Міністерство освіти та науки України
Національний університет «Чернігівська політехніка»
Навчально-науковий інститут електронних та інформаційних технологій
Кафедра інформаційних та комп'ютерних систем

Модуль «Асемблерна програма для обробки рядків»

Програмний проект

Виконав:
студент групи КІ-211 Книга Дмитрій Сергійович

Керівник:
професор, д.т.н. Лисенко Д.Е.

2023

Чернігів

Визначення	Назва	Примітки
КІ 211.13	**Документація**	
	Практичне завдання	
	Процедура одержує як параметр адресу sz-рядка, показує його на екрані. Цей же рядок в пам’яти записує в зворотньому порядку и показує його на екрані.	
КІ 211.7		
	Рефератна частина	
	Двійкова, десяткова та шістнадцяткова системи числення. Перевод цілих і натуральних чисел між цими системами	
КІ 211.11	**Комплекси**	
	Операційна система Microsoft Windows 11	
	Середовище Notepad++	
	Компілятор ассемблера	

Міністерство освіти та науки України
Національний університет «Чернігівська політехніка»
Навчально-науковий інститут електронних та інформаційних технологій
Кафедра інформаційних та комп'ютерних систем

Модуль «Асемблерна програма для обробки рядків»

Реферат

На тему: «Двійкова, десяткова та шістнадцяткова системи числення. Перевод цілих і натуральних чисел між цими системами»

Листів 14

Виконав:
студент групи КІ-211 Книга Дмитрій Сергійович

Керівник:
професор, д.т.н. Лисенко Д.Е.

2023

Чернігів

ЗМІСТ

1 ВСТУП

В сучасному світі, де інформаційні технології є невід'ємною частиною нашого життя, розуміння систем числення та їх перетворення стає дедалі важливішим. Однією з основних систем числення є десяткова система, яка використовує десять цифр (0-9) та базується на розрядному представленні чисел. Проте, десяткова система не є єдиною, адже у деяких сферах використовуються інші системи числення, такі як двійкова та шістнадцяткова.

Системи числення - це способи представлення чисел за допомогою визначеної кількості символів (цифр) та правил запису. Кожна система числення має свою власну базу (кількість символів, що використовуються), а також специфічні правила запису та обчислення.

Двійкова система числення використовує дві цифри (0 та 1) і використовується в електроніці, комп'ютерах та інших технічних пристроях. Шістнадцяткова система числення також є важливою у технічних галузях, але використовує шістнадцять цифр (0-9, а також A, B, C, D, E, F) для представлення чисел. Використання цих систем числення є необхідним для ефективної роботи з даними та забезпечення правильної передачі інформації.

Поняття переводу чисел між різними системами числення також є важливим для розуміння роботи комп'ютерних систем та інших технічних пристроїв. У цьому рефераті ми розглянемо особливості двійкової, десяткової та шістнадцяткової систем числення, а також навчимося переводити цілі та натуральні числа між ними. Знання цих принципів допоможе нам краще розуміти технології, що оточують нас у повсякденному житті, та дозволить ефективніше використовувати їх для вирішення різних задач.

2 ПЕРЕВЕДЕННЯ ЧИСЕЛ З 10 У 2 СИСТЕМИ

Переведення чисел з 10-ї системи у 2-у відбувається шляхом постійного ділення числа на 2[1]. Нижче на рисунках 1 та 2 наведено приклади:

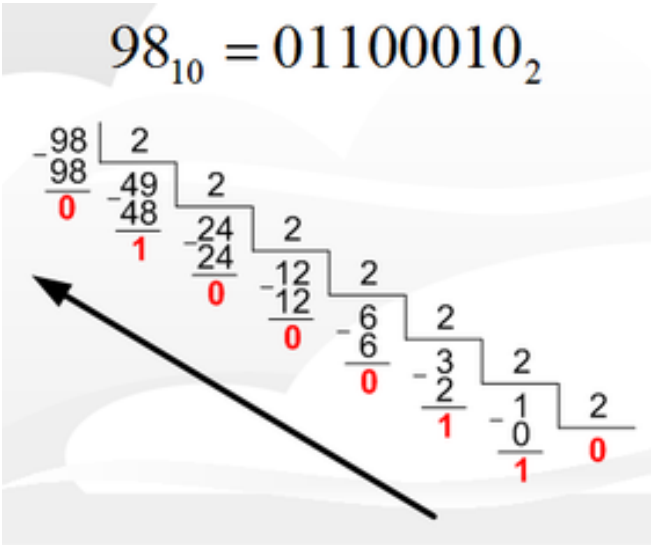


Рисунок 1 – Переведення з десяткової системи (98_{10}) у двійкову (01100010_2)

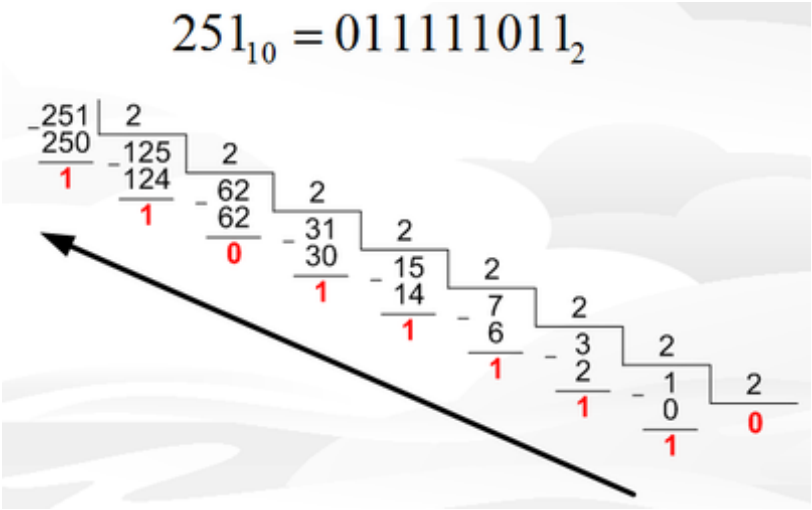


Рисунок 2 – Переведення з десяткової с истеми (251_{10}) у двійкову (011111011_2)

