

NHIRInstruction.java

```
1  // Copyright 2013 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas
2
3  package jminusminus;
4
5  import static jminusminus.CLConstants.*;
6  import static jminusminus.NPhysicalRegister.*;
7  import java.util.ArrayList;
8
9  /**
10   * High-level intermediate representation (HIR) of a JVM instruction.
11   */
12
13  abstract class NHIRInstruction {
14
15      /**
16       * Maps JVM opcode to a string mnemonic for HIR instructions. For example,
17       * the opcode imul is mapped to the string "*".
18       */
19      protected static String[] hirMnemonic;
20      static {
21          hirMnemonic = new String[256];
22          hirMnemonic[IADD] = "+";
23          hirMnemonic[ISUB] = "-";
24          hirMnemonic[IMUL] = "*";
25          hirMnemonic[MULTIANEWARRAY] = "multianewarray";
26          hirMnemonic[AALOAD] = "aaload";
27          hirMnemonic[IALOAD] = "iaload";
28          hirMnemonic[IASTORE] = "iastore";
29          hirMnemonic[IF_ICMPNE] = "!=";
30          hirMnemonic[IF_ICMPGT] = ">";
31          hirMnemonic[IF_ICMPLE] = "<=";
32          hirMnemonic[GETSTATIC] = "getstatic";
33          hirMnemonic[PUTSTATIC] = "putstatic";
34          hirMnemonic[INVOKESPECIAL] = "invokespecial";
35          hirMnemonic[INVOKESTATIC] = "invokestatic";
36          hirMnemonic[ARETURN] = "areturn";
37          hirMnemonic[RETURN] = "return";
38          hirMnemonic[IRETURN] = "ireturn";
39      }
40
41      /** The block containing this instruction. */
42      public NBasicBlock block;
43
44      /** Unique identifier of this instruction. */
45      public int id;
46
47      /** Short type name for this instruction. */
48      public String sType;
49
50      /** Long type name for this instruction. */
51      public String lType;
52
53      /** The LIR instruction corresponding to this HIR instruction. */
54      public NLIRInstruction lir;
55
56      /**
57       * Construct an NHIRInstruction object.
58       *
59       * @param block enclosing block.
60       * @param id identifier of the instruction.
61       */
62
63
64      protected NHIRInstruction(NBasicBlock block, int id) {
65          this(block, id, "", "");
66      }
```

```

67     }
68
69     /**
70     * Construct an NHIRInstruction object.
71     *
72     * @param block
73     *         enclosing block.
74     * @param id
75     *         identifier of the instruction.
76     * @param sType
77     *         short type name of the instruction.
78     * @param lType
79     *         long type name of the instruction.
80     */
81
82     protected NHIRInstruction(NBasicBlock block, int id, String sType,
83                               String lType) {
84         this.block = block;
85         this.id = id;
86         this.sType = sType;
87         this.lType = lType;
88     }
89
90     /**
91     * Return true if this instruction is the same as the other, false
92     * otherwise. Two instructions are the same if their ids are the same.
93     *
94     * @param other
95     *         the instruction to compare to.
96     * @return true if the instructions are the same, false otherwise.
97     */
98
99     public boolean equals(NHIRInstruction other) {
100         return this.id == other.id;
101     }
102
103     /**
104     * Convert and return a low level representation (LIR) of this HIR
105     * instruction. Also adds the returned LIR instruction to the list of LIR
106     * instructions for the block containing this instruction, along with any
107     * other intermediate LIR instructions needed.
108     *
109     * @return LIR instruction corresponding to this HIR instruction.
110     */
111
112     public NLIRInstruction toLir() {
113         return null;
114     }
115
116     /**
117     * Return the identifier of this instruction with the short type name
118     * prefixed.
119     *
120     * @return identifier of this IR instruction with the short type name
121     *         prefixed.
122     */
123
124     public String id() {
125         return sType + id;
126     }
127
128     /**
129     * Return a string representation of this instruction.
130     *
131     * @return string representation of this instruction.
132     */
133
134     public String toString() {
135         return sType + id;

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

