Compilers - Phase 3 Report

Naved Rizvi
Usman Ehtesham Gul

Design Internals

For the MIPS calling convention implementation, we followed this document as it was easier to follow the convention and there were good examples. We first store all the arguments of any function calls within the current function. Then we store all the save registers. After that we store the return address of the current function. After that we store all local variables of the current function.

To implement naive, we would load and store a variable from memory to a register at each reference. For the register allocation, we would keep track of the offset of a given variable and the total size of the stack. And during instruction selection, we would load and store each variable for given instruction. Before each function call, we would store the save registers and arguments back on the stack and after the end of the function call, we would load the save registers and arguments from the stack. We do this to save the caller's data so that it will not be overwritten during the function call. We use hashmaps to keep track of a variable and its offset. We store the save and temp registers in a stack. When we need a register, we pop a register off the stack and after we are done with the register we put it back on the stack.

To implement intra-block, we determined the basic blocks of each function and created a CFG from those basic blocks. After that, we determined the frequencies of variables per basic block and in descending order of frequency, we would assign a register to that variable. If we ran out of registers, then we would follow the naive algorithm and load from and store to memory on each reference of the variable. At each block entry, we would load the assigned variables from the stack to the corresponding register and at the exit of the block, we would store the values of the registers back to the stack.

To implement Briggs, we first created a CFG per function. Then we did liveness analysis across the basic blocks to determine the live range of each variable per instruction. From the liveness analysis, we built the interference graph and then performed graph coloring on the interference graph using Brigg's algorithm.

To build the CFG, we used a hashmap mapping a basic block to its successors. To build the interference graph, we created a Graph data structure which internally used a hashmap to keep track of nodes and a mapping from a node to its neighbors.

Code structure

We define a TargetCodeGenerator class which is the main entry point to convert an IR to MIPS code. In the TargetCodeGenerator, we create FunctionBlock objects that keep track of instructions inside a given function, the static variables, local variables and arguments. We pass each function block to a MipsCodeGenerator class which is responsible for generating MIPS code instructions from the IR. The MipsCodeGenerator class allocates the stack, does instruction selection for a given IR instruction, and emits instructions for loading from memory to a register and storing from a register to a memory location. We created a separate class for MIPS code generation for Briggs, called MipsCodeGeneratorBriggs, because the MipsCodeGenerator class had too much coupling between register allocation and instruction selection especially for the intra-block algorithm. So to be able to focus more easily on instruction selection from Briggs, we implemented a separate class.

We also defined separate code structures to handle Basic Blocks, CFG generation, interference graph creation, and liveness analysis to scope logic/responsibility as much as possible and reduce coupling between different classes.

Bugs/Features Not Fully Operational

Allocation

For the allocation tests, only 4 tests are passing for naive and intrablock (demo_selection_sort, demo_fib, demo_prng and demo_tak) and only one test passing for Briggs (demo_selection_sort)

Allocation with limits

Same 4 tests passing for naive and intrablock and same single test passing for Briggs for allocation with limits.

For both groups of tests, there are deficiencies in the following:

- Floats not fully working for program that have arrays of floats or functions with float arguments
- For Briggs, there are bugs in files with functions (other than main). This is due to not properly loading values from registers to the stack before a function call and storing back to those registers after a function call. If we had more time, we would try to fix this properly to get more tests passed. We manually test against all tests run for naive and for all test excluding function_*.tiger, tests were passing for Briggs.

For partial credit

For the possibility of getting some more partial credit if possible, there were a couple of submissions where we got demo_fib and demo_tak working for Briggs. Here are the Gradescope submissions for these tests passing:

- demo_fib: link
- demo_tak: link