STRING LIBRARY IMPLEMENTATION IN JACK

Project Report

As a part of the subject

ELEMENTS OF COMPUTING-II

DECLARATION

We hereby declare that the project report is completely done by our group with the cooperation of all the members of the group. Discussion process involved everyone in the group and implementation and execution was equally split by ourselves.

Place: Ettimadai
Date: 16-07-2022

Contents

Acknowledgement:

Abstract:

Here we are implementing the string library function in jack. Jack is a simple, object-based with java like syntax. And it used to write high-level programming language. In this project our aim is to see how high-level language converts into lower-level language. Compilation of jack files gives .VM files. Later, by using VM Translator that we had built in python programming language we get .asm file. Here we are developing our own assembler in python programming language to convert .asm files to .hack file. And finally hack file is dumped into the Hardware Simulator in ROM part and we get the output in the RAM.

AIM:

The aim of our project is to Implement the following String library functions in Jack.

1. Constructs a new empty string with a maximum length

2. Returns the character value at the specified index.

3. Returns the length of this string

4. De-allocates the string and frees its memory space.

1. <u>Introduction</u>

String library functions are provided, and we are asked to write a Jack program code to given functions and compile it and output is returned as **.vm** file. Later we simulate it on VM Translator that we had built. Jack is a simple object-based language that can be used to write high-level programs. And the Jack Compiler translates files written in the Jack programming language into Virtual Machine code. For each source. **jack** file, the compiler creates a corresponding .**vm** file, and stores the translated code in it.

Later VM file is to be converted into assembly language file, simply called **.asm** file. Here we are using our own assembler developed using python programming language. Finally, that assembly code file is to be converted to 16- bit machine code called .**hack** file. Then we compare the .**hack** files that are produced by our own assembler and built-in assembler.

Atter that we use hardware simulator tool to run the .**hack** file in ROM and the output is formed in RAM in the binary form.

This files, such as VM, ASM, and hack files, should be checked in the respective platforms, such as VM emulator, CPU emulator, hardware simulator, and assembler. These softwares are provided by the nand2tetris.

The Big Picture Overview:
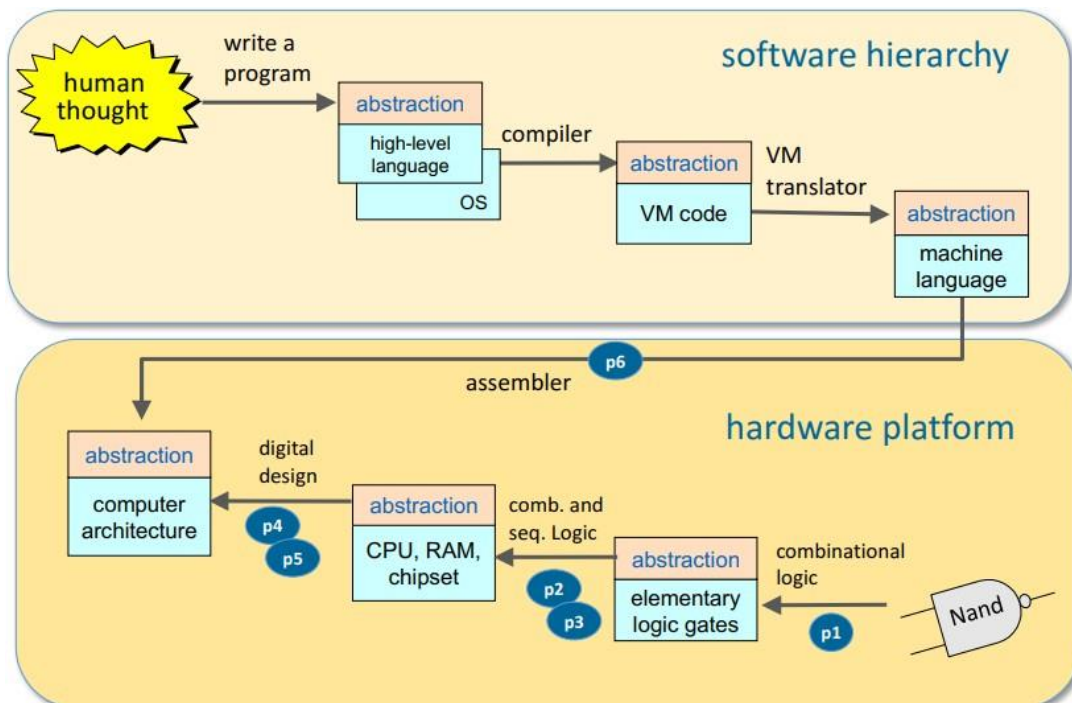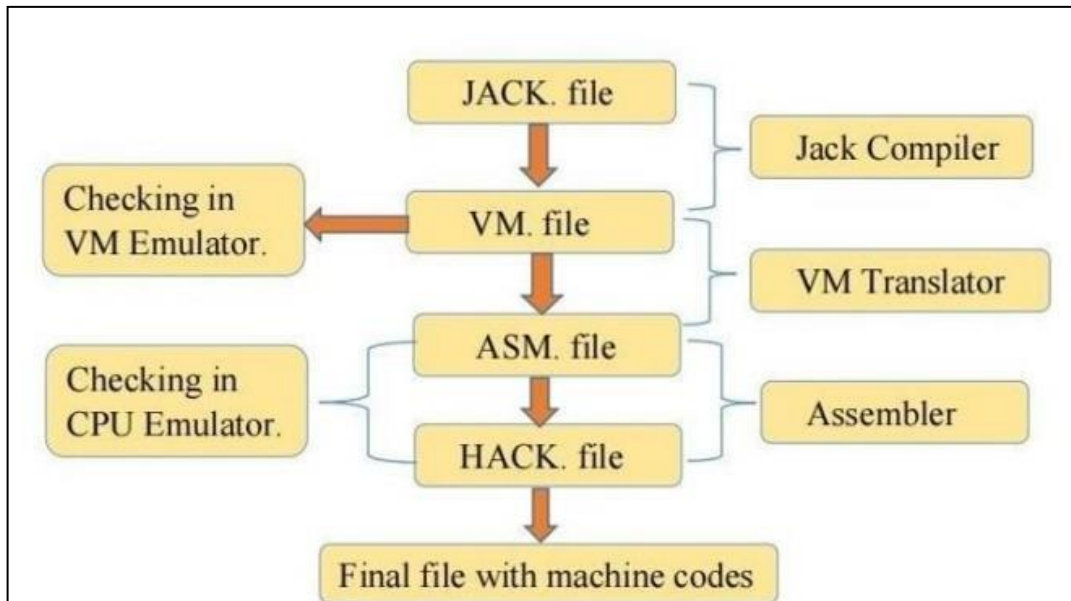
.JACK file: File which has code in JACK programming language format in it.

