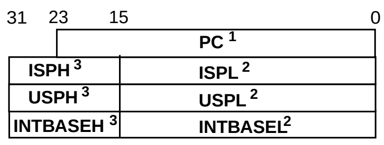
National Semiconductor®

CompactRISC™
CR16C Quick Reference

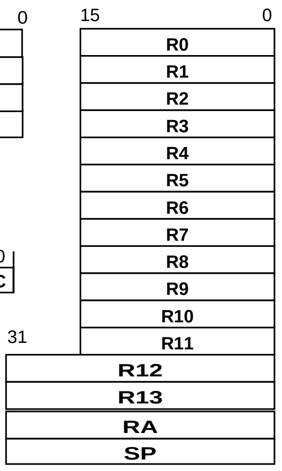
Register Set

Dedicated Address Registers

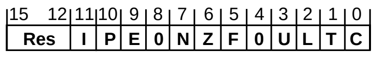


1 The LSB and eight MSBs are always cleared.
2 The LSB is always cleared.
3 The eight MSBs are always cleared.

General-Purpose Registers

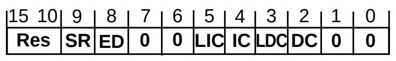


PSR - Processor Status Register



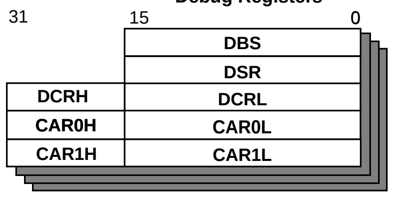
I - Global Interrupt Enable Bit.
P - Trace Pending Bit.
E - Local Interrupt Enable Bit.
N - Negative Bit.
Z - Zero Bit.
F - Flag Bit.
U - User Mode Bit.
L - Low Bit.
T - Trace Bit.
C - Carry Bit.

Configuration Register

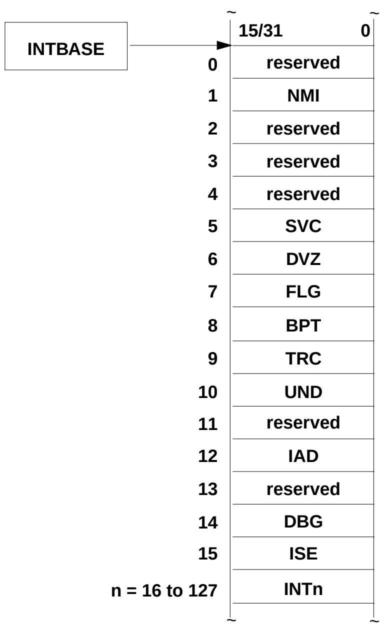


SR - Short Register Bit.
ED - Extended Dispatch Bit.
LIC - Lock Instruction Cache Bit.
IC - Instruction Cache Bit.
LDC- Lock Data Cache Bit.
DC - Data Cache Bit.

Debug Registers



Dispatch Table



0	reserved	
1	NMI	Non-Maskable Interrupt
2	reserved	
3	reserved	
4	reserved	
5	SVC	Supervisor Call Trap
6	DVZ	Divide by Zero Trap
7	FLG	Flag Trap
8	BPT	Breakpoint Trap
9	TRC	Trace Trap
10	UND	Undefined Instruction Trap
11	reserved	
12	IAD	Illegal Address Trap
13	reserved	
14	DBG	Debug Trap
15	ISE	In-System Emulator Interrupt
n = 16 to 127	INTn	Maskable Interrupts

