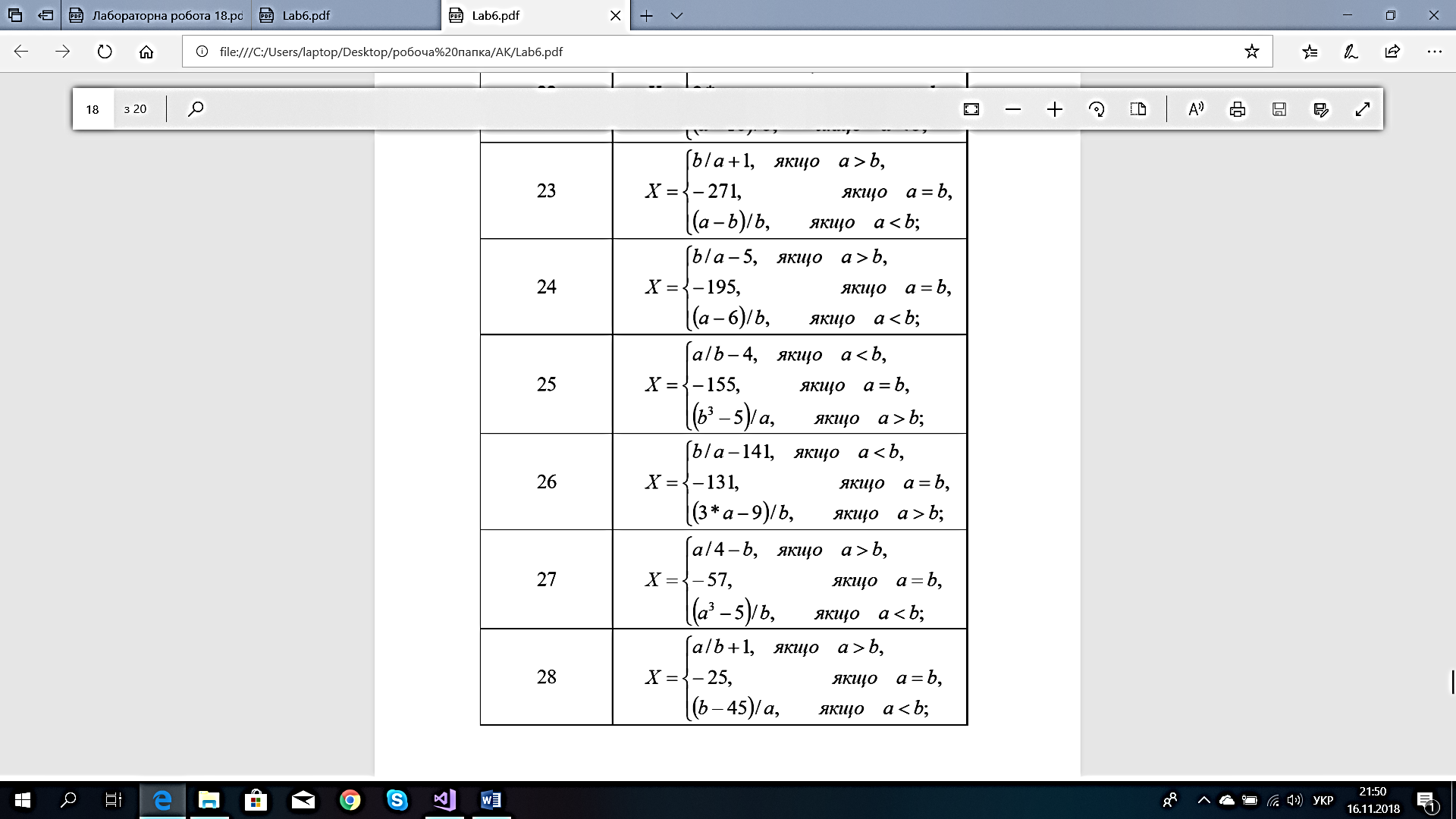
**ЛАБОРАТОРНА РОБОТА № 6**

**ОРГАНІЗАЦІЯ УМОВНИХ ПЕРЕХОДІВ**

***Мета:*** ознайомитися з основними командами мови Assembler для організації умовних переходів; набути практичних навичок в написанні програм з організацією умовних переходів на мові Assembler.

