Міністерство освіти і науки України   
Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

Лабораторна робота №7 з дисципліни «Операційні системи»  
**Система команд. Організація умовних переходів**

**Варіант: 15**

Виконав студент групи ТР-12

Каркушевський Владислав

Перевірила д.т.н., проф. Левченко Л. О.

КИЇВ 2022

**Мета роботи** : Вивчення команд умовних переходів і способів їх використання в асемблерних програмах для реалізації розгалужень в обчисленнях.

**Теоритична частина**:

Команди переходу забезпечують безумовні переходи усередині поточного кодового сегменту (внутрішній перехід) або з поточного сегменту в інший кодовий сегмент (зовнішній або міжсегментний перехід). Всі ці переходи реалізуються за допомогою команди JMP, яка має формат

**JMP [type] OPR**

де type - тип переходу:

* SHORT (короткий)
* NEAR (ближній)
* FAR (дальній)

За замовчуванням приймається NEAR;

OPR - ім'я, мітка або адресний вираз. Залежно від типу переходу (type) і/або структури операнда OPR можна виділити 5 видів команд безумовного переходу, з яких три визначають внутрішньосегментні переходи, а дві - міжсегментні переходи.

**Безумовний перехід JMP (JuMP)**

Використовується в програмі для організації безумовного переходу як всередині поточного сегмента, так і поза його межі. При певних умовах у захищеному режимі команда jmp може використовуватися для переключення задач.

**Синтаксис: jmp мітка**

**Алгоритм роботи команди.** Команда jmp в залежності від типу свого операнда змінює вміст або тільки одного регістра EIP або обох CS та EIP:

• якщо операнд є мітка у поточному сегменті команд, асемблер формує машинну команду, операнд якої є значення зі знаком, що є зсувом переходу відносно наступної за jmp команди. При такому переході змінюється тільки регістр eip/ip;

• якщо операнд у команді jmp – символічний ідентифікатор комірки пам’яті, асемблер вважає, що в ній знаходиться адреса, за якою треба передати управління;

Виконання команди не впливає на прапорці. Цю команду використовують для виконання ближніх та дальніх безумовних переходів.

**Умовний перехід JCC**

Використовується для переходу всередині сегмента команд в залежності від деякої умови

**Синтаксис:** jcc мітка

**Алгоритм роботи команди .** Перевірка стану прапорців в залежності від коду операції:

• якщо умова виконується, перейти до комірки, позначеної операндом;

• якщо умова не виконується, передати управління наступній команді. Алгоритм роботи команди jcxz/jecxz. Перевірка умови чи дорівнює нулю вміст регістру ecx/cx.

• якщо умова виконується, то вміст регістру ecx/cx дорівнює нулю, перейти до комірки, позначеної у операнді;

• якщо умова не виконується, то вміст регістру ecx/cx не дорівнює нулю, передати управління наступній команді, що стоїть після jcxz/jecxz

Команди умовного переходу зручно застосовувати для перевірки різних умов. Букви СС в команді JCC у скороченому вигляді описують умови переходу і мають значення:

**Е** – Equal (дорівнює),

**N** – Not (ні, не дорівнює),

**G** – Great (більше, використовується для чисел зі знаком),

**L** – Less (менше, використовується для чисел зі знаком),

**А** – Above (вище, більше, використовується для чисел без знаку),

**B** – Below (нижче, менше, використовується для чисел без знаку),

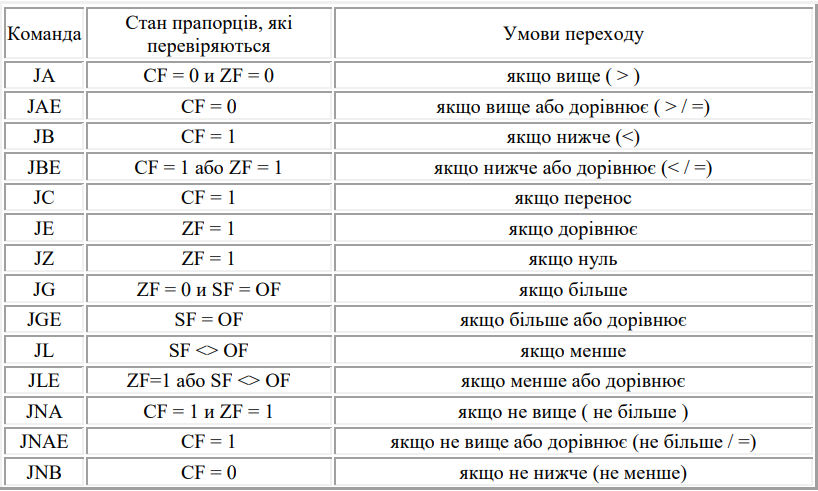
**C** – Carry Flag (прапорець перенесення, встановлюється в 1, якщо результат попередньої операції не вміщається в приймачі і сталося перенесення зі старшого біта або якщо потрібно зайняти (при відніманні). Інакше встановлений в 0),