CR16C Instruction Set

Mnemonic	Operands	Description	Flag
Load and Store			
LOADi	disp(Rbase), Rdest disp(RPbase), Rdest abs, Rdest [Rindex]abs, Rdest [Rindex]disp(RPbasex), Rdest	Load (register relative) Load (register pair relative) Load (absolute) Load (absolute index) Load (register relative index)	
LOADD	disp(Rbase), RPdest disp(RPbase), RPdest abs, RPdest [Rindex]abs, RPdest [Rindex]disp(RPbasex), RPdest	Load (register relative) Load (register pair relative) Load (absolute) Load (absolute index) Load (register relative index)	
STORi	Rsrc, disp(Rbase) Rsrc, disp(RPbase) Rsrc, abs Rsrc, [Rindex]abs Rsrc, [Rindex]disp(RPbasex)	Store (register relative) Store (register pair relative) Store (absolute) Store (abolute index) Store (register relative index)	
STORD	RPsrc, disp(Rbase) RPsrc, disp(RPbase) RPsrc, abs RPsrc, [Rindex]abs RPsrc, [Rin-dex]disp(RPbasex)	Store (register relative) Store (register pair relative) Store (absolute) Store (absolute index) Store (register index relative)	
STOR IMM	imm4, disp(Rbase) imm4, disp(RPbase) imm4, abs imm4, [Rindex]abs imm4, [Rin-dex]disp(RPbasex)	Store 4 bit imm (register relative) Store 4 bit imm (register pair rel) Store 4 bit imm (absolute) Store 4 bit imm (absolute index) Store 4 bit imm (register index rel)	
LOADM	imm3	Load 1 to 8 registers (R2-R5, R8-R11) from memory starting at R0	
LOADMP	imm3	Load 1 to 8 registers (R2-R5, R8-R11) from memory starting at (R1, R0)	
STORM	imm3	Store 1 to 8 registers (R2-R5, R8-R11) to memory starting at (R1)	
STORMP	imm3	Store 1 to 8 registers (R2-R5, R8-R11) to memory starting at (R7, R6)	
Moves			
MOVi	Rsrc/imm, Rdest	Move	
MOVD	RPsrc/imm, RPdest	Move Double	
MOVXB	Rsrc, Rdest	Move with sign extension	
MOVZB	Rsrc, Rdest	Move with zero extension	
MOVXW	Rsrc, RPdest	Move with sign extension	
MOVZW	Rsrc, RPdest	Move with zero extension	
Integer Arithmetic			
ADDi	Rsrc/imm, Rdest	Add	CF
ADDUi	Rsrc/imm, Rdest	Add	
ADDCi	Rsrc/imm, Rdest	Add with carry	CF
ADDD	RPsrc/imm, RPdest	Add Double	CF

CR16C Instruction Set (Continued)

Mnemonic	Operands	Description	Flag
MACQW	Rsrc1 Rsrc2, RPdest	Multiply signed Q15: RPdest := RPdest + Rsrc1 * Rsrc2	
MACSW	Rsrc1 Rsrc2, RPdest	Multiply signed and add result: RPdest := RPdest + Rsrc1 * Rsrc2	
MACUW	Rsrc1 Rsrc2, RPdest	Multiply unsigned and add result: RPdest := RPdest + Rsrc1 * Rsrc2	
MULi	Rsrc/imm, Rdest	Multiply: Rdest (8) := Rdest(8) * Rsrc(8)/Imm Rdest(16) := Rdest(16) * Rsrc(16)/Imm	
MULSB	Rsrc, Rdest	Multiply: Rdest (16) := Rdest(8) * Rsrc (8)	
MULSW	Rsrc, RPdest	Multiply: RPdest := RPdest(16) * Rsrc (16)	
MULUW	Rsrc, RPdest	Multiply: RPdest := RPdest(16) * Rsrc (16)	
SUBi	Rsrc/imm, Rdest	Subtract	CF
SUBD	RPsrc/Imm, RPdest	Subtract Double	CF
SUBCi	Rsrc/imm, Rdest	Subtract with carry	CF
Integer Comparison			
CMPi CMPD	Rsrc/imm, Rdest RPsrc/imm, Rpdest	Compare (Rdest – Rsrc/imm) Compare (RPdest - RPsrc/imm)	ZNL ZNL
BEQ0i	Rsrc, disp	Compare Rsrc to 0, branch if EQUAL	Z
BNE0i	Rsrc, disp	Compare Rsrc to 0, branch if NOT-EQUAL	Z
Logical and Boolean			
ANDi ANDD	Rsrc/imm, Rdest RPsrc/imm, RPdest	Logical AND Logical AND Double	
ORi ORD	Rsrc/imm, Rdest RPsrc/imm, RPdest	Logical OR Logical OR Double	
XORi XORD	Rsrc/imm, Rdest RPsrc/imm, RPdest	Logical exclusive OR Logical exclusive OR Double	
Scond	Rdest	Save condition code as boolean	
Shifts			
ASHUi ASHUD	Rsrc/imm, Rdest Rsrc/imm, RPdest	Arithmetic left/right shift Arithmetic left/right shift Double	
LSHi LSHD	Rsrc/imm, Rdest Rsrc/imm, RPdest	Logical left/right shift Logical left/right shift Double	
Bit Operations			
TBIT	Rposition/imm, Rsrc	Test bit in register	F
SBITi	Iposition, disp(Rbase) Iposition, disp(RPbase) Iposition, abs Iposition, [Rindex]abs Ipositiion, [Rin-dex]disp(RPbasex)	Set bit in mem (register relative) Set bit in mem (register pair rel) Set bit in mem (absolute) Set bit in mem (absolute index) Set bit in mem (register index rel)	F

