



inmos®

ANSI C Toolset Handbook

INMOS Limited

ST **SGS-THOMSON**
MICROELECTRONICS

INMOS is a member of the SGS-THOMSON Microelectronics Group

72 TDS 355 00

October 1992

Contents

Contents	i
Preface	iii
Host versions	iii
Toolset documentation set	iii
Documentation conventions	iii
ANSI C toolset	1
Standard file extensions	2
Tools	3
Transputer targets — options for icc & ilink	24
Debugger commands	25
Debugger symbolic functions	25
Debugger monitor page commands	26
Simulator commands	28
Runtime Library	29

© INMOS Limited 1992. This document may not be copied, in whole or in part, without prior written consent of INMOS.

 [®], **inmos** [®], IMS and occam are trademarks of INMOS Limited.

INMOS Limited is a member of the SGS-THOMSON Microelectronics Group.

 is a registered trademark of the SGS-THOMSON Microelectronics Group.

The C compiler implementation was developed from the Perihelion Software "C" Compiler and the Codemist Norcroft "C" Compiler.

INMOS Document Number: 72 TDS 355 00

Preface

Host versions

The documentation set which accompanies the ANSI C toolset is designed to cover all host versions of the toolset:

- IMS D7314 – IBM PC compatible running MS-DOS
- IMS D4314 – Sun 4 systems running SunOS.
- IMS D6314 – VAX systems running VMS.

Toolset documentation set

The documentation set comprises the following volumes:

- 72 TDS 345 01 *ANSI C Toolset User Guide*
- 72 TDS 346 01 *ANSI C Toolset Reference Manual*
- 72 TDS 347 01 *ANSI C Language and Libraries Reference Manual*
- 72 TDS 348 01 *ANSI C Optimizing Compiler User Guide*
- 72 TDS 354 00 *Performance Improvement with the DX314 ANSI C Toolset*
- 72 TDS 355 00 *ANSI C Toolset Handbook* (this document)
- 72 TDS 360 00 *ANSI C Toolset Master Index*

In addition a host specific *Delivery Manual* and a set of generic *Release Notes* are provided.

Documentation conventions

The following typographical conventions are used in this manual:

Bold type	Used to emphasize new or special terminology.
Teletype	Used to distinguish command line examples, code fragments, and program listings from normal text.
<i>Italic type</i>	In command syntax definitions, used to stand for an argument of a particular type. Used within text for emphasis and for book titles.
Braces { }	Used to denote optional items in command syntax.
Brackets []	Used in command syntax to denote optional items on the command line.
Ellipsis . . .	In general terms, used to denote the continuation of a series. For example, in syntax definitions denotes a list of one or more items.
	In command syntax, separates two mutually exclusive alternatives.

ANSI C toolset

Tool	Description
<i>icc</i>	Full ANSI C standard compiler with concurrency support. Generates object code for specific transputer targets or transputer classes.
<i>icconf</i>	The configurer which generates configuration binary files from configuration descriptions.
<i>icollect</i>	The code collector which generates executable code files.
<i>idebug</i>	The network debugger which provides post-mortem and interactive debugging of transputer programs.
<i>idump</i>	The memory dumper tool which dumps root transputer memory for post mortem debugging.
<i>iemit</i>	The memory configurer tool which helps to configure the transputer memory interface.
<i>ieprom</i>	The EPROM formatter tool which creates executable files for loading into ROM.
<i>ilibr</i>	The toolset librarian which creates libraries from compiled code files.
<i>ilink</i>	The toolset linker which links compiled code and libraries into a single unit.
<i>ilist</i>	The binary lister which displays binary files in a readable form.
<i>imakef</i>	The Makefile generator which creates Makefiles for toolset compilations.
<i>imap</i>	The map tool which generates a memory map for an executable file.
<i>iserver</i>	The host file server which loads programs onto transputer hardware and provides host communication.
<i>isim</i>	The T425 simulator which allows programs to be run without hardware.
<i>iskip</i>	The skip loader tool which loads programs over the root transputer.

Standard file extensions

Extension	Description
.bt1	Bootable file which can be loaded onto a transputer or transputer network. Created by icollect .
.btr	Executable code without a bootstrap. Created by icollect and used as input to ieprom .
.c	C source files.
.cfb	Configuration binary file. Created by icconf .
.cfs	Configuration description source file. Created by the user as a text file. Input to icconf .
.h	Header files for use in C source code.
.lbb	Library build files which specify the components of a library to ilibr .
.lib	Library file containing a collection of binary modules. Created by ilibr .
.lku	Linked unit. Created by ilink .
.lnk	Linker indirect files which specify the components of a program to be linked to ilink .
.map	Map file output by the collector.
.rsc	Dynamically loadable file. Created by icollect .
.tco	Compiled code file. Created by icc .

Tools

icc – ANSI C compiler

Compiles C source code.

Syntax: `icc filename {options}`

where: *filename* is the C program source code.

Options:

Option	Description
<i>Transputer type</i>	See page 22 for a list of options to specify transputer type.
AS	Assemble the input file to produce an object file. The compiler phase is suppressed.
C	Performs a syntax check only. Generates no object code. This option is ignored by the optimizing compiler.
D symbol	Defines a symbol. Same as <code>#define symbol 1</code> at the start of the source file.
D symbol=value	Defines a symbol and assigns a value. Same as <code>#define symbol value</code> at the start of the source file.
EC	Disables checks for invalid type casts. ANSI compliance check.
EP	Disables checks for invalid text after <code>#else</code> or <code>#endif</code> . ANSI compliance check.
EZ	Disables checks for zero-sized arrays. ANSI compliance check.
FC	Change the signedness property of plain <code>char</code> to be signed. The default is to compile <code>chars</code> as unsigned.
FH	Performs a number of software quality checks.
FM	Generates warning messages on <code>#defined</code> but unused macros.
FS	Directs the compiler to treat right-shifts of signed integers as arithmetic shifts.
FV	Reports all externally visible functions and variables which are declared but unreferenced, and have file scope.
G	Generates comprehensive debugging data. The default is to produce minimal debugging data. Debugging data is required for the correct operation of <code>idebug</code> .
HELP	Displays full help information for the tool.
I	Displays detailed progress information at the terminal as the compiler runs.

Option	Description
J dir	Adds <i>dir</i> to the list of directories to be searched for source files incorporated with the <code>#include</code> directive in extended search paths.
KS	Enables stack checking.
O outputfile	Specifies an output file. If no filename is given the compiler derives the output filename from the input filename stem and adds the <code>.tco</code> extension.
P mapfile	Produces a map of workspace for each function defined in the file, and a map of the static area of the whole file. The map is written to the file <i>mapfile</i> .
PP	Lists the preprocessed source file to <code>stdout</code> .
S	Compiles the source file to assembly language and writes it to a file. Assembly is suppressed and no object code is produced. The file is named after the input file and given the <code>.s</code> extension.
U symbol	Disables a symbol definition. Equivalent to <code>#undef symbol</code> at the start of the source file.
WA	Suppresses messages warning of '=' in conditional expressions.
WD	Suppresses messages warning of deprecated function declarations.
WF	Suppresses messages warning of implicit declarations of <code>extern int()</code> .
WN	Suppresses messages warning of implicit narrowing or lower precision.
WT	Suppresses messages warning of the possibility of less efficient code when compiled for a transputer class.
WV	Suppresses messages warning of non-declaration of <code>void</code> functions

icconf – configurer

Generates configuration binary files from configuration descriptions

Syntax: `icconf filename {options}`

where: *filename* is the configuration description file.

Options:

Option	Description
C	Checks the configuration description only. No configuration data file is generated.
G	This option is used when postmortem or interactively debugging and disables any ordering of process memory segments in the configuration code by the <code>order</code> and <code>location</code> process attributes. The G option significantly modifies the runtime behavior of the configured program because virtual though routing is used for all channel communication between processors. This results in a memory overhead and reduction in performance of communications. This option cannot be used with the GA , GP , RA or RO options.
GA	Generates a configuration which can be debugged using the <i>Advanced Toolset</i> debugger. This option has the same side effects as the G option except that the order and location attributes are not disabled. This option cannot be used with the G or GP options.
GP	This option is used when postmortem debugging and disables any ordering of process memory segments in the configuration code by the <code>order</code> and <code>location</code> process attributes. The runtime behavior of the application will be little different to the default behavior i.e. when no options are specified. Virtual routing is enabled and may be used. This option <i>may</i> be used with the RA option but <i>not</i> with the G , GA or RO options.
I	Displays extra information as the tool runs.
NV	Generates a configuration without virtual routing.
O filename	Specifies an output filename. If no output file is specified the configuration data file is given the base name of the input file and the <code>.cfb</code> extension is added.
P procname	Specifies the name of the root processor when configuring for EPROMs. <i>procname</i> must not be an element from an array of processors.

