

NLinearRegisterAllocator.java

```

1  // Copyright 2013 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas
2
3  package jminusminus;
4
5  import java.util.ArrayList;
6
7  /**
8   * Implements the Linear Scan register allocation algorithm.
9   */
10
11 public class NLinearRegisterAllocator extends NRegisterAllocator {
12     /**
13      * Interval queues for tracking the allocation process.
14      */
15     private ArrayList<NInterval> unhandled;
16     private ArrayList<NInterval> active;
17     private ArrayList<NInterval> inactive;
18
19     /**
20      * Used to keep track of which intervals get assigned to what physical
21      * register. Needed only in allocateBlockedRegFor.
22      */
23     private ArrayList<ArrayList<NInterval>> regIntervals;
24     private int[] freePos, usePos, blockPos;
25
26     /**
27      * Construct a linear register allocator for the given control flow graph.
28      * @param cfg
29      *      the control flow graph instance.
30      */
31
32     public NLinearRegisterAllocator(NControlFlowGraph cfg) {
33         super(cfg);
34         unhandled = new ArrayList<NInterval>();
35         active = new ArrayList<NInterval>();
36         inactive = new ArrayList<NInterval>();
37
38         // Instantiate usePositions and freePos to be the size of
39         // the physical registers used.
40         freePos = new int[NPhysicalRegister.MAX_COUNT];
41         usePos = new int[NPhysicalRegister.MAX_COUNT];
42         blockPos = new int[NPhysicalRegister.MAX_COUNT];
43         regIntervals = new ArrayList<ArrayList<NInterval>>();
44         for (int i = 0; i < NPhysicalRegister.MAX_COUNT; i++) {
45             regIntervals.add(new ArrayList<NInterval>());
46         }
47     }
48
49     /**
50      * Perform the linear register allocation, assigning physical registers to
51      * virtual registers.
52      */
53
54     public void allocation() {
55         // Build the intervals for the control flow graph.
56         this.buildIntervals(); // The correct intervals are now in intervals
57
58         // Add all intervals corresponding to vregs to unhandled list
59         for (int i = 32; i < cfg.intervals.size(); i++) {
60             this.addSortedToUnhandled(cfg.intervals.get(i));
61         }
62
63         // Allocate any fixed registers (a0, ..., a3 and v0) that were
64         // assigned during generation phase to the appropriate
65         // interval.
66

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

67     for (int i = 0; i < 32; i++) {
68         if (cfg.registers.get(i) != null) {
69             cfg.intervals.get(i).pRegister = (NPhysicalRegister)
cfg.registers
70                 .get(i);
71         }
72     }
73
74     // Assign stack offset (relative to fp) for formal parameters
75     // fourth and above, and stack offset (relative to sp) for
76     // arguments fourth or above.
77     for (NBasicBlock block : cfg.basicBlocks) {
78         for (NLIRInstruction lir : block.lir) {
79             if (lir instanceof NLIRLoadLocal) {
80                 NLIRLoadLocal loadLocal = (NLIRLoadLocal) lir;
81                 if (loadLocal.local >= 4) {
82                     NInterval interval = cfg.intervals
83                         .get(((NVirtualRegister) loadLocal.write)
84                             .number());
85                     interval.spill = true;
86                     interval.offset = loadLocal.local - 3;
87                     interval.offsetFrom = OffsetFrom.FP;
88                 }
89             }
90         }
91     }
92
93     NInterval currInterval; // the current interval
94     int psi; // the current interval's first start position
95     ArrayList<NInterval> tmp;
96
97     // Linear allocation begins; repeat so long as there are
98     // additional virtual registers to map to physical registers.
