Лабораторна робота 5

Тема: **БЕЗУМОВНІ ТА УМОВНІ ПЕРЕХОДИ**

Мета: Розкрити механізм передачі управління в програмі (цикли і переходи) для логічних порівнянь і програмної організації.

1 Теоретичні відомості

До цієї лабораторній роботи приклади майже виконувалися послідовно команда за командою. Однак, програмовані завдання рідко бувають такі прості. Більшість програм містять ряд циклів, в яких декілька команд повторюються до досягнення певного вимоги, і різні перевірки, визначають, які з кількох дій слід виконувати. Звичайним вимогою є перевірка - чи повинна програма завершити виконання. Ці вимоги включають передачу управління за адресою команди, яка не перебуває безпосередньо за виконуваної в сучасний момент командою. Передача управління може здійснюватися вперед для виконання нової групи команд або назад для повторення вже виконаних команд. Деякі команди можуть передавати управління, змінюючи нормальну послідовність кроків безпосередньої модифікацією значення зміщення в командному покажчику. Нижче наведені чотири способи передачі управління (всі будуть розглянуті в цій главі):

- **JMP** - безумовний перехід;
- **LOOP** – цикл;
- **Jnnn** (більше, менше, дорівнює) - умовний перехід;
- **CALL** - виклик процедури.

Зауважимо, що є три типи адрес: *SHORT*, *NEAR* і *FAR*. Адресація *SHORT* використовується при циклах, умовних переходів і деяких безумовних переходах. Адресація *NEAR* і *FAR* використовується для викликів процедур (*CALL*) і безумовних переходів, які не кваліфікуються, як *SHORT*. Всі три типи передачі управління впливають на вміст регістра *IP*; тип *FAP* також змінює регістр *CS*.

1.1 Безумовний перехід JMP

Однієї з команд, що зазвичай використовуються для передачі управління є команда *JMP*. Ця команда виконує безумовний перехід, тобто забезпечує передачу управління за будь-яких обставин.

Основною інструкцією переходу в наборі інструкцій процесора 8086 є інструкція *JMP*. Ця інструкція вказує процесору 8086, що в якості наступної за *JMP* інструкцією потрібно виконати інструкцію по цільовій мітці. Наприклад, після завершення виконання фрагмента програми:

```
.
.
.
mov   ax,1
jmp   SHORT AddTwoToAX
AddOneToAx:
inc   ax
jmp   SHORT AXIsSet
AddTwoToAX:
inc   ax
AXIsSet:
.
.
```

регістр *AX* буде містити значення 3, а інструкції *ADD* і *JMP*, наступні за міткою *AddOneToAX*, ніколи виконані не будуть. тут

інструкція:

```
jmp  AddTwoToAX
```

вказує процесору 8086, що потрібно встановити покажчик інструкцій IP в значення зміщення мітки *AddTwoToAX*, тому наступною виконуваної інструкцією буде інструкція:

```
add   ax,2
```

Іноді спільно з інструкцією JMP використовується операція SHORT. Для вказівки на цільову мітку інструкція *JMP* зазвичай використовується 16 бітовий зсув. Операція *SHORT* вказує Турбо Асемблеру, що потрібно використовувати не 16-бітове, а 8-бітове зміщення (що дозволяє заощадити в інструкції JMP один байт). Наприклад, останній фрагмент програми можна переписати так, що він стане на два байти коротше:

```
            .
            .
            .
            mov   ax,1
            jmp   SHORT AddTwoToAX
AddOneToAx:
            inc   ax
            jmp   SHORT AXIsSet
AddTwoToAX:
            inc   ax
AXIsSet:
            .
            .
            .
```

Недолік використання операції *SHORT* (короткий) полягає в тому, що короткі переходи можуть здійснювати передачу управління на мітки, віддалені від інструкції *JMP* не далі, ніж на 128 байтів, тому в деяких випадках Турбо Асемблер може повідомляти вам, що мітка недосяжна за допомогою короткого переходу . До того ж операцію *SHORT* має сенс використовувати для посилань вперед, оскільки для переходів назад (на попередні мітки) Турбо Асемблер автоматично використовує короткі переходи, якщо на мітку можна перейти за допомогою короткого переходу, і довгі в іншому випадку.

