ASSUMPTION UNIVERSITY
Vincent Mary School of Science & Technology
DEPARTMENT OF COMPUTER SCIENCE
COURSE OUTLINE

CS2202 (ComArch) Sem 1/2017
Project Presentation ISA

By Gaoyang 6019545

CONTENTS

01
Description

Will describe the specifications of the program, constitute and how to use the ISA instruction set for some conventional mathematical calculations.

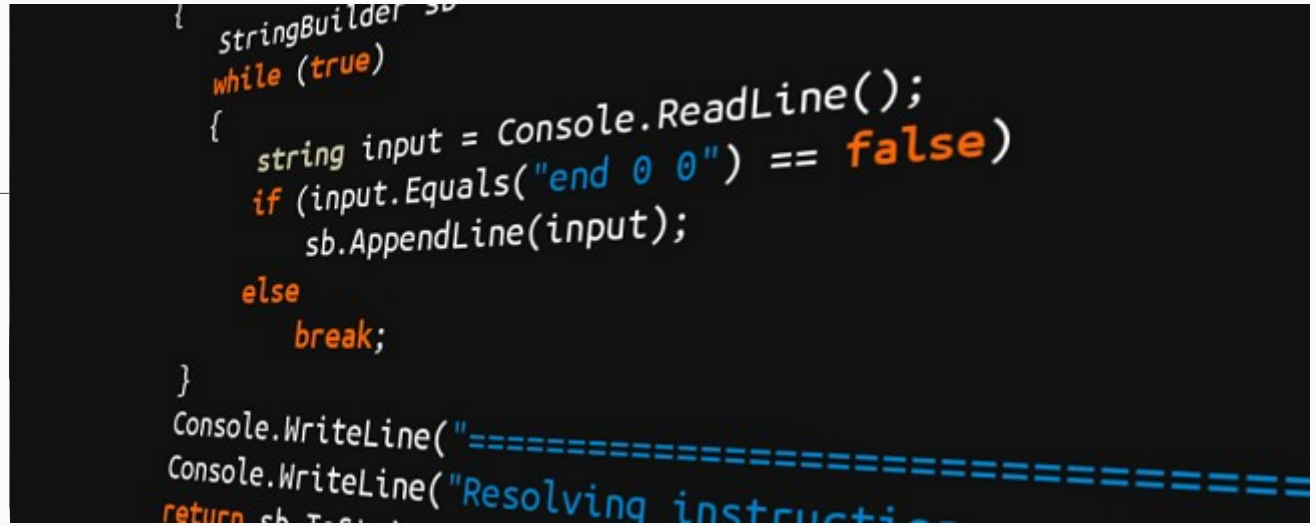
DESCRIPTION

INSTRUCTION SET

This simple CPU instruction set emulation program size is 24-bit, using C # language programming, C # good language style can easily control the binary data and encapsulation function.