PRE	Generates a configuration which can be profiled using the <i>Advanced Toolset</i> network execution profiler. This option has the same side effects as the GA option. Note: this option cannot be used with the GA or PRU options.
PRU	Generates a configuration which can be profiled using the <i>Advanced Toolset</i> network utilization profiler. This option has the same side effects as the GA option. Note: this option cannot be used with the GA or PRE options.
RA	Creates a file suitable for a boot-from-ROM application in which the user and system processes for the root processor and all other processors are loaded into RAM to execute.
RO	Creates a file suitable for a boot-from-ROM application in which the user and system processes for the root processor execute in ROM and for all other processors the user and system processes are loaded into RAM to execute.
RS romsize	Specifies the size of ROM on the root processor. Only valid when used with the ' RA ' or ' RO ' options. <i>romsize</i> is specified in decimal format and can be followed by ' K ' or ' M ' to indicate kilobytes or megabytes.
W	Disables configurer messages of severity <i>Warning</i> .
WP	Generates additional pedantic <i>Warning</i> messages.

icollect – code collector

Generates bootable code files. Also used to generate non-bootable files for dynamic loading or booting from ROM.

Syntax: `icollect filename {options}`

where: *filename* is a configuration data file created by a configurer or a single linked unit created by `ilink`.

Options:

Option	Description
B <i>filename</i>	Uses a user-defined bootstrap loader program in place of the standard bootstrap. The program is specified by <i>filename</i> and must conform to the rules described in appendix F. This option can only be used with the 'T' option (unconfigured mode) and cannot be used with the 'RA' and 'RO' options.
BM	Instructs the tool to use a different bootstrapping scheme, which uses the bottom of memory, see section 3.8. This option is only valid for configured programs i.e. when the 'T' option is <i>not</i> used.
C <i>filename</i>	Specifies a name for the debug data file. A filename must be supplied and is used as given. This option can only be used with the 'T' option (unconfigured mode) and cannot be used with the 'D' or 'K' options.
CM	Instructs the collector to add a bootstrap which will clear memory during the booting and loading of the transputer network. Intended for use with parity-checked memory (see section 3.4).
D	Disables the generation of the debug data file for single transputer programs. This option can only be used with the 'T' option (unconfigured mode).
E	Changes the setting of the transputer Halt On Error flag. HALT mode programs are converted so that they not stop when the error flag is set, and non HALT mode programs to stop when the error flag is set. This option can only be used with the 'T' option (unconfigured mode).
I	Displays progress information as the collector runs.

Option	Description
K	Creates a single transputer file with no bootstrap code. If no file is specified the output file is named after the input filename and given the <code>.rsc</code> extension. This option can only be used with the 'T' option (unconfigured mode).
M <i>memorysize</i>	Specifies the memory size available (in bytes) on the root processor for single transputer programs. <i>memorysize</i> is specified in bytes and may be given in decimal format (optionally followed by 'K' or 'M' to indicate Kilobytes or Megabytes respectively), or it may be specified in hexadecimal using the '#' or '\$' prefixes. This option can only be used with the 'T' option (unconfigured mode) and results in a smaller amount of code being produced (see section 3.3).
O <i>filename</i>	Specifies the output file. A filename must be supplied and is used as given. (See section 3.2.4).
P <i>filename</i>	Specifies a name for the memory map file. A filename must be supplied and is used as given. The file extension <code>.map</code> should be used when the file is to be used as input to <code>imap</code> , see chapter 12.
RA	Creates a file for processing by <code>ieprom</code> into a boot from ROM file to run in RAM. If no output file is specified the filename is taken from the input file and given the <code>.btr</code> extension. This option is only necessary when using the 'T' option (unconfigured mode) to create a ROM code file.
RO	Creates a file for processing by <code>ieprom</code> into a boot from ROM file to run in ROM. If no output file is specified the filename is taken from the input file and given the <code>.btr</code> extension. This option is only necessary when using the 'T' option (unconfigured mode) to create a ROM code file.
RS <i>romsize</i>	Specifies the size of ROM on the root processor in bytes. Only valid when used with the 'RA' or 'RO' options. <i>romsize</i> is specified in bytes and may be given in decimal format (optionally followed by 'K' or 'M' to indicate Kilobytes or Megabytes respectively), or it may be specified in hexadecimal using the '#' or '\$' prefixes. This option is only necessary when using the 'T' option (unconfigured mode) to create a ROM code file.

Option	Description
S <i>stacksize</i>	Specifies the extra runtime stack size in words for single transputer programs. <i>stacksize</i> is specified in words and may be given in decimal format (optionally followed by 'K' or 'M' to indicate Kilowords or Megawords respectively), or it may be specified in hexadecimal using the '#' or '\$' prefixes. This option can only be used with the 'T' option.
T	Creates a bootable file for a single transputer. The input file specified on the command line must be a linked unit. This option can not be used for programs linked with the <i>reduced</i> runtime library.
Y	Disables interactive debugging with <i>idebug</i> and reduces the amount of memory used. (See section 3.10). This option can only be used with the 'T' option (unconfigured mode).

idebug – network debugger

Provides post-mortem and breakpoint debugging.

Syntax: *idebug bootablefile {options}*

where: *bootablefile* is the bootable file to be debugged

Options:

Option	Description
A	Assert INMOS subsystem Analyse . Directs the debugger to assert Analyse on the sub-network connected to the root processor. Required when using B004 type boards.
AP	A replacement for the A option when running programs on boards from certain vendors. Asserts Analyse on the network connected to the root processor. Contact your supplier to see whether this option is applicable to your hardware. It does not apply to boards manufactured by INMOS.
B <i>linknumber</i>	Interactive breakpoint debug a network that is connected to the root processor via link <i>linknumber</i> . <i>idebug</i> executes on the root processor. Must be accompanied by the <i>iserver</i> 'SR' option.
C <i>type</i>	Specify a processor type (e.g. T425) instead of a class (e.g. TA) for programs that have not been configured.
D	Dummy debugging session. Can be used for familiarization with the debugger or establishing memory mappings. Must be accompanied by the <i>iserver</i> 'SR' option.
GXX	Improves symbolic debugging support for C++ source code. Should be specified when debugging C++ programs.
I	Display debugger version string. Must be accompanied by the <i>iserver</i> 'SR' option.

Option	Description
J <i>#hexdigits</i>	Takes a hexadecimal digit sequence of up to 16 digits and replicates it throughout the data regions of a program (stack, static, heap and vectorspace as appropriate) when interactive debugging. The digit sequence <i>must</i> be preceded by a hash, '#', character. Used when breakpoint debugging configured T426 programs.
K <i>#hexdigits</i>	As the J option but includes freespace. Used when interactive debugging non-configured T426 programs.
M <i>linknumber</i>	Postmortem debug a previous interactive debugging session. idebug executes on the root processor. Must be accompanied by the iserver 'SA' option.
N <i>filename</i>	Postmortem debug a program from a network dump file <i>filename</i> , created by idebug . The file is assumed to have the extension .dmp if none is specified. Must be accompanied by the iserver 'SR' option.
Q <i>variable</i>	Specify environment variable used to specify the ITERM file. The default is " ITERM ".
R <i>filename</i>	Postmortem debug a program that uses the root transputer. <i>filename</i> is the file that contains the contents of the root processor (created by idump or isim). The file is assumed to have the extension .dmp if none is supplied.
S	Ignore subsystem signals when interactive debugging.
T <i>linknumber</i>	Postmortem debug a program that does not use the root processor, on a network that is connected to link <i>linknumber</i> of the root processor. idebug executes on the root processor. Must be accompanied by the iserver 'SA' option.
XQ	Causes the debugger to request confirmation of the Quit command.

idump – memory dumper

Writes the root transputer's memory to a file. Used in debugging programs that use the root transputer.