Інструкцію *JMP* можна використовувати для переходу в інший сегмент коду, завантажуючи в одній інструкції і регістр *CS*, і регістр *IP*. Наприклад, у програмі:

```
            .
            .
            .
CSeg1       SEGMENT
            ASSUME   CS:Cseg1
            .
            .
            .
FarTarget           LABEL   FAR
            .
            .
            .
CSeg1       ENDS
            .
            .
            .
CSeg2       SEGMENT
            ASSUME   CS:CSeg2
            .
            .
            .
            jmp   FarTarget  ; переход дальнего типа
            .
            .
            .
CSeg2       ENDS
```

```
            .
            .
            .
```

Виконується перехід далекого типу.

Якщо ви хочете, щоб мітка примусово інтерпретувалася, як мітка далекого типу, можна використовувати операцію FAP PTR. Наприклад, у фрагменті програми:

```
            .
            .
            .
            jmp   FAR PTR NearLabel
            nop
 NearLabel:
            .
            .
            .
```

виконується перехід далекого типу на мітку *NearLabel*, хоча ця мітка знаходиться в тому ж сегменті коду, що і інструкція *JMP*.

Нарешті, ви можете виконати перехід за адресою, записаному в регістрі або в змінної пам'яті. наприклад:

```
            .
            .
            .
            mov   ax,OFFSET TestLabel
            jmp   ax
            .
            .
            .
 TestLabel:
            .
            .
            .
```

Тут виконується перехід на мітку TestLabel, так само, як і в наступному фрагменті:

```
            .
            .
            .
            .DATA
 JumpTarget        DW   TestLabel
            .
            .
            .
            .CODE
            .
            .
            .
            jmp   [JumpTarget]
            .
            .
            .
 TestLabel:
            .
            .
            .
```

1.2 Команда LOOP

Команда *JMP* в прикладі реалізує нескінченний цикл. Але більш імовірно підпрограма повинна виконувати певне число циклів. Команда *LOOP*, яка служить для цієї мети, використовує початкове значення в регістрі *CX*. У кожному циклі команда *LOOP* автоматично зменшує вміст регістра CX на 1. Поки значення в *CX* не дорівнює нулю, керування передається за адресою, вказаною в операнді, і якщо в *CX* буде 0, управління переходить на наступну після *LOOP* команду. Наприклад:

```
print:
  mov ah, 02h
   xor dx, dx
   pop dx
   add dx, '0'
   int 21h
   loop print
```

Тут виконується вивід результату цифр поетапно з корекцією коду ASCII, доки не виведеться все число.

1.3 Умовні переходи

У попередніх прикладах було показано, що команда *LOOP* зменшує на одиницю вміст регістра *CX* і перевіряє його: якщо не нуль, то управління передається за адресою, вказаною в операнді. Таким чином, передача управління залежить від конкретного стану. Асемблер підтримує велику кількість команд умовного переходу, які здійснюють передачу управління в залежності від станів прапорцевого регістра. Наприклад, при порівнянні вмісту двох полів наступний перехід залежить від значення прапора.

Команду *LOOP* в програмно можна замінити на дві команди: одна зменшує вміст регістра CX, а інша виконує умовний перехід:

Використання LOOP:	Використання умовного переходу:
LOOP A20	*DEC CX*
	JNZ A20

Команди *DEC* і *JNZ* діють аналогічно команді *LOOP*: зменшують вміст регістра *CX* на 1 і виконує перехід на мітку *A20*, якщо в *CX* не нуль. Команда *DEC* крім того встановлює прапор нуля в прапорцевому регістрі в стан 0 або 1. Команда *JNZ* потім перевіряє цю установку. У розглянутому прикладі команда *LOOP* хоча і має обмежене використання, але більш ефективна, ніж дві команди: *DEC* і *JNZ*.

Аналогічно командам *JMP* і *LOOP* операнд в команді *JNZ* містять задану значення відстані між кінцем команди *JNZ* і адресою *A20*, яке додається до командного покажчику. Ця відстань повинна бути в межах від -128 до +127 байт. У разі переходу за ці межі асемблер видасть повідомлення: "Relative jump out of range" (перевищені відносні межі переходу).

Розглядаючи призначення команд умовного переходу слід пояснити характер їх використання. Типи даних, над якими виконуються арифметичні операції і операції порівняння визначають якими командами користуватися: беззнаковими або знаковими. Беззнакові дані використовують усі біти як біти даних; характерним прикладом є символьні рядки: імена, адреси і натуральні числа. У знакових даних самий лівий біт являє собою знак, причому якщо його значення дорівнює нулю, то число позитивне, і якщо одиниці, то негативне. Багато числові значення можуть бути як позитивними так і негативними.

