Brian Cullinan

Project 9: Virtual Machine

11/19/2008

**Objectives:**

Create an interpreter for your language, and some example assembly-like code that will assemble and run on your Virtual Machine and can be presented in class.

**Overview:**

I eliminated the need for some of the commands that I had listed before. I first wrote a test program that I would like to run. I then went through and made it possible using the commands I needed. Then I went back and added the other useful commands.

**Approach:**

I first implemented the functions used to store and save in global memory. I had to change around the specification little to get what worked best. The command for saving in global memory accepts and input for an index of global memory, it saves the top value on the stack. The global load command pushes the provided index value on to the stack. I added a print command for basic output; it prints the top value on the stack.

Math operations now use the stack as well to do the job instead of using 2 input values. I removed some unnecessary commands just by using the stack more. The frame pointer points to the bottom of the frame because the stack pointer will be pointing to the top. That way programs can go through and check all the variables until it hits the frame pointer. The instruction pointer was simply to program because I can use the Key command to get the index of the current key. The stack pointer is always at the top of the stack, so it points to the bottom of the array that stores it. This can also be used to count how many items are stored in the stack.

The code can be moved through fairly easily. Most of the literals resolve to 4 byte words, so it makes it a little more complicated to read all the bytes and use them effectively. So certain commands like call and jump have to extract the address to jump to, and then set the position of the instruction pointer. It was difficult to think about since PHP has built in array\_push and array\_pop commands. But unlike a traditional stack PHP uses the bottom of the stack in order to save processing power. There is also a printc command that converts the number on the stack to a character and then prints it out.

**Conclusion:**

This project turned out to be a lot of fun and I can’t wait to make minor adjustments when programming the next project.

Sample Code:

push 1

push 1

gsave 0

gsave 1

loop1:

call jle

jumpif 3 endloop1

gload 0

print

push 32

printc

gload 1

print

push 32

printc

gload 0

gload 1

add

gsave 0

gload 1

gload 0

add

gsave 1

jump loop1

endloop1:

return

jle:

return

Sample Output:

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155 165580141 267914296 433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049 12586269025 20365011074 32951280099 53316291173 86267571272 139583862445 225851433717 365435296162 591286729879 956722026041 1548008755920 2504730781961 4052739537881 6557470319842 10610209857723 17167680177565 27777890035288 44945570212853 72723460248141 117669030460994 190392490709135 308061521170129 498454011879264 806515533049393 1304969544928657 2111485077978050

Interpreter.php:

<?php

$vm = new VM('assembly.vm');

class VM

{

var $stack = array();

var $global = array();

var $code = '';

var $ip = 0;

var $sp = 0;

var $fp = 0;

function VM($file)

{

// read in file and seperate commands into array

$fp = fopen($file, 'r');

$file = fread($fp, filesize($file));

fclose($fp);

$this->code = preg\_split('//', $file, -1);

for($i = 0; $i < count($this->code); $i++)

{

$this->code[$i] = dechex(ord($this->code[$i]));

}

// read the informational part of the file

// make sure it is a small vm file

//var\_dump($this->code);

if(next($this->code) == '53' && next($this->code) == '4d' && next($this->code) == '56' && next($this->code) == '4d')

{

$version = next($this->code) . next($this->code) . next($this->code) . next($this->code);

$var\_count = next($this->code) . next($this->code) . next($this->code) . next($this->code);

$code\_length = next($this->code) . next($this->code) . next($this->code) . next($this->code);

while(($op\_code = next($this->code)) !== false)

{

$this->ip = key($this->stack);

switch(hexdec($op\_code))

{

case 0x5A: // push

next($this->code);

array\_push($this->stack, hexdec(next($this->code) . next($this->code) . next($this->code) . next($this->code)));

break;

case 0x4B: // gsave index

next($this->code);

$index = hexdec(next($this->code) . next($this->code) . next($this->code) . next($this->code));

$this->global[$index] = array\_pop($this->stack);

break;

case 0x3C: // call

next($this->code);

$index = hexdec(next($this->code) . next($this->code) . next($this->code) . next($this->code));

array\_push($this->stack, key($this->code));

$this->fp = count($this->stack);

reset($this->code);

for($i = 0; $i < $index-1; $i++) next($this->code);

break;

case 0x3D: // return

$index = array\_pop($this->stack);

reset($this->code);

for($i = 0; $i < $index; $i++) next($this->code);

break;

case 0x3B: // jumpif

next($this->code);

$condition = hexdec(next($this->code) . next($this->code) . next($this->code) . next($this->code));

next($this->code);

$address = hexdec(next($this->code) . next($this->code) . next($this->code) . next($this->code));

break;

case 0x4A: // gload

next($this->code);

$index = hexdec(next($this->code) . next($this->code) . next($this->code) . next($this->code));

array\_push($this->stack, $this->global[$index]);

break;

case 0x01: // print

$value = array\_pop($this->stack);

print $value;

break;

case 0x02: // printc

$value = array\_pop($this->stack);

print chr($value);

break;

case 0x7A: // add

array\_push($this->stack, array\_pop($this->stack) + array\_pop($this->stack));

break;

case 0x7B: // sub

array\_push($this->stack, array\_pop($this->stack) - array\_pop($this->stack));

break;

case 0x7C: // mult

array\_push($this->stack, array\_pop($this->stack) \* array\_pop($this->stack));

break;

case 0x7D: // div

array\_push($this->stack, array\_pop($this->stack) / array\_pop($this->stack));

break;

case 0x7E: // pow

array\_push($this->stack, pow(array\_pop($this->stack), array\_pop($this->stack)));

break;

case 0x3A: // jump

next($this->code);

$index = hexdec(next($this->code) . next($this->code) . next($this->code) . next($this->code));

reset($this->code);

for($i = 0; $i < $index; $i++) next($this->code);

break;

case 0x00: // noop

break;

default:

}

$this->sp = count($this->stack);

}

}

else

{

return;

}

}

}

?>