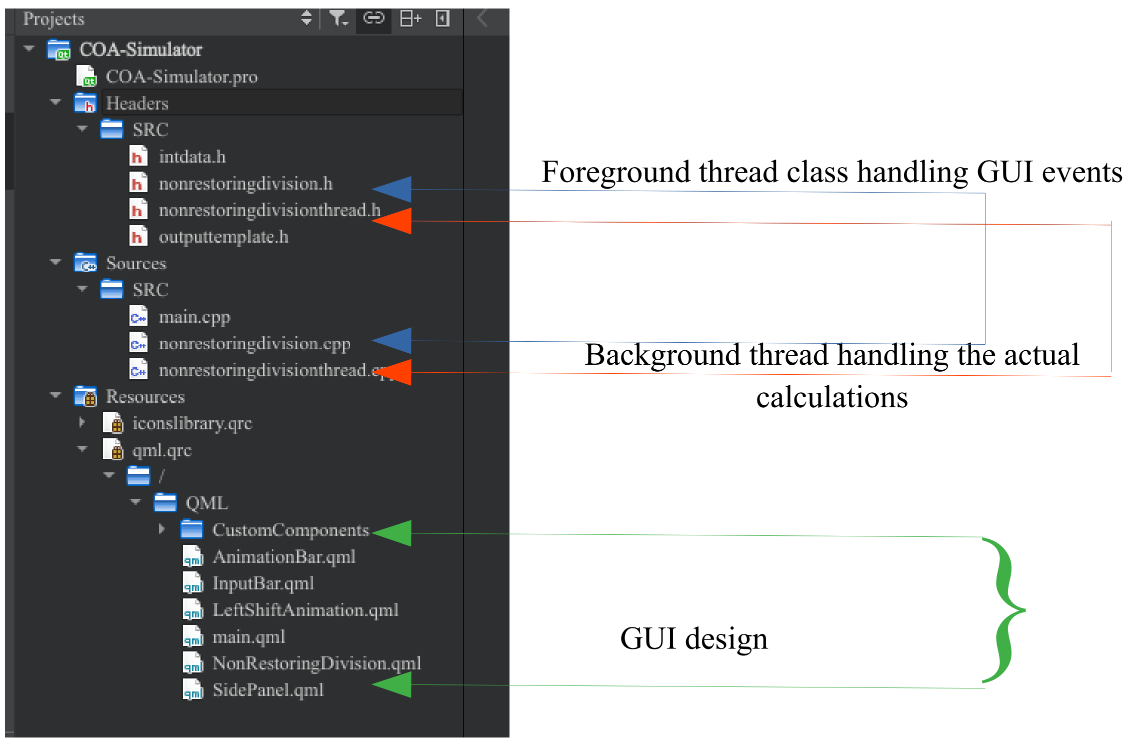
COA Mini Project
on
Simulation for Non-restoring Division Algorithm

By
Rajiv Singh (SE Comps B 58)
Sushrut Kuchik (SE Comps B 62)
Vivek Vaishya (SE Comps B 66)

Abstract

Non-Restoring Division Algorithm is used for division of two unsigned numbers. The main purpose is to develop a simulator which performs division of two unsigned binary numbers using non restoring concept. The simulation can developed using any known language. The simulation algorithm can be analysed by using algorithm analysis measures.

