Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОССУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОННИКИ

КАФЕДРА ИНФОРМАТИКИ

Пояснительная записка к лабораторным работам №1-3

По дисциплине «Архитектуры вычислительных систем»

По теме «Арифметические операции с целыми числами»

Выполнил:

студент гр. 653505

Хлопцев А. А.

Проверил:

Шиманский В. В.

Минск 2018

**Оглавление**

[Структурная схема проекта 3](#_Toc507704866)

[Основные классы. 4](#_Toc507704867)

[Основные классы. Встроенные в них функции и их краткое описание 4](#_Toc507704868)

[Program 5](#_Toc507704869)

[CPU 5](#_Toc507704870)

[Register 5](#_Toc507704871)

[ALU 5](#_Toc507704872)

[Checker 6](#_Toc507704873)

[Visual 6](#_Toc507704874)

[Программный код 7](#_Toc507704875)

[Program 7](#_Toc507704876)

[CPU 9](#_Toc507704877)

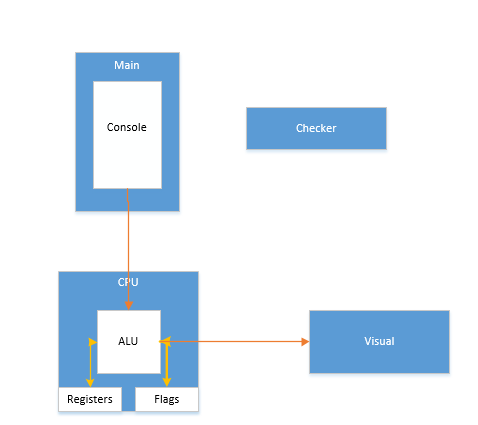
[Register 11](#_Toc507704878)

[ALU 12](#_Toc507704879)

[Checker 19](#_Toc507704880)

[Visual 23](#_Toc507704881)

# Структурная схема проекта



Main – главное приложение

Console – консоль ввода команд

Checker – приводит все запросы к стандартному формату. Проверяет корректность данных

CPU – хранит регистры и флаги

ALU – осуществляет работу над данными из регистров:

* Принимает команду из консоли
* Читает необходимые данные с регистров
* Выполняет арифметическое действие
* Возвращает результат в регистры
* Передаёт команду дальше в блок Visual

Visual – визуализирует команду. Представляет арифметическое действие в бинарном формате и выводит результат на консоль.

# Основные классы.

## Основные классы. Встроенные в них функции и их краткое описание

Жирным цветом обозначены основные функции (public, общий доступ)

Обычным цветом обозначены закрытые вспомогательные функции (private, доступ только отдельным элементам)

1. Program
   1. Main
   2. MyConsole
   3. MyConsoleCommand
   4. MyCommand
2. CPU
   1. Status
   2. ClearFlags
   3. Clear
   4. CreateFlags
3. Register
   1. Info
   2. Clear
   3. ReturnNumber(2 перегрузки)
4. ALU
   1. MOV
   2. SUB
   3. ADD
   4. MUL
   5. DIV
   6. TypeOfWord
   7. NumberInRegister
   8. RegisterInRegister
   9. CopyRegister
   10. AddNumberInRegister
   11. AddNumberInSmallRegister
   12. SUBforDIV
5. Checker
   1. Print
   2. IsConsoleCommand
   3. FormattingMyCommand
   4. FormattingMyConsoleCommand
   5. CheckCommandMOVandADD
   6. CheckCommandMULandDIV
   7. IsRegister
   8. IsNumber
   9. CheckNumber
   10. IsNagativResualtForADD
   11. IsNegativResultForMULL
   12. GiveMeMyBinaryValue
   13. ConvertBinaryValueForDIV
   14. GiveMeMyBinaryValueForMUL
   15. GiveMeMyNagativeValue
   16. NEG
   17. Equals
6. Visual
   1. VisualCommandADD
   2. VisualCommandMUL
   3. VisualIfNothingNot
   4. VisualCommandDIV

## Program

Отвечает за консоль, ввод и вывод команд.

Main – точка входа в программу, никакого доп. функционала.