.VM file: File we get after compiling the above .JACK file with the help of inbuilt JACK compiler. This file is tested in the VM Emulator tool.

.ASM file: File we get after converting the above .VM file with the help of VM Translator. This file is tested in the CPU Emulator tool.

.HACK file: The final file which has machine codes in it is obtained by the converting the above .ASM file with the Assembler. This file is tested by using Hardware Simulator tool.

String Functions in JACK:

This class implements the String data type and various string-related operations.

1. Constructor String new(int maxLength): Constructs a new empty string (of length zero) that can contain at most maxLength characters.
2. Method void dispose(): Disposes this string.
3. Method int length(): Returns the length of this string.
4. Method char charAt(int j): Returns the character at location j of this string.
5. Method void setCharAt(int j, char c): Sets the j'th element of this string to c.
6. Method String appendChar(char c): Appends c to this string and returns this string.
7. Method void eraseLastChar(): Erases the last character from this string.

2. Methodology

2.1 Jack code:

Jack is a simple object-based language that can be used to write high-level programs, which has JAVA like syntax.

Although Jack is a real programming language, we use Jack for applications like:

1. How to build a compiler.
2. How the compiler and the language interface with the operating system.

Here the Jack code is implemented for our question and the code contains creation of:

1. String with maximum length.
2. Function which returns the length of string.
3. Function that returns the element when user gives index number.
4. Disposing a string.

We wrote code in two files; one is Main.jack file and another is String.jack file. String file contains all functions and through main file we can give perform the operations such as taking input from the user and calling Methods implemented in String.jack.

Now we need to compile this jack file to .VM file.

Steps:

1. The Jack file which contains code should be placed at tool folder in Nanad2tetris.
2. Next step is going to the folder which contains jack file, then click on the address path of that folder.
3. And remove all that address and enter (cmd), then command prompt will open.
4. In command prompt give command (JackCompiler<space><fileName.Jack>). it will compile that if no errors are found.
5. After compiling we can see a new .VM file created in the same folder itself.

2.2 VM Code:

The VM language consists of arithmetic, memory access, program flow, and subroutine calling operations. Which deals with stack operations.

