

# BASIC COMPILER

## manual

### PURPOSE

Created by András Bognár (2011)  
For Gábor Bárány, using in his microcomputer system

### LANGUAGE

Command format:

[Command linenum] [Command] [Parameters]

10 PRINT "HELLO"

(With #UNNUMBERED directive you can leave [Command linenum], see directives section)

One command line can contain multiple commands, commands must be separated by ':' in one line:

10 A=1:B=2\*FR:GOTO 120

### VARIABLES

Every variable is stored as 16 bit integer

A non array variable needs no declaration, just simply: A=1

1 or 2 dimensional arrays must be declared : DIM ARRAYVAR[12] or DIM TMB[10,12]

Array can be 1 or 2 dimensional array, higher dimensions are not supported

valid:

10 DIM A[5]

20 DIM FIG[8,10]

30 G=7

40 A[2]=23

50 FIG[2,5]=21

### EXPRESSIONS

Numerical and boolean expressions are implemented

In numerical expression only '+', '-', '\*', '/' are allowed.

In boolean expression all of the operators are allowed that are in this list:

Operators:

'+'	-addition
'-'	-subtraction
'*'	-multiplication
'/'	-division

'AND'	-logical AND
'OR'	-logical OR
'NOT'	-logical NOT
'<'	-less than
'>'	-greater than
'='	-equal
'>='	-greater or equal
'<='	-less or equal
'<>'	-not equal

valid numerical expression examples:

```

10 ATOM[1,0]=1+ATOM[1+ATOMIC[8+D,0]*3,4*A]+8+3*ATOM[1,0]
20 B=A[1,2]*5+6*(RND/12+(AF+9)*(UM-AM))+AXLE-R*S/T
30 A[F,G]=A[F,G]*H[J,K]
40 A[F,G]=85*A[F,G*2]
50 HIK=A[F,G*4+3]/78+FE*(5+U)
60 E=8*A[S[23,J],78+E[2,H[4,5]]*2]+7

```

## COMMAND LIST

### LET

Assigns a value to a variable

```
10 LET A=35
```

Keyword LET can be left, be substituted with "" empty string:

```
10 S=31
```

### END

Finishes execution of the program

### STOP

Finishes execution of the program

### FOR NEXT

Create a for loop

This cycle runs 12 times:

```
10 FOR A=1 TO 12
```

```
20 NEXT A
```

It is also possible to use STEP expression:

```
10 FOR A=1 TO 12 STEP 2
```

```
20 NEXT A
```

Expressions can be complicated:

```
10 FOR A=F*HI+5*(A[1,2]-2) TO D*A/7 STEP R+S
```

20 NEXT A

## GOSUB RETURN

Jumps to a subroutine at a given line in the program, placing the return address on the stack

RETURN pops a subroutine return address from the stack and jumps to it

10 GOSUB 30

30 RETURN

## POP

Removes a subroutine return address from the stack

## IF THEN

Branches depending on whether a condition is true

IF <BooleanExpression> THEN <Command>

Boolean expression can contain all type of operators such as:

'+'	-addition
'-'	-subtraction
'*'	-multiplication
'/'	-division
'AND'	-logical AND
'OR'	-logical OR
'NOT'	-logical NOT
'<'	-less than
'>'	-greater than
'='	-equal
'>='	-greater or equal
'<='	-less or equal
'<>'	-not equal

It is valid format:

10 IF ((A=1) AND (F=3\*KL)) OR (s\*4>=d\*(6+E)) THEN GOTO 120

Brackets must be used for "AND", "OR", "NOT" operators such as:

(...) OR (...)

(...) AND (...)

NOT (...)

When "NOT" is used with "OR", "AND" :

(...) OR NOT (...)

(...) AND NOT (...)

"GOTO" can be left after "THEN"

These lines are equals:

10 IF A=1 THEN 20

10 IF A=1 THEN GOTO 20

After "THEN" multiple commands separated by ':' are executed when boolean expression is TRUE, else none is executed.

```
10 IF B=2 THEN S=3:F=3*S:GOTO 100
```

## GOTO

Jumps to a given line in the program

```
10 GOTO 100
```

only a number can be used after GOTO

## ON GOTO

A computed goto - performs a jump based on the value of an expression

```
10 ON A GOTO 100,120,130
```

If value of A=1 then it jumps to the line 100, if A=2 then it jumps to the line 120, and so on...

Maximum 6 line addresses can be given:

```
10 ON B GOTO 40,50,60,70,120,200
```

## PEEK

Returns the value at an address in memory

```
10 PEEK A,12893
```

Stores the 12893th memory address value to A variable

## POKE

Sets a value at an address in memory

```
10 POKE A,12893
```

Stores the value of A variable to the 12893th memory address

## REM

Marks a comment in a program

```
10 REM You can type here anything
```

## READ

Reads data from a DATA statement

```
10 READ DATA1,D
```

```
20 DATA D 2,10,20,456,32,78,231,459,443,121
```

```
30 DATA HEXDATA 0x1234,0x56AF,0x7A5B,0xFFFF
```

Reads the next data from dataline, starting from the beginning of the data list

If data pointer overflows than it flows to the next data line:

Reading 4 times from D1 DATA line , than the first data is read from D2 DATA line

```
10 READ D1,DT
```

```
20 DATA D1 1,2,3
```

```
30 PLOT 10,10
```

```
40 DATA D2 3,4,5
```

## RESTORE

Sets the position of where to read data from a DATA statement

```
10 READ DATA1,D
```

```
20 RESTORE DATA1
```

```
30 DATA DATA1 2,10,20,456,32,78,231,459,443,121
```

## RNG

Returns a pseudorandom number

10 RND A

Variable 'A' will be a random integer number from 0 to 255

## SOUND

Starts or stops playing a tone on a sound channel

10 SOUND 100

Play a 100Hz sound on sound channel

20 SOUND 0

Stop playing sound

## DELAY

Halt program executing for the specified time

10 DELAY 120

## Graphic commands

### COLOR

Sets a color (brightness) that is used by "PLOT" command

10 COLOR 35

### PLOT

Puts a pixel to the given position

10 PLOT 10,23

Pixel brightness is set to maximum as default.

Pixel brightness can be given by "COLOR" command

### PUTPIXEL

Puts a pixel to the given position with the given brightness

10 PUTPIXEL 11,32,A

Put to the (11,32) coordinates with color given by "A" variable

### GETPIXEL

Puts a pixel value from the given position

GETPIXEL Q,10,3

Variable "Q" will be the pixel brightness at position (10,3)

### CLS

Clear screen

### POSITION

Puts the graphic cursor to the given position

10 POSITION 23,15

## DRAWTO

Draws a line from the actual graphical cursor position to the given coordinates  
10 DRAWTO 75,46

## GETMAXX MX

Gets the maximum X coordinate of the screen

## GETMAXY MY

Gets the maximum Y coordinate of the screen

## GETMAXCOLORDEPTH CM

Gets the maximum color depth (brightness depth) of the screen

## VERSHIFTGRLINE

Shift the given vertical graphical line

Shifts 34th line UP:

10 VERSHIFTGRLINE 34,'U'

Shifts 34th line DOWN:

10 VERSHIFTGRLINE 34,'D'

## HORSHIFTGRLINE

Shift the given horisontal graphical line

Shifts 34th line RIGHT:

10 HORSHIFTGRLINE 34,'R'

Shifts 34th line LEFT:

10 HORSHIFTGRLINE 34,'L'

## PUTPATTERN TABLE1,10,10,20,20

Puts a square pattern to the screen from an array variable, or from DATA lines.

(10,10) is the upper left corner

(20,20) is the botton right corner of the square

TABLE1 must have a minimum size to store the sqare area color values to it

10 DIM TABLE[10,10]

20 PUTPATTERN TABLE1,10,10,20,20

or

10 DATA DATALN 1,34,671,23,21,72

20 PUTPATTERN #DATALN,10,10,20,20

You must use '#' character before name of dataline.

## GETPATTERN TABLE2,10,10,20,20

Gets a square pattern from the screen to an array variable.

(10,10) is the upper left corner

(20,20) is the botton right corner of the square

TABLE2 must have a minimum size to store the sqare area color values to it

10 DIM TABLE[10,10]

20 GETPATTERN TABLE2,10,10,20,20

PRINT „Hello Word”

Prints out the string or number to the screen at the cursor's position. Strings ended with „\n” will be printed out with new line feeding at the end of the printing.

```
10 PRINT „Please give me a number”
20 PRINT „Print out and new line \n”
30 PRINT 2*2
40 A=7:B=5
50 PRINT A*B+3
```

GOTOXY 10,4

Moves the text cursor to the given position. This cursor is used for only PRINT command

```
10 GOTOXY 7,6
20 PRINT „Hello”
```

## Keyboard commands

KEYBINPUT K

Gets the currently pressed key on keyboard to the given variable.  
If no key is pressed than it returns with zero.

WAITKEY K

Waits for a keyboard hit, and returned with the key code in the given variable.

## SID sound generation commands

These commands are to make easy using the SID C64 sound interface device.

