Register Set

Dedicated Address Registers

20 16	15 0
PC [1]	
0 0 0 0 0	ISP [2]
INTBASEH [4]	INTBASEL [3]

[1] Both programming models: The LSB is always cleared.
Small programming model: The five MSBs are always cleared
[2] The five MSBs, and the LSB are always cleared.
The five MSBs can not be accessed.
[3] The LSB is always cleared.
[4] Small programming model: register is cleared.

General-Purpose Registers

15 0
R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13/ERA
RA
SP

PSR - Processor Status Register

15 12	11	10	9	8	7	6	5	4	3	2	1	0
Res	I	P	E	0	N	Z	F	0	0	L	T	C

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

Debug Registers

20 15 0
DCR
DSR
CARH CARL

Configuration Register

15 9	8	7 0
Res	ED	Res

Dispatch Table

	15/31 0	
INTBASE →		
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	reserved	
13	reserved	
14	DBG	Debug Trap
15	ISE	In-System Emulator Interrupt
n = 16 to 127	INTn	Maskable Interrupts

CR16B Instruction Set

Mnemonic	Operands	Description	Flag
Load and Store			
LOADi	disp(Rbase), Rdest	Load (register relative)	
	disp(Rpair+1, Rpair), Rdest	Load (far-relative)	
	abs, Rdest	Load (absolute)	
STORi	Rsrc, disp(Rbase)	Store (register relative)	
	Rsrc, disp(Rpair +1, Rpair)	Store (far-relative)	
	Rsrc, abs	Store (absolute)	
	sm_imm, 0(Rbase) sm_imm, disp(Rbase) sm_imm, abs	Store small immediate in memory Rbase = (R0, R1, R8, R9)	
	sm_imm, abs	Store small immediate in memory Rbase = (R0, R1, R8, R9)	
LOADM	imm	Load 1 to 4 registers (R2-R5) from memory starting at address in R1, according to *imm* count value	
STORM	imm	Store 1 to 4 registers (R2-R5) to memory starting at address in R1, according to *imm* count value	
Moves			
MOVi	Rsrc/imm, Rdest	Move	
MOVX	Rsrc, Rdest	Move with sign extension	
MOVZ	Rsrc, Rdest	Move with zero extension	
MOVD	Imm, Rdest+1, Rdest	Move 21-bit immediate to register pair	
Integer Arithmetic			
ADDi	Rsrc/imm, Rdest	Add	CF
ADDUi	Rsrc/imm, Rdest	Add	
ADDCi	Rsrc/imm, Rdest	Add with carry	CF
MULi	Rsrc/imm, Rdest	Multiply: Rdest (8) := Rdest(8) * Rsrc(8) Rdest(16) := Rdest(16) * Rsrc(16)	
MULSB	Rsrc, Rdest	Multiply: Rdest (16) := Rdest(8) * Rsrc (8)	
MULSW	Rsrc, Rdest	Multiply: (Rdest+1, Rdest):= Rdest(16) * Rsrc (16)	
MULUW	Rsrc, Rdest	Multiply: (Rdest+1, Rdest):= Rdest(16) * Rsrc (16); (Rsrc = R0, R1, R8 and R9)	
SUBi	Rsrc/imm, Rdest	Subtract (Rdest := Rdest – Rsrc)	CF
SUBCi	Rsrc/imm, Rdest	Subtract with carry (Rdest := Rsrc – PSR.C)	CF
Integer Comparison			
CMPi	Rsrc/imm, Rdest	Compare (Rdest – Rsrc)	ZNL
BEQ0i	Rsrc, disp	Compare Rsrc to 0, branch if EQUAL Rsrc = (R0, R1, R8, R9 only}	Z
BNE0i	Rsrc, disp	Compare Rsrc to 0, branch if NOT-EQUAL Rsrc = (R0, R1, R8, R9 only)	Z

