```
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.Collections;
7   import java.util.TreeMap;
8
9   import static jminusminus.NPhysicalRegister.*;
10
11  /**
12   * A lifetime interval, recording the interval of LIR code for which the
13   * corresponding virtual register contains a useful value.
14   */
15
16  class NInterval implements Comparable<NInterval> {
17
18      /** Control flow graph instance. */
19      private NControlFlowGraph cfg;
20
21      /**
22       * The virtual register id corresponding to the index in the array list of
23       * NIntervals used by register allocation
24       */
25      public int vRegId;
26
27      /** All live ranges for this virtual register. */
28      public ArrayList<NRange> ranges;
29
30      /**
31       * All use positions (in LIR) and their types for this virtual register
32       */
33      public TreeMap<Integer, InstructionType> usePositions;
34
35      /**
36       * The NPhysicalRegister assigned to this interval. If an interval ends up
37       * needing more than one physical register it is split.
38       */
39      public NPhysicalRegister pRegister;
40
41      /** Whether or not to spill. */
42      public boolean spill;
43
44      /** From offset. */
45      public OffsetFrom offsetFrom;
46
47      /** Offset. */
48      public int offset;
49
50      /** Parent of this interval. */
51      public NInterval parent;
52
53      /** Children of this interval. */
54      public ArrayList<NInterval> children;
55
56      /**
57       * Construct a NInterval with the given virtual register ID for the given
58       * control flow graph.
59       *
60       * @param virtualRegID
61       *            program counter.
62       * @param cfg
63       *            The control flow graph.
64       */
65
66      public NInterval(int virtualRegID, NControlFlowGraph cfg) {
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
        this.cfg = cfg;
        this.ranges = new ArrayList<NRange>();
        this.usePositions = new TreeMap<Integer, InstructionType>();
        this.vRegId = virtualRegID;
        this.children = new ArrayList<NInterval>();
        this.parent = null;
        spill = false;
        offset = -1;
    }

    /**
     * This second constructor is used in instantiating children of a split
     * interval.
     *
     * @param virtualRegID
     *            program counter.
     * @param cfg
     *            The control flow graph.
     * @param childRanges
     *            The instruction ranges for this child.
     * @param parent
     *            The parent interval.
     */

    public NInterval(int virtualRegID, NControlFlowGraph cfg,
            ArrayList<NRange> childRanges, NInterval parent) {
        this.cfg = cfg;
        this.ranges = childRanges;
        this.usePositions = new TreeMap<Integer, InstructionType>();
        this.vRegId = virtualRegID;
        this.parent = parent;
        this.children = new ArrayList<NInterval>();
        spill = false;
        offset = -1;
    }

    /**
     * Add a new range to the existing ranges.
     *
     * @param newNRange
     *            - the NRange to add
     */

    public void addOrExtendNRange(NRange newNRange) {
        if (!ranges.isEmpty()) {
            if (newNRange.stop + 5 == ranges.get(0).start
                    || newNRange.rangeOverlaps(ranges.get(0))) {
                ranges.get(0).start = newNRange.start;
            } else {
                ranges.add(0, newNRange);
            }
        } else {
            ranges.add(newNRange);
        }
    }

    /**
     * Looks for the very first position where an intersection with another
     * interval occurs.
     *
     * NOTE: A.nextIntersection(B) equals B.nextIntersection(A)
     *
     * @param otherInterval
     *            the interval to compare against for intersection.
     * @return the position where the intersection begins.
     */

    public int nextIntersection(NInterval otherInterval) {
        int a = -1, b = -2;
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
        for (NRange r : this.ranges) {
            if (otherInterval.isLiveAt(r.start)) {
                a = r.start;
                break;
            }
        }
        for (NRange r : otherInterval.ranges) {
            if (this.isLiveAt(r.start)) {
                b = r.start;
                break;
            }
        }
        if (a >= 0 && b >= 0) {
            return a <= b ? a : b;
        } else {
            return a > b ? a : b;
        }
    }

    /**
     * The next use position of this interval after the first range start of the
     * foreign interval. If there is no such use, then the first use position is
     * returned to preserve data flow (in case of loops).
     *
     * @param currInterval
     *            the interval with starting point after which we want to find
     *            the next usage of this one.
     *
     * @return the next use position.
     */

    public int nextUsageOverlapping(NInterval currInterval) {
        int psi = currInterval.firstRangeStart();

        if (usePositions.ceilingKey(psi) != null) {
            return usePositions.ceilingKey(psi);
        } else if (!usePositions.isEmpty()) {
            return usePositions.firstKey();
        } else {
            return Integer.MAX_VALUE;
        }
    }

    /**
     * The first use position in this interval.
     *
     * @return the first use position.
     */

    public int firstUsage() {
        return usePositions.firstKey();
    }

    /**
     * Sets the start value of the very first range. Note: There will always be
     * at least one range before this method is used by buildIntervals.
     *
     * @param newStart
     *            the value to which the first range's start will be set.
     */
    public void newFirstRangeStart(int newStart) {
        // Check
        if (!ranges.isEmpty()) {
            ranges.get(0).start = newStart;
        }
    }

    /**
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
     * Register a use (read or write)>
     *
     * @param index
     *            the site of the use.