99     while (!unhandled.isEmpty()) {
100         currInterval = unhandled.remove(0);
101         psi = currInterval.firstRangeStart();
102         tmp = new ArrayList<NInterval>();
103         for (int i = 0; i < active.size(); i++) {
104             if (active.get(i).lastNRangeStop() < psi) {
105                 tmp.add(active.get(i));
106             } else if (!active.get(i).isLiveAt(psi)) {
107                 inactive.add(active.get(i));
108                 tmp.add(active.get(i));
109             }
110         }
111         for (NInterval nonActive : tmp) {
112             active.remove(nonActive);
113         }
114         tmp = new ArrayList<NInterval>();
115         for (int i = 0; i < inactive.size(); i++) {
116             if (inactive.get(i).lastNRangeStop() < psi) {
117                 tmp.add(inactive.get(i));
118             } else if (inactive.get(i).isLiveAt(psi)) {
119                 active.add(inactive.get(i));
120                 tmp.add(inactive.get(i));
121             }
122         }
123         for (NInterval nonInActive : tmp) {
124             inactive.remove(nonInActive);
125         }
126         if (!this.foundFreeRegFor(currInterval)) { // check
127             this.allocateBlockedRegFor(currInterval); // never fails
128         }
129         active.add(currInterval);
130     }
131     this.resolveDataFlow();
132 }
133
134 /**

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

135     * Adds a given interval onto the unhandled list, maintaining an order based
136     * on the first range start of the NIntervals.
137     *
138     * @param newInterval
139     *         the NInterval to sort onto unhandled.
140     */
141
142     private void addSortedToUnhandled(NInterval newInterval) {
143         if (unhandled.isEmpty()) {
144             unhandled.add(newInterval);
145         } else {
146             int i = 0;
147             while (i < unhandled.size()
148                 && unhandled.get(i).firstRangeStart() <= newInterval
149                     .firstRangeStart()) {
150                 i++;
151             }
152             unhandled.add(i, newInterval);
153         }
154     }
155
156     /**
157     * Allocates a free physical register for the current interval. Inspects
158     * active and inactive sets. Cannot split or alter the assigned physical
159     * register of any other interval but current.
160     *
161     * @param currInterval
162     *         the current interval for which a physical register is sought.
163     * @return true if a free physical register was found and allocated for
164     *         current interval, false otherwise
165     */
166
167     private boolean foundFreeRegFor(NInterval currInterval) {
168         this.initFreePositions(); // must be reset every iteration
169         for (NInterval activeInterval : active) {
170             if (activeInterval.pRegister != null)
171                 freePos[activeInterval.pRegister.number - NPhysicalRegister.T0] =
172                 0;
173         }
174         for (NInterval inactiveInterval : inactive) {
175             if (inactiveInterval.nextIntersection(currInterval) >= 0)
176                 freePos[inactiveInterval.pRegister.number
177                     - NPhysicalRegister.T0] = Math.min(
178                     freePos[inactiveInterval.pRegister.number
179                         - NPhysicalRegister.T0], inactiveInterval
180                         .nextIntersection(currInterval));
181         }
182
183         // The physical registers available are in NPhysicalRegister.getInfo
184         // static array. This is indexed from 0 to NPhysicalRegister.MAX_COUNT
185         int reg = this.getBestFreeReg();
186         if (freePos[reg] == 0)
187             return false;
188         else if (freePos[reg] > currInterval.lastNRangeStop()) {
189             currInterval.pRegister = NPhysicalRegister.regInfo[reg
190                 + NPhysicalRegister.T0];
191             cfg.pRegisters.add(NPhysicalRegister.regInfo[reg
192                 + NPhysicalRegister.T0]);
193             regIntervals.get(reg).add(currInterval);
194             return true;
195         } else {
196             this.addSortedToUnhandled(currInterval.splitAt(freePos[reg]));
197             currInterval.spill();
198             currInterval.pRegister = NPhysicalRegister.regInfo[reg
199                 + NPhysicalRegister.T0];
200             regIntervals.get(reg).add(currInterval);
201             return true;
202         }
203     }

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

203
204 /**
205  * Sets all free positions of pregisters available for allocation to a
206  * really high number.