136     }
137 }
138 }
139
140 /**
141  * HIR instruction corresponding to the JVM arithmetic instructions.
142  */
143
144 class NHIRArithmetic extends NHIRInstruction {
145
146     /** Opcode for the arithmetic operator. */
147     public int opcode;
148
149     /** Lhs HIR id. */
150     public int lhs;
151
152     /** Rhs HIR id. */
153     public int rhs;
154
155     /**
156      * Construct an NHIRArithmetic instruction.
157      *
158      * @param block
159      *     enclosing block.
160      * @param id
161      *     identifier of the instruction.
162      * @param opcode
163      *     opcode for the arithmetic operator.
164      * @param lhs
165      *     lhs HIR id.
166      * @param rhs
167      *     rhs HIR id.
168      */
169
170     public NHIRArithmetic(NBasicBlock block, int id, int opcode, int lhs,
171         int rhs) {
172         super(block, id, "I", "I");
173         this.opcode = opcode;
174         this.lhs = lhs;
175         this.rhs = rhs;
176     }
177
178     /**
179      * @inheritDoc
180      */
181
182     public NLIRInstruction toLir() {
183         if (lir != null) {
184             return lir;
185         }
186         NLIRInstruction ins1 = block.cfg.hirMap.get(lhs).toLir();
187         NLIRInstruction ins2 = block.cfg.hirMap.get(rhs).toLir();
188         lir = new NLIRArithmetic(block, NControlFlowGraph.lirId++, opcode,
189             ins1, ins2);
190         block.lir.add(lir);
191         return lir;
192     }
193
194     /**
195      * @inheritDoc
196      */
197
198     public String toString() {
199         return id() + ": " + block.cfg.hirMap.get(lhs).id() + " "
200             + hirMnemonic[opcode] + " " + block.cfg.hirMap.get(rhs).id();
201     }
202 }
203 }
204

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

205 /**
206  * HIR instruction corresponding to the JVM instructions representing integer
207  * constants.
208  */
209
210 class NHIRIntConstant extends NHIRInstruction {
211
212     /** The constant int value. */
213     public int value;
214
215     /**
216      * Construct an NHIRIntConstant instruction.
217      *
218      * @param block
219      *         enclosing block.
220      * @param id
221      *         identifier of the instruction.
222      * @param value
223      *         the constant int value.
224      */
225
226     public NHIRIntConstant(NBasicBlock block, int id, int value) {
227         super(block, id, "I", "I");
228         this.value = value;
229     }
230
231     /**
232      * @inheritDoc
233      */
234     public NLIRInstruction toLir() {
235         if (lir != null) {
236             return lir;
237         }
238         lir = new NHIRIntConstant(block, mControlFlowGraph.lirId++, value);
239         block.lir.add(lir);
240         return lir;
241     }
242
243     /**
244      * @inheritDoc
245      */
246
247     public String toString() {
248         return id() + ": " + value;
249     }
250 }
251
252 }
253
254 /**
255  * HIR instruction corresponding to the JVM instructions representing string
256  * constants.
257  */
258
259 class NHIRStringConstant extends NHIRInstruction {
260
261     /** The constant string value. */
262     public String value;
263
264     /**
265      * Construct an NHIRStringConstant instruction.
266      *
267      * @param block
268      *         enclosing block.
269      * @param id
270      *         identifier for the instruction.
271      * @param value
272      *         the constant string value.
273      */

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

274
275 public NHIRStringConstant(NBasicBlock block, int id, String value) {
276     super(block, id, "L", "Ljava/lang/String;");
277     this.value = value;
278 }
279
280 /**
281  * @inheritDoc
282  */
283
284 public NLIRInstruction toLir() {
285     if (lir != null) {
286         return lir;
287     }
288     lir = new NHIRStringConstant(block, NControlFlowGraph.lirId++, value);
289     block.lir.add(lir);
290     return lir;
291 }
292
293 /**
294  * @inheritDoc
295  */
296
297 public String toString() {
298     return id() + ": " + value;
299 }
300
301 }
302
303 /**
304  * HIR instruction representing an conditional jump instructions in JVM.
305  */
306
307 class NHIRConditionalJump extends NHIRInstruction {
308