BEQ1i	Rsrc, disp	Compare Rsrc to 1, branch if EQUAL Rsrc = (R0, R1, R8, R9 only)	Z
BNE1i	Rsrc, disp	Compare Rsrc to 1, branch if NOT-EQUAL Rsrc = (R0, R1, R8, R9 only)	Z
Logical and Boolean			
ANDi	Rsrc/imm, Rdest	Logical AND	
ORi	Rsrc/imm, Rdest	Logical OR	
Scond	Rdest	Save condition code as boolean	
XORi	Rsrc/imm, Rdest	Logical exclusive OR	
Shifts			
ASHUi	Rsrc/imm, Rdest	Arithmetic left/right shift	
LSHi	Rsrc/imm, Rdest	Logical left/right shift	
Bits			
TBIT	Rposition/imm, Rsrc	Test bit in register	F
SBITi	Iposition, 0(Rbase) Iposition, disp16(Rbase) Iposition, abs	Set a bit in memory; Rbase = {R0, R1, R8, R9}	F
CBITi	Iposition, 0(Rbase) Iposition, disp16(Rbase) Iposition, abs	Clear a bit in memory Rbase = {R0, R1, R8, R9}	F
TBITi	Iposition, 0(Rbase) Iposition, disp16(Rbase) Iposition, abs	Test a bit in memory Rbase = {R0, R1, R8, R9}	F
Processor Control			
DI		Disable maskable interrupts	E
EI		Enable maskable interrupts	E
EIWAIT		Enable interrupts and wait for interrupts	E
LPR	Rsrc, Rproc	Load processor register	CTLFZNEPI
SPR	Rproc, Rdest	Store processor register	
Jumps and Linkage			
Bcond	disp	Conditional branch	
BAL	Rlink, disp17 (Rlink+1, Rlink), disp21	Branch and link to a small address Branch and link to a large address	

CR16B Instruction Set (Continued)

Mnemonic	Operands	Description	Flag
PUSH	imm, Rsrc	Push “imm” number of registers on user stack, starting with Rsrc	
POP	imm, Rdest	Restore “imm” number of registers from user stack, starting with Rdest	
POPRET	imm, Rdest	Restore registers (like POP) and perform JUMP RA or JUMP (RA,ERA), depending on memory model	
Miscellaneous			
NOP		No operation	
WAIT		Wait for interrupt	

Glossary for CR16B Instruction Set

abs	Absolute address
disp16	16-bit displacement
disp(Rbase)	Relative addressing; Address = disp + (Rbase)
imm	Immediate value
Iposition	Bit position, specified as an immediate operand
large address	Address in the space of 0 to 2 Mbytes
Rbase	Base register. Any general-purpose register (R0 - R13, ERA, RA, SP), holding the base address of a memory variable
Rdest	Destination Register. Any general-purpose register (R0 - R13, ERA, RA, SP)
Rdest(8)	The LSB (8-bits) of any Destination Register (R0 - R13, ERA, RA, SP)
Rdest(16)	The complete 16-bits of any Destination Register (R0 - R13, ERA, RA, SP)
Rlink	Link Register. Any general-purpose register (R0 - R13, ERA, RA, SP), holding the address of the next sequential instruction, used as a return address.
Rposition	Bit position, stored in any general-purpose register (R0 - R13, ERA, RA, SP)
Rproc	Processor registers (PC, PSR, ISP, INTBASEH/L)
Rsrc	Source Register. Any general-purpose register (R0 - R13, ERA, RA, SP)
Rsrc(8)	The LSB (8-bits) of any Source Register (R0 - R13, ERA, RA, SP)
Rsrc(16)	The complete 16-bits of any Source Register (R0 - R13, ERA, RA, SP)
Rsrcr	A restricted subset of the general-purpose register set (R0, R1, R8, R9)
Rtarget	Target Register. Any general-purpose register (R0 - R13, ERA, RA, SP), holding the JUMP/JAL target address
sm_imm	Small immediate - 4-bit data
small address	Address in the space of 0 to 128 Kbytes

