The Virtual Machine and the Assembler

Version from 2023-03-23 11:48:20

Morten Rhiger

March 23, 2023

The virtual machine (VM) consists of a stack and a memory, both of limited size and both containing integers. The virtual machine reads and executes a sequence of instructions. The table of page 4 shows the instructions of the virtual machine (in the left column) and their effect on the stack as implemented by the virtual machine. In the table, the stack is written as an F# list, with its top element to the left (and st stands for a stack, while v, x, y, i, l, and p stands for stack elements).

In all cases except for `IHALT` and those marked by "Executes block labeled l", the virtual machine jumps to the next instruction in the sequence after having executed the current. Only `IALLOC`, `IGET`, and `ISET` access the memory. In the description of `IGET` and `ISET`, $M[p]$ is the memory cell at address p.

1 The VM module

The F# file `vm.dll` is a compiled library that implements the virtual machine. It contains a module called `VM` with one function

```
exec : int array → int
```

that executes an array of instructions on an empty stack. Both instructions and their arguments are encoded by integers. For example 354, 289, and 257 represent the instructions `IREAD`, `IADD`, and `IHALT` so the machine code program `[|354; 354; 289; 257|]` lets the user type in two integers and returns their sum:

```
> #r "vm.dll";;
> VM.exec [|354; 354; 289; 257|];;
Enter an integer: 3
Enter an integer: 4
```

```
Executed 4 instructions in 2026 ms.
val it : int = 7
```

We do not have to learn the encoding of instructions as integers; instead we let an *assember* generate these integers, from a more readable representation of instruction.

The virtual machine can be controlled by two flags: The command

`> VM.time <-` *flag*`;;`

turns on or off the timing message printed after a successful execution of a program. (The value *flag* is either `true` or `false`.) The command

`> VM.trace <-` *flag*`;;`

controls whether or not to display tracing information. Of active, then the entire list of instructions is displayed before execution the program and the current instructions being executed is displayed while execution the program. (Again, *flag* is a boolean value.)

2 The Asm module

The F# file `asm.dll` is a compiled library that implements an assember. It contains a module called `Asm` defining a datatype of symbolic instructions

```
type label = string
type inst =
  | IHALT
  | IPUSH   of int     | IPOP               | ISWAP
  | ILOAD   of int     | ISTORE  of int
  | IADD               | ISUB
  | IMUL               | IDIV               | IMOD
  | IEQ                | ILT
  | IJMP    of label
  | IJMPIF  of label
  | ICALL   of label   | IRETN
  | IALLOC             | IGET               | ISET
  | IWRITE             | IREAD
  | ILAB    of label
```

and one function

```
asm  : inst list → int array
```

This function serves two purposes: it translates the symbolic representation of instructions and their arguments into plain integers, and it replaces symbolic labels by addresses. For example,

```
> #r "asm.dll";;
> open Asm;;
> asm [ILAB "again"; IREAD; IWRITE; IJMP "again"];;
val it : int [] = [|354; 353; 321; 0|]
> VM.exec it;;
Enter an integer: 1
1
Enter an integer: 2
2
Enter an integer: 3
3
```

Instruction	Effect on stack			Remark
`IHALT`	$v :: st$			Program halts with value v
`IPUSH` i	st	$\longrightarrow$	$i :: st$	
`IPOP`	$v :: st$	$\longrightarrow$	st	
`ISWAP`	$y :: x :: st$	$\longrightarrow$	$x :: y :: st$	
`ILOAD` i	$v_0 :: \ldots :: v_i :: st$	$\longrightarrow$	$v_i :: v_0 :: \ldots :: v_i :: st$	
`ISTORE` i	$x :: v_0 :: \ldots :: v_i :: st$	$\longrightarrow$	$v_0 :: \ldots :: x :: st$	
`IADD`	$y :: x :: st$	$\longrightarrow$	$y + x :: st$	
`ISUB`	$y :: x :: st$	$\longrightarrow$	$y - x :: st$	
`IMUL`	$y :: x :: st$	$\longrightarrow$	$y \times x :: st$	
`IDIV`	$y :: x :: st$	$\longrightarrow$	$y/x :: st$	Integer division
`IMOD`	$y :: x :: st$	$\longrightarrow$	$y\%x :: st$	Integer remainder
`IEQ`	$y :: x :: st$	$\longrightarrow$	$1 :: st$	If $x = y$
`IEQ`	$y :: x :: st$	$\longrightarrow$	$0 :: st$	If $x \neq y$
`ILT`	$y :: x :: st$	$\longrightarrow$	$1 :: st$	If $x < y$
`ILT`	$y :: x :: st$	$\longrightarrow$	$0 :: st$	If $x \geq y$
`IJMP` l	st	$\longrightarrow$	st	Executes block labeled l
`IJMPIF` l	$1 :: st$	$\longrightarrow$	st	Executes block labeled l
`IJMPIF` l	$0 :: st$	$\longrightarrow$	st	
`ICALL` l	st	$\longrightarrow$	$l_r :: st$	Executes block labeled l; l_r is the address of the following instruction
`IRETN`	$l :: st$	$\longrightarrow$	st	Executes block labeled l
`IALLOC`	$n :: st$	$\longrightarrow$	$p :: st$	p points to n new memory cells
`IGET`	$p :: st$	$\longrightarrow$	$M[p] :: st$	
`ISET`	$v :: p :: st$	$\longrightarrow$	st	Sets $M[p] = v$
`IWRITE`	$v :: st$	$\longrightarrow$	st	Value v written to the screen
`IREAD`	st	$\longrightarrow$	$v :: st$	Value v typed by the user
`ILAB` l	st	$\longrightarrow$	st	Pseudo-instruction with no effect