Commodore Plus/4 and C16: Hardware and Advanced Basic Programming.

Version 1.3

by Janne Peräaho & Anders Persson

AUTHORS

Janne Peräaho E-mail: amity@surfeu.fi

Anders Persson http://listen.to/boray

ALL DOCUMENTS

This document is part of a document package for Plus/4, C16 and C116 users. The documents included are:

- "basic35.pdf", Janne's original Basic 3.5 manual (more or less)
- "short35.pdf", a Basic Quick Guide.
- "advanced.pdf", This document - The Hardware and Advanced Basic programming.
- "tedmon.pdf", a short description of the built in machine language monitor.

REFERENCES

Commodore Vic-20 and C64 programmers reference manual, Talking to TED by Levente Harsfalvi, Commodore plus/4 and c16 memory map.

INTRODUCTION

The TED chip is the graphics and sound chip of these computers. For a good description and a memory map of it, read this article:

http://www.canberra.edu.au/~scott/C=Hacking/C-Hacking12/gfx.html

Complete Memory maps of the Plus4/C16:

http://www.funet.fi/pub/cbm/maps/C16.MemoryMap
http://plus4.emucamp.com/tools/rommap/index.php

SLOW/FAST

These computers run at two processor speeds (~1.7MHz) and a slow (~0.9MHz) at the same time. Well, not at the same time, but it constantly switches between the fast and slow processor speeds. It's slow whenever the raster beam is on the visible screen, and fast otherwise. The resulting speed is about 1.1MHz for NTSC and 1.2MHz for PAL. If you switch off the display, it will run on the fast speed all the time. You can use this from Basic, just as the FAST and SLOW commands on the C128:

POKE 65286,0 (the screen goes blank) and back: POKE 65286,27

RESTORE KEY?

If you have used a vic20/c64/c128, then you might wonder where the "restore" key is? But there is none! To perform something like it, hold down run/stop and press the reset button. This enters the monitor. Type X and hit return to return to basic. The basic program remains in memory.

I don't know about you, but I find that procedure a little awkward. So why not make your own restore key instead? When programming Basic, the most common reason to use the vic20/c64/c128s restore key is to restore the colors, sound and graphics mode to the default. But this can be done with SYS commands! So let's define a function key:

```
KEY 1,"{clr home}SYS65418:SYS65409:SYS65412"+CHR$(13)
```

Then to perform the equivalence to the c64's runstop-restore, first press run/stop to break the program (if it's running), release and then press F1. How nice! If you like, you can put LIST in there too, because that is probably the command you most often like to do after the restore:

```
KEY 1,"{clr home}SYS65418:SYS65409:SYS65412:LIST"+CHR$(13)
```

If you like, you can also put this line in the program you are working on so that you don't need to type it in every time you turn on the computer.

The sys commands used are RESTOR, CINT and IOINIT from the Kernal.

Another little tip (if you don't have made your own restore key) and you just want to get out of a basic graphics mode without having to type GRAPHIC 0. Then just press any key followed by return. The basic will then answer with ?SYNTAX ERROR and go to GRAPHIC 0 by itself to display the error message.

WINDOW COMMAND

The C128 has a WINDOW basic command to set the output window to just a part of the screen. The C16/Plus4 has the same feature but no specific command for it. You can do it with Esc codes (Look at page 6 for the whole list) or with pokes. The Esc codes to use are Esc-T to set top of window, Esc-B to set bottom and Esc-N for the full screen (and clearing it). Also, pressing the Home key twice returns to full screen without clearing it. The pokes to use are 2021 (Screen bottom), 2022 (Screen top), 2023 (Screen left) and 2024 (Screen right). Also, the CHAR command restores the window to the whole screen without clearing it. If you only use the pokes, then you need to follow up with a PRINT"{home}"; or the output could start outside of the new window. So for something like the C128's window command, you could use this line (which probably is the shortest way of doing it):

```
CHAR, X1,Y1,CHR$(27)+"T":POKE2024,X2:POKE2021,Y2
```

Where (X1,Y1) is the top position and (X2,Y2) is the bottom position.

