МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

МЕТОДИЧНІ ВКАЗІВКИ
ЩОДО ВИКОНАННЯ ПРАКТИЧНИХ РОБІТ
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
«СИСТЕМНЕ ПРОГРАМУВАННЯ»
ДЛЯ СТУДЕНТІВ ДЕННОЇ ТА ЗАОЧНОЇ ФОРМ НАВЧАННЯ
ЗІ СПЕЦІАЛЬНОСТІ 123 – «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

КРЕМЕНЧУК 2021

Методичні вказівки щодо виконання практичних робіт з навчальної дисципліни «Системне програмування» для студентів денної та заочної форм навчання зі спеціальності 123 – «Комп'ютерна інженерія».

Укладач старш. викл. Ю. В. Зілінський
Рецензент к. т. н., доц. В. М. Сидоренко

Кафедра комп'ютерних та інформаційних систем

Затверджено методичною радою Кременчуцького національного університету імені Михайла Остроградського
Протокол № _____ від « ___ » _____ 20___ р.
Голова методичної ради _____ проф. В. В. Костін

ЗМІСТ

ВСТУП

Методичні вказівки містять три практичні роботи, передбачені для виконання у четвертому навчальному семестрі робочою навчальною програмою з навчальної дисципліни «Системне програмування» для студентів денної та заочної форм навчання зі спеціальності 123 – «Комп'ютерна інженерія».

Тематика практичних робіт спирається на тематику лабораторних робіт з навчальної дисципліни «Системне програмування» і не передбачає вивчення нових команд мікропроцесора. Матеріали практичних робіт орієнтовані на вивчення алгоритмів використання вже відомих з лабораторного практикуму команд мікропроцесора.

Перша практична робота присвячена вивченню алгоритмів перетворення чисел в різних системах числення із символьного формату в машинне подання і алгоритмів зворотних перетворень.

Система команд мікропроцесора та апаратура комп'ютера дозволяють виконувати операції введення/виведення інформації, поданої тільки у символьному вигляді (окремі символи або рядки символів), тому перед їх виконанням необхідно самостійно передбачити в програмі потрібні перетворення даних.

Друга практична робота присвячена вивченню алгоритмів «довгої» цілочисельної арифметики підвищеної точності.

Мови високого рівня зазвичай обмежені в наборі типів даних, з якими вони можуть працювати. Для зберігання цілих чисел застосовують окремі байти, слова, подвійні слова або подвійні подвійні слова. Використовуючи асемблер, можна придумати тип даних абсолютно будь-якого розміру (128 біт, 256 біт, …, 1024 біта, …) і легко визначити всі арифметичні операції з такими числами. Арифметику для багаторозрядних цілих чисел використовують, наприклад, у класі задач криптографії та криптоаналізу.

Третя практична робота присвячена вивченню алгоритмів арифметики з фіксованою комою.

Існує певний клас задач, де потрібно виконувати обчислення з дійсними числами, до того ж, бажано робити це дуже швидко, а висока точність результатів непотрібна. Використання для таких розрахунків команд арифметичного співпроцесора недоцільне, або може бути й зовсім неприйнятне.

Таким вимогам, наприклад, відповідають деякі задачі комп'ютерної графіки. Ураховуючи растровий принцип формування зображення на екрані монітора, похибка обчислень координат певного пікселю екрана, що може виникати у знакових розрядах дробової частини числа, тут не має вирішального значення, адже отримані результати розрахунків усе одно доведеться перетворювати на ціле число, що відповідає екранним координатам.

Окрім того, матеріали другої та третьої практичних робіт можуть бути використанні для розробки програмного забезпечення для мікроконтролерів. Використання алгоритмів «довгої» арифметики може значно розширити обчислювальні можливості 8- та 16-розрядних мікроконтролерів. Використання алгоритмів арифметики з фіксованою комою надає додаткові можливості мікроконтролерам, які не мають блока арифметики з плаваючою комою.

Під час практичної роботи студент ознайомлюється з теоретичним матеріалом, вивчає наведені приклади, розробляє алгоритм написання програми згідно з індивідуальним завданням.

За результатами виконання завдання практичної роботи студент складає звіт. Залік із практичної роботи студент одержує після співбесіди з викладачем, під час якої він повинен продемонструвати теоретичні знання з певної теми, обґрунтувати вибір алгоритму розв'язання задачі, надати вичерпні пояснення за текстом розробленої програми.