309     /** Lhs HIR id. */
310     public int lhs;
311
312     /** Rhs HIR id. */
313     public int rhs;
314
315     /** Test expression opcode. */
316     public int opcode;
317
318     /** Block to jump to on true. */
319     public NBasicBlock onTrueDestination;
320
321     /** Block to jump to on false. */
322     public NBasicBlock onFalseDestination;
323
324     /**
325      * Construct an NHIRConditionalJump instruction.
326      *
327      * @param block
328      *     enclosing block.
329      * @param id
330      *     identifier of the instruction.
331      * @param lhs
332      *     Lhs HIR id.
333      * @param rhs
334      *     Rhs HIR id.
335      * @param opcode
336      *     opcode in the test.
337      * @param onTrueDestination
338      *     block to jump to on true.
339      * @param onFalseDestination
340      *     block to jump to on false.
341      */
342

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

343     public NHIRConditionalJump(NBasicBlock block, int id, int lhs, int rhs,
344                               int opcode, NBasicBlock onTrueDestination,
345                               NBasicBlock onFalseDestination) {
346         super(block, id, "", "");
347         this.lhs = lhs;
348         this.rhs = rhs;
349         this.opcode = opcode;
350         this.onTrueDestination = onTrueDestination;
351         this.onFalseDestination = onFalseDestination;
352     }
353
354     /**
355      * @inheritDoc
356      */
357
358     public NLIRInstruction toLir() {
359         if (lir != null) {
360             return lir;
361         }
362         NLIRInstruction ins1 = block.cfg.hirMap.get(lhs).toLir();
363         NLIRInstruction ins2 = block.cfg.hirMap.get(rhs).toLir();
364         lir = new NHIRConditionalJump(block, NControlFlowGraph.lirId++, ins1,
365                                     ins2, opcode, onTrueDestination, onFalseDestination);
366         block.lir.add(lir);
367         return lir;
368     }
369
370     /**
371      * @inheritDoc
372      */
373
374     public String toString() {
375         return id() + ": if " + block.cfg.hirMap.get(lhs).id() + " "
376                + hirMnemonic[opcode] + " " + block.cfg.hirMap.get(rhs).id()
377                + " then " + onTrueDestination.id() + " else "
378                + onFalseDestination.id();
379     }
380 }
381
382 /**
383  * HIR instruction representing an unconditional jump instruction in JVM.
384  */
385
386
387 class NHIRGoto extends NHIRInstruction {
388
389     /** The destination block to unconditionally jump to. */
390     public NBasicBlock destination;
391
392     /**
393      * Construct an NHIRGoto instruction.
394      *
395      * @param block
396      *         enclosing block.
397      * @param id
398      *         identifier of the instruction.
399      * @param destination
400      *         the block to jump to.
401      */
402
403     public NHIRGoto(NBasicBlock block, int id, NBasicBlock destination) {
404         super(block, id, "", "");
405         this.destination = destination;
406     }
407
408     /**
409      * @inheritDoc
410      */
411

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

412     public NLIRInstruction toLir() {
413         if (lir != null) {
414             return lir;
415         }
416         lir = new NLIRGoto(block, NControlFlowGraph.lirId++, destination);
417         block.lir.add(lir);
418         return lir;
419     }
420
421     /**
422      * @inheritDoc
423      */
424
425     public String toString() {
426         return id() + ": goto " + destination.id();
427     }
428
429 }
430
431 /**
432  * HIR instruction representing method invocation instructions in JVM.
433  */
434
435 class NHIRInvoke extends NHIRInstruction {
436
437     /** Opcode of the JVM instruction. */
438     public int opcode;
439
440     /** Target for the method. */
441     public String target;
442
443     /** Name of the method being invoked. */
444     public String name;
445
446     /** List of HIR ids of arguments for the method. */
447     public ArrayList<Integer> arguments;
448
449     /**
450      * Construct an NHIRInvoke instruction
451      *
452      * @param block
453      *     enclosing block.
454      * @param id
455      *     identifier of the instruction.
456      * @param opcode