В якості прикладу припустимо, що регістр *AX* містить 11000110, а *BX* - 00010110. Команда

```
CMP AX,BX
```

порівнює вміст регістрів *AX* і *BX*. Якщо дані беззнакові, то значення в *AX* більше, а якщо знакові - то менше.

Будь-яку перевірку можна кодувати одним з двох мнемонічних кодів.

Наприклад, *JB* і *JNAE* генерує один і той же об'єктний код, хоча позитивну перевірку JB легше зрозуміти, чим негативну *JNAE*.

Таблиця 1.1

Переходи для беззнакових даних

Мнемоника	**Опис**	**Прапори, що перевіряються**
JE/JZ	Перехід, якщо дорівнює/нуль	ZF
JNE/JNZ	Перехід, якщо не дорівнює/нуль	ZF
JA/JNBE	Перехід, якщо вище / не нижче або дорівнює	ZF,CF
JAE/JNB	Перехід, якщо вище або дорівнює / не нижче	CF
JB/JNAE	Перехід, якщо нижче / не вище або дорівнює	CF
JBE/JNA	Перехід, якщо нижче або дорівнює / не вище	CF,AF

CF – прапорець переносу, SF – прапорець знаку, OF – прапорець переповнення, ZF – прапорець нуля, PF – прапорець парності.

Таблиця 1.2

Переходи для знакових даних

Мнемоника	**Опис**	**Прапори, що перевіряються**
JE/JZ	Перехід, якщо одно / нуль	ZF
JNE/JNZ	Перехід, якщо не дорівнює/нуль	ZF
JG/JNLE	Перехід, якщо більше / не менше або дорівнює	ZF,SF,OF
JGE/JNL	Перехід, якщо більше або дорівнює / не менше	SF,OF
JL/JNGE	Перехід, якщо менше / не більше або дорівнює	SF,OF

JLE/JNG	Перехід, якщо менше або дорівнює / не більше	ZF,SF,OF

Команди переходу для умови дорівнює або нуль (JE / JZ) і не дорівнює або не нуль (JNE / JNZ) присутні в обох списках для беззнакових і знакових даних. Стан дорівнює / нуль відбувається поза залежністю від наявності знака.

Таблиця 1.3

Спеціальні арифметичні перевірки

Мнемоника	**Опис**	**Прапори, що перевіряються**
JS	Перехід, якщо є знак (негативно)	SF
JNS	Перехід, якщо немає знаку (позитивно)	SF
JC	Перехід, якщо є перенос (аналогічно JB)	CF
JNC	Перехід, якщо немає переносу	CF
JO	Перехід, якщо є переповнення	OF
JNO	Перехід, якщо немає переповнення	OF
JP/JPE	Перехід, якщо паритет парний	PF
JNP/JP	Перехід, якщо паритет непарний	PF

Ще одна команда умовного переходу перевіряє дорівнює чи вміст регістра *CX* нулю. Ця команда необов'язково повинна розташовуватись безпосередньо за командою арифметики чи порівняння. Одним з місць для команди *JCXZ* може бути початок циклу, де вона перевіряє чи містить регістр *CX* ненульове значення.

Потрібно запам'ятати тільки, що для беззнакових даних є переходи по станам дорівнює, вище або нижче, а для беззнакових - дорівнює, більше або менше. Переходи по перевірках прапорів переносу, переповнення і паритету мають особливе призначення. Асемблер транслює мнемонічні коди в об'єктний код незалежно від того, яку з двох команд ви застосували. Однак, команди *JAE* і *JGE* будучи явно однаковими, перевіряють різні прапори.

1.4 Оператор CALL

Команда *CALL* передає управління ближньої або дальньої процедурі з запам'ятовуванням в стеці адреси точки повернення. Команда *RET* повертає керування з процедури викликає програмі, адреса повернення отримує з стека. Наприклад:

```
..code..
call 0455659
..more code..
Код с адреса 455659:
add ax, 500
mul ax, dx
ret
```

Коли виконується команда *CALL*, процесор передає управління на код з адреси 455659, і виконує його до команди *RET*, а потім повертає керування команді наступної за *CALL*. Код який викликається командою *CALL* називається процедурою. Ви можете помістити код, який ви часто використовуєте в процедуру і кожен раз коли він вам потрібен викликати його командою *CALL*.

