

```

/*
* -----
* "THE BEER-WARE LICENSE" (Revision 42):
* <doktor@dyregod.dk> wrote this file. As long as you retain this notice you
* can do whatever you want with this stuff. If we meet some day, and you
think
* this stuff is worth it, you can buy me a beer in return. Ulf Holm Nielsen
* -----
*
*/

package dk.dyregod.linda;

import dk.dyregod.linda.util.*;
import java.util.BitSet;
import java.util.Hashtable;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.*;

public class Maskine implements Runnable
{
    Linda shell;

    public final static int X = 0;
    public final static int Y = 1;
    public final static int A = 2;
    public final static int P = 3;
    public final static int I = 4;
    public final static int ADR = 5;
    public final static int LDR = 6;
    public final static int LAR = 7;
    public final static int IO = 8;

    int[] registers = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

    String[] names = { "X", "Y", "A", "P", "I", "ADR", "LDR", "LAR", "IO" };

    int cmd;
    Hashtable instructionSet = new Hashtable();

    boolean done = false;
    boolean halt = false;
    boolean stop = false;
    boolean wait = false;

    int type = 0;

    int[] lager = new int[100];

    int bus;

    public Maskine(Linda shell)
    {
        this.shell = shell;
    }

    void setStatus(String text)
    {
        shell.statusStr = text;
    }
}

```

```

/*
 * update() sørger for at opdatere GUI
 *
 */

public void update()
{
    shell.update();
    if (type == 2)
    {
        halt = true;

        while (halt)
        {
            try
            {
                wait();
            }
            catch (Exception e)
            {
            }
        }
    }
    else if ((type == 1) && (done) && (!stop))
    {
        halt = true;

        while (halt)
        {
            try
            {
                wait();
            }
            catch (Exception e)
            {
            }
        }
    }
}

/*
 * run() sørger for afviklingen af program
 *
 */

public void run()
{
    stop = false;
    int i = 0;
    while (((i = executeNext()) == 0) && (!stop))
    {
        update();
    };
    if ((i == 1) && (stop))
    {
        shell.getShell().getDisplay().syncExec(new Runnable()
        {
            public void run()
            {
                MessageBox box =
                    new MessageBox(shell.getShell(), SWT.ICON_INFORMATION

```

```

| SWT.OK);
                                box.setText(shell.getResourceString("Programmet er slut"));
;
                                box.setMessage(
                                    shell.getResourceString(
                                        "En stop kommando er nået på adresse " + shell.
form2.format(registers[P] - 1)));
                                box.open();
                                }
                            });
                        }
                    }
                }

/*
 * Mikroinstruktioner
 *
 * modtagFraBus(int register)
 * ventForInput()
 * sendTilBus(int register)
 * taelOp()
 * laesLager()
 * skrivLager()
 * ALUAdd()
 * ALUSub()
 * ALUDiv()
 * ALUMult()
 *
 */

public void modtagFraBus(int register)
{
    registers[register] = bus;
    bus = 0;

    if (register == I)
    {
        registers[ADR] = registers[I] % 100;
    }
    setStatus("Modtag fra bus til register " + names[register]);
    update();
}

public void ventForInput()
{
    setStatus("Venter på input ");
    update();
    shell.getShell().getDisplay().syncExec(new Runnable()
    {
        public void run()
        {
            shell.ioText.setEditable(true);
        }
    });
    wait = true;
    while (wait)
    {
        try
        {
            wait();
        }
        catch (Exception e)
        {
        }
    }
}

```

```

        setStatus("Modtag input i I/O register " + registers[IO]);
        update();
    }
    public void sendTilBus(int register)
    {
        bus = registers[register];
        setStatus(
            "Send indholdet af "
            + names[register]
            + " ("
            + registers[register]
            + ") til bussen");
        update();
    }
    public void taelOp()
    {
        registers[P]++;
        setStatus("Tæl P en op");
        update();
    }
    public void laesLager()
    {
        registers[LDR] = lager[registers[LAR]];
        setStatus(
            "Læs indeholdet af adressen i LAR ("
            + registers[LAR]
            + ") ind i LDR ("
            + registers[LDR]
            + ")");
        update();
    }
    public void skrivLager()
    {
        lager[registers[LAR]] = registers[LDR];
        setStatus(
            "Skriv indeholdet af LDR ("
            + registers[LDR]
            + ") til adressen LAR ("
            + registers[LAR]
            + ") peger på");
        update();
    }
    public void ALUAdd()
    {
        registers[A] = registers[X] + registers[Y];
        setStatus("ALU Add (X + Y = A)");
        update();
    }
    public void ALUSub()
    {
        registers[A] = registers[X] - registers[Y];
        setStatus("ALU Sub (X - Y = A)");
        update();
    }
    public void ALUDiv()
    {
        registers[A] = registers[X] / registers[Y];
        setStatus("ALU Div (X / Y = A)");
        update();
    }
    public void ALUMult()
    {
        registers[A] = registers[X] * registers[Y];
        setStatus("ALU Mult (X * Y = A)");
    }

