

Project 2: Program analyses, code transformations, and improved code generation — assigned incrementally and finally due on Friday 5 December — final presentations on Thursday 11 December (the official final exam day)

This second incremental release consists of Tasks 3–6. Release date 6 November. Submissions due 23 November.

Project 2 is a continuation of Project 1. You will now implement an optimizing compiler for the language WHILE.

Task 1: Consolidate all work from Project 1. (Tidy up and fix any problems you may have noticed.)

2.1 Labelled syntax

The labelled syntax decorates computational blocks (“program points”) with labels ℓ :

Commands $c ::= [x := a]^\ell \mid [\text{skip}]^\ell \mid c_1; c_2 \mid \text{if } [b]^\ell \text{ then } c_1 \text{ else } c_2 \text{ fi} \mid \text{while } [b]^\ell \text{ do } c_1 \text{ od}$

Task 2: Given an AST, produce a decorated AST with unique labels (conventionally, we use distinct numbers as labels). Display the decorated AST. Display (pretty-print) the annotated program in source form (with labels inside comments inserted at the appropriate place).

2.2 Control flow graphs

Task 3: Come up with a data structure (whichever is convenient in your implementation language) for representing control flow graphs. Convert the decorated AST into a CFG. Display the CFG.

2.3 Code generation

Task 4: From the CFG, generate virtual machine code. The code will superficially look like RISC-V assembly. Each variable of the WHILE program will be mapped to a RISC-V register, s_1, s_2, \dots , and we will pretend that there are as many s registers as we need for all the variables in the program.

You should use comments in the generated virtual RISC-V machine code to document which parts of the CFG / WHILE source the code corresponds to.

The function prologue will include instructions like this:

```
#s1<-input
ld s1,8(a0)
```

to load all the s registers from the array that was passed by reference from the main program (in register $a0$), and the function epilogue will include instructions like this:

```
#output<-s10
sd s10,80(a0)
```

to dump all the `s` registers back into this array so the main program can then print the final values of the variables.

For smaller WHILE programs, with no more than 11 variables, the generated virtual RISC-V machine code will be valid RISC-V assembly, which you can assemble and link with the generated C main program (which remains the same as in Project 1) and then run on our RISC-V machine.

2.4 Testing and performance evaluation

Task 5: Extensively test your compiler. If you haven't already, write unit test programs to demonstrate complete coverage of WHILE language features.

Task 6: Evaluate the performance of the generated code. Measure execution times using some example programs with few variables. Compare execution times achieved by your current Project 2 compiler with those achieved by your Project 1 compiler. (This is only a preliminary evaluation, so do not spend too much time on it, but you should already observe considerable improvement.)

What to turn in

Submit a tar file containing: your compiler code, the CFGs generated for some example programs (PDF), (virtual) RISC-V code for these example programs, test runs of some example programs, some performance results.