# Lab 5
# Smart IRs, part A: Control Flow Graph in SSA Form

## Objective

- Convert a CFG to SSA Form.
- Convert a CFG out of SSA Form.

During the previous lab, you wrote a dummy code generator for the MiniC language, and converted the linear representation into a CFG. In this lab the objective is to prepare the field for more advanced compilation techniques, which will allow us to emit more efficient RISCV code. We remind you there are slides on the course webpage to help: `https://github.com/Drup/cap-lab25`

This lab is in two parts, which will be graded together: at the end of lab 5b, you will have to deposit your work of this lab 5a as well as lab 5b.

**You will extend your previous code, in the same `MiniC` project, but in the `RegAlloc/` subdirectory**.

## 5.1 SSA Form

Most of the code related to SSA is located in the `RegAlloc/EnterSSA.py` and `TP05/ExitSSA.py` files. They respectively contain two main functions:

- `enter_ssa` which converts a control flow graph to SSA form.
- `exit_ssa` which removes the $\phi$ nodes, converting out SSA form.

The local documentation has been updated with:

- in the `Lib.PhiNode` module the notion of `PhiNode`, a subclass of `Statement` representing $\phi$ nodes.
- in the `Lib.Dominators` module several functions for performing CFG analyses related to dominance (computing the domination tree, ...).

The provided code uses Python sets, and you will have to manipulate these objects during this lab. Thus, we encourage you to consult the documentation on set operations:

- The tutorial `https://docs.python.org/3/tutorial/datastructures.html#sets`
- The API `https://docs.python.org/3/library/stdtypes.html#set`

The end goal is to complete the missing pieces for entering into SSA form and then for leaving SSA form, respectively in `RegAlloc/EnterSSA.py` and `TP05/ExitSSA.py`.

EXERCISE #1 ► **Understanding the construction of the dominance frontier**

Three main functions are provided in `Lib.Dominators`:

- `computeDom`, which computes the dominators of each block
- `computeDT`, which builds the domination tree of the CFG
- `computeDF`, which yields the dominance frontier of each block

Understand what each of these functions is doing and how they are working together. The algorithms they implement are those seen in the course.

You can look at the source of these functions in the local documentation.

## 5.2 Conversion to SSA Form

In this section, we will complete the `EnterSSA.py` file.

EXERCISE #2 ► **Putting everything together**

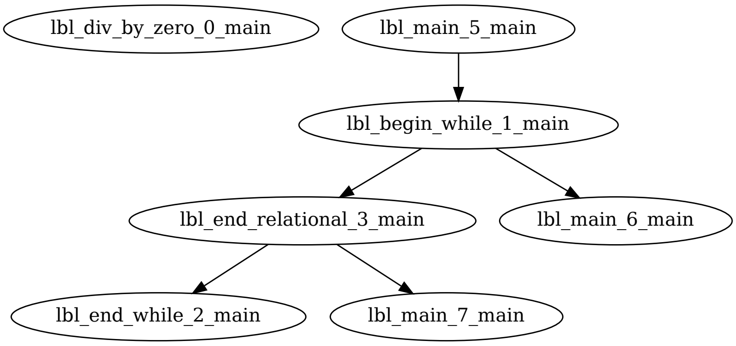


Figure 5.1: The domination tree of df03.c

On top of the dominance-related helper functions, `EnterSSA.py` contains two functions that are useful for going to SSA Form:

- provided the dominance frontier of the CFG, `insertPhis` inserts $\phi$ nodes in the CFG.
- then, `rename_variables` renames variables that appear in instructions or $\phi$ nodes, in accordance with the previously inserted $\phi$ nodes; `rename_block` is an auxiliary function for `rename_variables`.

For now, we do **not** ask you to work on the three incomplete functions `insertPhis`, `rename_block` and `rename_variables`, so your code will not be testable right away.

Complete the function `enter_ssa` so that it puts a given CFG into SSA form. For that purpose you will use functions from the dominators library and the incomplete functions from `EnterSSA.py`. This should take about 5 simple lines of code.

If you called the previously mentioned functions and Pyright[1] does not find any typing error, you are probably fine.

To run MiniCC with your SSA implementation, run

```
python3 MiniCC.py --mode=codegen-ssa --reg-alloc none /path/to/example.c
```

You can add the `--dom-graphs` option to the `MiniCC.py` invocation to view the domination tree and the CFG annotated with dominance frontiers graphically.

EXERCISE #3 ► **Testing the dominance-related functions**

Run your compiler in SSA mode on various programs (the files `dfxx.c` from Lab 4 are interesting test cases) and check that the graphical domination tree and dominance frontiers are correct.

For the test `CodeGen/tests/provided/dataflow/df03.c` the domination tree should be as depicted in figure 5.1 while the annotated CFG should look like figure 5.2.

EXERCISE #4 ► **Adding $\phi$ nodes**

We will now insert the $\phi$ nodes. We recall the following algorithm from the course

```
Insert-phi ::=
  for x in Vars:
    for d in Defs(x):
      for b in DF(d):
        if there are no ϕ-node associated to x in b:
          add one such ϕ-node
          add b to Defs(x)
```

**Complete in `EnterSSA.py` the procedure `insertPhis` which inserts the $\phi$ nodes. Pay attention on where to add the $\phi$ instructions in the blocks.**

At this point, you can use

[1] Reminder: `make test-pyright` allows you to run Pyright on your code at any time.

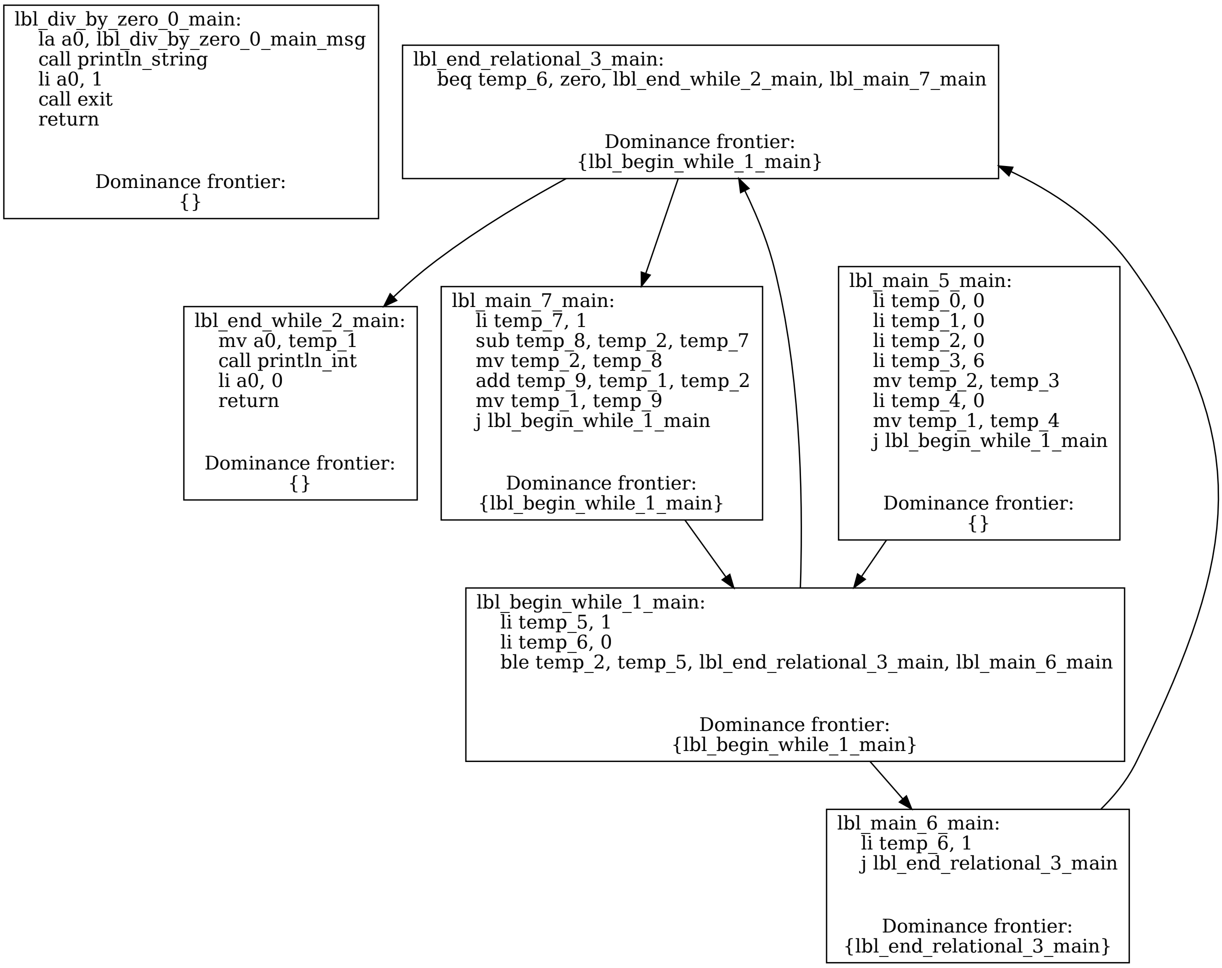


Figure 5.2: The annotated CFG of df03.c

```
python3 MiniCC.py --mode=codegen-ssa --reg-alloc none --ssa-graphs /path/to/example.c
```

to see the CFG almost under SSA Form: the $\phi$ nodes are all of the form `temp_x = ϕ(temp_x, ..., temp_x)`. The next step is to rename the variables.

EXERCISE #5 ► **Variable renaming**

Complete the missing pieces in the functions `rename_block` and `rename_variables`, following the algorithm of the course.

At this point, you should be able to call the `enter_ssa` procedure on any control flow graph to convert it to SSA Form. Use the `--ssa-graphs` option to visualize the resulting graph in SSA Form and verify its correctness.

## 5.3 Conversion out of SSA Form

In this section, we will complete the `ExitSSA.py` file.

The $\phi$ instructions we have added are convenient for manipulating the control flow graph, but are not implemented by processors. We need to remove them before emitting machine code.

EXERCISE #6 ► **Replacement of $\phi$ nodes by moves**

A $\phi$ node can be eliminated by creating *new blocks* containing moves.

For instance, consider the block $b_2$, with two parents $b_0$ and $b_1$, containing the following $\phi$ nodes:

$$x_2 = \phi(b_0 : x_0, b_1 : x_1)$$
$$y_2 = \phi(b0 : y_0, b_1 : y_1)$$

We will insert two new blocks:

- one between $b_0$ and $b_2$ containing the moves $x_2 \leftarrow x_0; y_2 \leftarrow y_0$
- one between $b_1$ and $b_2$ containing the moves $x_2 \leftarrow x_1; y_2 \leftarrow y_1$

1. **Complete the procedure `generate_moves_from_phis` which creates a list of `mv` instructions equivalent to the $\phi$ nodes to replace.**
2. **Complete the procedure `exit_ssa` which removes the $\phi$ nodes and add blocks containing the moves computed by `generate_moves_from_phis`. Do not forget to remove all edges and to modify instructions appropriately.**

At this point, you should be able to call the `exit_ssa` procedure on any control flow graph to convert it out of SSA Form. Use the `--graph` option to visualize the resulting CFG and verify its correctness.

EXERCISE #7 ► **Massive tests**

**At this point, all the tests should pass (or be skipped) after going in and out of SSA Form!** Check that all your tests from previous labs still work after transiting by SSA Form, using

```
make test-lab4 MODE=codegen-ssa
```