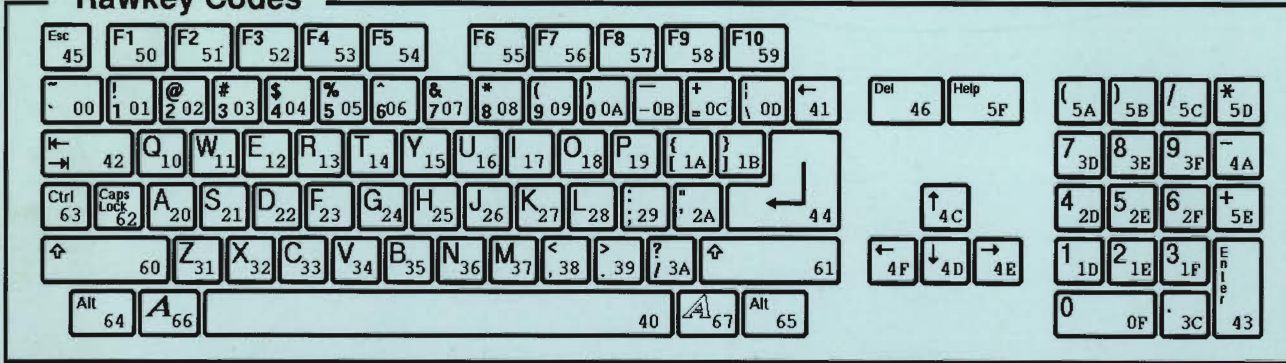
The fastest way to program the Commodore Amiga personal computer in C or assembly. All the details you need constantly, in one place.

AMIGA PROGRAMMER'S QUICK REFERENCE

Compatible with the latest versions of SAS/C and Aztec C.

By Mike McKittrick, D.E. Moseley, Dave Dean, and Richard Lucas

Rawkey Codes



Guru Meditations (System Errors)

SS	GE	XXXX	.ADDRESS

SS: SubSystem IDs

(High bit set indicates unrecoverable.)

00 - CPU trap. Defined by the 68000.
01 - Exec Library
02 - Graphics Library
03 - Layers Library
04 - Intuition Library
05 - Math Library
06 - CList Library
07 - DOS Library
08 - RAM Library
09 - Icon Library
10 - Audio Device
11 - Console Device
12 - Gameport Device
13 - Keyboard Device
14 - Track Disk Device
15 - Timer Device
20 - CIA Resource
21 - Disk Resource
22 - Misc Resource
30 - Bootstrap
31 - Workbench
32 - DiskCopy

GE: General Error Codes

00 - Not applicable
01 - Insufficient memory
02 - MakeLibrary Error
03 - OpenLibrary Error
04 - OpenDevice error
05 - OpenResource Error
06 - I/O Error

XXXX: Specific Error Codes

CPU Traps

02 - Bus Error
03 - Address Error
04 - Illegal Instruction
05 - Divide by Zero
06 - CHK Instruction
07 - TRAPV (Overflow)
08 - Privilege Violation
09 - Instruction Trace
0A - Line A Emulation
0B - Line F Emulation
TRAP 0 ... 15 20 ... 2F

Exec Library

81000001 - CPU trap vector checksum error
81000002 - ExecBase checksum error
81000003 - Library checksum failure
81000004 - No memory to make library
81000005 - Corrupted free memory list
81000006 - No memory for interrupt servers
81000007 - InitAPtr
81000008 - Semaphore corrupt
81000009 - Free twice
8100000A - Bogus exception

Graphics Library

82010001 - Copper display list, no memory
82010002 - Copper instruction list, no memory
82000003 - Copper list too long
82000004 - Copper intermediate list too long
82010005 - Copper list head, no memory
82010006 - Long frame, no memory
82010007 - Short frame, no memory
82010008 - Flood fill, no memory
82010009 - Text, no memory for TmpRas
8201000A - BltBitMap, no memory
8201000B - Region memory
82010030 - MakeVPort
82011234 - GfxNoLCM

Layers Library

83010001 - LayersNoMem

Intuition Library

84000001 - Unknown gadget type
84010002 - CreatePort, no memory
84010003 - Item plane alloc, no memory
84010004 - Sub alloc, no memory
84010005 - Plane alloc, no memory
84000006 - Item box top < RelZero
84010007 - Open screen, no memory
84010008 - OpenScreen's AllocRast, no memory
84000009 - Open sys screen, unknown type
8401000A - Add SW gadgets, no memory
8401000B - Open window, no memory
8400000C - Bad State Return entering Intuition
8400000D - Bad Message received from IDCMP
8400000E - Weird echo causing problem
8400000F - Couldn't open the Console Device Guru explanations

DOS Library

07010001 - No memory at startup
07000002 - EndTask didn't
07000003 - Qpkt failure
07000004 - Unexpected packet received
07000005 - Freevec failed
07000006 - Disk block sequence error
07000007 - Bitmap corrupt
07000008 - Key already free
07000009 - Invalid checksum
0700000A - Disk error
0700000B - Key out of range
0700000C - Bad overlay

TrackDisk Device

14000001 - Calibrate: seek error
14000002 - Delay: error on timer wait

Timer Device

15000001 - Bad request

Disk Resource

21000001 - Get unit: already has disk
21000002 - Interrupt: no active unit

Bootstrap

30000001 - Boot code returned an error

Rawkey Notes

Caps Lock generates a keycode when pressed, not when released. It generates code 62 when the LED is lit, E2 when the LED is extinguished. All other keys generate the codes shown in the diagram when pressed; plus 0x80 when released. Other codes:

68	left mouse button	FA	keyboard overflow
69	right mouse button	FC	keyboard selftest failed
6A	middle mouse button	FF	mouse movement only

Contents

Rawkey Codes 1
Guru Meditations/System Errors 1
Manx C Compiler Flags 2
Lattice C Compiler Flags 2
C Language Reference 4
printf and scanf codes 7
68000 Assembly Instructions 8
Screen and Console Codes 15
ASCII Character Table 16

VIDIA™

Aztec C Compiler Options

`cc` *[options] filename* (wildcards not permitted)

Option	Description
-3	Interpret following options using version 3.6 rules
-5	Interpret following options using version 5.0 rules
-a	Create but don't assemble assembly file
-at	Same as -a, but imbed C code as comments
-bd	Enable stack options
-bs	Produce `SDB` debugger info
-c2	Create 68020 code
-d	Define a symbol for the preprocessor
-fa	Generate code for Amiga IEEE floating point
-ff	Generate code for Motorola Fast Floating Point
-fm	Generate code for Aztec IEEE floating point (default)
-f8	Generate code for Motorola 68881 floating point
-hi	Use precompiled header file
-ho	Write precompiled header file
-i	Set path for include files
-k	Compile according to K&R (Unix Ver. 7) standard
-ma	Force alignment of short and long data items (default)
-mb	Generate public `.begin` symbol (default)
-mc	Use large code memory model
-md	Use large code data model
-me	Align strings on word boundaries
-mm	Put data in code segment
-ms	Put static strings in data segment
-o*name*	Use *name* for output file
-pa	Turn on ANSI preprocessor and trigraphs.
-pb	Make bitfields unsigned by default
-pc	Allow extra characters after `#endif` or `#else` (default)
-pe	Make enums occupy minimum amount of space (default)
-pl	Define integers to be 32 bit (default)
-po	Use Version 3.6 preprocessor
-pp	Make characters unsigned by default
-ps	Define integers to be 16 bit
-pt	Look for trigraphs in the input stream
-pu	Use unsigned preserving rules
-qa	Causes generated prototypes to use _PARMS syntax
-qf	Enable QuikFix
-qp	Generate prototypes for all non-static functions in file
-qq	Prevent startup and error messages from being displayed
-qs	Generate prototypes for all static functions defined in file
-qv	Generate verbose information on memory usage
-r4	Use a4 for register variables
-sa	Enable two-pass assembling for opimizations
-sb	Enable use of built-in in-lines for strcpy(), strcmp(), and strlen()
-sf	Generate an optimized `for(;;)`
-sm	Define the _C_MACROS_ macro to replace some functions with in-lines
-sn	If no local variables defined, no LINK and UNLK instructions generated
-so	A shorthand for -safmnprs
-sp	Delay the popping of arguments until necessary
-sr	Allocate registers based on weighted usage counts
-ss	Use only one copy of duplicate static strings
-su	Same as -sr but allocate to user specifed variables first
-wa	Complain about arguments which do not match the prototype specification
-wl	Shorthand for -waru (roughly equivalent to using `lint`)
-wn	Do not generate warnings on uncasted pointer to pointer conversions
-wo	Make pointer/int conflicts generate warnings rather than errors
-wp	Generate a warning if a function is called without a prototype defined
-wq	Print warnings to the file aztecC.err
-wr	Warn if function return type does not match declared type
-ws	Ignore all warnings
-wu	Warn about unused local variables
-ww	Allows compiler to continue beyond 5 errors

Aztec Linker

`ln` *[options] file1 file2 ...*

Option	Description
+a	Force each module to be aligned on a long word boundary
+c[cdb]	Force loading into chip memory of the specifed program section(s) (Code, Data, Bss)
+f[cdb]	Force the specified program sections to be loaded into fast memory (Code, Data, Bss)
-f*file*	Read command arguments from *file*
-g	Generate an `SDB` debugger .dbg file
+l	Treat the following files as libraries until another +l is found
-l*name*	Link with library *name*.lib. E.g.: -lc means link c.lib
-m	Disable warnings about module symbols overriding library symbols
-o*file*	Generate executable file *file*
+o[*i*]	Place the following object modules in code segment '*i*'
-q	Turns off `SDB` file generation
+q	Disable module-by-module display while linking
+sss +ss +s *val*	Specify 4 different model scatter loading ranging from all modules in one hunk to one hunk per module
-t	Generate an ASCII symbol table file
-w	Generate a `Wack`-readable symbol table. Also usable by `db`
-v	Specify verbose link

SAS/Lattice C Compiler Options

`lc` *[options] file1 file2 ...* (wildcards permitted)

Option		Description
-a		Load specifed area(s) into chip memory. One or more of the following letters must immediately follow -a (in any order):
	b	Bss (uninitialized data)
	c	Code segment
	d	Data segment
-b0		Access all data items by 32 bit address
-b1		Access all data items as a 16 bit offset from register a4
-C		Continue with next file if fatal compilation error detected
-c		Code generation, error reporting and source interpretation flags. One or more of the following letters must immediately follow -c (in any order):
	+	Compatibility mode for Lattice C++
	a	Enable full ANSI checking. Suppress precompiled header files, multiple includes of the same file, and more in order to do this
	c	Allow nested comments
	d	Allow $ character in identifiers
	e	Suppress the printing of source line with warning and error messages
	f	Complain if a function call is encountered without a prototype
	i	Suppress multiple includes of the same file
	k	Enable chip, near, and far keywords even with -ca specified
	l	Allow a pre-ANSI dialect
	m	Allow use multiple character constants; e.g. 'ab'
	n	Allow nesting of #define symbols
	o	Use pre-ANSI preprocessor of previous versions of the compiler
	r	Enable registerized parameter passing
	s	Use only one copy of duplicate static strings
	t	Enable warning messages for structure and unions tags used without being defined
	u	Force all char declarations to be treated as unsigned
	w	No warning for absence of return statements in a function defined as returning an int
	x	Treat all global data declarations as external
-d0		Disable all debugging information
-d1		Enable output of the line number/offset table
-d		Same as -d1
-d2		Output full debugging information for only those symbols and structures referenced by the program.
-d3		Same as -d2 and flush all registers at line boundaries
-d4		Output full debugging information for all symbols declared in the program
-d5		Same as -d4 and flush all registers at line boundaries
-e		Recognize extended character set used in Asian-language applications
-f		Same as -flm (default). If you specify both a floating point style and precision, it must be done in the same -f option. E.g., -ffm.