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.
-l*library*	Specify a standard library for the linker.
-O	Optimization - prefer speed over space.
-Os	Optimization - prefer space over speed.
-sbrel	Special space optimization - use the efficient static base relative access mode for selected variables. R12 is used as the base register. The total size of these variables is limited to 32 bytes. Use `#pragma sbrel(var)` to select these variables.
-fshort-enums	Optimize size of enumeration type
-Oi	Optimize, but treat all global variables as volatile.
-ON	Optimization - perform loop unrolling in addition to the default speed optimization.
-f*fixed-REG*	Do not use register (*REG*). e.g., `-ffixed-r8`: does not use `r8`.
-KFemulation	Link the application with the floating-point emulation library.
-finline -functions	Integrate all simple functions into their callers.
-KB*width*	Set target buswidth. Guide the compiler to align all stack variables in the most efficient manner for a given bus width. *width* =1, 2, 4.
-J*bytes*	Align the members of all structures as specified by *bytes*: *bytes* = 1, 2, 4.
-msmall	Generate code for CR16B small model (default).
-mlarge	Generate code for CR16B large model.
-mcr16a	Generate CR16A-compatible code.
-Wimplicit	Warn about functions with missing prototype declaration.
-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*.
-E	Run cpp only, direct output to stout
-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. Equivalent to `#define symbol def`.
-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 -Os -sbrel`	Compile and optimize the code for space. Use the static base relative access mode. Produce 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. Equivalent to `#define name def`.
-U*name*	Remove initial definition of a predefined name. Equivalent to `#undef name`.
-o *objectfile*	Name the output object file, *objectfile*.
-w	Suppress assembly warning messages.
-c	Run the C compiler pre-processor (cpp). The cpp output is the input of the asembler.
-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*.This includes linker flags as well as files/library list.
-msmall	Generate code for CR16B small model (default).
-mlarge	Generate code for CR16B large model.
-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 -ladb -lc -ld`	Link the objects files with the CompactRISC standard libraries.	cr.x
`crlink file1.o file2.o -ladb -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 -ladb -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 -ladb -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 -ladb -lc -lhfp`	Link the object files with `libstart`, `libc`, and the floating-point emulation library, `libhfp`.	cr.x

Debugger Command Lines

Definition	Syntax
alias	
Define macro	<*name*> = command
Define substitution macro	<*name*> = "command $$, $$"
List macro	<*name*>
Remove macro	-r **[**<*name*> **|** *]
autocommand	
Define a list of commands to be executed following an execution command such as step, next, reset, go.	<*command*>
Remove, disable, enable an entry. An entry id is the ordinal number shown in autocommand, without an argument.	**[**-r **|** -d **|** -e**]**%<*id*> **|** *
List autocommands	
break	
Add a hardware breakpoint	**[**-t**]** <*brkaddr_list*>**[**,c=<*RLexp*>**]** **[**,o=<*occ_cnt*>**][**,q=<*qualifier*>**]** **[**,s=<*size*>**]**
Remove, disable, enable entry for a hardware breakpoint	**[**-r **|** -d **|** -e**]**%<*id*> **|** *
call	
Call any function of the application. After execution, control returns to the debugger.	<*func_name*>(arg1,arg2,..,argn)
cd	
Change/display the working directory for creating/reading files.	**[**<*path*>**]**
comm	
Sets the communication channel to communicate with an ADB, or a simulator running on the host platform.	comm **[**-s**]** **[**-s <*communication_channel_name*> **]**
core	
Sets the current CPU core	core **[**CR16A **|** CR16BL **|** CR16BS**]**
debug	
Select an executable file (COFF) to debug	**[**-x **|** -n **|** -xn**]** <*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*>**]**

Debugger Command Lines (Continued)

