Project 2

Assembly Virtual Machine Code Operation and Syntax

Contents:

* General Use
* Declaring global variables
  + The “.data” section
* Writing your program
  + The “.text” section
* The “mov” instruction
* The “add” instruction
* The “sub” instruction
* The “print” command
* The “call” command
* The “peek” command

General Use

Upon booting the program, the program will prompt you to enter a file name for a prewritten program. NOTE: be sure to include the “.txt” extension of your program file. The VM will load the file and will execute the commands written in the file.

A few things to keep in mind:

* There are not any memory or instruction limitations
* Be sure to use proper syntax (defined in following sections), otherwise the VM will not work properly
* You do not have to use global variables if you do not want to
  + However, if you wish to use them, define them BEFORE you write your program

Declaring Global Variables Using .data

At the start of your program, you can create global variables to hold constants or any other predefined value (apart from decimal or fractional numbers). To initialize a global variable, the syntax is quite simple:

“variable name” + “ ” + “contents”

The variable name can be anything you would like. After you have named your variable, put a space after it, and finally write what you want the variable to store. Remember, the VM does not recognize decimal numbers or fractions when executing mathematical operations, so avoid them if possible.

Writing Your Program Using .text

Once you have declared your global variables for your program, you can begin writing instructions for the VM. It closely resembles x86 Assembly syntax, so it is easy to understand. The instruction list is as follows:

* Mov
* Add
* Sub
* Print
* Peek
* Call

All instruction specific syntax will be on the following pages. However, most have a 3-part command:

“instruction” + “ ” + “destination” + “ ” + “source”

This rule applies to all instructions listed above except for print, peek, and call. These are more specialized than the others, so they work differently.

The MOV Instruction

The mov instruction allows you to move data into a specific register from another register, a global variable, or a literal entered in the line.

Examples:

* mov r*n* r*n*
  + n is anything from 1 – 10
  + Any register is valid, but do not use the same register twice
* mov r*n var*
  + n is anything from 1- 10
  + var is a global variable declared in the .data section
* mov r*n x*
  + n is anything from 1- 10
  + x is any literal (can contain special characters)

Once executed, the destination register will be updated.

The ADD Instruction

The add instruction performs the addition operation on a register using the contents of another register, a global variable, or a number entered in the line. However, you cannot use decimal or fractional numbers.

Examples:

* add r*n* r*n*
  + n is anything from 1 – 10
  + Any register is valid, but do not use the same register twice
* add r*n var*
  + n is anything from 1- 10
  + var is a global variable declared in the .data section
* add r*n x*
  + n is anything from 1- 10
  + x is any integer number (can be positive or negative)

The SUB Instruction

The sub instruction performs the addition operation on a register using the contents of another register, a global variable, or a number entered in the line. However, you cannot use decimal or fractional numbers.

Examples:

* sub r*n* r*n*
  + n is anything from 1 – 10
  + Any register is valid, but do not use the same register twice
* sub r*n var*
  + n is anything from 1- 10
  + var is a global variable declared in the .data section
* sub r*n x*
  + n is anything from 1- 10
  + x is any integer number (can be positive or negative)

The PRINT Command

This command will display the contents of the specified general register. The VM will notify you if you are attempting to print out an empty register.

Example:

* print r*n*
  + n is anything from 1- 10
  + If it is populated, the stored value will be printed
  + If not, the VM will say the register’s memory is unused currently

The CALL Command

The VM contains 3 pre-defined functions: function1, function2, and function3. You can execute any of these functions by using “call”. Function1 prints a simple message, function 2 uses the stack functionality, and function3 prints another message.

Example:

* call function***n*** 
  + where n is between 1 – 3

The PEEK Command

The RIP register tracks the current line being executed by the VM. If you want, you can use the “peek” command to see where the VM is currently looking. The syntax is just simply typing “peek”, because it isn’t looking at a specific register.