Instruction Set

An instruction set is made up of a instruction list of 8 byte instructions and a constant region which contains all of the constant values used within the instruction list.

Instruction List

LoadConstant(ConstantIndex : Constant Location (4 bytes), Destination : Register (1 byte)) – Load a constant from the constants list and place it in the specified register.

Move (from : Register(1 byte) , to Register (1 byte)) – Copy whatever is at the from register to the to register.

JumpDirect(Where : 4 bytes) – Jump to the specified instruction.

JumpIndrect(Register : 1 byte) – Jump to the location specified by the register.

Add(left : Register (1 byte), right : Register (1 byte) , dest : Register (1 byte)) – Place the addition of the left and right registers in the destination register

Subtract(left : Register (1 byte), right : Register (1 byte), dest : Register (1 byte)) – Place the subtraction of the right register from the left register in the destination register

Multiply(left : Register (1 byte), right : Register (1 byte), dest : Register (1 byte)) – Multiply the left register value by the right register value and place it in the dest register

Divide(left : Register (1 byte), right : Register (1 byte), dest : Register (1 byte)) – Divide the left register by the right register and place it in the destination register

Equals(left : Register (1 byte), right : Register (1 byte)) – Test whether the left register is equal to the right register. If it is then execute the next instruction otherwise skip an instruction.

EqualsZero(register : Register (1 byte)) – Test whether the specified register equals zero. If it is then execute the next instruction else skip an instruction.

LessThan(left : Register (1 byte), right : Register (1 byte)) – Test whether the left register is less than the right register. If it is then execute the next instruction otherwise skip an instruction.

LessThanOrEqual(left : Register (1 byte), right : Register (1 byte)) – Test whether the left register is less than or equal to the right register. If it is then execute the next instruction otherwise skip an instruction.

NewArray(Length : Register (1 byte), Destination : Register (1 byte), TypeConstant : Constant location (4 bytes)) – Create a new array of the specified length and type and place a reference to it in the Destination register.

ArraySet(ArrayReg : Register (1 byte) , IndexReg : Register (1 byte), ValueReg : Register (1 byte)) – Set the value of the array at the specified index to be the value of the specified value register.

ArrayGet(ArrayReg : Register (1 byte), IndexReg : Register (1 byte), DestReg : Register (1 byte)) - Set value of the DestRegister to be the value of the array at the specified index

ArrayLength(ArrayReg : Register (1 byte), Dest : Register (1 byte)) – Place the length of the array at ArrayReg into Dest

PushRegisters(Start : Register (1 byte) , N : 1 Byte) – Push N registers to the stack starting from the start register.

PopRegisters(Start : Register (1 byte), N : 1 Byte) – Pop N registers starting from the Start + Nth register and ending with the Start register (The reverse order is so that push $0 10 pop $0 10 are complimentary)

PopNil() – Pop a register from the stack and discard it.

CallFunctionConstant(Constant : Int (4 bytes)) – Call a function given a function name in the constant area

CallFunction(Fn : Register (1 byte)) - Call a function with the name of the string that Fn is a reference to

Return - Returns to the previous function and sets the program counter to the instruction after the function call. If there is no function to return to then the VM->execute function will return