3 ПЕРЕВЕДЕННЯ ЧИСЕЛ З 10 У 2 СИСТЕМИ ЧЕРЕЗ АСЕМБЛЕР

За замовчуванням асемблер вважає всі числа, що зустрічаються в програмі, десятковими. Але інші системи числення можна вказати за допомогою тегів[3]:

b або y - для двійкових чисел;

o або q - для вісімкових;

d або t - для десяткових чисел;

h - для шістнадцяткових чисел.

Тег записується в кінці числа, разом із числом. Якщо в числі використовуються літерні символи (шістнадцяткові числа), на початку записують нуль - за правилами асемблера позначення чисел повинні починатися з цифри. Приклад наведено на рисунку 3:

```
.data
    var1 byte 00001111b ; 15 в двоичном представлении
    var2 byte 00001111y ; 15 в двоичном представлении
    var3 byte 17o ; 15 в восьмеричном представлении
    var4 byte 17q ; 15 в восьмеричном представлении
    var5 byte 15d ; 15 в десятичном представлении
    var6 byte 15t ; 15 в десятичном представлении
    var7 byte 0Fh ; 15 в шестнадцатеричном представлении
```

Рисунок 3 - Позначення чисел у асемблері

Переведення числа з десяткової системи у двійкову через ассемблер можна здійснити наступним чином:

1. Зберегти число, яке необхідно перевести, у регістр
2. Ініціювати змінну-лічильник зі значенням 0
3. Ініціювати змінну-результат зі значенням 0
4. Запустити цикл ділення числа на 2, доки число не стане рівним 0
5. На кожному кроці циклу:
    - Ділити число на 2
    - Записувати остачу від ділення до змінної-результату
    - Збільшувати лічильник на 1
6. Записати результат у зворотньому порядку від останнього біту до першого.

Ось приклад коду на мові асемблера для процесора x86-64, який переводить число з десяткової системи у двійкову:

```
.data
number dd 25
result times 32 db 0 ; змінна для зберігання результату
```

```
counter db 0 ; змінна-лічильник

.text
global _start

start:
mov eax, [number] ; зберегти число в регістр
mov ebx, 2 ; ділити на 2

loop_start:
cmp eax, 0 ; перевірити, чи число не дійшло до 0
je loop_end

xor edx, edx ; обнулити edx для ділення
div ebx ; поділити eax на ebx
mov [result + counter], dl ; записати остачу в результат
inc counter ; збільшити лічильник
jmp loop_start

loop_end:
; тут в змінній result знаходиться результат у зворотньому порядку
```

4 ПЕРЕВЕДЕННЯ ЧИСЕЛ З 2 У 10 СИСТЕМУ

Якщо зіставити числа з літерами, можна записувати текст у цифровій формі. Цей код використовує регістри процесора x86-64 (eax, ebx, edx) для збереження чисел і результатів дій. Лічильник зберігається у змінній типу "byte" (db), а результат у змінній типу "byte array" (times 32 db). Код записує остачі від ділення до змінної-результату в зворотньому порядку, тому його можна перевернути, щоб отримати результат у правильному порядку.

Кольори є комбінацією інтенсивності світлових потоків червоного, синього і зеленого - цю інтенсивність можна також задати в числових значеннях. Зображення легко уявити у вигляді мозаїки з кольорових квадратів, так що вони теж можуть бути представлені через числа.

Тобто, будь-які дані - це так чи інакше закодована інформація. Для зберігання інформації комп'ютер використовує **пам'ять.** Пам'ять комп'ютера поділена на багато комірок, які називаються **байтами**. Кожна з комірок зберігає невеликий обсяг даних і має числову адресу. Одного байта пам'яті вистачає лише для зберігання букви або невеликого числа. Для зберігання значних обсягів даних використовують мільярди байтів пам'яті. Кожен байт поділений на **вісім** менших елементів, які називаються **бітами**[6].

Термін bit *(від* англ. **bi**nary digi**t***) означає* **двійкова цифра**.

Біти можна уявити у вигляді «кнопок-перемикачів», які можуть бути або «увімкненими» або «вимкненими». У комп'ютерній техніці біти є **електричними компонентами**, які можуть мати електричний заряд, або ні. Наявність заряду можна ототожнити з перемикачем у ввімкненому положенні, а відсутність заряду - перемикачем у вимкненому положенні.

Тобто, **байт пам'яті** можна представити як набір перемикачів, кожен з яких переміщений або у положення ввімкнення (ON), або вимкнення (OFF). Наприклад, якщо в байті необхідно зберегти число 81, то перемикачі будуть розміщені у таких положеннях, приклад наведено на рисунку 4:

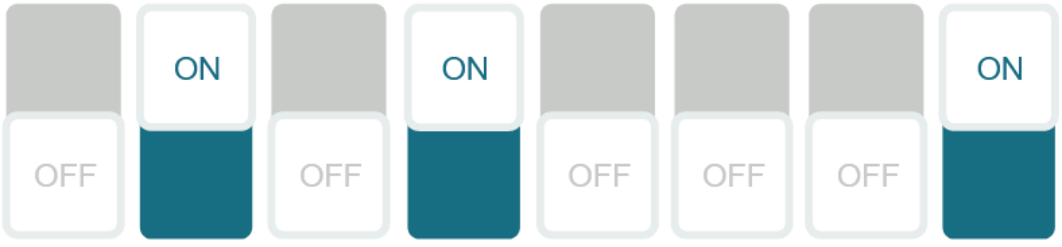


Рисунок 4 – **Байт пам'яті** як набір перемикачів: положення біт-перемикачів для зберігання числа **81**

Для збереження у байті англійської літери M положення перемикачів будуть наступними, зображено на рисунку 5:

Рисунок 5 – Приклад байту пам'яті Положення біт-перемикачів для зберігання у байті англійської літери **M**

Залежно від того, чи «ввімкнено» чи «вимкнено» біт, він може мати одне з двох різних значень:

- 0 (якщо біт «вимкнено»);
- 1 (якщо біт «увімкнено»).