CR16C Instruction Set (Continued)

Mnemonic	Operands	Description	Flag
CBITi	Iposition, disp(Rbase) Iposition, disp(RPbase) Iposition, abs Iposition, [Rindex]abs Ipositiion, [Rindex]disp(RPbasex)	Clear bit in mem (register relative) Clear bit in mem (register pair rel) Clear bit in mem (absolute) Clear bit in mem (absolute index) Clear bit in mem (register index rel)	F
TBITi	Iposition, disp(Rbase) Iposition, disp(RPbase) Iposition, abs Iposition, [Rindex]abs Ipositiion, [Rindex]disp(RPbasex)	Test bit in mem (register relative) Test bit in mem (register pair rel) Test bit in mem (absolute) Test bit in mem (absolute index) Test bit in mem (register index rel)	F
Processor Control			
DI		Disable maskable interrupts	E
EI		Enable maskable interrupts	E
EIWAIT		Enable maskable interrupts and WAIT	E
LPR	Rsrc, Rproc	Load processor register	CTLFZNEPI
LPRD	RPsrc, Rprocd	Load double processor register	CTLFZNEPI
SPR	Rproc, Rdest	Store processor register	
SPRD	Rprocd, RPdest	Store double processor register	
Jumps and Linkage			
Bcond	disp9 disp17 disp25	Conditional branch	
BAL	RPlink, disp25	Branch and link	
BR	disp9 disp17 disp25	Branch	
EXCP	vector	Trap (vector)	
Jcond	RPtarget	Conditional Jump	
JAL	RA, RPtarget RPlink, RPtarget	Jump and link	
JUMP	RPtarget	Jump	
JUSR	RPtarget	Jump and set PSR.U	
RETX		Return from exception	
PUSH	imm, Rsrc imm, Rsrc, RA	Push “imm” number of registers on user stack, starting with Rsrc and possibly including RA	
POP	imm, Rdest imm, Rdest, RA	Restore “imm” number of registers from user stack, starting with Rdest and possibly including RA	
POPRET	imm, Rdest, RA	Restore registers (like POP) and JUMP RA	
Miscellaneous			
CINV	options	Cache Invalidate	
NOP		No operation	
WAIT		Wait for interrupt	