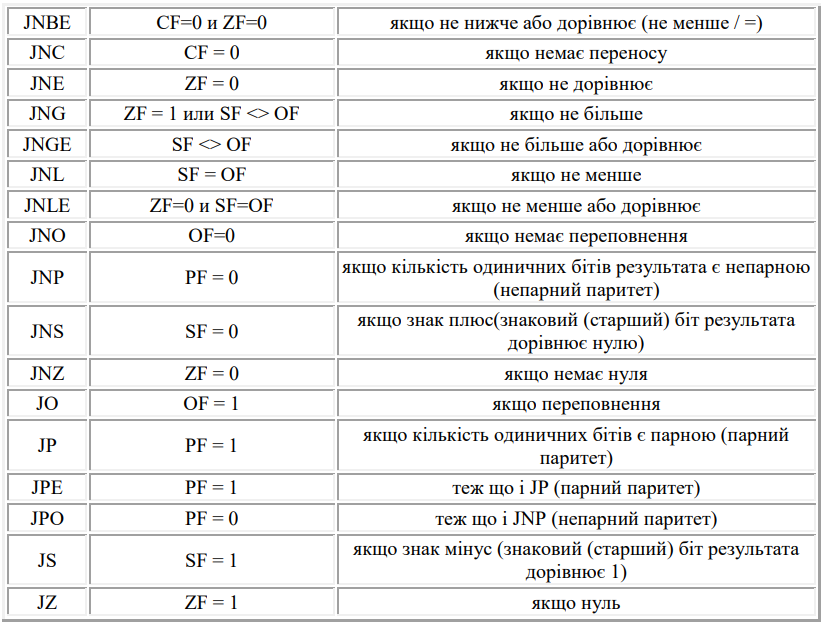
**O** – Overflow Flag (прапорець переповнення, встановлюється в 1, якщо результат попередньої арифметичної операції над числами зі знаком виходить за допустимі для них межі. Наприклад, якщо при складанні двох позитивних чисел виходить число зі старшим бітом, що дорівнює одиниці, тобто негативне і навпаки),

**S** - Sign Flag (прапорець знака, цей прапорець завжди дорівнює старшому біту результату),

**P** - Parity Flag (прапорець парності, встановлюється в 1, якщо молодший байт результату попередньої команди містить парну кількість бітів, які дорівнюються 1. Якщо кількість одиниць в молодшому байті є непарною, то цей прапорець дорівнює 0).

**Перелік команд умовного переходу, відповідні прапорці та умови переходу:**



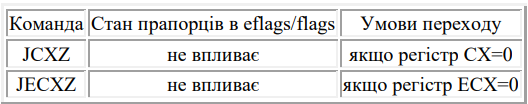


Логічні умови «більше» та «менше» відносяться до порівнянь цілочисельних значень зі знаком, умови «вище» та «нижче» - до порівнянь цілочисельних значень без знака.

Є декілька мнемонічних позначень однієї і тієї ж команди. Це пояснюється тим, що для мікропроцесора і8086 команди умовного переходу могли здійснювати тільки короткі переходи в межах -128 до +127, починаючи від наступної команди.

Починаючи з мікропроцесора і386, ці команди могли виконувати будь-які переходи в межах поточного сегмента команд. Це стало можливим шляхом введення в систему команд мікропроцесора додаткових команд.