Така **система числення**, яка складається з двох символів (у даному випадку 0 і 1), називається **двійковою** або **бінарною**.

Ось приклад числа, написаного в двійковій системі числення:

01110011

В нашому випадку, запис числа у двійковій системі числення буде наступним: Позиція кожної цифри 0 або 1 (**розряд**) у бінарному числі має своє власне значення. А саме, якщо переміщуватись ліворуч, починаючи від крайньої правої цифри, значення кожної позиції записують наступним чином (2 - **основа двійкової системи числення**)[6], зображено на рисунку 6:

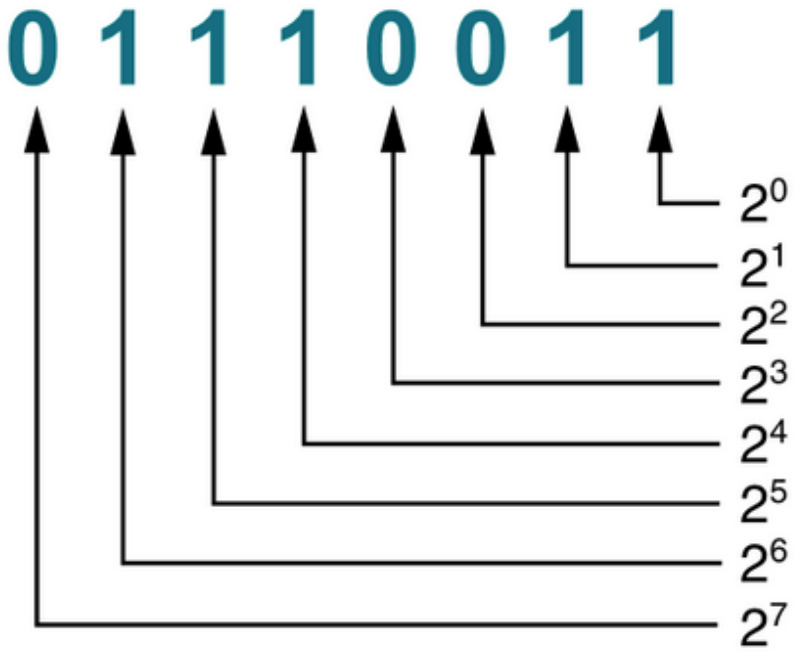


Рисунок 6 - Запис значень позицій двійкового числа **01110011**

Відповідно, якщо виконати обчислення (піднесення до степеня), значення кожної позиції будуть такими[6], зображено на рисунку 7:

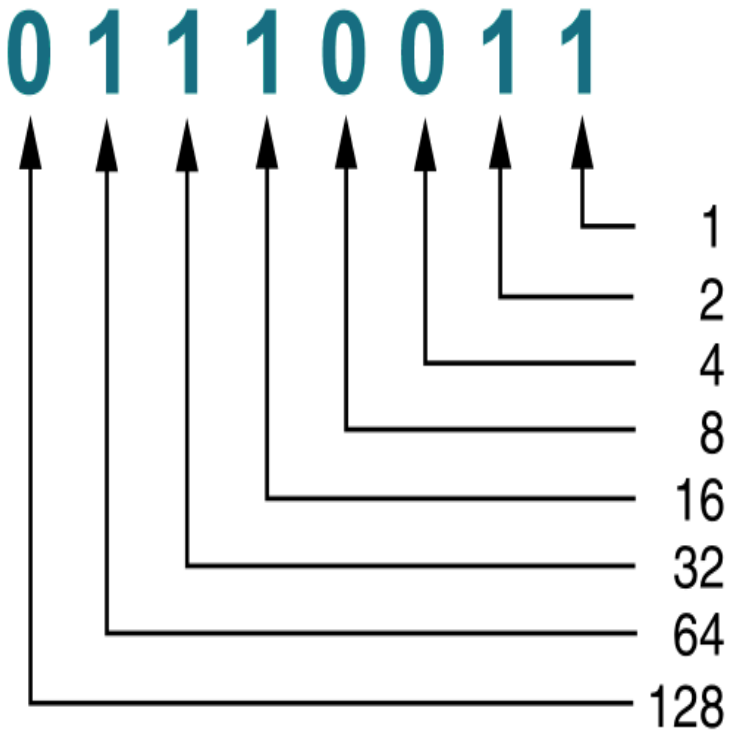


Рисунок 7 - Значення позицій двійкового числа **01110011**

Щоб визначити **десяткове значення** двійкового числа, необхідно додати значення позицій для всіх 1[6], зображено на рисунку 8:

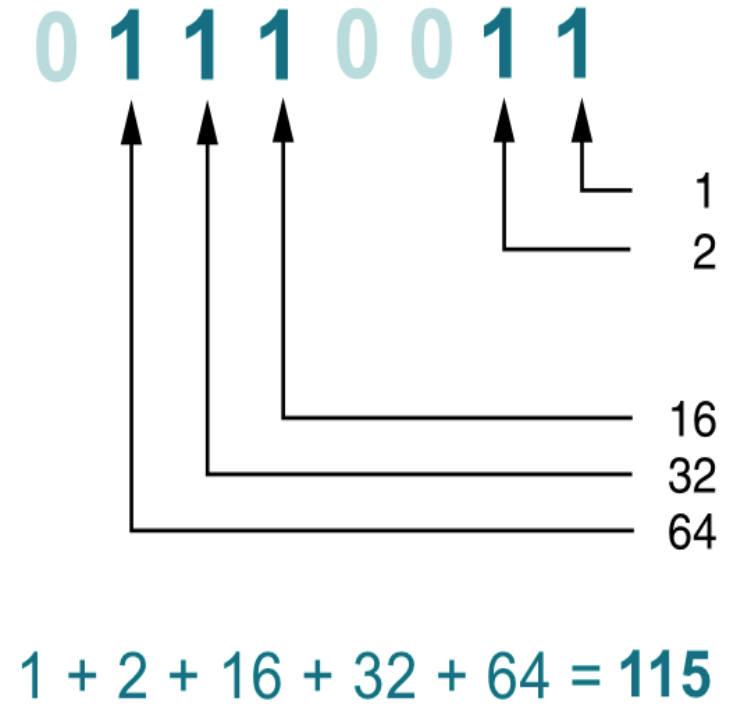


Рисунок 8 - Сумування значень позицій двійкового числа **01110011**

Сума усіх цих значень позицій становить 115. Таким чином, значення **двійкового числа** 01110011_2 становить 115_{10} у **десятковій системі числення**, якою користується людина в повсякденному житті. Число 115, що зберігається в байті пам'яті[6], можна зобразити таким чином на рисунку 9.

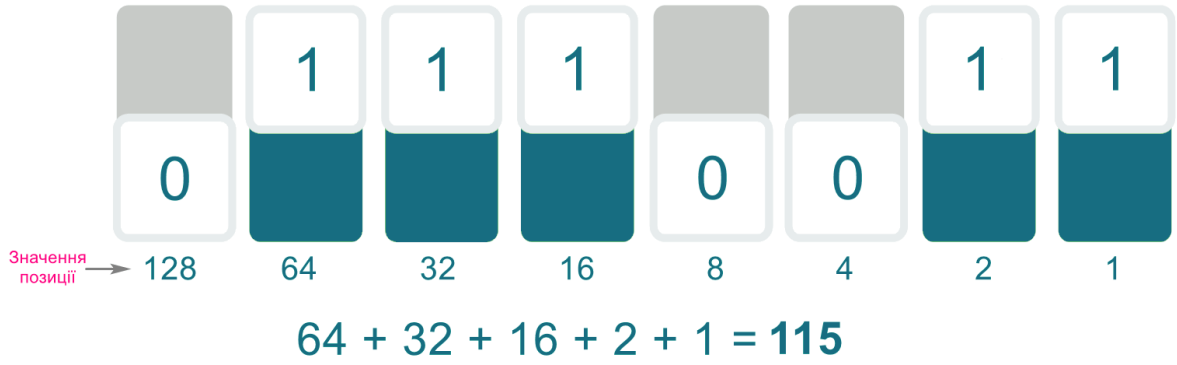


Рисунок 9 - Значення позицій двійкового числа **01110011** і біти-перемикачі

5 ПЕРЕВЕДЕННЯ ЧИСЕЛ З 2 У 10 СИСТЕМУ ЧЕРЕЗ АСЕМБЛЕР

Переведення чисел з двійкової системи числення в десяткову систему можна здійснити за допомогою алгоритму, що базується на позиційній системі числення. У цьому алгоритмі кожна цифра числа множиться на потужність основи (2) відповідної позиції і додається до загальної суми.

В асемблері, x64, masm 64 можна реалізувати цей алгоритм за допомогою послідовності операцій, які множать кожну цифру двійкового числа на відповідну потужність 2 та додають отриманий добуток до загальної суми.

Наприклад, для перетворення двійкового числа "1101" в десяткове, можна виконати наступні кроки в асемблері:

1. Зберігаємо двійкове число в регістрі, наприклад, AL:

```
mov al, 0b1101
```

2. Ініціалізуємо змінну для зберігання загальної суми:

```
xor ebx, ebx
```

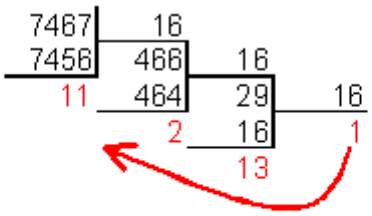
3. Переводимо двійкове число у десяткове, застосовуючи алгоритм позиційної системи числення. Для цього множимо кожну цифру числа на відповідну потужність двійки та додаємо до загальної суми:

```
mov ecx, 0    ; змінна для підрахунку позиції числа
mov edx, 2    ; потужність числа 2
loop_start:
and al, 1   ; взяти останню цифру числа
mul edx    ; помножити на потужність числа 2
add ebx, eax ; додати отриманий добуток до загальної суми
shr al, 1 ; зсунути число на одну позицію вправо
inc ecx ; збільшити лічильник позиції
cmp al, 0 ; перевірити, чи закінчився процес перетворення
jne loop_start ; якщо ні, продовжити цикл
```

6 ПЕРЕВЕДЕННЯ ЧИСЕЛ З 10 У 16 СИСТЕМУ

Для переведення десяткового числа в шістнадцяткову систему його необхідно послідовно ділити на 16 до тих пір, поки не залишиться залишок, менший або рівний 15. Число в шістнадцятковій системі записується як послідовність цифр останнього результату ділення і залишків від ділення в зворотному порядку[1].