MyConsole – консоль вывода программы

MyConsoleCommand – запуск, если в консоль задаётся консольная программа

MyCommand – запуск, если в консоль задаётся команда на языке assembler

## CPU

Хранит информацию о процессоре, его регистрах и флагах

Status – выводит информацию о данных в регистрах и флагах

ClearFlags – сбрасывает все флаги

Clear – сбрасывает все регистры, сбрасывает все флаги

CreateFlags – изменяет флаги в зависимости от данных в регистрах и команды Assembler

## Register

Регистр, содержит 2 младших регистра, и байт знака

Info – выводит информацию о регистре

Clear – сбрасывает информацию в старшем, младшем регистре и байт знака

ReturnNumber – возвращает результат регистра, функция без параметра вернёт результат всего регистра, с параметром вернёт значение старшего или младшего регистра

## ALU

Арифметическо-логическое устройство отвечает за все команды Assembler, берёт информацию из CPU, обрабатывает её и возвращает обратно.

MOV, SUB, ADD, MUL, DIV – выполнение обычных команд Assembler

TypeOfWord – определяет старший, младший, общий или цифра была передана в качестве параметра команде

NumberInRegister – кладёт число в общий регистр

RegisterInRegister – перемещает данные с одного регистра в другой (все виды регистров)

CopyRegister – копирует значения одного общего регистра во второй

AddNumberInRegister – добавляет какое-то численное значение к значению в общем регистре, проверка переполнения

AddNumberInSmallRegister – добавляет какое-то численное значение к значению в малом регистре (l, h), проверка переполнения

SubForDiv – специальная функция для вычисления промежуточных результатов операции DIV

## Checker

Отвечает за преобразование запросов между блоками и проверку значений

Print – вывод информации об эмуляторе посреди консоли

IsConsoleCommand – проверка, является ли данная команда консольной

FormattingMyConsoleCommand – приведение консольных команд к общему формату

FormattingMyCommand – приведение Assembler команд к общему формату

CheckCommandMOVandAdd – проверка корректности задания команд ADD и MOV

CheckCommandMULandDIv – проверка корректности задания команд MUL и DIV

IsRegister – проверка, является ли данная строка регистром

IsNumber – проверка является ли данная строка числом

CheckNumber – проверка, входит ли число в диапазон регистра

IsNegativResualtForADD – проверка, даст ли сложение отрицательный результат

IsNegativResualtForMULL – проверка, даст ли произведение отрицательный результат

GiveMeMyBinaryValue – конвентирует число в 2 систему для положительных или инвентированую бинарку для отрицательных

ConvertBinaryValueForDIV – перевод 2х переменных команды DIV в двоичную систему

GiveMeMyBinaryValueForMUL - конвентирует число в 2 систему для положительных или

дополненное до 1 бинарную строку для отрицательных чисел

NEG – инвентирует бинарное число с дополнение до единицы без добавление ведущих байтов

GiveMeMyNagativeValue - инвентирует бинарное число с дополнение до единицы без добавление ведущих байтов и переводит результат в 10 систему

Equals – особая функция для сравнения промежуточных значений в функции DIV

## Visual

VisualCommandADD – визуализирование команды ADD

VisualCommandMUL – визуализирование команды MUL

VisualCommandDIV – визуализирование команды DIV, когда первое значение по модулю больше второго

VisualIfNothingNot - визуализирование команды DIV, когда второе значение по модулю больше первого

# Программный код

## Program

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LabWorkABC1\_3

{

class Program

{

static void Main(string[] args)

{

Checker.Print("Emulator ALU v. 1.0.0");

MyConsole();

}

private static void MyConsole()

{

while(true)

{

Console.Write(".");

string command = Console.ReadLine();

if (command != "")

{

if (Checker.IsConsoleCommand(command) == true) MyConsoleCommand(command);

else

MyCommand(command);

}

}

}

private static void MyConsoleCommand(string command)

{

command = Checker.FormattingMyConsoleCommand(command);

switch(command)

{

case "status":

CPU.Status();

break;

case "clear":

CPU.Clear();

break;

case "exit":

Environment.Exit(0);

break;

default:

Console.WriteLine("Error console command!");

break;

}

}

private static void MyCommand(string command)

{

string[] words = Checker.FormattingMyCommand(command);

switch(words[0])

{

case "mov":

if (!Checker.CheckCommandMOVandADD(words))

Console.WriteLine("Error entering command MOV");

else

ALU.MOV(words[1], words[2]);

break;

case "add":

CPU.ClearFlags();

if (!Checker.CheckCommandMOVandADD(words))

Console.WriteLine("Error entering command ADD");

else

ALU.ADD(words[1], words[2]);

break;

case "sub":

CPU.ClearFlags();

if (!Checker.CheckCommandMOVandADD(words))

Console.WriteLine("Error entering command SUB");

else

ALU.SUB(words[1], words[2]);

break;

case "mul":