Syntax: **idump** *filename* *memorysize* [{ *startoffset* *length* }]

where: *filename* is the name of the dump file to be created.

memorysize is the number of bytes, starting at the bottom of memory, to be written to the file.

startoffset is an offset in bytes from the start of memory.

length is the amount of memory in bytes, starting at *startoffset*, to be dumped in addition to *memorysize*.

iemit – memory interface configurer

Evaluates memory configurations.

Syntax: `iemit options`

Options:

Option	Description
A	Produce ASCII output file.
E	Invoke interactive mode.
F filename	Specify input memory configuration file.
I	Select verbose mode. In this mode the user will receive status information about what the tool is doing during operation for example, reading or writing to a file.
O filename	Specify output filename.
P	Produce PostScript output file.

ieprom – EPROM program convertor

Formats bootable code for installation by ROM loaders.

Syntax: `ieprom filename {options}`

where: *filename* is the name of the control file.

Options:

Option	Description
I	Selects verbose mode. In this mode the user will receive status information about what the tool is doing during its operation, for example reading or writing to a file.
R	Directs <code>ieprom</code> to display the absolute address of the code reference point. This address can be used to locate within the memory map created by the <code>icollect 'P'</code> option.

i1ibr – librarian

Builds libraries of code from separate files.

Syntax: `i1ibr filenames {options}`

where: *filenames* is a list of input files separated by spaces.

Options:

Option	Description
F <i>filename</i>	Specifies a library indirect file.
I	Displays progress information as the library is built.
O <i>filename</i>	Specifies an output file. If no output file is specified the name is taken from the first input file and a <code>.lib</code> extension is added.

i1ink – linker

Links object files together, resolving external references to create a single linked unit.

Syntax: `i1ink [filenames] {options}`

where: *filenames* is a list of compiled files or library files.

Options:

Option	Description
<i>Transputer type</i>	See page 22 for a list of options to specify transputer type.
EX	Allows the extraction of modules without linking them.
F <i>filename</i>	Specifies a linker indirect file.
H	Generates the linked unit in HALT mode. This is the default mode for the linker and may be omitted for HALT mode programs. This option is mutually exclusive with the 'S' option.
I	Displays progress information as the linking proceeds.
KB <i>memorysize</i>	Specifies virtual memory required in Kilobytes.
LB	Specifies that the output is to be generated in LFF format, for use with the <code>i1boot</code> bootstrap tool and <code>i1conf</code> configurer tool used in earlier INMOS toolsets. (See footnote 2).
LC	Specifies that the output is to be generated in LFF format, for use with the <code>i1conf</code> tool used in earlier INMOS toolsets. (See footnote 2).
ME <i>entryname</i>	Specifies the name of the main entry point of the program and is equivalent to the <code>#mainentry</code> linker directive (See below).
MO <i>filename</i>	Generates a module information file with the specified name.
O <i>filename</i>	Specifies an output file.
S	Generates the linked unit in STOP mode. This option is mutually exclusive with the 'H' option.
T	Specifies that the output is to be generated in TCOFF format. This format is the default format.
U	Allows unresolved references.
X	Generates the linked unit in UNIVERSAL error mode, which can be mixed with HALT and STOP modes.
Y	Disables interactive debugging for OCCAM code. Used when linking in OCCAM modules compiled with interactive debugging disabled.

ilist – binary lister

Decodes and displays information from object files and bootable files.

Syntax: `ilist {filenames} {options}`

where: *filenames* is a list of one or more files to be displayed.

Options:

Option	Description
A	Displays all the available information on the symbols used within the specified modules.
C	Displays the code in the specified file as hexadecimal. This option also invokes the 'T' option by default.
E	Displays all exported names in the specified modules.
H	Displays the specified file(s) in hexadecimal format.
I	Displays full progress information as the lister runs.
M	Displays module data.
N	Displays information from the library index.
O filename	Specifies an output file. If more than one file is specified the last one specified is used.
P	Displays any procedural interfaces found in the specified modules.
R reference	Displays the library module(s) containing the specified reference. This option is used in conjunction with other option to display data for a specific symbol. If more than one library file is specified the last one specified is used.
T	Displays a full listing of a file in any file format.
W	Causes the lister to identify a file. The filename (including the search path if applicable) is displayed followed by the file type. This is the default option.
X	Displays all external references made by the specified modules.

imakef – Makefile generator

Creates Makefiles for toolset compilations.

Syntax: `imakef filenames {options}`

where: *filenames* is a list of target files for which makefiles are to be generated.

Options:

Option	Description
C	This option is used when incorporating C or FORTRAN modules into the program. It specifies that the list of files to be linked is to be read from a linker indirect file. This option <i>must</i> be specified for correct C or FORTRAN operation.
D	Disables the generation of debugging information in compilations. The default is to compile with full debugging information.
I	Displays full progress information as the tool runs.
M	Produce compiler, linker and collector map files for <code>imap</code> .
NI	Files in the directories in <code>ISEARCH</code> are not put into the makefile. This means that system files are not present, making it much easier to read.
O filename	Specifies an output file. If no file is specified the output file is named after the target file and given the <code>.mak</code> extension.
R	Writes a deletion rule into the makefile.
Y	Disables interactive (breakpoint) debugging in all compilations. The default is to compile with full breakpoint debugging information.

imap – memory mapper

Generates a memory map for an executable file.

Syntax: `imap filename { options }`

where: *filename* is the name of the file containing the map output from the collector.

Options:

Option	Description
A	Displays the list of symbols produced by the linker, including those symbols the linker identifies as not being used. This option will not override the 'R' option if it is used.
I	Displays progress information as <code>imap</code> processes information from the input files, such as the filenames of files as they are opened and closed.
O filename	Specifies an output file.
R	This option reduces the amount of detail generated by <code>imap</code> in two ways: <ul style="list-style-type: none"> the Module memory usage table only displays details for user modules i.e. 'USER' and 'SHARED_USER' processes. the Symbol table excludes those symbols containing a '%' character in their name. Such symbols are normally internal symbols e.g. C runtime library symbols.
ROM hex offset	This option is only applicable to, and must be specified for, code targeted at ROM. It enables a hexadecimal offset to be specified which represents the start address of the code in ROM. This offset will be added to the start address of any code which is to run in ROM, in <code>imap</code> 's output.

iserver – server/loader

Loads programs onto transputers and transputer boards and serves host communications.

Syntax: `iserver {options}`

Options:

Option	Description
SA	Analyses the root transputer and peeks 8K of its memory.
SB filename	Boots the program contained in the named file.
SC filename	Copies the named file to the root transputer link.
SE	Terminates the server if the transputer error flag is set or a control link error message is received.
SI	Displays progress information as the program is loaded.
SK interval	Specifies the number of seconds between attempts to access the resource.
SL name	Specifies the capability name.
SM	Invokes the session manager interface.
SP n	Sets the size of memory to peek on Analyse to <i>n</i> Kbytes.
SR	Resets the root transputer and its subsystem.
SS	Serves the link, i.e. provides host system support to programs communicating on the host link.
ST	All of the following command line is passed directly to the booted program as parameters.
Option 'SB filename' is equivalent to 'SR SS SI SC filename'.	

isim – T425 simulator

Simulates the execution of a program on the IMS T425.

Syntax: `isim program [programparameters] {options}`

where: *program* is the program bootable file.

programparameters is a list of parameters to the program. The list of parameters may follow the `isim 'N'` option and parameters must be separated by spaces.

Options:

Option	Description
B	Batch mode operation. The simulator runs in line mode i.e. full display data is not provided. Commands are read in from the input stream e.g. the keyboard and executed. The commands are not echoed to the output stream e.g. the display screen, as they are executed.
BQ	Batch Quiet mode. The simulator automatically executes the program specified on the command line and then terminates. If an error occurs, the appropriate message will be displayed. The debugging facilities of the simulator are not available in this mode.
BV	Batch Verify mode. Similar to batch mode, except that the commands and prompts displayed when running the simulator in interactive mode are echoed to the output stream e.g. the display.
I	Displays information about the simulator as it runs.
N	No more options for the simulator. Any options entered after this option will be assumed to be program parameters to be passed to the program running on the simulator.

iskip – skip loader

Allows programs to be loaded onto transputer networks beyond the root transputer.

Syntax: `iskip linknumber {options}`

where: *linknumber* is the link on the root transputer to which the target transputer network is connected.

