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 as possible code from the previous backend, and avoid code duplication. You are required to implement the following in the optimized backend:

1. **Instruction Selection and Abstract Assembly Generation**. Start with the low-level IR, rewrite accesses to method parameters as accesses to stack locations (using offsets from the frame pointer), build the DAG for the computation in 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 tile each DAG.

   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 LIR 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 the graph-coloring register allocation presented in class (and described in detail in the Tiger book). You are not required to implement register coalescing and removal of redundant move instructions (but feel free to implement these extensions and mention in the write-up if you do so).

   This part of the assignment will use the dataflow framework from the previous assignment to perform the live variable analysis required for register allocation. In contrast to the previous assignment, liveness information is being computed for the abstract assembly code (not the low IR representation).

   Make sure your liveness analysis and 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.

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 function (separately), along with the name of the function being printed. This option is applicable only if `-opt-backend` is also specified.
- Option `-print-ra` to print the register assignments for each function. The output for each function must print one line for each register, specifying which variables were allocated to that register, or that no variable is allocated to the register. This option is applicable only if `-opt-backend` is also specified.

What to turn in

Turn in on paper:

- 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.
- Feedback. Please tell us how much time you spent on each of the two parts of this assignment, what part was more challenging, and tell us how you think the assignment can be improved.

Turn in electronically:

- A file `pa5.tar.gz` containing just your source code and your test cases (in directories `/src` and `/test`).