CPU.ClearFlags();

if (!Checker.CheckCommandMULandDIV(words))

Console.WriteLine("Error entering command MUL");

else

ALU.MUL(words[1]);

break;

case "div":

CPU.ClearFlags();

if (!Checker.CheckCommandMULandDIV(words))

Console.WriteLine("Error entering command DIV");

else

ALU.DIV(words[1]);

break;

default:

Console.WriteLine("Error command!");

break;

}

}

}

}

## CPU

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LabWorkABC1\_3

{

static class CPU

{

public static Hashtable registors;

public static Dictionary<string, bool> flags;

static string[] nameflags = { "OF", "SF", "ZF", "AF", "PF", "CF" };

static CPU()

{

Register a = new Register("A");

Register b = new Register("B");

Register c = new Register("C");

Register d = new Register("D");

registors = new Hashtable

{

{ "a", a },

{ "b", b },

{ "c", c },

{ "d", d }

};

flags = new Dictionary<string, bool>();

foreach(var name in nameflags)

{

flags.Add(name, false);

}

}

public static void Status()

{

foreach(DictionaryEntry reg in registors)

{

(reg.Value as Register).Info();

}

Console.WriteLine();

foreach (var fg in flags)

{

Console.Write("{0}={1} ", fg.Key, Convert.ToSByte(fg.Value));

}

Console.Write("\n\n");

}

public static void ClearFlags()

{

foreach (var name in nameflags)

{

flags[name] = false;

}

}

public static void Clear()

{

foreach (DictionaryEntry reg in registors)

{

(reg.Value as Register).Clear();

}

ClearFlags();

}

public static void CreateFlags(long number, bool isSmall)

{

if (number == 0)

flags["ZF"] = true;

if (!isSmall && number > 65535)

flags["CF"] = true;

else

if (isSmall && number > 256)

flags["CF"] = true;

int countone = Convert.ToString(number, 2).Count(x => x == '1');

if (countone % 2 == 0)

flags["PF"] = true;

if (number < 0)

flags["SF"] = true;

}

}

}

## Register

class Register

{

public int h, l;

public bool neg;

string name;

public void Info()

{

Console.WriteLine(name + "X");

Console.WriteLine("{0} {1} = {2}", h.ToString("X2"), l.ToString("X2"), ReturnNumber());

}

public void Clear()

{

h = 0;

l = 0;

neg = false;

}

public Register(string name)

{

neg = false;

h = 0;

l = 0;

this.name = name;

}

public int ReturnNumber()

{

string val = h.ToString("X2") + l.ToString("X2");

if (!neg)

return Convert.ToInt32(val, 16);

else

return Convert.ToInt32(val, 16) - 65535 - 1;

}

public int ReturnNumber(string word)

{

switch(word[1])

{

case 'x':

return ReturnNumber();

case 'h':

return h;

default:

return l;

}

}

}

## ALU

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LabWorkABC1\_3

{

static class ALU

{

public static void MOV(string val1, string val2)

{

if (TypeOfWord(val2) == 0)

{

bool error = false;

int number = Checker.CheckedNumber(val2, ref error);

if (error == true)

return;

byte typereg = TypeOfWord(val1);

switch(typereg)

{

case 1:

NumberInRegister(CPU.registors[val1[0].ToString()] as Register, number);

break;

default:

NumberInSmallRegister(val1, number);

break;

}

}

else

{

RegisterInRegister(val1, val2);

}

}

private static byte TypeOfWord(string word)

{

if (Checker.IsNumber(word)) return 0;

switch (word[1])

{

case 'x':

return 1;

case 'h':

return 2;

default:

return 3;

}

}

private static void NumberInRegister(Register reg, long number)

{

//Register reg = CPU.registors[namereg] as Register;

string hexstr = number.ToString("X4");

hexstr = hexstr.Substring(hexstr.Length - 4);

if (number < 0)

reg.neg = true;

else

reg.neg = false;

reg.h = Convert.ToInt32(hexstr.Substring(0, 2), 16);

reg.l = Convert.ToInt32(hexstr.Substring(2), 16);

}

private static void NumberInSmallRegister(string namereg, int number)

{

Register reg = CPU.registors[namereg[0].ToString()] as Register;

if (Math.Abs(number) > 256)

{

Console.WriteLine("Constant too large!");

return;

}

string hexstr = number.ToString("X4");

if (namereg[1] == 'h')

reg.h = Convert.ToInt32(hexstr.Substring(hexstr.Length - 2, 2), 16);

else

reg.l = Convert.ToInt32(hexstr.Substring(hexstr.Length - 2, 2), 16);

