RFC 7373
BrainF***ingBears Design Specification

1. Introduction

For the purposes of this document, the following words may be used with a different meaning than usual: flexible, easy to understand, compact, expressive, imperative, functional, reusable, agile, performant, precise, programming language, simple, compact, RISC-like.

This document outlines how a reference implementation of this language should work.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", “SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

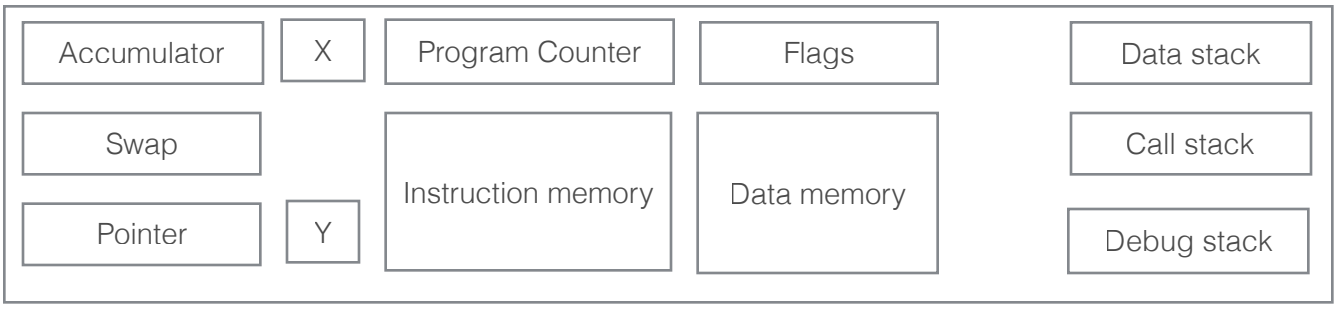
2. Definition

BrainF*ingBears** (pronounced: /bɹeɪnˈæst(ə)rɪskˈæst(ə)rɪskˈæst(ə)rɪskˈæst(ə)rɪskɪndʒiːbeə(r)s/, short: BFB) is RECOMMENDED to be a flexible, easy to understand, compact, expressive, imperative, functional, reusable, agile, performant, precise language for novice programmers.

3. Architecture

BrainF***ingBears SHALL NOT be a simple language with a compact, RISC-like instruction set for the manipulation of the different elements of an implementing hardware or interpreter.

The elements of a x87-BFB CPU MAY look like this:



In the interest of readability, the connections MUST be omitted from the chart. The reader SHALL refer to the instruction set and SHALL NOT identify the connection between each register or memory unit.

4. Instruction set

Byte	Description	Byte	Description
0-9	Call subroutine	S	Skip next instruction if accumulator <*
a	Copy memory value to accumulator	t	Skip next instruction if accumulator <=*
A	Load 0 to accumulator	T	Skip next instruction if accumulator >*
b	Break loop	u	Skip next instruction if not accumulator
B	Continue loop	U	Skip next instruction accumulator
c	Clear terminal screen	v	Load memory value into swap
d	Remove the top element of the stack	V	Save swap to current memory
D	Decrease accumulator by 1	w	Load accumulator into swap
e	Push accumulator onto debug stack	W	Save swap to accumulator
E	Clear the debug stack	x	Load memory value into x register
f	Push flags onto data stack	X	Load accumulator into x register
F	Pop data stack into flags	y	Load memory value into y register
i	Exit immediately, code = accumulator	Y	Load accumulator into y register
I	Increase accumulator by 1	z	Save accumulator into memory
l	Decrease memory value by 1	Z	Load memory value into memory pointer
L	Decrease memory value by 10	+	Add memory value to accumulator
m	Increase memory value by 1	-	Subtract memory value from accumulator
M	Increase memory value by 10	*	Multiply accumulator by memory value
n	Increase memory pointer by 1	/	Divide accumulator by memory value
N	Increase memory pointer by 10	%	Modulo accumulator by memory value
p	Decrease memory pointer by 1	?	(Non-blocking) read key into accumulator
P	Decrease memory pointer by 10	.	Output accumulator to screen at x,y
q	Skip next instruction if accumulator !=*	^	Return from subroutine
Q	Skip next instruction if accumulator ==*	{	Begin next subroutine
r	Load 0 into memory pointer	}	End current subroutine
R	Load 0 into memory value	$	Skip next instruction unconditionally
s	Skip next instruction if accumulator >=	#	Skip all instructions until the next line feed

Byte	Description	Byte	Description
'	Set flag marked by accumulator (mod 10)	;	Push memory pointer onto stack
"	Clear flag at accumulator (mod 10)	,	Pop memory pointer from stack
=	Skip if flag at accumulator is set	@	Sleep for 10 ms
~	Skip if flag at accumulator is not set	!	Randomize accumulator
[	Begin loop	\	Complement CMP flag*
]	End loop	I	Unset CMP flag*
>	Push accumulator onto stack	:	Halt execution and enter debug mode
<	Pop stack into accumulator		
)	Push program counter onto stack		
(	Pop program counter from stack		

* The CMP flag MUST be a special flag that which governs the behaviour of the q, Q, s, S, t and T instructions. When set, accumulator is compared with swap, otherwise it's compared with the memory value.

5. Debugging

The implementing engine SHOULD provide multiple debugging options, each of which MAY have a significantly limited usefulness and OPTIONALLY inconvenience the user.

The reference implementation MUST provide the following tools:

* The ability to run the code in debug, slow or very slow mode with the respective *debug, slow*, and *veryslow* command line parameters.
* The ability to enable the event logging at the cost of CPU time and memory usage with the *track* command line parameter.
* The ability to place no more and no less than a single breakpoint in the code at an arbitrary position by providing a number as a command line parameter.
* The debug screen, which SHOULD needlessly obscure part of the active screen and prints the value of each and every register, while providing options to view the event log and the operative memory.
* A memory viewer, which displays the value of all memory slots and SHOULD raise a fatal error if a memory slot has a very long number.
* A searchable, filterable event log viewer which MUST be populated with much more data than desirable or debuggable.

6. Availablilty of reference implementations

The reference implementation of the programming language specified by this document MUST be included in the same directory. It is RECOMMENDED however to visit its GitHub page at https://github.com/baliame/brainfuckingbears,

7. Usage of reference implementation

The reference implementation MUST NOT but SHALL require Python 3 and an ncurses compatible terminal. It is bundled with a reference implementation of a minesweeper game and SHOULD have the file name ‘minesweeper.min.bfb’.

RECOMMENDED usage of reference implementation:
python3 bfb.py <file name> [debug|slow|veryslow] [track] [line number]
where <> indicates a REQUIREd argument, and [] indicates an OPTIONAL argument.

A reference implementation MUST act accordingly with the following commands:
python3 bfb.py minesweeper.min.bfb - run normally.
python3 bfb.py minesweeper.min.bfb debug - start in debug mode
python3 bfb.py minesweeper.min.bfb slow track - run with 100ms delays and track instructions in the event log.
python3 bfb.py minesweeper.min.bfb veryslow track 51 - run with 1s delays between instructions and track instructions in the event log. Enter debug mode when the program counter hits 51.

8. Legal

Any reference or otherwise not reference implementations of the BrainF***ingBears language MUST use the GPLv2 license in accordance with the GPLv2 license.

This document was approved by the IETF on 29th September, 2014.