display S1-15(SUP)
rbak command (Apollo) I1-7(FLEX)
rcp command U2-11(FLEX)
Read file into memory
SETSTATUS READ command R3-169(XRAY)
read function R5-142(MCC), R9-31(MCC), U2-52(MCC), U3-44(MCC), U2-44(CCC)
read routine R3-83(XRAY), U2-23(XRAY)
Read-after-write verify S2-161(SUP)
Reading a line from standard input R5-80(MCC)
Reading bytes from a file R5-142(MCC)
Reading distribution tape or disk I1-7(FLEX), I1-8(FLEX), I1-16(FLEX)
Reading from a file R5-66(MCC), R5-71(MCC)
Reading value from input port R4-18(XRAY)
Read-only memory accesses
ROMACCESS command R3-147(XRAY)
realloc function R5-21(MCC), R5-143(MCC), R9-12(MCC)
relationship to free R5-67(MCC)
Recall previous command
HISTORY command R3-74(XRAY)
Record count record (S5) RF-4(ASM)
Record debugger commands in a file
LOG command R3-91(XRAY)
Recording a debug session S3-20(SUP)
Recursive functions
setting breakpoints R2-24(XRAY)
using GO command R2-24(XRAY)
Redirect stderr
+E option U2-16(CCC)
Redraw screen U3-35(XRAY)
Reduce viewport size
ZOOM command R3-213(XRAY)
Redundant code elimination R102(MCC), R5-7(XRAY)
Redundant jump optimization R1014(MCC), R5-14(XRAY)
Redundant load and store optimization R5-8(XRAY)
Redundant servers I3-8(FLEX)
Redundant store elimination R1010(MCC)
Reentrant code R3-6(MCC)
generating R9-10(MCC)
Reentrant functions R5-20(MCC), R9-10(MCC)
Reference R2-22(XRAY)
qualified R2-19(XRAY)
REFERENCE linker command line
option U3-14(ASM)
Reference type R2-10(CCC)
Refresh software mask S2-122(SUP)
Refreshing software address space S2-120(SUP)
Refreshing software addresses S2-119(SUP)
REG
Status Line viewport U3-24(XRAY)
REG assembler directive R5-63(ASM)
Register
keyword R3-5(MCC)
reserving a register R9-33(MCC)
reserving for special purposes R9-38(MCC)
storage class R3-5(MCC)
use R7-5(MCC)
Register (CPU) S3-28(SUP)
Register allocation R5-4(XRAY)
Register coloring
/optimize=register option U3-26(MCC)
-OR option U2-32(MCC), U2-29(CCC)
Register coloring optimization R5-4(XRAY)
Register contents
SETREG command R3-158(XRAY)
Register names (@) R3-158(XRAY)
Register optimization R5-4(XRAY)
Register restoration S2-123(SUP)
Register storage class R3-4(MCC)
Register support S1-9(SUP)
Register variables R2-16(XRAY)
Register-relative address mode U2-27(MCC), U2-24(CCC)
(see also Position-independent data)
Register-relative addressing R9-25(MCC)
Registers R3-3(ASM)–R3-6(ASM)
I/O device U2-31(MCC), U3-28(MCC), U2-28(CCC)
reserving
/reserve option U3-14(MCC)
-Kh option U2-25(MCC), U2-22(CCC)
Registers viewport U2-20(XRAY), U3-2(XRAY), U3-18(XRAY)
rel32 assembler command line flag U2-11(ASM), U3-11(ASM)
Related publications RP-3(XRAY)
Relative addressing R3-24(ASM)–R3-33(ASM)
Relative path settings U1-3(FLEX)
RELOAD command R3-141(XRAY)
Reload program code and data
RELOAD command R3-141(XRAY)
Relocatable expressions R3-24(ASM), R4-7(ASM)–R4-8(ASM)
Relocatable sections R9-3(ASM)
Relocatable symbols R2-9(ASM), R4-7(ASM)
Relocation flags, assembler R8-2(ASM)
Relocation types R9-8(ASM)
Remainder
computing R5-45(MCC), R5-99(MCC)
Remote installation U1-8(FLEX)
remove function R5-144(MCC)
Removing screens and viewports U3-33(XRAY)
VCLOSE command U3-33(XRAY)
Removing viewport
VCLOSE command R3-205(XRAY)
REPEAT . . . UNTIL assembler directive R7-12(ASM)
REPLACE librarian command U2-22(ASM), R13-16(ASM)
REPLACE librarian command line option U3-19(ASM)
REPT assembler directive R5-64(ASM)
RESADD linker command R10-54(ASM)
RESERVE line U3-5(FLEX)
/reserve option (VMS) R7-5(MCC), R9-34(MCC)
Reserved licenses UB-3(FLEX)
Reserved symbols R2-8(ASM)–R2-9(ASM), RA-1(XRAY)
NARG R2-9(ASM), R6-4(ASM), R8-3(ASM)
Reserved words RA-1(XRAY)
Reserving a register
/reserve option  U3-14(MCC)
-Kh option  U2-25(MCC), U2-22(CCC)
Reserving licenses  U2-11(FLEX)
Reserving licenses for specific users  U3-5(FLEX)
Reset
  restoring registers  S2-123(SUP)
RESET command  R3-142(XRAY)
Reset pulses  S2-52(SUP)
Reseting microprocessor
  RESET command  R3-142(XRAY)
Resetting start address
  RESTART command  R3-144(XRAY)
Resize viewport
  VOPEN command  R3-208(XRAY)
Resizing viewports  U2-28(XRAY), U4-7(XRAY)
RESMEM linker command  R10-55(ASM)
RESTART command  R3-144(XRAY), U2-20(XRAY)
Restarting script  I2-5(FLEX)
Restore
  not supported  S1-8(SUP)
RESTORE assembler directive  R5-65(ASM)
RESTORE command  R3-145(XRAY)
Restore memory and registers
  RESTORE command  R3-145(XRAY)
Restricted debugging information
  /debug=restricted option  U3-20(MCC)
  -Gr option  U2-21(MCC), U2-18(CCC)
RET instruction, using for interrupt procedures
  /ireturn=subroutine  U3-22(MCC)
  -Kr option  U2-26(MCC), U2-23(CCC)
Return codes
  UNIX/DOS  U2-28(ASM)
  VAX/VMS  U3-27(ASM)
return statement  R3-22(MCC)
RETURN statement in macros  R4-7(XRAY)
Returning a typed value  R9-5(MCC)
rewind function  R5-145(MCC)
Rewind input files
  RIN command  R3-146(XRAY)
Rewind output files
  ROUT command  R3-149(XRAY)
RFS command  S2-117(SUP)
RFSADR command  S2-119(SUP)
RFSASP command  S2-120(SUP)
RFSMSK command  S2-122(SUP)
RIN command  R3-146(XRAY)
RIRR command  S2-123(SUP)
ROM  S1-8(SUP), S2-124(SUP)
ROMACCESS  S2-124(SUP)
ROMACCESS command  R3-147(XRAY), U2-11(XRAY)
ROM-based systems  R9-43(MCC)
Root (@)  R2-20(XRAY)
Root (definition)  RD-1(XRAY)
Root names  R2-20(XRAY)
@root pseudo-register  R2-20(XRAY), RA-2(XRAY), RF-4(XRAY)
ROUT command  R3-149(XRAY)
Router  UB-2(FLEX)
Routines, user-modified  R9-29(MCC)
_RS6000 preprocessor symbol  R4-3(MCC)
rsh command  11-S(FLEX), 12-7(FLEX)
RTE instruction, using for interrupt procedures
  -nkr option  U2-26(MCC), U2-23(CCC)
RTS instruction, using for interrupt procedures
  -Kr option  U2-26(MCC), U2-23(CCC)
Run button  U2-49(CCC)
run command  U2-52(CCC)
Run poll  S2-126(SUP)
RUN_TIME  S2-127(SUP)
Running test scripts  I2-2(FLEX)
Run-time organization  R8-1(MCC)

S
s assembler command line flag  U2-11(ASM), U3-11(ASM)
-S option  U2-36(MCC), U2-33(CCC)
Sample program sieve.c  U2-1(XRAY)
Save
  not supported  S1-8(SUP)
SAVE assembler directive R5-66(ASM)
SAVE command R3-150(XRAY)
SAVE librarian command R13-17(ASM)
Saving
  command and data to a file
    JOURNAL command R3-86(XRAY)
  debugger commands in a file
    LOG command R3-91(XRAY)
  macros R4-9(XRAY)
  memory and registers to file
    SAVE command R3-150(XRAY)
Saving commands U4-7(XRAY)
  JOURNAL command U4-7(XRAY)
  LOG command U4-7(XRAY)
sbrk function R5-146(MCC), U2-52(MCC),
U3-44(MCC), U2-44(CCC)
Scalar (definition) RD-1(XRAY)
Scalar data types R6-4(MCC)
scanf function R5-14(MCC), R5-147(MCC)
  relationship to sscanf R5-161(MCC)
Scheduling, instructions
  /optimize=reorder option U3-27(MCC)
  -Or option U2-32(MCC), U2-29(CCC)
  _SCO preprocessor symbol R4-3(MCC)
Scope S3-22(SUP)
Scope button
  X Window support UA-13(XRAY)
scope button
  SunView support UH-13(XRAY)
SCOPE command R3-152(XRAY), U3-11(XRAY), U4-2(XRAY), U4-3(XRAY),
U4-12(XRAY), U4-14(XRAY)
Scope Loops S3-49(SUP)
Scoping rules R2-19(XRAY)
Scratch area address S2-128(SUP)
SCRATCH command S2-128(SUP)
scratch.xry file UH-15(XRAY)
Screen (definition) RD-1(XRAY)
Screen button
  X Window support UA-13(XRAY)
screen button
  SunView support UH-13(XRAY)
Screen commands U3-34(XRAY)
Screen refresh U3-35(XRAY)
Screen, activating
  VSCREEN command R3-210(XRAY)
Screens U1-5(XRAY), U3-1(XRAY)
  assembly-level screen U3-2(XRAY)
  defining U3-32(XRAY)
  deleting U3-33(XRAY)
  high-level U2-14(XRAY), U3-1(XRAY)
  I/O U2-25(XRAY)
  removing U3-33(XRAY)
  standard I/O screen U3-4(XRAY)
Scrolling U3-33(XRAY), U4-1(XRAY)
Scrolling viewports S3-18(SUP)
SEARCH S1-8(SUP)
SEARCH command R3-154(XRAY)
  size qualifiers R3-3(XRAY)
Search path, setting
  SETSTATUS ENVIRONMENT command R3-160(XRAY)
Search paths
  (see also Environment variables)
  nonstandard #include files
    /ipath option U3-23(MCC)
    -I option U2-22(MCC), U2-19(CCC)
  standard #include files
    /spath option U3-32(MCC)
    -J option U2-23(MCC), U2-20(CCC)
Searching
  for character in memory R4-23(XRAY)
  for character in string R4-30(XRAY)
  for next occurrence of string
    NEXT command R3-97(XRAY)
  for string
    FIND command R3-64(XRAY)
  for value
    SEARCH command R3-154(XRAY)
Searching for a character in memory R5-117(MCC)
SECT assembler directive R5-67(ASM)–R5-68(ASM)
SECT linker command R10-56(ASM)
Section alignment R9-5(ASM)
SECTION assembler directive R5-67(ASM)–R5-68(ASM)
Section names, specifying U2-28(MCC), U2-25(CCC)