}

private static void RegisterInRegister(string namefirstreg, string namesecondreg)

{

Register first = CPU.registors[namefirstreg[0].ToString()] as Register;

Register second = CPU.registors[namesecondreg[0].ToString()] as Register;

byte typefirst = TypeOfWord(namefirstreg);

byte typesecond = TypeOfWord(namesecondreg);

switch (typefirst)

{

case 1:

switch(typesecond)

{

case 1:

CopyRegister(first, second);

break;

default:

Console.WriteLine("Operand types do not match!");

break;

}

break;

case 2:

switch (typesecond)

{

case 2:

first.h = second.h;

break;

case 3:

first.h = second.l;

break;

default:

Console.WriteLine("Operand types do not match!");

break;

}

break;

case 3:

switch (typesecond)

{

case 2:

first.l = second.h;

break;

case 3:

first.l = second.l;

break;

default:

Console.WriteLine("Operand types do not match!");

break;

}

break;

}

}

private static void CopyRegister(Register first, Register second)

{

first.h = second.h;

first.l = second.l;

first.neg = second.neg;

}

public static void SUB(string val1, string val2)

{

if (TypeOfWord(val2) == 0)

{

val2 = "-" + val2;

ADD(val1, val2);

}

else

{

int v = (CPU.registors[val2[0].ToString()] as Register).ReturnNumber(val2);

NumberInRegister(CPU.registors[val2[0].ToString()] as Register, -v);

ADD(val1, val2);

NumberInRegister(CPU.registors[val2[0].ToString()] as Register, v);

}

}

public static void ADD(string val1, string val2)

{

if (TypeOfWord(val2) == 0)

{

bool error = false;

int number = Checker.CheckedNumber(val2, ref error);

if (error == true)

return;

byte typereg = TypeOfWord(val1);

int val = (CPU.registors[val1[0].ToString()] as Register).ReturnNumber(val1);

Visual.VisualCommandADD(val1, "number", val, Convert.ToInt32(val2));

switch (typereg)

{

case 1:

AddNumberInRegister(CPU.registors[val1[0].ToString()] as Register, number);

break;

default:

AddNumberInSmallRegister(val1, number, true);

break;

}

}

else

{

bool ok = true;

int v1 = (CPU.registors[val1[0].ToString()] as Register).ReturnNumber(val1);

int v2 = (CPU.registors[val2[0].ToString()] as Register).ReturnNumber(val2);

AddRegisterInRegister(val1, val2, ref ok);

if (ok)

Visual.VisualCommandADD(val1, val2, v1, v2);

}

}

private static void AddNumberInRegister(Register reg, int number)

{

int answer = reg.ReturnNumber() + number;

CPU.CreateFlags(answer, false);

NumberInRegister(reg, answer);

}

private static void AddNumberInSmallRegister(string namereg, int number, bool isNumber)

{

Register reg = CPU.registors[namereg[0].ToString()] as Register;

if (isNumber && Math.Abs(number) > 256)

{

Console.WriteLine("Constant too large!");

return;

}

if (namereg[1] == 'h')

number += reg.h;

else

number += reg.l;

CPU.CreateFlags(number, true);

string hexstr = number.ToString("X4");

if (namereg[1] == 'h')

reg.h = Convert.ToInt32(hexstr.Substring(hexstr.Length - 2, 2), 16);

else

reg.l = Convert.ToInt32(hexstr.Substring(hexstr.Length - 2, 2), 16);

}

private static void AddRegisterInRegister(string namefirstreg, string namesecondreg,ref bool ok)

{

Register first = CPU.registors[namefirstreg[0].ToString()] as Register;

Register second = CPU.registors[namesecondreg[0].ToString()] as Register;

byte typefirst = TypeOfWord(namefirstreg);

byte typesecond = TypeOfWord(namesecondreg);

switch (typefirst)

{

case 1:

switch (typesecond)

{

case 1:

AddNumberInRegister(first, second.ReturnNumber());

break;

default:

ok = false;

Console.WriteLine("Operand types do not match!");

break;

}

break;

case 2:

switch (typesecond)

{

case 2:

AddNumberInSmallRegister(namefirstreg, second.h, false);

break;

case 3:

AddNumberInSmallRegister(namefirstreg, second.l, false);

break;

default:

ok = false;

Console.WriteLine("Operand types do not match!");

break;

}

break;

case 3:

switch (typesecond)