207  */
208
209 private void initFreePositions() {
210     for (int i = 0; i < NPhysicalRegister.MAX_COUNT; i++) {
211         freePos[i] = Integer.MAX_VALUE;
212     }
213 }
214
215 /**
216  * The best free physical register.
217  *
218  * @return the register number.
219  */
220
221 private int getBestFreeReg() {
222     int freeRegNumber = 0;
223     for (int i = 0; i < NPhysicalRegister.MAX_COUNT; i++) {
224         if (freePos[i] > freePos[freeRegNumber])
225             freeRegNumber = i;
226     }
227     return freeRegNumber;
228 }
229
230 /**
231  * Allocates a register based on spilling an interval.
232  *
233  * @param currInterval
234  *        the current interval.
235  */
236
237 private void allocateBlockedRegFor(NInterval currInterval) {
238     this.initUseAndBlockPositions(); // must be reset every iteration
239     for (NInterval activeInterval : active) {
240         usePos[activeInterval.pRegister.number - NPhysicalRegister.T0] = Math
241             .min(usePos[activeInterval.pRegister.number
242                 - NPhysicalRegister.T0], activeInterval
243                 .nextUsageOverlapping(currInterval));
244     }
245     for (NInterval inactiveInterval : inactive) {
246         if (inactiveInterval.nextIntersection(currInterval) >= 0)
247             usePos[inactiveInterval.pRegister.number - NPhysicalRegister.T0]
248                 = Math
249                     .min(usePos[inactiveInterval.pRegister.number
250                         - NPhysicalRegister.T0], inactiveInterval
251                         .nextUsageOverlapping(currInterval));
252     }
253     int reg = this.getBestBlockedReg(); // this is just an index in the
254     // usePos array
255     if (usePos[reg] < currInterval.firstUsage()) {
256         // best to spill current - no reg assignment.
257         this.addSortedToUnhandled(currInterval.splitAt(currInterval
258             .firstUsage() - 5));
259         currInterval.spill();
260         NInterval splitChild = currInterval.splitAt(currInterval
261             .firstRangeStart());
262         this.addSortedToUnhandled(splitChild);
263         currInterval.spill();
264     } else {
265         // spilling frees reg for all of current
266         currInterval.pRegister = NPhysicalRegister.regInfo[reg
267             + NPhysicalRegister.T0];
268         for (NInterval i : regIntervals.get(reg)) {
269             if (currInterval.nextIntersection(i) >= 0) {
270                 NInterval splitChild = i.splitAt(currInterval
271                     .firstRangeStart());

```

```

271         this.addSortedToUnhandled(splitChild);
272         i.spill();
273     }
274 }
275 regIntervals.get(reg).add(currInterval);
276 }
277 }
278
279 /**
280  * Initialize use and block positions before processing each virtual
281  * register.
282  */
283
284 private void initUseAndBlockPositions() {
285     for (int i = 0; i < NPhysicalRegister.MAX_COUNT; i++) {
286         usePos[i] = Integer.MAX_VALUE;
287         blockPos[i] = Integer.MAX_VALUE;
288     }
289 }
290
291 /**
292  * Get the best blocked physical register.
293  *
294  * @return the register number.
295  */
296
297 private int getBestBlockedReg() {
298     int usableRegNumber = 0;
299     for (int i = 0; i < NPhysicalRegister.MAX_COUNT; i++) {
300         if (usePos[i] < usePos[usableRegNumber]) {
301             usableRegNumber = i;
302         }
303     }
304     return usableRegNumber;
305 }
306
307 /**
308  * Resolve the data flow after allocating registers, inserting additional
309  * saves and restores for registers to maintain consistency.
310  */
311 private void resolveDataFlow() {
312     // local data flow construction
313     // Devise an alternate way of doing this, perhaps with more
314     // clarity, will implement later, but has same effect.