**Хід роботи:**

**Завдання 1**: Написати програму для обчислення заданого умовного цілочисельного виразу для 8 бітних даних, використовуючи команди порівняння, умовного і безумовного переходів. Результат Х – теж цілочисельний і його діапазон (формат) залежить від специфіки вирішуваного умовного виразу. Провести тестові перевірки, відмітити нормальні та аномальні результати. Виконати покрокове виконання асемблерного коду та навести значення регістрів при їх виконанні.



Лістинг програми:

#include "stdafx.h"

#include <iostream>

#include <windows.h>

#include <stdlib.h>

int \_tmain(int argc, TCHAR\* argv[])

{ SetConsoleCP(1251);

SetConsoleOutputCP(1251);

signed char a, b, x\_c, x\_asm, er = 0, over = 0;

printf("Введіть значення з діапазону [-128...127]:\n");

printf("a = "); scanf\_s("%d", &a);

printf("b = "); scanf\_s("%d", &b);

if (a > b)

{ if (a == 0)

{ printf("Помилка: ");

}

else

{ x\_c = b / a - 5;

if (x\_c < -128 || x\_c > 127)

{ printf("Помилка: ");

}

else

{ printf("Результат на мові С X = %d\n", x\_c);

}

}

}

else if (a < b)

{ if (b == 0)

{ printf("Помилка: ");

}

else if ((a - 6) < -128 || (a - 6) > 127)

{ printf("Помилка: ");

}

else

{ x\_c = (a - 6) / b;

printf("Результат на мові С X = %d\n", x\_c);

}

}

else

{ x\_c = -195;

printf("Результат на мові С X = %d\n", x\_c);

}

\_\_asm

{ mov al, a // <al> = a

mov bl, b // <bl> = b

cmp al, bl // порівнюємо значення регістрів <al> та <bl>

jg mark1 // a > b

je mark2 // a = b

jl mark3 // a < b

mark1 :

cmp al, 0; // порівнюємо значення <al> з нулем

je error1; // помилка "ділення на 0"

xchg al, bl; // міняємо місцями значення регістрів <al> і <bl>

cbw;

idiv bl; // <al> = b/a

sub al, 5; // <al> = b/a-5

jo error2; // помилка "переповнення"

mov x\_asm, al; // x\_asm = b/a-5

jmp exit; // переходимо до мітки exit1

mark2:

mov x\_asm, -195; // x\_asm = -195

jmp exit; // переходимо до мітки exit1

mark3:

cmp bl, 0; // порівнюємо значення <bl> з нулем

je error3; // помилка "ділення на 0"

sub al, 6; // <al> = a-6

jo error4; // помилка "переповнення"

cbw; // <dl:al> = a-6

idiv bl; // <al> = (a-6)/b

mov x\_asm, al; // x\_asm = (a-6)/b

jmp exit; // переходимо до мітки exit1

error1:

mov er, 1;

jmp exit;

error2:

mov over, 1;

jmp exit;

error3:

mov er, 1;

jmp exit;

error4:

mov over, 1;

jmp exit;

exit:

}

if (er == 1)

{

printf("ділення на 0!\n");

}

else if (over == 1)

{

printf("переповнення!\n");

}

else if (er == 0 && over == 0)

{

printf("Результат на мові Assembler X = %d\n", x\_asm);

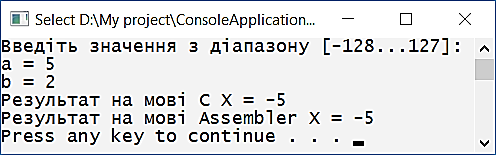
}

system("pause");

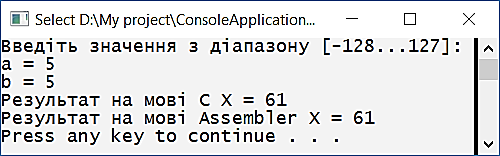
return 0;

}

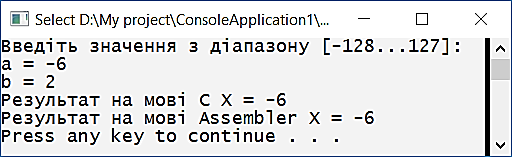
Результат виконання програми:



Перевірка роботи програми для значення a > b



Перевірка роботи програми для значення a = b

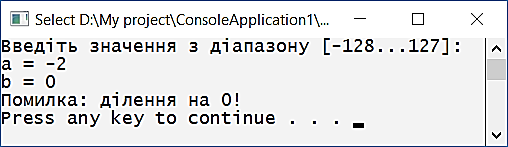


Перевірка роботи програми для значення a < b

**Значення регістрів при покроковому виконанні**

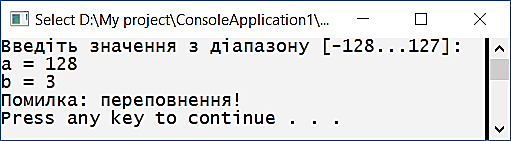
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Команда** | **Значення регістра** | | | | | | | **EFLAGS/FLAGS (CF, OF)** | |
| **al** | | **bl** | **cl** | | **<dl:al>** | |
| mov al, a | a | | н/в | н/в | | н/в | | **-** | |
| mov bl, b | н/в | | b | н/в | | н/в | | **-** | |
| cmp al, bl | Порівнюються регістри al і bl | | | | | | | | |
| jg mark1 | Якщо a>b, то переходимо на мітку mark1 | | | | | | | | |
| je mark2 | Якщо a=b, то переходимо на мітку mark2 | | | | | | | | |
| jl mark3 | Якщо a<b, то переходимо на мітку mark3 | | | | | | | | |
| **mark1:** | Мітка mark1(код буде виконуватись в цій мітці, якщо умова – істина) | | | | | | | | |
| cmp al, 0 | Порівнюємо значення <al> з нулем | | | | | | | | |
| je error1 | Якщо al = 0, то переходимо на мітку error1 | | | | | | | | |
| xchg al, bl | b | | a | н/в | | н/в | | **-** | |
| cbw | b | | н/в | н/в | | н/в | | **-** | |
| idiv bl | н/в | | н/в | н/в | | н/в | | 1 | |
| sub al, 5 | b/a-5 | | н/в | н/в | | н/в | | **-** | |
| jo error2 | Переходимо на мітку error2 | | | | | | | 1 | |
| mov x\_asm, al | н/в | | н/в | н/в | | н/в | | **-** | |
| jmp exit: | Перейти на мітку exit (вихід з ассемблерної вставки) | | | | | | | | |
| **mark2:** | Мітка mark2(код буде виконуватись в цій мітці, якщо умова – істина) | | | | | | | | |
| mov x\_asm, -195 | н/в | | н/в | н/в | | н/в | | **-** | |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **mark3:** | Мітка mark3(код буде виконуватись в цій мітці, якщо умова – істина) | | | | | | | | |
| cmp bl, 0 | Порівнюємо значення <bl> з нулем | | | | | | | | |
| je error3 | Якщо bl = 0, то переходимо на мітку error3 | | | | | | | | |
| sub al, 6 | a-6 | | н/в | н/в | | н/в | | **-** | |
| jo error4 | Переходимо на мітку error4 | | | | | | | 1 | |
| cbw | н/в | | н/в | н/в | | a-6 | | 1 | |
| idiv bl | (a-6)/b | | н/в | н/в | | н/в | | **-** | |
| mov x\_asm, al | н/в | | н/в | н/в | | н/в | | **-** | |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **error1:** | error1(помилка "ділення на 0") | | | | | | | | |
| mov er, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **error2:** | error2(помилка "переповнення") | | | | | | | | |
| mov over, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **error3:** | error3(помилка "ділення на 0") | | | | | | | | |
| mov er, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **error4:** | error4(помилка "переповнення") | | | | | | | | |
| mov over, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| exit: | exit(вихід з ассемблерної вставки) | | | | | | | | |

Перевіримо виникнення помилки при діленні на 0



Виникнення помилки при діленні на 0

Підберемо значення для перевірки виникнення переповнення



Виникнення переповнення

**Завдання 2**: Написати програму для обчислення заданого умовного цілочисельного виразу для 16бітних даних, використовуючи команди порівняння, умовного і безумовного переходів. Результат Х – теж цілочисельний і його діапазон (формат) залежить від специфіки вирішуваного умовного виразу. Провести тестові перевірки, відмітити нормальні та аномальні результати. Виконати покрокове виконання асемблерного коду та навести значення регістрів при їх виконанні.