{

case 2:

AddNumberInSmallRegister(namefirstreg, second.h, false);

break;

case 3:

AddNumberInSmallRegister(namefirstreg, second.l, false);

break;

default:

ok = false;

Console.WriteLine("Operand types do not match!");

break;

}

break;

}

}

public static void MUL(string reg)

{

Visual.VisualCommandMUL(reg);

Register mainReg = CPU.registors["a"] as Register;

int val1 = (CPU.registors["a"] as Register).ReturnNumber();

int val2 = (CPU.registors[reg[0].ToString()] as Register).ReturnNumber(reg);

long ans = val1 \* val2;

string s;

if (Math.Abs(ans) > 65535)

s = ans.ToString("X8");

else

s = ans.ToString("X4");

if (s.Length > 8)

s = s.Substring(s.Length - 8);

if (s.Length < 5)

{

ToReg(mainReg, s);

if (mainReg.h != 0)

CPU.flags["OF"] = true;

(CPU.registors["d"] as Register).Clear();

}

else

{

ToReg(mainReg, s.Substring(4));

ToReg(CPU.registors["d"] as Register, s.Substring(0, 4));

if (ans < 0)

(CPU.registors["d"] as Register).neg = true;

if ((CPU.registors["d"] as Register).ReturnNumber() != 0)

CPU.flags["OF"] = true;

}

CPU.CreateFlags(ans, false);

}

public static void DIV(string reg)

{

byte type = TypeOfWord(reg);

if (type == 1)

{

Register reg1 = CPU.registors["a"] as Register;

Register reg2 = CPU.registors["d"] as Register;

string first;

if (reg2.neg)

first = (-reg2.ReturnNumber()).ToString("X4");

else

first = reg2.ReturnNumber().ToString("X4");

bool isNeg = false;

if (reg1.neg || reg2.neg)

isNeg = true;

long v = reg1.ReturnNumber();

if (v < 0) v = -v;

long val = Convert.ToInt64(first + v.ToString("X4"), 16);

if (isNeg) val = -val;

Register reg3 = CPU.registors[reg[0].ToString()] as Register;

long valD = reg3.ReturnNumber();

if (valD == 0)

{

Console.WriteLine("Divide by zero");

return;

}

Visual.VisualCommandDIV(val, valD, reg);

NumberInRegister(reg1, val / valD);

NumberInRegister(reg2, val % valD);

if (isNeg)

{

reg1.neg = true;

reg2.neg = true;

}

else

{

reg1.neg = false;

reg2.neg = false;

}

}

else

if(type > 1)

{

Register first = CPU.registors["a"] as Register;

int val2 = (CPU.registors[reg[0].ToString()] as Register).ReturnNumber(reg);

int val1 = first.ReturnNumber();

if (val2 == 0)

{

Console.WriteLine("Divide by zero");

return;

}

first.l = val1 / val2;

first.h = val1 % val2;

if (first.l < 0) first.l = -first.l;

if (first.h < 0) first.h = -first.h;

if (val1 \* val2 < 0)

first.neg = true;

else

first.neg = false;

Visual.VisualCommandDIV(val1, val2, reg);

}

}

public static string SUBforDiv(string val1, string val2)

{

if (val1.Length > val2.Length)

val2 = "0" + val2;

string s = "";

StringBuilder ans = new StringBuilder(new string('0', val1.Length));

int over = 0;

val2 = Checker.NEG(val2);

for (int i = val1.Length - 1; i > -1; i--)

{

if (val1[i] == '1' && val2[i] == '1')

{

if (over == 0)

ans[i] = '0';

else

ans[i] = '1';

over = 1;

}

else

if (val1[i] == '0' && val2[i] == '0')

{

if (over == 0)

ans[i] = '0';

else

ans[i] = '1';

over = 0;

}

else

{

if (over == 0)

{

ans[i] = '1';

over = 0;

}

else

{

ans[i] = '0';

over = 1;

}

}

}

s = ans.ToString();

s = s.TrimStart('0');

return (s == "" ? "0" : s);

}

private static void ToReg(Register reg, string s)

{

reg.Clear();

reg.h = Convert.ToInt32(s.Substring(0, 2), 16);

reg.l = Convert.ToInt32(s.Substring(s.Length / 2), 16);

}

}

}

## Checker

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LabWorkABC1\_3

{

static class Checker

{

public static void Print(string text)

{

var width = Console.WindowWidth;

var padding = width / 2 + text.Length / 2;

Console.WriteLine("{0," + padding + "}", text);

