

HUG-BASIC-1K

HUGH ANDERSON FEB 1980*

HUG-BASIC

Hug-Basic is a small high-level language for the HUG-1802 computer. The language can be stored in RAM or PROM, and uses 1K bytes of memory. Input is via the hex keyboard; output appears on the screen as an 8 line by 16 character display.

Despite the extremely small size of the Hug-Basic interpreter, this package provides a good introduction to BASIC programming. The HUG-1802 with a Hug-Basic PROM or an extra 1K RAM mounted on-board, is the fastest, cheapest way to enter the computer world at an advanced level.

FEATURES

Lets look at some of the features of 'no-frills' Hug-Basic computing.

1. 26 Variables; A to Z, range 0 to 255. +, -, *, ÷, are all done to 16 bit accuracy.
2. The Basic programs are stored in a very compact form, so that the 512 spare bytes in the HUG-1802 is enough for quite a long program - 40 or so lines - and the commands in Hug-Basic do much more than CHIP-8 or Machine Language commands.
3. A computed GOTO and GOSUB is provided. This is a powerful addition to BASIC, and can be used to replace multiple IF-THEN statements.
4. DEBUGGER - If your program is not working the way you want it to, you can step through it slowly by holding down the STEP pushbutton of the computer. The number of the current line being executed will be displayed on the screen, and the program will slow down to about 4 lines per second.
5. All input is through the hex keyboard, and the special BASIC keywords can be entered by pressing only 2 keys. This makes program entry very fast.
6. Approximately 20 levels of subroutine nesting allowed. Multiple statements per line.
7. PLOT, PEEK, POKE, USR, RND all provided.
8. There are many other features of Hug-Basic which you will discover as you play with it.

This is probably the smallest BASIC interpreter ever written and you may find it educational to disassemble the code and 'see how its done'.

*Scanned in (and, hopefully, fixed scanning errors) in April/May 2023.

LIMITATIONS

In such a small BASIC there have to be limitations and 'bugs'. I'll mention the worst of them here so you know what to expect.

- The characters on the screen are formed from a coarse 3*5 matrix, so some of the letters look peculiar - in particular W and M. You soon get used to the display however.
- The Editor is unbelievably primitive - there are only two possible functions: **DE**lete the last line of a program or **APP**end (Add a line to the end of a program). This encourages you to get the program right the first time.
- 8 Bit, integer only arithmetic - This is not as bad as it sounds as all intermediate working in a calculation is done in 16 Bits. For games and graphics applications this is not a limitation at all.
- There is no 'immediate mode of operation'.
- When entering programs there is one strict formatting rule - DO NOT USE SPACES WILLY NILLY (whatever that means) - the general idea is that once you've started inputting a statement, do not use spaces until that statement is finished. The exception being within the parentheses of a PRINT statement.

OUTLINE

Hug-Basic 'Programs' are stored in the computer from location **#0600** to location **#07FF** if you have 3K of memory or from **#0600** to **#0BFF** if you have the full 4K of memory that the HUG-1802 can use without modifications. These 'Programs' are not executed directly by the microprocessor - instead Hug-Basic reads the 'Program' location by location and then decides what to do at each step. This indirect execution is called interpreting, and Hug-Basic is called an interpreter.

Hug-Basic programs run a little bit slower than comparable CHIP-8 ones, which in turn run slower than programs written in machine language. So, WHY do we bother with them? The advantage is that they are more readable, understandable and they are easier to change if there is something wrong. Compare the three short program segments below:

1	BASIC	LET A = A + 1	
2	CHIP-8	7A 01	
3	MACHINE CODE	F8 01	LOI #01
		F4	ADD
		73	STXD (etc,etc...)

The BASIC program is fairly clear in telling us what it wants to do (Add 1 to a number called A), but the CHIP-8 and MACHINE CODE segments are not at all readable. Incidentally, the machine code given above is only a section of the code needed to add 1 to a number stored in the computer's memory. CHIP-8 is ideal for Video Games, Hug-Basic is intended more for Question and Answer games, and for doing MATH.

There are commands in Hug-Basic for doing a wide range of things such as:

- **PRT** For **PR**inTing messages on the screen
- **PLT** For **PLo**Tting dots on the screen
- **END** For **END**ing a program and so on.

