Kerbal Operating System Instructions Documentation

- Version 1.0
- Unofficial, written as of December 2021, Kerbal Operating System release version 1.3.2.0

Contents

Preface

This document is intended to provide a comprehensive list of all current Kerbal Operating System opcodes/instructions and their functions. This guide builds off of many ideas that are introduced in the KSM File Docs. That is the format that these instructions are encoded in.

If any parts of this document are outdated or incorrect, please notify us by creating a GitHub issue.

All of the instructions listed in this document are sorted by the value of the instruction's opcode.

In order to understand the types of operands that each instruction takes, also see the KSM File Docs linked above.

About kOS instructions

As stated in the KSM docs, the kOS CPU is a stack-based computer emulated inside of Kerbal Space Program. Each instruction begins with an Opcode, which is basically just a number that tells kOS which instruction we want to run. Each instruction in this list will have the opcode. Opcodes are written as hexadecimal.

Because stacks work like they do, the first thing put on the stack is the last thing to come out. When stack arguments are listed on instructions, they are listed in the order they are popped off of the stack.

Terminology

Instruction - the basic unit of a kOS program
Opcode - the code which identifies instructions
Operand - parameters to the instruction that it goes with
Stack argument - an argument to an instruction stored on the stack

Instructions

End of File

Opcode	0x31
Operands	None
KASM Mnemonic	eof
Description	This instruction tells kOS that the current program's file has ended. kOS seems to expect this before an End of Program instruction. This instruction only causes the kOS CPU to stop executing and abort its context.

End of Program

Opcode	0x32
Operands	None
KASM Mnemonic	eop
Description	This instruction tells kOS that the current program has ended. Which aborts the current program. This is used to return back to the interpreter context once a program is finished executing. This can be explicitly written to end the program early.

No Operation

Opcode	0x33
Operands	None
KASM Mnemonic	nop
Description	Performs no operation: it does nothing. This can be used as a non-efficient way to try to delay the CPU, or just as a place to put a label in KASM, but it is largely unused.

Store

Opcode	0x34
Operand 1	String, StringValue
Stack Argument 1	The value to store
KASM Mnemonic	sto
Description	Consumes the topmost value of the stack, storing it into a variable named by the operand of this instruction. This will try to store the value in the lowest scope it can find the identifier in. If it isn't found in any outer scope, this creates a new **global** variable.

Unset

Opcode	0x35
Operands	None
Stack Argument 1	The identifier of the variable
KASM Mnemonic	uns
Description	Consumes the topmost value of the stack as an identifier, unsetting the variable referenced by this identifier. This will remove the variable referenced by this identifier in the innermost scope that it is set in.

Get Member

Opcode	0x36
Operand 1	String, StringValue
Stack Argument 1	The structure or identifier of the structure
KASM Mnemonic	gmb
Description	Consumes the topmost value of the stack, getting the suffix of it specified by the identifier operand and putting that value back on the stack. If this suffix refers to a method suffix, it will be called with no arguments.

Set Member

Opcode	0x37
Operand 1	String, StringValue
Stack Argument 1	The structure or identifier of the structure
KASM Mnemonic	smb
Description	Consumes a value and a destination object from the stack, setting the objects suffix specified by the identifier operand to the popped value.

Get Index

Opcode	0x38
Operands	None
Stack Argument 1	The index
Stack Argument 2	The collection
KASM Mnemonic	gidx
Description	Consumes an index and an target object from the stack, getting the indexed value from the object and pushing the result back on the stack.

Set Index

Opcode	0x39
Operands	None
Stack Argument 1	The value
Stack Argument 2	The index
Stack Argument 3	The object
KASM Mnemonic	sidx
Description	Consumes a value, an index, and an object from the stack, setting the specified index on the object to the given value.

Branch If False

Opcode	0x3a
Operand 1	(String, Int32)
Stack Argument 1	The boolean/value
KASM Mnemonic	bfa
Description	Consumes one value from the stack and branches to the given destination if the value was false. If the integer destination is provided, this represents a *relative* branch. If the value is 3, this will branch 3 instructions "down", and -3 is 3 instructions up. This uses 0 == false evaluation

Jump / Unconditional Branch

Opcode	0x3b
Operand 1	(String, Int32)
Stack Argument 1	The boolean/value
KASM Mnemonic	jmp
Description	Unconditionally branches to the given destination. If the integer destination is provided, this represents a *relative* branch. If the value is 3, this will branch 3 instructions "down", and -3 is 3 instructions up.