```

```

        update();
    }

    /*
    *
    * executeNext() indeholder logik til at håndtere
    * fortolkning af instruktionssættet samt mikrokode
    * til at hente næste instruktion.
    */

public int executeNext()
{
    done = false;

    sendTilBus(P);
    modtagFraBus(LAR);
    laesLager();
    sendTilBus(LDR);
    modtagFraBus(I);
    taelOp();

    cmd = registers[I] / 100;
    switch (cmd)
    {
        case 1 :
            STOP();
            done = true;
            return 1;
        case 2 :
            HENT();
            break;
        case 3 :
            GEM();
            break;
        case 4 :
            ADD();
            break;
        case 5 :
            SUB();
            break;
        case 6 :
            MULT();
            break;
        case 7 :
            DIV();
            break;
        case 8 :
            IND();
            break;
        case 9 :
            UD();
            break;
        case 10 :
            HOP();
            break;
        case 11 :
            HNUL();
            break;
        case 12 :
            HNEG();
            break;
    }
}

```

```

        default :
            shell.getShell().getDisplay().syncExec(new Runnable()
            {
                public void run()
                {
                    ErrorMessage.display(
                        shell.getShell(),
                        "Ukendt instruktion " + cmd + " i lagercelle " + (
registers[P] - 1),
                        "Fejl");
                }
            });
            done = true;
            stop = true;
            return 2;

        }
        done = true;
        return 0;
    }

    /*
    * reset() sætter alle registre til 0 opdaterer GUI.
    */
    public void reset()
    {
        for (int i = 0; i < registers.length; i++)
        {
            registers[i] = 0;
        }
        update();
    }

    /*
    * Instruktionssættet, en metode pr instruktion håndteres af executeNext()
    */
    public void STOP()
    {
    }
    public void HENT()
    {
        sendTilBus(ADR);
        modtagFraBus(LAR);
        laesLager();
        sendTilBus(LDR);
        modtagFraBus(A);
    }
    public void GEM()
    {
        sendTilBus(ADR);
        modtagFraBus(LAR);
        sendTilBus(A);
        modtagFraBus(LDR);
        skrivLager();
    }
    public void ADD()
    {
        sendTilBus(A);
        modtagFraBus(X);
        sendTilBus(ADR);
    }

```

```

        modtagFraBus(LAR);
        laesLager();
        sendTilBus(LDR);
        modtagFraBus(Y);
        ALUAdd();
    }
    public void SUB()
    {
        sendTilBus(A);
        modtagFraBus(X);
        sendTilBus(ADR);
        modtagFraBus(LAR);
        laesLager();
        sendTilBus(LDR);
        modtagFraBus(Y);
        ALUSub();
    }
    public void MULT()
    {
        sendTilBus(A);
        modtagFraBus(X);
        sendTilBus(ADR);
        modtagFraBus(LAR);
        laesLager();
        sendTilBus(LDR);
        modtagFraBus(Y);
        ALUMult();
    }
    public void DIV()
    {
        sendTilBus(A);
        modtagFraBus(X);
        sendTilBus(ADR);
        modtagFraBus(LAR);
        laesLager();
        sendTilBus(LDR);
        modtagFraBus(Y);
        ALUDiv();
    }
    public void IND()
    {
        ventForInput();
        sendTilBus(IO);
        modtagFraBus(LDR);
        sendTilBus(ADR);
        modtagFraBus(LAR);
        skrivLager();
    }
    public void UD()
    {
        sendTilBus(ADR);
        modtagFraBus(LAR);
        laesLager();
        sendTilBus(LDR);
        modtagFraBus(IO);
    }
    public void HOP()
    {
        sendTilBus(ADR);
        modtagFraBus(P);
    }
    public void HNUL()
    {
        if (registers[A] == 0)

```

```
        {
            sendTilBus(ADR);
            modtagFraBus(P);
        }
    }
    public void HNEG()
    {
        if (registers[A] < 0)
        {
            sendTilBus(ADR);
            modtagFraBus(P);
        }
    }
}
```