GETTING STARTED

CLEARING THE MACHINE

After first powering up the HUG-1802, the memory contains all sorts of random values. The Hug-Basic interpreter hiccups sometimes when reading these values so it is a good idea to clear the memory. Here is a short program that sets all the memory to one value, in this case to **#5B**. Load these values at location **#0400**:

loc	0400	F8
	0401	5B
	0402	5B

Now hit **(RESET) '6'**. This command runs a machine language program starting at the memory location displayed on the screen. Now load these values into location **#0600**:

loc	0600	32
	0601	00
	0602	00

The machine is now initialised and all set to run Hug-Basic.

LOADING HUG-BASIC

If you are using a Hug-Basic PROM there is no need to do this step. However if you have a tape version, you must load it into locs **#0C00 - #1000**. Do this by setting the TAPELOAD addresses:

loc	0400	0C
	0401	00
	0402	10
	0403	00

Start the tape running, and when you hear the *Leader* tone, press **(RESET) '4'**. The Hug-Basic interpreter will now be loaded into the computer's memory.

RUNNING HUG-BASIC

To start things going, set the memory pointer to **#0C00** and press ... **(RESET) '6'** (Run). HUG-BASIC will be displayed on the bottom of the screen, and the machine will then wait for you to press a key. At this stage there are two possible actions:

- Press **'O'** and a basic program that you have previously entered will start.
- Press **'I'** and your basic program will be listed from the beginning, and then you will enter the EDIT mode of operation.

Since we have just started up, let's press **'I'**. A **'O'** will be displayed - this is a listing of those three bytes we entered at **#0600**. Press **FF** and then **00** - the HUG-BASIC message will be displayed again - we are now back where we were before, ready to RUN or LIST a program.

Press **'I'** again. Nothing at all will happen; the machine is waiting for you to start loading a BASIC program, and its time to learn a few editor commands.

First of all, have a look at the various KEYCODE CHARTs (Appendix 5). You will need to keep these handy whenever you are using the machine.

ENTERING PROGRAMS

(YOU'VE JUST PRESSED 'I'). First of all, your program so far is listed. In this case, there is no program to list. NOTE that the listing can be halted at any time by holding down the **STEP** key. Next, enter the hex keycodes for your program using the chart.

- To **EXIT** the editor at any time, press **00**.
- To **DELe**te the last line of your program at any time, press **FF**.
- To enter a number, key **31 XX**, where **XX** is the number in hexadecimal. (ex: #65 = **101** decimal)
- To enter a linenumber, key **32 XX**, as above.

REMEMBER: DO NOT IMBED SPACES IN STATEMENTS.

Those are the rules, so let's load in the first program. Just for fun we will use a program that draws dots all over the screen at random. In this BASIC, the program looks like this:

```
10    PLT RND,RND
20    GOTO 10
```

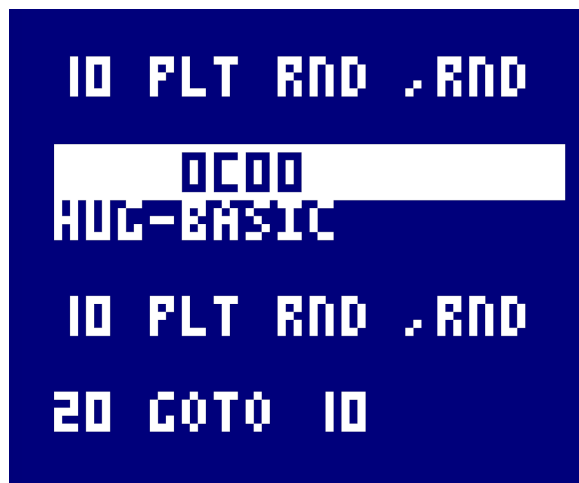
The **PLT** instruction plots a point at **X** and **Y** co-ordinates given by the next two values. In the program, the next two values are **RaNDom** and **RaNDom**, which have random values as the name suggests. To enter this program, key in the following:

```
32 0A 30 85 36 2C 36
32 14 30 EB 31 0A
```

(now exit with **00** and run with **0**). Dots will start appearing all over the screen.

Unfortunately, there is no way to stop this program except by resetting the computer. Press '**6**' to run HUG-BASIC again. The program is still in the machine and can be **LIST**ed or **RUN** again. ('**I**' or '**O**').