Glossary for CR16C Instruction Set

abs	Absolute address
dispn	displacement of n bits
imm	Immediate value
immn	Immediate value of n bits
Iposition	Bit position, specified as an immediate operand
i	Operand size, B = byte; W = word
R???	Any general-purpose register (R0 .. R12_L, R13_L, RA_L, SP)
Rsrc	Source register R???
Rdest	Destination register R???
Rbase	Base register for relative addressing R???
Rproc	Processor register
Rprocd	Double processor Register
Rindex	R12 or R13 used as an index register holding a base address
Rposition	Bit position register
RP???	Any general-purpose register pair
RPsrc	Source register pair RP???
RPdest	Destination register pair RP???
RPbase	Base register pair for relative addressing RP???
RPbasex	Base register pair for register relative index addressing only: (R1,R0), (R3,R2), (R4,R3), (R5,R4), (R6,R5), (R7,R6), (R9,R8), (R11,R10)
RPlink	Link register pair RP??? holding the address of the next sequential address (return address)
RPtarget	Target register pair RP???. The register holds a code address
RA	Return address register - used in push and pops to determine if the return address register (RA) should be pushed/popped on/from stack

Compiler Flags

Flag	Description
@*optfile*	Read command line arguments from *optfile*.
-g	Generate symbolic information for debugging the source code.
-c	Compile but do not link (do not invoke the linker automatically).
-S	Generate assembly code only.
-n	Embed C source lines as comment in assembly.
-o *filename*	Direct the output to a file named *filename*.
-l*library*	Specify a standard library for the linker.
-O	Optimization - prefer speed over space.
-Os	Optimization - prefer space over speed.
-fshort-enums	Optimize size of enumeration types.
-Oi	Optimize, but treat all global variables and pointer dereferences as volatile.
-ON	Optimization - perform loop unrolling in addition to the default speed optimization.
-f*fixed-REG*	Do not use register (*REG*).
-KFemulation	Link the application with the floating-point emulation library.
-finline -functions	Integrate all simple functions into their callers.
-KB*width*	Align variables to a boundary whose value is the smallest of *width* and the variable alignment requirement. *width* =1, 2(default).
-J*width*	Align members of structures to a boundary whose value is the smallest of *width* and the variable alignment requirement. *width* = 1, 2(default).
-mcr16c	Generate code for CR16C standard mode (default).
-mcr16csr	Generate code for CR16C compatible mode.
-mall-far	All static variables are far and all pointers are pointers to far.
-ansi	Accept strict ANSI C programs only.
-w	No warning diagnostics.
-Q	Error checking only.
-v	Verbose mode (show compilation stages).
-vn	Verbose mode without actually excecuting.
-z	Dump errors and warnings into <file>.err.
-zn*filename*	Dump errors and warnings into *filename*.
-P	Run cpp only, direct output to <file>.i
-I*dir*	Specify directory for include files.
-D*symbol*[=*def*]	Define cpp symbol, which can be assigned to a specific value.
-U*symbol*	Remove initial definition of symbol. Equivalent to `#undef symbol`.

Examples of Compiler Invocation Lines

Invocation Line	Description	Output
`crcc file.c -g`	Compile and produce symbolic debugging information. Invoke the linker using the default libraries.	`cr.x`
`crcc file1.c file2.c -g`	Compile `file1.c` and `file2.c`; produce symbolic debugging information for each file. Invoke the linker using the default libraries.	`cr.x`
`crcc file.c -g -c`	Compile only; produce symbolic debugging information.	`file.o`
`crcc file1.c file2.c -g -c`	Compile `file1.c` and `file2.c`; produce symbolic debugging information for each file.	`file1.o` `file2.o`
`crcc file.c -S`	Compile, but do not assemble; generate assembly code only.	`file.s`
`crcc file.c -S -n`	Compile, but do not assemble; generate assembly code which is annotated by the C source lines.	`file.s`
`crcc file.c -KFemulation`	Compile and link with the floating point emulation library.	`cr.x`
`crcc file.c -g -c -O`	Compile and optimize the code for speed. Generate debugging symbolic information.	`file.o`
`crcc file.c -g -c -Os`	Compile and optimize the code for space. Generate debugging symbolic information.	`file.o`
`crcc file.c -g -c -Idir`	Compile and produce symbolic debugging information. Look for the include files in the `dir` directory.	`file.o`
`crcc file.c -g -c -Ddef1`	Compile and produce symbolic debugging information. Define a preprocessor symbol called `def1`.	`file.o`
`crcc file.c -g -o file.x`	Compile, link and produce symbolic debugging information. The executable file is called `file.x`.	`file.x`
`crcc file.s`	Assemble and link the assembly file.	`cr.x`
`crcc file1.s file2.c`	Assemble `file1.s`, compile `file2.c` and link the two generated object files with the default libraries.	`cr.x`
`crcc file1.s file2.c file3.o`	Assemble `file1.s`, compile `file2.c` and link the two generated object files and `file3.o` with the default libraries.	`cr.x`