SCREEN CODES (TED text modes)

The Screen memory starts at **3072** by default.

The color/attribute memory starts at **2048** by default.

Normal Text (the default) (8x8 pixel characters)

- Screen memory: Bits 0-6 is one of 128 characters and bit 7 is reverse on/off.
- Color memory: Bits 0-3: Color, Bits 4-6: luminance, Bit 7: Flash on/off

Examples: Poke 3072,1 (puts an A at top left screen position)
Poke 3073,2+128 (puts a reverted B beside it)
Poke 2048,128+2+5*16 (makes the A light red and blinking)

256 Char display (8x8 pixel characters)

Turn on with Poke 65287,peek(65287) OR 128

- Screen memory: Bits 0-7 is one of 256 characters.
- Color memory: Bits 0-3: Color, Bits 4-6: luminance, Bit 7: Flash on/off

Multi Color (4x8 pixel characters)

With multicolor (lo-res) turned on, you have to design your own characters where every byte is built of four bit pairs = 4 pixels. Every pixel is in memory two bits, so there are four different combinations, 00, 01, 10, and 11 which each represent a different color.

Turn on with poke 65287,peek(65287)or16

- Screen memory: Bits 0-7 is one of 256 characters.
- Color memory:

Bit 3: Multicolor on/off. (Both multicolor (4x8) characters and hires (8x8) characters can be displayed on the same screen.)
Bits 0-2 Color and
bits 4-6 luminance for the "11"-pixels in the charter set.

For the rest of the combinations:

"00"=background (color 0) (address 65301 bits 0-3 color and 4-6 luminance), "01"=color 3 (address 65302) and "10"=(address 65303).

Extended Color (8x8 pixel characters)

(This means you can have different background colors to different hires characters on the same screen)

Turn on with Poke 65286, peek(65286)or64

- Screen memory:

Bits 0-5 is one of 64 characters.

Bits 6-7 is background color for the character:

"00"= normal background (color 0) (65301 bits 0-3 color and 4-6 luminance), "01"= color 3 (65302), "10"= (65303) and "11"= (65304).

- Color memory:

Bit 3: Extended color on/off.
Bits 0-2 Foreground Color and
bits 4-6 luminance.

(There are also two bitmap modes: Hires (320x200) and Multicolor (160x200). These are easiest used from basic with the GRAPHIC command. The only text modes that works with "split screen" seems to be Normal Text and 256 char display.)

DEFAULT CHARACTER SET:

SET 1	SET 2	POKE	SET 1	SET 2	POKE	SET 1	SET 2	POKE
@		0	U	u	21	*		42
A	a	1	V	v	22	+		43
B	b	2	W	w	23	,		44
C	c	3	X	x	24	—		45
D	d	4	Y	y	25	.		46
E	e	5	Z	z	26	/		47
F	f	6	[(Ä)	ä	27	ø		48
G	g	7	£(Ö)	ö	28	1		49
H	h	8	](Å)	å	29	2		50
I	i	9	↑		30	3		51
J	j	10	←		31	4		52
K	k	11	SPACE		32	5		53
L	l	12	!		33	6		54
M	m	13	"		34	7		55
N	n	14	#		35	8		56
O	o	15	$		36	9		57
P	p	16	%		37	:		58
Q	q	17	&		38	;		59
R	r	18	'		39	<		60
S	s	19	(		40	=		61
T	t	20	)		41	>		62

SET 1	SET 2	POKE
?		63
		64
♠	A	65
	B	66
	C	67
	D	68
	E	69
	F	70
	G	71
	H	72
	I	73
	J	74
	K	75
	L	76
	M	77
	N	78
	O	79
	P	80
●	Q	81
	R	82
♥	S	83

SET 1	SET 2	POKE
	T	84
	U	85
	V	86
	W	87
♣	X	88
	Y	89
◆	Z	90
	Ä	91
	Ö	92
	Å	93
		94
		95
SPACE		96
		97
		98
		99
		100
		101
		102
		103
		104
		105