Source Code Flow



Source Code

//nonrestoringdivision.h

```
#ifndef NONRESTORINGDIVISION_H
#define NONRESTORINGDIVISION_H

#include <QObject>
#include <QThread>

class NonRestoringDivision : public QObject
{
        Q_OBJECT
        Q_PROPERTY(QString InputA READ InputA WRITE setInputA NOTIFY InputAChanged)
        Q_PROPERTY(QString InputB READ InputB WRITE setInputB NOTIFY InputBChanged)
        Q_PROPERTY(int NumberSystem READ NumberSystem WRITE setNumberSystem NOTIFY
NumberSystemChanged)
        Q_PROPERTY(QList<QObject*> OperationSteps READ OperationSteps NOTIFY OperationStepsChanged)

        Q_PROPERTY(QStringList Operand1 READ Operand1 NOTIFY Operand1Changed)
        Q_PROPERTY(int FOperand1 READ FOperand1 NOTIFY FOperand1Changed)

        Q_PROPERTY(QStringList Operand2 READ Operand2 NOTIFY Operand2Changed)
        Q_PROPERTY(int FOperand2 READ FOperand2 NOTIFY FOperand2Changed)

        Q_PROPERTY(int Operation READ Operation)
        Q_PROPERTY(QString OperationText READ OperationText NOTIFY OperationTextChanged)

public:
        explicit NonRestoringDivision(QObject *parent = nullptr);


            ...
{And it follows the definitions for all the above properties exposed to QML}
            ...

signals:
            ...
{Signals to notify the QML thread of actions performed by CPP}
            ...

        void triggerThread();

public slots:
            ...
{Any updates from background thread and or the QML thread will be called upon here}
            ...
        void calculate();
        void receiveUpdate(QList<int>, QList<int>, QString, int op, bool dash, bool newStep);

private:
        QList<int> makeBin(QString bin); //convert binary string input into binary list
        QList<int> decToBin(int dec, int bits = 0); //convert decimal string input into binary list
        int binToDec(QList<int> bin);
        QList<int> initBin(QList<int> input, int bit); //initialize a binary number 000... with 'bit' number of bits
        QString binToStr(QList<int> binList, bool dash = false); //convert binary list to string for displaying

        QStringList makeString(QList<int>); //output for animation

        QString inputA;
        QString inputB;
        int numberSystem;
            ...
{And other instance variables needed at run time for calculation}
            ...
};

#endif // NONRESTORINGDIVISION_H
```

//nonrestoringdivision.h

```
void NonRestoringDivision::calculate(){
	if(numberSystem == 0){
		inputABin = makeBin(inputA);
		inputBBin = makeBin(inputB);
	}
	else if(numberSystem == 1){
		inputABin = decToBin(inputA.toInt(nullptr, 10));
		inputBBin = decToBin(inputB.toInt(nullptr, 10));
	}
	else if(numberSystem == 2){
		//just changing the base to 16 makes it usable with decToBin as well
		inputABin = decToBin(inputA.toInt(nullptr, 16));
		inputBBin = decToBin(inputB.toInt(nullptr, 16));
	}
	inputBStr = binToStr(inputBBin);

	outputCBin = initBin(outputCBin, inputABin.length());

	NonRestoringDivisionThread *nrdt = new NonRestoringDivisionThread(inputABin, inputBBin, outputCBin);
	connect(nrdt, &NonRestoringDivisionThread::updateList, this, &NonRestoringDivision::receiveUpdate);
	connect(this, &NonRestoringDivision::triggerThread, nrdt, &NonRestoringDivisionThread::calculate);
	nrdt->moveToThread(&workerThread);

	if(workerThread.isRunning())
		workerThread.quit();

	operationSteps.clear();
	emit OperationStepsChanged();

	workerThread.start();
	emit triggerThread();
}


void NonRestoringDivision::receiveUpdate(QList<int> InputA, QList<int> InputB, QString Comment, int op, bool
dash, bool newStep){
	operation = op;
	emit operationChanged();

	outputCBin = InputA;
	inputABin = InputB;

	OutputTemplate *out = new OutputTemplate(binToStr(InputA), binToStr(InputB, dash), inputBStr, Comment,
newStep);
	operationSteps.append(out);

	if(op != 1 && op != 4){
		emit Operand1Changed();
		emit Operand2Changed();
		emit OperationStepsChanged();
	}

	if(op == 0)
		operationText = "Initializing values, Accumulator with 00... and Multiplier with Q";
	else if(op == 1)
		operationText = "Left shift contents of A & Q by 1 bit and keep the Qₙ₋₁ empty, to be filled later";
	else if(op == 2)
		operationText = "Add contents of Accumulator and Multiplier and store the result in Accumulator, since A₀
bit is 1";
	else if(op == 3)
```

```
                operationText = "Subtract contents of Accumulator from Multiplier and store the result in Accumulator, 
since A₀ bit is 0";
        else if(op == 4)
                operationText = "Set A₀ bit as inverse Qₙ₋₁ bit";
        else if(op == 5)
                operationText = "Since final result in Accumulator is negative (11...), add A + M one more time and 
consider it as the result";
        else{
                fOperand1 = binToDec(outputCBin);
                emit FOperand1Changed();
                fOperand2 = binToDec(inputABin);
                emit FOperand2Changed();
                operationText = "Here's your final result";
        }

        emit OperationTextChanged();

}
```