Детальніше: команда *CALL* поміщає регістр *IP* (вказівник на наступну команду, яка повинна бути виконана) у стек, а команда *RET* витягує його і передає управління цією адресою. Ви також можете визначити аргументи, для програми (процедури), що викликається. Це можна зробити через стек:

```
push значение_1
push значение_2
call procedure
```

2 Порядок виконання роботи

2.1 Напишіть текст програми згідно свого варіанту представлений в Таб.2.1. V=N (де N - номер списку в журналі, V – номер варіанту. *A, B, C* – вважати в десятковій формі.

2.2 Повторіть повний цикл створення програми для Турбо Асемблера див. Лабораторну роботу 1 рис.1.1.

2.3 Запустіть програму на виконання.

2.4 Проаналізуйте та збережіть результат програми.

Таблиця 2.1

1)	$X=\left\{\begin{array}{ll}\frac{a-b}{a}+1, & \text{якщо } a<b,\\ 25, & \text{якщо } a=b,\\ \frac{a-5}{b}, & \text{якщо } a>b;\end{array}\right\}$	2)	$X=\left\{\begin{array}{ll}\frac{a-b}{a}-3, & \text{якщо } a>b,\\ 2, & \text{якщо } a=b,\\ \frac{a^3+1}{b}, & \text{якщо } a<b;\end{array}\right\}$
3)	$X=\left\{\begin{array}{ll}\frac{b}{a+5}, & \text{якщо } a<b,\\ -5, & \text{якщо } a=b,\\ \frac{a*a-b}{b}, & \text{якщо } a>b;\end{array}\right\}$	4)	$X=\left\{\begin{array}{ll}\frac{a}{b}+10, & \text{якщо } a<b,\\ -51, & \text{якщо } a=b,\\ \frac{a*b-4}{a}, & \text{якщо } a>b;\end{array}\right\}$
5)	$X=\left\{\begin{array}{ll}\frac{a}{b}-1, & \text{якщо } a>b,\\ -25, & \text{якщо } a=b,\\ \frac{b^3-5}{a}, & \text{якщо } a<b;\end{array}\right\}$	6)	$X=\left\{\begin{array}{ll}\frac{a}{b}-1, & \text{якщо } a>b,\\ -25, & \text{якщо } a=b,\\ \frac{b^3-5}{a}, & \text{якщо } a<b;\end{array}\right\}$
7)	$X=\left\{\begin{array}{ll}52*\frac{b}{a}+b, & \text{якщо } a>b,\\ -125, & \text{якщо } a=b,\\ \frac{a-5}{b}, & \text{якщо } a<b;\end{array}\right\}$	8)	$X=\left\{\begin{array}{ll}\frac{a*b-1}{a}, & \text{якщо } a>b,\\ 255, & \text{якщо } a=b,\\ \frac{a-5}{b}, & \text{якщо } a<b;\end{array}\right\}$
9)	$X=\left\{\begin{array}{ll}1-\frac{b}{a}, & \text{якщо } a>b,\\ -10, & \text{якщо } a=b,\\ \frac{a-5}{b}, & \text{якщо } a<b;\end{array}\right\}$	10)	$X=\left\{\begin{array}{ll}\frac{a}{b}+31, & \text{якщо } a>b,\\ -25, & \text{якщо } a=b,\\ \frac{5*b-1}{a}, & \text{якщо } a<b;\end{array}\right\}$
11)	$X=\left\{\begin{array}{ll}\frac{2+b}{a}, & \text{якщо } a>b,\\ -2, & \text{якщо } a=b,\\ \frac{a-5}{b}, & \text{якщо } a<b;\end{array}\right\}$	12)	$X=\left\{\begin{array}{ll}\frac{b}{a}+1, & \text{якщо } a<b,\\ 25, & \text{якщо } a=b,\\ \frac{a^3-5}{b}, & \text{якщо } a>b;\end{array}\right\}$