SET 1	SET 2	POKE
		106
		107
		108
		109
		110
		111
		112
		113
		114
		115
		116
		117
		118
		119
		120
		121
	✓	122
		123
		124
		125
		126
		127

YOUR OWN CHARACTER SET

First you have to tell that characters should be fetch from RAM. This is done by clearing the second bit of the TED register number 65298 ($FF12):

```
poke 65298, peek (65298) and 251
```

Then you must select a "memory page" where your character set is stored. Each page is 1024 bytes long, so on the C16 there are 16 possible locations:

```
Page 0: character set starts at memory address 0
Page 1: character set starts at memory address 1024
Page 2: character set starts at memory address 2048
...
Page 14: character set starts at memory address 14336
Page 15: character set starts at memory address 15360
```

(On the Plus4 there are more pages, probably 64)

Bits 2-7 of the TED register number 65299 ($FF13) determine, which page should be used. To select a page do as follows:

```
poke 65299, (peek (65299) and 3) or (page * 4)
```

Another way of doing the same thing is to take the first byte in the address (as long as one of the pages are used). For example, page 15's address is 15360. The hex number for this is $3c00. So you can just poke 65299, dec(“3c”).

When using Basic one should protect the character set so, that Basic's program text does not overwrite your character data. This can be done by setting the upper limit for the Basic text, i.e. the highest address Basic is allowed to use. To do that, use memory addresses 55 ($37) and 56 ($38). 55 is the lower half of the 16 bit address and 56 is the upper half. Read more on page 9: "Limiting the basic memory".

To switch back to ROM character set, just set the 2nd bit of the 65298 and set the page back to page 52:

```
poke 65298, peek (65298) or 4 : poke 65299,208
```

ESCAPE CODES

Cancel quote and insert mode	ESC O
Cancel started Esc code	ESC X
Erase to end of current line	ESC Q
Erase to start of current line	ESC P
Move to start of current line	ESC J
Move to end of current line	ESC K
Enable auto-insert mode	ESC A
Disable auto insert mode	ESC C
Delete current line	ESC D
Insert line	ESC I
Enable scrolling	ESC L
Disable scrolling	ESC M
Scroll up	ESC V
Scroll down	ESC W
Set bottom of screen window	ESC B
Set top of screen window	ESC T
Set window to full screen minus 1 and clear	ESC R
Set window to full screen and clear screen	ESC N

Example: Press Esc followed by A for auto insert mode, or PRINT CHR$(27);"W" to scroll the screen down from basic.

MORE SPECIAL KEYS

Pause output	CTRL S (any key to resume)
Slow down output	Commodore
Set window to full screen	Home twice

ROM

The C-16's ROM consists of three main parts: BASIC, TEDMON and KERNAL. BASIC is a Basic interpreter, which can run Basic programs stored in RAM. TEDMON is a machine language monitor for manipulating memory and writing short machine language programs. KERNAL is not a single runnable program, but a collection of system routines arranged as a Jump Table. The Plus/4 also includes a built in software package called "3-plus-1" that includes word processing, spread sheet, business graphics and a database.

HOW TO ACCESS ROM PROGRAMS?

BASIC is the primary user interface and it starts automatically when power is turned on. The secondary user interface is TEDMON, which can be entered by holding down the Reset button when turning the computer on.

You can also activate TEDMON from BASIC and vice versa. See Basic's MONITOR command and TEDMON's X command.

The KERNAL is used by the Basic interpreter and other programs. Using KERNAL routines requires a little machine language knowledge since it is required to use processor registers for exchanging information between a KERNAL routine and the routine caller. It is however possible to call KERNAL routines directly from Basic with the use of the SYS command and the following memory positions: 2034=A, 2035=X, 2036=Y. For example, POKE 2034,23 and the accumulator will contain the value 23 when the SYS command is started.

To call a KERNAL routine do as follows:

1. Set parameters.
2. Call KERNAL routine using JSR command (or SYS from Basic).
3. Handle any return error.

Note, that some routines need to be initialized with other routines before they can be used. For example, using the SAVE routine requires initialization with the SETLFS and SETNAM routines.

