Virtual Machine I - Processor

Sunday, March 6, 2022 4:35 PM

"Programmers are creators of the universes for which they alone are responsible. Universes of virtually unlimited complexity can be created in the form of computer programs"

- Joseph Weizenbaum, *Computer Power and Human Reason (1974)*

7.1 The Virtual Machine Paradigm

One way Computer Scientists have learned to decouple the process of compiling high level program is to break it up into two steps.

1. Parse the high-level code and translate into intermediate and abstract processing steps
2. The intermediate steps are then translated further into machine code.

Virtual Machine - a machine whose commands realize the intermediate processing steps into which high-level commands are translated.

Compiler - translates high-level code into intermediate VM commands

VM Translator - Translates the VM commands into further machine code of the target hardware.

Java Program -> Java Compiler -> VM Code -> Java VM (JVM)

The VM code can be shipped out to multiple different devices, from which JVM will turn it into runnable machine code for that device.

7.2 Stack Machine

VM languages seek a balance within the high-level and low-level design. You want

a lot of expressive power such as, arithmetic operations, push/pop, branching, and function commands. On the other hand you want your machine code to also trickle down to its most efficient form.

It doesn't matter if you can do a lot on the high-level side of things if the translation process produces insufficient code. Conversely, what's the point in making a high-level language if it runs as efficient in machine code, but does not offer any expressive power.

One way to solve this issue is to base the interim VM language on an abstract architecture called a stack machine.

The overall purpose of the Stack Machine will be showcased throughout the rest of the book. Don't get caught up within this chapter of the "why?" as that will be expressed later on. Only focus on the "how?" so the implementation is smooth.

7.2.1 Push and Pop

The centerpiece of the Stack Machine is an abstract data structure called the stack.

Stack - an abstract sequential (LIFO) data structure that can grow or shrink if need be. The stack has two main functions push and pop. Pushing will place a value on the top of the stack. While a pop will take the top value off the stack and return it.

7.2.2 Stack Arithmetic

Consider $x\ op\ y$ Where the operation op is applied to the operands x and y.

	Stack
	12
	3
	7
$sp \rightarrow$	

Add:

	Stack
	12

	10
$sp \rightarrow$	

Negate:

	Stack
	12
	-10
$sp \rightarrow$	

There are examples of stack arithmetic on page 131

7.2.3 Virtual Machine Segments

Segment	Role
argument	Represents the function's arguments
local	Represents the functions local variables
static	Represents the static variables seen by the function
constant	Represents the constant values 0,1,2,3,…,32767
this	Described in later chapters
that	Described in later chapters
pointer	Described in later chapters
temp	Described in later chapters

7.3 VM Specification, Part 1

A VM program is a sequence of VM commands that fall into the four categories:

- Push / Pop commands - transfer data between the stack and memory segments
- Arithmetic-logic commands - perform arithmetic and logic operations
- Branching commands - facilitate conditional and unconditional branching operations
- Function commands - facilitate function call-and-return operations.

Lines beginning with // are considered comments Blank spaces are permitted and ignored as well

Push / Pop Commands

Push Segment Index - Pushes the value of *segment[index]* onto the stack, where *segment* is argument, local, static, constant, this, that, pointer, or temp and *index* is a nonnegative integer.

Pop Segment Index - Pops the top stack value and stores it in *segment[index]* onto the stack, where *segment* is argument, local, static, this, that, pointer, or temp and *index* is a nonnegative integer.

Arithmetic-Logic Commands

Arithmetic Commands: add, sub, neg

Comparison Commands: eq, gt, lt

Logical commands: and, or, not

Command	Computes	Comment
add	$x + y$	Integer addition (two's compliment)
sub	$x - y$	Integer subtraction (two's compliment)
neg	$-y$	Arithmetic negation (two's compliment)
eq	$x == y$	equality
gt	$x > y$	Greater than
lt	$x < y$	Less than
and	$x\ And\ y$	Bit-wise And
or	$x\ Or\ y$	Bit-wise Or
not	$Not\ y$	Bit-wise Not