Sections
  absolute R3-21(ASM), R9-3(ASM)
  common R9-4(ASM)
  long R9-5(ASM)
  noncommon R9-4(ASM)
  program R4-1(ASM)-R4-4(ASM), R9-2(ASM)-R9-5(ASM)
  relocatable R3-22(ASM), R9-3(ASM)
  short R9-4(ASM)
  type R9-5(ASM)-R9-6(ASM)

SECTSIZE linker command R10-57(ASM)
Seed for random number generator R5-160(MCC)
SEEK_CUR macro R5-15(MCC), R5-72(MCC)
SEEK_END macro R5-15(MCC), R5-72(MCC)
SEEK_SET macro R5-15(MCC), R5-72(MCC)
Serial number of distribution I1-3(FLEX), I1-10(FLEX), U1-5(FLEX)
SERVER line U2-9(FLEX)
  required number U2-10(FLEX)

Servers
  (see License servers)
  daemons
    (see License server daemons)
  failure I1-14(FLEX)
  more than one in installation I2-7(FLEX)
  specifying during install I1-11(FLEX), I1-12(FLEX)

Session control commands R3-6(XRAY)
  HOST R3-6(XRAY), R3-75(XRAY)
  LOAD R3-6(XRAY), R3-89(XRAY)
  QUIT R3-6(XRAY), R3-138(XRAY)
  RELOAD R3-6(XRAY), R3-141(XRAY)
  RESTORE R3-6(XRAY), R3-145(XRAY)
  SAVE R3-6(XRAY), R3-150(XRAY)

SET assembler directive R5-69(ASM)
Set command S6-9(SUP)
set command (csh) I2-1(FLEX)
Set cursor position in viewport
  VSETC command R3-212(XRAY)
setbuf function R5-150(MCC)
setenv command (csh) I1-15(FLEX), I2-1(FLEX), U1-1(FLEX), U1-3(FLEX)
setjmp function R5-151(MCC)
  relationship to longjmp R5-107(MCC)
setjmp.h include file R5-9(MCC)
setlocale function R5-152(MCC)
  relationship to localeconv function R5-103(MCC)

SETMEM command R3-156(XRAY), U4-16(XRAY)
  size qualifiers R3-3(XRAY)
SETREG command R3-158(XRAY)
SETSTATUS S1-8(SUP)
SETSTATUS command UA-13(XRAY)
SETSTATUS DIR command R3-159(XRAY)
SETSTATUS ENVIRONMENT command R3-160(XRAY)
SETSTATUS EVENT command R3-161(XRAY)
SETSTATUS QUALIFY command R3-167(XRAY)
SETSTATUS READ command R3-169(XRAY)
SETSTATUS TRACE command R3-170(XRAY)
SETSTATUS TRIGGER command R3-173(XRAY)
SETSTATUS VERIFY command R3-177(XRAY)
SETSTATUS WRITE command R3-178(XRAY)

Setting a breakpoint U4-5(XRAY)

Setting breakpoints
  (see Breakpoints)
  BREAKACCESS command R3-25(XRAY)
  BREAKCOMPLEX command R3-28(XRAY)
  BREAKINSTRUCTION command S2-12(SUP), R3-30(XRAY)
  BREAKREAD command S2-16(SUP), R3-32(XRAY)
  BREAKWRITE command S2-20(SUP), R3-35(XRAY)

Setting colors UA-3(XRAY)
Setting instruction breakpoint
(see BREAKINSTRUCTION command)

Setting sign of char variables
/unsignedchar option U3-22(MCC)
-Ku option U2-27(MCC), U2-23(CCC)

Setting temporary breakpoints
(see Breakpoints)

Setting the value of characters in memory R5-122(MCC)

Setting value of characters in memory R4-26(XRAY)

Settings, current
saving to file
SAVE command R3-150(XRAY)

Setup
user U1-1(FLEX)
setvbuf function R5-154(MCC)
Severity of errors RC-2(MCC)

Shared
program R9-35(MCC)
Sharing variables R3-6(MCC)

Shift-Ins key (Apollo support) UB-3(XRAY)

Short circuit optimization R5-8(XRAY)
Short integer
reading from a stream R5-81(MCC)
writing to stream R5-138(MCC)

Short integers R7-1(MCC)

short type R3-2(MCC), R6-3(MCC), R6-11(MCC)

Short/long displacement optimizations R1015(MCC), R5-15(XRAY)

Short-circuit evaluation R3-10(MCC)

Short-circuit optimization R1011(MCC)
SHOW command R3-179(XRAY), U2-33(XRAY)

SIA command S2-129(SUP)
SIG_DFL R5-155(MCC)
SIG_ERR R5-155(MCC)
SIG_IGN R5-155(MCC)
SIGFPE R5-155(MCC)

signal function R5-155(MCC)

Signal handler, establishing R5-155(MCC)
signal.h include file R5-10(MCC)

Signals R5-10(MCC)
raise R5-141(MCC)
sending R5-141(MCC)
signal R5-155(MCC)

Signed data
keywords R4-6(CCC)

signed keyword R4-6(CCC)

signed types R3-1(MCC)
signed int type R3-2(MCC)
signed long type R3-2(MCC)
signed short type R3-2(MCC)

Simple breakpoints S4-3(SUP)

Simple relocatable expression R4-8(ASM)

Simple straddling R6-24(MCC)

Simulated I/O S7-1(SUP), U4-9(XRAY)
character input using XICE S7-1(SUP)
character input using XRAY S7-5(SUP)
character output using XHS S7-6(SUP)
character output using XICE S7-3(SUP)
character output using XRAY S7-6(SUP)

_simulated_input R3-83(XRAY)
_simulated_output R3-112(XRAY)

Simulating an interrupt
INTERRUPT command R3-84(XRAY)

sin function R5-156(MCC), U2-39(MCC), U3-25(MCC), U2-37(CCC)

Sine of a number, computing R5-156(MCC)

Single inheritance R3-10(CCC), RC-2(CCC), UA-2(CCC)

Single line assembler S2-2(SUP), S3-30(SUP)

Single quotation marks R2-4(MCC)

Single-precision format R6-5(MCC)

Single-stepping U1-4(XRAY), U2-17(XRAY)
(see also STEP command, STEPOVER command)

STEP command R3-193(XRAY)
STEPOVER command R3-195(XRAY)

sinh function R5-157(MCC), U2-39(MCC), U3-25(MCC), U2-37(CCC)

Size
aggregates R6-26(MCC)
data types R6-3(MCC)
unpacked structure R6-31(MCC)
SIZE command S2-130(SUP)
Size of file names R3-1(XRAY)
Size of viewport
   ZOOM command R3-213(XRAY)
Size optimization
   /optimize=size option U3-27(MCC)
   -Os option U2-32(MCC), U2-29(CCC)
size_t type R5-11(MCC), R5-17(MCC)
   bsearch function R5-34(MCC)
   calloc function R5-35(MCC)
   fwrite function R5-66(MCC)
   malloc function R5-111(MCC)
   mbfunc R5-113(MCC)
   mbstowcs function R5-114(MCC)
   mbtowc function R5-115(MCC)
   memccpy function R5-116(MCC)
   memchr function R5-117(MCC)
   memcmp function R5-118(MCC)
   memcmp function R5-119(MCC)
   memcpy function R5-120(MCC)
   memmove function R5-121(MCC)
   memset function R5-122(MCC)
   offsetof function R5-125(MCC)
   realloc function R5-139(MCC)
   setvbuf function R5-154(MCC)
   strcmp function R5-166(MCC)
   strlen function R5-167(MCC)
   strcspn function R5-168(MCC)
   strncat function R5-170(MCC)
   strxfrm function R5-181(MCC)
   wctomb function R5-200(MCC)
   zalloc function R5-203(MCC)
_SIZEOF_CHAR preprocessor symbol R4-2(MCC)
_SIZEOF_DOUBLE preprocessor symbol R4-2(MCC)
_SIZEOF_FLOAT preprocessor symbol R4-2(MCC)
_SIZEOF_INT preprocessor symbol R4-2(MCC)
_SIZEOF_LONG preprocessor symbol R4-2(MCC)
_SIZEOF_LONG_DOUBLE preprocessor symbol R4-2(MCC)
_SIZEOF_PTR preprocessor symbol R4-2(MCC)
_SIZEOF_SHORT preprocessor symbol R4-2(MCC)
sizeof unary operator R3-9(MCC)
Skipping part A I1-9(FLEX)
Skipping part B I1-16(FLEX)
SLO command S2-132(SUP)
Software
   breakpoints S4-6(SUP)
Software address refresh S2-119(SUP)
Software address space refresh S2-120(SUP)
Software breakpoint trap number S2-82(SUP)
Software mask refresh S2-122(SUP)
Software performance analysis S1-19(SUP)
Software refresh control S2-117(SUP)
Solving problems
   (see Error handling)
SORDER linker command R10-58(ASM)–R10-59(ASM)
Sorting table of data R5-139(MCC)
Source code
   displaying
      LIST command R3-87(XRAY)
   intermixed with assembly
      LINES option R3-107(XRAY)
Source file
   assembly U4-2(XRAY)
   high-level U4-2(XRAY)
   location U4-10(XRAY)
Source file include directive (#include) R4-6(MCC)
Source file problems, detecting
   /extra_checks U3-23(MCC)
   -v option U2-38(MCC), U2-35(CCC)
Source patching R4-12(XRAY)
Source-level screen  S1-5(SUP)
SPACE command  S2-134(SUP)
Space, testing for a  R5-92(MCC)
SPC assembler directive  R5-70(ASM)
Special characters
  librarian  R13-1(ASM)
  linker  R10-1(ASM)
Special interrupt vector  S2-129(SUP)
speed  S1-8(SUP)
sprintf function  R5-20(MCC), R5-158(MCC), R9-11(MCC)

