VM-specific v1.3.0 opcodes simulation

NOTES:

- changed META - it can be used for MSIZE simulation
- setting ergs per pubdata is done by separate opcode now (not part of `near_call`)
- incrementing TX counter is done by separate opcode now (not part of `far_call`)

Our VM has some opcodes that are not expressible in Solidity, but we can simulate them on compiler level by abusing “CALL” instruction. We use 2nd parameter of “CALL” (address) as a marker, and remaining 6 parameters as input parameters (we use “address”-like field since it’s kind of shorter type, if assembly block cares about types in Solidity). Unfortunately “CALL” returns only 1 stack parameter, but it looks sufficient for our purposes.

Please note, that some of the methods don’t modify state, so STATICCALL instead of CALL should be used for them. The type of the needed method is indicated in the rightmost column.

Call types are not validated and do not affect the simulation behavior, unless specified otherwise, like in `raw_far_call` and `system_call` simulations, where the call type is passed through.

For some simulations below we assume that there exist a hidden global pseudo-variable called `ACTIVE_PTR` for manipulations, since one can not easily load pointer value into Solidity’s variable.

Simulated opcode	CALL param 0 (gas)	CALL param 1 (address)	CALL param 2 (value)	CALL param 3 (input offset)	CALL param 4 (input length)	CALL param 5 (output offset)	CALL param 6 (output length)	Return value	call type	LLVM implementation	Motivation
`to_l1(is_first, in0, in1`	if_first (bool)	`0xFFFF`	in0 (u256)	in1 (u256)	`0xFFFF` to prevent optimizing out by Yul	0	0	_	call	`@llvm.syncvm.tol1(i256 %in0, i256 %in1, i256 %is_first)`	Send messages to L1
`code_source`	0	`0xFFFE`	-	0	`0xFFFF` to prevent optimizing out by Yul	0	0	address	staticcall	`@llvm.syncvm.context(i256 %param) ; param == 2` (see SyncVM.h)	Largely to be able to catch “delegatecalls” in system contracts (by comparing this == code_source)
`precompile(in0, ergs_to_burn, out0)`	in0 (u256)	`0xFFFD`	-	ergs_to_burn (u32)	`0xFFFF` to prevent optimizing out by Yul	0	0	out0	staticcall	`@llvm.syncvm.precompile(i256 %in0, i256 %ergs)`	way to trigger call to precompile in VM
`meta`	0	`0xFFFC`	-	0	`0xFFFF` to prevent optimizing out by Yul	0	0	u256 tight packing of https://github.com/matter-labs/zkevm_opcode_defs/blob/b000abebc27f88919e0087b7604b8c71ba5b3daf/src/definitions/abi/meta.rs#L6 (packing is in the link)	staticcall	`@llvm.syncvm.context(i256 %param) ; param == 3` (see SyncVM.h)	way to trigger call to meta information about some small pieces of the state in VM
`mimic_call(to, abi_data, implicit r5 = who to mimic)`	who_to_call	`0xFFFB`	0	abi_data	who_to_mimic	0	0	It is a call, so it WILL mess up the registers and WILL use `r1-r4` for our standard ABI convention and `r5` for the extra `who_to_mimic` argument	any in the code; mimic call in the bytecode	Runtime `*{i256, i1} __mimiccall(i256, i256, i256, *{i256, i1})`	
`system_mimic_call(to, abi_data, implicit r3, r4, r5 = who to mimic)`	who_to_call	`0xFFFA`	0	abi_data	who_to_mimic	value_to_put_into_r3	value_to_put_into_r4	It is a call, so it WILL mess up the registers and WILL use `r1-r4` for our standard ABI convention and `r5` for the extra `who_to_mimic` argument	any in the code; mimic call in the bytecode	Runtime `*{i256, i1} __mimiccall(i256, i256, i256, *{i256, i1})`	
`mimic_call_byref(to, ACTIVE_PTR, implicit r5 = who to mimic)`	who_to_call	`0xFFF9`	0	0	who_to_mimic	0	0	It is a call, so it WILL mess up the registers and WILL use `r1-r4` for our standard ABI convention and `r5` for the extra `who_to_mimic` argument	any in the code; mimic call in the bytecode	Runtime `*{i256, i1} __mimiccall(*i8 addrspace(3), i256, i256, *{i256, i1})`	Same as one above, but takes ABI data from `ACTIVE_PTR`
`system_mimic_call_byref(to, ACTIVE_PTR, implicit r3, r4, r5 = who to mimic)`	who_to_call	`0xFFF8`	0	0	who_to_mimic	value_to_put_into_r3	value_to_put_into_r4	It is a call, so it WILL mess up the registers and WILL use `r1-r4` for our standard ABI convention and `r5` for the extra `who_to_mimic` argument	any in the code; mimic call in the bytecode	Runtime `*{i256, i1} __mimiccall(*i8 addrspace(3), i256, i256, *{i256, i1})`	Same as one above, but takes ABI data from `ACTIVE_PTR`
`raw_far_call`	who_to_call	`0xFFF7`	0	0	abi_data (CAN be with “to system = true”)	output_offset	output_length	Same as for EVM `call`	call | static | delegate (the call type is preserved)		It’s very similar to “system_call” described below, but for the cases when we only need to have `to_system = true` set in ABI