Copyright 1990 by Vidia. All Rights Reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic or machine-readable form, without prior consent in writing from Vidia.

DISCLAIMER
Every effort has been made to make this product accuate and complete. However, this information is provided "as is," without any warranty of any kind, express or implied. In no event will Vidia, its affiliated companies, its employees, or the authors be liable for any damages, direct, indirect, incidental or consequential, resulting from any use of this information, even if Vidia or the Authors have been advised of the possibility of such damages. This disclaimer shall supersede any verbal or written statement to the contrary.

AMIGA is a registered trademark of Commodore-Amiga Inc. VIDIA, the Vidia film logo, the distinct shapes of the letters I, D, and A, and "Visual Media Tools" are trademarks of Vidia. Aztec C is a trademark of Manx Inc. Lattice is a registered trademark of Lattice Inc. SAS/C is a registered trademark of SAS Institute Inc.

Please send comments or suggestions to: VIDIA, P.O. Box 1180, Manhattan Beach, California 90266.

© Copyright 1990 VIDIA™

Option	Description
-fl	Generate code for Lattice IEEE floating point (default)
-ff	Generate code for Motorola Fast Floating Point
-fi	Generate code for Amiga IEEE floating point
-f8	Generate code for Motorola 68881 floating point
-fs	Treat all declarations as single precision
-fd	Treat all declarations as double precision
-fm	Treat **float** as single precision and **double** as double precision
-g	Big version of compiler generates cross-reference and listing using one or more of the following options:
c	Output a cross reference of all compiler-provided include files
d	Includes all #define symbols in cross reference
e	Display all excluded lines as controlled by #if and #ifdef
h	Include the contents of all include files found in include: directory as they were included by the source program
i	Includes the contents of all user provided include files
m	Display original source line and the line after macro expansion
n	Toggle narrow mode for listing
s	Toggle listing of the input source code
x	Toggles generation of the symbols encountered in the source file
-H	Use the specified precompiled header file
-h	Load specifed area(s) into fast memory. One or more of the following letters must immediately follow -h (in any order):
b	Bss (uninitialized data)
c	Code segment
d	Data segment
-i	Specifies directory in which to search for include files
-j	Alter warning and error reporting. One or more of the following sequences may immediately follow -j as many times as necessary:
-j*n*	Suppress printing for warning number *n*
-j*n*e	Treat any occurrence of warning *n* as an error
-j*n*i	Suppress printing for warning number *n*
-j*n*w	Enables printing of warning *n*
-L	Invoke the linker after successful compilation. -L alone means link without any additional options. One or more of the following letters may immediately follow -L (in any order):
a	Use the ADDSYM option of the linker
c	Use the SMALLCODE option of the linker
d	Use the SMALLDATA option of the linker
f	Specify lcmffp.lib be searched before standard libraries
h	Direct linker to output the hunk portion of the map file
l	Direct linker to include library information in the map file
m	Specify lcm.lib be searched before standard libraries
n	Use the NODEBUG options of the linker
o	Direct linker to include overlay information in the map file
s	Direct linker to produce a symbol listing in the map file
t	Select SMALLCODE SMALLDATA NODEBUG
v	Use the VERBOSE option of the linker
x	Direct linker to include cross reference information in the map file
-l	Align all objects on longword boundaries except chars, short ints, and structures that contain chars and short ints
-M	Compile only files with dates later than object file date
-m0	Generate code which will run on a Motorola 68000 processor
-m1	Generate code which will run on a Motorola 68010 processor
-m2	Generate code which will run on a Motorola 68020 processor
-m3	Generate code which will run on a Motorola 68030 processor
-ma	Generate code which will run on any Motorola 680x0 processor
-m	The following -m options control compilation optimizations. One or more of the following letters may immediately follow -m (in any order):
c	Disables the CORE deferred stack cleanup optimization
r	Disables the automatic registerization of variables
s	Generate optimizations which result in reduction of space instead of time
t	Generate optimizations which result in reduction of time instead of space
-n	Retain only 8 characters for all identifiers
-O	Invoke global optimizer
-o*name*	Output object file to specified *name*; *name* can be a file or directory
-p	Write result of preprocessing input file to output file
-ph	Generate a precompiled header file
-pr	Generate a prototype file. One or more of the following letters may follow:
e	Don't prototype static functions
p	Generate prototypes with _PARMS symbol
s	Generate prototypes for static functions only
-q*dir*	Write quad file to specified directory
-q*n*e	Quit compilation after *n* errors
-q*n*w	Quit compilation after *n* warnings
-q*n*w*m*e	Quit compilation after *n* warnings or *m* errors
-q	Quit after any warning or error
-R*file*	Put output object modules into specified library *file*
-r	Control how the compiler generates subroutine calls. If you specify both a subroutine call method and a parameter passing method, it must be done in the same -r option. E.g. -r0s
-r0	Default all subroutine calls to far
-r1	Default all subroutine calls to near (Default)
-rr	Causes compiler to use registerized parameters
-rs	Causes compiler to use standard stack parameters passing (Default)
-rb	Causes compiler to use registerized parameters but still generate a prologue that handles both styles of parameter passing
-s	Name the code segment "text", the data segment "data", and Bss segment "udata"
-sc=*name*	Use *name* as the name for the program segment
-sd=*name*	Use *name* as the name for the data segment
-sb=*name*	Use *name* as the name for the bss segment
-u	Undefines all preprocessor symbols which are normally predefined by the compiler
-v	Disables generation of stack checking code
-w	Treat all ints as 16 bit short values
-x	Treat all global declarations as externals
-y	All functions load register a4 with linker symbol `LinkerDB`

SAS/Lattice Linker

`blink` [FROM | ROOT] *file(s)* [*to file*] [WITH *file*] [VER *file*] [LIBRARY | LIB *file(s)*] [MAP *file map_options*] [XREF *file*] [*options*]

Option	Description
ADDSYM	Emit HUNK_SYMBOL information for debugging
BATCH	Don't pause after each undefined symbol
BUFSIZE *n*	Set the I/O buffer size to *n*
CHIP	Specify that all hunks are to be put in chip memory
DEFINE *symbol* = *symbol*	Define a symbol to be used during linking
DEFINE *symbol* = *val*	Assign a symbol the specified value
FANCY	Enable the use of printer control characters in the map file
FAST	Specify that all hunks are to be put in fast memory
FASTER	Do-nothing option for ALINK compatibility
FROM *file(s)*	Specify the object files to be linked together
FWIDTH *n*	Specify number of columns in map file
HEIGHT *n*	Specify number of lines on a page in map file
HWIDTH *n*	Specify field width for Hunk names in map file
INDENT *n*	Number of columns to indent on a line in the map file
LIB *file(s)*	Specify the library files to be searched
LIBRARY *file(s)*	Same as LIB
MAP *file options*	Specify what file to send the map to and the options for it
MAXHUNK *n*	Limit the maximum size hunk created when coalescing hunks
ND	Same as NODEBUG
NOALVS	Prevent linker from creating ALVs
NODEBUG	Prevent any HUNK_DEBUG symbol information from appearing in the executable
OVERLAY	Specify the start of an overlay tree
OVLYMGR *file*	Direct linker to use *file* as the Overlay Manager
PRELINK	Output a prelink file for later linking
PLAIN	Turn off the FANCY map file option
PWIDTH *n*	Specify width of program unit name files in map file
QUIET	Output error messages only
ROOT file(s)	Specify the object files that are the primary input to the linker
SC	Same as SMALLCODE
SD	Same as SMALLDATA
SWIDTH *n*	Specify the width of the symbol names field in map file
SMALLCODE	Coalesce all Code segments into one hunk
SMALLDATA	Coalesce all Data segments into one hunk
TO *file*	Specify executable file to create
VER *file*	Same as VERIFY
VERBOSE	Print out the name of each file as it is processed
VERIFY *file*	Send all linker output messages to *file*
WIDTH *n*	Set the maximum line length for the map and cross reference
WITH *file*	Read linker command options from *file*
XREF *file*	Send all cross-reference information to *file*

 © Copyright 1990

C LANGUAGE REFERENCE

In this reference, type fonts and styles have certain meanings. `Typewriter-style` text indicates that the text should appear exactly that way in source code. *Italics* indicate a parameter for which you need to supply a name. Reserved keywords appear in **boldface**. In syntax specifications, square brackets ([ ]) indicate an optional item, and a pipe symbol (|) means select one of the choices.