Add

Opcode	0x3c
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	add
Description	Consumes 2 values from the stack, pushing back the sum of the 2 values. Value1 + Value2

Subtract

Opcode	0x3d
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	sub
Description	Consumes 2 values from the stack, pushing back the difference of the 2 values. Value2 - Value1

Multiply

Opcode	0x3e
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	mul
Description	Consumes 2 values from the stack, pushing back the product of the 2 values. Value1 * Value2

Divide

Opcode	0x3f
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	div
Description	Consumes 2 values from the stack, pushing back their quotient. Value2 / Value1

Power

Opcode	0x40
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	pow
Description	Consumes 2 values from the stack, pushing back the result of raising the second value to the power of the first. Value2 ^ Value1

Compare Greater Than

Opcode	0x41
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	cgt
Description	Consumes 2 values from the stack, pushing back a boolean of if the second is greater than the first. Value2 > Value1

Compare Less Than

Opcode	0x42
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	clt
Description	Consumes 2 values from the stack, pushing back a boolean of if the second is less than the first. Value2 < Value1

Compare Greater Than or Equal

Opcode	0x43
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	cge
Description	Consumes 2 values from the stack, pushing back a boolean of if the second is greater than or equal to the first. Value2 >= Value1

Compare Less Than or Equal

Opcode	0x44
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	cle
Description	Consumes 2 values from the stack, pushing back a boolean of if the second is less than or equal to the first. Value2 <= Value1

Compare Equal

Opcode	0x45
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	ceq
Description	Consumes 2 values from the stack, pushing back a boolean of if the second is equal to the first. Value2 == Value1

Compare Not Equal

Opcode	0x46
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	cne
Description	Consumes 2 values from the stack, pushing back a boolean of if the second is not equal to the first. Value2 != Value1

Negate

Opcode	0x47
Operands	None
Stack Argument 1	The value
KASM Mnemonic	neg
Description	Consumes one value from the stack, pushing back the mathematical negation of the value (i.e. 99 becomes -99)

Convert to Boolean

Opcode	0x48
Operands	None
Stack Argument 1	The value
KASM Mnemonic	bool
Description	Consumes a value from the stack, coercing it to a boolean and then pushing it back. This uses the nonzero=true Boolean interpretation.

Logical Negate / Not

Opcode	0x49
Operands	None
Stack Argument 1	The value
KASM Mnemonic	not
Description	Consumes a value from the stack, pushing back the logical not of the value. If the value on the stack is not a BooleanValue, this will treat it as one using nonzero=true Boolean interpretation.

Logical And

Opcode	0x4a
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	and
Description	Consumes 2 values from the stack, pushing back a boolean of if both values were true. If one or more of the values on the stack are not BooleanValues, this will attempt to treat them as Booleans using the nonzero=true Boolean interpretation. This is not used by the KerboScript compiler, which instead uses short-circuit logic.

Logical Or

Opcode	0x4b
Operands	None
Stack Argument 1	Value1
Stack Argument 2	Value2
KASM Mnemonic	or
Description	Consumes 2 values from the stack, pushing back a boolean of if either of values were true. If one or more of the values on the stack are not BooleanValues, this will attempt to treat them as Booleans using the nonzero=true Boolean interpretation. This is not used by the KerboScript compiler, which instead uses short-circuit logic.

Call Function

Opcode	0x4c
Operand 1	(String, Null) - The destination label
Operand 2	(String, Null) - The destination of the call
KASM Mnemonic	call
Description	Calls a subroutine, leaving the result on the stack. What actually happens under the hood depends on what type of call is happening, but the end result is always the arguments being consumed and the result being put back. The main thing to know is that both operands to this instruction are mutually exclusive. The other operand can always be Null and nothing bad will happen. The destination label is used to call user-defined functions by the function's label. The destination (operand 2) can be any built-in functions like "print()" or "stage()". If it is a delegate, then the operand 2 should be the string "", and the delegate should be on the top of the stack. Either way, after the delegate or lack thereof, the arguments to the function should be there if any, followed unconditionally by an ArgMarker.