											(responsibility of the user, NOT the compiler), but we do not actually need to pass anything through `r3` and `r4` (so we can save on setting them or zeroing them, whatever)
`raw_far_call_byref`	who_to_call	`0xFFF6`	0	0	`0xFFFF` to prevent optimizing out by Yul	output_offset	output_length	Same as for EVM `call`	call | static | delegate (the call type is preserved)		Same as one above, but takes ABI data from `ACTIVE_PTR`
`system_call`	who_to_call	`0xFFF5`	value_to_put_into_r3 (only for `call` with 7 arguments)	value_to_put_into_r4	abi_data (MUST have "to system" set)	value_to_put_into_r5	value_to_put_into_r6	Same as for EVM `call`	call | static | delegate (the call type is preserved)	to call system contracts, like MSG_VALUE_SIMUALTOR. We may need 4 different formal definitions for cases when we would want to have integer/ptr in `r3` and `r4`	
`system_call_byref`	who_to_call	`0xFFF4`	value_to_put_into_r3 (only for `call` with 7 arguments)	value_to_put_into_r4	`0xFFFF` to prevent optimizing out by Yul	value_to_put_into_r5	value_to_put_into_r6	Same as for EVM `call`	call | static | delegate (the call type is preserved)	to call system contracts, like MSG_VALUE_SIMUALTOR. We may need 4 different formal definitions for cases when we would want to have integer/ptr in `r3` and `r4`	Same as one above, but takes ABI data from `ACTIVE_PTR`
`set_context_u128`	0	`0xFFF3`	value	0	`0xFFFF` to prevent optimizing out by Yul	0	0	-	call		
`set_pubdata_price`	in0	`0xFFF2`	0	0	`0xFFFF` to prevent optimizing out by Yul	0	0	-	call	`context.set_ergs_per_pubdata in0` in assembly	
`increment_tx_counter`	0	`0xFFF1`	0	0	`0xFFFF` to prevent optimizing out by Yul	0	0	-	call	`context.inc_tx_num` in assembly	
`ptr_calldata`	0	`0xFFF0`	-	0	`0xFFFF` to prevent optimizing out by Yul	0	0		staticcall	one passed in `r1` on `far_call` to the callee (save in very first instructions on entry)	Loads as INTEGER!
`call_flags`	0	`0xFFEF`	-	0	`0xFFFF` to prevent optimizing out by Yul	0	0		staticcall	one passed in `r2` on `far_call` to the callee (save in very first instructions on entry)	
`ptr_return_data`	0	`0xFFEE`	-	0	`0xFFFF` to prevent optimizing out by Yul	0	0		staticcall	one passed in `r1` on return from `far_call` back to the caller (save in very first instruction in the corresponding branch!)	Loads as INTEGER!
`event_initialize`	in1	`0xFFED`	-	in2	`0xFFFF` to prevent optimizing out by Yul	0	0		call		
`event_write`	in1	`0xFFEC`	-	in2	`0xFFFF` to prevent optimizing out by Yul	0	0		call		
`load_calldata_into_active_ptr`	0	`0xFFEB`	-	0	`0xFFFF` to prevent optimizing out by Yul	0	0		staticcall	loads value of `@calldataptr` (from `r1` at the entry point of the contract into virtual `ACTIVE_PTR`) `ACTIVE_PTR`	
`load_returndata_into_active_ptr`	0	`0xFFEA`	-	0	`0xFFFF` to prevent optimizing out by Yul	0	0		staticcall	loads value of the latest `@returndataptr` (from the `r1` at the point of return from the child into virtual `ACTIVE_PTR`)	
`ptr_add_into_active`	in1	`0xFFE9`	-	0	`0xFFFF` to prevent	0	0		staticcall	performs `ptr.add ACTIVE_PTR, in1, ACTIVE_PTR`	

					optimizing out by Yul						
`ptr_shrink_into_active`	in1	`0xFFE8`	-	0	`0xFFFF` to prevent optimizing out by Yul	0	0		staticcall	performs `ptr.shrink ACTIVE_PTR, in1, ACTIVE_PTR`	
`ptr_pack_into_active`	in1	`0xFFE7`	-	0	`0xFFFF` to prevent optimizing out by Yul	0	0		staticcall	performs `ptr.pack ACTIVE_PTR, in1, ACTIVE_PTR`	
`multiplication_high`	in1	`0xFFE6`	-	in2	`0xFFFF` to prevent optimizing out by Yul	0	0	Returns the higher register (the overflown part)	staticcall		
`extra_abi_data`	0	`0xFFE5`	-	0	`0xFFFF` to prevent optimizing out by Yul	0	0		staticcall	ones passed in `r3-r12` on `far_call` to the callee (saved in the very first instructions in the entry)	

Requirements for calling system contracts

By default, all system contracts up to the address `0xFFFF` require that the call was done via system call (i.e. `call_flags&2 != 0` .

Exceptions:

- BOOTLOADER_FORMAL address as the users need to be able to send money there.

Meaning of ABI params:

- MSG_VALUE_SIMULATOR: `extra_abi_data_1 = value || whether_the_call_is_system`, where || denotes the concatenation, value should occupy first 128 bits, while `whether_the_call_is_system` is a 1-bit flag that denotes whether the call should be a system call. `extra_abi_data_2` is the address of the callee.
- No meaning for the rest