CHARACTERS

CHARACTER SET

letters	`A B C D E F G H I J K L M N O P Q R S T U V W X Y Z` `a b c d e f g h i j k l m n o p q r s t u v w x y z`
digits	`0 1 2 3 4 5 6 7 8 9`
underscore	`_`
punctuation	`! # % & * ' " ( ) [ ] + - / = , . : ; < > ? \ { } ~ ^`

ESCAPE CODES

code	ASCII	meaning	code	ASCII	meaning
`\"`		"	`\f`	FF	form feed
`\'`		'	`\n`	NL	newline *
`\?`		?	`\r`	CR	carriage return
`\\`		\	`\t`	HT	horizontal tab
`\0`	NUL	string terminator	`\v`	VT	vertical tab
`\a`	BEL	audible tone	`\`*ooo*		octal number
`\b`	BS	backspace	`\x`*hh*		hexadecimal number

* Newlines (`\n`) are represented by LF in Amiga text files, LF in Unix, CR/LF in MS-DOS, and CR on Macintosh. Whitespace characters are: *blank*, `\t`, `\v`, `\n`, `\r`, and `\f`.

PREPROCESSING

INCLUDE DIRECTIVES

Inserts the contents of the specified file into the source code stream.

`#include` *<stdhdr.h>*	< > signifies a standard header; only the include path is searched.
`#include` *"userfile"*	Searches specified path first, then include path.

DEFINE DIRECTIVES

`#define` *MACRO constant*	Pre-processor replaces every instance of the word *MACRO* with the phrase *constant* throughout the source file.
`#define` *macro(param) expr*	Replaces every instance of *macro* with *expr*, substituting the argument *param* wherever it occurs in *expr*.
`#define` *macro(p1,p2,...) expr*	Multiple arguments *p1*, *p2*, etc.
`#undef` *name*	Remove a macro definition.

In a `#define` with macro parameters, *expr* may contain the following constructs:

`#`*param*	Replace `#`*param* by the string literal "*arg*".
token`##`*param*	Replace *param* with the macro argument, then concatenate to form *tokenarg*. *param*`##`*param*, *token*`##`*token*, and *param*`##`*token* are also legal concatenations.

For example, consider the source:

```
#define mac(p)  p + #p + n##p + p##n + n##n + p##p
#define ARG  28
   mac(x)
   mac(ARG)
```

After pre-processing, the source code stream is:

```
x + "x" + nx + xn + nn + xx
28 + "ARG" + nARG + ARGn + nn + ARGARG
```

CONDITIONAL DIRECTIVES

`#if` *expr*	Test *expr*; if true, retain the block following the `#if` directive as part of the source file.
`#ifdef` *expr*	If *expr* is defined, retain the block following `#else` (equivalent to "`#if defined` *expr*").
`#ifndef` *expr*	If *expr* is not defined, retain the block following the directive (equivalent to "`#if !defined` *expr*").
`#else`	If the previous `#if`... or `#elif` expression evaluated to false, retain the block following the `#else` directive.
`#elif` *expr*	If the previous `#if`... or `#elif` expression evaluated to false, test *expr*; if true, retain the block following the `#elif` directive.
`#endif`	Conclude an `#if`... block.

In the descriptions above, *expr* can be

`defined` *MACRO*	True if *MACRO* is defined.
`!defined` *MACRO*	True if *MACRO* is not defined.
expr1 compar expr2	Where *compar* can be: `> < == >= <= !=`
expr1	True if *expr1* is non-zero, false otherwise.

expr can also contain arithmetic operators (`+ - * / %`) and bitwise operators (`& | ^ ~ >> <<`).

OTHER DIRECTIVES

`#line` *linen [file]*	Change the current line number and filename to *linen* and *file*, respectively.
`#error` *text*	Report a diagnostic message, including *text*.
`#pragma` *directive*	Conveys nonstandard information to the translator; if directive is understood, the translator processes it, otherwise it is ignored.
`#`	Null directive; has no effect.
`#asm`	Begin assembly. (Aztec only)
`#endasm`	End assembly. (Aztec only)

PREDEFINED MACROS

`__FILE__`	Name of the current source file, in quotes.
`__LINE__`	Current line number in the source file.
`__DATE__`	Date in the format "Mmm dd yyyy".
`__STDC__`	1 if the compiler is a standard C environment
`__TIME__`	Time in the format "hh:mm:ss".

SYNTAX

ANSI STANDARD KEYWORDS

`asm`	`default`	`for`	`short`	`union`
`auto`	`do`	`goto`	`signed`	`unsigned`
`break`	`double`	`if`	`sizeof`	`void`
`case`	`else`	`int`	`static`	`volatile`
`char`	`enum`	`long`	`struct`	`while`
`const`	`extern`	`register`	`switch`	
`continue`	`float`	`return`	`typedef`	

SAS C KEYWORDS

`chip`	`far`	`huge`	`near`	`__aligned`
`__asm`	`__regargs`	`__stdargs`	`__saveds`	`__interrupt`

AZTEC C KEYWORDS

`pascal`

NAMES

Names must begin with a letter or underscore, and may contain letter, underscore, or digit characters. ANSI C limits names to 31 significant characters (6 for names with external linkage).

CONSTANTS

ddd	decimal integer	*dd.d*	double
0*ooo*	octal integer	*dd*F	float (f or F)
0x*hh*	hexadecimal int (x or X)	*d.de±dd*	double (e or E)
*ddd*L	long integer (l or L)	*dd.d*L	long double (l or L)
*ddd*U	unsigned int (u or U)	'*x*'	character constant
*ddd*LU	unsigned long integer	"*xxxx*"	string literal

STATEMENTS

In the descriptions below, *stmt* may be replaced by either *single_stmt*; or a compound statement of the form { *single_stmt; single_stmt; ...single_stmt;* }.

;	Null statement.
expr;	Evaluate *expr* (typically a function).
`break;`	Immediately exit from innermost **for**, **do**, **while**, or

© Copyright 1990 VIDIA™

switch.

`continue;`	Begin next loop iteration of **for**, **do**, or **while**.
`do` *stmt* `while (` *expr* `);`	Do *stmt* while *expr* is true.
`for (` *init*; *test*; *updt* `)` *stmt*	do *init*; check *test*: if true, do *stmt* then *updt*, if false exit loop; repeat.
`goto` *label* ;	Jump to identifier *label*.
`if (` *expr* `)` *stmt*	If *expr* is true, then do *stmt*.
`if (` *expr* `)` *stmt1* `else` *stmt2*	If *expr* is true, do *stmt1*, else do *stmt2*.
label :	Label marker for **goto** statement.
`return` *expr*;	Exit function with return value *expr*.
`return;`	Exit function with no return value.
`switch (` *c_expr* `) {`	Jump to **case** statement whose *c_expr* matches the switch *c_expr*.
`case` *c_expr* : *stmt*	Execute *stmt* if **switch** and **case** *c_expr*'s match.
`default :` *stmt*	Do *stmt* if no case matched **switch**'s *c_expr*.
`}`	
`while (` *expr* `)` *stmt*	While *expr* is true, do *stmt*.

TYPES

ARITHMETIC TYPES

Character Types	range	storage size
char (SAS)	-128 to 127	1 byte
char (Aztec)	-127 to 127	1 byte
unsigned char	0 to 255	1 byte
signed char (SAS)	-128 to 127	1 byte
signed char (Aztec)	-127 to 127	1 byte
Integer Types		
signed short int (SAS)	-32,768 to 32,767	2 bytes
signed short int (Aztec)	-32,767 to 32,767	2 bytes
signed int (Aztec 3.6)	-32,767 to 32,767	2 bytes
signed int (Aztec 5.0)	-2,147,483,647 to 2,147,483,647	4 bytes
signed int (SAS)	-2,147,483,648 to 2,147,483,647	4 bytes
signed long int (SAS)	-2,147,483,648 to 2,147,483,647	4 bytes
signed long int (Aztec)	-2,147,483,647 to 2,147,483,647	4 bytes
unsigned short int	0 to 65,535	2 bytes
unsigned int (Aztec 3.6)	0 to 65,535	2 bytes
unsigned int (Aztec 5.0)	0 to 4,294,967,295	4 bytes
unsigned int (SAS)	0 to 4,294,967,295	4 bytes
unsigned long int	0 to 4,294,967,295	4 bytes

Bitfields
Bitfields must be members of a structure or union. A bitfield acts as an integer with data range defined by the number of bits you allocate to it. You cannot use the & operator on a bitfield. The types available are:

int	0 to $2^{(n-1)}$
unsigned int	0 to $2^{(n-1)}$
signed int	(part of ANSI C but not available in SAS C or Aztec C)

Enumerated Types
An enumerated variable has a data range defined by listing in the declaration the set of all possible values the variable may assume. The default promotion is to type **int**. The first element has value 0.

Floating Point Types

float	+/-10e-37 to +/-10e+38	4 bytes
long float	+/-2.2e-308 to +/-1.797693e+308	8 bytes
double	+/-2.2e-308 to +/-1.797693e+308	8 bytes
long double	+/-2.2e-308 to +/-1.797693e+308	8 bytes

DERIVED TYPES

Pointers
A pointer is a variable which contains an address of another data object or function. The range of a pointer is the memory address space of the computer. Size is four bytes.

Structures
A structure is a group of related variables, collected so that related information can be handled together. The closest analogy is that of a record in a database. Size depends on the members comprising the structure, and is equal to the sum of the individual variable sizes.

Unions
A union is a group of dissimilar data objects which share storage space. A union is used when the data to be stored may be one of several types which might change dynamically during execution. Storage space required is equal to the largest data type in the union.

Array Types
An array is a group of identical data elements collected together and indexed by an integer address. A two-dimensional array is an array of one-dimensional arrays. Indices may range from 0 to n-1, where n is the dimensioned size of the array.

VOID TYPE
Data storage of type void holds nothing. No space is allocated. A function which does not return a value can be declared as returning type void. A variable declared as pointer to void is a generic pointer.

TYPE QUALIFIERS

`const`	Program cannot alter the value stored.
`volatile`	External agents may alter the value stored.

DECLARATIONS

DECLARATION SYNTAX

[*qual*] [`unsigned|signed`] **`char`** [*] *var* [*arr_decl*] [= *init*] ;
[*qual*] [`unsigned|signed`] [`long|short`] [**`int`**] [*] *var* [*arr_decl*] [= *init*] ;
[*qual*] [`long`] **`float`** [*] *var* [*arr_decl*] [= *init*] ;
[*qual*] [`long`] **`double`** [*] *var* [*arr_decl*] [= *init*] ;
[*qual*] **`struct`** [*tag_name*] { *list_of_decl* } [*] [*var*] [*arr_decl*] [= *init*] ;
[*qual*] **`union`** [*tag_name*] { *list_of_decl* } [*] [*var*] [*arr_decl*] [= *init*] ;
[*qual*] **`enum`** [*tag_name*] { *list_of_elem* } [*] [*var*] [*arr_decl*] [= *init*] ;

where:

qual	can be **volatile** or **const** or both;
var	is the data object name;
arr_decl	is an array size construct: [], [*size*], [*size*][*size*], etc.;
init	is a single constant or a set in the form { i_1, ..., i_n }.
tag_name	is an optional name you can give to the **struct**, **union**, or **enum** type being defined;
list_of_decl	is a list of declarations, some of which may be bitfields, separated by semi-colons (;);
list_of_elem	is a list of all possible values of the enum variable, separated by commas (,).