COMPOSITION

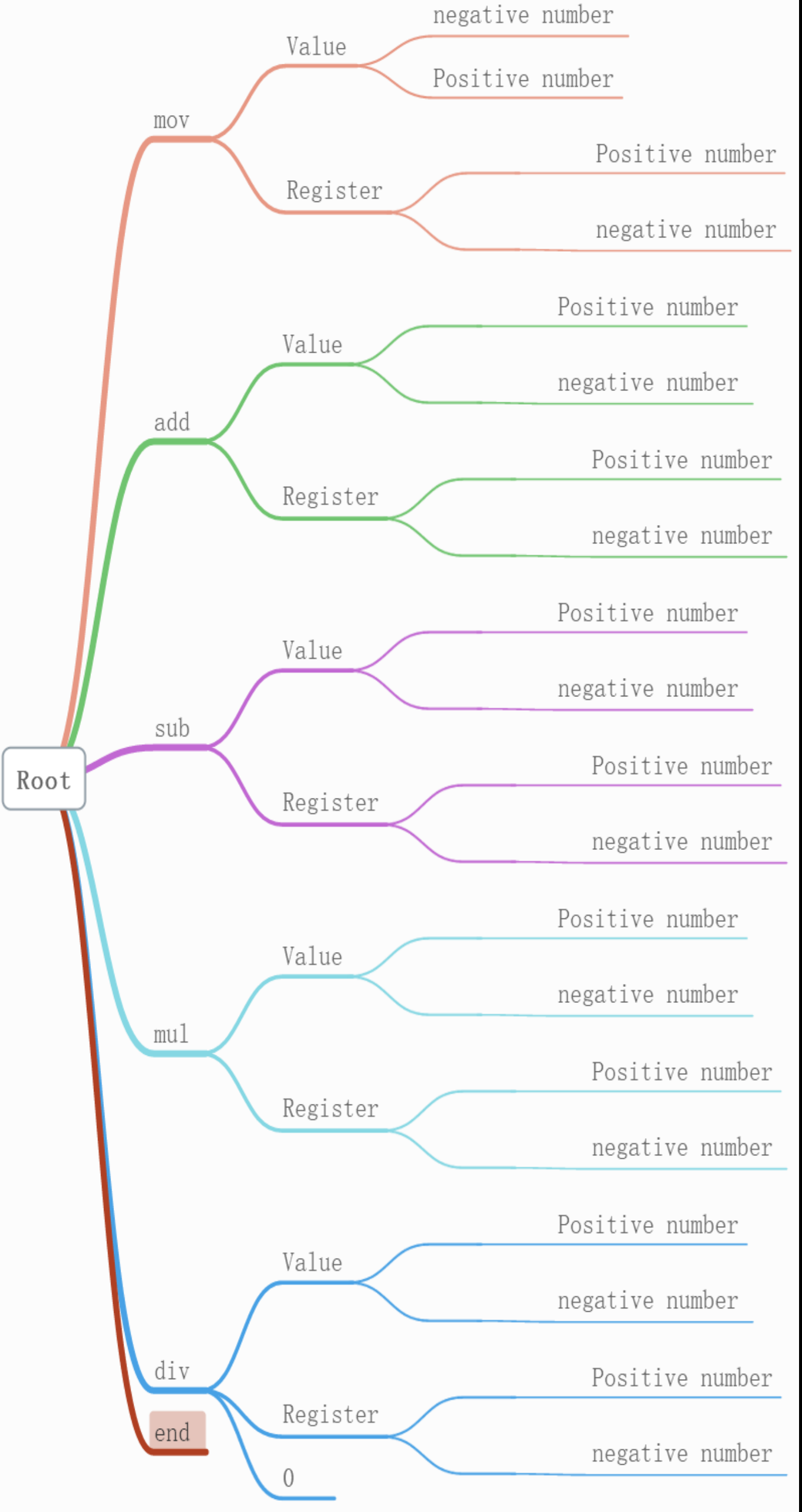
4-bit opcode, 3-bit operand, 17-bit data
Calculate and store positive and negative numbers
Floats can be stored but the fractional part is discarded
Total 24-bit



Opcode	Binary	Operand	Binary
mov	0001	r0	000
add	0010	r1	001
sub	0011	r2	010
mul	0100	r3	011
div	0101	r4	100
end	0000	r5	101
		r6	110

Initialization

```
r0 0000000000000000
r1 0000000000000000
r2 0000000000000000
r3 0000000000000000
r4 0000000000000000
r5 0000000000000000
r6 0000000000000000
rm 00000000000000000000000000000000
re 0000000000000000
```


Root
mov
Value
negative number
Positive number
Register
Positive number
negative number
add
Value
Positive number
negative number
Register
Positive number
negative number
sub
Value
Positive number
negative number
Register
Positive number
negative number
mul
Value
Positive number
negative number
Register
Positive number
negative number
div
Value
Positive number
negative number
Register
Positive number
negative number
0
end