SID S1

Initializes SID registers. Writes from the first data bytes of the data line to the SID registers from register 0 to register 24.

```
10 SID S1
20 REM   |VOICE1           |VOICE2           |VOICE3           |Filter
DATA S1 0xFF0F,0x0000,0x1169,0x6600,0x0000,0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0100
```

So 0xFF is written to SID register 0, 0x0F is written to the second register, and so on.  
This command does not increments data pointer

SIDS S1

Same as SID S1 but it increments the data pointer with 24

#### SVOICE1 S1

Voice 1 registers are written in from the data line. Similarly to SID S1 command, initializes SID voice registers from 0 to register 6. This command does not increments data pointer.

#### SVOICE1S S1

Same as SVOICE1 S1 but it increments the data pointer.

#### SVOICE2 S1

Similar to SVOICE1 S1

#### SVOICE2S S1

Similar to SVOICES1 S1

#### SVOICE3 S1

Similar to SVOICE1 S1

#### SVOICE3S S1

Similar to SVOICES1 S1

#### SFILTER S1

Similar to SVOICE1 S1 but it updates SID filter registers.

#### SFILTERS S1

Similar to SVOICES1 S1 but it updates SID filter registers.

#### SFREQ1 0xFFFF

Sets the frequency of the voice 1 from variable.

#### SFREQ1S S1

Sets the frequency of the voice 1 from data line.

#### SPW1 0xFFFF

Sets the PWM of the voice 1 from variable.

#### SPW1S S1

Sets the PWM of the voice 1 from data line.

#### SCONTR1 0x00FF

Sets the control register of the voice 1 from variable.

#### SADSR1 0xFFFF

Sets the ADSR register of the voice 1 from variable.

#### SGATE1

Hits the Voice 1

#### SRELEASE1

Releases the Voice 1



These commands are similar:

SFREQ2 0xFFFF  
SFREQ2S S1  
SPW2 0xFFFF  
SPW2S S1  
SCONTR2 0x00FF  
SADSR2 0x00FF  
SGATE2  
SRELEASE2

SFREQ3 0xFFFF  
SFREQ3S S1  
SPW3 0xFFFF  
SPW3S S1  
SCONTR3 0x00FF  
SADSR3 0x00FF  
SGATE3  
SRELEASE3

SFC 0xFFFF  
Sets FC register

SRESFILT 0x00FF  
Sets filter register

SVOL 0x00FF  
Sets volume

SPOTX VAR  
Reads out Potentiometer X

SPOTY VAR  
Reads out Potentiometer Y

SRND VAR  
Reads out Random number

SENV3 VAR  
Reads out Envelope of Voice 3

### **DEBUG / TEST commands** *(used only in emulator, for software tests only)*

#TEST  
Prints out the result of an arithmetical or a boolean expression to the screen.  
10 #TEST A\*23+(45/D[12])\*2

#BREAK

Stops the program running, sets the processor into HALT state.  
10 #BREAK

## DIRECTIVES

This is about compiling directives to set the compiler to different modes of operation

### ASM-END

Inline assembly

It is possible to insert assembly lines between the BASIC lines, these assembly lines must be written between „ASM” and „END” reserved words like this:

```
10 CLS
20 PRINT „Assembly inlines”
30 ASM
MOV B,C
MVI C,0x12A0
STA [0xC000]
END
40 X=6*Y
50 PRINT „Code ended”
```

