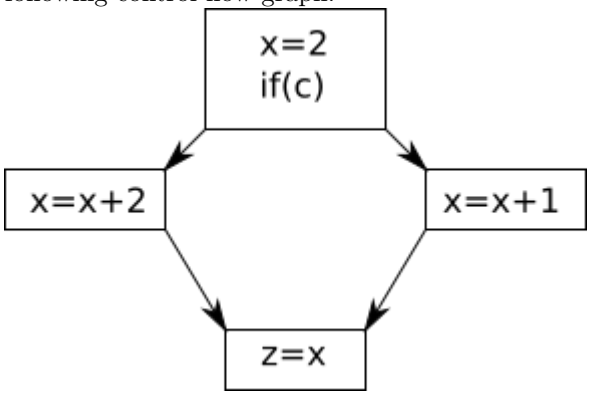
TDT4205 Problem Set 6

Answers are to be submitted via Blackboard by April 28^{th}.
This assignment will account for 10% of your final mark.

1 Data flow analysis (40%)

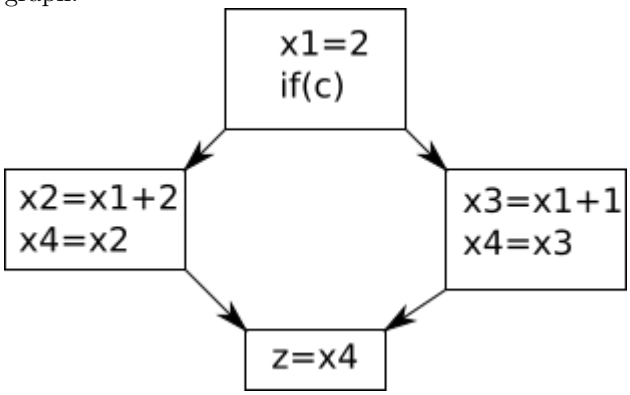
1.1 (15%)

Determine and solve the data flow equations of live variable analysis for the following control flow graph:



1.2 (15%)

If we convert the previous control flow graph into Static Single Assignment (SSA) form and back again, it results in the following, equivalent control flow graph:



Determine and solve its data flow equations for live variable analysis.

1.3 (10%)

Briefly, and in your own words, describe the effect(s) of SSA conversion on the results of live variable analysis.

2 Code Generation Part II (60%)

In the previous assignment we started generating code, but since we haven't implemented control structures we aren't able to run any really interesting programs yet. We will now complete our compiler by implementing if and while statements. This involves implementing relations, as well as IF/ELSE, WHILE-DO, and CONTINUE.

2.1 IF and ELSE statements (30%)

An if statement contains 2-3 children: RELATION, IF-block and an optional ELSE-block. Begin by implementing RELATION. This should work quite similarly to EXPRESSION and ASSIGNMENT, except the instruction to insert is `cmp`.

Next, add a conditional branch depending on the type of relation, to skip the if block if the condition is not met. Simply put, the generated code might look something like this:

```
// compute a
// push a
// compute b
// pop a
```

```
cmp a, b
jne .ELSE
// if-block
jmp .ENDIF
.ELSE:
// opt. else-block
.ENDIF:
// continued program flow
```

Remember that if statements may be nested, so make sure to name your labels unambiguously, e.g. by tracking nesting with a counter and a stack. Those who wish may also look into numeric labels[1].

2.2 WHILE statement (20%)

The WHILE statement can be made to work quite similarly to the IF statement. Compute the condition and skip the block if it's false, then at the end of the block insert a jump back to start.

```
.WHILE:
// compute a and b
cmp a, b
jne .ENDWHILE
// while-block
jmp .WHILE
.ENDWHILE:
// continued program flow
```

2.3 NULL statement (10%)

When running a while-loop we might want to skip some iterations using the `continue` keyword. Implement the handling of NULL_STATEMENT to insert a jump back to the start of the inner-most loop.

Congratulations, you have now built your very own compiler from start to end!

[1] https://docs.oracle.com/cd/E19120-01/open.solaris/817-5477/esqaq/index.html