CS 6410: Compilers

Fall 2023

HW 5 – x86-64. Runtime organization. Code shape.

Assigned: Wednesday, November 1, 2023, Due: Saturday, December 2, 2023

Instructor: Tamara Bonaci
Khoury College of Computer Sciences
Northeastern University – Seattle

- Please submit your homework as a single .pdf file through Canvas.
- You do not have to type in your submission - hand-written and then scanned, or photographed documents are fine, as long as the total size of your document is not too big, and your document is readable.
- This assignment is meant to be worked on individually, and you should submit it by **11:59pm on Saturday, December 2, 2023**.

1 Problem 1

Consider the following class:

```
class Foo {
  public int maxv(int x, int y) {
    if (x < y) return x;
    else if (y < x)
    return this.maxv(y, x);
    else
    return x;
} }
```

Please translate method `maxv` into x86-64 assembly language. You should use the standard runtime conventions for parameter passing (including the `this` pointer), register usage, and so forth , including using %rbp as a stack frame pointer.
Since class `Foo` has only one method, `maxv`, you should assume that the vtable layout for the class has a single pointer to this method at offset +8.
`call` instruction hints: if %rax contains a pointer to (i.e., the memory address of) the first instruction in a method, then you can call the method by executing call %rax. If %rax contains the address of a vtable, we can call a method whose pointer is at offset d in that vtable by executing call d(%rax).

2 Problem 2

Consider the following MiniJava program:

```
class Base {
  int a;
  int b;
  public int f(int n) {
    b = n + 1;
    return n + 2;
  }
  public int g(int n) {
    return a + n;
  }

  public int setA(int v){
    a = v;
    return a;
  }

  public int setB(int v){
    b = v;
    return b;
  }
}


class Sub extends Base {
  int c;
  public int setC(int v) {
    c = v;
    return c;
  }

  public int g(int n) {
    c = this.f(b);
    return b + n;
    }
}
```

Please translate method `g(n)` in class `Sub` into x86-64 assembly language. You should use the standard runtime conventions for parameter passing (including the this pointer), register usage, and so forth, including using %rbp as a stack frame pointer.
`call` instruction hints: if %rax contains a pointer to (i.e., the memory address of) the first instruction in a method, then you can call the method by executing call %rax. If %rax contains the address of a vtable, we can call a method whose pointer is at offset d in that vtable by executing call d(%rax).

Reference and ground rules for x86-64 code:

– You must use the Linux/gcc assembly language, and must follow the x86-64 function call, register, and stack frame conventions.
– Argument registers: %rdi, %rsi, %rdx, %rcx, %r8, %r9 in that order.
– Called function must save and restore %rbx, %rbp, and %r12-%r15 if these are used in the function
– Function result returned in %rax/
– %rsp must be aligned on a 16-byte boundary when a call instruction is executed
– %rbp must be used as the base pointer (frame pointer) register for this question, even though this is not strictly required by the x86-64 specification.
– Pointers and ints are 64 bits (8 bytes) each, as in MiniJava.
– Your x86-64 code must implement all of the statements in the original method. You may not rewrite the method into a different form that produces equivalent results (i.e., restructuring or reordering the code). Other than that, you can use any reasonable x86-64 code that follows the standard function call and register conventions – you do not need to mimic the code produced by your MiniJava compiler.
– Please include brief comments in your code to help us understand what the code is supposed to be doing.