Options:

Option	Description
E	Directs <code>iskip</code> to monitor the subsystem error status and terminates when it becomes set.
I	Displays detailed progress information as the tool loads.
R	Reset subsystem. Resets all transputers connected downstream of link <i>linknumber</i> . Does <i>not</i> reset the root transputer.
RP	A replacement for the R option when running programs on boards from certain vendors. Contact your supplier to see whether this option is applicable to your hardware. It does not apply to boards manufactured by INMOS.

Transputer targets — options for `icc` & `ilink`

Option	Description
TA	Specifies target transputer class TA (T400, T414, T425, T426, T800, T801, T805).
TB	Specifies target transputer class TB (T400, T414, T425, T426)
T212	Specifies a T212 target processor.
T222	Specifies a T222 target processor. Same as T212
M212	Specifies a M212 target processor. Same as T212
T2	Same as T212, T222 and M212
T225	Specifies a T225 target processor.
T3	Same as T225.
T400	Specifies a T400 target processor. Same as T425.
T414	Specifies a T414 target processor. This is the default processor type and may be omitted when the target processor is a T414 processor.
T4	Same as T414 (default).
T425	Specifies a T425 target processor.
T426	Specifies a T426 target processor.
T5	Same as T400, T425 and T426.
T800	Specifies a T800 target processor.
T8	Same as T800.
T801	Specifies a T801 target processor. Same as T805.
T805	Specifies a T805 target processor.
T9	Same as T801 and T805.

Debugger commands

Debugger symbolic functions

BACKTRACE	Locate to the calling function or procedure.
END OF FILE	Go to the last line in the file.
CHANGE FILE	Display a different source file.
CHANNEL	Locate to the process waiting on a channel.
CONTINUE FROM †	Restart a stopped process from the current line.
ENTER FILE	Change to an included source file.
EXIT FILE	Return to the enclosing source file.
FINISH	Quit the debugger.
GET ADDRESS	Display the location of a source line in memory.
GOTO LINE	Go to a specific line in the file.
HELP	Display a summary of commonly used symbolic functions.
INFO	Display process information (e.g. instruction pointer, process descriptor, process name).
INSPECT	Display the type and value of a source code symbol.
INTERRUPT †	Force the debugger into the Monitor page without stopping the program.
MODIFY †	Change the value of a variable in memory.
MONITOR	Change to the monitor page.
RELOCATE	Locate back to the last location line.
RESUME †	Resume a process stopped at a breakpoint.
RETRACE	Undo a BACKTRACE .
SEARCH	Search for a specified string.
TOGGLE BREAK †	Set or clear a breakpoint on the current line.
TOGGLE HEX	Enables/disables hex-oriented display of constants and variables for C.
TOP	Locate back to the error or last source code location.
TOP OF FILE	Go to the first line in the file.

Note: † = Functions only available in interactive mode.

Debugger monitor page commands

Key	Meaning	Description
A*	ASCII	View a region of memory in ASCII.
B†*	Breakpoint	Display the Breakpoint menu enabling breakpoints to be set, cleared or listed.
C	Compare	Compare the code on the network with the code that should be there to ensure that the code has not been corrupted.
D*	Disassemble	Display the transputer instructions at a specified area of memory.
E	Next Error	Switch the current display information to that of the next processor in the network which has halted with its error flag set.
F*	Select file	Select a source file for symbolic display using the filename of the object file produced for it.
G	Goto process	Goto symbolic debugging for a particular process.
H*	Hex	View a region of memory in hexadecimal.
I*	Inspect	View a region of memory in a symbolic type. Types are expressed as standard OCCAM types.
J†*	Jump	Start or resume the application program.
K	Processor names	Display the names and types of all processors in the network.
L	Links	Display instruction pointers and process descriptors for the processes currently waiting for input or output on a transputer link, or for a signal on the Event pin.
M	Memory map	Display the memory map of the current processor.
N*	Network dump	Copy the entire state of the transputer network into a 'network dump' file in order to allow continued (off-line) debugging at a later date.
O*	Specify process	Resume the source level symbolic features of the debugger for a particular process.
P*	Processor	Switch the current display information to that of another processor.
Q	Quit	Leave the debugger and return to the host operating system.

† = Interactive mode only.
* = String editing functions available for these commands.

Key	Meaning	Description
R	Run queues	Display instruction pointers and process descriptors of the processes on either the high or low priority active process queue.
S†	Show messages	Display the Messages menu enabling the default actions of the debugger to debug support functions to be changed.
T	Timer queues	Display instruction pointers, the process descriptors and the wake-up times of the processes on either the high or low priority timer queue.
U†	Update	Update the monitor page display to reflect the current state of the processor.
V	Process names	Display the memory map of processes on the current processor.
W†*	Write	Write to any portion of memory in a symbolic type. Types are expressed as standard OCCAM types.
X	Exit	Return to symbolic mode.
Y†	Postmortem	Change an interactive breakpoint debugging session into a post-mortem debug session.
Z	Virtual links	Display instruction pointers and process descriptors for processes waiting on the configurer's software virtual links.
?	Help	Display help information.

† = Interactive mode only.
* = String editing functions available for these commands.

Simulator commands

Key	Meaning	Description
A	ASCII	Displays a portion of memory in ASCII.
B	Break points	Breakpoint menu.
D	Disassemble	Displays transputer instructions at a specified area of memory.
G	Go	Runs (or resumes) the program.
H	Hex	Displays a portion of memory in hexadecimal.
I	Inspect	Displays a portion of memory in any OCCAM type.
J	Jump into program	Runs (or resumes) the program. Same as G.
L	Links	Displays <code>Iptr</code> and <code>Wptr</code> for processes waiting for input or output on a link, or for a signal on the Event pin.
M	Memory map	This option is not supported for the current toolset.
N	Create dump file	Creates a core dump file.
P	Program boot	Simulates a program 'boot' onto the transputer.
Q	Quit	Quits the simulator.
R	Run queue	Displays <code>Iptr</code> and <code>Wptr</code> for processes on the high or low priority active process queues.
S	Single step	Executes the next transputer instruction.
T	Timer queue	Displays <code>Iptr</code> , <code>Wptr</code> , and wake-up times for processes on the high or low priority timer queues.
U	Assign register	Assigns a value to a register.
?	Help	Displays help information.
?†	Query state	Displays values of registers and queue pointers.
.†	Where	Displays next <code>Iptr</code> and transputer instruction.
† Batch mode commands.		

, , , Scroll the display.

, Display help information.

Redraw the screen.

Quit the simulator.

Runtime Library

abort Aborts the program.

```
#include <stdlib.h>
void abort(void);
```

abs Calculates the absolute value of an integer.

```
#include <stdlib.h>
int abs(int j);
```

acos Calculates the arc cosine of the argument.

```
#include <math.h>
double acos(double x);
```

acosf Calculates the arc cosine of a float number.

```
#include <mathf.h>
float acosf(float x);
```

alloc86 Allocates a block of host memory. MS-DOS only.

```
#include <dos.h>
pcpointer alloc86(int n);
```

asctime Converts a broken-down-time structure to an ASCII string.

```
#include <time.h>
char* asctime(const struct tm *timeptr);
```

asin Calculates the arc sine of the argument.

```
#include <math.h>
double asin(double x);
```

asinf Calculates the arc sine of a float number.

```
#include <mathf.h>
float asinf(float x);
```

assert Inserts diagnostic messages.

```
#include <assert.h>
void assert(int expression);
```

atan Calculates the arc tangent of the argument.

```
#include <math.h>
double atan(double x);
```

atan2 Calculates the arc tangent of y/x.

```
#include <math.h>
double atan2(double y, double x);
```

atan2f Calculates arc tangent of y/x where both are floats.

```
#include <mathf.h>
float atan2f(float y, float x);
```

atanf Calculates the arc tangent of a float number.

```
#include <mathf.h>
float atanf(float x);
```

atexit Specifies a function to be called when the program ends.

```
#include <stdlib.h>
int atexit(void (*func)(void));
```

atof Converts a string of characters to a double.

```
#include <stdlib.h>
double atof(const char *nptr);
```

atoi Converts a string of characters to an int.

```
#include <stdlib.h>
int atoi(const char *nptr);
```

atol Converts a string of characters to a long integer.

```
#include <stdlib.h>
long int atol(const char *nptr);
```

bdos Performs a simple MS-DOS function. MS-DOS only.

```
#include <dos.h>
int bdos(int dosfn, int dosdx, int dosal);
```