See the "KERNAL Jump Table" below for more information on available KERNAL routines.

Basic example:

```
10 SYS DEC("FFE4") : REM "GETIN"
20 A=PEEK(2034) : IF A=0 THEN 10
30 A$ = CHR$(A)
```

Equals to:

```
10 GETKEY A$
```

KERNAL JUMP TABLE

Seems to be exactly the same as for the C64. Please consult the C64 programmer's reference manual for parameters. Available on http://www.funet.fi/pub/cbm/c64/manuals/

Routine	Address	Function
CINT	$FF81	Initializes screen editor.
IOINIT	$FF84	Initializes I/O devices.
RAMTAS	$FF87	Tests RAM.
RESTOR	$FF8A	Restores vectors to initial state.
VECTOR	$FF8D	Changes vectors for user.
SETMSG	$FF90	Controls O.S. messages.
SECOND	$FF93	Sends SA after LISTEN.
TKSA	$FF96	Sends SA after TALK.
MEMTOP	$FF99	Sets/reads top of memory.
MEMBOT	$FF9C	Sets/reads bottom of memory.
SCNKEY	$FF9F	Scans keyboard.
SETTMO	$FFA2	Sets timeout in DMA disk.
ACPTR	$FFA5	Handshakes serial bus or DMA disk byte in.
CIoUT	$FFA8	Handshakes serial bus or DMA disk byte out.
UNTLK	$FFAB	Sends UNTALK out serial bus or DMA disk.
UNLSN	$FFAE	Sends UNLISTEN out serial bus or DMA disk.

LISTN	$FFB1	Sends LISTEN out serial bus or DMA disk.
TALK	$FFB4	Sends TALK out serial bus or DMA disk.
READSS	$FFB7	Returns I/O STATUS byte.
SETLFS	$EF8A	Sets LA, FA, SA.
SETNAM	$FFBD	Sets file name.
OPEN	$FFC0	Opens logical file.
CLOSE	$FFC3	Closes logical file.
CHKIN	$FFC6	Opens channel in.
CHOUT	$FEC9	Opens channel out.
CLRCH	$FECC	Closes I/O channels.
SETLFS	$FFBA	Sets logical file number and command of device.
CHRIN	$FFCF	Inputs one character from keyboard.
CHROUT	$FFD2	Outputs one character to screen.
LOAD	$FFD5	Loads/verifies memory from device.
SAVE	$FFD8	Saves memory area to device.
SETTIM	$FFDB	Sets internal clock.
RDTIM	$FFDE	Reads internal clock.
STOP	$FFE1	Scans STOP key.
GETIN	$FFE4	Gets one character from keyboard queue.
CLALL	$FFE7	Closes all files.
UDTIM	$FFEA	Increments clock.
SCRORG	$FFED	Screen org.
PLOT	$FFF0	Gets/sets cursor position.
IOBASE	$FFF3	Returns location of start of I/O.

3-PLUS-1

The extra software package built into the Plus/4 is started by pressing the F1 key followed by return. The word processor is started by default. For the other parts, hold down Commodore and press C, then type TC (followed by return) for Spreadsheet, TF for file manager (database) or TW to jump back to the word processor.

THE WORD PROCESSOR

Here are the word processor's commands: Press Commodore+C followed by

CA - Directory	CB - Create block	CM - Clear memory
CP - Clear pointers	CT - Clear tabs	DB - Delete block
DF - Delete file	DL - Delete line of text	EP - Erase pointer
IB - Insert block	ID - Format disk	IL - insert text line
LF - Load file	MF - Merge file	PR - Print document
RE - Search and replace	SF - Save file	SP - Set pointer
SR - Search	*P - Prind doc in memory	

Keys:

CLR - move to bottom	Ctrl-= - Set a Tab	Ctrl-9 - reverse (formatting)
Ctrl-0 reverse off	Del - Backspace	C=-C - Command mode
C=-R - move to col 41	C=-L move to left margin	C=-Q - Repeat prev keystroke
C=-@ - Undo return	Home - Home	Insert - Insert
Return - Terminate line	Sh-Return - to left marg.	Shift-= - Tab key