### #UNNUMBERED

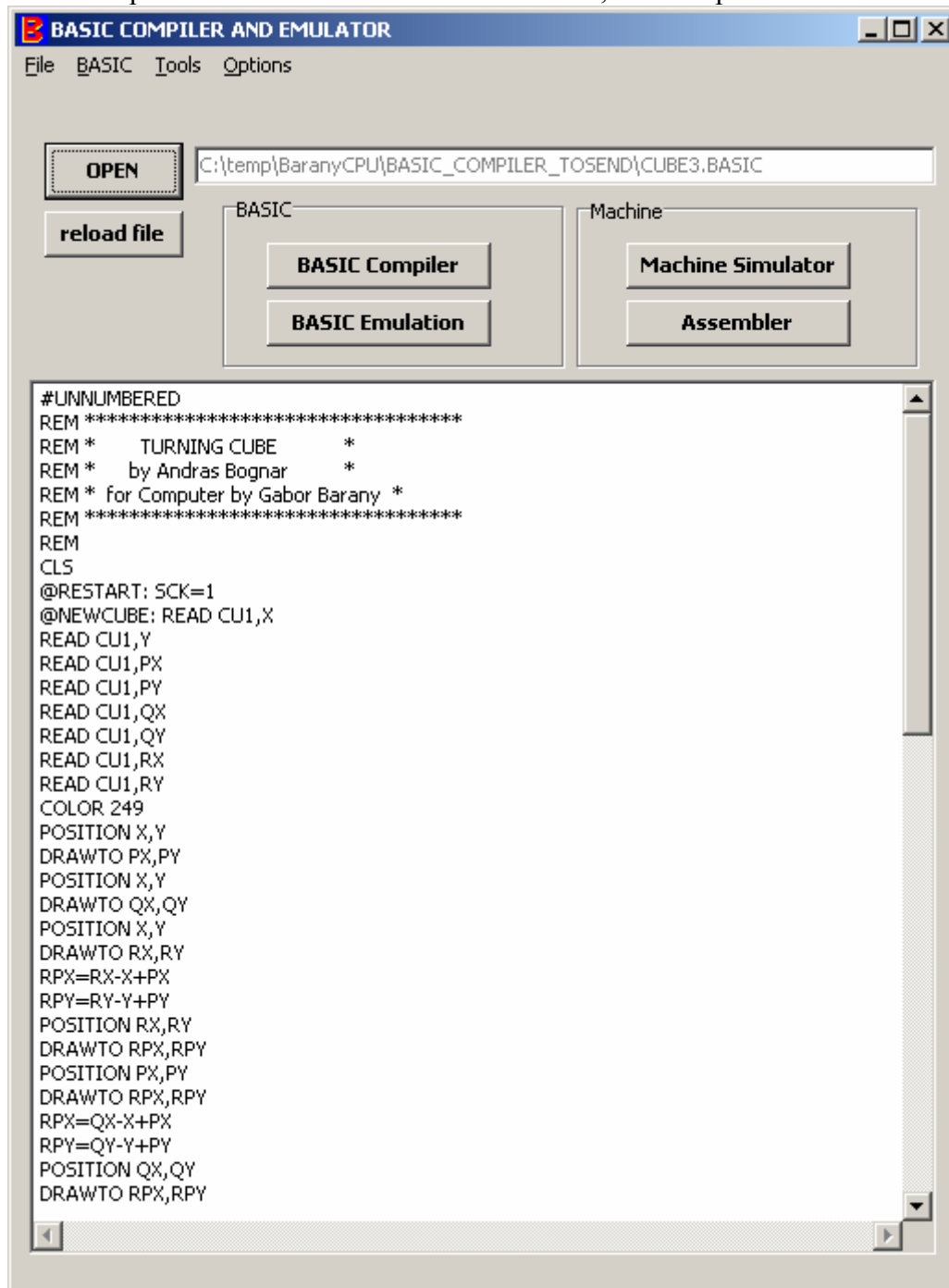
Using un-numbered BASIC lines, make the BASIC compiler to number automatically the BASIC lines. With this directive we can leave the strating [Command linenum] number. The code must start with „#UNNUMBERED” directive. Labels must be used for jumping commands like GOTO, GOSUB, IF-THEN. See example below:

```
#UNNUMBERED
CLS
@RESTART: SCK=1
@NEWCUBE: READ CU1,X
READ CU1,Y
READ CU1,PX
COLOR 249
POSITION X,Y
DRAWTO PX,PY
POSITION X,Y
GOTO @IDE
X=Y*2
ASM
MOV B,A
STA [0xC005]
END
IF X<>2 THEN @ODA
GOSUB @SUBROUTINE2
```

DRATO QX,QY  
@IDE:  
CLS

## USING IDE (*Integrated Development Environment*)

You can open a \*.BASIC file and edit and save it , and compile and start emulation:



### Buttons:

OPEN: opens the .BASIC file.

Reload file: Reloads the file into the editor.

BASIC compiler: Compiles the BASIC code to machine code

BASIC emulation: Emulates the BASIC lines in the emulator

Machine simulator: Simulates the compiled machine code in the machine simulator

Assembler: Opens and assembles the intermediate created ASM file. It is only for hacking in the intermediate assembly file to modify the compilation.

**HotKeys:**

CTRL+O: Open a \*.BASIC file to load it to the editor.

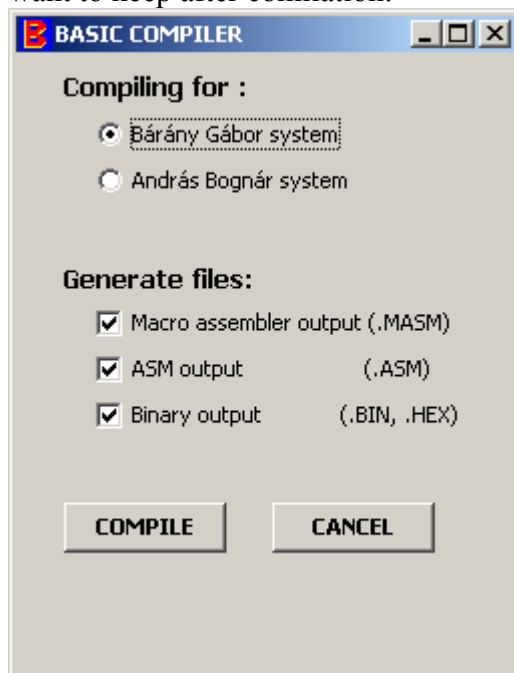
CTRL+S: Save the edited file to a \*.BASIC file.