Приклад переведення числа 746710 та 500 в шістнадцяткову систему зображено на рисунках 10 та 11.



$7467_{10} = 1D2B_{16}$

Рисунок 10 – Приклад переведення з 10-ї в 16-у систему

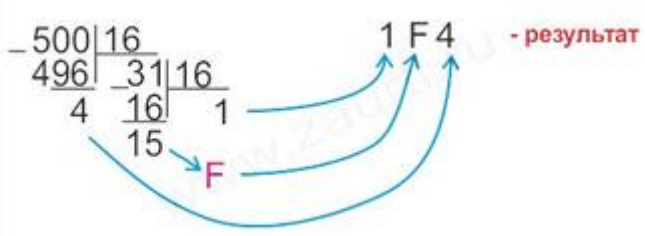


Рисунок 11 – Приклад переведення з 10-ї в 16-у систему

7 ПЕРЕВЕДЕННЯ ЧИСЕЛ З 10 У 16 СИСТЕМУ ЧЕРЕЗ АСЕМБЛЕР

Переведення чисел з десяткової системи числення в шістнадцяткову можна здійснити за допомогою алгоритму, що також базується на позиційній системі числення. У цьому алгоритмі кожне число ділиться на основу системи числення (16), отримуючи залишок та цілу частину. Остання конвертується в відповідну шістнадцяткову цифру та записується у вихідний рядок. Цей процес повторюється, поки число не стане меншим за основу системи числення.

В асемблері, x64, masm 64 можна реалізувати цей алгоритм за допомогою послідовності операцій, які ділять число на 16 та записують отримані залишок та цілу частину у вихідний рядок у відповідному форматі.

Наприклад, для перетворення десяткового числа "255" в шістнадцяткове, можна виконати наступні кроки в асемблері:

1. Зберігаємо десяткове число в регістрі, наприклад, AX:

```
mov ax, 255
```

2. Ініціалізуємо змінну для зберігання рядка у шістнадцятковому форматі та лічильник для позиції у рядку:

```
mov esi, offset hex_string   ; адреса рядка у шістнадцятковому форматі
xor ecx, ecx                 ; лічильник позиції у рядку
```

3. Переводимо десяткове число у шістнадцяткове, застосовуючи алгоритм позиційної системи числення. Для цього ділимо число на 16 та записуємо отримані залишок та цілу частину у вихідний рядок у відповідному форматі:

```
loop_start:
xor edx, edx     ; очистити регістр після попередньої операції
div word ptr 16  ; ділити число на 16
push dx          ; зберегти отриманий залишок у стеку
```

8 ПЕРЕВЕДЕННЯ ЧИСЕЛ З 16 У 10 СИСТЕМУ

Для переводу шістнадцятирічного числа в десяткове необхідно його записати у вигляді многочлена, що складається з творів цифр числа і відповідного ступеня числа 16, і обчислити за правилами десяткової арифметики[1]:

$$X_{16} = A_n \cdot 16^{n-1} + A_{n-1} \cdot 16^{n-2} + A_{n-2} \cdot 16^{n-3} + ... + A_2 \cdot 16^1 + A_1 \cdot 16^0$$

Приклад переведення числа $\mathrm{FDA1}_{16}$ в десяткову систему числення зображено на рисунку 1.

$$\mathrm{FDA1}_{16} = 15 \cdot 16^3 + 13 \cdot 16^2 + 10 \cdot 16^1 + 1 \cdot 16^0 = 64929_{10}$$

Рисунок 1 – Переведення з шістнадцятирічного числа в десяткове

9 ПЕРЕВЕДЕННЯ ЧИСЕЛ З 16 У 10 СИСТЕМУ ЧЕРЕЗ АСЕМБЛЕР

Переведення шістнадцятирічного числа в десяткове можна виконати за допомогою алгоритму переведення зі старших розрядів до молодших, який включає наступні кроки[4]:

1. Ініціалізація змінних:

```
mov ecx, 0          ; регістр, що відповідає за поточну позицію у шістнадцятковому числі
mov edx, 0          ; регістр, що зберігає проміжний результат у десятковому вигляді
```

2. Зчитування шістнадцяткового числа:

```
mov ebx, hex_number ; ebx містить адресу першого байту шістнадцяткового числа
mov al, byte ptr [ebx+ecx] ; зчитати шістнадцяткову цифру у поточній позиції
```

3. Перетворення шістнадцяткової цифри у десяткову цифру:

```
cmp al, '0'      ; перевірити, чи є цифра меншою за '0'
jb skip          ; якщо так, пропустити
cmp al, '9'      ; перевірити, чи є цифра більшою за '9'
ja skip          ; якщо так, пропустити
sub al, '0'      ; відняти код символу '0', щоб отримати десяткову цифру
jmp convert_done ; перейти до кроку 5
```

4. Перетворення шістнадцяткової літери у десяткову цифру:

```
cmp al, 'a'      ; перевірити, чи є літера меншою за 'a'
jb skip          ; якщо так, пропустити
cmp al, 'f'      ; перевірити, чи є літера більшою за 'f'
ja skip          ; якщо так, пропустити
sub al, 'a' - 10 ; відняти код символу 'a' і додати 10, щоб отримати десяткову цифру
jmp convert_done ; перейти до кроку 5
```

5. Додавання десяткової цифри до проміжного результату:

```
mov bl, al       ; зберегти десяткову цифру у регістрі
```

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Komplogika.URL:https://komplogika.jimdofree.com/%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8-%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%8F/%D0%BF%D0%B5%D1%80%D0%B5%D0%B2%D1%96%D0%B4-%D1%87%D0%B8%D1%81%D0%B5%D0%BB/

2. Системи числення: алгоритми переведення чисел з однієї позиційної системи числення в іншу. URL https://sites.google.com/site/sistemicislenna/algoritmi-perevedenna-cisel-z-odniei-pozicijnoi-sistemi-cislenna-v-insu

3. Системы счисления. URL: https://asmbase.ru/uchebnyj-kurs/007-sistemy-schisleniya/

4. Ивашин, В., і Коваленко, А. (2013). Ассемблер и принципы программирования (3-е изд.). К.: Издательство "Факт".

