Tree: 1f06f3751d

Find file Copy path

rars / rars / assembler / DataTypes.java

TheThirdOne Fix auipc and lui argument checks

1f06f37 on 7 Jun

1 contributor

Raw Blame History

166 lines (151 sloc) 6.21 KB

```
package rars.assembler;

/*
Copyright (c) 2003-2006,  Pete Sanderson and Kenneth Vollmar

Developed by Pete Sanderson (psanderson@otterbein.edu)
and Kenneth Vollmar (kenvollmar@missouristate.edu)

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject
to the following conditions:
```

```

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

(MIT license, http://www.opensource.org/licenses/mit-license.html)
 */


/**
 * Information about data types.
 *
 * @author Pete Sanderson
 * @version August 2003
 **/

public final class DataTypes {
    /**
     * Number of bytes occupied by double is 8.
     **/
    public static final int DOUBLE_SIZE = 8;
    /**
     * Number of bytes occupied by float is 4.
     **/
    public static final int FLOAT_SIZE = 4;
    /**
     * Number of bytes occupied by word is 4.
     **/
    public static final int WORD_SIZE = 4;
    /**
     * Number of bytes occupied by halfword is 2.
     **/
    public static final int HALF_SIZE = 2;
    /**
     * Number of bytes occupied by byte is 1.
     **/
    public static final int BYTE_SIZE = 1;
    /**
     * Number of bytes occupied by character is 1.
     **/
    public static final int CHAR_SIZE = 1;
    /**
     * Maximum value that can be stored in a word is 2<sup>31</sup>-1
     **/
    public static final int MAX_WORD_VALUE = Integer.MAX_VALUE;
    /**
     * Lowest value that can be stored in aword is -2<sup>31</sup>
     **/
    public static final int MIN_WORD_VALUE = Integer.MIN_VALUE;
    /**
     * Maximum value that can be stored in a halfword is 2<sup>15</sup>-1
     **/
    public static final int MAX_HALF_VALUE = 32767; //(int)Math.pow(2,15)
    /**
     * Lowest value that can be stored in a halfword is -2<sup>15</sup>
     **/
    public static final int MIN_HALF_VALUE = -32768; //0 - (int) Math.pow(
    /**
     * Maximum value that can be stored in a 12 bit immediate is 2<sup>11<
     **/
    public static final int MAX_IMMEDIATE_VALUE = 0x000007FF;
    /**
     * Lowest value that can be stored in a 12 bit immediate is -2<sup>11<
     **/
    public static final int MIN_IMMEDIATE_VALUE = 0xFFFFF800;
    /**
     * Maximum value that can be stored in a 20 bit immediate is 2<sup>19<
     **/
    public static final int MAX_UPPER_VALUE = 0x000FFFFF;
    /**
     * Lowest value that can be stored in a 20 bit immediate is -2<sup>19<
     **/
    public static final int MIN_UPPER_VALUE = 0x00000000;
    /**
     * Maximum value that can be stored in a byte is 2<sup>7</sup>-1
     **/
    public static final int MAX_BYTE_VALUE = Byte.MAX_VALUE;
    /**
     * Lowest value that can be stored in a byte is -2<sup>7</sup>
     **/
    public static final int MIN_BYTE_VALUE = Byte.MIN_VALUE;
    /**
     * Maximum positive finite value that can be stored in a float is same
     **/
    public static final double MAX_FLOAT_VALUE = Float.MAX_VALUE;
    /**
     * Largest magnitude negative value that can be stored in a float (neg
     **/
    public static final double LOW_FLOAT_VALUE = -Float.MAX_VALUE;

    /**
     * Get length in bytes for numeric RISCV directives.
     *
     * @param direct Directive to be measured.
     * @return Returns length in bytes for values of that type.  If type i
     * (or not implemented yet), returns 0.
     **/

    public static int getLengthInBytes(Directives direct) {
        if (direct == Directives.FLOAT)
            return FLOAT_SIZE;
        else if (direct == Directives.DOUBLE)
            return DOUBLE_SIZE;
        else if (direct == Directives.WORD)
            return WORD_SIZE;
        else if (direct == Directives.HALF)
            return HALF_SIZE;
        else if (direct == Directives.BYTE)
            return BYTE_SIZE;
        else
            return 0;
    }


    /**
     * Determines whether given integer value falls within value range for
     *
     * @param direct Directive that controls storage allocation for value.
     * @param value  The value to be stored.
     * @return Returns <tt>true</tt> if value can be stored in the number (
     * by the given directive (.word, .half, .byte), <tt>false</tt> otherw
     **/
    public static boolean outOfRange(Directives direct, int value) {
        // Hex values used here rather than constants because there aren't
        return (direct == Directives.HALF && (value < MIN_HALF_VALUE || va
                (direct == Directives.BYTE && (value < MIN_BYTE_VALUE || v
    }

    /**
     * Determines whether given floating point value falls within value ra
     * For float, this refers to range of the data type, not precision.  E
     * be stored in a float with loss of precision.  It's within the range
     * stored in a float because the exponent 500 is too large (float allo
     *
     * @param direct Directive that controls storage allocation for value.
     * @param value  The value to be stored.
     * @return Returns <tt>true</tt> if value is within range of
     * the given directive (.float, .double), <tt>false</tt> otherwise.
     **/
    public static boolean outOfRange(Directives direct, double value) {
        return direct == Directives.FLOAT && (value < LOW_FLOAT_VALUE || v
    }
}
```