457      *     opcode of the JVM instruction.
458      * @param target
459      *     target of the method.
460      * @param name
461      *     name of the method.
462      * @param arguments
463      *     list of HIR ids of arguments for the method.
464      * @param sType
465      *     return type (short name) of the method.
466      * @param lType
467      *     return type (long name) of the method.
468      */
469
470     public NHIRInvoke(NBasicBlock block, int id, int opcode, String target,
471         String name, ArrayList<Integer> arguments, String sType,
472         String lType) {
473         super(block, id, sType, lType);
474         this.opcode = opcode;
475         this.target = target;
476         this.name = name;
477         this.arguments = arguments;
478     }
479
480     /**

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

481     * @inheritDoc
482     */
483
484     public NLIRInstruction toLir() {
485         if (lir != null) {
486             return lir;
487         }
488
489         // First four arguments are stored in physical registers
490         // (a0, ..., a3) and the rest are on the stack.
491         // Allocate space on stack for arguments fourth or
492         // above; [0, block.cfg.offset - 1].
493         if (this.arguments.size() - 4 > block.cfg.offset) {
494             block.cfg.offset = this.arguments.size() - 4;
495         }
496
497         ArrayList<NRegister> arguments = new ArrayList<NRegister>();
498         ArrayList<NPhysicalRegister> froms = new ArrayList<NPhysicalRegister>();
499         ArrayList<NVirtualRegister> tos = new ArrayList<NVirtualRegister>();
500         for (int i = 0; i < this.arguments.size(); i++) {
501             int arg = this.arguments.get(i);
502             NLIRInstruction ins = block.cfg.hirMap.get(arg).toLir();
503             if (i < 4) {
504                 // Generate an LIR move instruction (move1) to save
505                 // away the physical register a0 + i into a virtual
506                 // register, and another LIR move instruction (move2)
507                 // to copy the argument from the virtual register
508                 // it's in to the physical register a0 + i.
509                 String sType = block.cfg.hirMap.get(arg).sType;
510                 String lType = block.cfg.hirMap.get(arg).lType;
511                 NPhysicalRegister from = NPhysicalRegister.regInfo[A0 + i];
512                 block.cfg.registers.set(A0 + i, from);
513                 NVirtualRegister to = new NVirtualRegister(
514                     NControlFlowGraph.regId++, sType, lType);
515                 block.cfg.registers.add(to);
516                 NLIRMove move1 = new NLIRMove(block, NControlFlowGraph.lirId++,
517                     from, to);
518                 block.lir.add(move1);
519                 NLIRMove move2 = new NLIRMove(block, NControlFlowGraph.lirId++,
520                     ins.write, from);
521                 block.lir.add(move2);
522                 arguments.add(NPhysicalRegister.regInfo[A0 + i]);
523
524                 // Remember the froms and the tos so we can restore
525                 // the
526                 // values of a0 + i registers.
527                 froms.add(from);
528                 tos.add(to);
529             } else {
530                 NLIRStore store = new NLIRStore(block,
531                     NControlFlowGraph.lirId++, i - 4, OffsetFrom.SP,
532                     ins.write);
533                 block.lir.add(store);
534                 arguments.add(ins.write);
535             }
536         }
537
538         lir = new NLIRInvoke(block, NControlFlowGraph.lirId++, opcode, target,
539             name, arguments, sType, lType);
540         block.lir.add(lir);
541
542         // If the function returns a value, generate an LIR move
543         // instruction to save away the value in the physical
544         // register v0 into a virtual register.
545         if (lir.write != null) {
546             NVirtualRegister to = new NVirtualRegister(
547                 NControlFlowGraph.regId++, sType, lType);
548             NLIRMove move = new NLIRMove(block, NControlFlowGraph.lirId++,
549                 NPhysicalRegister.regInfo[V0], to);

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder


```

550         block.cfg.registers.add(to);
551         block.lir.add(move);
552         lir = move;
553     }
554
555     // Generate LIR move instructions to restore the a0, ..., a3
556     // instructions.
557     for (int i = 0; i < tos.size(); i++) {
558         NLIRMove move = new NLIRMove(block, NControlFlowGraph.lirId++, tos
559             .get(i), froms.get(i));
560         block.lir.add(move);
561     }
562
563     return lir;
564 }
565
566 /**
567  * @inheritDoc
568  */
569
570 public String toString() {
571     String s = id() + ": " + hirMnemonic[opcode] + " " + target + "."
572         + name + "(";
573     for (int arg : arguments) {
574         s += block.cfg.hirMap.get(arg).id() + " ";
575     }
576     s += ")";
577     return s;
578 }
579 }
580
581 /**
582  * HIR instruction representing a JVM return instruction.
583  */
584
585
586 class NHIRReturn extends NHIRInstruction {
587
588     /** JVM opcode for the return instruction. */
589     public int opcode;
590
591     /** Return value HIR id. */
592     public int value;
593
594     /**
595      * Construct an NHIRReturn instruction.
596      *
597      * @param block enclosing block.
598      * @param id identifier of the instruction.
599      * @param opcode JVM opcode for the return instruction.
600      * @param value return value HIR id.
601      */
602
603     public NHIRReturn(NBasicBlock block, int id, int opcode, int value) {
604         super(block, id,
605             (value == -1) ? "" : block.cfg.hirMap.get(value).sType,
606             (value == -1) ? "" : block.cfg.hirMap.get(value).lType);
607         this.opcode = opcode;
608         this.value = value;
609     }
610
611     /**
612      * @inheritDoc
613      */
614 }

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

619     public NLIRInstruction toLir() {
620         if (lir != null) {
621             return lir;
622         }
623         NLIRInstruction result = null;
624         if (value != -1) {
625             result = block.cfg.hirMap.get(value).toLir();
626             NLIRMove move = new NLIRMove(block, NControlFlowGraph.lirId++,
627                 result.write, NPhysicalRegister.regInfo[V0]);
628             block.lir.add(move);
629             block.cfg.registers.set(V0, NPhysicalRegister.regInfo[V0]);
630         }
631         lir = new NLIRReturn(block, NControlFlowGraph.lirId++, opcode,
632             (result == null) ? null : NPhysicalRegister.regInfo[V0]);
633         block.lir.add(lir);
634         return lir;
635     }
636
637     /**
638      * @inheritDoc
639      */
640
641     public String toString() {
642         if (value == -1) {
643             return id() + ": " + hirMnemonic[opcode];
644         }
645         return id() + ": " + hirMnemonic[opcode] + " "
646             + block.cfg.hirMap.get(value).id();
647     }
648 }
649 }
650
651 /**
652  * HIR instruction representing JVM (put) field instructions.
653  */
654
655 class NHIRPutField extends NHIRInstruction {
656
657     /** Opcode of the JVM instruction. */
658     public int opcode;
659
660     /** Target for the field. */
661     public String target;
662
663     /** Name of the field being accessed. */
664     public String name;
665
666     /** HIR id of the value of the field. */
667     public int value;
668
669     /**
670      * Construct an NHIRPutField instruction.
671      *
672      * @param block
673      *     enclosing block.
674      * @param id
675      *     identifier of the instruction.
676      * @param opcode
677      *     JVM opcode for the instruction.
678      * @param target
679      *     target for the field.
680      * @param name
681      *     name of the field.
682      * @param sType
683      *     type (short name) of the field.
684      * @param lType
685      *     type (long name) of the field.
686      * @param value
687      *     HIR id of the value of the field.