//nonrestoringdivisionthread.h

```
#ifndef NONRESTORINGDIVISIONTHREAD_H
#define NONRESTORINGDIVISIONTHREAD_H

#include <QObject>
#include <QThread>

class NonRestoringDivisionThread : public QObject
{
        Q_OBJECT
public:
        explicit NonRestoringDivisionThread(QList<int>, QList<int>, QList<int>, QObject *parent = nullptr);

        void calculate();

signals:
        void updateList(QList<int>, QList<int>, QString, int op, bool dash = false, bool newStep = false);

private:
        int binToDec(QList<int> bin);//convert binary list to decimal
        QList<int> decToBin(int dec, int bits = 0); //convert decimal string input into binary list

        QList<int> inputABin, inputBBin, outputCBin;
        uint wait;
};

#endif // NONRESTORINGDIVISIONTHREAD_H
```

//nonrestoringdivisionthread.cpp

```
void NonRestoringDivisionThread::calculate(){

        emit updateList(outputCBin, inputABin, "(0) Initial Values", 0);
        QThread::msleep(wait);
        int decBInput = binToDec(inputBBin);

        int count;
```

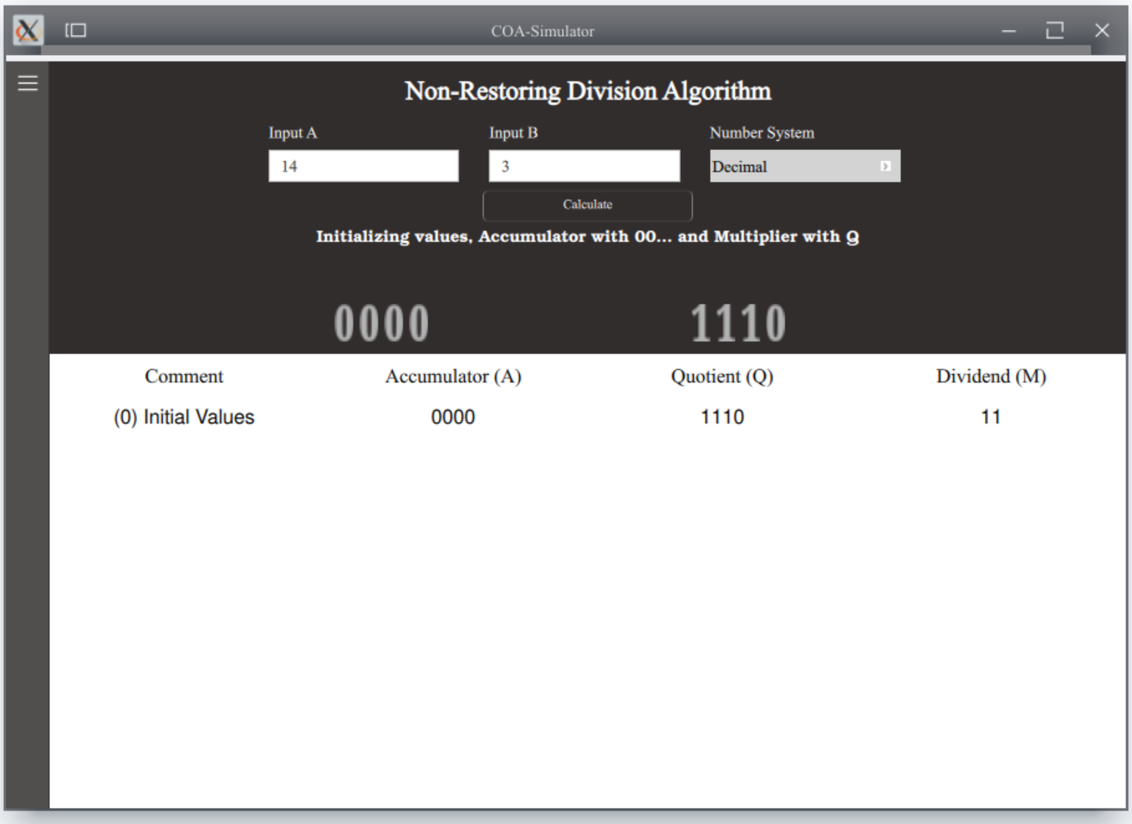
```
    for(count = 0; count < inputABin.length(); count++){
        outputCBin.removeLast();
        outputCBin.prepend(inputABin.last());
        inputABin.removeLast();

        emit updateList(outputCBin, inputABin, "(" + QString::number(count + 1) + ") Left Shift A + Q", 1, true,
true);
        QThread::msleep(wait);

        if(outputCBin.last() == 1){
            outputCBin = decToBin(binToDec(outputCBin) + decBInput, inputABin.length() + 1);
            emit updateList(outputCBin, inputABin, "A ← A + M", 2, true);
            QThread::msleep(wait);
        }
        else{
            outputCBin = decToBin(binToDec(outputCBin) - decBInput, inputABin.length() + 1);
            emit updateList(outputCBin, inputABin, "A ← A - M", 3, true);
            QThread::msleep(wait);
        }

        inputABin.prepend(!outputCBin.last());
        emit updateList(outputCBin, inputABin, "A₀ ← !Qₙ₋₁", 4);
        QThread::msleep(wait);
    }

    if(outputCBin.last() == 1){
        int negativeRes = binToDec(outputCBin);
        negativeRes += decBInput;
        outputCBin = decToBin(negativeRes, outputCBin.length());
        emit updateList(outputCBin, inputABin, "(" + QString::number(count++) + ") For negative output", 5,
false, true);
    }

    emit updateList(outputCBin, inputABin, "(" + QString::number(count) + ") Final Output", 6, false, true);

}


int NonRestoringDivisionThread::binToDec(QList<int> bin){
    int output = 0;
    for (int i = 0; bin.length() > 0; i++) {
        output += bin.first()*pow(2, i);
        bin.removeFirst();
    }
    return output;
}

QList<int> NonRestoringDivisionThread::decToBin(int dec, int bits){
    QList<int> bin;
    if(bits > 0){
        while(bits > 0){
            bin.append(dec & 0x1);
            dec = dec >> 1;