Definition	Syntax
info	
Display debugger information	
input	
Execute command script file	<*file_name*>
list	
List memory	-m **[**h **|** o **|** d**]** **[**b **|** c **|** w **|** f **|** p **|** l **|** i**]** <addr_range>
List source file	<qualified_lineno>
log	
Create a log file or append to an existing one (default = crdb.log)	**[**-a**]**<file_name>
Disable / enable the log file	**[**-d **|** -e**]**
Give the read only / write only attributes	**[**-i **|** -o**]**
Expand aliases	**[**-f **|** -u**]**
modify	
Modify memory	**[**-b **|** -c **|** -w **|**-f **|** -p **|** -l**]**<address> **|** <addr_range>**[**,<value>**[**,value**]]**
Modify string	-a <string_pointer>,<string>
Modify register	%<reg_name>,<value>
next	
Execute the next source line	**[**-c **|**-n**]** **[**<*from_addr*> **[***//***<*end_addr*>**]]**
nextins	
Execute the next assembly instruction	**[**-c **|**-n**]** **[**<*from_addr*> **[***//***<*end_addr*>**]]**
pause	
Suspend program execution	
quit	
Quit the debugger	
radix	
Set radix for output display	**[**8 **|** 10 **|** 16**]**
reset	
Reset the application and the target board	
resume	
Resume execution of an application, that was suspended using the pause command	
saveconfig	
Save the current debugger configuration in a file (default crdb.env)	**[**<*file_name*>**]**
savestate	
Save the current debugging state (default crdb.ctx)	**[**<*file_name*>**]**

Debugger Command Lines (Continued)

Definition	Syntax
set	
Define debugger's variable and function keys	<*name*> = <*string*>
Undefine debugger's variable and function keys	-r <*name*> **|** *
Display variables assignment	<*name*>
setstate	
Restore debugging state, as saved with savestate command (default crdb.env)	<*file_name*>
softbreak	
Add a hardware breakpoint	**[**-t**]** <*softbreak_list*> **[**,<c=<*RLexp*>**]** **[**,o=<*occ_cnt*>**]**
Remove, disable, enable a hardware breakpoint	**[**-r **|** -d **|** -e**]**%<*id*> **|** *
srcmode	
Set the source file window display mode	**[**-s **|** -m**]**
srcpath	
Set a directory for the source files	<*pathname_list*>
Remove a directory for the source files	-r **[**<*path*> **|** *]
stdio	
Redirect stdin to a file	-i <*filename*>
Redirect stdout to a file	-o**[**a**]** <*filename*>
Redirect stderr to a file	-e**[**a**]** <*filename*>
Redirect stdout, stderr to a file	-oe**[**a**]** <*filename*>
Redirect stdin, stdout, stderr to output Window	-**[**ieo**]**w
List current virtual I/O standard files	
step	
Step one source line	**[**-c **|** -n**]** **[**<*from_addr*> **[**//<*end_addr*>**]]**
stepins	
Step one assembly instruction	**[**-c **|** -n**]** **[**<*from_addr*> **[**//<*end_addr*>**]]**
symbol	
Display symbol info	**[*** **|** <*pattern*>***]**
Display watch symbols	-l **[*** **|** <*pattern*>***]**
Display symbol tag	-t **[**<*datatype*>/<*tagname*>**|**<*symbol-name*> **|** *]
Display module symbols	-f <*qualified_modulename*>
Display global symbol	-g <*pattern*>*

sync

Display the PC corresponding line

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		Data Type	
$$	Macro argument	-a	ASCII string
*	All items	-b	Byte
-r	Remove an entry point from a list	-c	

Glossary for Debugger Commands (Continued)

Break	
`-t`	Temporary breakpoint. This breakpoint is deleted after it occurs
`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`	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
`s=size`	Number of bytes for which the data break is applicable (default is the data type of the symbol). or the target integer 2 - 16-bits core 4 - 32-bits core

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>` .

Debug	
`-x`	Selects a file, and downloads it without its symbols
`-n`	Selects a file and downloads only its symbol table

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

modify	
`-a`	String copy

next / nextins / step / stepins	
`-c`	Continues execution of source line until the end of program
`-n`	Executes x lines (x = command argument: a given number)

set	
`-k`	Key definition

srcmode	
`-s`	Enables source-only display (for C program, C lines only)
`-m`	Enable mixed mode (for C program; C and assembly lines)

where	
`-c`	Current source line with arguments
`-v`	Stack history

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