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

688     */
689
690     public NHIRPutField(NBasicBlock block, int id, int opcode, String target,
691         String name, String sType, String lType, int value) {
692         super(block, id, sType, lType);
693         this.opcode = opcode;
694         this.target = target;
695         this.name = name;
696         this.value = value;
697     }
698
699     /**
700     * @inheritDoc
701     */
702
703     public NLIRInstruction toLir() {
704         if (lir != null) {
705             return lir;
706         }
707         NLIRInstruction result = block.cfg.hirMap.get(value).toLir();
708         lir = new NLIRPutField(block, NControlFlowGraph.lirId++, opcode,
709             target, name, sType, lType, result);
710         block.lir.add(lir);
711         return lir;
712     }
713
714     /**
715     * @inheritDoc
716     */
717
718     public String toString() {
719         return id() + ": " + hirMnemonic[opcode] + " " + target + "." + name
720             + " = " + block.cfg.hirMap.get(value).id();
721     }
722 }
723
724
725 /**
726  * HIR instruction representing the JVM (get field) instructions.
727  */
728
729 class NHIRGetField extends NHIRInstruction {
730
731     /** Opcode of the JVM instruction. */
732     public int opcode;
733
734     /** Target for the field. */
735     public String target;
736
737     /** Name of the field being accessed. */
738     public String name;
739
740     /**
741     * Construct an NHIRGetField instruction.
742     *
743     * @param block
744     *     enclosing block.
745     * @param id
746     *     identifier of the instruction.
747     * @param opcode
748     *     JVM opcode for the instruction.
749     * @param target
750     *     target for the field.
751     * @param name
752     *     name of the field.
753     * @param sType
754     *     type (short name) of the field.
755     * @param lType
756     *     type (long name) of the field.

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

757     */
758
759     public NHIRGetField(NBasicBlock block, int id, int opcode, String target,
760         String name, String sType, String lType) {
761         super(block, id, sType, lType);
762         this.opcode = opcode;
763         this.target = target;
764         this.name = name;
765     }
766
767     /**
768     * @inheritDoc
769     */
770
771     public NLIRInstruction toLir() {
772         if (lir != null) {
773             return lir;
774         }
775         lir = new NLRGetField(block, NControlFlowGraph.lirId++, opcode,
776             target, name, sType, lType);
777         block.lir.add(lir);
778         return lir;
779     }
780
781     /**
782     * @inheritDoc
783     */
784
785     public String toString() {
786         return "NLRGetField[" + hIRMemoryInfo(opcode) + " " + target + "]." + name;
787     }
788 }
789
790 /**
791 * HIR instruction representing JVM array creation instructions.
792 */
793
794
795 class NHIRNewArray extends NLRInstruction {
796
797     /** Opcode of the JVM instruction. */
798     public int opcode;
799
800     /** Dimension of the array. */
801     public int dim;
802
803     /**
804     * Construct an NHIRNewArray instruction.
805     *
806     * @param block
807     *     enclosing block.
808     * @param id
809     *     identifier of the instruction.
810     * @param opcode
811     *     JVM opcode for the instruction.
812     * @param dim
813     *     dimension of the array.
814     * @param sType
815     *     type (short name) of the array.
816     * @param lType
817     *     type (long name) of the array.
818     */
819
820     public NHIRNewArray(NBasicBlock block, int id, int opcode, int dim,
821         String sType, String lType) {
822         super(block, id, lType, sType);
823         this.opcode = opcode;
824         this.dim = dim;
825     }

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

826
827 /**
828  * @inheritDoc
829  */
830
831 public NLIRInstruction toLir() {
832     if (lir != null) {
833         return lir;
834     }
835     lir = new NLIRNewArray(block, NControlFlowGraph.lirId++, opcode, dim,
836         sType, lType);
837     block.lir.add(lir);
838     return lir;
839 }
840
841 /**
842  * @inheritDoc
843  */
844
845 public String toString() {
846     return id() + ": " + hirMnemonic[opcode] + " " + lType + " [" + dim
847         + "]";
848 }
849
850 }
851
852 /**
853  * HIR instruction representing JVM array load instructions.
854  */
855
856 class NHIRLoad extends NHIRInstruction {
857
858     /** Opcode of the JVM instruction. */
859     public int opcode;
860
861     /** HIR id of the array reference. */
862     public int arrayRef;
863
864     /** HIR id of the array index. */
865     public int index;
866
867     /**
868      * Construct an NHIRLoad instruction.
869      *
870      * @param block
871      *     enclosing block.
872      * @param id
873      *     identifier of the instruction.
874      * @param opcode
875      *     JVM opcode for the instruction.
876      * @param arrayRef
877      *     HIR id of the array reference.
878      * @param index
879      *     HIR id of the the array index.
880      * @param sType
881      *     type (short name) of the array.
882      * @param lType
883      *     type (long name) of the array.
884      */
885
886     public NHIRLoad(NBasicBlock block, int id, int opcode, int arrayRef,
887         int index, String sType, String lType) {
888         super(block, id, sType, lType);
889         this.opcode = opcode;
890         this.arrayRef = arrayRef;
891         this.index = index;
892     }
893
894     /**