Bitfield declaration syntax: [*type*] [*var*] : *#bits*
type is either **unsigned** or **int**. ANSI C also permits **signed**. In Lattice C, *type* must be specified.

Enum lists: Individual elements of an enum list may be assigned values. For example: `enum {a=5, b=9} x;`

EXAMPLE DECLARATIONS

```
volatile unsigned char joystick;
char company[]="Vidia(tm)";      /* may omit array size if init list supplied */
long unsigned int a;             /* long, short, signed, and unsigned may be in any order for int */
int signed short b;
short ThreeD[9][9][9];           /* default type is int */
float *pf;                       /* pf points to float storage */
const double pi=3.14159265;
struct bitfield {
    int a:2, b:2;                /* unsigned is default for bitfields */
    unsigned c:6,:1,d:1;         /* :1 is padding between c and d */
    int e:4, :0, f:5;            /* :0 forces field f to next int boundary */
} xbits;
enum primary { red, green, blue };
union { long L; double D; } l_or_d;
```

STRUCT, UNION, AND ENUM DECLARATION STYLES
There are three ways to declare **struct**, **union**, or **enum** data objects.

`struct` *struct_tag* { *list_of_elem* };
`struct` *struct_tag* *var*;

[or] `struct` *struct_tag* { *list_of_elem* } *var*;

[or] `struct` { *list_of_elem* } *var*;

union and **enum** declarations are analogous to the example for **struct** above.

VARIABLE STORAGE CLASSES

Auto

The variable is local to the block, and outside statements cannot alter it, even inadvertently. The variable is created when the block is executed. It disappears and becomes undefined when the block is exited. **auto** is the default storage class for variable declarations within a block.

Static

The variable retains its value between invocations of the block. It behaves in all other ways like storage class **auto**. A **static** variable declared outside a function is known only to functions within the current source file.

Extern

The variable is defined later in the current source code module, or in another source code module. The variable exists before the function is invoked, and retains its value after the function finishes. **extern** is the default storage class for variables declared outside all blocks.

Register

Advice to the compiler that the variable will be used frequently, and that it should be stored in a CPU register if possible. Only **auto** variables and function parameters may have storage class **register**. It is illegal to take the address (&-operator) of a **register** variable.

FUNCTION STORAGE CLASSES

Static

The function is callable only by functions in the current source file.

Extern

The function is known to functions in the current source file, other source files, and other routines linked into the program. **extern** is the default function storage class.

TYPEDEF

Creates a new data type name. The new type name may be used to declare variables in the same way that standard types are declared.

typedef type new_type_name ;

```
typedef int *colortable;
colortable commandscreen;  /* commandscreen is a pointer to int */
```

DIFFICULT DECLARATIONS

Reading C declarations can be difficult. It may help to note that array [ ] and function ( ) operators take precedence over pointer * operators. The following function declarations and variable definitions can be used as templates:

Declaration	Meaning
`int *f[5];`	array of 5 pointers to **int**
`int (*g)[6];`	pointer to array of 6 **int**
`int **h[7];`	array of 7 pointers to pointer to **int**
`int *(*i)[8];`	pointer to array of 8 pointers to **int**
`int (**j)[9];`	pointer to pointer to array of 9 **int**
`char *a(void);`	function(**void**) returning pointer to **char**
`char (*b)(void);`	pointer to function(**void**) returning **char**
`char **c(void);`	function(**void**) returning pointer to pointer to **char**
`char *(*d)(void);`	pointer to function(**void**) returning pointer to **char**
`char (**e)(void);`	pointer to pointer to function(**void**) returning **char**
`void (*f(int))(float);`	function(**int**) returning a pointer to function(**float**) which returns **void**
`void (*(*f)(int))(float);`	pointer to function(**int**) returning a pointer to function(**float**) which returns **void**
`double f(char (*)(int));`	function returning **double** which takes as its only parameter a pointer to a function(**int**) returning **char**

It is frequently more convenient to **typedef** an intermediate type. To define a type, replace the variable or function name with the name of the new type and precede the whole thing with **typedef**. For example,

```
typedef char (*ptr_2_func)(void);
ptr_2_func pf;
```

declares a variable pf which is a pointer to function(**void**) returning **char**, the same type as b above.

DATA OBJECT INITIALIZERS

There are at least two ways to initialize multi-dimensional arrays. One is to know that the last index varies most rapidly. The other is to use nested braces:

```
int array[3][3] = { { 1,2,3 }, { 4,5,6 }, { 7,8,9 } };
```

When initializing a structure array, the member "index" varies more rapidly than the array index:

```
struct style {
    char *name;
    int menuposition;
} textstyle[]={{"plain",4}, {"bold",5}, {"italic",6}};
```

Only the first member of a union may be initialized:

```
union { char c; int i; float f } obj = '\r';
```

FUNCTIONS

PROTOTYPES (FUNCTION DECLARATIONS)

The compiler can perform type checking on function invocations if a function prototype is declared prior to the appearance of function calls in the source. A prototype has the form

return_type func_name (*arg_type* [*arg_name*], ... , *arg_type* [*arg_name*]) ;

If a function expects a varying number of arguments, the last argument must be an ellipsis (...).

FUNCTION DEFINITIONS

There are two ways to write a function definition.

Classic C Syntax

```
type func_name(arg1, arg2, ... )
type arg1;
type arg2; ...
{
    statements
}
```

ANSI C Syntax

```
type func_name(type arg1, type arg2, ... )
{
    statements
}
```

The ANSI syntax is more concise and allows better type checking.

FUNCTION CALLS

A function which returns type void (does not return a value) should be called by placing it by itself in an expression:

function_name(*arg1*, *arg2*, ...);

A function which returns a value may be invoked anywhere an expression of that type is allowed, or by itself as in the syntax diagram above.

Scalar arguments are passed by value; i.e., the function receives a copy of the value, and cannot change the argument in the calling function. Array arguments are passed by address; i.e., the function receives the array's address, and it is able to alter values within the array. Scalar structures and unions are passed by value.

The value returned by a function is supplied by the argument of a **return** statement in the function. The type of the return value is the declared type of the function.

ACCESSING COMMAND LINE ARGUMENTS

Arguments appearing on the command line that invoked the program may be accessed by declaring main as

```
     main(int argc, char *argv[])
(or) main(int argc, char **argv)
```

where *argc* is the number of arguments on the command line, including the program name, and *argv*[ ] holds command line arguments. For example, if the command line is `more prg.doc` then

argc = 2
argv[0] = "more"
argv[1] = "prg.doc"

© Copyright 1990 VIDIA™

EXPRESSIONS

OPERATOR GROUPING

In the table, operators in the same group have the same level of precedence, and are evaluated in the direction indicated. Non-zero is true, zero false.

<table>
<tr><th>Operator</th><th>Effect</th><th>Evaluation Order</th></tr>
<tr><td>x++</td><td>postincrement</td><td>left to right</td></tr>
<tr><td>x--</td><td>postdecrement</td><td></td></tr>
<tr><td>x[i]</td><td>subscript</td><td></td></tr>
<tr><td>x(i)</td><td>function call</td><td></td></tr>
<tr><td>x.i</td><td>member i of structure x</td><td></td></tr>
<tr><td>x->i</td><td>member i of structure pointed to by x</td><td></td></tr>
<tr><td>sizeof(x)</td><td>size of (bytes)</td><td>right to left</td></tr>
<tr><td>++x</td><td>preincremnt</td><td></td></tr>
<tr><td>--x</td><td>predecrement</td><td></td></tr>
<tr><td>&x</td><td>address of</td><td></td></tr>
<tr><td>*x</td><td>value stored at address x</td><td></td></tr>
<tr><td>+x</td><td>plus</td><td></td></tr>
<tr><td>-x</td><td>minus</td><td></td></tr>
<tr><td>~i</td><td>bitwise NOT</td><td></td></tr>
<tr><td>!x</td><td>logical NOT</td><td></td></tr>
<tr><td>(type) x</td><td>type cast</td><td></td></tr>
<tr><td>x * i</td><td>multiply</td><td>left to right</td></tr>
<tr><td>x / i</td><td>divide</td><td></td></tr>
<tr><td>x % i</td><td>remainder</td><td></td></tr>
<tr><td>x + i</td><td>add</td><td>left to right</td></tr>
<tr><td>x - i</td><td>subtract</td><td></td></tr>
<tr><td>x << i</td><td>left shift</td><td>left to right</td></tr>
<tr><td>x >> i</td><td>right shift</td><td></td></tr>
<tr><td>x < i</td><td>less than</td><td>left to right</td></tr>
<tr><td>x <= i</td><td>less than or equal</td><td></td></tr>
<tr><td>x > i</td><td>greater than</td><td></td></tr>
<tr><td>x >= i</td><td>greater than or equal</td><td></td></tr>
<tr><td>x == i</td><td>equals</td><td>left to right</td></tr>
<tr><td>x != i</td><td>not equals</td><td></td></tr>
<tr><td>x & i</td><td>bitwise AND</td><td>left to right</td></tr>
<tr><td>x ^ i</td><td>bitwise exclusive OR</td><td>left to right</td></tr>
<tr><td>x | y</td><td>bitwise inclusive OR</td><td>left to right</td></tr>
<tr><td>x && i</td><td>logical AND</td><td>left to right</td></tr>
<tr><td>x || i</td><td>logical OR</td><td>left to right</td></tr>
<tr><td>z ? x : y</td><td>conditional (x if true, y if false)</td><td>right to left</td></tr>
<tr><td>x = i</td><td>assignment</td><td>right to left</td></tr>
<tr><td>x *= i</td><td>multiply assign</td><td></td></tr>
<tr><td>x /= i</td><td>divide assign</td><td></td></tr>
<tr><td>x %= i</td><td>remainder assign</td><td></td></tr>
<tr><td>x += i</td><td>add assign</td><td></td></tr>
<tr><td>x -= i</td><td>subtract assign</td><td></td></tr>
<tr><td>x <<= i</td><td>left shift assign</td><td></td></tr>
<tr><td>x >>= i</td><td>right shift assign</td><td></td></tr>
<tr><td>x &= i</td><td>bitwise AND assign</td><td></td></tr>
<tr><td>x ^= i</td><td>bitwise exclusive OR assign</td><td></td></tr>
<tr><td>x |= i</td><td>bitwise OR assign</td><td></td></tr>
<tr><td>x , y</td><td>rightmost value</td><td>left to right</td></tr>
</table>

TYPE CONVERSIONS

Promotion

basic type	promoted type
signed char, short, signed bitfield, int	**int**
char, unsigned char, unsigned short, signed short, bitfield, unsigned bitfield	**int** if conversion preserves value, otherwise convert to **unsigned int**
long, signed long	**long**
unsigned long	**unsigned long**

Balancing

In any infix expression *e1 op e2*, the type of the expression is the type of *e1* or *e2*, whichever occurs later in the following list: **int**, **unsigned int**, **long**, **unsigned long**, **float**, **double**, **long double**.