13)	$X = \begin{cases} \frac{b}{a} + 61, & \text{якщо } a > b, \\ -5, & \text{якщо } a = b, \\ \frac{b-a}{b}, & \text{якщо } a < b; \end{cases}$	14)	$X = \begin{cases} \frac{a}{b} + 1, & \text{якщо } a > b, \\ -2, & \text{якщо } a = b, \\ \frac{a-b}{a}, & \text{якщо } a < b; \end{cases}$
15)	$X = \begin{cases} \frac{3*a-5}{b}, & \text{якщо } a < b, \\ -4, & \text{якщо } a = b, \\ \frac{a^3+b}{a}, & \text{якщо } a > b; \end{cases}$	16)	$X = \begin{cases} \frac{b}{a} - 1, & \text{якщо } a < b, \\ -295, & \text{якщо } a = b, \\ \frac{a-235}{b}, & \text{якщо } a > b; \end{cases}$
17)	$X = \begin{cases} 2*\frac{a}{b} + 1, & \text{якщо } a > b, \\ -445, & \text{якщо } a = b, \\ \frac{b+5}{a}, & \text{якщо } a < b; \end{cases}$	18)	$X = \begin{cases} \frac{a}{b} + 1, & \text{якщо } a > b, \\ a + 25, & \text{якщо } a = b, \\ \frac{a*b-2}{a}, & \text{якщо } a < b; \end{cases}$
19)	$X = \begin{cases} \frac{b}{a} + 10, & \text{якщо } a > b, \\ 3425, & \text{якщо } a = b, \\ \frac{2*a-5}{b}, & \text{якщо } a < b \end{cases}$	20)	$X = \begin{cases} \frac{a*a-b}{a}, & \text{якщо } a > b, \\ -a, & \text{якщо } a = b, \\ \frac{a*b-1}{b}, & \text{якщо } a < b; \end{cases}$
21)	$X = \begin{cases} \frac{b+1}{a}, & \text{якщо } a > b, \\ -b, & \text{якщо } a = b, \\ \frac{a-5}{b}, & \text{якщо } a < b; \end{cases}$	22)	$X = \begin{cases} \frac{a}{b} - 1, & \text{якщо } a < b, \\ 25 - a, & \text{якщо } a = b, \\ \frac{b-5}{a}, & \text{якщо } a > b; \end{cases}$
23)	$X = \begin{cases} \frac{b}{a} + 2, & \text{якщо } a > b, \\ -11, & \text{якщо } a = b, \\ \frac{a-8}{b}, & \text{якщо } a < b; \end{cases}$	24)	$X = \begin{cases} \frac{a}{b} + 2, & \text{якщо } a > b, \\ 8, & \text{якщо } a = b, \\ \frac{b-9}{a}, & \text{якщо } a < b \end{cases}$
25)	$X = \begin{cases} \frac{b+5}{a}, & \text{якщо } a < b, \\ -5, & \text{якщо } a = b, \\ \frac{b-5}{b}, & \text{якщо } a > b; \end{cases}$	26)	$X = \begin{cases} \frac{a}{b} + 1, & \text{якщо } a < b, \\ -71, & \text{якщо } a = b, \\ \frac{a-b}{a}, & \text{якщо } a > b; \end{cases}$
27)	$X = \begin{cases} \frac{b}{a} - 7, & \text{якщо } a > b, \\ 43, & \text{якщо } a = b, \\ \frac{a^3-b}{b}, & \text{якщо } a < b; \end{cases}$	28)	$X = \begin{cases} -5 + \frac{b}{a}, & \text{якщо } a > b, \\ 45, & \text{якщо } a = b, \\ \frac{3*a-6}{b}, & \text{якщо } a < b; \end{cases}$
29)	$X = \begin{cases} \frac{a}{b} + 7, & \text{якщо } a > b, \\ -125, & \text{якщо } a = b, \\ \frac{3*b+9}{a}, & \text{якщо } a < b; \end{cases}$	30)	$X = \begin{cases} \frac{a}{b} - 4, & \text{якщо } a < b, \\ -55, & \text{якщо } a = b, \\ \frac{b-5}{a}, & \text{якщо } a > b; \end{cases}$
31)	$X = \begin{cases} \frac{a}{b} + 20, & \text{якщо } a > b, \\ 110, & \text{якщо } a = b, \\ \frac{a-b}{a}, & \text{якщо } a < b; \end{cases}$	32)	$X = \begin{cases} \frac{a}{b} + 11, & \text{якщо } a > b, \\ -11, & \text{якщо } a = b, \\ \frac{3*b-9}{a}, & \text{якщо } a < b; \end{cases}$

