Project 3c: Code Generation
Due: Sunday, April 22, 2018, by 11:59pm

Description
We now have everything we need from the front end and middle to generate machine code.

In this final phase of the project, you will walk the IR from Project 3b and generate MIPS machine code that runs under MARS.

Since this is another non-trivial task to complete, I am imposing a milestone approach to the project. You must do the first **five** steps of this project in the order below. You will only be graded by the highest level that works completely (plus partial credit for the next milestone attempted.) The three additional features should only be attempted if these required features work.

Milestone 1: Hello, 9
```
class Test {
    public static void main(String[] args) {
        System.out.println(9);
    }
}
```

No additional classes, no register allocation, no expressions. Just print 9 on the screen.

Milestone 2: Hello 5+4
```
class Test {
    public static void main(String[] args) {
        System.out.println(5+4);
    }
}
```

Milestone 1 plus an expression with constants. The expression can only use $t0-$t9. Abort if there are more than 9 values necessary (you do not need to implement reuse of the registers).

Milestone 3: Hello 9 with functions and parameters
```
class Test {
    public static void main(String[] args) {
        System.out.println(new Test2().Start(9));
    }
}
class Test2 {
    public int Start(int y) {
        return y;
    }
}
```

Call an identity function and print out the result. Note that you do not need to implement “new” yet since there are no class scope variables.

Call multiple functions with 4 or less parameters.

Milestone 4: Hello with variables

```
class Test {
    public static void main(String[] args) {
        System.out.println(new Test2().Start(0));
    }
}
class Test2 {
    public int Start(int y) {
        //Arbitrary straight-line code
        return y;
    }
}
```

The expressions can use all general purpose registers that are not $zero, $a0-$a3, $v0, or used in manipulating the stack ($sp, $fp). Abort if you need more. Save all registers on function call. Restore all on function return.

Milestone 5: Control Flow and Recursion

Support if and while (iffalse and goto IR statements).

Support recursive functions (probably trivial with Milestone 4 and control flow).

Additional Work

For the remaining points, we will implement three additional features:

1. Register Allocation
2. Objects and Arrays
3. Optimization

You may approach these three in any order, but the register allocator has three sub steps that will guide you and grading.

RegAlloc Milestone 1: Register Allocation, no spills, no coalescing.

Compute the liveness of the variables and use graph coloring to allocate them to the registers as in Milestone 4. Abort if a spill is required.

Precolor parameters into $a0…$a3 and return values in $v0.

RegAlloc Milestone 2: Register Allocation, no spills, coalescing.

Add coalescing to RegAlloc Milestone 2.

RegAlloc Milestone 3: Register Allocation, spills, calling convention

Eliminate the naïve spilling of all registers in the function prologue and epilogue.

Support spills.

Support the proper MIPS calling convention by having calls interfere with the caller-saved ($t0-$t9) registers.

Objects and Arrays

Create dynamic objects and arrays with new. An array of n integers will require (n+1)*4 bytes of storage. At position n+0, store the length of the array. Locations n+4 … n+4*length should hold the array data.

Optimization

Implement any optimization of your choosing. This **must** be something that reduces dynamic instruction count so that we can see the effect of the optimization in MARS. Write up what you implemented in your README and have the optimization enabled by passing -O1 to your program so we can see the differing code. Provide an example that exercises your optimization.

Submission

Please copy your zipfile to `unixs.cis.pitt.edu` and then copy your file to:

```
~jrmst106/submit/1622
```

Please remove all .class files and version control directories before submission. Make sure to name the file with both you and your partner's username.

Do include:

- All of the source code
- A README
- A Makefile
- Examples testing each Milestone, named Milestone1.java … Milestone10.java