BitCnt Count the number of bits set.

```
#include <misc.h>
int BitCnt(int word);
```

BitCntSum Count the number of bits set and sum with an integer.

```
#include <misc.h>
int BitCntSum(int word, int count_in);
```

BitRevNBits Reverse the order of the least significant bits of an integer.

```
#include <misc.h>
int BitRevNBits(int numbits, int data);
```

BitRevWord Reverse the order of the bits in an integer.

```
#include <misc.h>
int BitRevWord(int data);
```

BlockMove Copy a block of memory

```
#include <misc.h>
void BlockMove(void *dest, const void *source, size_t n);
```

bsearch Searches a sorted array for a given object.

```
#include <stdlib.h>
void *bsearch(const void *key,
              const void *base,
              size_t nmemb, size_t size,
              int (*compar)(const void *,
                           const void *));
```

call_without_gsb Calls the pointed to function without passing the gsb.

```
#include <misc.h>
void call_without_gsb( void (*fn_ptr)(void),
                      int number_of_words_for_parameters,
                      ... )
```

calloc Allocates memory space for an array of items and initializes the space to zeros.

```
#include <stdlib.h>
void *calloc(size_t nmemb, size_t size);
```

ceil Calculates the smallest integer not less than the argument.

```
#include <math.h>
double ceil(double x);
```

ceilf Calculates the smallest integer not less than the float argument.

```
#include <mathf.h>
float ceilf(float x);
```

ChanAlloc Allocates and initializes a channel.

```
#include <channel.h>
Channel *ChanAlloc(void);
```

ChanIn Inputs data on a channel.

```
#include <channel.h>
void ChanIn(Channel *c, void *cp, int count);
```

ChanInChanFail Inputs data on a link channel or aborts.

```
#include <channel.h>
int ChanInChanFail(Channel *chan, void *cp,
int count, Channel *failchan);
```

ChanInChar Inputs one byte on a channel.

```
#include <channel.h>
unsigned char ChanInChar(Channel *c);
```

ChanInInt Inputs an integer on a channel.

```
#include <channel.h>
int ChanInInt(Channel *c);
```

ChanInit Initializes a channel pointer.

```
#include <channel.h>
void ChanInit(Channel *chan);
```

ChanInTimeFail Inputs data on a channel or times out.

```
#include <channel.h>
int ChanInTimeFail(Channel *chan, void *cp,
int count, int time);
```

ChanOut Outputs data on a channel.

```
#include <channel.h>
void ChanOut(Channel *c, void *cp, int count);
```

ChanOutChanFail Outputs data or aborts on failure.

```
#include <channel.h>
int ChanOutChanFail(Channel *chan, void *cp,
int count, Channel *failchan);
```

ChanOutChar Outputs one byte on a channel.

```
#include <channel.h>
void ChanOutChar(Channel *c, unsigned char ch);
```

ChanOutInt Outputs an integer on a channel.

```
#include <channel.h>
void ChanOutInt(Channel *c, int n);
```

ChanOutTimeFail Outputs data on a channel or times out.

```
#include <channel.h>
int ChanOutTimeFail(Channel *chan, void *cp,
int count, int time);
```

ChanReset Resets a channel.

```
#include <channel.h>
int ChanReset(Channel *c);
```

clearerr Clears error and end of file indicators for a file stream.

```
#include <stdio.h>
void clearerr(FILE *stream);
```

clock Determines the amount of processor time used.

```
#include <time.h>
clock_t clock(void);
```

close Closes a file. File handling primitive.

```
#include <iocntrl.h>
int close(int fd);
```

cos Calculates the cosine of the argument.

```
#include <math.h>
double cos(double x);
```

cosf Calculates the cosine of a float number.

```
#include <mathf.h>
float cosf(float x);
```

cosh Calculates the hyperbolic cosine of the argument.

```
#include <math.h>
double cosh(double x);
```

coshf Calculates the hyperbolic cosine of a float number.

```
#include <mathf.h>
float coshf(float x);
```

CrcByte Calculate CRC of most significant byte of an integer.

```
#include <misc.h>
int CrcByte(int data, int crc_in, int generator);
```

CrcFromLsb Calculates the CRC of a byte sequence starting at the least significant bit.

```
#include <misc.h>
int CrcFromLsb(const char *string, size_t length,
int generator, int old_crc);
```

CrcFromMsb Calculates the CRC of a byte sequence starting at the most significant bit.

```
#include <misc.h>
int CrcFromMsb(const char *string, size_t length,
int generator, int old_crc);
```

CrcWord Calculate CRC of an integer.

```
#include <misc.h>
int CrcWord(int data, int crc_in, int generator);
```

creat Creates a file for writing. File handling primitive.

```
#include <iocntrl.h>
int creat(char *name, int flag);
```

ctime Converts a calendar time value to a string.

```
#include <time.h>
char *ctime(const time_t *timer);
```

debug_assert Stops process/alerts debugger if condition fails.

```
#include <misc.h>
void debug_assert(const int exp);
```

debug_message Inserts a debugging message.

```
#include <misc.h>
void debug_message(const char *message);
```

debug_stop Stops a process and notifies the debugger.

```
#include <misc.h>
void debug_stop(void);
```

difftime Calculates the difference between two calendar times.

```
#include <time.h>
double difftime(time_t time1, time_t time0);
```

DirectChanIn Inputs data on a channel.

```
#include <channel.h>
void DirectChanIn(Channel *c, void *cp, int count);
```

DirectChanInChar Input one byte on a channel.

```
#include <channel.h>
unsigned char DirectChanInChar(Channel *c);
```

DirectChanInInt Inputs an integer on a channel.

```
#include <channel.h>
int DirectChanInInt(Channel *c);
```

DirectChanOut Outputs data on a channel.

```
#include <channel.h>
void DirectChanOut(Channel *c, void *cp, int count);
```

DirectChanOutChar Outputs one byte on a channel.

```
#include <channel.h>
void DirectChanOutChar(Channel *c, unsigned char ch);
```

DirectChanOutInt Outputs an integer on a channel.

```
#include <channel.h>
void DirectChanOutInt(Channel *c, int n);
```

div Calculates the quotient and remainder of a division.

```
#include <stdlib.h>
div_t div(int numer, int denom);
```

exit Terminates a program.

```
#include <stdlib.h>
void exit(int status);
```

exit_noterminate Version of **exit** for configured processes.

```
#include <misc.h>
void exit_noterminate(int status);
```

exit_repeat Terminates a program so that it can be restarted.

```
#include <misc.h>
void exit_repeat(int status);
```

exit_terminate Version of **exit** for configured processes.

```
#include <misc.h>
void exit_terminate(int status);
```

exp Calculates the exponential function of the argument.

```
#include <math.h>
double exp(double x);
```

expf Calculates the exponential function of a float number.

```
#include <mathf.h>
float expf(float x);
```



fabs Calculates the absolute value of a floating point number.

```
#include <math.h>
double fabs(double x);
```

fabsf Calculates the absolute value of a float number.

```
#include <mathf.h>
float fabsf(float x);
```

fclose Closes a file stream.

```
#include <stdio.h>
int fclose(FILE *stream);
```

feof Tests for end of file.

```
#include <stdio.h>
int feof(FILE *stream);
```

ferror Tests for a file error.

```
#include <stdio.h>
int ferror(FILE *stream);
```

fflush Flushes an output stream.

```
#include <stdio.h>
int fflush(FILE *stream);
```

fgetc Reads a character from a file stream.

```
#include <stdio.h>
int fgetc(FILE *stream);
```

fgetpos Obtains the value of the file position indicator.

```
#include <stdio.h>
int fgetpos(FILE *stream, fpos_t *pos);
```

fgets Reads a line from a file stream.

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);
```

filesize Determines the size of a file. File handling primitive.

```
#include <iocntrl.h>
long int filesize(int fd);
```

floor Calculates the largest integer not greater than the argument.

```
#include <math.h>
double floor(double x);
```

floorf float form of **floor**.

```
#include <mathf.h>
float floorf(float x);
```

fmod Calculates the floating point remainder of x/y .

```
#include <math.h>
double fmod(double x, double y);
```

fmodf Calculates the floating point remainder of x/y .

```
#include <mathf.h>
float fmodf(float x, float y);
```

fopen Opens a file.