Type Casting

You may coerce the type of any expression by placing a **cast** in front of it. If x is type **int**, then `(float) x` is type **float**. A pointer may be converted into a pointer of any type. Arithmetic types may be converted into any other arithmetic type. Pointers may be coverted only to integer types, and only integer types may be converted to pointers.

STANDARD LIBRARY

FPRINTF, PRINTF, SPRINTF CONVERSION CODES

Conversion specifications have the format:
%[*flags*][*width*][.*precision*][*size*][*type*]

flags:

-	left-adjust result within the field
+	force generation of signs (+ or -)
space	force leading blank for positive number, - for negative
#	for o, x, and X: prefix non-xero output with 0, 0x, and 0X for f, e, and E: force result to contain a decimal point for g and G: force result to contain decimal point and prevent elimination of trailing zeros.

width:

nn	number that specifies the minimum field width
0nn	minimum field width, zero (0) used as padding character
*	get width value from integer in argument list
(none)	use default precision (6 for conversions e and f)

precision: (meaning depends on conversion type)

type c	precision is ignored
types d,o,u,x,X	minimum number of digits
types e,E,f	number of digits after decimal point
types g,G	maximum number of significant digits
type s	maximum number of characters output from string
*	get precision from integer in argument list

size:

(none)	unmodified conversion type
h	short
l	long

type:

c	character
d	integer, output in decimal
e,E	double, output with exponent
f	double
g,G	double, output in most compact form
o	unsigned integer, output in octal
p,P	pointer, output in hexadecimal
s	string
u	unsigned integer, output in decimal
x,X	unsigned integer, output in hexadecimal

FSCANF, SCANF, SSCANF CONVERSION CODES

(white space)	scan past any number of spaces, tabs, or newline characters until non-white space character is read
(other char)	next input character must match or input is aborted

Conversion specifications have the format:
%[*][*width*][*size*][*type*]

*	convert input but do not store result
width	maximum input field width

size:

(none)	standard type
h	**short** conversion
l	**long** conversion

type:

c	single character input into **char** storage
d	decimal input into **int** storage
e,f,g	floating point input into **float** storage
n	number of characters read thus far (into **int** storage)
o	octal input into **int** storage
s	string input into array of **char** storage
u	unsigned decimal input into **unsigned int** storage
x	hexadecimal input into **int** storage

© Copyright 1990

68000 ASSEMBLY LANGUAGE

OPERANDS:

```
An  - address register                     PC  - program counter
Dn  - data register                        SSP - supervisor stack pointer
Rn  - any data or address register         USP - user stack pointer
SR  - status register                      SP  - active stack pointer (A7)
CCR - condition codes (low byte of SR)     n   - any register number (0 to 7)

<ea>   - effective address                 #<data> - immediate data
#xxx   - immediate operand                 <label> - displacement from current PC
```

CONDITION CODE SUMMARY:

```
X  - extend flag                           *  - set or cleared
N  - negative flag                         0  - always cleared
Z  - zero flag                             1  - always set
V  - overflow flag                         -  - not affected
C  - carry flag                            u  - undefined
```

ADDRESSING MODES:

```
Dn        - data register direct
An        - address register direct
(An)      - address register indirect
(An)+     - address register indirect with postincrement
-(An)     - address register indirect with predecrement
d(An)     - address register indirect with displacement  (d= 16-bit signed number)
d(An,Rn)  - address register indirect with index  (d= 8-bit signed number)
Abs.w     - absolute short address (16-bit signed address)
Abs.l     - absolute long address  (24-bit address)
d(PC)     - program counter with displacement  (d= 16-bit signed number)
d(PC,Rn)  - program counter with index  (d= 8-bit signed number)
Imm       - immediate data
```

INSTRUCTION SET:

```
ABCD   Dy,Dx          Size = (Byte)                  X N Z V C
                                                     * u * u *
ABCD   -(Ay),-(Ax)
```

1. Normally the Z-bit is set via programming prior to this instruction.

```
ADD    <ea>,Dn        Size = (Byte, Word, Long)      X N Z V C
                                                     * * * * *
       Dn  An  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  d(PC)  d(PC,Rn)  Imm
```

1. If <ea> is An then word or long word only and condition codes not affected.

```
ADD    Dn,<ea>
       --  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---    ----      ---

ADDA   <ea>,An        Size = (Word, Long)            X N Z V C
                                                     - - - - -
       Dn  An  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  d(PC)  d(PC,Rn)  Imm

ADDI   #<data>,<ea>   Size = (Byte, Word, Long)      X N Z V C
                                                     * * * * *
       Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---    ----      ---

ADDQ   #<data>,<ea>   Size = (Byte, Word, Long)      X N Z V C
                                                     * * * * *
       Dn  An  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---    ----      ---
```

1. If <ea> is An then word or long word only and condition codes not affected.

```
ADDX   Dy,Dx          Size = (Byte, Word, Long)      X N Z V C
                                                     * * * * *
ADDX   -(Ay),-(Ax)
```

1. Normally the Z-bit is set via programming prior to this instruction.

```
AND    <ea>,Dn        Size = (Byte, Word, Long)      X N Z V C
                                                     - * * 0 0
       Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  d(PC)  d(PC,Rn)  Imm
```

Amiga Defines and Typedefs

APTR	STRPTR *
BOOL	short
BPTR	long
BSTR	long
BYTE	char
BYTEBITS	unsigned char
COUNT	short
CPTR	ULONG
DOUBLE	double
FLOAT	float
GLOBAL	extern
IMPORT	extern
LONG	long
LONGBITS	unsigned long
REGISTER	register
SHORT	short
STATIC	static
STRPTR	unsigned char *
TEXT	unsigned char
UBYTE	unsigned char
UCOUNT	unsigned short
ULONG	unsigned long
USHORT	unsigned short
UWORD	unsigned short
VOID	void
WORD	short
WORDBITS	unsigned short

The following types of data require CHIP memory:
- sprite data
- image data
- audio data
- trackdisk device buffers
- pointer data
- icon data

Bitplane Interleaving

Sprite Data	interleaved
Image Data	non-interleaved
Pointer Data	interleaved
Icon Data	non-interleaved
Fill patterns	non-interleaved
Line patterns	non-interleaved

© Copyright 1990 VIDIA™

AND
```
Dn,<ea>

--  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---  ----  ---
```

ANDI
```
#<data>,<ea>     Size = (Byte, Word, Long)     X N Z V C
                                               - * * 0 0
Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---  ----  ---
```

ANDI
```
#xxx,CCR         Size = (Byte)                 X N Z V C
                                               * * * * *
```

ANDI
```
#xxx,SR          Size = (Word)                 X N Z V C
                                               * * * * *
```

If you don't have code to handle a window refresh, and there is the possiblity that your window will need refreshing by you, you MUST set NOCAREREFRESH in NewWindow.Flags.

ASL/ASR
```
Dx,Dy            Size = (Byte, Word, Long)     X N Z V C
                                               * * * * *
```

ASL/ASR
```
#<data>,Dy       Size = (Byte, Word, Long)   (shift = 1 to 8)
```

ASL/ASR
```
<ea>             Size = (Word)               (shift = 1)

--  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---  ----  ---
```

Bcc
```
<label>          Size = (Byte, Word)           X N Z V C
                                               - - - - -
CC  carry clear                 LS  Low or same
CS  carry set                   LT  less than      (s)
EQ  equal                       MI  minus          (s)
GE  greater or equal  (s)       NE  not equal
GT  greater than      (s)       PL  plus           (s)
HI  high                        VC  overflow clear (s)
LE  less or equal     (s)       VS  overflow set   (s)

(s) = signed numbers
```

Handy Addresses

Commodore Business Machines, Inc.
1200 Wilson Drive
West Chester, PA 19380-4231

Vidia
P.O. Box 1180
Manhattan Beach, CA 90266

BCHG
```
Dn,<ea>          Size = (Byte, Long)           X N Z V C
                                               - - * - -
```

BCHG
```
#<data>,<ea>

Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---  ----  ---
```

1. Register- Bit number is in data register in the instruction.
2. Immediate- Bit number in second word of instruction.
3. If <ea> is data register then bit number is modulo 32.
4. If <ea> is memory then bit number is modulo 8.
5. If <ea> is Dn then Long only; all others are byte only.

How To Become a Certified Developer

Write to:

CATS-Information
1200 Wilson Drive
West Chester, PA 19380-4231

Include a self-addressed, stamped, 9" x 12" envelope.

BCLR
```
Dn,<ea>          Size = (Byte, Long)           X N Z V C
                                               - - * - -
```

BCLR
```
#<data>,<ea>

Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---  ----  ---
```

1. Register- Bit number is in data register in the instruction.
2. Immediate- Bit number in second word of instruction.
3. If <ea> is data register then bit number is modulo 32.
4. If <ea> is memory then bit number is modulo 8.
5. If <ea> is Dn then Long only; all others are byte only.