315     for (NInterval i : cfg.intervals) {
316         if (cfg.registers.get(i.vRegId) != null) {
317             if (i.spill) {
318                 for (int c = 0; c < i.children.size(); c++) {
319                     if (i.endsAtBlock() == i.children.get(c)
320                         .startsAtBlock()) {
321                         if (c == 0) {
322                             addStoreInstruction(i, i.lastNRangeStop());
323                             addLoadInstruction(i.children.get(c),
324                                 i.children.get(c).firstRangeStart());
325                         } else {
326                             addStoreInstruction(i.children.get(c - 1),
327                                 i.children.get(c - 1).lastNRangeStop());
328                             addLoadInstruction(i.children.get(c),
329                                 i.children.get(c).firstRangeStart());
330                         }
331                     }
332                 }
333             }
334         }
335     }
336
337     // resolution of global data flow
338     for (NBasicBlock b : cfg.basicBlocks) {
339         for (NBasicBlock s : b.successors) {

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

340         for (int i = s.liveIn.nextSetBit(0); i >= 0; i = s.liveIn
341             .nextSetBit(i + 1)) {
342             NInterval parent = cfg.intervals.get(i);
343             NInterval from = parent.childAtOrEndingBefore(b);
344             NInterval to = parent.childAtOrStartingAfter(s);
345             if (!from.equals(to)) {
346                 addStoreInstruction(from, from.usePositions.floorKey(b
347                     .getLastLIRInstId()));
348                 to = getSegmentWithNearestUse(to, s.getFirstLIRInstId());
349                 if (to.usePositions.ceilingEntry(s.getFirstLIRInstId())
350                     .getValue() == InstructionType.read)
351                     // no use loading prior to a write.
352                     addLoadInstruction(to, to.usePositions.ceilingKey(s
353                         .getFirstLIRInstId()));
354             }
355         }
356     }
357 }
358
359 /**
360  * Get the the interval segment that contains the nearest first use.
361  *
362  * @param i
363  *     the interval segment (could be a parent or child).
364  * @param id
365  *     the lir id after which a use is sought.
366  * @return the interval segment that that contains the first use at or after
367  *     the id position and is associated with the interval i through a
368  *     sibling or child relationship. Returns i if there is a use after
369  *     id within i. Null if no interval exists that is related to i and
370  *     contains a use position at or after id.
371  */
372
373 private NInterval getSegmentWithNearestUse(NInterval i, int id) {
374     if (i.usePositions.ceilingEntry(id) != null)
375         return i;
376     else {
377         NInterval parent = i;
378         int idx = 0;
379         if (i.isChild()) {
380             parent = i.parent;
381             idx = parent.children.indexOf(i) + 1;
382         }
383         for (; idx < parent.children.size(); idx++) {
384             if (parent.children.get(idx).usePositions.ceilingEntry(id) !=
385                 null)
386                 return parent.children.get(idx);
387         }
388         return null;
389     }
390 }
391
392 /**
393  * Adds a store instruction right after a use position specified by id.
394  *
395  * @param from
396  *     the interval which this use position is a part of.
397  * @param id
398  *     the id of the use position.
399  */
400
401 private void addStoreInstruction(NInterval from, int id) {
402     NBasicBlock b = cfg.blockAt(id);
403     id++;
404     if (b.idIsFree(id)) { // assumes always same instr
405         b.insertLIRInst(new NLIRStore(b, id, from.offset, from.offsetFrom,
406             from.pRegister));
407     }

```

```

408     }
409
410     /**
411     * Adds a store instruction right before a use position specified by id.
412     *
413     * @param to
414     *       the interval which this use position is a part of.
415     * @param id
416     *       the id of the use position.
417     */
418
419     private void addLoadInstruction(NInterval to, int id) {
420         NBasicBlock s = cfg.blockAt(id);
421         id--;
422         if (s.idIsFree(id)) { // assumes always same instr
423             s.insertLIRInst(new NLIRLoad(s, id, to.offset, to.offsetFrom,
424                 to.pRegister));
425         }
426     }
427
428 }

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder