CSCI 361 Virtual Machine Notes

Kyle Krstuich

March 27, 2025

History

The model of computation in the 1990's

```
High level language
      | Compiler
Machine Language
      | Assembler
Op codes/Binary
      | control logic
Hardware
```

Later on the model was changed.

```
High level language
      | Virtual machine translator
Runtime Virtual Machine
      | Stack based computing
Machine Language # can also have a language compile directly to machine code
      | Assembler
Op codes/Binary
      | control logic
Hardware
```

Run time virtual machine is a lower lever application that has its own machine code that you compile your higher level language to. This makes it so that you don't Hae to write you code specific to an architecture. This however, is slow.
This week, we assume that we have code for the virtual machine. Me make a Python program translator for machine language using stack based computing.

Building a VM

VM supports elementary stack operations. LIFO data structure.

- Pop() off
- Push() onto

A stack is usually a segment of RAM with a stack pointer(SP) just above the stack. This is useful because you can push in constant time.

```
push(x):
  stack[SP] = x
  SP++

pop():
  SP--
  return stack[SP]
```

Stacks are lossy or lose data at the time of removal. While RAM is lossy at time of input, because RAM overwrites values, while the stack 'forgets' values.
In other classes I played with the top of the stack. However the stacks in this class work from the bottom. This is because when the stack grows it does down in memory where there is more room.

Stack Arithmetic

Pop operands to preform operations. Push result back.
Operands needed to implement:

- ADD; pops top two values off stack, add them, and push the result. Takes the deeper value first.

  ```
  a = pop()
  b = pop()
  push(b+a)
  ```

- SUB; pops top two values off stack, subtracts them, and then push the result. Takes the deeper value first.

  ```
  a = pop()
  b = pop()
  push(b-a)
  ```

- NEG *
- EQ
- GT
- LT
- AND
- OR
- NOT 8

* are urinary operators.

Class example

```
operation D = (2-x) + (y+5)
x=5, y=3

1. push(2) ->             |2|
                          |SP|

2. push(@x) ->            |2|
                          |x|
```

```
                          |SP|

3. SUB  # 2-x ->          |-3|
                          |SP|

4. push(@y) ->            |-3|
                          |y|
                          |SP|

5. push(5) ->             |-3|
                          |y|
                          |5|
                          |SP|

6. ADD # y+5 ->           |-3|
                          |8|
                          |SP|

7. ADD # (2-x) + (y+5) -> |5|
                           |SP|
```

Binary operations require 2 arguments such ass ADD. Unary operators only require 1 argument such as NOT. The structure for the operations is very different. Pops are not a direct memory access, it is an indirect address.

More History

In the 90's and early 2000's compilers did not have a virtual machines, and were tightly coupled to the hardware architecture. There were many different compilers, free and paid.

Modern compilers compile the high level code to bytecode, which is read by the virtual machine, to then translate that bytecode into assembly, then assembled into binary. This is also called a two-stage compilation process.

For Tuesday build push/pop to the D register.

Another Class Example

```
(x<7) or (y=8)
x=12, y=8

1. push(x) ->              |x|
                               |SP|

2. push(7) ->                  |x|
                               |7|
                               |SP|

3. LT # x < 7 ->               |FALSE|
                               |SP|

4. push(y) ->                  |FALSE|
                               |y|
                               |SP|

5. push(8) ->                  |FALSE|
                               |y|
                               |8|
                               |SP|

6. EQ # does y = 8? ->         |FALSE|
                               |TRUE|
                               |SP|

7. OR # (x>7) or (y=8) ->      |TRUE|
                               |SP|
```

The VM File

.vm files is what you read in. The output will be hack assembly. VM files are on the website. There are three formats for commands in the VM file.

1. Command
2. Command argument
3. Command argument arguments

Memory Access Commands

We need to say

- Push segment index
- Pop segment index

Where a segment is:

- Argument *: used to store activation record

- local *: also used to store activation record
- static -
- constant +: for direct push of literal, is a 'virtual' segment.
- this/that *
- pointer *
- temp +

* for each function
- for all functions in VM file
+ for *all* functions