BRA
```
<label>          Size = (Byte, Word)           X N Z V C
                                               - - - - -
```

BSET
```
Dn,<ea>          Size = (Byte, Long)           X N Z V C
                                               - - * - -
```

BSET
```
#<data>,<ea>

Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---  ----  ---
```

1. Register- Bit number is in data register in the instruction.
2. Immediate- Bit number in second word of instruction.
3. If <ea> is data register then bit number is modulo 32.
4. If <ea> is memory then bit number is modulo 8.
5. If <ea> is Dn then Long only; all others are byte only.

Red-Green-Blue Color Values

Color	R	G	B
black	0	0	0
blue	0	0	15
blue green	0	11	11
brick red	13	0	0
cyan	0	15	15
gray	7	7	7
lemon	15	15	0
lime green	11	15	0
magenta	15	1	15
orange	15	9	0
purple	9	1	15
red	15	0	0
white	15	15	15

 © Copyright 1990

BSR

```
<label>          Size = (Byte, Word)              X N Z V C
                                                  - - - - -
```

BTST

```
Dn,<ea>          Size = (Byte, Long)              X N Z V C
                                                  - - * - -
```

BTST

```
#<data>,<ea>

Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  d(PC)  d(PC,Rn)  ---
```

1. Register- Bit number is in data register in the instruction.
2. Immediate- Bit number in second word of instruction.
3. If <ea> is data register then bit number is modulo 32.
4. If <ea> is memory then bit number is modulo 8.
5. Size is Long only if <ea> is Dn.

CHK

```
<ea>,Dn          Size = (Word)                    X N Z V C
                                                  - * u u u
Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  d(PC)  d(PC,Rn)  Imm
```

1. If Dn < 0 or Dn > (ea) then TRAP

CLR

```
<ea>             Size = (Byte , Word, Long)       X N Z V C
                                                  - 0 1 0 0
Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---  ----  ---
```

CMP

```
<ea>,Dn          Size = (Byte, Word, Long)        X N Z V C
                                                  - * * * *
Dn  An  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  d(PC)  d(PC,Rn)  Imm
```

1. If <ea> is 'An' then word or long word only.

CMPA

```
<ea>,An          Size = (Word, Long)              X N Z V C
                                                  - * * * *
Dn  An  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  d(PC)  d(PC,Rn)  Imm
```

1. Word operands are sign extended to 32 bits before the operation is done.

CMPI

```
#<data>,<ea>     Size = (Byte, Word, Long)        X N Z V C
                                                  - * * * *
Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---  ----  ---
```

CMPM

```
(Ay)+,(Ax)+      Size = (Byte, Word, Long)        X N Z V C
                                                  - * * * *
```

DBcc

```
Dn,<label>       Size = (Word)                    X N Z V C
                                                  - - - - -

CC  carry clear              LS  Low or same
CS  carry set                LT  less than      (s)
EQ  equal                    MI  minus          (s)
GE  greater or equal (s)     NE  not equal
GT  greater than     (s)     PL  plus           (s)
HI  high                     VC  overflow clear (s)
LE  less or equal    (s)     VS  overflow set   (s)

(s) = signed numbers
```

DIVS

```
<ea>,Dn          Size = (Word)                    X N Z V C
                                                  - * * * 0
Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  d(PC)  d(PC,Rn)  Imm
```

1. Quotient is in the lower word (least significant 16 bits).
2. Remainder is in the upper word (most significant 16 bits).
3. Division by zero causes a trap.
4. Overflow occurs if quotient larger than a 16-bit signed integer.
5. Sign of the remainder is always the same as the dividend unless the remainder is zero.

DIVU

```
<ea>,Dn          Size = (Word)                    X N Z V C
                                                  - * * * 0
Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  d(PC)  d(PC,Rn)  Imm
```

1. Same notes as DIVS except for 5.

EOR

```
Dn, <ea>         Size = (Byte, Word, Long)        X N Z V C
                                                  - * * 0 0
Dn  --  (An)  (An)+  -(An)  d(An)  d(An,Rn)  Abs.w  Abs.l  ---  ----  ---
```

Note Frequencies

Note-Octave	Hz (cyc/s)
C-3	130.8
C#-3	138.6
D-3	146.8
D#-3	155.6
E-3	164.8
F-3	174.6
F#-3	185.0
G-3	196.0
G#-3	207.7
A-3	220.0
A#-3	233.1
B-3	246.9
C-4	261.6
C#-4	277.2
D-4	293.7
D#-4	311.1
E-4	329.6
F-4	349.2
F#-4	370.0
G-4	392.0
G#-4	415.3
A-4	440.0
A#-4	466.2
B-4	493.9
C-5	523.3
C#-5	554.4
D-5	587.3
D#-5	622.3
E-5	659.3
F-5	698.5
F#-5	740.0
G-5	784.0
G#-5	830.6
A-5	880.0
A#-5	932.3
B-5	987.8

For notes above the fifth octave, multiply the corresponding note in the fifth octave by two. For notes below the third octave, divide third octave notes by two. Middle C is note C-4.

© Copyright 1990 VIDIA™

EORI `#<data>,<ea>` Size = (Byte, Word, Long)

```
X N Z V C
- * * 0 0
Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---
```

EORI `#xxx,CCR` Size = (Byte)

```
X N Z V C
* * * * *
```

EORI `#xxx,SR` Size = (Word)

```
X N Z V C
* * * * *
```

EXG `Rx,Ry` Size = (Long)

```
X N Z V C
- - - - -
```

1. Exchange data registers.
2. Exchange address registers.
3. Exchange a data register and an address register.

EXT `Dn` Size = (Word, Long)

```
X N Z V C
- * * 0 0
```

ILLEGAL Unsized

```
X N Z V C
- - - - -
```

JMP `<ea>` Unsized

```
X N Z V C
- - - - -
-- -- (An) --- --- d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) ---
```

JSR `<ea>` Unsized

```
X N Z V C
- - - - -
-- -- (An) --- --- d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) ---
```

LEA `<ea>,An` Size = (Long)

```
X N Z V C
- - - - -
-- -- (An) --- --- d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) ---
```

LINK `An,#<displacement>` Unsized

```
X N Z V C
- - - - -
```

1. Displacement is signed 16-bit.

LSL/LSR `Dx,Dy` Size = (Byte, Word, Long)

```
X N Z V C
* * * 0 *
```

LSL/LSR `#<data>,Dy` Size = (Byte, Word, Long) (shift = 1 to 8)

LSL/LSR `<ea>` Size = (Word) (shift = 1)

```
-- -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---
```

LSL: C ← Operand ← 0; X ← (copy of bit shifted into C)

LSR: 0 → Operand → C; → X (copy of bit shifted into C)

MOVE `<ea>,<ea>` Size = (Byte, Word, Long)

```
X N Z V C
- * * 0 0
Dn An (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) Imm
```

1. Addressing modes above for source <ea>.
2. Addressing modes below for dest <ea>.
3. If 'An' is source then word and long word only.

```
Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---
```

MOVE `<ea>,CCR` Size = (Word)

```
X N Z V C
* * * * *
Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) Imm
```

1. Source operand upper byte is ignored.

MOVE `<ea>,SR` Size = (Word)

```
X N Z V C
* * * * *
Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) Imm
```

MOVE `SR,<ea>` Size = (Word)

```
X N Z V C
- - - - -
Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---
```

Library Names and Base Addresses

Name (append .library)	Base Address
diskfont	DiskfontBase
dos	DOSBase
exec	SysBase
expansion	ExpansionBase
graphics	GfxBase
icon	IconBase
intuition	IntuitionBase
layers	LayersBase
mathffp	MathBase
mathieeedoubbas	MathIeeeDoubBasBase
mathieeedoubtrans	MathIeeeDoubTransBase
mathtrans	MathTransBase
ramlib	(private)
romboot	(private)
translator	TranslatorBase
version	(private)

Library Versions

0	Any version
30	V1.0 (obsolete)
31	V1.1 (NTSC only-obsolete)
32	V1.1 (PAL only-obsolete)
33	V1.2 (oldest version still in use)
34	V1.3 (adds autoboot)
36	V2.0 (latest)

 © Copyright 1990

MOVE USP,An Size = (Long) X N Z V C: - - - - -

MOVE An,USP

1. Contents of the user stack pointer are transferred to or from the specified address register.

MOVEA <ea>,An Size = (Word, Long) X N Z V C: - - - - -

Dn An (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) Imm

1. Word source operands are signed extended to 32-bits.

MOVEM <reg list>,<ea> Size = (Word, Long) X N Z V C: - - - - -

-- -- (An) --- -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---

MOVEM <ea>,<reg list> Size = (Word, Long)

-- -- (An) --- -(An) d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) ---

1. Word transfers to registers result in a 32-bit sign extended long word into both address and data registers.

MOVEP Dx, d(Ay) Size = (Word, Long) X N Z V C: - - - - -

MOVEP d(Ay),Dx

MOVEQ #<data>,Dn Size = (Long) X N Z V C: - * * 0 0

1. Data is a signed 8-bit number.
2. Data is signed extended to all 32 bits.

MULS <ea>,Dn Size = (Word) X N Z V C: - * * 0 0

Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) Imm

1. Result is a 32-bit signed number.

MULU <ea>,Dn Size = (Word) X N Z V C: - * * 0 0

Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) Imm

1. Result is a 32-bit unsigned number.

NBCD <ea> Size = (Byte) X N Z V C: * u * u *

Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---

1. Normally the Z-bit is set via programming prior to this instruction.

NEG <ea> Size = (Byte, Word, Long) X N Z V C: * * * * *

Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---

NEGX <ea> Size = (Byte, Word, Long) X N Z V C: * * * * *

Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---

1. Normally the Z-bit is set via programming prior to this instruction.

NOP Unsized X N Z V C: - - - - -

NOT <ea> Size = (Byte, Word, Long) X N Z V C: - * * 0 0

Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---

OR <ea>,Dn Size = (Byte, Word, Long) X N Z V C: - * * 0 0

Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) Imm

OR Dn,<ea>

-- -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---

ORI #<data>,<ea> Size = (Byte, Word, Long) X N Z V C: - * * 0 0

Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---

> The output of Text() is clipped to the width of the rastport, even if rendering starts beyond the left edge of it.

> SAS/C formatted I/O functions are not re-entrant.

> amiga.lib formatted I/O functions do not handle floating point.

> Don't ReplyMsg(IntuiMessage*) to a closed window.

> sizeof() is computed at compile time.