Libraries used by the Compiler

Libraries used by default	libstart, libc, libd
Libraries used when Floating-Point emulation is required (i.e., when `-KFemulation` command line option is used)	libstart, libc, libhfp

Assembler Flags

Flag	Description
-g	Produce line number information for symbolic debugging.
-L[*filename*]	Produce listing information. Listing is redirected to *filename*, if specified, or to the standard output, if not.
-MO	Invoke only macro-processing phase.
-MP[*filename*]	Print the macro procesing output. Output is redirected to *filename*, if specified, or to the standard output, if not.
-D*name*[**=***def*]	Define cpp symbol, which can be assigned to a specific value. The `-c` option must precede this option.
-U*name*	Undefine cpp symbol. The `-c` option must precede this option.
-o *objectfile*	Name the output object file, *objectfile*.
-w	Suppress assembly warning messages.
-c	Run the C compiler pre-processor (cpp) on the input of the assembler.
-I*dir*	Search for the include files in the *dir* directory. The `-c` option must precede this option.
-n	Disable displacement size optimization.
@*optfile*	Read input for the invocation line from *optfile*.
-mcr16c	Generate code for CR16C standard mode (default).
-mcr16csr	Generate code for CR16C compatible mode.
-z	Dump errors and warnings into <file>.err.
-zn*filename*	Dump errors and warnings into *filename*.

Examples of Assembler Invocation Lines

Invocation Line	Description	Output
`crasm file.s`	Assemble the file.	file.o
`crasm file.s -g`	Assemble the file; add line number information for symbolic debugging.	file.o
`crasm file.s -Lfile.lis`	Assemble the file; generate a listing file.	file.o file.lis
`crasm file.s -MO -MP file.mac`	Invoke macro-processing only, and direct the output to `file.mac`.	file.mac
`crasm file.s -Idir -g -c`	Assemble the file; add line number information for debugging. Look for include files in the `dir` directory.	file.o

Linker Flags

Flag	Description
`-m`	Generate a map file.
`-d` *`filename`*	Link the application using a linker directive file, *`filename`*.
`-l`*`x`*	Add the standard library, *`libx.a`* to the list of input libraries.
`-L`*`dir`*	Search for standard libraries in *`dir`* directory.
`-e` *`symbol`*	Specify the program entry point symbol (default: `start`).
@*`optfile`*	Read input for the invocation line from *`optfile`*. This includes linker flags as well as files/library list.
`-o` *`filename`*	Direct the linking output (CompactRISC executable file) to a file, *`filename`*.
`-f` *`fill_value`*	Fill output section gaps with *`fill_value`*.
`-z`	Dump errors and warnings into <file>.err.
`-zn`*`filename`*	Dump errors and warnings into *`filename`*.

Examples of Linker Invocation Lines

Invocation Line	Description	Output
`crlink file1.o file2.o -lstart -lc -ld`	Link the object files with the CompactRISC standard libraries.	cr.x
`crlink file1.o file2.o -lstart -lc -ld -o file.x`	Link the objects files with the CompactRISC standard libraries. Name the output executable file, `file.x`.	file.x
`crlink file1.o file2.o -lstart -lc -ld -m > map`	Link the objects files with the CompactRISC standard libraries. Produce a map file.	cr.x map
`crlink file1.o file2.o -lstart -lc -ld -d linker.def`	Link the objects files with the CompactRISC standard libraries. Use the specified linker directive file (`linker.def`)	cr.x
`crlink file1.o file2.o -lstart -lc -lhfp`	Link the object files with `libstart`, `libc`, and the floating-point emulation library, `libhfp`.	cr.x