02
Module

Introduce the design and implementation of key components

```
string[,] opcode = { { "mov", "0001" }, { "add", "0010" }, { "sub", "0011" }, { "mul", "0100" }, { "div", "0101" }, { "e
string[,] operand = { { "r0", "000" }, { "r1", "001" }, { "r2", "010" }, { "r3", "011" }, { "r4", "100" }, { "r5", "101"
string[,] values = { { "r0", "0" }, { "r1", "0" }, { "r2", "0" }, { "r3", "0" }, { "r4", "0" }, { "r5", "0" }, { "r6", "
string[] opcode1 = { "mov", "add", "sub", "mul", "div", "end" };
string[] opcode2 = { "0001", "0010", "0011", "0100", "0101", "0000" };
string[] operand1 = { "r0", "r1", "r2", "r3", "r4", "r5", "r6", "r7" };
string[] operand2 = { "000", "001", "010", "011", "100", "101", "110", "111" };
string[] values1 = { "r0", "r1", "r2", "r3", "r4", "r5", "r6", "r7" };
string[] values2 = { "0", "0", "0", "0", "0", "0", "0", "0" };
int[] round = { 0, 0, 0, 0, 0 };
int[] roundx = { 0, 0, 0, 0, 0 };
int[] intvalues = { 0, 0, 0, 0, 0, 0, 0, 0 };
```

Initialize the operands and operators, and the value of a register, because the flexibility of the two-dimensional array is worse than the one-dimensional array, so the binary operator is used to synchronize the binary operator.

```
for (int j = 0; j < svc.GetLength(0); j++)
{
    pc = "PC[" + ii + "] ";
    instr = svc[j, 0] + " " + svc[j, 1] + " " + svc[j, 2];
    binstr = bincodes[j, 0] + " " + bincodes[j, 1] + " " + bincodes[j, 2];
    int y = Array.IndexOf(opcode1, svc[j, 0]);
    clk = round[y].ToString();
    strtump = pc + "      " + instr + "          " + binstr + "                    " + clk;
    prints[ii] = strtump;
    ii = ii + 1;

}
```

Initialize the operands and operators, and the value of a register, because the flexibility of the two-dimensional array is worse than the one-dimensional array, so the binary operator is used to synchronize the binary operator.From the original code to resolve the operator and the operand has a third bit of uncertainty data. Calculate the number of instructions and all into binary code.

```
for (int i = 0; i < codes.GetLength(0); i++)
{
    if (codes[i, 0] == "mov")
    {
        if (System.Text.RegularExpressions.Regex.IsMatch(codes[i, 2], @"^[-]?[0-9]+$"))
        {
            s = codes[i, 2];
            int num = Array.IndexOf(operand1, codes[i, 1]);
            int s0 = Convert.ToInt32(s);
            s = ConvertBits(s0).ToString();
            values2[num] = s;
            nus = nus + 1;
            culvalue(values2, intvalues);
            for (int ix = 0; ix < values1.Length; ix++)
            {
                if (ix == num)
                {
                    Console.WriteLine("{0} = {1} [{2}]", values1[ix], intvalues[ix], values2[ix]);
                }
                else
                {
                    // Console.WriteLine("{0} : [{1}] : {2}", values1[ix], values2[ix], intvalues[ix]);
                }

            }
            continue;
        }
        else
        {
            int num = Array.IndexOf(operand1, codes[i, 2]);
            int num0 = Array.IndexOf(operand1, codes[i, 1]);
            values2[num0] = values2[num];
            nus = nus + 1;
            culvalue(values2, intvalues);
            for (int ix = 0; ix < values1.Length; ix++)
            {
                if (ix == num)
                {
                    Console.WriteLine("{0}={1} [{2}]", values1[ix], intvalues[ix], values2[ix]);
                }
                else
                {
                    // Console.WriteLine("{0} : [{1}] : {2}", values1[ix], values2[ix], intvalues[ix]);
                }

            }
            continue;
        }
    }
```