1 ПЕРЕЛІК ПРАКТИЧНИХ РОБІТ

Практична робота № 1

Тема. Організація операцій уведення/виведення чисельних даних

Мета: вивчення і набуття практичних навичок реалізації алгоритмів перетворення рядків символів на машинне зображення і зворотних перетворень.

Уведення/виведення рядків символів

Мікропроцесор зберігає і обробляє дані, подані лише в двійковому вигляді (двійкові числа, BCD-числа, коди символів). Система команд мікропроцесора та апаратура комп'ютера дозволяють виконувати операції уведення/виведення інформації поданої *тільки у символьному вигляді* (окремі символи або рядки символів). Тому перед виконанням операції уведення/виведення необхідно самостійно передбачити в програмі потрібні перетворення даних. Розглянемо деякі засоби програмної організації операцій уведення/виведення.

Уведення/виведення інформації для текстового режиму роботи відеосистеми під керуванням ОС Windows здійснюється за допомогою консольного вікна. У консольному вікні відображається інформація, що вводиться з клавіатури, і повідомлення, що виводяться. Поняття консолі більш широке. Консоль – це інтерфейс, що забезпечує підтримку програм, які працюють у текстовому режимі. Консоль складається з буфера введення та одного або декількох екранних буферів. Буфер уведення являє собою чергу, кожний запис якої містить інформацію про події введення. Події введення – це натискання і відпускання клавіш на клавіатурі та миші, а також дії користувача безпосередньо з самим вікном (переміщення, мінімізація і т. д.). Екранний буфер – це двовимірний масив, що містить коди символів та їх колір – байт-атрибут.

Інтерфейс прикладного програмування (Application Programming Interface – Win API) містить достатній набір функцій, що забезпечують роботу консольного додатка. Розглянемо деякі з них, а також функції стандартної бібліотеки MASM.

Функція BOOL WINAPI AllocConsole (VOID)

Призначення: Створення консолі для додатка.

Вхідні параметри: Відсутні.

Вихідні значення: У разі успішного завершення повертає значення TRUE, інакше – FALSE.

Функція BOOL WINAPI FreeConsole (VOID)

Призначення: Звільнення консолі.

Вхідні параметри: Відсутні.

Вихідні значення: У разі успішного завершення повертає значення TRUE, інакше – FALSE.

Функція MASM StdIn proc lpszBuffer:DWORD, bLen:DWORD

Призначення: Уведення тексту з клавіатури та розміщення його в буфері. Закінчення введення з натисканням клавіші Enter.

Вхідні параметри: lpszBuffer – буфер, куди поміщається текст із клавіатури; bLen – розмір буфера.

Вихідні значення: Немає.

Функція MASM StdOut proc (lpszText: DWORD)

Призначення: Відображення ASCIIZ – рядка на екрані монітора з поточної позиції курсора.

Вхідні параметри: lpszText – адреса ASCIIZ-рядка, який необхідно відобразити.

Вихідні значення: Немає.

Функція MASM locate proc x:DWORD, y:DWORD

Призначення: Установлення курсора на екрані монітора за вказаними координатами.

Вхідні параметри: X – координата екрана монітора; Y – координата екрана монітора.

Вихідні значення: Немає.

Перетворення десяткових чисел в ASCII форматі на машинне зображення

З уведенням чисел у десятковому форматі, тобто у вигляді рядка із символів 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 з використанням функції StdIn у пам’яті формується рядок байтів з ASCII кодів десяткових цифр. ASCII код цифри

відрізняється від значення, що вона являє собою, на величину 30h. Символ “0” кодується як 30h, “1” – як 31h, ... , “9” – як 39h

Рядок із цифр “1234” з уведенням буде поданий у пам’яті у вигляді:

31h	32h	33h	34h	0Ah	0Dh
1	2	3	4	5	6

Байти 1 - 4 : ASCII коди введених символів,

Байт 5 : ASCII код символу керування – перехід на початок рядка (0Ah)

Байт 6 : ASCII код клавіші Enter (0Dh)

Зменшивши ASCII код кожної цифри на значення 30h, отримаємо четвірку байтів, що являтиме собою еквівалент десяткового числа 1234 у незапакованому BCD форматі, де кожний байт являтиме собою двійковий еквівалент відповідної десяткової цифри.