            bits--;
        }
    }
    else{
        while (dec > 0) {
            bin.append(dec % 2);
            dec /= 2;
        }
    }

    return  bin;
}
```

```
//AnimationBar.qml

import QtQuick 2.7

Rectangle {
    id: animationBar
    property var shiftingBit: nonRestoringDivisionCPP.Operand2
    color: "transparent"

    Rectangle{
        id: operationText
        width: parent.width
        height: parent.height*0.3
        anchors.horizontalCenter: parent.horizontalCenter
        color: "transparent"
        Text {
            anchors.topMargin: 5
            width: parent.width
            height: parent.height
            text: nonRestoringDivisionCPP.OperationText
            font.family: "URW Bookman"
            color: "white"
            font.pointSize: 10
            horizontalAlignment: Text.AlignHCenter
        }
    }

    Text {
        text: nonRestoringDivisionCPP.FOperand1
        visible: text != "-32767"
        font.pointSize: 14
        font.bold: true
        color: "blue"
        x: operand1.x + operand1.width/2
        y: operand1.y - operand1.height
    }

    ListView{
        id: operand1
        x: parent.x + parent.width*0.45 - width*2
        width: (height*0.5 + 5)*count
        height: parent.height*0.3
        anchors.bottom: parent.bottom
        anchors.bottomMargin: 10
        model: nonRestoringDivisionCPP.Operand1
        orientation: ListView.Horizontal
        spacing: 5

        delegate: Rectangle{
            height: parent.height
            width: height*0.5
            color: "transparent"
```

```
                Image {
                    width: parent.width
                    height: parent.height
                    source: "/local/assets/" + modelData + "-grey.png"
                    sourceSize.height: height*0.75
                    sourceSize.width: width*0.75
                    anchors.centerIn: parent
                }
            }
        }

        Text {
            text: nonRestoringDivisionCPP.FOperand2
            visible: text != "-32767"
            font.pointSize: 14
            font.bold: true
            color: "blue"
            x: operand2.x + operand1.width/2
            y: operand2.y - operand2.height
        }

        ListView{
            id: operand2
            x: parent.x + parent.width*0.55 + width*0.5
            width: (height*0.5 + 5)*count
            height: parent.height*0.3
            anchors.bottom: parent.bottom
            anchors.bottomMargin: 10
            model: nonRestoringDivisionCPP.Operand2
            orientation: ListView.Horizontal
            spacing: 5

            delegate: Rectangle{
                height: parent.height
                width: height*0.5
                color: "transparent"
                Image {
                    width: parent.width
                    height: parent.height