ACTION: ASSIGNMENT

Through the regular expression to determine whether the second operand is a number or a register name, if it is a number of it into a binary into the first operand shown in the register, before that positive or negative, And then operate

ACTION: ADDITION

```
if (codes[i, 0] == "add")
{
    if (System.Text.RegularExpressions.Regex.IsMatch(codes[i, 2], @"^[-]?[0-9]+$"))
    {
        int num = Array.IndexOf(operand1, codes[i, 1]);
        s = codes[i, 2];
        int s0 = Convert.ToInt32(s);
        string stru = values2[num];
        if (stru.Length <= 32)
        {

            for (int sn = stru.Length; sn < 32; sn++)
            {
                if (stru[0] == '1')
                {
                    stru = '1' + stru;
                }
                else
                {
                    stru = '0' + stru;
                }

            }
        }
        int s1 = Convert.ToInt32(stru, 2);
        s1 = s1 + s0;
        s = ConvertBits(s1).ToString();
        values2[num] = s;
        nus = nus + 1;
        culvalue(values2, intvalues);
        for (int ix = 0; ix < values1.Length; ix++)
        {
            if (ix == num)
            {
                Console.WriteLine("{0}={1} [{2}]", values1[ix], intvalues[ix], values2[ix]);
            }
            else
            {
                // Console.WriteLine("{0} : [{1}] : {2}", values1[ix], values2[ix], intvalues[ix]);
            }

        }
        continue;
    }
    else
    {
        int num1 = Array.IndexOf(operand1, codes[i, 1]);
        int num2 = Array.IndexOf(operand1, codes[i, 2]);
        string str1 = values2[num1];
        string str2 = values2[num2];
        int s1 = Convert.ToInt32(str1, 2);
        int s2 = Convert.ToInt32(str2, 2);
        s1 = s1 + s2;
        s = ConvertBits(s1).ToString();
        values2[num1] = s;
        nus = nus + 1;
        culvalue(values2, intvalues);
        for (int ix = 0; ix < values1.Length; ix++)
        {
```

```
if (codes[i, 0] == "sub")
{
    if (System.Text.RegularExpressions.Regex.IsMatch(codes[i, 2], @"^[-]?[0-9]+$"))
    {
        int num = Array.IndexOf(operand1, codes[i, 1]);
        s = codes[i, 2];
        int s0 = Convert.ToInt32(s);
        string stru = values2[num];
        if (stru.Length <= 32)
        {
            for (int sn = stru.Length; sn < 32; sn++)
            {
                if (stru[0] == '1')
                {
                    stru = '1' + stru;
                }
                else
                {
                    stru = '0' + stru;
                }

            }
        }
        int s1 = Convert.ToInt32(stru, 2);
        s1 = s1 - s0;
        s = ConvertBits(s1).ToString();
        values2[num] = s;
        nus = nus + 1;
        culvalue(values2, intvalues);
        for (int ix = 0; ix < values1.Length; ix++)
        {
            if (ix == num)
            {
                Console.WriteLine("{0}={1} [{2}]", values1[ix], intvalues[ix], values2[ix]);
            }
            else
            {
                // Console.WriteLine("{0} : [{1}] : {2}", values1[ix], values2[ix], intvalues[ix]);
            }

        }
        continue;
    }
```

ACTION: SUBTRACTION

```
else
{
    int num1 = Array.IndexOf(operand1, codes[i, 1]);
    int num2 = Array.IndexOf(operand1, codes[i, 2]);
    string str1 = values2[num1];
    string str2 = values2[num2];
    int s1 = Convert.ToInt32(str1, 2);
    int s2 = Convert.ToInt32(str2, 2);
    s1 = s1 - s2;
    s = ConvertBits(s1).ToString();
    values2[num1] = s;
    nus = nus + 1;
    culvalue(values2, intvalues);
    for (int ix = 0; ix < values1.Length; ix++)
    {
        if (ix == num1)
        {
```