33)	$X = \begin{cases} \frac{b}{a} - 5, & \text{якщо } a > b, \\ 22, & \text{якщо } a = b, \\ \frac{a-9}{b}, & \text{якщо } a < b; \end{cases}$	34)	$X = \begin{cases} 2 * \frac{a}{b} + 1, & \text{якщо } a > b, \\ -5, & \text{якщо } a = b, \\ \frac{a^3-9}{a}, & \text{якщо } a < b; \end{cases}$
35)	$X = \begin{cases} \frac{b}{a} + 2, & \text{якщо } a > b, \\ -57, & \text{якщо } a = b, \\ \frac{a-b}{b}, & \text{якщо } a < b; \end{cases}$	36)	$X = \begin{cases} \frac{b}{a} + 1, & \text{якщо } a > b, \\ -20, & \text{якщо } a = b, \\ \frac{a-45}{b}, & \text{якщо } a < b; \end{cases}$
37)	$X = \begin{cases} \frac{b}{a} + 1, & \text{якщо } a > b, \\ -44, & \text{якщо } a = b, \\ \frac{a^3+b}{b}, & \text{якщо } a < b; \end{cases}$	38)	$X = \begin{cases} \frac{b}{a} + 2, & \text{якщо } a > b, \\ -100, & \text{якщо } a = b, \\ \frac{b-100}{a}, & \text{якщо } a < b; \end{cases}$
39)	$X = \begin{cases} \frac{a}{b} + 201, & \text{якщо } a > b, \\ -800, & \text{якщо } a = b, \\ \frac{a-5}{a}, & \text{якщо } a < b; \end{cases}$	40)	$X = \begin{cases} \frac{a}{b} + 1, & \text{якщо } a > b, \\ -100, & \text{якщо } a = b, \\ \frac{a*b-9}{a}, & \text{якщо } a < b; \end{cases}$
41)	$X = \begin{cases} \frac{a}{b} + 1, & \text{якщо } a > b, \\ -425, & \text{якщо } a = b, \\ \frac{3*a-b}{a}, & \text{якщо } a < b; \end{cases}$	42)	$X = \begin{cases} \frac{a}{b} - a, & \text{якщо } a < b, \\ -b, & \text{якщо } a = b, \\ \frac{a*b-8}{a}, & \text{якщо } a > b; \end{cases}$
43)	$X = \begin{cases} \frac{a}{b} + 5, & \text{якщо } a < b, \\ -b, & \text{якщо } a = b, \\ \frac{b-9}{a}, & \text{якщо } a > b; \end{cases}$	44)	$X = \begin{cases} \frac{b^2}{a} + 1, & \text{якщо } a > b, \\ 2 * a, & \text{якщо } a = b, \\ \frac{a-10}{b}, & \text{якщо } a < b; \end{cases}$
45)	$X = \begin{cases} \frac{b}{a} + 8, & \text{якщо } a > b, \\ 455, & \text{якщо } a = b, \\ \frac{a-b}{b}, & \text{якщо } a < b; \end{cases}$	46)	$X = \begin{cases} \frac{b}{a} + 1, & \text{якщо } a > b, \\ 271, & \text{якщо } a = b, \\ \frac{a-b}{b}, & \text{якщо } a < b; \end{cases}$
47)	$X = \begin{cases} \frac{a}{b} - 37, & \text{якщо } a > b, \\ 3, & \text{якщо } a = b, \\ \frac{a^3-b}{a}, & \text{якщо } a < b; \end{cases}$	48)	$X = \begin{cases} \frac{b}{a} - 5, & \text{якщо } a > b, \\ -195, & \text{якщо } a = b, \\ \frac{a-6}{b}, & \text{якщо } a < b; \end{cases}$
49)	$X = \begin{cases} \frac{a}{b} - 7, & \text{якщо } a > b, \\ -195, & \text{якщо } a = b, \\ \frac{b^3-9}{a}, & \text{якщо } a < b; \end{cases}$	50)	$X = \begin{cases} \frac{a}{b} - 4, & \text{якщо } a < b, \\ -155, & \text{якщо } a = b, \\ \frac{b^3-5}{a}, & \text{якщо } a > b; \end{cases}$
51)	$X = \begin{cases} \frac{a}{b} - 42, & \text{якщо } a < b, \\ -11, & \text{якщо } a = b, \\ \frac{a^3-8}{a}, & \text{якщо } a > b; \end{cases}$	52)	$X = \begin{cases} \frac{b}{a} - 141, & \text{якщо } a < b, \\ -131, & \text{якщо } a = b, \\ \frac{a-6}{b}, & \text{якщо } a > b; \end{cases}$

3 Зміст звіту

3.1 Назва та мета роботи

3.2 Блок-схема алгоритму програми згідно свого варіанту Turbo Assembler

3.3 Текс програми

3.4 Результат виконання програми

3.5 Висновки по роботі

4 Контрольні питання

4.1 Назвіть безумовні переходи?

4.2 Назвіть команду циклу?

4.3 Назвіть умовні переходи?

4.4 Назвіть команду виклику процедури?

4.5 Назвіть знакові умовні переходи?

4.6 Назвіть беззнакові умовні переходи?