# Compiler Optimization Report

Logical expectation was that briggs would beat intrablock and intrablock would beat naive. Our implementation does not meet our expectations but the results are logical.

### 5.1) Benchmark Test: benchmark1 (0.0/0.0)

```
naive :          #reads : 191  #writes 146  #branches 61  #other 268 CORRECT Program
ib :          #reads : 227  #writes 226  #branches 61  #other 608 CORRECT Program
briggs :          #reads : 11  #writes 10  #branches 61  #other 608 CORRECT Program
```

### 5.2) Benchmark Test: benchmark2 (0.0/0.0)

```
naive :          #reads : 377  #writes 270  #branches 61  #other 392 CORRECT Program
ib :          #reads : 289  #writes 288  #branches 61  #other 1042 CORRECT Program
briggs :          #reads : 11  #writes 10  #branches 61  #other 1042 CORRECT Program
```

Instruction count:

benchmark1:

naive: 154 ib: 252 briggs: 190

benchmark2:

naive: 224 ib: 342 briggs: 260

Cause: Most of the difference between the number of instructions comes from saving and loading of values in blocks and functions. We see that intrablock has a lot of those, because at the start of each block, we load all local variables to stack and at the end - save them. In naive we don't use save registers at all, so none of these instructions are added to assembly. Briggs, on the other hand, doesn't save/load save registers per block, nevertheless it does so per function. So while it uses less instructions than ib, it still surpasses usage of naive allocation.

We analyzed how much value loading and flushing did each add take.
In our IR each add looked something like this:

```
add, _2_a, _2_b, _2_tmp_6
assign, _2_a, _2_tmp_6,
```

Naive allocation mips assembly:

```
lw $t3, -100($fp)
lw $t2, -92($fp)
add $t1, $t3, $t2
sw $t1, -36($fp)
```

```
lw $t2, -36($fp)
move $t3, $t2
sw $t3, -100($fp)
```

intrablock and briggs mips assembly:

```
move $t3, $s7
move $t2, $s6
add $t1, $t3, $t2
move $s4, $t1
move $t2, $s4
move $t3, $t2
move $s7, $t3
```

Here we see that all the loads and stores in the naive algorithm become just register moves. In intrablocka and briggs one obvious problem is that we could be doing add and move operations directly on the $sX registers but we made a tradeoff between easy implementation and less register copying(see `LoadedVariable` in design report).

Main observations:

- Naive algorithm is sometimes better than intrablock because naive does not have to save registers during function calls. This is caused by all the unnecessary saving and loading when lots of small blocks are involved.
    - This could be fixable partly by only flushing registers which have been written to and loading only those which will be read. For now we load variable even if it's only going to be written to.
- Briggs beats both algorithms in all cases. This is nice.

Shortcomings:

- We are saving all the save registers at the start of every function even if we don't use some of them at all inside the function (not inside blocks in intrablock). easily fixable
    - This is not really a big issue since Briggs is able to outperform naive even with this overhead. (But we assume most of briggs reads are writes are actually because of this)
- We could be using more saved registers but because of the previous shortcoming it is not optimal for now. easily fixable after the previous shortcoming is fixed.