     * @param type
     *            the instruction type.
     */

    public void addUsePosition(Integer index, InstructionType type) {
        usePositions.put(index, type);
    }

    /**
     * Check if this vreg is alive at a given index.
     *
     * @param atIndex
     *            the index at which to see if this register is live.
     */

    public boolean isLiveAt(int atIndex) {
        for (NRange r : ranges) {
            if (atIndex >= r.start && atIndex <= r.stop) {
                return true;
            }
        }
        return false;
    }

    /**
     * The range in this interval in which the LIR instruction with the given id
     * is live, or null. This will never return null if called for an interval
     * from the active list after it has been set up by the allocate method.
     *
     * @param id
     *            LIR instruction id
     * @return range in which LIR instruction with given id is live, or null.
     */

    private NRange liveRangeAt(int id) {
        for (NRange r : ranges) {
            if (id >= r.start && id <= r.stop) {
                return r;
            }
        }
        return null;
    }

    /**
     * Write the interval information to STDOUT.
     *
     * @param p
     *            for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        if (cfg.registers.get(vRegId) != null) {
            String s = cfg.registers.get(vRegId).name() + ": ";
            for (NRange r : ranges) {
                s += r.toString() + " ";
            }
            if (pRegister != null) {
                s += "-> " + pRegister.name();
            } else {
                s += "-> None";
            }
            if (spill) {
                if (offsetFrom == OffsetFrom.FP) {
                    s += " [frame:" + offset + "]";
                } else {
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
                    s += " [stack:" + offset + "]";
                }
            }
            p.printf("%s\n", s);
            for (NInterval child : this.children) {
                child.writeToStdOut(p);
            }
        } else if (this.isChild()) {
            String s = "\tv" + this.vRegId + ": ";
            for (NRange r : ranges) {
                s += r.toString() + " ";
            }
            if (pRegister != null) {
                s += "-> " + pRegister.name();
            } else {
                s += "-> None";
            }
            if (offsetFrom == OffsetFrom.FP) {
                s += " [frame:" + offset + "]";
            } else {
                s += " [stack:" + offset + "]";
            }
            p.printf("%s\n", s);
            for (NInterval child : this.children) {
                child.writeToStdOut(p);
            }
        }
    }

    /**
     * The start position for the first range.
     *
     * @return the start position.
     */

    public int firstRangeStart() {
        if (ranges.isEmpty())
            return -1;
        else
            return ranges.get(0).start;
    }

    /**
     * The stop position for the last range.
     *
     * @return the stop position.
     */

    public int lastNRangeStop() {
        if (ranges.isEmpty())
            return -1;
        else
            return ranges.get(ranges.size() - 1).stop;
    }

    /**
     * Compare start positions (for ordering intervals).
     *
     * @param other
     *            interval to compare to.
     *
     * @return ordering value.
     */

    public int compareTo(NInterval other) {
        return this.firstRangeStart() - other.firstRangeStart();
    }

    /**
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
     * Two intervals are equal if they have the same virtual register ID.
     *
     * @param other
     *            the interval we are comparing ourself with.
     * @return true if the two intervals are the same, false otherwise.
     */

    public boolean equals(NInterval other) {
        return (this.vRegId == other.vRegId);
    }

    /**
     * Split the current interval at the given index. Responsible for splitting
     * a range if the index falls on one, moving remaining ranges over to child,
     * and moving appropriate usePositions over to the child.
     *
     * @param idx
     *            the index at which this interval is to be split
     *
     * @return the child interval which is to be sorted onto unhandled. If there
     *         was no child created in the case of a pruning this interval is
     *         returned.
     */

    public NInterval splitAt(int idx) {

        ArrayList<NRange> childsRanges = new ArrayList<NRange>();
        if (this.isLiveAt(idx)) { // means split falls on a range
            // Assumptions: if a range is LIVE on an index, then there
            // exist usePositions at or before the index
            // within this same range.
            NRange liveRange = this.liveRangeAt(idx);
            int splitTo = idx;
            splitTo = usePositions.ceilingKey(idx);
            childsRanges.add(liveRange.splitRange(splitTo, idx - 5));
        }

        // The following two for loops take care of any untouched ranges
        // which start after the split position and must be moved to the
        // child interval.
        for (NRange r : ranges) {
            if (r.start > idx) {
                childsRanges.add(r);
            }
        }
        for (NRange r : childsRanges) {
            ranges.remove(r);
        }

        NInterval child = new NInterval(cfg.maxIntervals++, cfg, childsRanges,
                this.getParent());
        cfg.registers.add(null); // expand size of cfg.registers to
        // avoid null pointer exception when printing.

        // transfer remaining use positions
        while (this.usePositions.ceilingKey(idx) != null)
            child.usePositions
                    .put(this.usePositions.ceilingKey(idx), this.usePositions
                            .remove(this.usePositions.ceilingKey(idx)));
        this.getParent().children.add(child);
        return child;
    }

    /**
     * The parent interval.
     *
     * @return the parent interval.
     */

```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
412     private NInterval getParent() {
413         if (parent != null)
414             return parent;
415         else
416             return this;
417     }
418
419     /**
420      * The child interval at a given instruction index.
421      *
422      * @param idx
423      *            The instruction index.
424      * @return the child interval.
425      */
426
427     public NInterval childAt(int idx) {
428         for (NInterval child : children) {
429             if (child.isLiveAt(idx)) {
430                 return child;
431             }
432         }
433         return this;
434     }
435
436     /**
437      * A child of this interval which is live or ends before the given basic
438      * block's end.
439      *
440      * @param b
441      *            the basic block.
442      * @return the child of this interval which ends at or nearest (before) this
443      *         basic block's end (last lir instruction index).
444      */
445
446     public NInterval childAtOrEndingBefore(NBasicBlock b) {
447         int idx = b.getLastLIRInstId();
448         for (NInterval child : children) {
449             if (child.isLiveAt(idx))
450                 return child;
451         }
452         NInterval tmp = this;
453         int highestEndingAllowed = b.getFirstLIRInstId();
454         for (NInterval child : children) {
455             // get the child which ends before idx but also ends closest to idx
456             if (child.lastNRangeStop() < idx
457                     && child.lastNRangeStop() > highestEndingAllowed) {
458                 tmp = child;
459                 highestEndingAllowed = tmp.lastNRangeStop();
460             }
461         }
462         return tmp;
463     }
464
465     /**
466      * The child of this interval which is live or starts after the given basic
467      * block's start
468      *
469      * @param b
470      *            the basic block
471      * @return the child of this interval which starts at or nearest (after)
472      *         this basic block's start (fist lir instruction index).
473      */
474
475     public NInterval childAtOrStartingAfter(NBasicBlock b) {
476         int idx = b.getFirstLIRInstId();
477         for (NInterval child : children) {
478             if (child.isLiveAt(idx))
479                 return child;
480         }
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
481         NInterval tmp = this;
482         int lowestStartAllowed = b.getLastLIRInstId();// block's end
483         for (NInterval child : children) {
484             if (child.firstRangeStart() > idx
485                     && child.firstRangeStart() < lowestStartAllowed) {
486                 tmp = child;
487                 lowestStartAllowed = tmp.firstRangeStart();
488             }
489         }
490         return tmp;
491     }
492 
493     /**
494      * Returns the basic block in which this interval's start position falls.
495      *
496      * @return basic block in which this interval's start position falls.
497      */
498 
499     public int startsAtBlock() {
500         for (NBasicBlock b : this.cfg.basicBlocks) {
501             if (this.firstRangeStart() >= b.getFirstLIRInstId()
502                     && this.firstRangeStart() <= b.getLastLIRInstId())
503                 return b.id;
504         }
505         return -1; // this will never happen
506     }
507 
508     /**
509      * The basic block in which this interval's end position falls.
510      *
511      * @return the basic block number.
512      */
513 
514     public int endsAtBlock() {
515         for (NBasicBlock b : this.cfg.basicBlocks) {
516             if (this.lastNRangeStop() >= b.getFirstLIRInstId()
517                     && this.lastNRangeStop() <= b.getLastLIRInstId()) {
518                 return b.id;
519             }
520         }
521         return -1; // this will never happen
522     }
523 
524     /**
525      * Assigns an offset to this interval (if one hasn't been already assigned).
526      * Assigns that same offset to any (newly created) children.
527      */
528 
529     public void spill() {
530         this.spill = true;
531         if (this.offset == -1) {
532             this.offset = cfg.offset++;
533             this.offsetFrom = OffsetFrom.SP;
534         }
535         for (NInterval child : children) {
536             if (child.offset == -1) {
537                 child.offset = this.offset;
538                 child.offsetFrom = this.offsetFrom;
539             }
540         }
541     }
542 
543     // The following two methods are used for insertion of move instructions
544     // for spills.
545 
546     /**
547      * Is this interval a child interval?
548      *
549      * @return true or false.
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
     */