```
#include <stdio.h>
FILE *fopen(const char *filename,
            const char *mode);
```

fprintf Writes a formatted string to a file.

```
#include <stdio.h>
int fprintf(FILE *stream, const char *format, ...);
```

fputc Writes a character to a file stream.

```
#include <stdio.h>
int fputc(int c, FILE *stream);
```

fputs Writes a string to a file stream.

```
#include <stdio.h>
int fputs(const char *s, FILE *stream);
```

fread Reads records from a file.

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t nmem,
             FILE *stream);
```

free Frees an area of memory.

```
#include <stdlib.h>
void free(void *ptr);
```

free86 Frees host memory space allocated by **alloc86**. MS-DOS only.

```
#include <dos.h>
void free86(pcpointer p);
```

freopen Opens a file that may already be open.

```
#include <stdio.h>
FILE *freopen(const char *filename, const char *mode,
              FILE *stream);
```

frexp Separates a floating point number into a fraction and an integral power of 2.

```
#include <math.h>
double frexp(double value, int *exp);
```

frexpf Separates a floating point number of type **float** into a fraction and an integral power of 2.

```
#include <mathf.h>
float frexpf(float value, int *exp);
```

from_host_link Retrieve the channel coming from the host.

```
#include <hostlink.h>
Channel* from_host_link( void )
```

from86 Transfers host memory to the transputer. MS-DOS only.

```
#include <dos.h>
int from86(int len, pcpointer there, char *here);
```

fscanf Reads formatted input from a file stream.

```
#include <stdio.h>
int fscanf(FILE *stream, const char *format, ...);
```

fseek Sets the file position indicator to a specified offset.

```
#include <stdio.h>
int fseek(FILE *stream, long int offset,
          int whence);
```

fsetpos Sets the file position indicator to an `fpos_t` value obtained from `fgetpos`.

```
#include <stdio.h>
int fsetpos(FILE *stream, const fpos_t *pos);
```

ftell Returns the position of the file position indicator for a file stream.

```
#include <stdio.h>
long int ftell(FILE *stream);
```

fwrite Writes records from an array into a file.

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size,
             size_t nmemb, FILE *stream);
```

get_bootlink_channels Obtains the channels associated with the boot link.

```
#include <bootlink.h>
int get_bootlink_channels( Channel** in_ptr,
                          Channel** out_ptr )
```

get_code_details_from_channel Retrieves details from a dynamically loadable file that is transmitted over a channel.

```
#include <fnload.h>
int get_code_details_from_channel(Channel* in_channel,
                                 fn_info* fn_details)
```

get_code_details_from_file Retrieves details from a dynamically loadable file which is stored on disc.

```
#include <fnload.h>
int get_code_details_from_file(const char* filename,
                              fn_info* fn_details,
                              size_t* file_hdr_size)
```

get_code_details_from_memory Retrieves details from the image of a dynamically loadable file which is stored in internal memory .

```
#include <fnload.h>
int get_code_details_from_memory(const void* addr_of_file_image,
                                fn_info* fn_details,
                                size_t* file_hdr_size,
                                loaded_fn_ptr* function_pointer)
```

get_details_of_free_memory Reports the details of memory considered by the configurer to be unused.

```
#include <misc.h>
int get_details_of_free_memory( void** base_of_free_memory,
                               size_t* size_of_free_memory )
```

get_details_of_free_stack_space Reports the limits of free space on current stack.

```
#include <misc.h>
void get_details_of_free_stack_space(void** stack_limit_ptr,
                                    size_t* remaining_stack_space_ptr)
```

get_param Reads parameters from the configuration level. Applies only to configured processes.

```
#include <misc.h>
void *get_param(int n);
```

getc Gets a character from a file.

```
#include <stdio.h>
int getc(FILE *stream);
```

getchar gets a character from stdin

```
#include <stdio.h>
int getchar(void);
```

getenv Returns a pointer to the string associated with a host environment variable.

```
#include <stdlib.h>
char *getenv(const char *name);
```

getkey Reads a character from the keyboard.

```
#include <iocntrl.h>
int getkey(void);
```

gets Reads a line from from `stdin`

```
#include <stdio.h>
char *gets(char *s);
```

gmtime Converts a calendar time to a broken-down time, expressed as a UTC time.

```
#include <time.h>
struct tm *gmtime(const time_t *timer);
```

halt_processor Halts the processor

```
#include <misc.h>
void halt_processor(void);
```

host_info Gets data about the host system.

```
#include <host.h>
void host_info(int *host, int *os, int *board);
```

int86 Performs a MS-DOS software interrupt. MS-DOS only.

```
#include <dos.h>
int int86(int intno, union REGS *inregs,
          union REGS *outregs);
```

int86x Software interrupt with segment register setting. MS-DOS only.

```
#include <dos.h>
int int86x(int intno, union REGS *inregs,
           union REGS *outregs,
           struct SREGS *segregs);
```

intdos Performs an MS-DOS interrupt. MS-DOS only.

```
#include <dos.h>
int intdos(union REGS *inregs,
           union REGS *outregs);
```

intdosx MS-DOS interrupt with segment register setting. MS-DOS only.

```
#include <dos.h>
int intdosx(union REGS *inregs,
            union REGS *outregs,
            struct SREGS *segregs);
```

isalnum Tests whether a character is alphanumeric.

```
#include <ctype.h>
int isalnum(int c);
```

isalpha Tests whether a character is alphabetic.

```
#include <ctype.h>
int isalpha(int c);
```

isatty Tests for a terminal stream.

```
#include <iocntrl.h>
int isatty(int fd);
```

iscntrl Tests whether a character is a control character.

```
#include <ctype.h>
int iscntrl(int c);
```

isdigit Tests whether a character is a decimal digit.

```
#include <ctype.h>
int isdigit(int c);
```

isgraph Tests whether a character is printable (non-space).

```
#include <ctype.h>
int isgraph(int c);
```

islower Tests whether a character is a lower-case letter.

```
#include <ctype.h>
int islower(int c);
```

isprint Tests whether a character is printable (includes space).

```
#include <ctype.h>
int isprint(int c);
```

ispunct Tests to see if a character is a punctuation character.

```
#include <ctype.h>
int ispunct(int c);
```

isspace Tests to see if a character is one which affects spacing.

```
#include <ctype.h>
int isspace(int c);
```

isupper Tests whether a character is an upper-case letter.

```
#include <ctype.h>
int isupper(int c);
```

isxdigit Tests to see if a character is a hexadecimal digit.

```
#include <ctype.h>
int isxdigit(int c);
```

labs Calculates the absolute value of a long integer.

```
#include <stdlib.h>
long int labs(long int j);
```

ldexp Multiplies a floating point number by an integer power of two.

```
#include <math.h>
double ldexp(double x, int exp);
```

ldexpf Multiplies a float number by an integral power of two.

```
#include <mathf.h>
float ldexpf(float x, int exp);
```

ldiv Calculates the quotient and remainder of a long division.

```
#include <stdlib.h>
ldiv_t ldiv(long int numer, long int denom);
```

load_code_from_channel Receives the code block of a dynamically loadable file from a channel and copies it into internal memory.

```
#include <fnload.h>
loaded_fn_ptr load_code_from_channel(Channel* in_channel,
                                     const fn_info* fn_details, void* dest)
```

load_code_from_file Transfers code from a dynamically loadable file to internal memory.

```
#include <fnload.h>
loaded_fn_ptr load_code_from_file(const char* filename,
                                   const fn_info* fn_details,
                                   size_t file_hdr_size,
                                   void* dest)
```

load_code_from_memory Transfers code from a dynamically loadable file from one area of internal memory to another.

```
#include <fnload.h>
loaded_fn_ptr load_code_from_memory(const void* src,
                                     const fn_info* fn_details,
                                     size_t file_hdr_size,
                                     void* dest)
```

localeconv Gets numeric formatting data for the current locale.

```
#include <locale.h>
struct lconv *localeconv(void);
```

localtime Converts a calendar time into a broken-down time, expressed as local time.

```
#include <time.h>
struct tm *localtime(const time_t *timer);
```

log Calculates the natural logarithm of the double argument.

```
#include <math.h>
double log(double x);
```

logf Calculates the natural logarithm of a float number.

```
#include <mathf.h>
float logf(float x);
```

log10 Calculates the base-10 logarithm of the double argument.