All operations are done on a stack. It is also function-based. Each function has its own stand-alone code and is separately handled. The VM language has a single 16-bit data type that can be used as an integer, a Boolean, or a pointer.

VM emulator: Once a high-level program is compiled into VM code, the program can run on any hardware platform equipped with a suitable VM implementation. In this chapter we start building the VM implementation on the Hack platform and use a VM emulator.

VM Translator: used to translate High-level language to low-level language, from VM code to assembly code.

Using VM Translator that was built using python code, we can convert .VM file to .asm file (assembly language).

2.3 Assembly code:

Assembler is a program which converts basic computer instructions into binary code. It reads the assembly different language instructions and convert them.

An opcode basically gives information about the particular instruction. The symbolic representation of the opcode (machine level instruction) is called mnemonics and these mnemonics are used as objects.

We built this using python programming language which basically reads the different type of instructions and we used constructors for reading the instructions and used-self operator for calling the objects that are created in the constructor.

While, running the Assembler file that was built by us. We need to place the vm file and assembler in a similar folder and run. Then the output is produced in that folder itself.

Finally, the output produced by the assembler(.hack) file and this file later compared to the file that is produced by the inbuilt assembler that is present in the nand2tetris.

2.4   Hardware Simulator:

Software simulation is often used when evaluating computer systems. It may, however, be time consuming and expensive, especially when many runs with slightly differing parameters have to be done or when small changes in the model have to be performed frequently. In such situations, and when not too much detail and only small amounts of output data are required, the hardware simulator described in this paper may be an effective tool. The hardware simulator is an apparatus on which physical models of computer systems can be built.

To run the machine codes generated by our own assembler in the RAM.hdl, we need to run the Computer.hdl program.

In order to run the hdl code of computer we need to add other two hdl codes.The two hdl codes are CPU and Memory.

After loading the computer.hdl code in hardware simulator, below the screen part the RAM and ROM file location will appear.

In the ROM folder we will load the hack file which is formed from the assembler.

After running the hack file in the RAM part, we will get the result in binary form.

3. Experiments/ Results

Jack compilation:
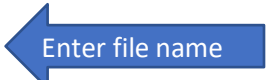
Result:

VM Translator:

```
def translate(self, vm_file):
    parser = Parser(vm_file)
    self.cw.set_file_name(vm_file)
    while parser.has_more_commands:
        parser.advance()
        self.cw.write('// ' + ' '.join(parser.curr_instruction))
        if parser.command_type == 'C_PUSH':
            self.cw.write_push_pop('C_PUSH', parser.arg1, parser.arg2)
        elif parser.command_type == 'C_POP':
            self.cw.write_push_pop('C_POP', parser.arg1, parser.arg2)
        elif parser.command_type == 'C_ARITHMETIC':
            self.cw.write_arithmetic(parser.arg1)

Main("Main.vm")
```
Enter file name

Result:

hackassembler	7/13/2022 4:00 PM	Python File	8 KB
Main	7/13/2022 4:33 PM	ASM File	53 KB

Output File

<u>Assembler:</u>

```python
infile = "Main.asm"
outfile = "Main.hack"
cleanlines = clean_file(infile)
myassembler = hackAssembler()
codelines = myassembler.handle_labels(cleanlines)

hackcode = []
for line in codelines:
    if line[0] == '@':
        cmd = myassembler.handle_a_expr(line)
        hackcode.append(cmd)
    else:
        cmd = myassembler.handle_c_expr(line)
        hackcode.append(cmd)

outfile = open(outfile, 'w');
for cmd in hackcode:
    outfile.write(cmd[0] + '\n')
outfile.close()
```

Result:

hackassembler	7/13/2022 10:21 PM	Python File	8 KB
Main	7/13/2022 10:02 PM	ASM File	53 KB
Main.hack	7/13/2022 10:21 PM	HACK File	82 KB
Main.vm	7/13/2022 3:56 PM	VM File	25 KB
VM_Translator	7/13/2022 10:02 PM	Python File	9 KB

Output File

RAM.hdl

Conclusion:

In this Project we have implemented and shown, how jack code works and how a Hack Computer works, how high-level language is abstracted at machine level.

We have successfully implemented, compiled and shown the results of String Library and it's functioning using programming in Jack Programming Language.

Reference:

1. Text Book : The Elements Of Computing Systems.

2. Nand2tetris - https://www.nand2tetris.org/

3. Jack programming -

   https://www.cs.huji.ac.il/course/2002/nand2tet/docs/ch_9_jack.pdf

4. JACK OS Errors

Thank you!