Лістинг програми:

#include "pch.h"

#include <iostream>

#include <windows.h>

#include <stdlib.h>

int main(int argc, TCHAR\* argv[])

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

short int a, b, x\_c, x\_asm, er = 0, over = 0;

printf("Введіть значення з діапазону [-32768...32767]:\n");

printf("a = "); scanf\_s("%hi", &a);

printf("b = "); scanf\_s("%hi", &b);

if (a > b)

{

if (a == 0)

{ printf("Помилка: ");

}

else

{ x\_c = b / a - 5;

if (x\_c < -128 || x\_c > 127)

{

printf("Помилка: ");

}

else

{ printf("Результат на мові С X = %hi\n", x\_c);

}

}

}

else if (a < b)

{

if (b == 0)

{

printf("Помилка: ");

}

else if ((a - 6) < -128 || (a - 6) > 127)

{

printf("Помилка: ");

}

else

{

x\_c = (a - 6) / b;

printf("Результат на мові С X = %hi\n", x\_c);

}

}

else

{ x\_c = -195;

printf("Результат на мові С X = %hi\n", x\_c);

}

\_\_asm

{ mov ax, a // <ax> = a

mov bx, b // <bx> = b

cmp ax, bx // порівнюємо значення регістрів <ax> та <bx>

jg mark1 // a > b

je mark2 // a = b

jl mark3 // a < b

mark1 :

cmp ax, 0; // порівнюємо значення <ax> з нулем

je error1; // помилка "ділення на 0"

xchg ax, bx; // міняємо місцями значення регістрів <ax> і <bx>

cwd;

idiv bx; // <ax> = b/a

sub ax, 5; // <ax> = b/a-5

jo error2; // помилка "переповнення"

mov x\_asm, ax; // x\_asm = b/a-5

jmp exit1; // переходимо до мітки exit1

mark2

mov x\_asm, -195; // x\_asm = -195

jmp exit1; // переходимо до мітки exit1

mark3:

cmp bx, 0; // порівнюємо значення <bx> з нулем

je eror3; // помилка "ділення на 0"

sub ax, 6; // <ax> = a-6

jo error4; // помилка "переповнення"

cwd; // <ax:dx> = a-6

idiv bx; // <ax> = (a-6)/b

mov x\_asm, ax; // x\_asm = (a-6)/b

jmp exit1; // переходимо до мітки exit1

error1:

mov er, 1;

jmp exit1;

error2:

mov over, 1;

jmp exit1;

error3:

mov er, 1;

jmp exit1;

error4:

mov over, 1;

jmp exit1;

exit1:

}

if (er == 1)

{

printf("ділення на 0!\n");

}

else if (over == 1)

{ printf("переповнення!\n");

}

else if (er == 0 && over == 0)

{

printf("Результат на мові Assembler X = %hi\n", x\_asm);

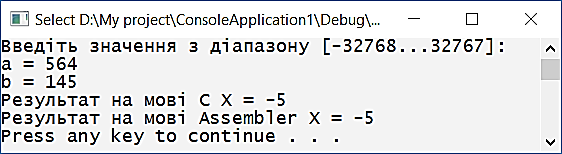
}

system("pause");

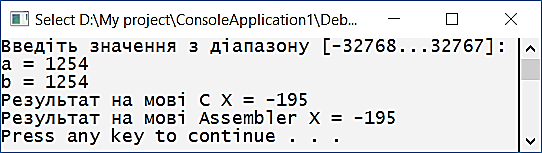
return 0;

}

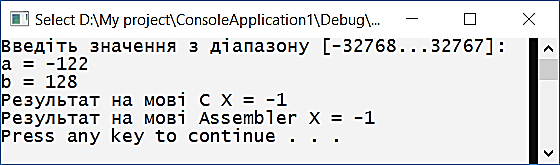
Результат виконання програми:



Перевірка роботи програми для значення a > b



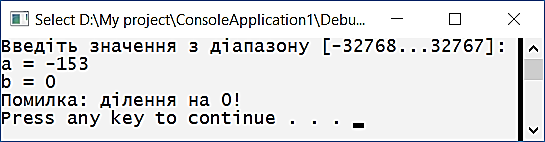
Перевірка роботи програми для значення a = b



Перевірка роботи програми для значення a < b

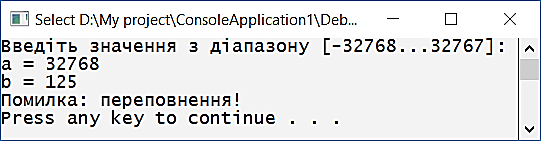
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Команда** | **Значення регістра** | | | | | | | **EFLAGS/FLAGS (CF, OF)** | |
| **ax** | | **bx** | **cx** | | **<ax:dx>** | |
| mov ax, a | a | | н/в | н/в | | н/в | | **-** | |
| mov bx, b | н/в | | b | н/в | | н/в | | **-** | |
| cmp ax, bx | Порівнюються регістри ax і bx | | | | | | | | |
| jg mark1 | Якщо a>b, то переходимо на мітку mark1 | | | | | | | | |
| je mark2 | Якщо a=b, то переходимо на мітку mark2 | | | | | | | | |
| jl mark3 | Якщо a<b, то переходимо на мітку mark3 | | | | | | | | |
| **mark1:** | Мітка mark1(код буде виконуватись в цій мітці, якщо умова – істина) | | | | | | | | |
| cmp ax, 0 | Порівнюємо значення <ax> з нулем | | | | | | | | |
| je error1 | Якщо ax = 0, то переходимо на мітку error1 | | | | | | | | |
| xchg ax, bx | b | | a | н/в | | н/в | | **-** | |
| cwd | b | | н/в | н/в | | н/в | | **-** | |
| idiv bx | н/в | | н/в | н/в | | н/в | | 1 | |
| sub ax, 5 | b/a-5 | | н/в | н/в | | н/в | | **-** | |
| jo error2 | Переходимо на мітку error2 | | | | | | | 1 | |
| mov x\_asm, ax | н/в | | н/в | н/в | | н/в | | **-** | |
| jmp exit: | Перейти на мітку exit (вихід з ассемблерної вставки) | | | | | | | | |
| **mark2:** | Мітка mark2(код буде виконуватись в цій мітці, якщо умова – істина) | | | | | | | | |
| mov x\_asm, -195 | н/в | | н/в | н/в | | н/в | | **-** | |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **mark3:** | Мітка mark3(код буде виконуватись в цій мітці, якщо умова – істина) | | | | | | | | |
| cmp bx, 0 | Порівнюємо значення <bx> з нулем | | | | | | | | |
| je error3 | Якщо bx = 0, то переходимо на мітку error3 | | | | | | | | |
| sub ax, 6 | a-6 | | н/в | н/в | | н/в | | **-** | |
| jo error4 | Переходимо на мітку error4 | | | | | | | 1 | |
| cwd | н/в | | н/в | н/в | | a-6 | | 1 | |
| idiv bx | (a-6)/b | | н/в | н/в | | н/в | | **-** | |
| mov x\_asm, ax | н/в | | н/в | н/в | | н/в | | **-** | |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| error1: | error1(помилка "ділення на 0") | | | | | | | | |
| mov er, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **error2:** | error2(помилка "переповнення") | | | | | | | | |
| mov over, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **error3:** | error3(помилка "ділення на 0") | | | | | | | | |
| mov er, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **error4:** | error4(помилка "переповнення") | | | | | | | | |
| mov over, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| exit: | exit(вихід з ассемблерної вставки) | | | | | | | | |

Перевіримо виникнення помилки при діленні на 0



Виникнення помилки при діленні на 0

Підберемо значення для перевірки виникнення переповнення



Виникнення переповнення

**Завдання 3**: Написати програму для обчислення заданого умовного цілочисельного виразу для 32бітних даних, використовуючи команди порівняння, умовного і безумовного переходів. Результат Х – теж цілочисельний і його діапазон (формат) залежить від специфіки вирішуваного умовного виразу. Провести тестові перевірки, відмітити нормальні та аномальні результати. Виконати покрокове виконання асемблерного коду та навести значення регістрів при їх виконанні.