Document formatting: The following keywords can be embedded in the text. They are written in reverse video and typed in lowercase. ":" is used to separate multiple instructions on the same line and ";" is used to terminate the instruction. For example papersize50;

asc	Send ascii character to the printer.
center	Center this line.
justify	Justify text
linkfile	Link documents at print time (with PR). Example: linkfile "opa";
lmarg	Set left marginal. Example: lmarg9; (default is 0)
nextpage	Page break
nojustify	Turn off justify (default)
nowrap	Turns of word wrap (for spreadsheets)
no#page	Deactivates #page command
other	Used with non-Commodore printers (normal Ascii-printers).
pagelen	Sets page length. (Default is 60).
pagepause	Pause printing after every page.
papersize	Changes size of the paper from default 66. Use together with pagelen.
pause	Pause printing until you press return.
rmarg	Set right margin (default is 77).
set#pg	Set page number. Used together with #page.
#page	Print page number at the bottom of the page.
wrapon	Turn word wrap on (default)

READING THE KEYBOARD

The easiest way to read the keyboard from Basic is to use the GET or GETKEY commands. Here is a third way: PEEK(198). The computer checks the keyboard every 60th of a second and stores a value corresponding to the key pressed at memory location 198. The stored value is a bit pattern of the keyboard matrix, not a PET-ASCII or screen value of the key. Below is a list of all the values. For example, if you like to test the F1 key: IF PEEK(198)=4 THEN...

Key	Value	Key	Value
Esc	52	Run/Stop	No Effect
1	56	Shift Lock	No Effect
2	59	a	10
3	8	s	13
4	11	d	18
5	16	f	21
6	19	g	26
7	24	h	29
8	27	j	34
9	32	k	37
Left Arrow	48	l	42
Right Arrow	51	:	45
Up Arrow	43	;	50
Down Arrow	40	*	49
Inst/Del	0	Return	1
Ctrl	No Effect	Commodore	No Effect
q	62	Left Shift	No Effect
w	9	z	12
e	14	x	23
r	17	c	20
t	22	v	31
y	25	b	28
u	30	n	34
i	33	m	36
o	38	,	47
p	41	.	44
@	7	/	55
+	54	Right Shift	No Effect
-	46	£	2
Clear/Home	57	=	53
F1	4	Space	60
F2	5	No key	64
F3	6		
Help	3		

LIMITING THE BASIC MEMORY

LET'S START WITH THE END

Basic programs use the free memory while running. If you have stored your own machine language programs into the memory or your own character set, you have to inform Basic not to use that area of memory, or otherwise Basic will overwrite your machine language programs or your character set with temporary data.

To protect your machine language programs and/or your custom character set, the easiest way is to place them at the end of the memory and to inform Basic that the memory available for Basic ends right before your machine language programs and/or custom character set begins.

The highest address of Basic memory can be read and set through memory locations 55 and 56. Location 55 contains the lower 8 bits of the address and 56 the upper 8 bits.

To read the current "Top of Basic memory" (=end position), type the following line in Basic and press Return:

```
PRINT PEEK(55) + PEEK(56) * 256
```

To protect your machine language programs and/or custom character set, follow

these steps:

1. Calculate the total size needed for your machine language programs and/or custom character set. (SIZE)
2. Get the "top of Basic memory", as instructed earlier. (TOBM)
3. Calculate new "top of Basic memory" (NTOBM): TOBM - (SIZE + 1).
4. Calculate low and high parts of the NTOBM:
   HI = INT(NTOBM / 256)
   LO = NTOBM - (HI * 256)
5. Set new "top of Basic memory":
   POKE 55, LO
   POKE 56, HI
   CLR

Note! The CLR statement at the end of step 5 is essential. It sets a lot of pointers used by Basic to the "top of Basic memory" and Basic will crash unless it is executed after the POKEs.

Note! If you have a Plus/4 and want your program to be C16 compatible, then instead of calculating from the real top (end) of memory (64768), calculate from 16374 instead.