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

895     * @inheritDoc
896     */
897
898     public NLIRInstruction toLir() {
899         if (lir != null) {
900             return lir;
901         }
902         NLIRInstruction arrayRef = block.cfg.hirMap.get(this.arrayRef).toLir();
903         NLIRInstruction index = block.cfg.hirMap.get(this.index).toLir();
904         lir = new NLIRALoad(block, NControlFlowGraph.lirId++, opcode, arrayRef,
905             index, sType, lType);
906         block.lir.add(lir);
907         return lir;
908     }
909
910     /**
911     * @inheritDoc
912     */
913
914     public String toString() {
915         return id() + ": " + hirMnemonic[opcode] + " "
916             + block.cfg.hirMap.get(arrayRef).id() + "["
917             + block.cfg.hirMap.get(index).id() + "];"
918     }
919 }
920 }
921
922 /**
923  * HIR instruction representing JVM array store instructions.
924  */
925
926 class NHIRASTore extends NHIRInstruction {
927
928     /** Opcode of the JVM instruction. */
929     public int opcode;
930
931     /** HIR id of the array reference. */
932     public int arrayRef;
933
934     /** HIR id of the array index. */
935     public int index;
936
937     /** HIR id of the value to store. */
938     public int value;
939
940     /**
941     * Construct an NHIRASTore instruction.
942     *
943     * @param block
944     *     enclosing block.
945     * @param id
946     *     identifier of the instruction.
947     * @param opcode
948     *     JVM opcode for the instruction.
949     * @param arrayRef
950     *     HIR id of the array reference.
951     * @param index
952     *     HIR id of the array index.
953     * @param value
954     *     HIR id of the value to store.
955     * @param sType
956     *     type (short name) of the array.
957     * @param lType
958     *     type (long name) of the array.
959     */
960
961     public NHIRASTore(NBasicBlock block, int id, int opcode, int arrayRef,
962         int index, int value, String sType, String lType) {
963         super(block, id, sType, lType);

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

964     this.opcode = opcode;
965     this.arrayRef = arrayRef;
966     this.index = index;
967     this.value = value;
968 }
969
970 /**
971  * @inheritDoc
972  */
973
974 public NLIRInstruction toLir() {
975     if (lir != null) {
976         return lir;
977     }
978     NLIRInstruction arrayRef = block.cfg.hirMap.get(this.arrayRef).toLir();
979     NLIRInstruction index = block.cfg.hirMap.get(this.index).toLir();
980     NLIRInstruction value = block.cfg.hirMap.get(this.value).toLir();
981     lir = new NLIRASTore(block, NControlFlowGraph.lirId++, opcode,
982         arrayRef, index, value, sType, lType);
983     block.lir.add(lir);
984     return lir;
985 }
986
987 /**
988  * @inheritDoc
989  */
990
991 public String toString() {
992     return id() + ": " + hirMnemonic[opcode] + " "
993         + block.cfg.hirMap.get(arrayRef).id() + " "
994         + block.cfg.hirMap.get(index).id() + " = "
995         + block.cfg.hirMap.get(value).id();
996 }
997 }
998
999 /**
1000  * HIR instruction representing phi functions.
1001  */
1002
1003 class NHIRPhiFunction extends NHIRInstruction {
1004
1005     /** List of HIR ids of arguments for the phi function. */
1006     public ArrayList<Integer> arguments;
1007
1008     /** Local variable index. */
1009     public int local;
1010
1011     /**
1012      * Construct an NHIRPhiFunction instruction.
1013      *
1014      * @param block
1015      *     enclosing block.
1016      * @param id
1017      *     identifier of the instruction.
1018      * @param arguments
1019      *     list of HIR ids of arguments for the phi function.
1020      * @param local
1021      *     local variable index.
1022      */
1023
1024     public NHIRPhiFunction(NBasicBlock block, int id,
1025         ArrayList<Integer> arguments, int local) {
1026         super(block, id, "", "");
1027         this.arguments = arguments;
1028         this.local = local;
1029     }
1030 }
1031
1032 /**