Лістинг програми:

#include "pch.h"

#include <iostream>

#include <windows.h>

#include <stdlib.h>

int main(int argc, TCHAR\* argv[])

{ SetConsoleCP(1251);

SetConsoleOutputCP(1251);

long a, b, x\_c, x\_asm, er = 0, over = 0;

printf("Введіть значення з діапазону [-214783648...214783647]:\n");

printf("a = "); scanf\_s("%li", &a);

printf("b = "); scanf\_s("%li", &b);

if (a > b)

{ if (a == 0)

{ printf("Помилка: ");

}

else

{ x\_c = b / a - 5;

if (x\_c < -128 || x\_c > 127)

{ printf("Помилка: ");

}

else

{ printf("Результат на мові С X = %li\n", x\_c);

}

}

}

else if (a < b)

{ if (b == 0)

{ printf("Помилка: ");

}

else if ((a - 6) < -128 || (a - 6) > 127)

{ printf("Помилка: ");

}

else

{ x\_c = (a - 6) / b;

printf("Результат на мові С X = %li\n", x\_c);

}

}

else

{ x\_c = -195;

printf("Результат на мові С X = %li\n", x\_c);

}

\_\_asm

{ mov eax, a // <eax> = a

mov ebx, b // <ebx> = b

cmp eax, ebx // порівнюємо значення регістрів <eax> та <ebx>

jg mark1 // a > b

je mark2 // a = b

jl mark3 // a < b

mark1 :

cmp eax, 0; // порівнюємо значення <ax> з нулем

je error1; // помилка "ділення на 0"

xchg eax, ebx; // міняємо місцями значення регістрів <eax> і <ebx>

cdq;

idiv ebx; // <eax> = b/a

sub eax, 5; // <eax> = b/a-5

jo error2; // помилка "переповнення"

mov x\_asm, eax; // x\_asm = b/a-5

jmp exit1; // переходимо до мітки exit1

mark2:

mov x\_asm, -195; // x\_asm = -195

jmp exit1; // переходимо до мітки exit1

mark3:

cmp ebx, 0; // порівнюємо значення <ebx> з нулем

je error3; // помилка "ділення на 0"

sub eax, 6; // <eax> = a-6

jo error4; // помилка "переповнення"

cdq; // <eax:edx> = a-6

idiv ebx; // <eax> = (a-6)/b

mov x\_asm, eax; // x\_asm = (a-6)/b

jmp exit1; // переходимо до мітки exit1

error1:

mov er, 1;

jmp exit1;

error2:

mov over, 1;

jmp exit1;

error3:

mov er, 1;

jmp exit1;

error4:

mov over, 1;

jmp exit1;

exit1:

}

if (er == 1)

{ printf("ділення на 0!\n");

}

else if (over == 1)

{ printf("переповнення!\n");

}

else if (er == 0 && over == 0)

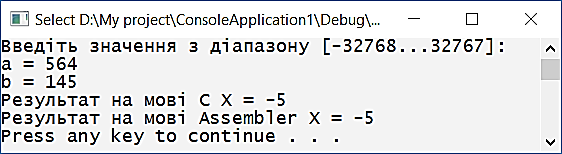
{ printf("Результат на мові Assembler X = %li\n", x\_asm);

}

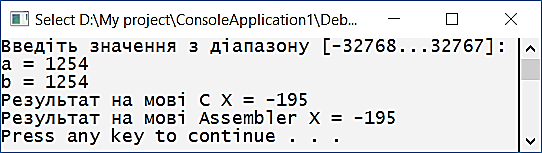
system("pause");

}

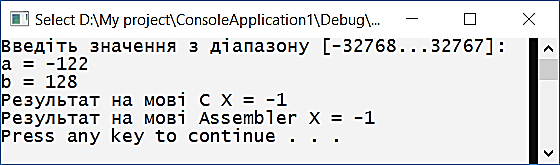
Результат виконання програми:



Перевірка роботи програми для значення a > b



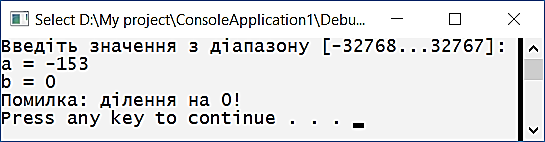
Перевірка роботи програми для значення a = b



Перевірка роботи програми для значення a < b

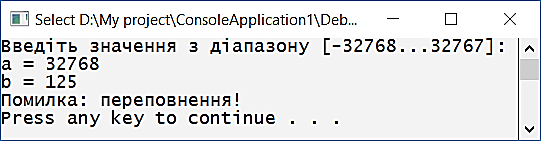
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Команда** | **Значення регістра** | | | | | | | **EFLAGS/FLAGS (CF, OF)** | |
| **eax** | | **ebx** | **ecx** | | **<eax:edx>** | |
| mov eax, a | a | | н/в | н/в | | н/в | | **-** | |
| mov ebx, b | н/в | | b | н/в | | н/в | | **-** | |
| cmp eax, ebx | Порівнюються регістри eax і ebx | | | | | | | | |
| jg mark1 | Якщо a>b, то переходимо на мітку mark1 | | | | | | | | |
| je mark2 | Якщо a=b, то переходимо на мітку mark2 | | | | | | | | |
| jl mark3 | Якщо a<b, то переходимо на мітку mark3 | | | | | | | | |
| **mark1:** | Мітка mark1(код буде виконуватись в цій мітці, якщо умова – істина) | | | | | | | | |
| cmp eax, 0 | Порівнюємо значення <eax> з нулем | | | | | | | | |
| je error1 | Якщо ax = 0, то переходимо на мітку error1 | | | | | | | | |
| xchg eax, ebx | b | | a | н/в | | н/в | | **-** | |
| cdq | b | | н/в | н/в | | н/в | | **-** | |
| idiv ebx | н/в | | н/в | н/в | | н/в | | 1 | |
| sub eax, 5 | b/a-5 | | н/в | н/в | | н/в | | **-** | |
| jo error2 | Переходимо на мітку error2 | | | | | | | 1 | |
| mov x\_asm, eax | н/в | | н/в | н/в | | н/в | | **-** | |
| jmp exit: | Перейти на мітку exit (вихід з ассемблерної вставки) | | | | | | | | |
| **mark2:** | Мітка mark2(код буде виконуватись в цій мітці, якщо умова – істина) | | | | | | | | |
| mov x\_asm, -195 | н/в | | н/в | н/в | | н/в | | **-** | |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **mark3:** | Мітка mark3(код буде виконуватись в цій мітці, якщо умова – істина) | | | | | | | | |
| cmp ebx, 0 | Порівнюємо значення <ebx> з нулем | | | | | | | | |
| je error3 | Якщо ebx = 0, то переходимо на мітку error3 | | | | | | | | |
| sub eax, 6 | a-6 | | н/в | н/в | | н/в | | **-** | |
| jo error4 | Переходимо на мітку error4 | | | | | | | 1 | |
| cdq | н/в | | н/в | н/в | | a-6 | | 1 | |
| idiv ebx | (a-6)/b | | н/в | н/в | | н/в | | **-** | |
| mov x\_asm, eax | н/в | | н/в | н/в | | н/в | | **-** | |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| error1: | error1(помилка "ділення на 0") | | | | | | | | |
| mov er, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **error2:** | error2(помилка "переповнення") | | | | | | | | |
| mov over, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **error3:** | error3(помилка "ділення на 0") | | | | | | | | |
| mov er, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| **error4:** | error4(помилка "переповнення") | | | | | | | | |
| mov over, 1 | н/в | н/в | | | н/в | | н/в | | **-** |
| jmp exit: | Перейти на мітку exit(вихід з ассемблерної вставки) | | | | | | | | |
| exit: | exit(вихід з ассемблерної вставки) | | | | | | | | |

Перевіримо виникнення помилки при діленні на 0



Виникнення помилки при діленні на 0

Підберемо значення для перевірки виникнення переповнення



Виникнення переповнення

***Висновок:*** в ході виконання лабораторної роботи ознайомлено з основними командами мови Assembler для організації умовних переходів; набуто практичних навичок в написанні програм з організацією умовних переходів на мові Assembler.