                    source: "/local/assets/" + modelData + "-grey.png"
                    sourceSize.height: height*0.75
                    sourceSize.width: width*0.75
                    anchors.centerIn: parent
                }
            }
        }

        Rectangle{
            id: shiftingBitAnimatingRect
            height: parent.height*0.3
            width: height*0.5
            color: "transparent"
            visible: false
            y: operand2.y

            Image {
                width: parent.width
                height: parent.height
                source: "/local/assets/" + shiftingBit[0] + "-grey.png"
                sourceSize.height: height*0.75
            sourceSize.width: width*0.75
            anchors.centerIn: parent
        }
}

SequentialAnimation{
        id: shiftBitAnimation
        NumberAnimation{
                target: shiftingBitAnimatingRect
                property: "x"
                from: operand2.x
                to: operand1.x + operand1.width
                duration: 2000
        }
        PropertyAnimation{
                target: shiftingBitAnimatingRect
                property: "visible"
                to: false
        }

}

Rectangle{
        id: invertBitAnimatingRect
        height: 25
        anchors.top: operationText.bottom
        anchors.left: operand1.left
        color: "transparent"
        visible: false
        Image {
                width: parent.width
                height: parent.height
                source: "/local/assets/invert-bit-connectorpng.png"
                sourceSize.width: parent.width
        }
}

SequentialAnimation{
        id: invertBitAnimation
        NumberAnimation{
                target: invertBitAnimatingRect
                property: "width"
                from: 0
                to: (operand2.x + operand2.width*1.2) - operand1.x
                duration: 1500
        }
        PropertyAnimation{
                target: invertBitAnimatingRect
                property: "visible"
                to: false
                duration: 500
        }
}

Connections{
        target: nonRestoringDivisionCPP
        onOperationChanged:{
                if(nonRestoringDivisionCPP.Operation === 1){
                        shiftingBitAnimatingRect.visible = true
                        shiftBitAnimation.start()
```

```
                    }
                    else if(nonRestoringDivisionCPP.Operation === 4){
                            invertBitAnimatingRect.visible = true
                            invertBitAnimation.start()
                    }
            }
    }

    Connections{
            target: shiftBitAnimation
            onFinished: nonRestoringDivisionCPP.letFurtherUpdateHappen()
    }

    Connections{
            target: invertBitAnimation
            onFinished: nonRestoringDivisionCPP.letFurtherUpdateHappen()
    }
}
```