**BASIC COMPILER**

Compiles the opened BASIC code into machine code.

Only Bárány Gábor system is implemented, András Bognár system option is dummy.

Creates intermediate files and output files, you can choose from the menu which files you want to keep after compilation:



Intermediate files:

.MASM	-macro assembler output
.ASM	-assembly file
.OBJ	-assembler intermediate file

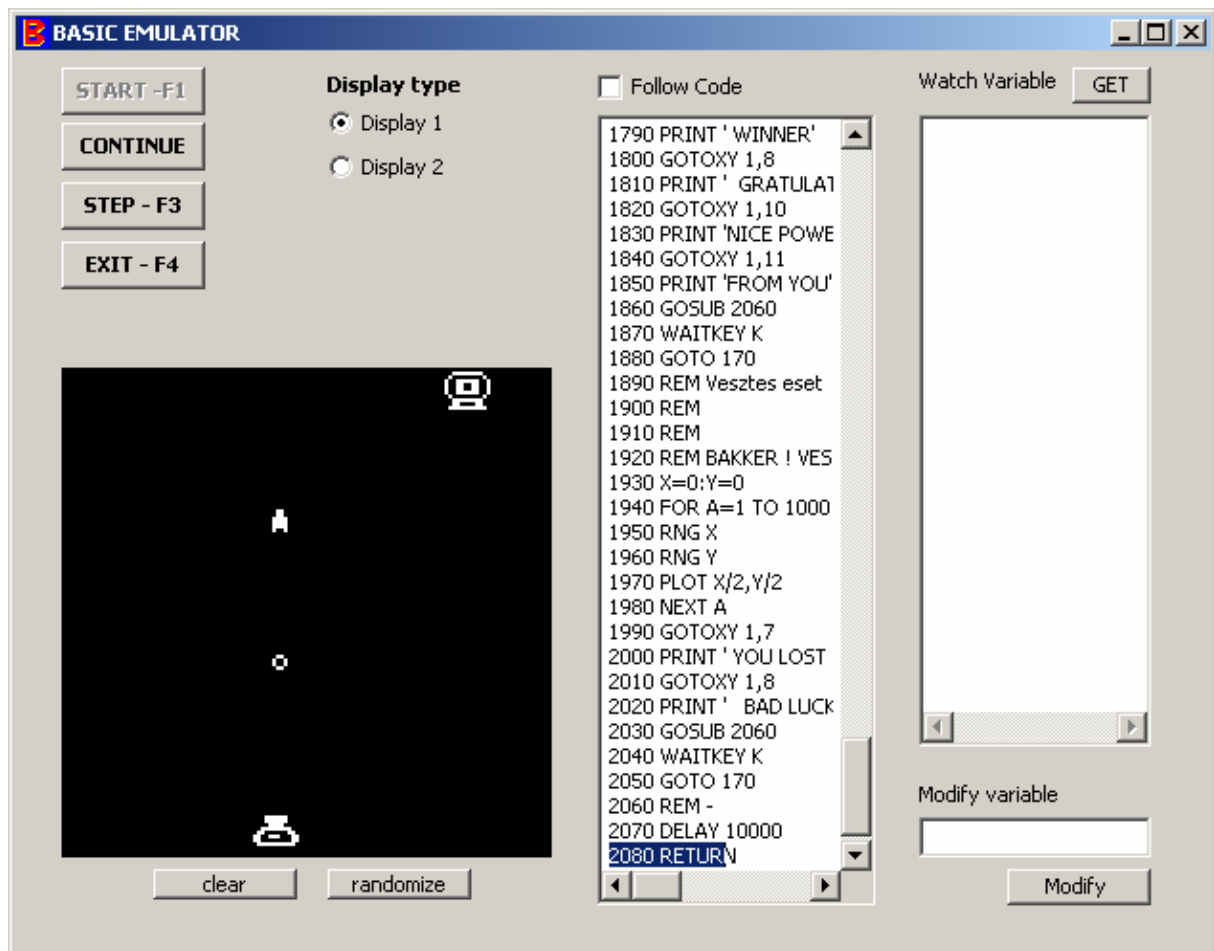
Output files:

.BIN	-binary file
.HEX	-machine code in hexadecimal text file.

**BASIC EMULATOR**

Starts emulation of the BASIC code.

The code will appear in edit field, but editing is disabled.



## Buttons:

„Start”

Hotkey: **F1**

Starts emulation of BASIC code.