© Copyright 1990 VIDIA™

ORI #xxx,CCR Size = (Byte) — X N Z V C: * * * * *

ORI #xxx,SR Size = (Word) — X N Z V C: * * * * *

PEA <ea> Size = (Long) — X N Z V C: - - - - -

-- -- (An) --- --- d(An) d(An,Rn) Abs.w Abs.l d(PC) d(PC,Rn) ---

RESET Unsized — X N Z V C: - - - - -

ROL/ROR Dx,Dy Size = (Byte, Word, Long) — X N Z V C: - * * 0 *

ROL/ROR #<data>,Dy Size = (Byte, Word, Long) (shift = 1 to 8)

ROL/ROR <ea> Size = (Word) (shift = 1)

-- -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---

ROXL/ROXR Dx,Dy Size = (Byte, Word, Long) — X N Z V C: * * * 0 *

ROXL/ROXR #<data>,Dy Size = (Byte, Word, Long) (shift = 1 to 8)

ROXL/ROXR <ea> Size = (Word) (shift = 1)

-- -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---

RTE Unsized — X N Z V C: * * * * *

1. Condition codes set according to the content of the word on the stack.

RTR Unsized — X N Z V C: * * * * *

1. Condition codes set according to the content of the word on the stack.

RTS Unsized — X N Z V C: - - - - -

SBCD Dy,Dx Size = (Byte) — X N Z V C: * u * u *

SBCD -(Ay),-(Ax)

1. Normally the Z bit is set before the start of an operation.

Scc <ea> Size = (Byte) — X N Z V C: - - - - -

CC	carry clear		LS	Low or same	
CS	carry set		LT	less than	(s)
EQ	equal		MI	minus	(s)
GE	greater or equal	(s)	NE	not equal	
GT	greater than	(s)	PL	plus	(s)
HI	high		VC	overflow clear	(s)
LE	less or equal	(s)	VS	overflow set	(s)

(s) = signed numbers

Dn -- (An) (An)+ -(An) d(An) d(An,Rn) Abs.w Abs.l --- ---- ---

STOP #xxx Unsized — X N Z V C: * * * * *

1. Condition codes set according to the immediate operand.

Manx Environment Variables

CCOPTS	options for compiler cc
CCEDIT	QuikFix editor
CLIB	link library directory
CCTEMP	intermediate assembler file directory
INCLUDE	system #include file directory

SAS ASSIGNs

INCLUDE:	system #include file directory
LC:	compiler and utility program directory
LIB:	link library directory
QUAD:	temporary file directory

Putting System Requesters on Your Application's Custom Screen

If your program runs on a custom screen, make sure you tell DOS to put system requesters for your program on your screen, not the Workbench screen. Here's how to do it:

```
#include <libraries/dos.h>
struct Process *MyProcess ;
struct Window *MyWindow ;
APTR OldReqWindow ;
    ...
/* Put system requesters on my window in my screen */
/* Find my process structure */
    MyProcess = (struct Process*)FindTask(NULL) ;
/* Save old value for reseting later */
    OldReqWindow = MyProcess->pr_WindowPtr ;
/* Attach requesters to any */
/* open window (MyWindow) on the screen */
    MyProcess->pr_WindowPtr = (APTR)MyWindow ;
    ...
/* Restore original value */
    MyProcess->pr_WindowPtr = OldReqWindow ;
```

© Copyright 1990

SUB <ea>,Dn — Size = (Byte, Word, Long) — X N Z V C: * * * * *

Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	Abs.w	Abs.l	d(PC)	d(PC,Rn)	Imm

1. If <ea> is 'An' the word or long word only.

SUB Dn,<ea>

--	--	(An)	(An)+	-(An)	d(An)	d(An,Rn)	Abs.w	Abs.l	---	----	---

SUBA <ea>,An — Size = (Word, Long) — X N Z V C: - - - - -

Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	Abs.w	Abs.l	d(PC)	d(PC,Rn)	Imm

1. Word source operands are sign extended to 32 bit quantities before the operation is done.

SUBI #<data>,<ea> — Size = (Byte, Word, Long) — X N Z V C: * * * * *

Dn	--	(An)	(An)+	-(An)	d(An)	d(An,Rn)	Abs.w	Abs.l	---	----	---

SUBQ #<data>,<ea> — Size = (Byte, Word, Long) — X N Z V C: * * * * *

Dn	An	(An)	(An)+	-(An)	d(An)	d(An,Rn)	Abs.w	Abs.l	---	----	---

1. If <ea> is an 'An' then word or long word only and does not affect the condition codes.

SUBX Dy,Dx — Size = (Byte, Word, Long) — X N Z V C: * * * * *

SUBX -(Ay),-(Ax)

1. Normally the Z bit is set before the start of an operation.

SWAP Dn — Size = (Word) — X N Z V C: - * * 0 0

1. N bit is set if most significant bit of the 32-bit result is set.
2. Z bit is set if the 32-bit result is zero. Cleared otherwise.

TAS <ea> — Size = (Byte) — X N Z V C: - * * 0 0

Dn	--	(An)	(An)+	-(An)	d(An)	d(An,Rn)	Abs.w	Abs.l	---	----	---

1. The high order bit of the operand is set.

TRAP #<vector> — Unsized — X N Z V C: - - - - -

1. Sixteen TRAP instruction vectors are available.

TRAPV — Unsized — X N Z V C: - - - - -

TST <ea> — Size = (Byte, Word, Long) — X N Z V C: - * * 0 0

Dn	--	(An)	(An)+	-(An)	d(An)	d(An,Rn)	Abs.w	Abs.l	---	----	---

UNLK An — Unsized — X N Z V C: - - - - -

SAS/C Compiler Limits
1. The maximum value of the constant expression defining the size of a single subscript of an array is two less that the largest unsigned target machine int.
2. The maximum length of an input source lines is 512 characters.
3. The maximum size of a string constant is 256 characters.
4. Macros can have no more than 16 arguments.
5. The maximum length of text substitution for a macro is 512 characters.
6. The maximum level of #include file nesting is 16.

Regular Expressions

(As used by SAS's grep and LSE, and Aztec's grep and Z. Note: This differs from the Unix standard.)

Special Characters

$ ^ * - + [ ] . \

< >	*(Aztec Z only)*
!	*(SAS only)*
\n	newline
\s	space
\b	backspace
\t	tab
\\	backslash
\x*hh*	hex number; *hh* are hex digits

Wildcards

Match any single character

Character Classes

[xyz]	Match any listed character
[a-z]	Match any character in the range
!	Match any except (ONLY when the first char in a character class); e.g.: [!a-z] means any character except in range a to z *(SAS only)*
^	Match any except *(Aztec Z only; otherwise works like !)*

Closures

*	zero or more of
+	one or more of

*A character class may be followed by a closure; e.g. [a-z]**

Anchored Searches

^	Match pattern at start of line; must be first character in pattern
$	Match pattern only at end of line
<	Match at start of word *(Aztec Z only)*
>	Match at end of word *(Aztec Z only)*

Variable Number of Arguments

Here's how to access a variable number of arguments using ANSI C.

```
#include <stdarg.h>

void foo(char *format, ...)
{
   va_list ap ;
   long j1, j2, *jp ;

   va_start(ap, format) ;
   j1 = va_arg(ap, long) ;
   jp = va_arg(ap, long*) ;
   j2 = va_arg(ap, long) ;
   va_end(ap) ;
   /* Now j1 = 1, j2 = 2, and jp holds
   the address of 'ip' in main */
}

void main()
{
   long i1 = 1, i2 = 2, ip = 5 ;

   foo("dummy", i1, &ip, i2) ;
}
```


© Copyright 1990 VIDIA™

ANSI Screen Control Sequences

Backspace:	08 (^H)
Horiz. Tab:	09 (^I) *(8 characters)*
LineFeed:	0A (^J)
Vertical Tab:	0B (^K)
Form Feed:	0C (^L) *(clear screen)*
Return:	0D (^M)
Shift In:	0E (^N) *(use upper 128 characters)*
Shift Out:	0F (^O) *(use lower 128 characters)*
Escape:	1B (^[) *(Esc)*
CSI:	9B

In all cases below, the Control Sequence Introducer (CSI) can be either 9B or Escape followed by '['. M and N are ASCII-encoded numbers. For example, the command to move the cursor up fourteen spaces would be "Esc[14A", or "1B 5B 31 34 41" in hexadecimal. In all codes below numbers are given in hexadecmial with ASCII equivalents in parenthesis to the right. If the parameter in square brackets is omitted, it defaults to one, except where indicated otherwise.

Reset to Initial State:	CSI 63	(b)
Insert N spaces:	CSI [N] 40	(@)
Cursor up N spaces:	CSI [N] 41	(A)
Cursor down N spaces:	CSI [N] 42	(B)
Cursor right N spaces:	CSI [N] 43	(C)
Cursor left N spaces:	CSI [N] 44	(D)
Cursor to column 1, N lines down:	CSI [N] 45	(E)
Cursor to column 1, N lines up:	CSI [N] 46	(F)
Move cursor to row M column N:	CSI [M] 3B [N] 48	(; H)
Erase to end of window:	CSI 4A	(J)
Erase to end of line:	CSI 4B	(K)
Insert a line above this one:	CSI 4C	(L)
Delete this line:	CSI 4D	(M)
Delete N characters:	CSI [N] 50	(P)
Scroll up N lines:	CSI [N] 53	(S)
Scroll down N lines:	CSI [N] 54	(T)
Translate LF to CR/LF:	CSI 32 30 68	(2 0 h)
Don't translate LF:	CSI 32 30 6C	(2 0 l)
Device Status Report (cursor position):	CSI 36 6E	(6 n)

Select Graphic Rendition:

CSI <code> 3B <code> 3B <code> ... 6D (; ; ... m)

where <code> is one of the following:

30 [M]	M= pen 0-7	change foreground color	(3 0-7)
31 [M]	M = pen 0-7	change background color	(4 0-7)

- (no code) reset foreground to 1, background to 0, plain text

00 - plain text
01 - boldface
03 - italics
04 - underscore
07 - inverse video

Amiga Console Control Sequences

Set Page Length to N*:	CSI [N] 74	(t)
Set Line Length to N*:	CSI [N] 75	(u)
Set Left Offset to N*:	CSI [N] 78	(x)
Set Top Offset to N*:	CSI [N] 79	(y)
Window Status Report (window size):	CSI 71	(q)
Enable Scroll:	CSI 3E 31 68	(> 1 h)
Disable Scroll:	CSI 3E 31 6C	(> 1 l)
Autowrap On:	CSI 3F 37 68	(? 7 h)
Autowrap Off:	CSI 3F 37 6C	(? 7 l)

Set Raw Events**:

CSI <code> 3B <code> 3B <code> ... 7B (; ; ... {)

where code is one of the following:

0	- no operation (used internally)
1	- RAW keyboard input (Intuition filters all except select button)
2	- RAW mouse input
3	- event (set whenever your window is active)
4	- pointer position
6	- timer
7	- gadget pressed
8	- gadget released
9	- requester activity
10	- menu numbers
11	- close gadget
12	- window resized
13	- window refreshed
14	- preferences changed
15	- disk removed
16	- disk inserted

Reset Raw Events**:

CSI <code> 3B <code> 3B <code> ... 7D (; ; })

where codes are the same as above.