Return from a Function

Opcode	0x4d
Operand 1	(Int16)
KASM Mnemonic	ret
Description	Returns from a Call instruction, popping a number (operand 1) of scope depths off the stack as it does so. It evals the topmost thing on the stack. to remove any local variable references and replace them with their current values, and then performs the equivalent of a popscope, then jumps back to where the routine was called from. It also checks to ensure that the argument stack contains the arg bottom marker. If it does not, that proves the number of parameters consumed did not match the number of arguments passed and it throws an exception (to avoid stack misalignment that would happen if it tried to continue).

Push

Opcode	0x4e
Operand 1	Any
KASM Mnemonic	push
Description	Pushes a constant value onto the stack.

Pop

Opcode	0x4f
Operands	None
KASM Mnemonic	pop
Description	Pops a value off the stack, discarding it.

Duplicate

Opcode	0x50
Operands	None
KASM Mnemonic	dup
Description	Push the thing atop the stack onto the stack again so there are now two of it atop the stack.

Swap

Opcode	0x51
Operand 1	Any
KASM Mnemonic	swap
Description	Swaps the order of the top 2 values on the stack.

Evaluate

Opcode	0x52
Operands	None
KASM Mnemonic	eval
Description	Replaces the topmost thing on the stack with its evaluated, fully dereferenced version. For example, if the variable foo contains value 4, and the top of the stack is the identifier named "$foo", then this will replace the "$foo" with a 4.

Add Trigger

Opcode	0x53
Operand 1	(bool) - If unique. True if the trigger being added should be called with an argument that identifies this instance/entrypoint uniquely at runtime.
Operand 2	(Int32) - The interrupt priority level of the trigger
KASM Mnemonic	addt
Description	Pops a function pointer from the stack and adds a trigger that will be called each cycle. The argument (to the opcode, not on the stack) contains the Interrupt Priority level of the trigger. For one trigger to interrupt another, it needs a higher priority, else it waits until the first trigger is completed before it will fire.

Remove Trigger

Opcode	0x54
Operands	None
Stack Argument 1	Function pointer
KASM Mnemonic	rmvt
Description	Pops a function pointer from the stack and removes any triggers that call that function pointer.

Wait

Opcode	0x55
Operands	None
Stack Argument 1	Time to wait in seconds
KASM Mnemonic	wait
Description	Pops a duration in seconds from the stack and yields execution for that amount of game time.

Get Method

Opcode	0x57
Operand 1	String, StringValue
Stack Argument 1	The structure or identifier of the structure
KASM Mnemonic	gmet
Description	Get Method is *exactly* the same thing as Get Member, and is in fact a subclass of it. The only reason for the distinction is so that at runtime the instruction can tell whether the getting of the member was done with method call syntax with parentheses, like SHIP:NAME(), or non-method call syntax, like SHIP:NAME. It needs to know whether there is an upcoming Call instruction coming next or not, so it knows whether the delegate will get dealt with later or if it needs to perform it now.

Store Local

Opcode	0x58
Operand 1	String, StringValue
Stack Argument 1	The value to store
KASM Mnemonic	stol
Description	Consumes the topmost value of the stack, storing it into a variable named in the identifier operand of this instruction. The variable must not exist already in the local nesting level, and it will NOT attempt to look for it in the next scoping level up. Instead it will attempt to create the variable anew at the current local nesting scope.

Store Global

Opcode	0x59
Operand 1	String, StringValue
Stack Argument 1	The value to store
KASM Mnemonic	stog
Description	Consumes the topmost value of the stack, storing it into a variable named by the identifier operand of this instruction. The variable will always be stored at a global scope, overwriting whatever else was there if the variable already existed. It will ignore local scoping and never store the value in a local variable

Push Scope

Opcode	0x5a
Operand 1	(Int16) - the unique id of this scope frame
Operand 2	(Int16) - the unique id of the scope frame this scope is inside of
KASM Mnemonic	bscp
Description	Pushes a new variable namespace scope (for example, when a "{" is encountered in a block-scoping language like C++ or Java or C#.) From now on any local variables created will be made in this new namespace. Has no argument stack effect.

Pop Scope

Opcode	0x5b
Operand 1	(Int16) - the number of levels
KASM Mnemonic	escp
Description	Pops a variable namespace scope. From now on any local variables created within the previous scope are orphaned and gone forever ready to be garbage collected. It is possible to give it an argument of more than 1 to pop more than one nesting level of scope, to handle the case where you are breaking out of more than one nested level at once. (i.e. such as might happen with a break, return, or exit keyword).

