Assignment 3

(due Fri, 9/12/2011)

Compiler Design I (Kompilatorteknik I) 2011

DISCLAIMER: In the code listings of the following exercises we use a syntax that closely resembles C. However the C language allows neither nested function definitions (which appear in Exercise 2) nor true by-reference parameter passing (which some parts of Exercise 1 assume).

1 Parameter passing

Suppose we have the following program:

```
1   int foo(int a) {
2     int b = a++;
3     return b * a
4   }
5
6   int bar(int b) {
7     return foo(b);
8   }
9
10  int main() {
11    int c;
12    c = 6;
13    return bar(c);
14  }
```

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 Activation records and scoping

In the following listing we show an excerpt of a program in a language that allows nested definition of functions, is *statically scoped* and uses *calls-by-value*:

```
 1  int foo(int a) {
 2    return 2 * a
 3  }
 4
 5  int bar(int b, int c) {
 6
 7    int baz(int d, int e) {
 8      int f, g;
 9      f = 2 * e + b;
10      g = foo(f + d);
11      return g;
12    }
13
14    int h;
15
16    h = baz(2 * b, c);
17    return h + 42
18  }
```

Suppose we have in our `main()` function a call to `bar(1, 2)`. Assume that the compiler allocates all the variables (named and temporary) on the stack.

1. What will be the return value of this call?
2. Give a diagram of the state of the stack right before the return of the call to function `foo` on line 10. Your diagram should contain at least the following information regarding the calls to `foo`, `bar` and `baz` functions (including the initial call):
   (a) boundaries of the activation records
   (b) location of the actual parameters for the calls to these functions
   (c) location of the return values
   (d) location of the return addresses
   (e) location and contents of the control links and access links
   (f) *Optional:* Location of temporary variables
   (g) *Optional:* Contents of actual parameters
3. Show where the values of the following variables are located in the stack and describe the code that should be generated by the compiler to access these values in the specified locations (including any points of reference, like the stack pointer). You may use as many registers as you like to calculate any addresses that are needed:
   (a) Variable `f` on line 10
   (b) Variable `e` on line 9
   (c) Variable `b` on line 9

3 Code generation

Suppose that we want to generate code for the expression:

```
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.

4 Local optimizations

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

```
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?

5 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.)

Instructions

There are two ways to submit this assignment:

1. Submit a physical copy of your answers in my mailbox (Aronis Stavros, 59) on the 4th floor of building 1, opposite the 'fika' room.

2. Send an email with an electronic copy of your answers to `stavros.aronis@it.uu.se`.

Good luck!