CS335 Project Milestone 4

Akhil Jain
200077
akhilj20@iitk.ac.in

Harsh Jain
200412
harshj20@iitk.ac.in

Jasjot Singh
200468
jasjots20@iitk.ac.in

1 Introduction

Continuing from all the previous milestones, in this milestone we have implemented a translator that generates x86_64 assembly instructions from the 3AC code generated in the previous milestone. The assembly file is capable of being compiled by the gcc compiler. The assembly file is always named **code.s** and is generated in the **src** directory. The 3AC file gets generated in the **tests** directory.

1.1 Basic Features

The following basic features have been supported by our implementation:

- Primitive data types and x86 support
- Multidimensional arrays (upto 3D arrays)
- All the basic operators
- Control flow including **if-else, for** and **while**
- Support for **recursion**
- Support for **println()**
- Support for **classes** and **objects**
- Type casting upto 3AC

1.2 Optional Features

We have also included support for the following optional features:

- do-while support
- Array initialization of the form **int arr[] = {1,2,3,4,5}** works as well (this includes multidimensional arrays). Also, support for both C-style and Java-style array declarations has been added
- Inheritance

1.3 Command lines

Please first switch to the **src** directory before running these commands. The compilation and execution instructions are as follows:

1. make clean
2. make
3. ./milestone4 ../tests/testcase.java
4. make run
5. ./code

We prefer that the 3rd command is executed with the verbose flag (−−verbose)

- The **make clean** command removes all the unnecessary files
- The **make** command then compiles all the files necessary for executing the code
- The third command runs the executable and generates the 3AC file (named as testcase.3ac) and the **code.s** assembly file
- The fourth command first makes the object file from the assembly file and then generates the executable. It has two commands in it.
- The final command runs the executable

Note:
For the third command, you can also add additional arguments as follows that change what is generated as output:

- **./milestone4 ../tests/testcase.java −−verbose**
  This is to get detailed error reports
- **./milestone4 ../tests/testcase.java −−3ac**
  This will generate only the 3AC files and not the .s assembly file
- **./milestone4 ../tests/testcase.java −−symtab**
  This will generate the symbol tables as well in a separate directory called **symTables**
- **./milestone4 ../tests/testcase.java −−dot**
  This will generate the parse tree and then you can use the dot command for viewing it
- **./milestone4 ../tests/testcase.java −−help**
  This will lead you to the helper page

Note:
Our assembly file does not require any manual changes to run it with the gcc compiler

1.4 Extensions to 3AC

- Stack manipulation has been changed slightly. We push the base pointer after entering the function now, and we pop the arguments after the **ret** call
- We have added the default constructor code in the assembly file if it has not already been defined in the Java file
- We have introduced a new codeblock **main** which is the *pseudo-main* which is used my **main.2** and **main.2** is the actually main class and has the assembly instructions corresponding to that
- The 3AC code segments such as **t0** $= -8(\%rbp)$ and t0 = const/t1/%rbp have been replaced by $-8(\%rbp)$ = const/t1/%rbp

Note: Static support is not complete

2 Contribution Table

1	Akhil Jain	200077	akhilj20@iitk.ac.in	35
2	Harsh Jain	200412	harshj20@iitk.ac.in	36
3	Jasjot Singh	200468	jasjots20@iitk.ac.in	29

3 Assumptions

These are the assumptions made in the 4th milestone:

- The Java print command has to be run as **System.println()** and NOT as System.out.println()
- You cannot access a field array without creating an object in the main function
- The name of the Java testcase file and the class containing the main function in the Java file must be the same
- The main function must have the argument **String[] args** as per the Java specifications
- A function or a class must be declared first before calling it.