Assignment Description

In this assignment, you will build an new compiler backend that optimizes code generation by performing instruction selection and register allocation. The old backend, which generates unoptimized code, should remain functional. However, it is recommended that you reuse as much code as possible from the previous backend. You are required to implement the following in the optimized backend:

1. **Instruction Selection and Abstract Assembly Generation**. Start with the three-address code and CFG representation, rewrite accesses to method parameters as accesses to stack locations (using offsets from the frame pointer), perform local value numbering to build the DAG of each basic block, then tile the DAG and generate abstract assembly code. The generated code consists of x86 assembly instructions, where local and temporary variables are treated as "abstract" machine registers. You will design a set of tiles for a number of assembly instructions and use the greedy algorithm (Maximal Munch) to perform the tiling.

   Note that, for expressions that represent heap data (e.g., `o.f` or `v[i]`), the same expression may represent different values at different points in the program, because the referenced object may change. Make sure this fact is reflected in your DAG construction.

   Your compiler should include comments in the generated code so that one can easily identify the code generated for each tile. The comments should also indicate the corresponding three-address instruction(s) that the tile consists of. You will be generating 32-bit Pentium code, so all variables and registers will take a full 32 bits of storage. Although it would be possible to generate code that stores booleans more compactly, this data type will be stored in 32-bit words, as in the previous assignments.

2. **Register Allocation**. Next, you will translate the assembly code from the previous step into actual assembly code. For this, you will implement local register allocation for each basic block.

   Make sure your register allocator deals properly with instructions that operate on fixed registers, and correctly handles callee-save and caller-save registers. Callee-saved registers should be pushed on the stack only when the invoked method updates their value; caller-saved registers should be saved only if they are live at the call.

   Each instruction in the generated assembly code must indicate which registers have been freed at that point, which registers have been allocated, and to what variables they have been allocated. Failure to produce this information will result in point deductions.

Command line invocation. As in the previous assignments, your compiler must be invoked with a single file name as argument, representing the program that you want to compile. In

addition to all of the options from the previous assignments, your compiler must support the following command-line options:

- Option `-opt-backend` to run the optimized backend. Without this flag, the compiler will use the backend developed earlier in Assignment 3 and generate unoptimized machine code.
- Option `-print-aa` to print the abstract assembly code for each method (separately), and each basic block. Make sure you clearly indicate the method name, and basic blocks are clearly specified in the output. This option is applicable only if `-opt-backend` is also specified.

Bonus Points

For an additional 10 bonus points, implement register allocation via graph coloring, as discussed in class. This will also require using the dataflow framework from the previous assignment to perform live variable analysis. In contrast to the previous assignment, liveness information is being computed for the abstract assembly code. Mention in the writeup if you have implemented this algorithm. The command line option for this transformation is `-ra-coloring`.

What to turn in

- A file `pa5.zip` containing just your source code and your test cases (in directories `/src` and `/test`).
- A clear and concise document describing the your code structure and testing strategy. You should include a description of your design for the abstract assembly code, for tiles, for the implementation of register allocation, and any other important components, algorithms, or techniques that you used in the assignment.