  relationship to vsprintf  R5-199(MCC)
sqrt function  R5-159(MCC), U2-39(MCC), U3-25(MCC), U2-37(CCC)
Square root of a number, computing  R5-159(MCC)
srand function  R5-21(MCC), R5-160(MCC)
S-record format  RF-1(ASM)–RF-7(ASM)
  data record (S1)  RF-2(ASM)
  header record (S0)  RF-2(ASM)
  module record  RF-1(ASM)
  record count record (S5)  RF-4(ASM)
  symbol record  RF-1(ASM)
srun command  U2-52(CCC)
sscanf function  R5-20(MCC), R5-161(MCC), R9-11(MCC)
Stack  R5-4(XRAY)
  16-bit quantities
    /min_push_size=2 option  U3-22(MCC)
    -K2 option  U2-24(MCC), U2-21(CCC)
  32-bit quantities
    /min_push_size=4 option  U3-22(MCC)
    -K4 option  U2-24(MCC), U2-21(CCC)
adjust instructions  R5-18(XRAY)
disabling frame sharing
  /frames  U3-21(MCC)
  -Kf option  U2-25(MCC), U2-22(CCC)
explicit references to  R2-23(XRAY)
frame (definition)  RD-2(XRAY)
frames  R7-6(MCC)
imPLICIT references to  R2-23(XRAY)
level (@)  R2-2(XRAY)
moving down levels
  DOWN command  R3-56(XRAY)
moving up levels
  UP command  R3-202(XRAY)
pointer  R7-5(MCC)
popping
  /optimize=combine_pops option  U3-25(MCC)
    -Oc option  U2-31(MCC), U2-28(CCC)
Stack level
  @ symbol  R3-60(XRAY)
Stack levels
  displaying  U4-4(XRAY)
  jumping  U4-4(XRAY)
Stack viewport  U2-20(XRAY), U3-2(XRAY), U3-21(XRAY)
Stage directory  I1-6(FLEX)
files in  I1-8(FLEX), I1-9(FLEX)
Standard C operators  RC-1(XRAY)
Standard definition
  macros  R5-11(MCC)
  types
    ptdiff_t  R5-11(MCC)
    size_t  R5-11(MCC)
    wchar_t  R5-11(MCC)
Standard error
  (see stderr)
getting error messages  R5-128(MCC)
Standard I/O screen  U3-4(XRAY)
  (see I/O screen)
STD  R3-80(XRAY), R3-110(XRAY)
Standard I/O viewport, display
  STDIO option  R3-108(XRAY)
Standard input
  (see stdin)
reading a character  R5-78(MCC)
Standard output
  (see stdout)
getting a string  R5-137(MCC)
writing a character  R5-135(MCC)
Start button
  X Window support  UA-13(XRAY)
START linker command  R10-60(ASM), R10-63(ASM)

Start program execution
GO command  R3-69(XRAY)

Starting address  R9-6(ASM)
linker commands
START  R10-63(ASM)
RESTART command  R3-144(XRAY)

Starting debugger  S1-2(SUP)
Starting XICE  S1-4(SUP)

.STARTOF. operator  R2-18(ASM)

Start-up
license server daemons  I2-2(FLEX), I2-3(FLEX), U1-2(FLEX), U2-6(FLEX)
manual  I2-7(FLEX)
mllicense.daemon  I2-3(FLEX), U2-3(FLEX)

STARTUP command  R3-180(XRAY), U4-3(XRAY), U4-9(XRAY)

Start-up options saved
STARTUP command  R3-180(XRAY)

Start-up routine  U2-53(MCC), U3-44(MCC), U2-45(CCC)

startup.xry file  U2-10(XRAY), U4-9(XRAY)

Statement
assembler  R2-1(ASM)–R2-4(ASM)

Statement debugging  R2-11(XRAY)

Statements  R3-18(MCC)
break  R3-19(MCC)
compound  R3-19(MCC)
continue  R3-19(MCC)
do-while  R3-20(MCC)
expression  R3-19(MCC)
for  R3-20(MCC)
goto  R3-21(MCC)
if  R3-21(MCC)
if-else  R3-22(MCC)
labeled  R3-21(MCC)
return  R3-22(MCC)
switch  R3-23(MCC)
while  R3-24(MCC)

Static
keyword  R3-5(MCC)
storage class  R3-5(MCC)
variables  R6-42(MCC)
names  R11-2(MCC)

Static constructor  R2-11(CCC)

Static data members  RC-4(CCC), UA-4(CCC)

Static destructor  R2-12(CCC)

Static member functions  RC-4(CCC), UA-4(CCC)

Static members  R2-15(CCC)
class  R2-15(CCC)
data  R2-15(CCC)

Static variables  R2-15(XRAY)

STATUS  S1-8(SUP)

Status
license server  U1-4(FLEX), U3-7(FLEX)
license server daemons  U1-4(FLEX), U3-7(FLEX)
licenses  U1-4(FLEX), U3-2(FLEX), U3-6(FLEX)

STATUS BUFFER command  R3-183(XRAY)

Status button
X Window support  UA-13(XRAY)

status button
SunView support  UH-13(XRAY)

STATUS command  R3-182(XRAY), U3-28(XRAY)

STATUS EVENT command  R3-187(XRAY)

Status Line viewport  U2-15(XRAY), U3-1(XRAY), U3-22(XRAY)
assembly-level  U3-2(XRAY)
break field (BREAK #)  U3-24(XRAY)
CPU field (CPU)  U3-24(XRAY)
help field (HELP)  U3-24(XRAY)
info field  U3-24(XRAY)
module field (MODULE)  U3-24(XRAY)
OPTION CPU command  U3-24(XRAY)
REG  U3-24(XRAY)
status field (Command)  U3-23(XRAY)
status field (Define)  U3-23(XRAY)
status field (Execute)  U3-23(XRAY)
status field (Include)  U3-23(XRAY)
status field (Input)  U3-23(XRAY)
status field (Macro) U3-23(XRAY)
status field (Output) U3-23(XRAY)
status field (Reading) U3-23(XRAY)
status field (View) U3-23(XRAY)
status field (Working) U3-23(XRAY)
version field (MRI) U3-24(XRAY)
VOPEN command U3-22(XRAY)
WARNING U3-24(XRAY)

Status mnemonics supported S2-72(SUP)
Status of XRAY or target
STATUS command R3-182(XRAY)
STATUS QUALIFY command R3-188(XRAY)
STATUS TRACE command R3-189(XRAY)
STATUS TRIGGER command R3-191(XRAY)
STD
standard I/O screen R3-80(XRAY), R3-110(XRAY)

stdarg.h include file R5-10(MCC)
__STDC__ preprocessor symbol R4-1(MCC), U2-43(MCC), U3-36(MCC)
__STDC__ preprocessor symbol U2-8(MCC), U2-16(MCC), U3-11(MCC), U2-12(CCC)

stddef.h include file R5-11(MCC)

stderr R5-12(MCC), R5-13(MCC), R5-15(MCC), U3-2(FLEX), U3-3(FLEX), UA-1(FLEX)
attaching streams to other files R5-69(MCC)
directing diagnostic messages
/diagnostics_to=stderr option U3-21(MCC)
-Feo option U2-20(MCC)
mlicense messages U3-2(FLEX), U3-3(FLEX)
mlicense.daemon messages U2-8(FLEX), U3-1(FLEX), UA-2(FLEX)
write string to R5-137(MCC)

STEP button
SunView support UH-13(XRAY)
X Window support UA-13(XRAY)

Step button
X Window support UA-13(XRAY)

step button
SunView support UH-13(XRAY)

STEP command R3-193(XRAY), U1-4(XRAY), U4-3(XRAY)
assembly-level mode U2-22(XRAY)
high-level mode U2-16(XRAY)

STEP command key U2-16(XRAY)
Apollo support UB-5(XRAY)
DECstation support UC-4(XRAY)
HP support UD-6(XRAY)
IBM RS/6000 support UE-4(XRAY)
Motorola Delta Series support UF-4(XRAY)
PC support UG-4(XRAY)
Sun support UH-4(XRAY)
VT terminal support UI-4(XRAY)

stepo button
SunView support UH-13(XRAY)

stdio.h include file R5-2(MCC), R5-12(MCC), R5-14(MCC), R5-20(MCC), R5-150(MCC), R5-193(MCC), R9-11(MCC)

stdlib.h include file R5-2(MCC), R5-16(MCC), R5-17(MCC)

stdout R5-12(MCC), R5-14(MCC), R5-15(MCC)
attaching streams to other files R5-69(MCC)
directing diagnostic messages
/diagnostics_to=message option U3-21(MCC)
-Feo option U2-20(MCC)
mlicense messages U3-2(FLEX), U3-3(FLEX)
mlicense.daemon messages U2-8(FLEX), U3-1(FLEX), UA-2(FLEX)
write string to R5-137(MCC)

STEP button
SunView support UH-13(XRAY)
X Window support UA-13(XRAY)

Step button
X Window support UA-13(XRAY)

step button
SunView support UH-13(XRAY)

STEP command R3-193(XRAY), U1-4(XRAY), U4-3(XRAY)
assembly-level mode U2-22(XRAY)
high-level mode U2-16(XRAY)

STEP command key U2-16(XRAY)
Apollo support UB-5(XRAY)
DECstation support UC-4(XRAY)
HP support UD-6(XRAY)
IBM RS/6000 support UE-4(XRAY)
Motorola Delta Series support UF-4(XRAY)
PC support UG-4(XRAY)
Sun support UH-4(XRAY)
VT terminal support UI-4(XRAY)

stepo button
SunView support UH-13(XRAY)
STEPOVER button
Sun View support UH-13(XRAY)
X Window support UA-13(XRAY)
STEPOVER command  R3-195(XRAY), U1-4(XRAY), U2-19(XRAY), U4-3(XRAY)
STEPOVER command key
Apollo support UB-4(XRAY), UC-4(XRAY)
DECstation support UC-4(XRAY)
HP support UD-7(XRAY)
IBM RS/6000 support UE-4(XRAY)
Motorola Delta Series support UF-4(XRAY)
PC support UG-4(XRAY)
Sun support UH-4(XRAY)
VT terminal support UI-3(XRAY)
StepOvr button
X Window support UA-13(XRAY)
Stepping U4-3(XRAY)
(see also Single-stepping, STEP command, STEPOVER command)
Stepping speed
SPEED option R3-108(XRAY)
Step-through interrupts S2-136(SUP)
STI command S2-136(SUP)
stop button
Sun View support UH-13(XRAY)
Stop execution U4-6(XRAY)
Stop monitoring expressions
NOMONITOR command R3-102(XRAY)
Stop XRAY session
QUIT command R3-138(XRAY)
Storage
allocation of data types R6-3(MCC)
classes R3-4(MCC)
keywords R3-4(MCC)
dynamically allocated
changing the size R5-143(MCC)
layout R6-1(MCC)
Storage classes R2-15(XRAY), R4-8(CCC)
(see also Symbolic references, storage classes)
Straddling bits R6-23(MCC)
strcat function R5-162(MCC)
strcat macro R4-29(XRAY)
strstr function R5-163(MCC)
strstr macro R4-30(XRAY)
strcmp function R5-164(MCC)
relationship to strcoll R5-165(MCC)
strcmp macro R4-31(XRAY)
strcoll function R5-165(MCC)
relationship to strxfrm R5-181(MCC)
strcpy function R5-166(MCC)
strcpy macro R4-32(XRAY)
strcspn function R5-167(MCC)
Stream
association to a file R5-68(MCC)
reading a long integer R5-79(MCC)
reading a short integer R5-81(MCC)
writing a long integer R5-136(MCC)
writing a short integer R5-138(MCC)
Strength reduction R102(MCC)
optimization R109(MCC)
Strength reduction optimization R5-5(XRAY), R5-11(XRAY)
strerror function R5-168(MCC)
striccp macro R4-33(XRAY)
String R6-16(MCC)
comparing two strings R5-164(MCC), R5-165(MCC), R5-171(MCC)
concatenating two strings R5-162(MCC), R5-170(MCC)
conversion to unsigned long integer R5-180(MCC)
copying one string to another string R5-166(MCC)
copying string R5-172(MCC)
finding first occurrence of same character from two strings R5-173(MCC)
finding first occurrence of same sequence of characters from two strings R5-176(MCC)
finding last occurrence of a character in a string R5-174(MCC)
first occurrence of a character R5-163(MCC)
formatted string conversion R5-161(MCC)
formatting R5-158(MCC), R5-199(MCC)
functions R5-17(MCC)–R5-18(MCC), R5-162(MCC)–R5-181(MCC)
initial segment length R5-167(MCC), R5-175(MCC)
length R5-169(MCC)
literals R2-6(MCC)
locating tokens R5-178(MCC)
multibyte
  conversion from wide character R5-201(MCC)
  conversion from wide character string R5-200(MCC)
  conversion to wide character string R5-114(MCC)
reading from a file R5-57(MCC)
transformation R5-181(MCC)
writing a string to a file R5-65(MCC)
writing to standard output R5-137(MCC)

strings.h include file R5-17(MCC)