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

1033     * Infer type for this phi function. It is essentially the type of the
1034     * arguments.
1035     */
1036
1037     public void inferType() {
1038         for (int arg : arguments) {
1039             if (!block.cfg.hirMap.get(arguments.get(0)).sType.equals("")) {
1040                 sType = block.cfg.hirMap.get(arguments.get(0)).sType;
1041                 lType = block.cfg.hirMap.get(arguments.get(0)).lType;
1042                 break;
1043             }
1044         }
1045     }
1046
1047     /**
1048     * @inheritDoc
1049     */
1050
1051     public NLIRInstruction toLir() {
1052         if (lir != null) {
1053             return lir;
1054         }
1055         lir = new NLIRPhiFunction(block, NControlFlowGraph.lirId++, sType,
1056             lType);
1057         return lir;
1058     }
1059
1060     /**
1061     * @inheritDoc
1062     */
1063
1064     public String toString() {
1065         String s = "[ ";
1066         for (int ins : arguments) {
1067             if (block.cfg.hirMap.get(ins) != null)
1068                 s += block.cfg.hirMap.get(ins).sType + ins + " ";
1069         }
1070         s += "]";
1071         return s;
1072     }
1073 }
1074 }
1075
1076 /**
1077  * HIR instruction representing a formal parameter.
1078  */
1079
1080 class NHIRLoadLocal extends NHIRInstruction {
1081
1082     /** Local variable index. */
1083     public int local;
1084
1085     /**
1086     * Construct an NHIRLoadLocal instruction.
1087     *
1088     * @param block
1089     *     enclosing block.
1090     * @param id
1091     *     identifier of the instruction.
1092     * @param local
1093     *     local variable index.
1094     * @param sType
1095     *     short type name of the instruction.
1096     * @param lType
1097     *     long type name of the instruction.
1098     */
1099
1100     public NHIRLoadLocal(NBasicBlock block, int id, int local, String sType,
1101         String lType) {

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder


```

1102     super(block, id, sType, lType);
1103     this.local = local;
1104 }
1105
1106 /**
1107  * @inheritDoc
1108  */
1109
1110 public NLIRInstruction toLir() {
1111     if (lir != null) {
1112         return lir;
1113     }
1114     lir = new NLIRLoadLocal(block, NControlFlowGraph.lirId++, local, sType,
1115                             lType);
1116     block.lir.add(lir);
1117     return lir;
1118 }
1119
1120 /**
1121  * @inheritDoc
1122  */
1123
1124 public String toString() {
1125     return id() + ": LDLOC " + local;
1126 }
1127
1128 }
1129
1130 /**
1131  * HIR Instruction representing a local (not formal) variable.
1132  */
1133
1134 class NHIRLocal extends NHIRInstruction {
1135     /** Local variable index */
1136     public int local;
1137
1138     /**
1139      * Construct an NHIRLocal instruction.
1140      *
1141      * @param block
1142      *     enclosing block.
1143      * @param id
1144      *     identifier of the instruction.
1145      * @param local
1146      *     local variable index.
1147      * @param sType
1148      *     short type name of the instruction.
1149      * @param lType
1150      *     long type name of the instruction.
1151      */
1152
1153     public NHIRLocal(NBasicBlock block, int id, int local, String sType,
1154                     String lType) {
1155         super(block, id, sType, lType);
1156         this.local = local;
1157     }
1158
1159     /**
1160      * @inheritDoc
1161      */
1162
1163     public String toString() {
1164         return id() + ": LOC " + lType;
1165     }
1166 }
1167
1168 }
1169

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder