```
// MicroEmulator.java - MICROPROCESSOR EMULATOR
//
// MODULE INDEX
// NAME                          CONTENTS
// MicroButton                   Button listener class
// MicroButton.MicroButton       Button listener constructor
// MicroButton.mouseClicked      Process mouse clicked event
// MicroButton.mousePressed      Process mouse button pressed event
// MicroButton.mouseReleased     Process mouse button released event
// MicroButton.mouseEntered      Process mouse entered component event
// MicroButton.mouseExited       Process mouse exited component event
// MicroButton.buttonIsPushed    Test if the button is pushed
// Repainter                     Repainter class
// Repainter.run                 Code to be executed in the event proc thread
// Repainter.Repainter           Repainter constructor
// init                          Initiation
// paint                         Paint the image
// start                         Start execution of the applet
// stop                          Stop execution of the applet
// getAppletInfo                 Get applet information
// readData                      Read from the data address space
// writeData                     Write to the data address space
// run                           Processor thread main line
//
// MAINTENANCE HISTORY
// DATE         PROGRAMMER AND DETAILS
// 21-09-06     JS      Original
//
//-----------------------------------------------------------------------------

// IMPORTS

import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import java.net.*;
import java.lang.*;

//-----------------------------------------------------------------------------

// MICROPROCESSOR EMULATOR CLASS DEFINITION

public class MicroEmulator
extends Applet
implements Runnable
{
    //-------------------------------------------------------------------------

    // INSTANCE DATA

    static final int    DATA_SIZE = 240;        // Data memory segment size
    static final int    PROG_SIZE = 65536;      // Program memory segment size
    static final int    IO_BASE = 0xf0;         // I/O register base address
    static final int    IO_CNT = 8;             // Number of I/O registers
    static final String PROG_NAME="MicroProg.bin"; // Executable program name

    MicroButton         buttonArr[];            // Button array
    Rectangle           indRectArr[];           // Indicator rectangle array
    boolean             indStateArr[];          // Indicator state array
    byte                dataMem[];              // Data memory
    byte                progMem[];              // Program memory
    Thread              procThread;             // Processor thread
    Applet              thisApplet;             // Reference to this applet
```

```
//---------------------------------------------------------------------------

// BUTTON LISTENER CLASS

private class MicroButton
extends Button
implements MouseListener
{
    //-----------------------------------------------------------------------

    // INSTANCE VARIABLES

    boolean         mouseDown;              // Mouse button is down
    boolean         cursorOver;             // Cursor is over the control

    //-----------------------------------------------------------------------

    // CONSTRUCTOR

    MicroButton (
        String      label)          // Button label
    {
        super (label);
        addMouseListener (this);
    }

    //-----------------------------------------------------------------------

    // MOUSE CLICKED

    public void
    mouseClicked (
        MouseEvent  me)             // Mouse event
    {
        // Empty
    }

    //-----------------------------------------------------------------------

    // MOUSE BUTTON PRESSED

    public void
    mousePressed (
        MouseEvent  me)             // Mouse event
    {
        mouseDown = true;
    }

    //-----------------------------------------------------------------------

    // MOUSE BUTTON RELEASE

    public void
    mouseReleased (
        MouseEvent  me)             // Mouse event
    {
        mouseDown = false;
    }

    //-----------------------------------------------------------------------

    // MOUSE ENTERED COMPONENT

    public void
    mouseEntered (
        MouseEvent  me)             // Mouse event
```

```
    {
        cursorOver = true;
    }

    //-------------------------------------------------------------------------

    // MOUSE EXITED COMPONENT

    public void
    mouseExited (
        MouseEvent  me)             // Mouse event
    {
        cursorOver = false;
    }

    //-------------------------------------------------------------------------

    // TEST IF THE BUTTON IS PUSHED

    public boolean
    buttonIsPushed ()
    {
        return cursorOver && mouseDown;
    }
}

//-----------------------------------------------------------------------------

// REPAINTER CLASS

private class Repainter
implements Runnable
{
    //-------------------------------------------------------------------------

    // CLASS INSTANCE VARIABLES

    Rectangle       repaintRect;    // Rectangle to repaint

    //-------------------------------------------------------------------------

    // CODE TO BE EXECUTED IN THE EVENT PROCESSING THREAD

    public void run ()
    {
        thisApplet.repaint (repaintRect.x, repaintRect.y,
            repaintRect.width, repaintRect.height);
    }

    //-------------------------------------------------------------------------

    // CONSTRUCTOR

    Repainter (
        Rectangle   rect)           // Rectangle to repaint
    {
        repaintRect = rect;
    }
}

//-----------------------------------------------------------------------------

// INITIATION

public void
init ()
```

```
{
    int             i;              // General purpose index
    int             unit;           // Scaling unit
    URL             progUrl;        // Program file URL
    HttpURLConnection progCon;      // Program download connection instance
    InputStream     progStream;     // Program file input stream
    int             ofs;            // Offset in the input stream
    int             len;            // Length read

    // Load the pointer to this applet

    thisApplet = this;

    // Load the dialog box units

    unit = (getFontMetrics(getFont()).getHeight() + 7) / 8;

    // Create the indicator rectangles

    indRectArr = new Rectangle [ IO_CNT ];
    indStateArr = new boolean [ IO_CNT ];
    for (i = 0; i < IO_CNT; i++) {
        indRectArr[i] = new Rectangle ((16 + i*24)*unit, 16*unit,
            16*unit, 4*unit);
        indStateArr[i] = false;
    }

    // Create the buttons and indicator rectangles

    setLayout (null);
    buttonArr = new MicroButton [ IO_CNT ];
    for (i = 0; i < IO_CNT; i++) {
        buttonArr[i] = new MicroButton (Integer.toString(i));
        buttonArr[i].setBounds ((16 + i*24)*unit, 48*unit,
            16*unit, 16*unit);
        add (buttonArr[i]);
    }

    // Create the memory structures

    dataMem = new byte [ DATA_SIZE ];
    progMem = new byte [ PROG_SIZE ];

    // Download the executable file from the server

    try {
        progUrl = new URL (getCodeBase(), PROG_NAME);
        progCon = (HttpURLConnection) progUrl.openConnection();
        progStream = progCon.getInputStream ();
        ofs = 0;
        do {
            len = progStream.read (progMem, ofs, PROG_SIZE-ofs);
            if (len != -1) ofs += len;
        } while (len != -1 && ofs < PROG_SIZE);
        progStream.close ();
    }
    catch (IOException e) {
        throw new RuntimeException (e.toString());
    }

    // Create the processor thread

    procThread = new Thread (this);
}

//-----------------------------------------------------------------------
```

```
// PAINT THE IMAGE

public void
paint (
    Graphics        g)              // Reference to graphics instance
{
    int             i;              // General purpose index

    for (i = 0; i < IO_CNT; i++) {
        if (indStateArr[i])
            g.setColor (Color.green);
        else
            g.setColor (Color.black);
        g.fillRect (indRectArr[i].x, indRectArr[i].y,
            indRectArr[i].width, indRectArr[i].height);
    }
}

//-------------------------------------------------------------------------

// START EXECUTION OF THE APPLET

public void
start ()
{
    super.start ();
    procThread.start ();
}

//-------------------------------------------------------------------------

// STOP EXECUTION OF THE APPLET

public void
stop ()
{
    procThread.interrupt ();
    try {
        procThread.join ();
    }
    catch (InterruptedException e) {
        // Empty
    }
    super.stop ();
}

//-------------------------------------------------------------------------

// GET APPLET INFORMATION

public String
getAppletInfo ()
{
    return "MicroEmulator, Copyright, 2006, E-Genting Sdn. Bhd.";
}

//-------------------------------------------------------------------------

// READ FROM THE DATA ADDRESS SPACE

int
readData (
    int             adr)            // Address to read
{
    if (adr >= 0 && adr < DATA_SIZE)
```

```
        return dataMem[adr] & 0xff;
    if (adr >= IO_BASE && adr < IO_BASE+IO_CNT)
        return buttonArr[adr-IO_BASE].buttonIsPushed() ? 1 : 0;
    throw new RuntimeException ("invalid data address");
}

//------------------------------------------------------------------------

// WRITE TO THE DATA ADDRESS SPACE

void
writeData (
    int             adr,            // Address to read
    int             val)            // Value to write
throws InterruptedException
{
    // RAM writes

    if (adr >= 0 && adr < DATA_SIZE) {
        dataMem[adr] = (byte)val;
    }

    // I/O register writes

    else if (adr >= IO_BASE && adr < IO_BASE+IO_CNT) {
        if (indStateArr[adr-IO_BASE] != ((val & 1) != 0)) {
            indStateArr[adr-IO_BASE] = (val & 1) != 0;
            try {
                EventQueue.invokeAndWait
                    (new Repainter(indRectArr[adr-IO_BASE]));
            }
            catch (InterruptedException e) {
                throw e;
            }
            catch (Exception e) {
                throw new RuntimeException (e.toString());
            }
        }
    }

    // Other addresses are unacceptable

    else
        throw new RuntimeException ("invalid data address");
}

//------------------------------------------------------------------------

// PROCESSOR THREAD MAIN LINE

public void
run ()
{
    int             i;              // General purpose index
    boolean         c;              // Carry bit
    int             a;              // The accumulator
    int             sp;             // The stack pointer
    int             pc;             // The program counter

    // Initialise the program counter

    c = false;
    a = 0;
    sp = 0;
    pc = 0;
```

```
// Catch interrupted exceptions

try {

    // Execute instructions until interrupted

    while ( ! Thread.interrupted()) {

        // Process instructions depending on the instruction code
        // in the program memory pointed to by the accumulator

        switch (progMem[pc]) {

        // Load accumulator, immediate

        case 0x10:
            a = progMem[pc+1] & 0xff;
            pc += 2;
            break;

        // Load accumulator, direct

        case 0x11:
            a = readData (progMem[pc+1] & 0xff);
            pc += 2;
            break;

        // Load stack pointer, immediate

        case 0x12:
            sp = progMem[pc+1] & 0xff;
            pc += 2;
            break;

        // Store accumulator, direct

        case 0x13:
            writeData (progMem[pc+1] & 0xff, a & 0xff);
            pc += 2;
            break;

        // Add, direct

        case 0x20:
            a += readData (progMem[pc+1] & 0xff);
            c = a >= 256;
            a &= 255;
            pc += 2;
            break;

        // Subtract, direct

        case 0x21:
            a -= readData (progMem[pc+1] & 0xff);
            c = a < 0;
            a &= 255;
            pc += 2;
            break;

        // Branch

        case 0x30:
            pc += 2 + progMem[pc+1];
            break;

        // Branch if carry set
```

```
                case 0x31:
                    if (c)
                        pc += 2 + progMem[pc+1];
                    else
                        pc += 2;
                    break;

                // Jump, direct

                case 0x40:
                    pc = (int)(progMem[pc+1]&0xff) << 8 | (progMem[pc+2]&0xff);
                    break;

                // Call subroutine

                case 0x41:
                    dataMem[--sp] = (byte)(pc+3);
                    dataMem[--sp] = (byte)(pc+3 >> 8);
                    pc = (int)(progMem[pc+1]&0xff) << 8 | (progMem[pc+2]&0xff);
                    break;

                // Return from subroutine

                case 0x42:
                    pc = (dataMem[sp] & 0xff) << 8 | (dataMem[sp+1] & 0xff);
                    sp += 2;
                    break;

                // Other operation codes are unsupported

                default:
                    throw new RuntimeException ("unsupported operation code 0x"
                        + Integer.toString (progMem[pc] & 0xff, 16));
                }
            }
        }
        catch (InterruptedException e) {
            // Empty
        }
    }
}
```