Note from 2023: When you load this program, it will look like this when listed:



```
10 PLT RND ,RND
0C00
HUG-BASIC
10 PLT RND ,RND
20 GOTO 10
```

GENERAL NOTES

These notes are not intended as an introduction to the BASIC language. There should be plenty of good books in your local library. However, the format of BASIC programs is shown below.

1. A Hug-Basic program consists of one or more BASIC LINES.
2. A Hug-Basic LINE consists of a LINE-NUMBER followed by one or more BASIC STATEMENTS.
3. A Hug-Basic LINENUMBER is entered by keying **32 XX** as shown before.
4. Hug-Basic STATEMENTS are entered by keying in their codes from the chart.

The remainder of this manual is made up of descriptions of the various statements available in Hug-Basic-IK.

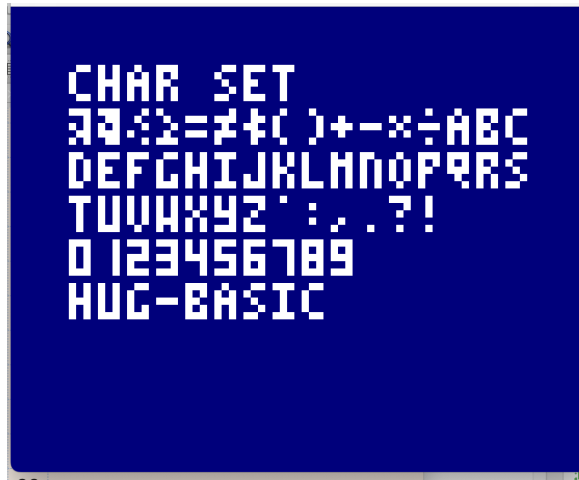
STATEMENTS

PRINT: The **PRinT** statement appears on the screen as **PRT**. It's keycode is **63**. There are three ways of using **PRT**:

1. **PRT "string"** - This will print a string of characters within the inverted commas.
2. **PRT (some value)** - This will evaluate some arithmetic expression that you give it, and then print the result as a 3-digit decimal number. Leading zeroes do not appear, and it is left-justified.
3. **PRT \$(some value)** - Again the arithmetic expression is evaluated, but this time the value is taken to be a character code.

10	PRT "(cls)CHAR SET(1f)"	32	0A	30	63	2A	02
		12	17	10	21	30	22
		14	23	01	2A		
15	LET A = 2	32	0F	30	BC	10	07
		31	02				
20	A = A + 1 PRT \$A	32	14	30	10	07	10
		0C	31	01	30	63	09
		10					
Example:	25 IF A ≠ 48 THEN GOTO 20	32	19	30	B1	10	08
		31	30	3A	EB	31	14
	30 PRT "(1f)" A = O	32	1E	30	63	2A	01
		2A	30	10	07	31	00
	35 PRT A: A = A + 1	32	23	30	63	10	2B
		30	10	07	10	0C	31
		01					
	40 IF A ≠ 10 THEN GOTO 35	32	28	30	B1	10	08
		31	0A	3A	EB	31	23

Note from 2023: When you run this program, it will look like this:



STATEMENTS

(Continued)

LET: Assignment statements can be preceded by the word **LET**, but it is not strictly necessary. The keycode is **BC**, and appears on the screen as **LET**.

Example:

LET $A = A + 5$	or $A = A + 5$
LET $B = ((\text{RND} \div 16) + 5) \div 3$	

Always use lots of brackets to separate out the operations that you wish to do. You can only have two elements at any one level of bracketing:

e.g. $((X+Y)+Z)+D$ **NOT** $X+Y+Z+D$

The railroad diagrams for evaluating the expressions are given in Appendix 2.

REMARK: Or **REM**. When the interpreter finds **REM** on a line, it skips over the line and starts on the next line. This is useful for documenting a program. The keycode is **96**, and appears as **REM**.

END: When **END** is encountered, the program stops, and control returns to the user. (HUG-BASIC is shown and the machine waits for you to press 'I' or 'O'). The keycode is **91**. **END** can be used anywhere - within subroutines etc.

INPUT: The **IN**put statement causes the BASIC to stop and wait for a key to be pressed on the hex keyboard. The value of the key is put in to the variable which is next. The keycode is **47**.

Example:	<pre> 10 IN A: IN B 15 PRT (A * 10) + B 20 IN C END </pre>	<pre> 32 0A 30 47 10 2B 30 47 11 32 0F 30 63 0A 10 0E 31 0A 0B 0C 11 32 14 30 47 12 30 91 </pre>
-----------------	---	--

This program waits for two keys to be pressed, and then prints them as a decimal number. Line 20 above is a handy line to put on the end of your programs - it causes a wait until a key is pressed before returning to the mainline.

PLOT: **PLoT (value), (value)** - This instruction evaluates the two arithmetic expressions, and plots a point on the screen using the first and second values as **X** and **Y** co-ordinates respectively. **(0,0)** is the top left hand corner, and **(63 ,47)** is the bottom right hand corner. The keycode is **85**, and it looks like **PLT**.

Example:	<pre> 10 PRT (cls): A = 0 15 PLT A,A 20 A = A + 1 25 IF A = 47 THEN END 30 GOTO 15 </pre>	<pre> 32 0A 30 63 2A 02 2A 2B 30 10 07 31 00 32 0F 30 85 10 2C 10 32 14 30 10 07 10 0C 31 01 32 19 30 B1 10 07 31 2F 3A 91 32 1E 30 EB 31 0F </pre>
-----------------	---	--

STATEMENTS

(Continued)

POKE: **POKE (location), (data)** - This instruction **POKEs** a data value into any location in the computer's memory. To **POKE** locations higher than location 255, use an arithmetic expression. The keycode is **9F**.

Example:

10	POKE	128*9,255	32	0A	30	9E	31	80
			0E	31	09	2C	31	FF

This will put a little bar at the top left hand corner of the screen.

USR: **USR (location)** - This allows one to run a machine language program at the location given. The program runs with **R3** as the program counter, and to return to HUG-BASIC, set the PC to 4. (**D4**). You can use registers 7,C,D,E and F as you wish. The keycode for **USR** is **C9**.

Example:

10	USR	224	32	0A	30	C9	31	E0
----	-----	-----	----	----	----	----	----	----

This runs a machine language program in the CHIP-8 PROM which erases the screen.

GOTO: **GOTO (some value)** - When the interpreter finds a **GOTO**, the expression afterwards is evaluated. Then the computer searches for a linenurnber to match that value, and then carries on from this new line. If no match is found, the program ends. The 'computed' nature of the **GOTO** (and **GOSUB**) is useful for doing multi-way branches - say after an input from the keyboard. It can save many **IF-THEN** statements. Have a look at the MATH-QUIZ game to see an example of the usefulness of the computed **GOTO**. The keycode for **GOTO** is **EB**.

GOSUB: **GOSUB (some value)** - **GOSUB** is worked out as for **GOTO** above, but before jumping to the new line, the machine 'saves it's place'. When the machine next encounters a **ReTurN** statement, control returns to where it left off before. If you have to do some operations several times in a program, it is often easiest to make it a subroutine, and call it as often as you want from the main line of the program with **GOSUB**. There are examples of subroutine usage in the MATH-QUIZ game (Appendix 4). If a subroutine calls a subroutine we are 'nesting' subroutines. In this version of BASIC you can 'nest' subroutines about 20 deep - but be careful doing it. The keycode for **GOSUB** is **DD**.

RETURN: **ReTurN** - returns from a **GOSUB**. The keycode for **RTN** is **D2**.

IF - THEN: **IF (some condition) THEN (carry on here)** The condition can take one of three forms:

1. **IF (value)=(value) THEN**
2. **IF (value)≠(value) THEN**
3. **IF (value)≥(value) THEN**

The two expressions are evaluated and compared. If the condition is true, execution continues after the **THEN**, otherwise we skip to the next line. Key codes are **B1 .. IF**, **3A .. THEN**.

FUNCTIONS

PEEK: **PEEK (value)** - This is the opposite to the **POKE** statement, and it allows you to look at any memory location during a BASIC program. The value of the **PEEK** function is an 8-bit value found at the **PEEK** location. The keycode for **PEEK** is **33**.

RANDOM: **RaNDom** - The value of this function is a random value between 0 and 255. It may only be used on the right hand side of a statement. (For example you cannot say **LET RND = 10**). The keycode for **RND** is **36**.

VARIABLES

There are 26 1-byte variables named **A...Z**.

Variable **Z** is the only unusual one, as it's value will change after a **PLT** or **PRT** statement. It will have the value **1** if a previously plotted point was overwritten. Otherwise it will have the value **0**.

Mathematics - add, subtract, multiply and divide are all done to 16-bit accuracy, but when the values are stored back in the variables, they revert to 8-bit values. All divisions are rounded.

APPENDIX 1 (SUMMARY OF HUG-BASIC)

STATEMENTS:

- LET (variable)=(arithmetic expression) LET optional
- IF (condition) THEN (statement)
- GOTO (arithmetic expression)
- GOSUB (arithmetic expression) .. RTN
- PRT "(string)" or PRT (exp) or PRT \$(exp)
- PLT (exp), (exp) (X,Y)
- POKE (exp), (exp) (location,data)
- IN (variable name)
- USR (exp)
- END
- REM

FUNCTIONS:

- PEEK (exp)
- RND

FEATURES:

- A (*string*) above can be any sequence of printable characters, terminated by an inverted comma "".
- A (*condition*) above is any two arithmetic expressions separated by = equals, \neq not equal, or \geq greater than or equal.
- A (*variable name*) above is any of the letters A .. Z.
- Arithmetic expressions are evaluated using the railroad diagram given in the next appendix.

REMEMBER NO SPACES WITHIN STATEMENTS.

RAILROAD DIAGRAMS:



APPENDIX 3 (OTHER TECHNICAL DETAILS)

MEMORY MAP:

0000	03FF	CHIP-8/MONITOR
0400	0401	TAPE START ADDRESS
0402	0403	TAPE END ADDRESS
0404	0405	MEMORY POINTER/MONITOR
0406		SCRATCH
0407		Y CURSOR
0408		X CURSOR
0409	0438	SUBROUTINE STACK
0439	0450	EXPRESSION EVALUATION STACK
0451	045D	DISPLAY WORKSPACE
045E	0460	THREE DISPLAY BYTES
0461	0465	CHAR WORKSPACE
0466	047F	BASIC VARIABLES A .. Z
0480	05FF	SCREEN DISPLAY MEMORY
0600	0BFF	BASIC TEXT STORAGE AREA
0C00	0FFF	BASIC INTERPRETER

REGISTER USAGE:

R0	DMA POINTER
R1	INTERRUPT SERVICE
R2	SUBROUTINE AND MACHINE STACK
R3	PROGRAM COUNTER
R4	RETURN PC .. SCRT
R5	CALL PC . . SCRT
R6	DATA STACK7/VARIABLE POINTER
R7	SPARE
R8	TIMERS
R9	RANDOM NUMBER GENERATOR
RA	"I" POINTER
RB	BASIC TEXT POINTER
RC	SPARE
RD	SPARE
RE	SPARE
RF	SPARE

ERROR HANDLING:

When the Hug-Basic interpreter detects a program error or a divide-by-zero error, execution stops and your program is listed from where the error was found. This gives a good indication of where the error was.

APPENDIX 4 (MATH-QUIZ)

1	PRT "(cls) MATH QUIZ(lf)	32	01	63	2A	02	1C	10	23	17	30
	1 + (lf) 2 - (lf)	20	24	18	29	01	31	01	30	0C	01
	3 * (lf) 4 ÷ (lf)	31	02	30	0D	01	31	03	30	0E	01
	5 EXIT"	31	04	30	0F	01	31	05	30	14	27
		18	23	2A							
2	IN A S = 0 P = 0	32	02	47	10	22	07	31	00	1F	07
		31	00								
3	GOSUB A * 10 GOTO 1	32	03	DD	10	0E	31	0A	EB	31	01
10	A = RND ÷ 7 B = RND ÷ 8	32	0A	10	07	36	0F	31	07	11	07
	D = A + B O = 12	36	0F	31	08	13	07	10	0C	11	1E
		07	31	0C							
11	GOSUB 45 IF P ≠ 10	32	0B	DD	31	2D	B1	1F	08	31	0A
	THEN GOTO 10	3A	EB	31	0A						
12	GOSUB 60 RTN	32	0C	DD	31	3C	D2				
20	B = RND ÷ 7 A = B + (RND ÷ 7)	32	14	11	07	36	0F	31	07	10	07
	D = A - B O = 13	11	0C	0A	36	0F	31	07	0B	13	07
		10	0D	11	1E	07	31	0D			
21	GOSUB 45 IF P ≠ 10	32	15	DD	31	2D	B1	1F	08	31	0A
	THEN GOTO 20	3A	EB	31	14						
22	GOTO 12	32	16	EB	31	0C					
30	B = RND ÷ 22 A = RND ÷ 25	32	1E	11	07	36	0F	31	16	10	07
	D = A * B O = 14	36	0F	31	19	13	07	10	0E	11	1E
		07	31	0E							
31	GOSUB 45 IF P ≠ 10	32	1F	DD	31	2D	B1	1F	08	31	0A
	THEN GOTO 30	3A	EB	31	1E						
32	GOTO 12	32	20	EB	31	0C					
40	B = RND ÷ 26 A = B * (RND ÷ 30)	32	28	11	07	36	0F	31	1A	10	07
	D = A ÷ B O = 15	11	0E	0A	36	0F	31	1E	0B	13	07
		10	0F	11	1E	07	31	0F			
41	GOSUB 45 IF P ≠ 10	32	29	DD	31	2D	B1	1F	08	31	0A
	THEN GOTO 40	3A	EB	31	28						
42	GOTO 12	32	2A	EB	31	0C					
45	PRT "(lf)WHAT IS(lf)"	32	2D	63	2A	01	26	17	10	23	30
	PRT A PRT \$0 PRT B	18	22	01	2A	63	10	63	09	1E	63
	PRT "? (lf)"	11	63	2A	2E	01	2A				
46	IF 9 ≥ D THEN IN I	32	2E	B1	31	09	06	13	3A	47	18
	GOTO 48	EB	31	30							
47	IN I IN J I = (i * 10) + J	32	2F	47	18	47	19	18	07	0A	18
		0E	31	0A	0B	0C	19				
48	PRT I P = P + 1	32	30	63	18	1F	07	1F	0C	31	01
	IF I = D THEN S = S + 1	B1	18	07	13	3A	22	07	22	0C	31
	PRT "(lf)OK" RTN	01	63	2A	01	1E	1A	2A	D2		
49	PRT "(lf)NO ITS"	32	31	63	2A	01	1D	1E	30	18	23
	PRT D RTN	22	30	2A	63	13	D2				
50	END	32	32	91							
60	PRT "(lf)SCORE:(lf)"	32	3C	63	2A	01	22	12	1E	21	14
	PRT S PRT " OUT OF 10"	2B	01	2A	63	22	63	2A	30	1E	24
	IN A RTN	23	30	1E	15	30	31	0A	2A	47	10
		D2									

This is a math quiz program. You get ten problems. Afterwards you get a score out of 10.

APPENDIX 5 (KEYCODE CHARTS)

CHARACTER CODE CHART (#01-#30)

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
01	LF	09	\$	11	B	19	J	21	R	29	Z
02	CLS	0A	(12	C	1A	K	22	S	2A	"
03	☒	0B)	13	D	1B	L	23	T	2B	:
04	☒	0C	+	14	E	1C	M	24	U	2C	,
05	☒	0D	-	15	F	1D	N	25	V	2D	.
06	≥	0E	*	16	G	1E	O	26	W	2E	?
07	=	0F	/	17	H	1F	P	27	X	2F	!
08	≠	10	A	18	I	20	Q	28	Y	30	SPACE

(I do not know what characters ☒, ☒ and ☒ were supposed to be.)

EDITOR AND OPERATION CODES CHART

	Code	Meaning	Code	Meaning
After running at #0C00:	0	Run program	1	Run Editor
Within the editor:	00	Exit editor	31	Number: 31 XX
	FF	Delete last line	32	Line number: 32 XX

BASIC KEYWORDS CHART (#33-#EB)

Code	Basic op	Code	Basic op	Code	Basic op	Code	Basic op
33	PEEK	85	PLOT	B1 ... 3A	IF ... THEN	DD	GOSUB
36	RANDOM	91	END	BC	LET	EB	GOTO
47	INPUT	96	REM	C9	USR		
63	PRINT	9F	POKE	D2	RETURN		