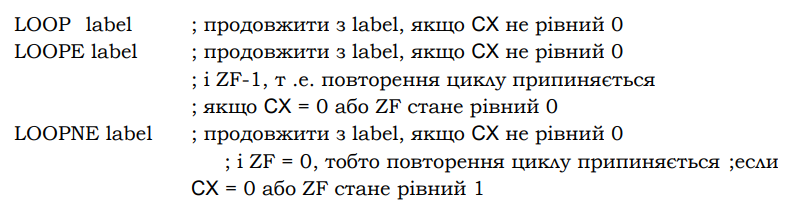
Для переходу між сегментами треба комбінувати команди умовного переходу та команду безумовного переходу jmp. Застосування jcxz/jecxz:



Команду jcxz/jecxz зручно використовувати для організації цикла та ланцюжкових команд, які використовують регістр ecx/cx.

Ця команда виконує тільки близькі переходи в межах -128 до +127 байт, починаючи від наступної 4 команди. Її використовують для попередньої перевірки лічильника циклу в регістрі cx для того, щоб обійти цикл, якщо його лічильник є нульовим.

Використовуючи команди умовних і безумовних переходів, можна реалізувати різні види розгалуження в програмі, у тому числі і цикли. Але для реалізації циклів з певним числом повторень в системі команд х86 є спеціальні команди управління циклами, кожна з яких при виконанні зменшує на 1 вміст CX, а потім використовує його нове значення для ухвалення рішення про перехід:



Для команди LOOPE є еквівалент LOOPZ, а для команди LOOPNE - LOOPNZ.

Очевидно, що перед входом в цикл потрібно підготувати лічильник циклів в СХ, для запобігання входу в цикл з нульовим значенням СХ корисно використовувати команду короткого переходу.

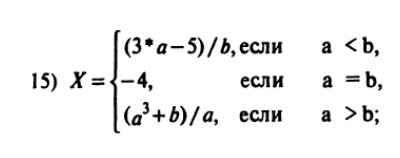
JCXZ label ; перехід, якщо СХ = 0

При використанні умовних команд управління циклом в тілі циклу повинні бути команди, що впливають на прапорець ZF.