5. Зуев, И., і Сергеев, В. (2012). Ассемблер для процессоров Intel. СПб: Питер.

6. Системы счисления. URL: https://nrs.rozh2sch.org.ua/

Міністерство освіти та науки України
Національний університет «Чернігівська політехніка»
Навчально-науковий інститут електронних та інформаційних технологій
Кафедра інформаційних та комп’ютерних систем

Модуль «Асемблерна програма для обробки рядків»

Програмний проект

Листів 10

Виконав:
студент групи КІ-211 Книга Дмитрій Сергійович

Керівник:
професор, д.т.н. Лисенко Д.Е.

2023

Чернігів

АНОТАЦІЯ

практичної Книги Дмитрія Сергійовича на тему: «Асемблерна програма для обробки рядків»

Об'єктом роботи є програма, написана на асемблері, призначена для обробки рядків.

Метою роботи є розробка програми, яка здійснює виведення рядка на екран у звичайному порядку та запис його в пам'ять у зворотньому порядку, а також виведення отриманого результату на екран.

Розглядається реалізація такої програми, яка може бути використана, де потрібно обробляти рядки.

У ході виконання роботи потрібно:

• описати алгоритм програми та її структуру(псевдокод);

• створити блок-схему програми;

• розробити та реалізувати програму на асемблері для обробки рядків;

• продемонструвати роботу програми.

Виконання роботи проводилось у відповідності з поставленими вимогами.

Результатом роботи є розроблена та протестована програма на асемблері для обробки рядків,, що може бути використана, де потрібно обробляти рядки.

ЗМІСТ

1 АНАЛІЗ ОТРИМАНОГО ЗАВДАННЯ

1.1 Завдання

Процедура одержує як параметр адресу sz-рядка, показує його на екрані. Цей же рядок в пам’яти записує в зворотньому порядку и показує його на екрані.

Згідно до завдання треба написати програму, що отримує рядок з нульовим символом в кінці. Програма має інвертувати його й результат має виводитися на екран.

1.2 Псевдокод

Псевдокод - це неформальний запис алгоритму, який використовує структуру поширених мов програмування, але нехтує деталями коду, неістотними для розуміння алгоритму (опис типів, виклик підпрограм тощо). Мова програмування доповнюється природною мовою, компактними математичними позначеннями. Псевдокод є зрозумілішим, ніж програми, формою запису алгоритмів.

Псевдокод, що буде наведений нижче стане прообразом програми наведений, відповідно до завдання у лістингу 1.1.

Лістинг 1.1 – Псевдокод

```
Declare hInstance, hIcon, tEdit, pbuff, buff as DWORD

entry_point:
        hInstance = GetModuleHandle(0)
        hIcon = LoadImage(hInstance, 10, IMAGE_ICON, 256, 256, LR_DEFAULTCOLOR)
        DialogBoxParam(hInstance, 10, 0, PartsWindow, hIcon)
        exit

PartsWindow:
        buffer[260] = ""
        switch (uMsg)
                case WM_INITDIALOG:
                        SendMessage(hWin, WM_SETICON, 1, lParam)
                        tEdit = GetDlgItem(hWin, 11)
                        SetFocus(tEdit)
                        pbuff = buffer
                case WM_COMMAND:
                        switch (wParam)
                                case Revers:
                                        GetWindowText(tEdit, pbuff, sizeof buffer)
```

```
                        if (rax == 0)
                                MessageBox(hWin, "Введіть текст або натисніть Закрити вікно", "Текст не введено", MB_ICONINFORMATION)
                                SetFocus(tEdit)
                        else
                                rbx = 0
                                rcx = sizeof buffer
                                rsi = pbuff
                                n:
                                        if (byte ptr [rsi] == 0) goto m
                                        rsi = rsi + 1
                                        rbx = rbx + 1
                                        rcx = rcx - 1
                                        goto n
                                m:
                                rcx = rbx
                                rsi = rsi - 1
                                rdi = buff
                                loopp:
                                        al = byte ptr [rsi]
                                        byte ptr [rdi] = al
                                        rdi = rdi + 1
                                        rsi = rsi - 1
                                        loop loopp
                                MessageBox(0, addr buff, "Текст у зворотньому порядку", MB_ICONINFORMATION)
                        case Close:
                                goto exit_dialog
                endswitch
        case WM_CLOSE:
                exit_dialog:
                EndDialog(hWin, 0)
        endswitch
        rax = 0
        ret
```

1.3 Блок-схема

Блок-схема – це представлення алгоритму розв’язання або аналізу задачі за допомогою геометричних елементів (блоків), які позначають операції, потік, дані тощо.

