CS 335 Semester 2022–2023-II: Project Milestone - 4

Harsh Trivedi (200422), Hitesh Anand (200449), L Gokulnath (200542)

April 21, 2023

Features Supported

1. The following basic features are supported:
    - Int,byte,long,short are supported.
    - Multidimensional Arrays are supported upto maximum 3 Dimensions.
    - Basic Operators such as:
        - Arithmetic Operators : +, -, *, /, %
        - Preincrement, Postincrement, Predecrement, Postdecrement
        - Relational Operators : $>, <, >=, <=, ==, !=$
        - Bitwise Operators : $\&, |, \hat{}, \tilde{}, <<, >>, >>>$
        - Logical Operators : $\&\&, ||, !$
        - Assignment Operators : $=, +=, -=, *=, /=, \&=, |=, =, \%=, <<=, >>=, >>>=$
        - Ternary Operator
    - Control Flow via:
        - If-else statements
        - For loops
        - While loops
    - Method Declaration and Method Invocation
    - Recursion
    - Printing functions such as System.out.println() for integer only.
    - Classes and Objects
2. The following additional features are also supported:
    - Import Statements

3. The following stack manipulation features are supported:
    - On invoking methods, parameters are pushed from right-to-left, following the **cdecl** calling convention. We have implemented this using the 'pushparam' instruction
    - Assuming a word size of 8, the stack pointer is increased by a value = 8 * Number of parameters
    - Return instruction in the IR represents a return statement in the source code
    - When a called method returns, the stackpointer is changed back to its original value
4. Features that could not be implemented
    - Static variables could not be implemented. We could not implement this feature due to lack of sufficient time. However, We had planned to implement the static variables under .data section, and we could have implemented these featues if there was more time.

Test Cases

In total, 10 test cases are provided to test the features mentioned above. These test cases test the following features:

- test1.java : simple test case for function calls and object creation
- test2.java : while loops, and if-else
- test3.java : Use of this pointer
- test4.java : complex arithmetic operations
- test5.java : test for arrays
- test6.java : test for short-circuiting
- test7.java : If-else and while example
- test8.java : test for recursion
- test9.java : object creation from a separate call
- test10.java : test for printing numbers in range 1 to n using while loop

Extensions from Previous Milestones

- Along with the complete 3AC file, a separate 3AC file has been created for each function, and field declarations for each class.
- In the symbol table and 3AC files, scope information has been added as a suffix to distinguish between multiple variables having same name.
- The naming convention for the individual function 3AC files is : <ClassName>_<FunctionName>.3ac
- The naming convention for the individual function symbol table files is : <ClassName>_<FunctionName>.csv
- We have created a program to generate x86_64 assembly code using this modified 3ac.

Individual Contributions

	Name	Roll Number	Email ID	Contribution(%)
1	Harsh Trivedi	200422	tharsh20@iitk.ac.in	33.33
2	Hitesh Anand	200449	ahitesh20@iitk.ac.in	33.33
3	L Gokulnath	200542	lgokulnath20@iitk.ac.in	33.33

Compilation and Execution Instructions

For Compiling, simply run the following command in the directory containing the Makefile:

Compilation

```
$ cd milestone4/src
$ make
```

For execution, there are four possible options:

- −−input : To specify the path of the input file (Required)
- −−output : To specify the path of the output txt file for 3AC(Required)
- −−verbose : If used, the user can see the exact rules that are being used during the parsing. And also the func tree and whole scope tree and the ir in terminal. (Optional)
- −−help : Can be used to see the available options and their usage

An example using the first three options:

Execution

```
$ ./parser --input=../tests/test_1.java --output=3ac.txt --verbose
```

The resulting output will contain the IR (in a 3AC representation) with the stack pointer manipulation instructions (as mentioned in detail above) in the folder named temporary.
Now to create assembly for it first we will execute the assembly generator using the command

Compiling and Execution

```
$ g++ asmgen.cpp;./a.out --input=<input_file> --output=<output_file>
$ gcc <output_file> -o <executable_file>
$ ./executable_file
```

For execution, there are three possible options:

- −−input : To specify the path of the input file (Required). Here mentioned the file name which is the output of above command as it is.
- −−output : To specify the path of the output .s file for assembly code(Required)
- −−help : Can be used to see the available options and their usage

An example in the continuation of the above one is:

Execution

```
$ g++ asmgen.cpp;./a.out --input=3ac.txt --output=test.s
$ gcc test.s -o test
$ ./test
```

Here, test.s contains the assembly code for the IR contained in the 3ac.txt file. We compile the test.s file to create the executable file named 'test'. Finally, we simply run the executable file 'test' and observe the output.

Assembly Convention and assumptions:

1. Only integer type of primitive data type is allowed. And we have allotted to all integer 8 bytes of space. And this maintain the alignment property.
2. All the arrays and objects are allocated on heap as done in java.
3. We have used 6 registers to transfer arguments to a function, and if there are more, we push them onto the stack. Here, the first register is reserved for this pointer.
4. While transferring the arrays as an argument we transfer the pointer to the array and its width in the order: a, w1, w2.. (which can go to both stack or register depending on number of argument).
5. Also, we have assumed that the maximum dimension for array is 3.