„Break” / „Continue”

Hotkey: **F2**

Stops the running code, this button changes to „Continue” immediately, pressing it again, the BASIC code continues from the point where it is stopped before.

„Step”

Hotkey: **F3**

When code is not running, the code is stopped, it can be executed step by step.

Only one command line is executed when „Step” is pressed, and returns to stopped mode after the execution. It is for debugging the code.

„Exit”

Hotkey: **F4**

Exiting from emulator.

„Display Type”

The emulated screen display can be changed to another.  
Resolution and color depth is changed.

[Dummy]

Not yet implemented

„Follow code”

Hotkey: **F12**

When checked, the emulator always highlights the currently executed BASIC line, and scrolls to it when it is invisible to make it visible.

It has high CPU usage, maybe only „hot keys” remain usable. (Press F2 to stop it)

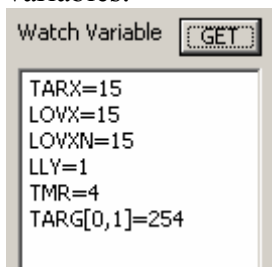
„Watch variable”

You can observe variables during running of code.

Type in the variables to observe into edit window:



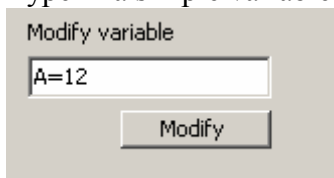
If the program is running then variables completed automatically with its values after every execution of each line. If program is stopped then simply press „GET” to see values of variables.



„Modify variable”

You can modify variable here.

Type in a simple variable modifier expression into edit box, and press „Modify”

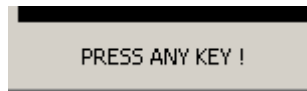


## Keyboard

Pressing any key on keyboard keycode is stored into buffer that „KEYBINPUT” and

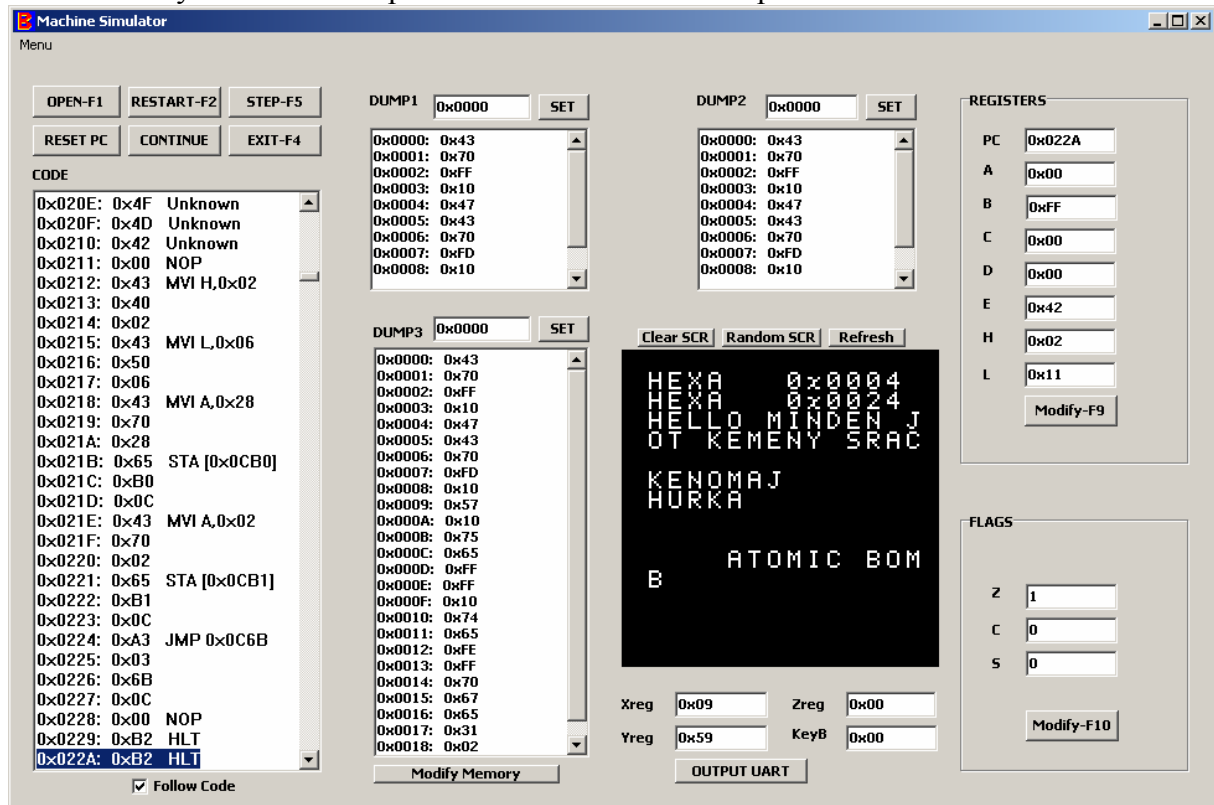
„WAITKEY” reads out.