Debugger Command Lines

Definition	Syntax
break	
Lists the hardware breakpoints	
Add a hardware breakpoint	[-t] [-w] <brkaddr_list>[,c=<RLexp>] [,o=<occ_cnt>][,q=<qualifier>] [,p=<brkaddr>[,x=<qualifier>]]
Remove, disable, enable entry hardware breakpoint	[-r | -d | -e]%<id> | *
cd	
Display the working directory for creating/reading files.	
Change the working directory for creating/reading files.	[<path>]
comm	
Sets the communcation channel to communicate with an ADB, or a simulator running on the host platform.	comm [-s] [-s <communication_channel_name>]
core	
Displays the current CPU core	
Sets the current CPU core	core [CR16C | CR16CSR]
debug	
Select an executable file (COFF) to debug	<file_name>
debugmode	
Select debugging mode	[-e | -d] [startup | exitcode]
find	
Find a pattern in memory	[-a | -b | -c | -w | -f | -p | -l | -i] <value>, <addr_range>
findsrc	
Find a string in the current source file	[-f | -b | -n] [<string>] [,<file_name>]
go	
Execute the user program	[-c] [<from_addr>][//<end_addr>]
list	
List memory	-m [h | o | d] [b | c | w | f | p | l | i] <addr_range>
List source file	<qualified_lineno>
modify	
Modify memory	[-b | -c | -w |-f | -p | -l] <addr_range>[,<value>[,value]]
Modify string	-a <string_pointer>,<string>
Modify register	%<reg_name>,<value>
next	
Execute the next **x** source lines	[-n < x>]
nextins	

Debugger Command Lines (Continued)

Definition	Syntax
Execute the next **x** assembly instructions	**[**-n < x>**]**
quit	
Quit the debugger	
radix	
Set radix for output display	**[**8 **|** 10 **|** 16**]**
reset	
Reset the application and the target board	
saveconfig	
Save the current debugger configuration in a file (default crdb.env)	**[**<*file_name*>**]**
savestate	
Save the current debugging state in a file (default crdb.ctx)	**[**<*file_name*>**]**
setstate	
Restore debugging state, as saved with savestate command (default crdb.env)	<*file_name*>
softbreak	
Lists the software breakpoints	
Add a software breakpoint	**[**-t**]** <*softbreak_list*> **[**,<c=<*RLexp*>**]** **[**,o=<*occ_cnt*>**]**
Remove, disable, enable a software breakpoint	**[**-r **|** -d **|** -e**]**%<*id*> **|** *
srcmode	
Set the source file window display mode	**[**-s **|** -m**]**
srcpath	
Set a directory path for the source files	<*pathname_list*>
Remove a directory path	-r **[**<*path*> **|** *****]**
step	
Step **x** source lines	**[**-n <x>**]**
stepins	
Step **x** assembly instructions	**[**-n <x>**]**
symbol	
Display symbol info	**[*** **|** <*pattern*>*****]**
Display local symbols	-l **[*** **|** <*pattern*>*****]**
Display symbol tag	-t **[**<*datatype*>/<*tagname*>**|**<*symbol-name*> **|** *****]**
Display module symbols	-f <*qualified_modulename*>
Display global symbols	-g **[**<*pattern*>*****]**

Debugger Command Lines (Continued)

Definition	Syntax
`sync`	
Display the line corresponding to the PC	
`verbose`	
Display all the communication data between the debugger and the target	
`view`	
Display data	<*expression*> **[**,<*print_specifier*>**]**
Display register	%<*reg_name*> **[**,<*print_specifier*>**]**
`watch`	
Select a variable to be displayed automatically in the watch variable window	<*expression*> **[**,<*print_specifier*>**]**
Remove, disable, enable selection	**[**-r **|** -d **|** -e**]** %<*id*>**|** *
`where`	
Show the program context, at any point	**[**-c **|** -v**]** **[**<*func_symbol*>**[**@<*symbol*>**]]**