```
#include <math.h>
double log10(double x);
```

log10f Calculates the base-10 logarithm of a float number.

```
#include <mathf.h>
float log10f(float x);
```

longjmp Performs a non-local jump to the given environment.

```
#include <setjmp.h>
void longjmp(jmp_buf env, int val);
```

lseek Repositions the current file position. File handling primitive.

```
#include <iocntrl.h>
int lseek(int fd, long int offset, int origin);
```

malloc Allocates an area of memory.

```
#include <stdlib.h>
void *malloc(size_t size);
```

max_stack_usage Report runtime stack usage.

```
#include <misc.h>
long max_stack_usage(void);
```

mblen Determines the number of bytes in a multibyte character.

```
#include <stdlib.h>
int mblen(const char *s, size_t n);
```

mbstowcs Converts multibyte sequence to `wchar_t` sequence.

```
#include <stdlib.h>
size_t mbstowcs(wchar_t *pwcs, const char *s, size_t n);
```

mbtowc Converts multibyte character to type `wchar_t`.

```
#include <stdlib.h>
int mbtowc(wchar_t *pwc, const char *s, size_t n);
```

memchr Finds first occurrence of a character in an area of memory.

```
#include <string.h>
void *memchr(const void *s, int c, size_t n);
```

memcmp Compares characters in two areas of memory.

```
#include <string.h>
int memcmp(const void *s1, const void *s2,
           size_t n);
```

memcpy Copies characters from one area of memory to another (no memory overlap allowed).

```
#include <string.h>
void *memcpy(void *s1, const void *s2, size_t n);
```

memmove Copies characters from one area of memory to another.

```
#include <string.h>
void *memmove(void *s1, const void *s2, size_t n);
```

memset Fills a given area of memory with the same character.

```
#include <string.h>
void *memset(void *s, int c, size_t n);
```

mktime Converts a broken-down time into a calendar time.

```
#include <time.h>
time_t mktime(struct tm *timeptr);
```

modf Splits a double number into fractional and integral parts.

```
#include <math.h>
double modf(double value, double *intptr);
```

modff Splits the float argument into fractional and integral parts.

```
#include <mathf.h>
float modff(float value, float *intptr);
```

Move2D Two-dimensional block move.

```
#include <misc.h>
void Move2D(const void *src, void *dst, int width,
            int nrows, int srcwidth, int dstwidth);
```

Move2DNonZero Two-dimensional block move of non-zero bytes.

```
#include <misc.h>
void Move2DNonZero(const void *src, void *dst, int width,
                  int nrows, int srcwidth, int dstwidth);
```

Move2DZero Two-dimensional block move of zero bytes.

```
#include <misc.h>
void Move2DZero(const void *src, void *dst, int width,
               int nrows, int srcwidth, int dstwidth);
```

open Opens a file stream. File handling primitive.

```
#include <iocntrl.h>
int open(char *name, int flags);
```

perror Writes an error message to standard error.

```
#include <stdio.h>
void perror(const char *s);
```

pollkey Gets a character from the keyboard.

```
#include <iocntrl.h>
int pollkey(void);
```

pow Calculates x to the power y.

```
#include <math.h>
double pow(double x, double y);
```

powf Calculates x to the power of y where both x and y are floats.

```
#include <mathf.h>
float powf(float x, float y);
```

printf Writes a formatted string to standard output.

```
#include <stdio.h>
int printf(const char *format, ...);
```

ProcAfter Blocks a process until a specified transputer clock time.

```
#include <process.h>
void ProcAfter(int time);
```

ProcAlloc Allocates the space for and sets up a parallel process.

```
#include <process.h>
Process *ProcAlloc(void (*func)(),
                  int wsize, int param_words, ...);
```

ProcAllocClean Cleans up after a process setup using ProcAlloc.

```
#include <process.h>
void ProcAllocClean(Process *p);
```

ProcAlt Waits for input on one of a number of channels.

```
#include <process.h>
int ProcAlt(Channel *c1, ...);
```

ProcAltList Waits for input on one of a list of channels.

```
#include <process.h>
int ProcAltList(Channel **clist);
```

ProcGetPriority Returns the priority of the calling process.

```
#include <process.h>
int ProcGetPriority(void);
```

ProcInit Sets up a parallel process.

```
#include <process.h>
int ProcInit(Process *p, void (*func)(), int *ws,
             int wsize, int param_words, ...);
```

ProcInitClean Cleans up after a process set up using ProcInit.

```
#include <process.h>
void ProcInitClean(Process *p);
```

ProcJoin Waits for a number of asynchronous processes to terminate.

```
#include <process.h>
int ProcJoin(Process *p1, ...);
```

ProcJoinList Waits for a number of asynchronous processes to terminate.

```
#include <process.h>
int ProcJoinList(Process **p);
```

ProcPar Starts a group of processes in parallel.

```
#include <process.h>
void ProcPar(Process *p1, ...);
```

ProcParam Changes process arguments.

```
#include <process.h>
void ProcParam(Process *p, ...);
```

ProcParList Starts a group of parallel processes.

```
#include <process.h>
void ProcParList(Process **plist);
```

ProcPriPar Starts a pair of processes at high and low priority.

```
#include <process.h>
void ProcPriPar(Process *phigh, Process *plow)
```

ProcReschedule Reschedules a process.

```
#include <process.h>
void ProcReschedule(void);
```

ProcRun Starts a process at the current priority.

```
#include <process.h>
void ProcRun(Process *p);
```

ProcRunHigh Starts a high priority process.

```
#include <process.h>
void ProcRunHigh(Process *p);
```

ProcRunLow Starts a low priority process.

```
#include <process.h>
void ProcRunLow(Process *p);
```

ProcSkipAlt Checks specified channels for ready input.

```
#include <process.h>
int ProcSkipAlt(Channel *c1, ...);
```

ProcSkipAltList Checks a list of channels for ready input.

```
#include <process.h>
int ProcSkipAltList(Channel **clist);
```

ProcStop De-schedules a process.

```
#include <process.h>
void ProcStop(void);
```

ProcTime Determines the transputer clock time.

```
#include <process.h>
int ProcTime(void);
```

ProcTimeAfter Determines the relationship between clock values.

```
#include <process.h>
int ProcTimeAfter(const int time1, const int time2);
```

ProcTimeMinus Subtracts two transputer clock values.

```
#include <process.h>
int ProcTimeMinus(const int time1, const int time2);
```

ProcTimePlus Adds two transputer clock values.

```
#include <process.h>
int ProcTimePlus(const int time1, const int time2);
```

ProcTimerAlt Checks input channels with time out.

```
#include <process.h>
int ProcTimerAlt(int time, Channel *c1, ...);
```

ProcTimerAltList Checks a list of channels for input with time out.

```
#include <process.h>
int ProcTimerAltList(int time, Channel **clist)
```

ProcWait Suspends a process for a specified time.

```
#include <process.h>
void ProcWait(int time);
```

putc Writes a character to a file stream.

```
#include <stdio.h>
int putc(int c, FILE *stream);
```

putchar Writes a character to standard output.

```
#include <stdio.h>
int putchar(int c);
```

puts Writes a line to standard output.

```
#include <stdio.h>
int puts(const char *s);
```

qsort Sorts an array of objects.

```
#include <stdlib.h>
void qsort(void *base, size_t nmem, size_t size,
           int (*compar)(const void *, const void *));
```

raise Forces a pseudo-exception via a signal handler.

```
#include <signal.h>
int raise(int sig);
```

rand Generates a pseudo-random number.

```
#include <stdlib.h>
int rand(void);
```

read Reads bytes from a file. File handling primitive.

```
#include <iocntrl.h>
int read(int fd, char *buf, int n);
```

realloc Changes the size of an object previously allocated using `malloc`, `calloc` or `realloc`.

```
#include <stdlib.h>
void *realloc(void *ptr, size_t size);
```

remove Removes a file.

```
#include <stdio.h>
int remove(const char *filename);
```

rename Renames a file.

```
#include <stdio.h>
int rename(const char *old, const char *new);
```

rewind Sets the file position indicator to the start of a file stream.

```
#include <stdio.h>
void rewind(FILE *stream);
```

scanf Reads formatted data from standard input.

```
#include <stdio.h>
int scanf(const char *format, ...);
```

segread Reads host processor segment registers. MS-DOS only.