If „WAITKEY” is executed a message is appears at the bottom while waiting for a keypress:



## MACHINE SIMULATOR

This simulates the real computer  
Automatically the recent compiled machine code will be opened



You can see here the disassembled machine code and 3 memory dumps that's start address can be set, and you can see the machine registers and processor flags too.

With „Modify” buttons you can modify the processor registers and flags too.

The graphic display is operating as the real computer's display, Xreg and Yreg and Zreg are displayed. The Keyb register belong to the keyboard hardware.

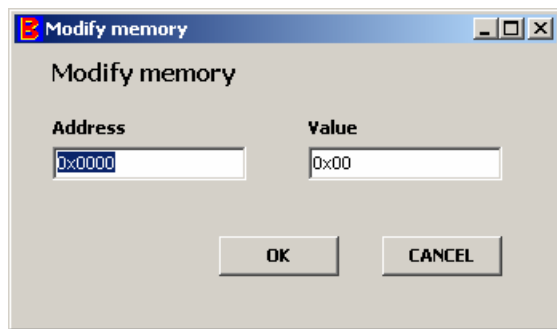
### OUTPUT UART:

Writing to the UART writes a file buffer that can be written out to a file with this button.

### MODIFY MEMORY:

It modifies the content of the operative memory.





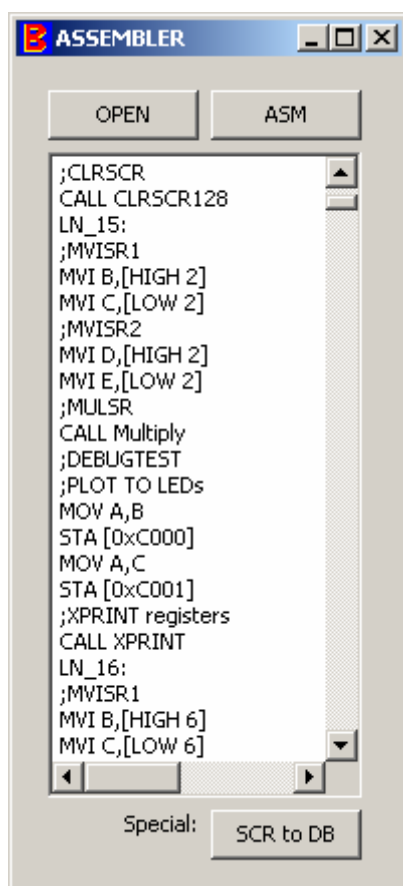
FOLLOW CODE:

Hotkey: **F12**

Switching it off causes that only the display will be updated and the code will run much faster, otherwise every dump, registers, flags will be updated during the execution of the code.

## ASSEMBLER

It automatically opens the currently compiled BASIC code assembly intermediate file. You can load a new ASM file or you can hack the intermediate ASM file here and re-assemble that.



## EXAMPLE CODE

Here is a code example:

```
10 REM *****
20 REM *      LOVO JATEK                      *
30 REM *    by Andras Bogнар                *
40 REM *  for Computer by Gabor Barany      *
50 REM *****
60 REM
70 REM GOMBOK: Hasznald a numerikus bilentyuzetet!
80 REM 1-Balra mozog
90 REM 3-Jobbra mozog
95 REM 5-Loves
98 REM
100 DIM LOVO[3,2]:DIM LOVODEL[3,2]
110 DIM TARG[3,3]:DIM TARGDEL[3,3]
120 GETMAXX MX
130 GETMAXY MY
140 GETMAXCOLORDEPTH CM
150 CM=CM-1
160 MX=MX-1
170 MY=MY-1
180 CLS
185 RESTORE LOVOPIC:RESTORE TARGPIC
190 FOR Y=0 TO 1
200 FOR X=0 TO 2
210 READ LOVOPIC,LOVO[X,Y]:LOVODEL[X,Y]=0
215 IF LOVO[X,Y]=1 THEN LOVO[X,Y]=CM
220 NEXT X
230 NEXT Y
240 FOR Y=0 TO 2
250 FOR X=0 TO 2
260 READ TARGPIC,TARG[X,Y]:TARGDEL[X,Y]=0
265 IF TARG[X,Y]=1 THEN TARG[X,Y]=CM
270 NEXT X
280 NEXT Y
290 REM LOVO es TARG kezdokoordinatai
300 LOVX=15:LOVY=MY-2
310 TARX=10:TARY=1:DIR=1:TMR=0
315 LLY=1:LLX=LOVX:TLY=TARY+3:TLX=TARX:LOVXN=LOVX
320 PUTPATTERN TARG,TARX,TARY,TARX+2,TARY+2
330 PUTPATTERN LOVO,LOVX,LOVY,LOVX+2,LOVY+1
340 REM data lines for game elements
350 DATA LOVOPIC 0,1,0,1,1,1
360 DATA TARGPIC 1,1,1,1,0,1,1,1,1
370 REM ----- main loop -----
380 IF DIR=2 THEN 500
385 REM Celtargy jobbra mozgatasa
390 TARX=TARX+1
400 IF TARX+3>=MX THEN DIR=2
410 PUTPATTERN TARGDEL,TARX-1,TARY,TARX+1,TARY+2
420 PUTPATTERN TARG,TARX,TARY,TARX+2,TARY+2
440 GOTO 540
445 REM Celtargy balra mozgatasa
500 TARX=TARX-1
510 IF TARX<=1 THEN DIR=1
520 PUTPATTERN TARGDEL,TARX+1,TARY,TARX+3,TARY+2
530 PUTPATTERN TARG,TARX,TARY,TARX+2,TARY+2
535 REM Billentyuzet vizsgalata
540 KEYBINPUT KEY
```

```

550 IF KEY=49 THEN 630
570 IF KEY=51 THEN 680
580 IF KEY=53 THEN 710
590 IF KEY=32 THEN 710
600 GOTO 830
610 REM Billentyuzet rutinok
620 REM Mozgatas jobbra , balra, lovedek inditasa
630 LOVXN=LOVX-1
640 IF LOVXN<1 THEN LOVXN=1
650 GOTO 800
680 LOVXN=LOVXN+1
690 IF LOVXN+3>=MX THEN LOVXN=LOVXN-1
700 GOTO 800
705 REM Lovedek inditasa es mozgatas feltele
710 IF LLY>(TARY+4) THEN 800
720 LLX=LOVX+1:LLY=LOVY-1
790 REM Tankunk kirajzolsa, elotte torlese
800 PUTPATTERN LOVODEL,LOVX,LOVY,LOVX+2,LOVY+1
810 PUTPATTERN LOVO,LOVXN,LOVY,LOVXN+2,LOVY+1
820 LOVX=LOVXN
825 REM Celtargy lovedeknek kezelese, idozito csokkentese
830 TMR=TMR+1
840 IF TMR>30 THEN TLY=TARY+3:TLX=TARX:TMR=0
850 IF TLY<(LOVY-1) THEN 900
855 PUTPIXEL TLX,TLY,0
860 GOTO 940
900 PUTPIXEL TLX,TLY,0
910 PUTPIXEL TLX,TLY+1,250
920 TLY=TLY+1
925 IF (TLY>=(LOVY-1)) AND ((LOVX=TLX)OR((LOVX+1)=TLX)OR((LOVX+2)=TLX)) THEN 2010
930 REM Lovedekunk mozgatas
940 IF LLY>(TARY+4) THEN LLY=LLY-1:GOTO 970
950 PUTPIXEL LLX,LLY-1,0
960 GOTO 1000
970 PUTPIXEL LLX,LLY,0
980 PUTPIXEL LLX,LLY-1,250
985 IF (LLY<=(TARY+4)) AND ((TARX=LLX)OR((TARX+1)=LLX)OR((TARX+2)=LLX)) THEN 1010
990 REM Ugras vissza az elejere
1000 GOTO 370
1010 REM
1020 REM
1030 REM HURRA ! NYERTEL
1040 PLOT 5,5
1050 PLOT 5,6
1060 PLOT 5,7
1070 PLOT 4,7
1080 PLOT 7,4
1090 PLOT 7,5
1100 PLOT 7,6
1110 PLOT 7,7
1120 PLOT 7,8
1130 PLOT 8,4
1140 PLOT 8,6
1150 PLOT 8,8
1155 PLOT 10,3
1160 PLOT 10,4
1170 PLOT 10,5
1180 PLOT 10,6
1190 PLOT 10,8
1200 WAITKEY K

```

```
1210 GOTO 120
2010 REM
2020 REM
2030 REM BAKKER ! VESZTES VAGY
2040 PLOT 8,8
2050 PLOT 9,8
2060 PLOT 10,8
2070 PLOT 8,9
2080 PLOT 9,9
2090 PLOT 10,9
2100 PLOT 8,10
2110 PLOT 9,10
2120 PLOT 10,10
2130 PLOT 12,10
2140 PLOT 12,9
2150 PLOT 13,10
2160 PLOT 13,9
2170 PLOT 15,10
2180 WAITKEY K
2190 GOTO 120
```