Glossary for Debugger Commands

General	
`$$`	Macro argument
`*`	All items
`-r`	Remove an entry point from a list
`-d`	Disable
`-e`	Enable
`$b`	Address of the absolute beginning of the function (the prologue)
`$c`	Address of the first instruction after the prologue
`$e`	Address of the first instruction of the epilogue
`$x`	Address of the last instruction (RETURN) of the epilogue

Data Type	
`-a`	ASCII string
`-b`	Byte
`-c`	Char
`-s`	String
`-w`	Word
`-f`	Float
`-p`	Pointer
`-l`	Long
`-i`	Assembly instruction
`-x`	Unsigned hexadecimal
`-d`	Signed decimal
`-u`	Unsigned decimal
`-o`	Octal

Glossary for Debugger Commands (Continued)

Break	
`-t`	Temporary breakpoint. This breakpoint is deleted after it occurs
`-w`	Watchpoint.
`brkaddr_list`	List of breakpoints
`c=RLexp`	Relational or logical expression which must be evaluated as true for the breakpoint condition
`o=occ_cnt`	Number of times the address is referenced before execution is interrupted.
`q=qualifier` `x=qualifier`	Type of access (default = A) E - pc-match. Code at this address is executed A - Data at this address is accessed: read or write R - Data at this address is read W -Data at this address is written
`p=prerequisite break`	Address of the first breakpoint in a sequence type breakpoint.

Comm	
`-s`	Sets the debugger to communicate with a simulator running on the host platform.
`-s <name>`	Sets the debugger to communicate with an ADB using the communication channel, `<name>`.
`-c`	Closes the current communication channel.
`-l`	Displays all available communication names.
`-f <name>`	Forces DBGCOM to open the communication channel, `<name>` .

findsrc	
`-f`	Forward search
`-b`	Backward search
`-n`	Next find in the same direction

go	
`-c`	The debugger does not stop at breakpoint just update the windows

srcmode	
`-s`	Source-only display (for C program, C lines only)
`-m`	Mixed mode (for C program, C and assembly lines)

where	
`-c`	Stack history
`-v`	Local variables

ASCII Character Set

Char	7-bit Hex Number	Char	7-bit Hex Number	Char	7-bit Hex Number	Char	7-bit Hex Number
NUL	00	Space	20	@	40	‘	60
SOH	01	!	21	A	41	a	61
STX	02	"	22	B	42	b	62
ETX	03	#	23	C	43	c	63
EOT	04	$	24	D	44	d	64
ENQ	05	%	25	E	45	e	65
ACK	06	&	26	F	46	f	66
Bell	07	’	27	G	47	g	67
BS	08	(	28	H	48	h	68
HT	09	)	29	I	49	i	69
LF	0A	*	2A	J	4A	j	6A
VT	0B	+	2B	K	4B	k	6B
FF	0C	,	2C	L	4C	l	6C
CR	0D	-	2D	M	4D	m	6D
SO	0E	.	2E	N	4E	n	6E
SI	0F	/	2F	O	4F	o	6F
DLE	10	0	30	P	50	p	70
DC1	11	1	31	Q	51	q	71
DC2	12	2	32	R	52	r	72
DC3	13	3	33	S	53	s	73
DC4	14	4	34	T	54	t	74
NAK	15	5	35	U	55	u	75
SYN	16	6	36	V	56	v	76
ETB	17	7	37	W	57	w	77
CAN	18	8	38	X	58	x	78
EM	19	9	39	Y	59	y	79
SUB	1A	:	3A	Z	5A	z	7A
ESC	1B	;	3B	[	5B	{	7B
FS	1C	<	3C	\	5C	|	7C
GS	1D	=	3D	]	5D	}	7D
RS	1E	>	3E		5E	~	7E
US	1F	?	3F	¨	5F	DEL	7F