```
#include <dos.h>
void segread(struct SREGS *segregs);
```

SemAlloc Allocates and initializes a semaphore.

```
#include <semaphor.h>
Semaphore *SemAlloc(int value);
```

SemInit Initializes an existing semaphore.

```
#include <semaphor.h>
void SemInit(Semaphore *sem, int value);
```

SemSignal Releases a semaphore.

```
#include <semaphor.h>
void SemSignal(Semaphore *sem);
```

SemWait Acquires a semaphore.

```
#include <semaphor.h>
void SemWait(Semaphore *sem);
```

server_transaction Calls any `iserver` function.

```
#include <iocntrl.h>
int server_transaction(char *message, int length,
                      char *reply);
```

set_abort_action Sets/queries action taken by abort.

```
#include <misc.h>
int set_abort_action(int mode);
```

setbuf Controls file buffering.

```
#include <stdio.h>
void setbuf(FILE *stream, char *buf);
```

setjmp Sets up a non-local jump.

```
#include <setjmp.h>
int setjmp(jmp_buf env);
```

setlocale Sets or interrogates part of the program's locale.

```
#include <locale.h>
char *setlocale(int category, const char *locale);
```

setvbuf Defines the way that a file stream is buffered.

```
#include <stdio.h>
int setvbuf(FILE *stream, char *buf, int mode,
            size_t size);
```

signal Defines the way that errors and exceptions are handled.

```
#include <signal.h>
void (*signal(int sig, void (*func)(int)))(int);
```

sin Calculates the sine of the argument.

```
#include <math.h>
double sin(double x);
```

sinf Calculates the sine of a float number.

```
#include <mathf.h>
float sinf(float x);
```

sinh Calculates the hyperbolic sine of the argument.

```
#include <math.h>
double sinh(double x);
```

sinhf Calculates the hyperbolic sine of a float number.

```
#include <mathf.h>
float sinhf(float x);
```

sprintf Writes a formatted string to another string.

```
#include <stdio.h>
int sprintf(char *s, const char *format, ...);
```

sqrt Calculates the square root of the argument.

```
#include <math.h>
double sqrt(double x);
```

sqrtf Calculates the square root of the float argument.

```
#include <mathf.h>
float sqrtf(float x);
```

srand Sets the seed for pseudo-random numbers generated by rand.

```
#include <stdlib.h>
void srand(unsigned int seed);
```

sscanf Reads formatted data from a string.

```
#include <stdio.h>
int sscanf(const char *s, const char *format, ...);
```

strcat Appends one string to another.

```
#include <string.h>
char *strcat(char *s1, const char *s2);
```

strchr Finds the first occurrence of a character in a string.

```
#include <string.h>
char *strchr(const char *s, int c);
```

strcmp Compares two strings.

```
#include <string.h>
int strcmp(const char *s1, const char *s2);
```

strcoll Compares two strings (transformed according to the program's locale).

```
#include <string.h>
int strcoll(const char *s1, const char *s2);
```

strcpy Copies a string into an array.

```
#include <string.h>
char *strcpy(char *s1, const char *s2);
```

strcspn Counts the number of characters at the start of a string which do not match any of the characters in another string.

```
#include <string.h>
size_t strcspn(const char *s1, const char *s2);
```

strerror Maps an error number to an error message string.

```
#include <string.h>
char *strerror(int errnum);
```

strftime Does a formatted conversion of a broken-down time to a string.

```
#include <time.h>
size_t strftime(char *s, size_t maxsize,
                const char *format,
                const struct tm *timeptr);
```

strlen Calculates the length of a string.

```
#include <string.h>
size_t strlen(const char *s);
```

strncat Appends one string onto another (up to a maximum number of characters).

```
#include <string.h>
char *strncat(char *s1, const char *s2, size_t n);
```

strncmp Compares the first n characters of two strings.

```
#include <string.h>
int strncmp(const char *s1, const char *s2, size_t n);
```

strncpy Copies a string into an array (to a maximum number of characters).

```
#include <string.h>
char *strncpy(char *s1, const char *s2, size_t n);
```

strpbrk Finds the first character in one string present in another string.

```
#include <string.h>
char *strpbrk(const char *s1, const char *s2);
```

strrchr Finds the last occurrence of a given character in a string.

```
#include <string.h>
char *strrchr(const char *s, int c);
```

strspn Counts the number of characters at the start of a string which are also in another string.

```
#include <string.h>
size_t strspn(const char *s1, const char *s2);
```

strstr Finds the first occurrence of one string in another.

```
#include <string.h>
char *strstr(const char *s1, const char *s2);
```

strtod Converts the initial part of a string to a double and saves a pointer to the rest of the string.

```
#include <stdlib.h>
double strtod(const char *nptr, char **endptr);
```

strtok Converts a delimited string into a series of string tokens.

```
#include <string.h>
char *strtok(char *s1, const char *s2);
```

strtol Converts the initial part of a string to a long int and saves a pointer to the rest of the string.

```
#include <stdlib.h>
long int strtol(const char *nptr,
               char **endptr, int base);
```

strtoul Converts the initial part of a string to an unsigned long int and saves a pointer to the rest of the string.

```
#include <stdlib.h>
unsigned long int strtoul(const char *nptr,
                        char **endptr, int base);
```

strxfrm Transforms a string according to the locale and copies it into an array (up to a maximum number of characters).

```
#include <string.h>
size_t strxfrm(char *s1, const char *s2, size_t n);
```

system Passes a command to host operating system for execution.

```
#include <stdlib.h>
int system(const char *string);
```

tan Calculates the tangent of the argument.

```
#include <math.h>
double tan(double x);
```

tanf Calculates the tangent of a float number.

```
#include <mathf.h>
float tanf(float x);
```

tanh Calculates the hyperbolic tangent of the argument.

```
#include <math.h>
double tanh(double x);
```

tanhf Calculates the hyperbolic tangent of a float number.

```
#include <mathf.h>
float tanhf(float x);
```

time Reads the current time.

```
#include <time.h>
time_t time(time_t *timer);
```

tmpfile Creates a temporary binary file.

```
#include <stdio.h>
FILE *tmpfile(void);
```

tmpnam Creates a unique filename.

```
#include <stdio.h>
char *tmpnam(char *s);
```

to_host_link Retrieve the channel going to the host.

```
#include <hostlink.h>
Channel* to_host_link( void )
```

to86 Transfers transputer memory to the host. MS-DOS only.

```
#include <dos.h>
int to86(int len, char *here, pcp pointer there);
```

tolower Converts upper-case letter to its lower-case equivalent.

```
#include <ctype.h>
int tolower(int c);
```

toupper Converts lower-case letter to its upper-case equivalent.

```
#include <ctype.h>
int toupper(int c);
```

ungetc Pushes a character back onto a file stream.

```
#include <stdio.h>
int ungetc(int c, FILE *stream);
```

unlink Deletes a file.

```
#include <iocntrl.h>
int unlink(char *name);
```

va_arg Accesses a variable number of arguments in a function definition.

```
#include <stdarg.h>
type va_arg(va_list ap, type );
```

va_end Cleans up after accessing variable arguments.

```
#include <stdarg.h>
void va_end(va_list ap);
```

va_start Initializes a pointer to a variable number of function arguments in a function definition.

```
#include <stdarg.h>
void va_start(va_list ap, parmN);
```

vfprintf An alternative form of `fprintf`. Which accepts a variable argument list in `va_list` form.

```
#include <stdio.h>
int vfprintf(FILE *stream, const char *format,
va_list arg);
```

vprintf An alternative form of `printf`. Which accepts a variable argument list in the form of a `va_list`.

```
#include <stdio.h>
int vprintf(const char *format, va_list arg);
```

vsprintf An alternative form of `sprintf`. Which accepts a variable argument list in the form of a `va_list`.

```
#include <stdio.h>
int vsprintf(char *s, const char *format,
             va_list arg);
```

wcstombs Converts `wchar_t` sequence to multibyte sequence.

```
#include <stdlib.h>
size_t wcstombs(char *s, const wchar_t *pwcs, size_t n);
```

wctomb Converts type `wchar_t` to multibyte character.

```
#include <stdlib.h>
int wctomb(char *s, wchar_t wchar);
```

write Writes bytes to a file. File handling primitive.

```
#include <iocntrl.h>
int write(int fd, char *buf, int n);
```