Strings
  comparing R4-31(XRAY), R4-33(XRAY), R4-35(XRAY)
  comparison R4-31(XRAY)
  concatenating R4-29(XRAY)
  copying R4-32(XRAY)
  expression string R3-62(XRAY), R3-154(XRAY), R3-156(XRAY), R3-197(XRAY)
  locating characters R4-30(XRAY)
  searching
    NEXT command R3-97(XRAY)
strings section R8-4(MCC), U2-16(MCC), U3-15(MCC), U2-13(CCC)
naming
  /rename options U3-31(MCC)
  -NS option U2-29(MCC), U2-26(CCC)
specifying address mode
  /strings_addr_as options U3-16(MCC)
  -as options U2-17(MCC), U2-14(CCC)

strlen function R5-169(MCC)

strlen macro R4-34(XRAY)
strncat function R5-170(MCC)
strncpy function R5-171(MCC)
strncpy macro R4-35(XRAY)
strncpy function R5-172(MCC)
strpbrk function R5-173(MCC)
strchr function R5-174(MCC)
strspn function R5-175(MCC)
strstr function R5-176(MCC)
strtol function R5-177(MCC), R9-12(MCC)
strtok function R5-21(MCC), R5-178(MCC)
strtol function R5-179(MCC), R9-12(MCC)
stroull function R5-180(MCC), R9-12(MCC)

Structure
  return value R7-4(MCC)
Structure alignment R6-29(MCC)
  68000/68020 R6-36(MCC)
Structure member
determining offset R5-125(MCC)
Structure members, aligning
  /align option U3-11(MCC)
  /ssmultiple option U3-11(MCC)
  -Z options U2-40(MCC), U2-37(CCC)
Structure padding R6-29(MCC)
Structure size
  -Zm option U2-40(MCC), U2-38(CCC)
Structured control directive code (+) R7-14(ASM)
Structured control directives
  (see Directives, assembler)
Structured control expressions R7-1(ASM)–R7-3(ASM)
Structured directives, nesting R7-14(ASM)
Structures R6-17(MCC)
  iconv R5-100(MCC)
  packed R6-27(MCC)
  packing R6-39(MCC)
  size R6-31(MCC)
  size, 68000 R6-31(MCC)
  tm R5-19(MCC)
**Structures, packed**

- `/define option` U3-21(MCC)
- `-D option` U2-18(MCC), U2-15(CCC)

- `/define option` R5-181(MCC)

- `strxfrm function` R5-6(XRAY)

- `Stub` RC-4(CCC), UA-4(CCC)

- `Stubs` R3-15(CCC)

- `Subexpression optimization` R5-6(XRAY)

- `Substring operation` R2-24(CCC)

- `Subsection` R9-3(ASM)

- `Suffix, C++ files` +z option U2-38(CCC)

**Summary message**

- `suppression`
  - `/suppress=summary` U3-30(MCC)
  - `-Qs option` U2-36(MCC), U2-33(CCC)

**Summary of command line options**

- U2-11(MCC), U3-5(MCC), U2-7(CCC)

**Sun**

- `host-specific information` UH-1(XRAY)

**Sun arch command**

- IA-1(FLEX)

**Sun support**

- `MOVE TO BOTTOM` control key UH-3(XRAY)

- `MOVE TO TOP` control key UH-3(XRAY)

**_SUN3 preprocessor symbol**

- R4-3(MCC)

**_SUN4 preprocessor symbol**

- R4-3(MCC)

**Suns**

- `message display` U1-3(FLEX), U2-8(FLEX), U3-1(FLEX), UA-2(FLEX)

- `mixed architectures` U2-10(FLEX)

**Supported operating systems**

- IP-1(MCC), UP1(MCC), IP-1(XRAY)

**Suppressing executable file**

- `-c option` U2-18(MCC), U2-14(CCC)

**Swab function**

- R5-182(MCC)

**Swapping bytes in memory**

- R5-182(MCC)

**switch statement**

- R3-23(MCC)

**switch statements**

- R5-7(XRAY)

**Symbol**

- `name size` linker RG-3(ASM)

**Symbol (definition)**

- R3-4(XRAY)

**Symbol commands**

- R3-9(XRAY)
  
  - ADD R3-9(XRAY), R3-18(XRAY)
  
  - CONTEXT R3-9(XRAY), R3-43(XRAY)
  
  - DELETE R3-9(XRAY), R3-48(XRAY)
  
  - PRINTTYPE R3-9(XRAY), R3-127(XRAY)

**Symbol names**

- convention U2-37(MCC), U3-13(MCC), U2-34(CCC)

**prepend dot**

- `/prepend=dot` U3-12(MCC)

**prepend underscore**

- `/prepend=underscore` U3-12(MCC)

**suppress prefix**

- `/noprepend option` U3-12(MCC)

**Symbol record**

- RF-1(ASM)

**Symbolic addressing**

- R2-4(ASM)–R2-6(ASM)

**Symbolic references**

- R2-15(XRAY), U4-14(XRAY)

**data types**

- R2-16(XRAY)

**type casting**

- R2-18(XRAY)

**type conversion**

- R2-17(XRAY)

**variable references**

- R2-22(XRAY)

**program stack**

- R2-23(XRAY)

**explicit references**

- R2-23(XRAY)

**implicit references**

- R2-23(XRAY)

**qualified reference**

- R2-19(XRAY)

**scoping rules**

- R2-19(XRAY)

**storage classes**

- global R2-15(XRAY)

- local R2-16(XRAY)

- register R2-16(XRAY)
static R2-15(XRAY)

Symbols  R2-7(ASM)–R2-10(ASM), R2-7(XRAY)
absolute  R2-9(ASM), R4-7(ASM)
active  R4-20(XRAY)
address  R2-9(XRAY)
address range  R2-10(XRAY)
as command parameters  R3-4(XRAY)
creating  R3-18(XRAY)
debug information  U4-13(XRAY)
debugger  R2-8(XRAY), R3-18(XRAY)
deleting  DELETE command  R3-48(XRAY)
displaying  PRINTSYMBOLS command  R3-127(XRAY)
SYMBOLS command  R3-108(XRAY)
external  R4-6(ASM)
invalid  R2-8(ASM)
keywords  R2-9(XRAY)
local  R2-16(XRAY), R4-8(XRAY)
macro  R2-8(XRAY)
PRINTSYMBOLS cannot find  U4-14(XRAY)
program  R2-7(XRAY), R3-18(XRAY)
qualified  R3-4(XRAY)
relocatable  R2-9(ASM), R4-7(ASM)
reserved  R2-8(XRAY), RA-1(XRAY)
NARG  R2-9(ASM)
type  PRINTTYPE command  R3-130(XRAY)
valid  R2-8(ASM)
XRAY cannot find  U4-14(XRAY)

SYMTRAN linker command  R10-61(ASM)–R10-62(ASM)

Syntax
addressing mode  R3-15(ASM)–R3-20(ASM)
assembler  R2-6(ASM)–R2-17(ASM)
effective address fields  R3-14(ASM)
librarian  R13-1(ASM)–R13-2(ASM)
linker  R10-1(ASM)–R10-4(ASM)

Syntax checking only
/syntax_only option  U3-33(MCC)
-y option  U2-40(MCC), U2-37(CCC)

Syntax, ANSI C  RB-1(MCC)
Syntax, compiler  U2-1(MCC), U3-1(MCC)
sys  U3-3(MCC)
SYS$SCRATCH  U3-4(MCC), U3-26(ASM)
SYSTARTUP.COM file (VMS)  U3-3(MCC)
System data  R9-37(MCC)
System functions  R5-19(MCC), R9-31(MCC)
close  R5-39(MCC), R9-31(MCC)
.NewReader  R5-51(MCC)
 creat  R5-42(MCC)
relationship to fclose  R5-51(MCC)
relationship to rclose  R5-39(MCC)
relationship to read  R5-142(MCC)
relationship to write  R5-202(MCC)
_exit  R5-48(MCC)
lsseek  R5-108(MCC), R9-31(MCC)
relationship to fseek  R5-72(MCC)
relationship to fsetpos  R5-73(MCC)
relationship to rewind  R5-145(MCC)
open  R5-126(MCC), R9-31(MCC)
relationship to fclose  R5-39(MCC)
relationship to fopen  R5-62(MCC)
relationship to read  R5-142(MCC)
relationship to write  R5-202(MCC)
read  R5-142(MCC), R9-31(MCC)
sbrk  R5-146(MCC)
unlink  R5-192(MCC), R9-31(MCC)
relationship to remove  R5-144(MCC)
write  R5-202(MCC), R9-31(MCC)
relationship to fflush  R5-54(MCC)

T
assembler command line flag  U2-11(ASM), U3-11(ASM)

TAD command  S2-137(SUP)
Tagging functions
/trace option  U3-33(MCC)
-Kt option  U2-26(MCC), U2-23(CCC)
tags section  R8-6(MCC), U2-16(MCC), U2-26(MCC), U3-15(MCC), U3-33(MCC), U2-13(CCC), U2-23(CCC)

Master Index-95
tail command 12-4(FLEX)
tan function  R5-183(MCC), U2-40(MCC), U3-25(MCC), U2-37(CCC)
Tangent of a number, computing  R5-183(MCC)
tanh function  R5-184(MCC), U2-40(MCC), U3-25(MCC), U2-37(CCC)