}

public static bool IsConsoleCommand(string text)

{

if (text[0] == '>')

return true;

else

return false;

}

public static string[] FormattingMyCommand(string command)

{

command = command.ToLower();

string[] words = command.Split(new char[] { ' ', ',' }, StringSplitOptions.RemoveEmptyEntries);

return words;

}

public static string FormattingMyConsoleCommand(string command)

{

command = command.Remove(0, 1);

command = command.Replace(" ", "");

command = command.ToLower();

return command;

}

public static bool CheckCommandMOVandADD(string[] words)

{

if (words.Length != 3)

return false;

if (IsRegister(words[1]) == false)

return false;

if (IsNumber(words[2]) == false && IsRegister(words[2]) == false)

return false;

return true;

}

public static bool CheckCommandMULandDIV(string[] words)

{

if (words.Length != 2)

return false;

if (IsRegister(words[1]) == false)

return false;

return true;

}

public static bool IsRegister(string text)

{

if (text.Length != 2)

return false;

if ((text[0] == 'a' || text[0] == 'b' || text[0] == 'c' || text[0] == 'd') && (text[1] == 'x' || text[1] == 'h' || text[1] == 'l'))

return true;

else

return false;

}

public static bool IsNumber(string text)

{

return int.TryParse(text, out int number);

}

public static int CheckedNumber(string text, ref bool ok)

{

int number = int.Parse(text);

if (number > 65535 || number < -32767)

{

Console.WriteLine("Out range value!");

ok = true;

return 0;

}

else

return number;

}

public static bool IsNegativResultForADD(int val1, int val2)

{

if (val1 < 0 && val2 < 0)

return true;

if (val1 < 0 && -val1 > val2)

return true;

if (val2 < 0 && -val2 > val1)

return true;

return false;

}

public static bool IsNegativeResultForMUL(long val1, long val2)

{

if (val1 \* val2 > 0)

return false;

else

return true;

}

public static string GiveMeMyBinaryValue(int val)

{

if (val >=0)

return Convert.ToString(val, 2);

StringBuilder s = new StringBuilder(Convert.ToString(-val, 2));

for (int i = 0; i < s.Length; i++)

{

if (s[i] == '1')

s[i] = '0';

else

s[i] = '1';

}

return new string('1', 17 - s.Length) + s.ToString();

}

public static void ConvertBinaryValueForDIV(long val1, long val2, ref string first, ref string second)

{

first = Convert.ToString(Math.Abs(val1), 2);

second = Convert.ToString(Math.Abs(val2), 2);

}

private static string GiveMeMyBinaryValueForMUL(int val)

{

if (val >= 0)

return Convert.ToString(val, 2);

StringBuilder s = new StringBuilder(Convert.ToString(-val, 2));

for (int i = 0; i < s.Length; i++)

{

if (s[i] == '1')

s[i] = '0';

else

s[i] = '1';

}

int k = s.Length - 1;

while (s[k] == '1')

s[k--] = '0';

if (k >= 0)

s[k] = '1';

else

return "1" + s.ToString();

return s.ToString();

}

public static int GiveMeMyNegativeValue(string val)

{

StringBuilder s = new StringBuilder(val);

for (int i = 0; i < s.Length; i++)

{

if (s[i] == '1')

s[i] = '0';

else

s[i] = '1';

}

return -Convert.ToInt32(s.ToString(), 2);

}

public static string NEG(string val)

{

StringBuilder s = new StringBuilder(val);

for (int i = 0; i < s.Length; i++)

{

if (s[i] == '1')

s[i] = '0';

else

s[i] = '1';

}

int k = s.Length - 1;

while (s[k] == '1')

s[k--] = '0';

if (k >= 0)

s[k] = '1';

else

return "1" + s.ToString();

return s.ToString();

}

public static bool Equals(string val1, string val2)

{

if (val1.Length < val2.Length)

return false;

if (val1.Length > val2.Length)

return true;

for (int i = 0; i < val1.Length; i++)

if (val1[i] == val2[i])

continue;

else

{

return val1[i] > val2[i];

}

return true;

}

}

}

## Visual

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LabWorkABC1\_3

{

static class Visual

{

public static void VisualCommandADD(string namefirst, string namesecond, int val1, int val2)

{

string first = Checker.GiveMeMyBinaryValue(val1);

string second = Checker.GiveMeMyBinaryValue(val2);

//Console.WriteLine(first + " " + second);

string help = new string('0', Math.Abs(first.Length - second.Length));

if (first.Length > second.Length)

second = help + second;

else

first = help + first;

help = new string('0', Math.Max(first.Length, second.Length) + 1);

StringBuilder ans = new StringBuilder(help);

int i = first.Length;

int over = 0, step = 0;

bool exit = false;

while (true)

{

i--;

if (i < 0) break;

Console.WriteLine("\nStep : {0}\nOverflow = {1}",step, over == 1 ? true : false);

Console.WriteLine("{0, -7}| {1, 20}",namefirst,first);

Console.WriteLine("{0, -7}| {1, 20}",namesecond,second);

Console.WriteLine(new string('-', 29));

Console.WriteLine("answer | {0, 20}", ans);

if (!exit)

if (Console.ReadKey(true).Key == ConsoleKey.Escape)

exit = true;

step++;

if (first[i] == '1' && second[i] == '1' )

{

if (over == 0)

ans[i + 1] = '0';

else

ans[i + 1] = '1';

over = 1;

}

else

if (first[i] == '0' && second[i] == '0')

{

if (over == 0)

ans[i + 1] = '0';

else

ans[i + 1] = '1';

over = 0;

}

else

{

if (over == 0)

{

ans[i + 1] = '1';

over = 0;

}

else

{

ans[i + 1] = '0';

over = 1;

}

}

}

if (!Checker.IsNegativResultForADD(val1, val2) && over == 1) ans[0] = '1';

if ((!Checker.IsNegativResultForADD(val1, val2) && (val1 < 0 || val2 < 0)) || (val1 < 0 && val2 < 0))

{

int j = ans.Length - 1;

while (ans[j] == '1')

ans[j--] = '0';

ans[j] = '1';

}

Console.WriteLine("\n{0, -7}| {1, 20}", namefirst, first);

Console.WriteLine("{0, -7}| {1, 20}", namesecond, second);

Console.WriteLine(new string('-', 29));

Console.WriteLine("answer | {0, 20}\n", ans);

string Ans = ans.ToString();

int len = (namefirst[1] == 'x' ? 16 : 8);

if (Ans.Length > len)

Ans = Ans.Substring(Ans.Length - len);

if (Checker.IsNegativResultForADD(val1, val2))

{

int intans = Checker.GiveMeMyNegativeValue(Ans);

Console.WriteLine("Answer: {0} => {1} = {2} ({3})\n", ans, Ans, intans, intans.ToString("X"));

}

else

Console.WriteLine("Answer: {0} => {1} = {2} ({3})\n", ans, Ans, Convert.ToInt32(Ans, 2), Convert.ToInt32(Ans, 2).ToString("X"));

}

public static void VisualCommandMUL(string val)

{

int val1 = (CPU.registors["a"] as Register).ReturnNumber();

int val2 = (CPU.registors[val[0].ToString()] as Register).ReturnNumber(val);

string first = "", second = "";

Checker.ConvertBinaryValueForDIV(val1, val2, ref first, ref second);

//Console.WriteLine(Convert.ToString(val2, 2));

string help = new string('0', first.Length + second.Length);

StringBuilder ans = new StringBuilder(help);

int len = ans.Length;

int step = 0;

bool exit = false;

Console.WriteLine("{0, -7}| {1}", "AX", first.PadLeft(len));

Console.WriteLine("{0, -7}| {1}", val.ToUpper(), second.PadLeft(len));

Console.WriteLine(new string('-', ans.Length + 9));

int i = second.Length;

//int t = second.Length;

while (true)

{

i--;

//t--;

if (i < 0) break;

help = "";

for (int j = 0; j < first.Length; j++)

if (first[j] == '0' || second[i] == '0')

help += '0';

else

help += '1';

int k = first.Length - 1;

int over = 0, j1 = 0;

for (int j = len - 1; k > -1; j--, k--)

{

int prval = ((ans[j] - '0') + over + (help[k] - '0'));

ans[j] = (char)(prval % 2 + '0');

over = prval / 2;

j1 = j;

}

if (over > 0)

ans[j1 - 1] = '1';

if (!exit)

if (Console.ReadKey(true).Key == ConsoleKey.Escape)

exit = true;

Console.WriteLine("step {0}| {1}", (++step).ToString("D2"), help.PadLeft(len--));

}

Console.WriteLine(new string('-', ans.Length + 9));

Console.WriteLine("{0, -7}| {1}", "answer", ans);

string Ans = ans.ToString();

if (Ans.Length > 16)

Ans = Ans.Substring(Ans.Length - 16);

int x;

if (Checker.IsNegativeResultForMUL(val1, val2))

{

x = val1 \* val2;

Ans = Checker.NEG(Ans);

Console.WriteLine("Answer: {0} => {1} = {2} ({3})\n", ans, Ans, x, x.ToString("X"));

}

else

{

x = Convert.ToInt32(Ans, 2);

Console.WriteLine("Answer: {0} => {1} = {2} ({3})\n", ans, Ans, x, x.ToString("X"));

}

help = x.ToString("X8");

if (help.Length > 8)

help = help.Substring(help.Length - 7);

Console.WriteLine("Result: {0} = {1}({4}), DX = {2}, AX = {3}\n", Ans, x, help.Substring(0, 4), help.Substring(4), help);

}

private static void VisualIfNothingNot(long val1, long val2, string first, string second, string reg)

{

string ans = "0";

string help = first;

long div = Convert.ToInt64(ans, 2);

long mod = Convert.ToInt64(help, 2);

if (val1 / val2 < 0)

{

div = -div;

mod = -mod;

}

Console.WriteLine("{0} = {1}, {2} = {3}", val1, first, val2, second);

Console.WriteLine("Answer = {0} = {1}({2})", ans, div, div.ToString("X4").Substring(div.ToString("X4").Length - 4));

Console.WriteLine("Modulo = {0} = {1}({2})", help, mod, mod.ToString("X4").Substring(mod.ToString("X4").Length - 4));

if (reg[1] == 'l' || reg[1] == 'h')

{

Console.WriteLine("AH = {1}, AL = {0}\n", div.ToString("X2").Substring(div.ToString("X").Length - 4), mod.ToString("X2").Substring(mod.ToString("X2").Length - 4));

}

else

{

Console.WriteLine("AX = {0}, DX = {1}\n", div.ToString("X4").Substring(div.ToString("X4").Length - 4), mod.ToString("X4").Substring(mod.ToString("X4").Length - 4));

}

}

public static void VisualCommandDIV(long val1, long val2, string reg)

{

string first = "", second = "";

bool isNeg = Checker.IsNegativeResultForMUL(val1, val2);

Checker.ConvertBinaryValueForDIV(val1, val2, ref first, ref second);

if (Math.Abs(val1) < Math.Abs(val2))

{

VisualIfNothingNot(val1, val2, first, second, reg);

return;

}

string ans = "";

string help = "";

int step = 0, len = Math.Max(first.Length, second.Length), padleft = 0;

Console.WriteLine("{0, -7}| {1}| {2}{4}{3}", "Step:", first, second, "|Answer:", new string(' ', first.Length \* 2 + 2 - second.Length));

bool exit = false;

int i = 0;

while (true)

{

if (!exit)

if (Console.ReadKey(true).Key == ConsoleKey.Escape)

exit = true;

help += first[i];

int l = help.Length;

help = help.TrimStart('0');

padleft += (l - help.Length);

Console.Write("Step{0} | {1}", (++step).ToString("D2"), (help == "" ? "0" : help).PadLeft(padleft + help.Length));

if (!Checker.Equals(help, second))

{

ans += "0";

ans = ans.TrimStart('0');

}

else

{

ans += "1";

l = help.Length;

help = ALU.SUBforDiv(help, second);

padleft += (l - help.Length);

}

Console.WriteLine("{1}|{0}",ans, new string(' ', first.Length \* 3 - help.Length - padleft + 4));

if (++i == first.Length)

break;

}

if (ans == "")

ans = "0";

if (help == "")

help = "0";

long div = Convert.ToInt64(ans, 2);

long mod = Convert.ToInt64(help, 2);

if (isNeg)

{

div = -div;

mod = -mod;

}

Console.WriteLine("\nAnswer = {0} = {1}({2})", ans, div, div.ToString("X4").Substring(div.ToString("X4").Length - 4));

Console.WriteLine("Modulo = {0} = {1}({2})", help, mod, mod.ToString("X4").Substring(mod.ToString("X4").Length - 4));

if (reg[1] == 'l' || reg[1] == 'h')

{

Console.WriteLine("AH = {1}, AL = {0}", div.ToString("X2").Substring(div.ToString("X").Length - 4), mod.ToString("X2").Substring(mod.ToString("X2").Length - 4));

}

else

{

Console.WriteLine("AX = {0}, DX = {1}", div.ToString("X4").Substring(div.ToString("X4").Length - 4), mod.ToString("X4").Substring(mod.ToString("X4").Length - 4));

}

}

}

}