Блок-схема, відповідно до завдання, наведена на рисунку 1.1

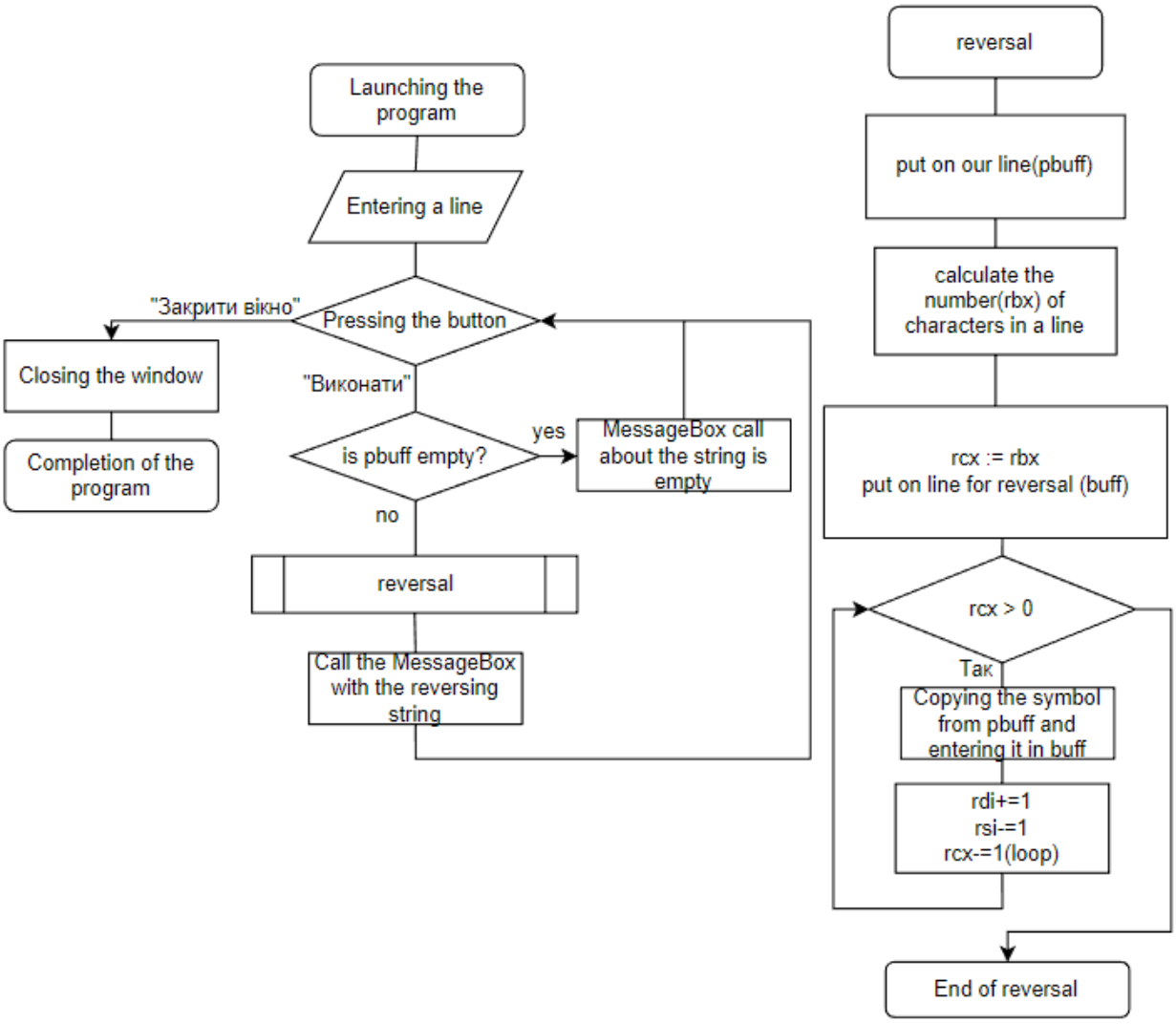


Рисунок 1.1 – Блок-схема програми

2 ВИКОНАННЯ ПРОЕКТУ

2.1 Код програми

Код програми, відповідно до завдання, наведений в лістингу 2.1

Лістинг 2.1 – Код програми

```
include \masm64\include64\masm64rt.inc

.data?
    hInstance dq ?
    hIcon dq ?
    tEdit dq ?
    pbuff dq ?
    buff dq ?

.code
entry_point proc
    mov hInstance, rvcall(GetModuleHandle,0)
    mov hIcon, rv(LoadImage,hInstance,10,IMAGE_ICON,256,256,
LR_DEFAULTCOLOR)
    invoke DialogBoxParam,hInstance,10,0,ADDR
PartsWindow,hIcon
    .exit
entry_point endp

PartsWindow proc
hWin:QWORD,uMsg:QWORD,wParam:QWORD,lParam:QWORD
    LOCAL buffer[260]:BYTE
    .switch uMsg
    .case WM_INITDIALOG
        invoke SendMessage,hWin,WM_SETICON,1,lParam
        mov tEdit, rvcall(GetDlgItem,hWin,11)
        invoke SetFocus, tEdit
        mov pbuff, ptr$(buffer)
    .case WM_COMMAND
  .switch wParam
    .case 12
        invoke GetWindowText,tEdit,pbuff,sizeof buffer
        .if rax == 0
            rcall MessageBox,hWin,"Введіть текст або натисніть
Закрити вікно","Текст не введено",MB_ICONINFORMATION
            rcall SetFocus,tEdit
        .else
            xor rbx,rbx
            mov rcx,sizeof buffer
            mov rsi,pbuff
        n:
            cmp byte ptr [rsi],0
            jz m
            inc rsi
            inc rbx
```

```
            dec rcx
            jnz n
        m:
            mov rcx,rbx
            dec rsi
            lea rdi,buff

        loopp:

            mov al,byte ptr [rsi]
            mov byte ptr [rdi],al
            inc rdi
            dec rsi
            loop loopp

            invoke MessageBox,0,addr buff,"Текст у зворотньому
порядку",MB_ICONINFORMATION
        .endif
    .case 13
    jmp exit_dialog
    .endsw
    .case WM_CLOSE
    exit_dialog:
    rcall EndDialog,hWin,0
    .endsw
    xor rax, rax
    ret
PartsWindow endp
end
```