Tape
(see also 9-track tape)
(see also Cartridge tape)
(see Distribution)
tar command  11-7(FLEX), 11-19(FLEX)
Target functions  R2-14(XRAY), R3-38(XRAY)
Target memory test  S2-38(SUP), S2-40(SUP), S2-42(SUP), S2-44(SUP), S2-46(SUP), S2-48(SUP)
Target RAM test  S2-34(SUP)
TCEBRK command  S2-138(SUP)
TCP/IP installation on Apollos  IB-1(FLEX)
version required  IB-1(FLEX)
TCP/IP port number  I3-2(FLEX)
default I3-2(FLEX)
missing I2-8(FLEX)
TCP/IP protocol  U2-11(FLEX)
TED command  S2-139(SUP)
template keyword  R4-6(CCC), RB-1(CCC)
Template processing  R4-6(CCC)
Temporary breakpoints  S4-5(SUP), U4-6(XRAY)
setting
(see Breakpoints)
Temporary file creation
VAX/VMS  U3-26(ASM)
Temporary file placement
DOS  U2-5(ASM)
Temporary files
UNIX
alternate locations  U2-4(MCC)
VMS
alternate locations  U3-4(MCC)
Temporary files location  U4-11(XRAY)
TERM environment variable  UC-2(XRAY), UD-2(XRAY), UD-3(XRAY)
Terminal change  S1-8(SUP)
color  S1-8(SUP)
Terminate XRAY session
QUIT command  R3-138(XRAY)
Termination
abnormal program  R5-22(MCC)
normal program  R5-47(MCC), R5-48(MCC)
TERMINFO environment variable  UD-4(XRAY)
TEST command  R3-197(XRAY)
size qualifiers  R3-3(XRAY)
Test coverage
ANALYZE command  R3-22(XRAY)
data collection  R6-2(XRAY)
ANALYZE  R6-2(XRAY)
PRINTANALYSIS command  R3-115(XRAY)
program_unit  R3-22(XRAY), R3-115(XRAY), R3-124(XRAY)
reporting results  R6-3(XRAY)
branch coverage  R6-3(XRAY)
function coverage  R6-3(XRAY)
instruction coverage  R6-3(XRAY)
line coverage  R6-3(XRAY)
PRINTANALYSIS  R6-3(XRAY)
tutorial  R6-3(XRAY)
Test scripts, running  12-2(FLEX)
Testing installation  12-1(FLEX)
Tests  R5-88(MCC)
TGR  S6-13(SUP)
event system  S6-11(SUP)
external  S6-11(SUP)
this keyword  R4-6(CCC)
this pointer  R2-7(CCC), RC-4(CCC), UA-4(CCC)
Threads
definition  R9-11(MCC)
multiple  R9-11(MCC)
throw keyword  R4-6(CCC), RB-1(CCC)
TID command  S2-141(SUP)
Time
conversion to string R5-25(MCC), R5-43(MCC)
processor time used R5-38(MCC)
Time and date functions R5-19(MCC)
asctime R5-25(MCC)
clock R5-38(MCC)
cftime R5-43(MCC)
difftime R5-44(MCC)
gmtime R5-82(MCC)
localtime R5-104(MCC)
time R5-185(MCC)
Time base
maximum S6-3(SUP)
time function R5-185(MCC)
Time measurement
maximums S6-10(SUP)
Time optimization
optimize=time option U3-28(MCC)
-ot option U2-33(MCC), U2-29(CCC)
_TIME__ preprocessor symbol R4-2(MCC)
Time stamp
compilation
-Vt option U2-38(MCC), U2-35(CCC)
installation S6-4(SUP)
Time stamp information
saving in file S6-15(SUP)
Time Stamp module
count occurences S6-26(SUP)
Time Stamp module S6-1(SUP)-??
absolute time S6-3(SUP)
code accesses S6-3(SUP)
counter overflow S6-7(SUP), S6-10(SUP)
elapsed time S6-2(SUP), S6-16(SUP)
interrupt latency S6-7(SUP)
maximums S6-10(SUP)
module linkage activity S6-3(SUP)
relative time S6-3(SUP)
switches S6-11(SUP)
time base switch S6-10(SUP)
Time stamp module
counter S6-12(SUP)
using S6-6(SUP)
viewing information S6-14(SUP)
time.h include file R5-2(MCC), R5-19(MCC)
Timeout S2-87(SUP)
Timeout warning S2-93(SUP)
Timestamp S2-151(SUP), S2-153(SUP)
timestamp units S2-154(SUP)
Title for listing file
/title option U3-29(MCC)
-Flt option U2-20(MCC), U2-17(CCC)
tm structure R5-19(MCC)
TMP environment variable U2-5(ASM), U2-16(ASM)
TMP_MAX macro R5-15(MCC)
tmpfile() VAX function U3-4(MCC), U3-26(ASM)
toascii function R5-186(MCC)
Tokens
location in string R5-178(MCC)
_tolower function R5-188(MCC)
tolower function R5-3(MCC), R5-8(MCC), R5-187(MCC)
Toolchain compatibility S1-1(SUP)
Toolkit U2-1(FLEX)
(see also Programs)
access I3-6(FLEX)
definition I1-1(FLEX)
directory installed in U1-1(FLEX), U1-2(FLEX)
specifying in UNIX start-up file U1-1(FLEX)
error messages UC-11(FLEX)
executing a program U1-2(FLEX)
expiration date I3-3(FLEX)
host located on U2-10(FLEX)
hosts, specifying U2-1(FLEX)
installing I1-6(FLEX)
installing subset of I1-2(FLEX)
networks, specifying U2-1(FLEX)
testing I2-2(FLEX)
version number I3-3(FLEX)
Toolkit components U1-1(MCC)
Toolkit, components U1-1(CCC)
_totoupper function R5-190(MCC)
toupper function R5-3(MCC), R5-8(MCC), R5-189(MCC)
TRACE S2-145(SUP)
Trace S2-59(SUP), S2-60(SUP), S2-65(SUP),
S2-138(SUP), S2-151(SUP), S2-153(SUP)
capturing S3-37(SUP)
clearing S3-37(SUP)
disassembling S3-39(SUP)
displaying S3-38(SUP)
not supported S1-8(SUP)
recording bus timing information S2-29(SUP)
saving to a file S3-40(SUP)
Trace a subroutine S4-22(SUP)
Trace buffer
SETSTATUS QUALIFY command R3-167(XRAY)
Trace buffer, clear S2-142(SUP)
TRACE command R3-199(XRAY)
Trace command key
Apollo trace support UB-6(XRAY)
DECstation trace support UC-6(XRAY)
HP trace support UD-8(XRAY)
IBM RS/6000 trace support UE-5(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC trace support UG-5(XRAY)
Sun trace support UH-6(XRAY)
VT terminal trace support UI-5(XRAY)
Trace Command Mode R6-21(XRAY), R6-22(XRAY)
Trace commands
STATUS BUFFER R3-183(XRAY)
STATUS EVENT R3-187(XRAY)
STATUS QUALIFY R3-188(XRAY)
STATUS TRACE R3-189(XRAY)
STATUS TRIGGER R3-191(XRAY)
Trace cycle number S2-143(SUP)
Trace events
SETSTATUS EVENT command R3-161(XRAY)
Trace interval S2-144(SUP)
Trace mode S2-145(SUP), R6-20(XRAY)
alignment R3-171(XRAY), R3-200(XRAY)
prefetch R3-171(XRAY), R3-200(XRAY)
trigger R3-173(XRAY)
Trace mode (definition) RD-2(XRAY)
Trace mode keys UB-5(XRAY), UC-5(XRAY),
UD-7(XRAY), UE-4(XRAY), UF-4(XRAY), UG-4(XRAY), UH-5(XRAY),
UI-4(XRAY)
Trace support
buffer displayed
STATUS BUFFER command R3-183(XRAY)
command mode R6-21(XRAY), R6-22(XRAY)
data collection
TRACE command R3-170(XRAY),
R3-199(XRAY)
event definition RD-1(XRAY)
event sequence definition RD-1(XRAY)
events
STATUS EVENT command R3-187(XRAY)
frame definition RD-1(XRAY)
program flow R6-19(XRAY)
qualifications for trace buffer
STATUS QUALIFY command R3-188(XRAY)
SETSTATUS EVENT command R3-161(XRAY)
SETSTATUS QUALIFY command R3-167(XRAY)
SETSTATUS TRIGGER command R3-173(XRAY)
setting triggers
SETSTATUS TRIGGER
command R3-173(XRAY)
STATUS BUFFER command R3-183(XRAY)
STATUS EVENT command R3-187(XRAY)
STATUS QUALIFY command R3-188(XRAY)
STATUS TRACE command R3-189(XRAY)
STATUS TRIGGER command R3-191(XRAY)
summary of settings
STATUS TRACE command R3-189(XRAY)
TRACE command R3-170(XRAY), R3-199(XRAY)
trace mode definition RD-2(XRAY)
trigger definition RD-2(XRAY)
triggers
STATUS TRIGGER command R3-191(XRAY)
tutorial R6-24(XRAY)
Trace viewport U2-15(XRAY), U2-16(XRAY), U3-25(XRAY), U4-4(XRAY)
EXPAND command U3-26(XRAY)
Tracing RF-7(XRAY)
Tracing information
FRAMESTOP option R3-106(XRAY)
track command key
Apollo trace support UB-6(XRAY)
DECstation trace support UC-6(XRAY)
HP trace support UD-8(XRAY)
IBM RS/6000 trace support UE-5(XRAY)
Motorola Delta Series trace support UF-6(XRAY)
PC trace support UG-6(XRAY)
Sun trace support UH-6(XRAY)
VT terminal trace support UI-6(XRAY)
Trailer bytes R6-31(MCC), R6-34(MCC)
Transcendental functions
(see Mathematical functions)
Trap numbers S2-82(SUP)
TRCCLR command S2-142(SUP)
TRCFRAME command S2-143(SUP)
TRCINT command S2-144(SUP)
TRCMODE command S2-145(SUP)
Trigger (definition) RD-2(XRAY)
Trigger (trace) R3-173(XRAY)
Trigger, define a S2-146(SUP)
Triggers
arming S2-76(SUP), S2-78(SUP)

Triggers, trace
SETSTATUS TRIGGER command R3-173(XRAY)
STATUS TRIGGER command R3-191(XRAY)
Tri-state of address bus S2-137(SUP)
Truncating identifiers
/truncate_identifiers U3-12(MCC)
_ut option U2-8(MCC), U2-37(MCC), U2-34(CCC)
try keyword R4-6(CCC), RB-1(CCC)
TSTAMP command S2-151(SUP), S2-153(SUP)
TTL assembler directive R5-71(ASM)
TUNITS command S2-154(SUP)
Tutorial
assembly-level mode U2-20(XRAY)
high-level mode U2-12(XRAY)
macros U2-26(XRAY)
XEL toolchain and XICE debugger S3-1(SUP)
Type casting R3-9(MCC), R3-15(MCC), R2-18(XRAY)
Type checking
TYPECHECK option R3-108(XRAY)
Type conversion R3-14(MCC)
Type conversions R2-17(XRAY)
Type of symbol
PRINTTYPE command R3-130(XRAY)
typedef R3-4(MCC), R3-18(MCC)
typedef declarations R4-10(CCC)
typeof operator R3-13(MCC)
typeof operator, enabling
/mri_extensions option U3-24(MCC)
Types
(see also Data, types)
uintptr_t
fgetpos function R5-56(MCC)
fsetpos function R5-73(MCC)
jmp_buf R5-9(MCC)
lconv structure R5-100(MCC)
ptrdiff_t R5-11(MCC)
returned for functions R9-5(MCC)
sizet R5-11(MCC), R5-17(MCC)
va_list  R5-10(MCC)
wchar_t  R5-11(MCC)
mbstowcs function  R5-114(MCC)
mbtowc function  R5-113(MCC), R5-115(MCC)
wctomb function  R5-200(MCC), R5-201(MCC)