ON THE OTHER SIDE

You can also move the start of the basic memory instead, but that is a bit more difficult as the whole basic program then needs to be moved. You can alter the start position of the basic memory this way:

```
POKE 44,x : POKE x*256,0 : NEW
```

...and the basic memory will then start at x*256. If x for example is 32, then the basic memory will start at 32*256=8192. But as a NEW command is used, it is tricky to do this at run time. It needs multi part loading, poking into the keyboard buffer etc, or another trick is to have two basic programs in memory where one is starting the other one... Complicated stuff... but on the expanded vic-20 this is quite common as the vic chip only can address the first internal 5K and not the expanded higher memory. But anyway, for us, it's much easier to move the end limit of the memory instead, as described above... except for if you use the following little trick on the Plus/4:

```
GRAPHIC1:GRAPHIC0
```

...Then (as described in the Basic 3.5 document under GRAPHIC), the whole basic program is moved at runtime from 4097 to 16385 taking 12kb from the basic memory. If you don't use the graphics, you can use this 12kb area for other things. And even if you use the graphics, the 2kb area 4097-6143 is still unused and available for other things. But this approach will give programs that NOT are compatible with the C16! Well, as long as you don't use the extra 4097-6143 area and only the 10kb graphics area, then it's C16 compatible, but then you will only have 2kb left for the basic...

FORMAT OF BASIC PROGRAMS IN MEMORY

Basic stores programs in memory in a special compressed format to conserve memory. Conversion occures when you enter a new line in your Basic program: each reserved word (PRINT, GOTO, INPUT...) found in the line will be converted to a single byte called a token and stored into the memory. When you list your program, LIST command translates the tokens and displays them as the original reserved words.

A Basic line has the following format in the memory:

```
{LON HIN} {LOL HIL} CONTENTS EOL|EOP
```

{LON HIN} is a 16-bit pointer to the start of the next Basic line. LON contains the lower 8 bits of the address and HIN the upper 8 bits. In case of last Basic line, {LON HIN} points to EOP.

{LOL HIL} is a 16-bit linenumber. LOL contains the lower 8 bits of the linenumber and HIL the upper 8 bits.

CONTENTS is the contents of the Basic line. Line composes a mixture of tokens and PET-ASCII characters: bytes with a value less than 128 are conventional characters and bytes greater than or equal to 128 are tokens.

EOL is an end of line mark. Its byte value is zero. EOL is used at the end of every Basic line, except the last line.

EOP is an end of program mark. It consists of two bytes both having a value of zero. EOP is used at the end of the last Basic line to indicate that program listing ends.

BASIC TOKENS

as described above...

Token (Hex)	Token (Dec)	Keyword
80	128	END
81	129	FOR
82	130	NEXT
83	131	DATA
84	132	INPUT#
85	133	INPUT
86	134	DIM
87	135	READ
88	136	LET
89	137	GOTO
8A	138	RUN
8B	139	IF
8C	140	RESTORE
8D	141	GOSUB
8E	142	RETURN
8F	143	REM
90	144	STOP
91	145	ON
92	146	WAIT
93	147	LOAD
94	148	SAVE
95	149	VERIFY
96	150	DEF
97	151	POKE
98	152	PRINT#
99	153	PRINT
9A	154	CONT
9B	155	LIST
9C	156	CLR
9D	157	CMD
9E	158	SYS
9F	159	OPEN
A0	160	CLOSE
A1	161	GET
A2	162	NEW
A3	163	TAB(
A4	164	TO
A5	165	FN
A6	166	SPC(
A7	167	THEN
A8	168	NOT
A9	169	STEP
AA	170	+
AB	171	-
AC	172	*
AD	173	/
AE	174	^
AF	175	AND
B0	176	OR
B1	177	>
B2	178	=
B3	179	<
B4	180	SGN
B5	181	INT
B6	182	ABS
B7	183	USR
B8	184	FRE
B9	185	POS
BA	186	SQR