Rc-файл наведений в лістингу 2.2

Лістинг 2.2 – Rc-файл

```
#define IDD_DLG 10
#define IDC_Edit 11
#define IDC_Revers 12
#define IDC_Close 13
#define IDC_STC 14

#include "/masm64/include64/resource.h"

10 ICON DISCARDABLE "icon.ico"

IDD_DLG DIALOGEX 0,-12,165,45
CAPTION "Виведення тексту у зворотньому порядку"
FONT 10,"Segoe UI",400,0,0
STYLE WS_POPUP|WS_VISIBLE|WS_CAPTION|WS_SYSMENU|DS_CENTER
BEGIN
  CONTROL
"",IDC_Edit,"Edit",WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,9,12,1
45,10,WS_EX_CLIENTEDGE
```

```
  CONTROL
"Виконати",IDC_Revers,"Button",WS_CHILDWINDOW|WS_VISIBLE|WS_TA
BSTOP,15,27,50,12
  CONTROL "Закрити
вікно",IDC_Close,"Button",WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP
,99,27,50,12
  CONTROL "Поле
введення",IDC_STC,"Static",WS_CHILDWINDOW|WS_VISIBLE,15,3,87,9
END
```

2.2 Результат виконання програми

На рисунках 2.1 – 2.3 нижче програма запускається декілька разів:

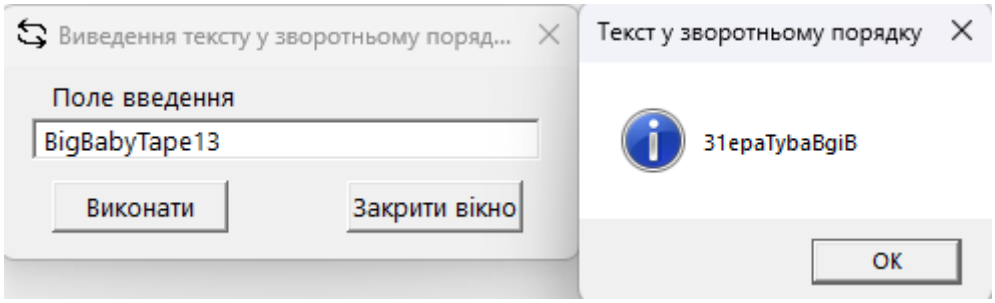


Рисунок 2.1 – Виконання програми анлійськими літерами

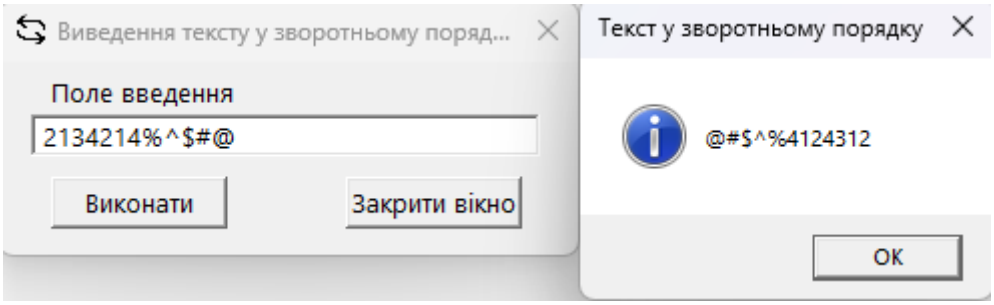


Рисунок 2.2 – Виконання програми числами та знаками

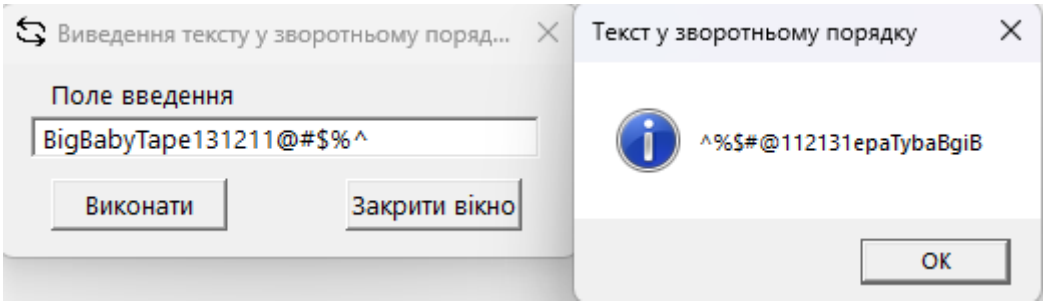


Рисунок 2.3 – Виконання програми з літерами, числами та знаками

ВИСНОВКИ

В ході виконання курсової роботи була досягнута основна її мета – розробити програму, написану на асемблері, яка друкує рядок на екран у звичайному порядку, записує його в пам’ять у зворотному порядку та відображає результат на екрані.

Згідно з поставленою задачею було описано алгоритм програми та її структуру, створено блок-схему для більшого розуміння та простішого виконання завдання.

Програма має діалогове вікно з текстовим полем і двома кнопками. Можна ввести текст у текстове поле, натиснути кнопку «Виконати», щоб змінити текст у зворотному напрямку та відобразити його у вікні, або натиснути кнопку «Закрити вікно», щоб закрити вікно.

Правильність роботи здійснювалася шляхом запуску програми і кожного разу введення різних символів.

Отже, можна зробити висновок, що мета курсової роботи була досягнута успішно. Було розроблено програму на асемблері, яка здійснює обробку рядків, та проведено тестування її роботи.