Types, data  R2-16(XRAY)
Type-safe linkage  RC-4(CCC), UA-4(CCC)

U
-u linker command line option  U2-15(ASM)
-U option  U2-36(MCC), U2-33(CCC)
U undefined label error  R7-14(ASM)
-ui option  U2-37(MCC), U2-34(CCC)
UIR command  S2-156(SUP)
Unary expressions  R3-8(MCC)
#define directive  R4-6(MCC), R4-31(MMC)
Undefining label error  (U)  R7-14(ASM)

Undefining preprocessor macros
/undefined option  U3-33(MCC)
-U option  U2-36(MCC), U2-33(CCC)
ungetc function  R5-191(MCC)
Uninitialized data section  U2-16(MCC), U3-15(MCC), U2-13(CCC)
Uninitialized static data section
naming
/rename option  U3-31(MCC)
-NZ option  U2-29(MCC), U2-26(CCC)

Union return value  R7-4(MCC)
union type  R6-4(MCC)
Unions  R6-22(MCC)
UNIX
(see UNIX/DOS)
command line  U2-1(MCC)
command line examples  U2-41(MCC)
command line syntax  U2-1(CCC)
compiler invocation  U2-1(CCC)
compiler syntax  U2-1(MCC)
compiler use  U2-1(MCC), U2-1(CCC)
compiling a program  U2-53(MCC), U2-45(CCC)

environment variables  U2-3(MCC)
file locations  U2-3(MCC)
file name defaults  U2-2(MCC)
invoking compiler  U2-1(MCC)
libraries provided  U2-50(MCC)
option form, positive and negative  U2-10(MCC)
option, command line
descriptions of  U2-10(MCC)
system functions  U2-52(MCC), U2-44(CCC)

UNIX file locations  U2-3(CCC)
UNIX level-1 functions  U2-43(CCC)
UNIX level-1+ elements  R6-2(CCC)
_UNIX preprocessor symbol  R4-3(MCC)
UNIX start-up file  U1-1(FLEX)
specifying license file path in  I1-15(FLEX), I2-1(FLEX), I2-6(FLEX)
specifying location of toolkit in  I2-1(FLEX)

UNIX System V/386
accessing files in nondefault
directories  I3-2(MCC)
distribution format  I3-1(MCC)
installation procedure  I3-1(MCC)

UNIX/DOS  U2-1(ASM)—U2-28(ASM)
assembler  U2-2(ASM)—U2-12(ASM)
command line flags  U2-5(ASM)—U2-11(ASM)
abspcadd  U2-6(ASM)
brb  U2-6(ASM)
brl  U2-6(ASM)
ubs  U2-6(ASM)
brw  U2-6(ASM)
case  U2-7(ASM)
cex  U2-7(ASM)
ccl  U2-7(ASM)
cre  U2-7(ASM)
d  U2-7(ASM)
e  U2-7(ASM)
frl  U2-7(ASM)
frs  U2-7(ASM)
g  U2-7(ASM)
i U2-7(ASM)
llen U2-8(ASM)
mc U2-8(ASM)
md U2-8(ASM)
mex U2-8(ASM)
nest U2-8(ASM)
o U2-8(ASM)
old U2-8(ASM)
op U2-8(ASM)
opnop U2-9(ASM)
p U2-9(ASM)
pco U2-9(ASM)
pcr U2-10(ASM)
pcs U2-10(ASM)
quick U2-10(ASM)
r U2-10(ASM)
rel32 U2-11(ASM)
s U2-11(ASM)
t U2-11(ASM)
w U2-11(ASM)
x U2-11(ASM)

command line options U2-2(ASM)–U2-4(ASM)
-file name defaults U2-4(ASM)
-invocation examples U2-23(ASM)–U2-24(ASM)
-linker U2-13(ASM)–U2-20(ASM)
-command line options U2-13(ASM)–U2-15(ASM)
- C U2-13(ASM)
- c U2-13(ASM)
- F U2-13(ASM)
- o U2-14(ASM)
- r U2-14(ASM)
- u U2-15(ASM)
- V U2-15(ASM)

file name defaults U2-16(ASM)
illegal option combinations U2-15(ASM)
-invocation examples U2-17(ASM)–U2-20(ASM)
-return codes U2-28(ASM)

utility programs
IEE2AOUT U2-24(ASM)–U2-28(ASM)

Unknown command S3-21(SUP)
UNKNOWN TYPE U4-15(XRAY)
unlink function R5-192(MCC), R9-31(MCC),
U2-52(MCC), U3-44(MCC), U2-44(CCC)

relationship to remove R5-144(MCC)

Unlinking a file name R5-192(MCC)
unpacked keyword R4-4(CCC)
unpacked keyword, disabling
-nx option U2-40(MCC), U2-37(CCC)

unpacked type R6-38(MCC)

Unreachable (dead) code
elimination R102(MCC), R5-2(XRAY)

Unreachable code, compiler
optimization R5-5(XRAY)

Unreferenced variable R2-22(XRAY)

Unsigned char default
/unsignedchar option U3-22(MCC)
-Ku option U2-27(MCC), U2-23(CCC)

Unsigned integer
conversion to ASCII string R5-96(MCC)
Unsigned long integer
conversion from string R5-180(MCC)
conversion to ASCII string R5-110(MCC)

unsigned types R3-1(MCC), R6-3(MCC)
unsigned char type R6-10(MCC)
unsigned int type R3-2(MCC), R6-13(MCC)
unsigned long type R3-2(MCC), R6-14(MCC)
unsigned short type R3-2(MCC), R6-11(MCC)

/unsignedchar option (VMS) R4-3(MCC)

Unsupported commands
clock S1-7(SUP)
CPU S1-7(SUP)
Ice S1-7(SUP)
interrupt S1-7(SUP)
Noise S1-8(SUP)
nointerrupt S1-8(SUP)
nomem S1-8(SUP)
restore S1-8(SUP)
save S1-8(SUP)
trace S1-8(SUP)

until macro R4-36(XRAY), U4-6(XRAY)
Unused definition optimization R5-5(XRAY)

Unzoom viewport
ZOOM command R3-213(XRAY)

UP S2-158(SUP)

UP command R3-202(XRAY)
-upd option U2-37(MCC), U2-34(CCC)

Update internal registers S2-156(SUP)
UPL(upload hex data to host) S2-159(SUP)
UPLFMT(specify upload format) S2-160(SUP)

Upload S2-159(SUP)
Upload format S2-160(SUP)

Upper-case characters
converting to R5-189(MCC), R5-190(MCC)
testing for R5-93(MCC)

UPPERCASE linker command R10-63(ASM)
-upu option U2-37(MCC), U2-34(CCC)
-us option U2-37(MCC), U2-34(CCC)
User access I3-6(FLEX)

User setup U1-1(FLEX)
User-defined types R3-18(MCC)
User-modified routines
for embedded systems R9-29(MCC)

Using debugger commands
DEFINE command U2-26(XRAY)
ZOOM command U2-28(XRAY)

Using debugger macros U1-3(XRAY), U2-26(XRAY)
ADD command U2-26(XRAY)
FOPEN command U2-33(XRAY)
INCLUDE command U2-26(XRAY)
macro definitions
listprime U2-28(XRAY)
readprime U2-28(XRAY)
SHOW command U2-33(XRAY)
VCLOSE command U2-33(XRAY)

USR_MRI U2-4(MCC), U2-5(MCC), U2-6(MCC), U2-23(MCC), I2-2(FLEX), U1-1(FLEX), UA-2(FLEX), U2-4(CCC), U2-20(CCC), U2-21(CCC)

USR_MRI environment variable R3-1(XRAY), U2-6(XRAY), U4-11(XRAY)

-ut option U2-8(MCC), U2-37(MCC), U2-34(CCC)

-ut0 option U2-37(MCC), U2-34(CCC)

Utilities
(see Commands)

Utility commands R3-10(XRAY)
ALIAS R3-10(XRAY), R3-20(XRAY)
CEXPRESSION R3-10(XRAY), R3-38(XRAY)

DOWN R3-10(XRAY), R3-56(XRAY)
ERROR R3-10(XRAY), R3-59(XRAY)
HELP R3-10(XRAY), R3-72(XRAY)
HISTORY R3-10(XRAY), R3-74(XRAY)
INCLUDE R3-10(XRAY), R3-79(XRAY)
JOURNAL R3-10(XRAY), R3-86(XRAY)

LOG R3-10(XRAY), R3-91(XRAY)
MODE R3-10(XRAY), R3-92(XRAY)
OPTION R3-10(XRAY), R3-103(XRAY)
PAUSE R3-10(XRAY), R3-114(XRAY)

RESTART R3-10(XRAY), R3-142(XRAY)

Documentation Set for 68000/EC000/HC000 and 68302
SETSTATUS DIR R3-10(XRAY), R3-159(XRAY)
SETSTATUS ENVIRONMENT R3-10(XRAY), R3-160(XRAY)
SETSTATUS READ R3-10(XRAY), R3-169(XRAY)
SETSTATUS VERIFY R3-10(XRAY), R3-177(XRAY)
SETSTATUS WRITE R3-10(XRAY), R3-178(XRAY)
STARTUP R3-10(XRAY), R3-180(XRAY)
STATUS R3-10(XRAY), R3-182(XRAY)
TRACE R3-199(XRAY)
UP R3-10(XRAY), R3-202(XRAY)

Utility programs
IEE2AOUT U2-24(ASM)-U2-28(ASM), U3-21(ASM)-U3-26(ASM)
-utn option (UNIX/DOS) R4-3(MCC)

Apollo trace support UB-6(XRAY)
DECstation support UC-4(XRAY)
HP support UD-6(XRAY)
HP trace support UD-7(XRAY)
IBM RS/6000 support UE-3(XRAY)
IBM RS/6000 trace support UE-4(XRAY)
Motorola Delta Series support UF-3(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC support UG-3(XRAY)
PC trace support UG-5(XRAY)
Sun support UH-4(XRAY)
Sun trace support UH-5(XRAY)
VT terminal support UI-3(XRAY)
VT terminal trace support UI-5(XRAY)

VACTIVE-1 command key
Apollo support UB-4(XRAY)
Apollo trace support UB-5(XRAY)
DECstation support UC-3(XRAY)
DECstation trace support UC-5(XRAY)
HP support UD-6(XRAY)
HP trace support UD-7(XRAY)
IBM RS/6000 support UE-3(XRAY)
IBM RS/6000 trace support UE-4(XRAY)
Motorola Delta Series support UF-3(XRAY)
Motorola Delta Series trace support UF-5(XRAY)
PC support UG-3(XRAY)
PC trace support UG-5(XRAY)
Sun support UH-4(XRAY)
Sun trace support UH-5(XRAY)
VT terminal support UI-3(XRAY)
VT terminal trace support UI-5(XRAY)