Store Exists

Opcode	0x5c
Operand 1	String, StringValue
Stack Argument 1	The value to store
KASM Mnemonic	stoe
Description	Consumes the topmost value of the stack, storing it into a variable described by identifer operand of this instruction, which must already exist as a variable before this is executed. Unlike Store, Store Exist will NOT create the variable if it does not already exist. Instead it will cause an error. (It corresponds to KerboScript's @LAZYGLOBAL OFF directive)

Push Delegate

Opcode	0x5d
Operand 1	(Byte, Int16, Int32)
Operand 2	(Bool) - If it should be captured as a closure or not.
KASM Mnemonic	phdl
Description	Pushes a delegate object onto the stack, optionally capturing a closure. What it means to capture as a closure means that it will store a reference to how the variable scopes are when it was executed.

Branch True

Opcode	0x5e
Operand 1	(String, Int32)
Stack Argument 1	The boolean/value
KASM Mnemonic	btr
Description	Consumes one value from the stack and branches to the given destination if the value was true. If the integer destination is provided, this represents a *relative* branch. If the value is 3, this will branch 3 instructions "down", and -3 is 3 instructions up. This uses 0 == false evaluation

Variable Exists

Opcode	0x5f
Operands	None
Stack Argument 1	The identifier
KASM Mnemonic	exst
Description	Tests if the identifier atop the stack is an identifier that exists in the system and is accessible in scope at the moment. If the identifier doesn't exist, or if it does but it's out of scope right now, then it results in a FALSE, else it results in a TRUE. The result is pushed onto the stack for reading.

Argument Bottom

Opcode	0x60
Operands	None
Stack Argument 1	Any
KASM Mnemonic	argb
Description	Asserts that the next thing on the stack is the argument bottom marker. If it's not the argument bottom, it throws an error. This does NOT pop the value from the stack - it merely peeks at the stack top. The actual popping of the arg bottom value comes later when doing a return, or a program bottom exit.

Test Argument Bottom

Opcode	0x61
Operands	None
Stack Argument 1	Any
KASM Mnemonic	targ
Description	Tests whether or not the next thing on the stack is the argument bottom marker. It pushes a true on top if it is, or false if it is not. In either case it does NOT consume the arg bottom marker, but just peeks for it.

Test Trigger Cancelled

Opcode	0x62
Operands	None
KASM Mnemonic	tcan
Description	ests whether or not the current subroutine context on the stack that is being executed right now is one that has been flagged as cancelled by someone having called SubroutineContext.Cancel(). This pushes a True or a False on the stack to provide the answer. This should be the first thing done by triggers that wish to be cancel-able by other triggers. (For example if someone unlocks steering in one trigger, the steering function should not be run after that even if it had been queued up at the start of this physics tick) If you are a trigger that wishes to be cancel-able in this fashion, your trigger body should start by first calling this to see if you have been cancelled, and if it returns true, then you should return early without doing the rest of your body.

Push Relocate Later

Opcode	0xce
Operand 1	(String)
KASM Mnemonic	prl
Description	An "ugly placeholder" to handle the fact that sometimes the KerboScript compiler creates an Push instruction that uses relocatable DestinationLabels as a temporary place to store their arguments in the list until they get added to the program. For more information, see the kOS code on GitHub. Not really used much.

Push Delegate Relocate Later

Opcode	0xcd
Operand 1	(String)
Operand 2	(Bool) - Should capture as a closure
KASM Mnemonic	pdrl
Description	This serves the same purpose as Push Relocate Later instruction, except it's for use with UserDelegates instead of raw integer IP calls. What this means in simpler terms is that this instruction is used to push a function as a delegate onto the stack in the proper way that kOS will make it do what you think it will do. This will turn the function's label into a real location *after* it is loaded.

Label Reset

Opcode	0xf0
Operand 1	(String)
KASM Mnemonic	lbrt
Description	Most instructions' Label fields are just string-ified numbers for their index position. But sometimes, when they are the entry point for a function call (from a lock expression), the label is an identifier string. When this is the case, then the mere position of the opcode within the program is not enough to store the label. Therefore, for import/export to a KSM file, in this case the numeric label needs to be stored. It is done by creating a dummy instruction that is just a no-op instruction intended to be removed when the program is actually loaded into memory and run. It exists purely to store, as an argument, the label of the next opcode to follow it. See the KSM File Docs for more information.