ACTION: MULTIPLICATION

```
rm = ssr;
values2[num] = s;
nus = nus + 1;
culvalue(values2, intvalues);
for (int ix = 0; ix < values1.Length; ix++)
{
    if (ix==num)
    {
        Console.WriteLine("rm:{0}={1} [{2}]", values1[ix],intvalues[ix],rm);
    }
    else {
        //  Console.WriteLine("{0} : [{1}] : {2}", values1[ix], values2[ix], intvalues[ix]);
    }

}
continue;
```

```
if (codes[i, 0] == "mul")
{
    if (System.Text.RegularExpressions.Regex.IsMatch(codes[i, 2], @"^[-]?[0-9]+$"))
    {
        int num = Array.IndexOf(operand1, codes[i, 1]);
        s = codes[i, 2];
        int s0 = Convert.ToInt32(s);
        string stru = values2[num];
        if (stru.Length <= 32)
        {
            for (int sn = stru.Length; sn < 32; sn++)
            {
                if (stru[0] == '1')
                {
                    stru = '1' + stru;
                }
                else
                {
                    stru = '0' + stru;
                }
            }
        }
        int s1 = Convert.ToInt32(stru, 2);
        s1 = s1 * s0;
        s = ConvertBits(s1).ToString();
        string ssr = s;
        if (ssr.Length <= 32)
        {
            for (int sn = ssr.Length; sn < 32; sn++)
            {
                if (ssr[0] == '1')
                {
                    ssr = '1' + ssr;
                }
                else
                {
                    ssr = '0' + ssr;
                }
            }
        }
```

```
}
else
{
    int num1 = Array.IndexOf(operand1, codes[i, 1]);
    int num2 = Array.IndexOf(operand1, codes[i, 2]);
    string str1 = values2[num1];
    string str2 = values2[num2];
    int s1 = Convert.ToInt32(str1, 2);
    int s2 = Convert.ToInt32(str2, 2);
    if (s2 == 0)
    {
        Console.WriteLine("0 can not do divisor");
        break;
```

ACTION: DIVISION

```
if (codes[i, 0] == "div")
{
if (System.Text.RegularExpressions.Regex.IsMatch(codes[i, 2], @"^[-]?[0-9]+$"))
{
    int num = Array.IndexOf(operand1, codes[i, 1]);
    s = codes[i, 2];
    int s0 = Convert.ToInt32(s);
    string stru = values2[num];
    if (stru.Length <= 32)
    {
        for (int sn = stru.Length; sn < 32; sn++)
        {
            if (stru[0] == '1')
            {
                stru = '1' + stru;
            }
            else
            {
                stru = '0' + stru;
            }
        }
    }
    int s1 = Convert.ToInt32(stru, 2);
    if (s0 == 0)
    {
        Console.WriteLine("0 can not do divisor");
        break;
    }
    int dc = s1 % s0;
    re = ConvertBits(dc).ToString();
    s1 = s1 / s0;
    s = ConvertBits(s1).ToString();
    values2[num] = s;
    nus = nus + 1;
    culvalue(values2, intvalues);
    for (int ix = 0; ix < values1.Length; ix++)
    {
        if (ix == num)
        {
            Console.WriteLine("{0}={1} [{2}] re={3} [{4}]", values1[ix],  intvalues[ix],values2[ix],dc, re);
        }
        else
        {
            //Console.WriteLine("{0} : [{1}] : {2}", values1[ix], values2[ix], intvalues[ix]);
        }
    }
    continue;
```

NUMBER CONVERSION

```
static StringBuilder ConvertBits(int val)
{
    int bitMask = 1 << 15;
    StringBuilder bitBuffer = new StringBuilder();
    for (int i = 1; i <= 16; i++)
    {
        if ((val & bitMask) == 0)
            bitBuffer.Append("0");
        else
            bitBuffer.Append("1");
        val <<= 1;
    }
    return bitBuffer;
}
```

THANKS FOR WATCHING

REPORTER : GaoYang