Make Cursor Invisible:	CSI 30 20 70	(0 *space* p)
Make Cursor Visible:	CSI 20 70	(*space* p)

* If the parameter N is omitted, the value is recalculated for the current window size and font.

** Set adds codes to the stream, Reset deletes codes from the stream.

Console Device Report Stream

In non-raw mode the stream consists of ASCII characters except in the following instances:

Key	Unshifted	Shifted
F1	<CSI>0~	<CSI>10~
F2	<CSI>1~	<CSI>11~
F3	<CSI>2~	<CSI>12~
F4	<CSI>3~	<CSI>13~
F5	<CSI>4~	<CSI>14~
F6	<CSI>5~	<CSI>15~
F7	<CSI>6~	<CSI>16~
F8	<CSI>7~	<CSI>17~
F9	<CSI>8~	<CSI>18~
F10	<CSI>9~	<CSI>19~
Help	<CSI>?~	<CSI>?~
Up	<CSI>A	<CSI>T
Down	<CSI>B	<CSI>S
Right	<CSI>C	<CSI><space>@
Left	<CSI>D	<CSI><space>A

Cursor Position Report: <CSI> <row> 3B <column> 52 (; R)

Window Bounds Report:<CSI> 31 3B 31 3B <# char lines> 3B <# char columns> 72 (1 ; 1 ; r)

Raw Event Report:
<CSI> <class> 3B <subclass> 3B <keycode> 3B <qualifiers> 3B <x> 3B <y> 3B <seconds> 3B <micros> 7C (; ; ; ; ; ; ; |)

Raw Keyboard Report:
<CSI> 31 3B 30 3B <keycode> 3B <qualifier> 3B <prev1> 3B <prev2> 3B <seconds> 3B <microseconds> 7C (1 ; 0 ; ; ; ; ; ; |)

 © Copyright 1990

ASCII Character Table

Binary	Dec	Hex	Char	ASCII	Binary	Dec	Hex	Char	Binary	Dec	Hex	Char	Binary	Dec	Hex	Char
00000000	0	0	^@	NUL null	01000000	64	40	@	10000000	128	80		11000000	192	C0	À
00000001	1	1	^A	SOH start of header	01000001	65	41	A	10000001	129	81		11000001	193	C1	Á
00000010	2	2	^B	STX start of text	01000010	66	42	B	10000010	130	82		11000010	194	C2	Â
00000011	3	3	^C	ETX end of text	01000011	67	43	C	10000011	131	83		11000011	195	C3	Ã
00000100	4	4	^D	EOT end of transmission	01000100	68	44	D	10000100	132	84		11000100	196	C4	Ä
00000101	5	5	^E	ENQ enquiry	01000101	69	45	E	10000101	133	85		11000101	197	C5	Å
00000110	6	6	^F	ACK acknowledge	01000110	70	46	F	10000110	134	86		11000110	198	C6	Æ
00000111	7	7	^G	BEL bell	01000111	71	47	G	10000111	135	87		11000111	199	C7	Ç
00001000	8	8	^H	BS backspace	01001000	72	48	H	10001000	136	88		11001000	200	C8	È
00001001	9	9	^I	HT horizontal tab	01001001	73	49	I	10001001	137	89		11001001	201	C9	É
00001010	10	A	^J	LF linefeed	01001010	74	4A	J	10001010	138	8A		11001010	202	CA	Ê
00001011	11	B	^K	VT vertical tab	01001011	75	4B	K	10001011	139	8B		11001011	203	CB	Ë
00001100	12	C	^L	FF form feed	01001100	76	4C	L	10001100	140	8C		11001100	204	CC	Ì
00001101	13	D	^M	CR carriage return	01001101	77	4D	M	10001101	141	8D		11001101	205	CD	Í
00001110	14	E	^N	SO shift out	01001110	78	4E	N	10001110	142	8E		11001110	206	CE	Î
00001111	15	F	^O	SI shift in	01001111	79	4F	O	10001111	143	8F		11001111	207	CF	Ï
00010000	16	10	^P	DLE data link escape	01010000	80	50	P	10010000	144	90		11010000	208	D0	Ð
00010001	17	11	^Q	DC1 device control 1, XON	01010001	81	51	Q	10010001	145	91		11010001	209	D1	Ñ
00010010	18	12	^R	DC2 device control 2	01010010	82	52	R	10010010	146	92		11010010	210	D2	Ò
00010011	19	13	^S	DC3 device control 3, XOFF	01010011	83	53	S	10010011	147	93		11010011	211	D3	Ó
00010100	20	14	^T	DC4 device control 4	01010100	84	54	T	10010100	148	94		11010100	212	D4	Ô
00010101	21	15	^U	NAK negative acknowledge	01010101	85	55	U	10010101	149	95		11010101	213	D5	Õ
00010110	22	16	^V	SYN synchronous idle	01010110	86	56	V	10010110	150	96		11010110	214	D6	Ö
00010111	23	17	^W	ETB end transmission block	01010111	87	57	W	10010111	151	97		11010111	215	D7	×
00011000	24	18	^X	CAN cancel	01011000	88	58	X	10011000	152	98		11011000	216	D8	Ø
00011001	25	19	^Y	EM end of medium	01011001	89	59	Y	10011001	153	99		11011001	217	D9	Ù
00011010	26	1A	^Z	SUB end of file; substitute	01011010	90	5A	Z	10011010	154	9A		11011010	218	DA	Ú
00011011	27	1B	^[	ESC escape	01011011	91	5B	[	10011011	155	9B		11011011	219	DB	Û
00011100	28	1C	^\	FS file separator	01011100	92	5C	\	10011100	156	9C		11011100	220	DC	Ü
00011101	29	1D	^]	GS group separator	01011101	93	5D	]	10011101	157	9D		11011101	221	DD	Ý
00011110	30	1E	^^	RS record separator	01011110	94	5E	^	10011110	158	9E		11011110	222	DE	Þ
00011111	31	1F	^_	US unit separator	01011111	95	5F	_	10011111	159	9F		11011111	223	DF	ß
00100000	32	20		SP space	01100000	96	60	`	10100000	160	A0		11100000	224	E0	à
00100001	33	21	!		01100001	97	61	a	10100001	161	A1	¡	11100001	225	E1	á
00100010	34	22	"		01100010	98	62	b	10100010	162	A2	¢	11100010	226	E2	â
00100011	35	23	#		01100011	99	63	c	10100011	163	A3	£	11100011	227	E3	ã
00100100	36	24	$		01100100	100	64	d	10100100	164	A4	¤	11100100	228	E4	ä
00100101	37	25	%		01100101	101	65	e	10100101	165	A5	¥	11100101	229	E5	å
00100110	38	26	&		01100110	102	66	f	10100110	166	A6	¦	11100110	230	E6	æ
00100111	39	27	'		01100111	103	67	g	10100111	167	A7	§	11100111	231	E7	ç
00101000	40	28	(		01101000	104	68	h	10101000	168	A8	¨	11101000	232	E8	è
00101001	41	29	)		01101001	105	69	i	10101001	169	A9	©	11101001	233	E9	é
00101010	42	2A	*		01101010	106	6A	j	10101010	170	AA	ª	11101010	234	EA	ê
00101011	43	2B	+		01101011	107	6B	k	10101011	171	AB	«	11101011	235	EB	ë
00101100	44	2C	,		01101100	108	6C	l	10101100	172	AC	¬	11101100	236	EC	ì
00101101	45	2D	-		01101101	109	6D	m	10101101	173	AD	-	11101101	237	ED	í
00101110	46	2E	.		01101110	110	6E	n	10101110	174	AE	®	11101110	238	EE	î
00101111	47	2F	/		01101111	111	6F	o	10101111	175	AF	¯	11101111	239	EF	ï
00110000	48	30	0		01110000	112	70	p	10110000	176	B0	°	11110000	240	F0	ð
00110001	49	31	1		01110001	113	71	q	10110001	177	B1	±	11110001	241	F1	ñ
00110010	50	32	2		01110010	114	72	r	10110010	178	B2	²	11110010	242	F2	ò
00110011	51	33	3		01110011	115	73	s	10110011	179	B3	³	11110011	243	F3	ó
00110100	52	34	4		01110100	116	74	t	10110100	180	B4	´	11110100	244	F4	ô
00110101	53	35	5		01110101	117	75	u	10110101	181	B5	µ	11110101	245	F5	õ
00110110	54	36	6		01110110	118	76	v	10110110	182	B6	¶	11110110	246	F6	ö
00110111	55	37	7		01110111	119	77	w	10110111	183	B7	•	11110111	247	F7	÷
00111000	56	38	8		01111000	120	78	x	10111000	184	B8	¸	11111000	248	F8	ø
00111001	57	39	9		01111001	121	79	y	10111001	185	B9	¹	11111001	249	F9	ù
00111010	58	3A	:		01111010	122	7A	z	10111010	186	BA	º	11111010	250	FA	ú
00111011	59	3B	;		01111011	123	7B	{	10111011	187	BB	»	11111011	251	FB	û
00111100	60	3C	<		01111100	124	7C	|	10111100	188	BC	¼	11111100	252	FC	ü
00111101	61	3D	=		01111101	125	7D	}	10111101	189	BD	½	11111101	253	FD	ý
00111110	62	3E	>		01111110	126	7E	~	10111110	190	BE	¾	11111110	254	FE	þ
00111111	63	3F	?		01111111	127	7F	DEL	10111111	191	BF	¿	11111111	255	FF	ÿ

The Amiga uses the ECMA-94 Latin1 International 8-bit character set.

© Copyright 1990 VIDIA™