    public boolean isChild() {
        if (this.parent != null) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * Is this interval a parent interval? (Ie, does it have children?)
     *
     * @return true or false.
     */

    public boolean isParent() {
        return !this.children.isEmpty();
    }

}

/** The types of stack pointers. **/
enum OffsetFrom {
    FP, SP
};

/** The types of possible uses. */
enum InstructionType {
    read, write
};

/**
 * A liveness range (for an interval).
 */

class NRange implements Comparable<NRange> {

    /** The range's start position. */
    public int start;

    /** The range's stop position. */
    public int stop;

    /**
     * Construct a liveness range extending from start to stop (positions in the
     * code).
     *
     * @param start
     *            start position.
     * @param stop
     *            stop position.
     */
    public NRange(int start, int stop) {
        this.start = start;
        this.stop = stop;

    }

    /**
     * Mutates current range to be only as long as the split point and returns
     * the remainder as a new range.
     *
     * @param newStart
     *            the split location
     * @param oldStop
     *            the split location
     * @return the new NRange which begins at the split position and runs to the
     *         end of this previously whole range.
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

```
619      */
620
621     public NRange splitRange(int newStart, int oldStop) {
622         NRange newRange = new NRange(newStart, stop);
623         this.stop = oldStop;
624
625         return newRange;
626     }
627
628     /**
629      * Does this liveness range overlap with another?
630      *
631      * @param a
632      *             The other range.
633      * @return true or false.
634      */
635
636     public boolean rangeOverlaps(NRange a) {
637         if (a.start < this.start) {
638             if (a.stop <= this.start) {
639                 return false;
640             } else {
641                 return true;
642             }
643         } else if (a.start < this.stop) {
644             return true;
645         } else {
646             return false;
647         }
648     }
649
650     /**
651      * One liveness range comes before another if its start position comes
652      * before the other's start position.
653      *
654      * @param other
655      *             the other range.
656      *
657      * @return comparison.
658      */
659
660     public int compareTo(NRange other) {
661         return this.start - other.start;
662     }
663
664     /**
665      * The string representation of the range.
666      *
667      * @return "[start,stop]"
668      */
669
670     public String toString() {
671         return "[" + start + ", " + stop + "]";
672     }
673
674 }
675
```

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder