Lesson 4

Based on Assignment 3, 2011

Compiler Design I (Kompilatorteknik I) 2018

DISCLAIMER: In the code listings of the following exercises we use a syntax that closely resembles C. However the C language does not allow by-reference parameter passing (which some parts of Exercise 1 assume).

1 Parameter passing

Suppose we have the following program:

```c
int foo(int a) {
    int b = a++;
    return b * a
}

int bar(int b) {
    return foo(b);
}

int main() {
    int c;
    c = 6;
    return bar(c);
}
```

Assume that the compiler allocates all the variables in the stack.

The single arguments of `foo` and `bar` can be passed either by-value or by-reference. For each of the four possible combinations (i.e. both arguments by-value, `foo`’s argument by-value & `bar`’s argument by-reference, etc.) describe:

1. What kind of data should the compiler put in the argument slots of the activation records for the calls on lines 7 and 13?

2. What kind of assembly code will be necessary to retrieve the value of variable `a` on line 2?
2 Code generation

Suppose that we want to generate code for the expression:

```plaintext
cond
  <p1> => <e1>;
  <p2> => <e2>;
  ...
  <pn> => <en>;
  1 => <e{n+1}>
dnoc
```

The evaluation of a `cond` expression begins with the evaluation of the predicate `<p1>` (if it exists, `n` can also be zero). If `<p1>` evaluates to a non-zero value, then `<e1>` is evaluated, and the evaluation of the `cond` expression is complete. If `<p1>` evaluates to zero, then `<p2>` is evaluated, and this process is repeated until one of the predicates evaluates to a non-zero value. The value of the `cond` expression is the value of the expression `<ei>` corresponding to the first predicate `<pi>` that evaluates to a non-zero value. If all the predicates evaluate to zero, then the value of the `cond` expression is `<e{n+1}>`.

Write a code generation function: `cgen(cond <p1> => <e1>; ...; <pn> => <en>; 1 => <e{n+1}>; dnoc)` for this conditional expression.

3 Local optimizations

Consider the following basic block, in which all variables are integers and `**` denotes exponentiation:

```plaintext
a := b + c
z := a ** 2
x := 0 * b
y := b + c
w := y * y
u := x + 3
v := u + w
```

Assume that the only variables that are live at the exit of this block are `v` and `z`. In order, apply the following optimizations to this basic block. Show the result of each transformation.

1. algebraic simplification
2. common sub-expression elimination
3. copy propagation
4. constant folding
5. dead code elimination

When you have completed part 5, the resulting program will still not be optimal. What optimizations, in what order, can you apply to optimize the result of 5 further?
4 Register allocation

Consider the following program.

L0:  e := 0
    b := 1
    d := 2
L1:  a := b + 2
    c := d + 5
    e := e + c
    f := a * a
    if f < c goto L3
L2:  e := e + f
    goto L4
L3:  e := e + 2
L4:  d := d + 4
    b := b - 4
    if b != d goto L1
L5:

This program uses six temporaries, a-f. Assume that the only variable that is live on exit from this program is e. Draw the register interference graph. (Drawing a control-flow graph and computing the sets of live variables at every program point may be helpful.)