Valid hosts IP-1(FLEX), I1-1(FLEX)
Valid operating system UP-1(FLEX)

Value comparison
returning the greater of two values R5-112(MCC)
returning the lesser of two values R5-123(MCC)

Variable argument
macros R5-10(MCC)
Variable initialization
/init_locals option U3-22(MCC)
-KI option U2-25(MCC), U2-22(CCC)

Variable names, truncating
/truncate_identifiers U3-12(MCC)
-ut option U2-8(MCC), U2-37(MCC), U2-34(CCC)

Variable values S2-80(SUP), S2-90(SUP), S2-162(SUP)

Variables R3-1(MCC)
(see also Symbols)
allocation of variables R6-41(MCC)
assigning value to U4-16(XRAY)
debug information U4-13(XRAY)
declaring R4-8(CCC)
declaring value
PRINTVALUE command R3-131(XRAY)
FILL command U4-16(XRAY)
format U4-15(XRAY)
initializations R9-39(MCC)
initializing R4-9(CCC)
local R6-42(MCC)
in the function prologue R7-7(MCC)
monitoring U2-18(XRAY), U2-22(XRAY)
continuously U2-19(XRAY)
scalar U4-16(XRAY)

names R7-11(MCC)
external R11-1(MCC)
inside asm R9-6(MCC)
public R11-1(MCC)
static R11-2(MCC)

referencing
(see also Symbolic references)
register RE-11(XRAY)
saving initialized variables in ROM R9-39(MCC)

SETMEM command U4-16(XRAY)

setting breakpoints at access to U4-

static R5-20(MCC), R6-42(MCC)
structure members, aligning
/align option U3-11(MCC)
/ssmultiple option U3-11(MCC)
-Z options U2-40(MCC), U2-37(CCC)
UNKNOWN TYPE U4-15(XRAY)

vars section R8-5(MCC), U2-16(MCC), U3-15(MCC), U2-13(CCC)
naming
/rename option U3-31(MCC)
-NI option U2-29(MCC), U2-25(CCC)
specifying address mode
/initvars_addr_as options U3-16(MCC)
-ai options U2-17(MCC), U2-13(CCC)

VAX
(see VAX/VMS)
VAX/VMS U3-1(ASM)–U3-27(ASM)
assembly U3-2(ASM)–U3-12(ASM)

command line flags U3-4(ASM)–U3-11(ASM)
abspcadd U3-6(ASM)
brb U3-6(ASM)
brl U3-6(ASM)
brs U3-6(ASM)
brw U3-6(ASM)
case U3-7(ASM)
cex U3-7(ASM)
cl U3-7(ASM)
ere U3-7(ASM)
d U3-7(ASM)
e U3-7(ASM)
frl U3-7(ASM)
frs U3-7(ASM)
g U3-7(ASM)
i U3-7(ASM)
llen U3-8(ASM)
mc U3-8(ASM)
md U3-8(ASM)
mex U3-8(ASM)
nest U3-8(ASM)
o U3-8(ASM)
old U3-8(ASM)
op U3-8(ASM)
opnop U3-9(ASM)
p U3-9(ASM)
pco U3-9(ASM)
pcr U3-10(ASM)
pcs U3-10(ASM)
quick U3-10(ASM)
r U3-10(ASM)
rel32 U3-11(ASM)
s U3-11(ASM)
t U3-11(ASM)
w U3-11(ASM)
x U3-11(ASM)

command line options U3-3(ASM)–U3-4(ASM)
DEFINE U3-3(ASM)
FLAGS U3-3(ASM)
IPATH U3-3(ASM)
LIST U3-3(ASM)
NOLIST U3-3(ASM)
NOOBJECT U3-3(ASM)
OBJECT U3-3(ASM)
VERSION U3-3(ASM)

file name defaults U3-4(ASM)

invitation examples U3-12(ASM)–U3-13(ASM)

VERSION U3-15(ASM)

illegal option combinations U3-15(ASM)

invocation examples U3-16(ASM)

invocation syntax U3-17(ASM)

return codes U3-26(ASM)

temporary files U3-27(ASM)

utility programs

IEE2AOUT U3-21(ASM)–U3-26(ASM)

-Vb option U2-37(MCC), U2-34(CCC)
VCLEAR command R3-204(XRAY)
VCLOSE command R3-205(XRAY), U2-33(XRAY)

-Vd option U2-37(MCC), U2-34(CCC)

Vendor daemon UB-3(FLEX)

(see MRI daemon)

Vendor daemon line

(see License file)

Verbose mode toggle

get_license UA-1(FLEX)
mlicense U3-2(FLEX)
mlicense.daemon U2-8(FLEX), U3-2(FLEX), UA-2(FLEX)

Verbose mode, enabling and disabling

-V options U2-37(MCC), U2-34(CCC)

VERIFY command S2-161(SUP)

Verifying installation

(see Testing installation)
VERSION assembler command line option U3-3(ASM)
VERSION librarian command line option U3-19(ASM)
VERSION linker command line option U3-15(ASM)
Version number of features I3-3(FLEX)
Versions of Flexible License Manager, differences IC-1(FLEX), IC-2(FLEX)
vfprintf function R5-197(MCC)
-Vi option U2-38(MCC), U2-35(CCC)
View viewport U3-28(XRAY)
  displaying
    VIEW option R3-108(XRAY)
    STATUS BUFFER command U3-29(XRAY)
  trace display format U3-29(XRAY)
  update Code viewport R6-21(XRAY), U3-29(XRAY)
Viewing viewports
  SCROLL option R3-108(XRAY)
Viewport
  attaching macro
    VMACRO command R3-206(XRAY)
  clearing
    VCLEAR command R3-204(XRAY)
  cursor position
    VSETC command R3-212(XRAY)
  making active
    VACTIVE command R3-203(XRAY)
Viewport (definition) RD-2(XRAY)
Viewport commands S3-17(SUP), R3-11(XRAY), U3-34(XRAY)
  VACTIVE R3-11(XRAY), R3-203(XRAY)
  VCLEAR R3-11(XRAY), R3-204(XRAY)
  VCLOSE R3-11(XRAY), R3-205(XRAY)
  VMACRO R3-11(XRAY), R3-206(XRAY)
  VOPEN R3-11(XRAY), R3-208(XRAY)
  VSSCREEN R3-11(XRAY), R3-210(XRAY)
  VSETC R3-11(XRAY), R3-212(XRAY)
  ZOOM R3-11(XRAY), R3-213(XRAY)
Viewports U3-6(XRAY)
  active viewport U3-33(XRAY)
    VACTIVE+1 command key U3-33(XRAY)
VACTIVE-1 command key U3-33(XRAY)
Break viewport U2-18(XRAY), U2-21(XRAY), U3-7(XRAY)
Code viewport U2-16(XRAY), U3-1(XRAY), U3-2(XRAY), U3-10(XRAY)
Command viewport U2-18(XRAY), U3-1(XRAY), U3-2(XRAY), U3-12(XRAY)
Data viewport U2-18(XRAY), U3-1(XRAY), U3-2(XRAY), U3-13(XRAY)
defining U3-32(XRAY)
deleting U3-33(XRAY)
Error viewport U3-14(XRAY)
Help viewport U3-15(XRAY)
Journal viewport U3-17(XRAY)
Log viewport U3-17(XRAY)
numbers U3-6(XRAY)
open for writing U3-34(XRAY)
Registers viewport U2-20(XRAY), U3-2(XRAY), U3-18(XRAY)
removing U3-33(XRAY)
resizing U2-28(XRAY), U4-7(XRAY)
scrolling a viewport U3-33(XRAY)
  CTRL-F control key U3-34(XRAY)
  CTRL-U control key U3-34(XRAY)
  MOVE DOWN command key U3-34(XRAY)
  MOVE TO BOTTOM command key U3-33(XRAY)
  MOVE TO TOP command key U3-33(XRAY)
  MOVE UP command key U3-34(XRAY)
setting cursor U3-35(XRAY)
Stack viewport U2-20(XRAY), U3-2(XRAY), U3-21(XRAY)
Status Line viewport U2-15(XRAY), U3-1(XRAY), U3-2(XRAY), U3-22(XRAY)
Trace viewport U2-15(XRAY), U2-
16(XRAY), U3-25(XRAY), U4-4(XRAY)
View viewport U3-28(XRAY)
zooming a viewport U3-34(XRAY)
ZOOM command U3-34(XRAY)
ZOOM command key U3-34(XRAY)

Virtual
function R3-12(CCC), R3-14(CCC)
    compared to nonvirtual R3-14(CCC)
    pure R3-14(CCC), RC-4(CCC)
functions
    pure UA-3(CCC)
    keyword R4-6(CCC)
    member classes R4-6(CCC)
    member functions R4-6(CCC)
Virtual derivation R3-2(CCC)
Virtual function RC-4(CCC)
virtual function UA-4(CCC)
virtual keyword R4-6(CCC)
Virtual table definition
    +e option R3-18(CCC)
    +ne option R3-18(CCC)
Virtual table generation
    +e option U2-16(CCC)
VMACRO command R3-206(XRAY), U3-32(XRAY)

VMS
(see VAX/VMS)
accessing files in nondefault directories I2-3(MCC)
command line U3-1(MCC)
command line examples U3-35(MCC)
compiler syntax U3-1(MCC)
compiler use U3-1(MCC)
compiling a program U3-45(MCC)
distribution format I2-1(MCC)
file locations U3-3(MCC)
file name defaults U3-2(MCC)
installation procedure I2-1(MCC)
invoking compiler U3-1(MCC)
option form, positive and negative U3-5(MCC)
option, command line
descriptions of U3-5(MCC)

VMS installation I2-1(XRAY)
distribution format I2-1(XRAY)
installation procedure I2-1(XRAY)
terminal support I2-1(XRAY)
    configuration parameters I2-1(XRAY)
    using XRAY I2-3(XRAY)
_VMS preprocessor symbol R4-3(MCC)
void type R3-2(MCC), R3-3(MCC)
void* pointer R4-7(CCC)
Volatile
data R4-6(CCC)
generating information
    -nA command line option R4-6(CCC)
    member functions R4-6(CCC)
volatile keyword R4-6(CCC)
volatile member function RC-4(CCC), UA-4(CCC)
volatile object RC-4(CCC), UA-4(CCC)
volatile type R3-17(MCC)
volatile variables, disabling optimizations
    /stablemem option U3-27(MCC)
    -Ob option U2-30(MCC), U2-27(CCC)
VOPEN command R3-208(XRAY), U3-22(XRAY), U3-32(XRAY), U3-34(XRAY), U4-7(XRAY)
vprintf function R5-198(MCC)
VSCREEN command R3-210(XRAY), U2-24(XRAY), U3-1(XRAY), U3-2(XRAY), U3-33(XRAY)
VSCREEN command key
    Apollo support UB-4(XRAY)
    DECstation support UC-4(XRAY)
    HP support UD-6(XRAY)
    IBM RS/6000 support UE-3(XRAY)
    Motorola Delta Series support UF-4(XRAY)
    PC support UG-4(XRAY)
    Sun support UH-4(XRAY)
    VT terminal support UI-3(XRAY)
VSETC command R3-212(XRAY)
vssprintf function R5-20(MCC), R5-199(MCC), R9-11(MCC)
-Vt option U2-38(MCC), U2-35(CCC)
VT terminal support
host-specific information UI-1(XRAY)
MOVE TO BOTTOM control key UI-2(XRAY)
MOVE TO TOP control key UI-2(XRAY)
-Vw option U2-38(MCC), U2-35(CCC)