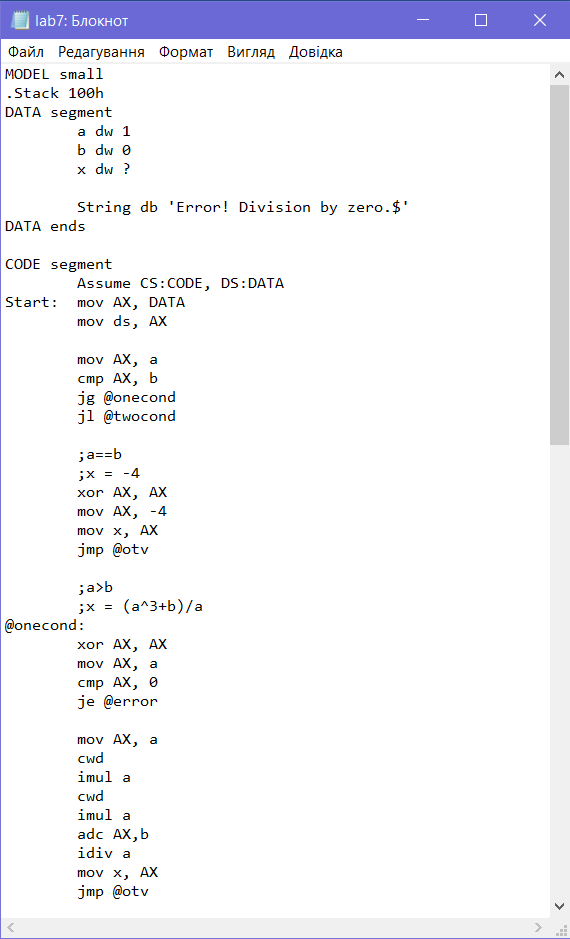
**Завдання**

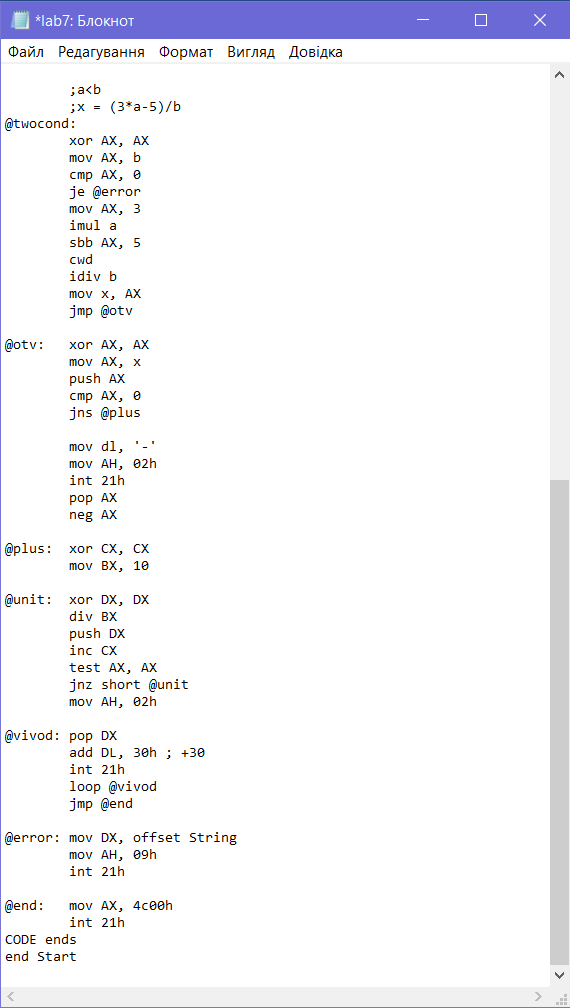
Обчислити умовний цілочисельний вираз у форматах Integer та Word, використовуючи команди порівняння. Результат перевірити на область допустимих значень.

**Варіант 15**



**Код програми:**





**Розглянемо детальніше, та пояснимо основні дії:**

**MODEL small** ;модель пам’яті для ехе-файлів

**.Stack 100h** ;виділяємо PSP 256 байт

**DATA segment** ;початок сегменту даних

**a dw 1**;вихідні числа які використовуються в формулі

**b dw 0**

**x dw ?** ;та змінна для збереження

**String db 'Error! Division by zero.$'** повідомлення, яке виводиться коли ;знаменник = 0

**DATA ends** ;кінець сегменту даних

**CODE segment** ;початок сегменту коду

**Assume CS:CODE, DS:DATA**;директива assume повідомляє, що регістр cs ; вказує на початок сегменту коду, ds – сегменту даних

