```
1   // Copyright 2013 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas
2
3   package jminusminus;
4
5   import java.util.ArrayList;
6   import java.util.LinkedList;
7   import java.util.Queue;
8   import static jminusminus.NPhysicalRegister.*;
9
10  /**
11   * Implemets a naive register allocation method. Each interval is considered
12   * live for the entire cfg. Intervals are assigned physical registers on a first
13   * come basis. When we run out of registers, we reuse the ones already assigned
14   * and spill.
15   */
16
17  public class NNaiveRegisterAllocator extends NRegisterAllocator {
18
19      /**
20       * Construct a NNaiveRegisterAllocator.
21       *
22       * @param cfg
23       *            an instance of a control flow graph.
24       */
25
26      public NNaiveRegisterAllocator(NControlFlowGraph cfg) {
27          super(cfg);
28      }
29
30      /**
31       * Build intervals with (naive) register allocation information in them.
32       */
33
34      public void allocation() {
35          // In this allocation scheme, each interval just has a single
36          // range spanning the entire cfg.
37          for (NInterval interval : cfg.intervals) {
38              NBasicBlock lastBlock = cfg.basicBlocks
39                      .get(cfg.basicBlocks.size() - 1);
40              NLIRInstruction lastLir = lastBlock.lir
41                      .get(lastBlock.lir.size() - 1);
42              interval.ranges.add(new NRange(0, lastLir.id));
43          }
44
45          // Allocate any fixed registers (a0, ..., a3 and v0) that were
46          // assigned during generation phase to the appropriate
47          // interval.
48          for (int i = 0; i < 32; i++) {
49              if (cfg.registers.get(i) != null) {
50                  cfg.intervals.get(i).pRegister = (NPhysicalRegister)
cfg.registers
51                          .get(i);
52              }
53          }
54
55          // Assign stack offset (relative to fp) for formal parameters
56          // fourth and above, and stack offset (relative to sp) for
57          // arguments fourth or above.
58          for (NBasicBlock block : cfg.basicBlocks) {
59              for (NLIRInstruction lir : block.lir) {
60                  if (lir instanceof NLIRLoadLocal) {
61                      NLIRLoadLocal loadLocal = (NLIRLoadLocal) lir;
62                      if (loadLocal.local >= 4) {
63                          NInterval interval = cfg.intervals
64                                  .get(((NVirtualRegister) loadLocal.write)
65                                          .number());
```

```
                    interval.spill = true;
                    interval.offset = loadLocal.local - 3;
                    interval.offsetFrom = OffsetFrom.FP;
                }
            }
        }
    }

    // Allocate registers.
    Queue<NInterval> assigned = new LinkedList<NInterval>();
    for (int i = 32, j = 0; i < cfg.intervals.size(); i++) {
        NInterval interval = cfg.intervals.get(i);
        if (interval.pRegister == null) {
            if (j >= NPhysicalRegister.MAX_COUNT) {
                // Pull out (from a queue) a register that's
                // already assigned to another interval and
                // re-assign it to this interval. But then
                // we have a spill situation, so
                // create an offset for the spill.
                NInterval spilled = assigned.remove();
                spilled.spill = true;
                if (spilled.offset == -1) {
                    spilled.offset = cfg.offset++;
                    spilled.offsetFrom = OffsetFrom.SP;
                }
                interval.pRegister = spilled.pRegister;
                interval.spill = true;
                if (interval.offset == -1) {
                    interval.offset = cfg.offset++;
                    interval.offsetFrom = OffsetFrom.SP;
                }
            } else {
                // Allocate free register to interval.
                NPhysicalRegister pRegister = NPhysicalRegister.regInfo[T0
                        + j++];
                interval.pRegister = pRegister;
                cfg.pRegisters.add(pRegister);
            }
            assigned.add(interval);
        }
    }

    // Make sure that inputs of LIR instructions are not all
    // assigned the
    // same register. Also, Handle spills, i.e., generate loads
    // and
    // stores where needed.
    for (int i = 1; i < cfg.basicBlocks.size(); i++) { // We
        // ignore
        // block B0
        NBasicBlock block = cfg.basicBlocks.get(i);
        ArrayList<NLIRInstruction> newLir = new ArrayList<NLIRInstruction>();
        for (NLIRInstruction lir : block.lir) {
            newLir.add(lir);
        }
        for (NLIRInstruction lir : block.lir) {
            int id = lir.id;

            if (lir.reads.size() == 2) {
                NInterval input1 = cfg.intervals.get(
                        lir.reads.get(0).number()).childAt(id);
                NInterval input2 = cfg.intervals.get(
                        lir.reads.get(1).number()).childAt(id);
                if (input1.pRegister == input2.pRegister) {
                    input2.pRegister = NPhysicalRegister.regInfo[T0
                            + (input2.pRegister.number() + 1)
                            % NPhysicalRegister.MAX_COUNT];
                }
            }
```

```
135
136                 // Loads.
137                 for (int j = 0; j < lir.reads.size(); j++) {
138                     NInterval input = cfg.intervals.get(
139                             lir.reads.get(j).number()).childAt(id);
140                     if (input.spill) {
141                         NLIRLoad load = new NLIRLoad(block, id
142                                 - lir.reads.size() + j, input.offset,
143                                 input.offsetFrom, input.pRegister);
144                         newLir.add(newLir.indexOf(lir), load);
145                     }
146                 }
147
148                 // Stores.
149                 if (lir.write != null) {
150                     NInterval output = cfg.intervals.get(lir.write.number());
151                     if (output.spill) {
152                         NLIRStore store = new NLIRStore(block, id + 1,
153                                 output.offset, output.offsetFrom, lir.write);
154                         newLir.add(newLir.indexOf(lir) + 1, store);
155                     }
156                 }
157             }
158             block.lir = newLir;
159         }
160     }
161
162 }
```