Screenshots



Non-Restoring Division Algorithm

Input A: 14

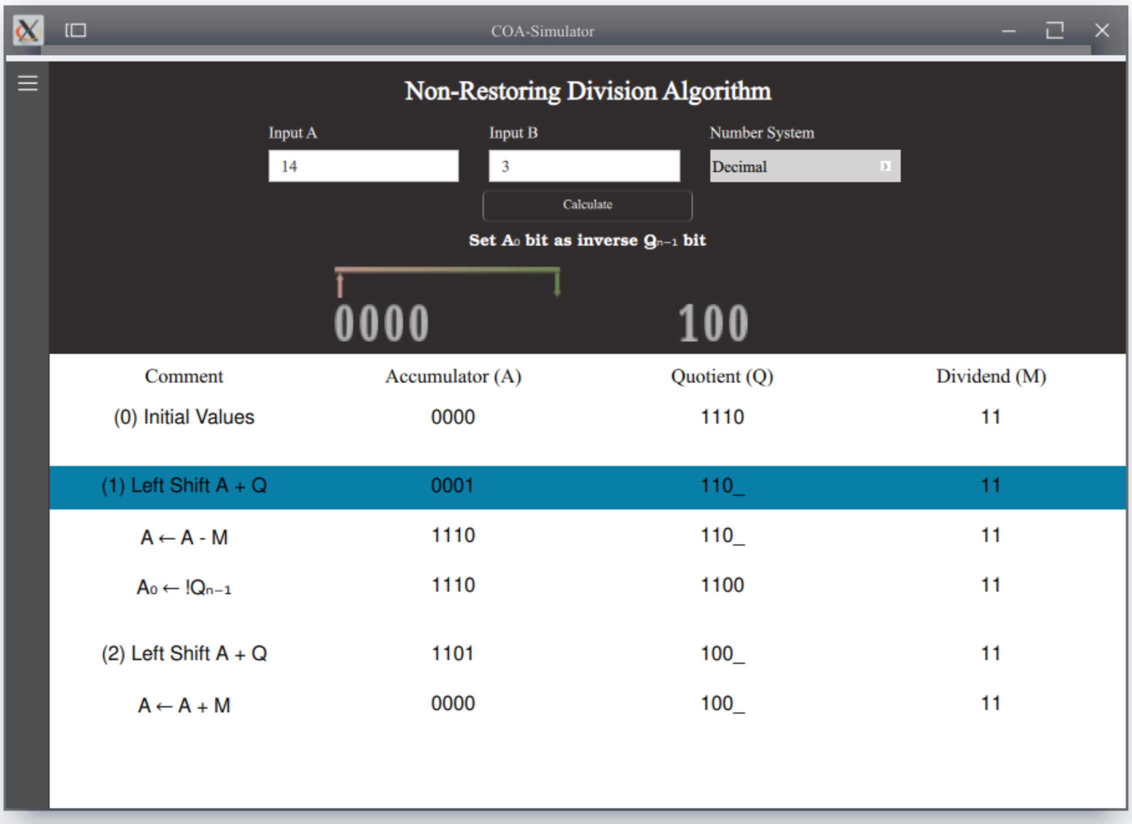
Input B: 3

Number System: Decimal

Initializing values, Accumulator with 00... and Multiplier with Q

0000   1110

Comment	Accumulator (A)	Quotient (Q)	Dividend (M)
(0) Initial Values	0000	1110	11