**Start:** ;точка входу

**mov AX, DATA** ;ініціалізація регістру ds

**mov DS, AX**;адресою сегмента даних

**mov AX, a**; a🡪AX

**cmp AX, b;** порівняння двох чисел а і b

**jg @onecond**; a>b

**jl @twocond** ;a<b

**;a==b**

**;x = -4**

**xor AX, AX ;** обнуляємо регістр ах

**mov AX, -4** ;-4🡪AX

**mov x, AX** ; AX 🡪x

**jmp @otv** ;безумовний перехід до @otv

**;a>b**

**;****x = (a^3+b)/a**

**@onecond:**

**xor AX, AX** ; обнуляємо регістр ах

**mov AX, a** ; a🡪AX

**cmp AX, 0;** перевіряємо чи знаменник дорівнює нулю

**je** **@error** ;якщо дорівнює нулю викликаємо @error

**mov AX, a**; a🡪AX

**cwd**;команда перетворення слова у подвійне слово

**imul a;** a\*a🡪AX

**cwd**;команда перетворення слова у подвійне слово

**imul a;** a^2\*a🡪AX

**adc AX,b;**a^3+b🡪AX

**idiv a;(**a^3+b)/a🡪AX

**mov x, AX**; AX 🡪x

**jmp @otv**;безумовний перехід до @otv

**;a<b**

**;x = (3\*a-5)/b**

**@twocond:**

**xor AX, AX**; обнуляємо регістр AX

**mov AX, b**; b🡪AX

**cmp AX, 0;** перевіряємо чи знаменник дорівнює нулю

**je @error** ;якщо дорівнює нулю викликаємо @error

**mov AX, 3** ;3🡪AX

**imul a** ;3\*a🡪AX

**sbb AX, 5** ;3\*a-5🡪AX

**cwd** ;команда перетворення слова у подвійне слово

**idiv b**;( 3\*a-5)/b🡪AX

**mov x, AX**; AX 🡪x

**jmp @otv**;безумовний перехід до @otv

**@otv: xor AX, AX**; обнуляємо регістр AX

**mov AX, x** ;x🡪AX

**push AX**;зберігаємо в стек

**cmp AX, 0**;порівнюємо ах з 0

**jns @plus** ; якщо знак плюс(знаковий (старший) біт результата ; дорівнює нулю)

**mov dl, '-'**

**mov AH, 02h**

**int 21h** ; виклик DOS - функція переривання

**pop AX** ;вивантажуємо значення зі стеку

**neg AX** ;перетворюємо від’ємне число у позитивне

**@plus: xor CX, CX** ;лічильник остач

**mov BX,10** ;завантажуємо основу 10-ї системи числення

**@unit: xor DX, DX**;остача при ділені

**div BX;**виконуємо ділення на 10

**push DX**;результат у зворотньому порядку утворює число у

;новій системі счислення

**inc CX**;збільшуємо лічильник

**test AX, AX**;перевіряємо, чи ціла частина = 0

**jnz short @unit** ;ділимо, поки ціла частина не дорівнює нулю

**mov AH, 02h** ;виводимо остачу

**@vivod: pop DX** ;дістаємо зі стеку збережені числа

**add DL, 30h** ;переводимо їх в ASCII-коди

**int 21h** ; виклик DOS - функція переривання

**loop @vivod** ;повторюємо, поки СХ не дорівнює 0

**jmp @end** ;вихід

**@error: mov DX, offset String** ;виведення повідомленняпро помилку

**mov AH, 09h** ; Функція DOS 9h означає виведення на екран

**int 21h** ; виклик DOS - функція переривання

**@end: mov AX, 4c00h :** ;коректно завершуємо нашу програму, виходимо

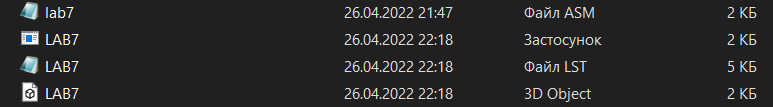
**int 21h** ; виклик DOS - функція переривання

**CODE ends** ;кінець сегменту коду

**end Start** ;кінець програми

Запустимо **транслятор** та створими лістинг(\*.lst) та об’єктний файл(\*.obj) за допомогою команди: **tasm /la /zi lab7.asm**

Запустимо **компоновщик** та створимо файл (\*.exe) за допомогою команди: **tlink /x /v lab7.obj**

****

Запустимо нашу програму ввівши команду **lab7.exe**

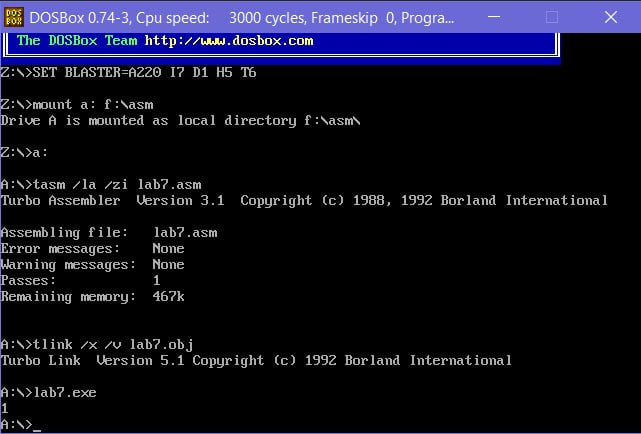
1. a>b

x = (a^3+b)/a

x=(1^3+0)/1

a=1

b=0

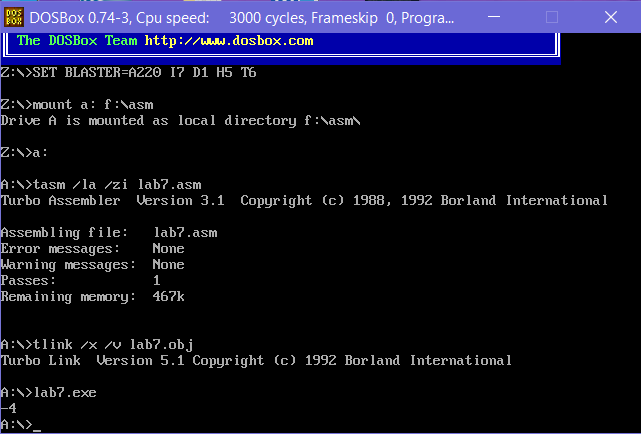


2)a=b

x = -4

a=2

b=2



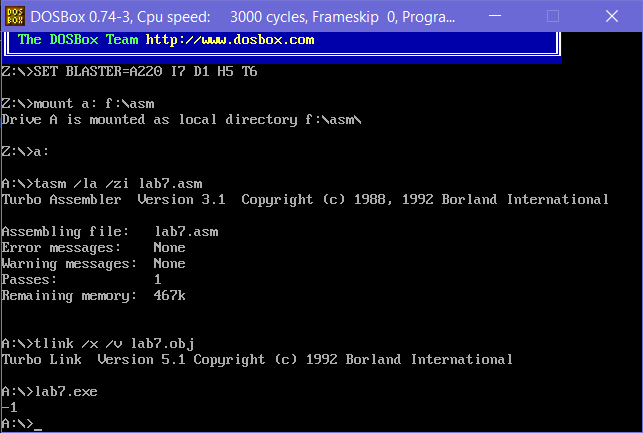
3)a<b

x = (3\*a-5)/b

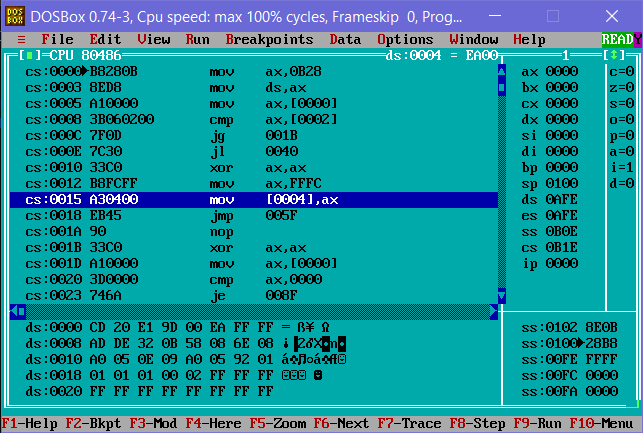
x=(1^3+0)/1

a=1

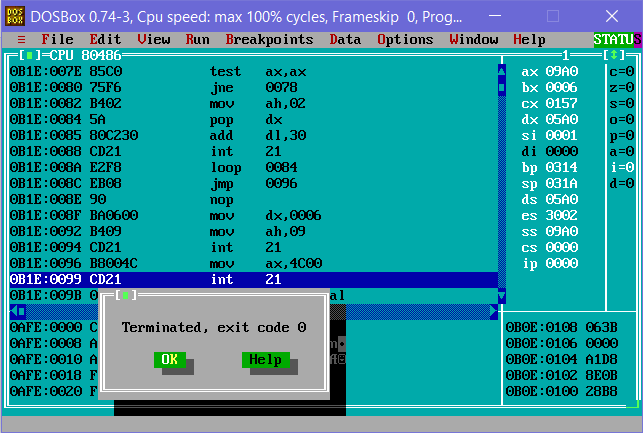
b=2

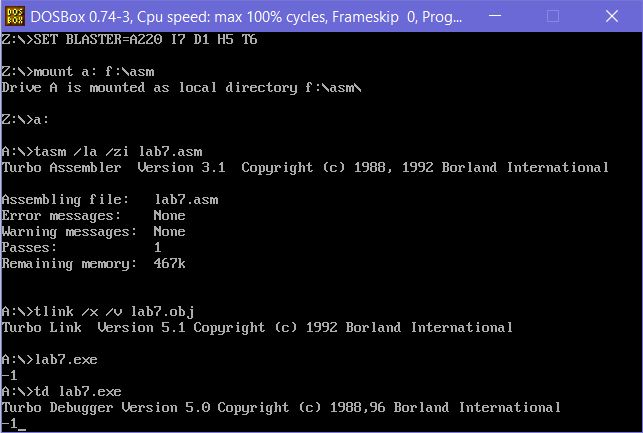


Запустимо **налагоджувач**



За допомогою клавіші **F8**(або **F7**) покроково виконуємо нашу програму і виводимо отримане словосполучення в консольному режимі





**Висновок:**

На цій лабораторній роботі було освоєно команди умовних переходів і способів їх використання в асемблерних програмах для реалізації розгалужень в обчисленнях. Було використано арифметичні команди. Було запрограмовано блок за допомогою команд умовних переходів для обчислення арифметичних виразів на прикладі реалізації математичних формул з використанням арифметичних команд асемблера за індивідуальними завданнями. Виконано тестові перевірки, проаналізовано результати. Виведено результат на екран.