W
assembler command line flag U2-11(ASM), U3-11(ASM)
+w option U2-36(CCC)
-Wa option U2-38(MCC), U2-35(CCC)
@wait_state pseudo-register RA-2(XRAY), RF-4(XRAY)
WARN linker command R10-64(ASM)
WARNING
Status Line viewport U3-24(XRAY)
#warning directive R4-32(MCC)
relation to #pragma warn R4-30(MCC)
Warning messages RC-1(MCC)
(see also Messages)
enumeration types R4-15(CCC)
suppression
/suppress=warning option U3-30(MCC)
-Qw option U2-36(MCC), U2-33(CCC)

WARNING-xxx (stub routine) called R5-13(MCC), R9-31(MCC)
ftell R5-74(MCC), R5-108(MCC), R5-126(MCC), R5-192(MCC)
wchar_t type R5-11(MCC)
mbstowcs function R5-114(MCC)
mbtowc function R5-113(MCC), R5-115(MCC)
wstombs function R5-200(MCC)
wctomb function R5-201(MCC)
wstombs function R5-200(MCC)
wctomb function R5-201(MCC)
relationship to wstombs R5-200(MCC)
Weak externals
initialize to zero
/weakextern=initzero option U3-34(MCC)
-X0 option U2-39(MCC), U2-36(CCC)
no value
/weakextern=public option U3-34(MCC)
-Xp option U2-39(MCC), U2-36(CCC)
uninitialized globals
/weakextern=common option U3-34(MCC)
-Xe option U2-39(MCC), U2-36(CCC)
when macro R4-37(XRAY), U4-6(XRAY), U4-7(XRAY)
WHEN-THEN statements
syntax S4-14(SUP)
WHILE...ENDW assembler directive R7-13(ASM)
while statement R3-24(MCC)
WHILE statement in macros R4-7(XRAY)
White space, testing for R5-92(MCC)
Who should install l1-1(FLEX)
Windows U1-5(XRAY)
-WI option U2-38(MCC), U2-35(CCC)
word macro R4-38(XRAY)
Word size R6-23(MCC)
Wrapper, C++ R5-6(CCC)
write function R5-202(MCC), R9-31(MCC), U2-52(MCC), U3-44(MCC), U2-44(CCC)
relationship to fflush R5-54(MCC)
write routine R3-112(XRAY), U2-23(XRAY)
Writing
memory to file
SETSTATUS WRITE command R3-178(XRAY)
to a file
FOPEN command R3-66(XRAY)
to ROM
ROMACCESS command R3-147(XRAY)
Writing a character to standard output R5-135(MCC)
Writing a short integer to a stream R5-
Writing a string to standard output  R5-137(MCC)
Writing bytes to a file  R5-202(MCC)
Writing to a file  R5-76(MCC)
Writing value to output port  R4-27(XRAY)

X-Y-Z
x assembler command line flag  U2-11(ASM), U3-11(ASM)
-x option  U2-39(MCC), U2-36(CCC)
-x option (UNIX/DOS)  R4-3(MCC)
X Window U3-1(FLEX)
X Window support
buttons
   (see Buttons)
colors, setting UA-3(XRAY)
DECstation terminals  UC-1(XRAY), UD-1(XRAY)
environment variables
color UA-3(XRAY)
display UA-4(XRAY), UA-5(XRAY)
fonts UA-3(XRAY)
XRAY_BG_COLOR UA-3(XRAY)
XRAY_CUR_COLOR UA-3(XRAY)
XRAY_FONT_PATH UA-3(XRAY)
XRAY_FONTB UA-1(XRAY)
XRAY_FONTN UA-1(XRAY)

fonts.dir UA-3(XRAY)
HP support UD-2(XRAY)
mkfontdir command UA-3(XRAY)
mouse support
   (see Mouse support)
OpenWindows UA-6(XRAY)
setting colors UA-3(XRAY)
Sun UH-2(XRAY)
   xhost command UA-5(XRAY)
X windows S1-2(SUP)
-X0 option U2-39(MCC), U2-36(CCC)
-Xc option U2-39(MCC), U2-36(CCC)
XCOM assembler directive  R5-72(ASM)
XDEF assembler directive  R5-73(ASM)
   xhost command UA-5(XRAY)
XRAY Debugger
description U1-2(MCC), U1-2(ASM), U1-3(CCC)
   fully qualified path names
   /debug=fullfilename option U3-19(MCC)
   -Gf option U2-21(MCC), U2-18(CCC)
genrating debugging information
   /debug option U3-18(MCC)
   -g option U2-22(MCC), U2-19(CCC)
line number information
   /debug=lines option U3-19(MCC)
   -Gl option U2-21(MCC), U2-18(CCC)
multiple statements on line
   /debug=multi_stmt option U3-19(MCC)
   -Gm option U2-21(MCC)
   restricted information
   /debug=restricted option U3-20(MCC)
   -Gr option U2-21(MCC), U2-18(CCC)
XRAY environment variable  R3-1(XRAY), R3-160(XRAY), U2-3(XRAY), U4-10(XRAY)
XRAY file format  R3-90(XRAY), U2-3(XRAY)
XRAY format
   IEEE-695 U2-2(XRAY)
XRAY Source Explorer
   -Gs option U2-22(MCC), U2-19(CCC)
XRAY with C++ RG-1(XRAY)
XRAY_BG_COLOR environment

XHS68K, features of  U1-2(FLEX)
XHS68KLIB environment variable  U2-5(XRAY)
XICEVARS command S2-162(SUP)
XOR_CHANGE environment variable UA-5(XRAY)
XOS_OFF environment variable RG-1(XRAY)
XOS68K I1-2(FLEX)
-Xp option U2-39(MCC), U2-36(CCC)
XRAY command line options  U2-9(XRAY)
XRAY commands S1-1(SUP)
   unsupported S1-7(SUP)

Master Index-109
variable UA-3(XRAY)
XRAY_CUR_COLOR environment variable UA-3(XRAY)
XRAY_DISPLAY environment variable UA-4(XRAY), UH-2(XRAY)
XRAY_FG_COLOR environment variable UA-3(XRAY)
XRAY_FONTB environment variable UA-1(XRAY)
XRAY_FONTN environment variable UA-1(XRAY)
XRAYFONT environment variable R3-105(XRAY), U2-7(XRAY)
XRAYLIB environment variable R3-1(XRAY), U2-4(XRAY), U4-11(XRAY)
XRAYTMP environment variable U2-7(XRAY), U4-11(XRAY)
XREF assembler directive R5-74(ASM)
-y option U2-40(MCC), U2-37(CCC)
+z option U2-38(CCC)
-Z2 option R6-36(MCC), U2-40(MCC), U2-37(CCC)
-Z4 option R6-36(MCC), U2-40(MCC), U2-37(CCC)
zalloc function R5-21(MCC), R5-203(MCC), R9-12(MCC)
  relationship to free R5-67(MCC)
  relationship to realloc R5-143(MCC)
Zero initialization
local variables
  /init_locals option U3-22(MCC)
  -KI option U2-25(MCC), U2-22(CCC)
zerovars section R6-42(MCC), R8-5(MCC), U2-16(MCC), U3-15(MCC), U2-13(CCC)
naming
  /rename options U3-31(MCC)
  -NZ option U2-29(MCC), U2-26(CCC)
-Zm option R6-27(MCC), R6-31(MCC), R6-35(MCC), U2-40(MCC), U2-38(CCC)
Zoom (definition) RD-2(XRAY)
ZOOM command R3-213(XRAY), U2-28(XRAY), U4-7(XRAY)
Applied Microsystems Corporation

Applied Microsystems Corporation maintains a worldwide network of direct sales offices committed to quality service and support. For information on products, pricing, or delivery, please call the nearest office listed below. If you are unsure which office to contact, call 1-800-426-3925 for assistance.

**CORPORATE OFFICE**
Applied Microsystems Corporation
5020 148th Avenue Northeast
P.O. Box 97002
Redmond, WA 98073-9702
(206) 882-2000
1-800-426-3925
Customer Support
1-800-ASK-4AMC
TRT TELEX 185196
Fax (206) 883-3049

**EUROPE**
Applied Microsystems Corporation Ltd
AMC House
South Street
Wendover
Aylesbury, Bucks
HP22 6EF England
44 (0) 296-625462
Telex 265871 REF WOT 004
Fax 44 (0) 296-623460

**GERMANY**
Applied Microsystems GmbH
Dammstrasse 6
W-6453 Seligenstadt
Germany
06182/9203-0
Fax 06182/9203-15

**JAPAN**
Applied Microsystems Japan, Ltd.
Nihon Seimei
Nishi-Gotanda Building
7-24-5 Nishi-Gotanda
Shinagawa-Ku
Tokyo 141, Japan
3-3493-0770
Fax 3-3493-7270

**U.S. SALES OFFICES**
Applied Microsystems Corporation of Washington
3333 Bowers Avenue
Suite #220
Santa Clara, CA 95054
(408) 727-5433
Fax (408) 727-9011

Applied Microsystems Corporation of Washington
25909 Pala Place
Suite #280
Mission Viejo, CA 92691
(714) 588-0585
Fax (714) 588-1476

Applied Microsystems Corporation
14643 Dallas Parkway
Suite 230, LB-76
Dallas, Texas 75240
(214) 991-6344
Fax (214) 991-4581

Applied Microsystems Corporation
919E North Plum Grove Road
Schaumburg, IL 60173
(708) 240-2000
Fax (708) 240-1309

Applied Microsystems Corporation of Washington
6 Cabot Place
Stoughton, MA 02072
(617) 341-3121
Fax (617) 341-0245
<table>
<thead>
<tr>
<th>Part No.</th>
<th>Revision History</th>
<th>Minimum SW Ver.</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>922-17320-00</td>
<td>First release of XICE for 68000 and 68302 on an EL1600 emulator</td>
<td>XICE -PC 5.04, XICE-SUN 5.02</td>
<td>5/92</td>
</tr>
<tr>
<td>922-17320-01</td>
<td>Added support for X-windows, C++, operations during run</td>
<td>XICE 6.10</td>
<td>10/92</td>
</tr>
<tr>
<td>922-17320-02</td>
<td>Add 68HC/EC000</td>
<td>XICE 6.40</td>
<td>5/93</td>
</tr>
</tbody>
</table>

P/N 922-17320-02
May 1993