BB	187	RND
BC	188	LOG
BD	189	EXP
BE	190	COS
BF	191	SIN
C0	192	TAN
C1	193	ATN
C2	194	PEEK
C3	195	LEN
C4	196	STR$
C5	197	VAL
C6	198	ASC
C7	199	CHR$
C8	200	LEFT$
C9	201	RIGHT$
CA	202	MID$
CB	203	GO
CC	204	RGR
CD	205	RCLR
CE	206	RLUM
CF	207	JOY
D0	208	RDOT
D1	209	DEC
D2	210	HEX$
D3	211	ERR$
D4	212	INSTR
D5	213	ELSE
D6	214	RESUME
D7	215	TRAP
D8	216	TRON
D9	217	TROFF
DA	218	SOUND
DB	219	VOL
DC	220	AUTO
DD	221	PUDEF
DE	222	GRAPHIC
DF	223	PAINT
E0	224	CHAR
E1	225	BOX
E2	226	CIRCLE
E3	227	GSHAPE
E4	228	SSHAPE
E5	229	DRAW
E6	230	LOCATE
E7	231	COLOR
E8	232	SCNCLR
E9	233	SCALE
EA	234	HELP
EB	235	DO
EC	236	LOOP
ED	237	EXIT
EE	238	DIRECTORY
EF	239	DSAVE
F0	240	DLOAD
F1	241	HEADER
F2	242	SCRATCH
F3	243	COLLECT
F4	244	COPY
F5	245	RENAME
F6	246	BACKUP
F7	247	DELETE
F8	248	RENUMBER
F9	249	KEY
FA	250	MONITOR
FB	251	USING
FC	252	UNTIL
FD	253	WHILE
FE	254	*NOT USED*
FF	255	PI

BASIC OPTIMIZATION

The following is true for all versions of the 8 bit CBM Basic and is not particular for Basic 3.5. (At least I think so).

GOTO and GOSUB

Because of the format of the basic (see page 10), where every basic line only have one pointer to the next line and none to the previous, then jumps (GOTO and GOSUB) will take different time to execute depending on the situation. Jumps to lines just below the current one will be very fast, while jumps to lines above the current line will be very fast for the first lines of the program but slower and slower the lower down in the program the line is. So if you have a big program and on the last line you jump one line up to the second last line, then this will be quite slow as the basic will have to go through every line pointer in the program. The RETURN command is however always of the same speed.

The tip here is to always put your heavy main routine FIRST in your program, only preceded by one line that just jumps down to the parts setting up your program etc. Then to go back to the main routine at the second first line. For example:

```
0 GOSUB 500 : REM SETUP EVERYTHING
1 ...Main routine
```

Heavily used subroutines should also be put as high as possible or just below the calling routine.

When your program is ready, you can also renumber it with RENUMBER 0,1
This way the line numbers will be as small as possible, saving space and also speeding it up a little.

VARIABLES

Variables are faster than constants, as long as you don't have too many variables. For example:

```
A=132:FORT=1TO100:B=PEEK(A):NEXT
```

is faster than:

```
FORT=1TO100:B=PEEK(132):NEXT
```

Another important issue is that different variables has different speed! The thing to remember here is that the variables used first in the program is the ones that will be the fastest. So a good way to speed your program up is to analyze your heavy routine to see which variables that are used the most. Then just use them (set some value to them) the first thing you do in your program. For example, on line number 0:

```
0 X=1:L=1:T=1:R=1
```

After that, X will be fastest, followed by L, and then T, and then R...

One could be fooled to think that integer variables (A% etc) would be faster than floating point variables (A etc), but it's the other way around! Normal floating point variables are the fastest.

TESTING SPEED

To test your routine for speed improvements, you can use the TI variable like this:

```
10 E=TI
.
.
. your routine
.
.
120 PRINT TI-E
```

The faster it gets, the lower is the value displayed.

COMPILERS

If you compile your program into machine language (for example using the Austro compiler), then none of these rules are true. In fact, constants are then faster than variables and integers are then faster than floating point. So it would be wise do decide if you are going to compile or not before you start to design your program.