Non-Restoring Division Algorithm

Input A: 14

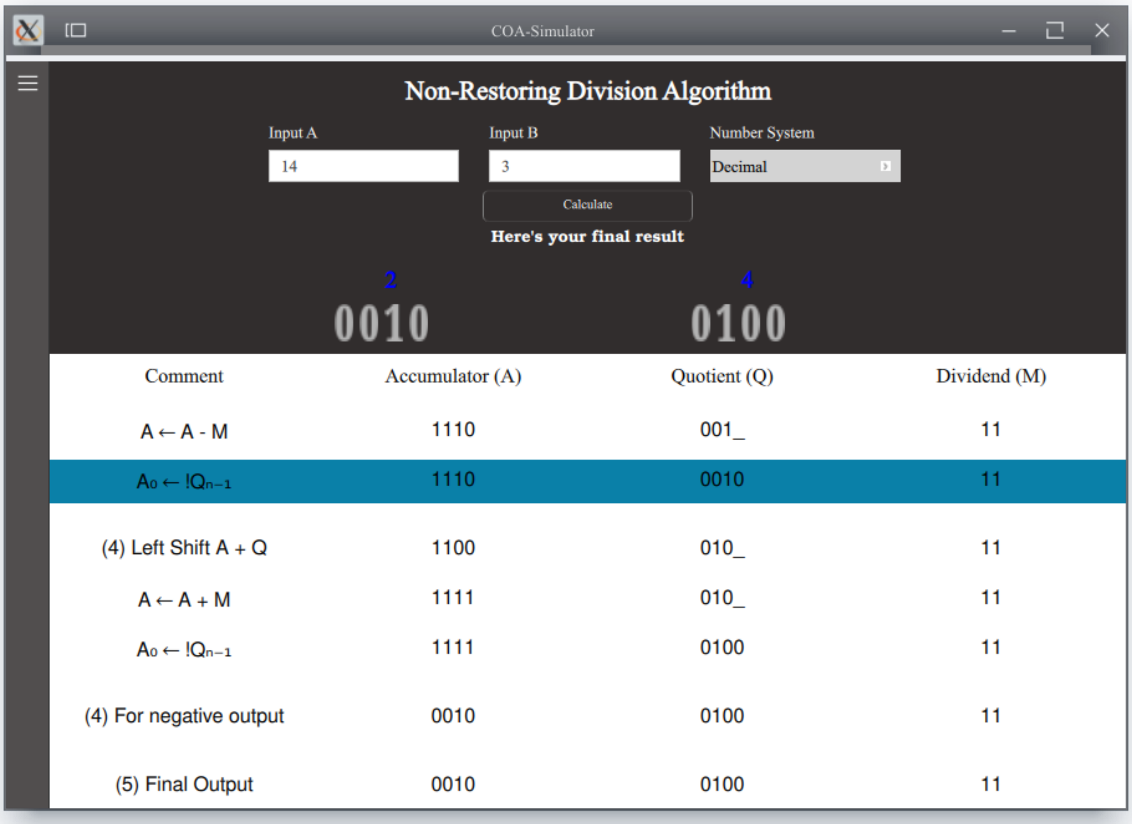
Input B: 3

Number System: Decimal

Set A_0 bit as inverse Q_{n-1} bit

0000 100

Comment	Accumulator (A)	Quotient (Q)	Dividend (M)
(0) Initial Values	0000	1110	11
(1) Left Shift A + Q	0001	110_	11
$A \leftarrow A - M$	1110	110_	11
$A_0 \leftarrow !Q_{n-1}$	1110	1100	11
(2) Left Shift A + Q	1101	100_	11
$A \leftarrow A + M$	0000	100_	11



Non-Restoring Division Algorithm

Input A: 14

Input B: 3

Number System: Decimal

Here's your final result

2 4

0010 0100

Comment	Accumulator (A)	Quotient (Q)	Dividend (M)
$A \leftarrow A - M$	1110	001_	11
$A_0 \leftarrow !Q_{n-1}$	1110	0010	11
(4) Left Shift A + Q	1100	010_	11
$A \leftarrow A + M$	1111	010_	11
$A_0 \leftarrow !Q_{n-1}$	1111	0100	11
(4) For negative output	0010	0100	11
(5) Final Output	0010	0100	11