Можна скористатися відомими формулами розкладання для позиційних систем числення. Загалом у позиційній системі числення значення числа, що має зображення

$$A_p = \pm a_{n-1}a_{n-2}...a_1a_0, a_{-1}a_{-2}...a_{-m}, \qquad (1.1)$$

може бути подане у вигляді суми:

$$\left\|A_p\right\| = \pm\left(a_{n-1}\cdot p^{n-1} + a_{n-2}\cdot p^{n-2} + ... + a_0\cdot p^0 + a_{-1}\cdot p^{-1} + ... + a_{-m}\cdot p^{-m}\right), \qquad (1.2)$$

де n – загальна кількість розрядів у цілій, а m – у дробовій частині числа, a_k – цифра k-го розряду 0<=a_k<=p-1, p – основа системи числення, p^{n-i} (i=1..n) – вага *i*-го розряду цілої, а p^{-j} (j=1..m) – вага *j*-го розряду дробової частини числа.

За відсутності дробової частини числа

$$\left\|A_p\right\| = \pm\left(a_0\cdot p^0 + a_1\cdot p^1 + a_2\cdot p^2 + ... + a_{n-2}\cdot p^{n-2} + a_{n-1}\cdot p^{n-1}\right). \qquad (1.3)$$

Помноживши значення кожного байта починаючи з молодшого, на відповідний ваговий коефіцієнт (10^0, 10^1, 10^2, і. т.д.) і підраховуючи суму часткових добутків, отримаємо абсолютне значення числа в двійковій системі числення: 4 x 10^0 + 3 x 10^1 + 2 x 10^2 + 1 x 10^3 = 1234, або 04h x 01h + 03h x 0Ah + 02h x 64h + 01h x 03D8h = 04D2h. Таблиця 1.1 демонструє наведену схему.

Таблиця 1.1

Операція	Результат	
	Десятковий	Шістнадцятковий
4 x 1 =	0004d	0004h
3 x 10 =	0030d	001Eh
2 x 100 =	0200d	00C8h
1 x 1000 =	1000d	03E8h
Сума	1234d	04D2h

Неважко переконатися, що 1234d = 04D2h. Таким чином, процедура перетворення полягає в наступному:

1. Зарезервувати місце для зберігання вагового коефіцієнта кожної десяткової цифри числа. Початкове значення вагового коефіцієнта взяти таким що дорівнює 1.

2. Зарезервувати місце для зберігання суми часткових добутків. Початкове значення суми взяти таким що дорівнює 0.

3. Почати з наймолодшого байту числа в ASCII форматі й обробляти справа наліво.

4. Завантажити в регістр байт з ASCII кодом десяткової цифри.

5. Зменшити значення регістра на 30h (або видалити трійки з чотирьох лівих бітів регістра)

6. Виконати множення регістра на значення поточного вагового коефіцієнта й додати результат до суми часткових добутків.

7. Якщо оброблено всі байти числа, перейти до п. 9, інакше – до п. 8

8. Перерахувати ваговий коефіцієнт, помноживши його поточне значення на 10d, й перейти до п. 4.

9. Закінчити обробку.

Наведений вище алгоритм передбачає виконання двох операцій множення (для перерахунку вагового коефіцієнта та його множення на цифру числа) та

однієї операції додавання (до суми часткових добутків) на кожному кроці. Крім того потрібно попередньо визначати довжину символьного рядка.

Як видно з (1.3) формула розкладання для позиційних систем числення за відсутності дробової частини числа зводиться до обчислення значення многочлена, яке може бути виконано за схемою Горнера. Таблиця 1.2 демонструє цю схему.

Таблиця 1.2

Операція	Результат	
	Десятковий	Шістнадцятковий
1 + 10 x 0 =	0001d	0001h
2 + 10 x 1 =	0012d	000Ch
3 + 10 x 12 =	0123d	007Bh
4 + 10 x 123 =	1234d	04D2h

Таким чином, процедура перетворення полягає в наступному:

1. Зарезервувати місце для зберігання результату. Поточне початкове значення результату взяти таким що дорівнює 0.

2. Почати з наймолодшого байту числа в ASCII форматі й обробляти зліва направо.

3. Завантажити в регістр байт з ASCII кодом десяткової цифри.

4. Зменшити значення регістра на 30h (або видалити трійки з чотирьох лівих бітів регістра)

5. Виконати додавання до регістра поточного значення результату помноженого на десять.

6. Якщо оброблено всі байти числа, перейти до п. 7, інакше – до п. 3

7. Закінчити обробку

Наведений вище алгоритм передбачає виконання однієї операції множення та однієї операції додавання на кожному кроці. Програмну реалізацію алгоритму демонструє функція, наведена у прикладі 1.1.

Приклад 1.1. Функція uDecASCIIToBin виконує перетворення цілих без знакових десяткових чисел в ASCIIZ форматі в машинне зображення для 32-бітного режиму процесора.

Функція приймає у регістрі ESI адресу початку ASCIIZ-рядка і повертає у регістрі EAX машинне зображення числа.

```
uDecASCIIToBin proc   uses esi ebx
        xor ebx,ebx
        cld
@@:     lodsb
        or al,al
        jz @F
        and eax,0Fh
        lea ebx, [ebx+ebx*4]
        lea ebx, [eax+ebx*2]
        jmp @B
@@:     mov eax,ebx
        ret
uDecASCIIToBin endp
```

Для знакових чисел потрібно пропустити символ знаку, передати uDecASCIIToBin у регістрі ESI адресу першої цифри числа в ASCIIZ-рядку, для негативних чисел виконати перетворення результату у регістрі EAX в додатковий код.

Перетворення з машинного зображення в десяткові числа в ASCII форматі

Дана процедура включає в себе процес, зворотний розглянутому вище. Для того, щоб перетворити ціле число із системи числення з основою p_1 на систему числення з основою p_2, необхідно послідовно ділити його і частки від ділення на основу нової системи числення p_2, доки не отримаємо частку меншу, ніж p_1. Остання частка – старша цифра числа в новій системі числення. Інші цифри – це залишки від ділення, які потрібно записати у порядку, зворотному їх отриманню.

Як приклад розглянемо перетворення шістнадцяткового числа 04D2h, отриманого в попередньому розділі, знову на десяткове число в ASCII форматі.

Таблиця 1.3

Операція	Частка	Залишок
04D2h / 0Ah	07Bh	04h ←
07Bh / 0Ah	0Ch	03h ↑
0Ch / 0Ah	01h →	02h ↑

Оскільки після третього кроку отримуємо частку 01h, яка менша, ніж 0Ah, то процедура завершується. Результат виписується, починаючи від останньої частки до залишку від першого ділення (у напрямку стрілок).

Перед виведенням на екран потрібно отримати ASCII коди десяткових цифр, збільшивши значення відповідних їм байтів на 30h. Програмну реалізацію алгоритму демонструє функція, наведена у прикладі 1.2.

Приклад 1.2. Функція uBinToDecASCII для 32-бітного режиму процесора виконує перетворення машинного подання цілих без знакових чисел на еквівалентний ASCIIZ-рядок в десятковій системі числення.

Функція приймає у регістрі EAX число, у регістрі ESI адресу початку ASCIIZ-рядка.

```
uBinToDecASCII proc   uses esi edi ebx
        mov esi,edi
        mov ebx,10
@@:  xor edx,edx
        div ebx
        or dl,30h
        mov byte ptr [esi],dl
        inc esi
        or eax,eax
```

```
        jnz @B
        mov byte ptr [esi],0
@@:     dec esi
        mov al,byte ptr [esi]
        xchg byte ptr [edi],al
        mov byte ptr [esi],al
        inc edi
        cmp edi,esi
        jb @B
        ret
uBinToDecASCII endp
```

Для знакових чисел потрібно спочатку визначити знак числа перевіркою старшого біту регістра EAX і записати за результатами перевірки першим у буфер ASCIIZ-рядка символ «-» або «+». Для негативних чисел отримати в регістрі EAX прямий код числа і передати uBinToDecASCII в регістрі ESI адресу другого байта в буфері ASCIIZ-рядка. Для позитивних чисел просто передати uBinToDecASCII в регістрі ESI адресу другого байта в буфері ASCIIZ-рядка.

Перетворення шістнадцяткових чисел в ASCII на машинне зображення

У разі введення чисел у шістнадцятковому вигляді, тобто як рядка із символів 0, 1, 2, 3, 4, 5, 6, 7, 8,. 9, A, a, B, b, C, c, D, d, E, e, F, f у пам'яті формується рядок байтів з ASCII кодів цих символів. Як зазначалося раніше, ASCII коди десяткових цифр відрізняються від своїх двійкових еквівалентів на величину 30h.

Унаслідок аналізу таблиці ASCII кодів з'ясовується, що коди шістнадцяткових символів "a", "b", "c", "d", "e", "f" відрізняються від своїх десяткових еквівалентів (10d, 11d, 12d, 13d, 14d, 15d) на 57h, а коди символів "A", "B", "C", "D", "E", "F" – на 37h. Отже, процедура перетворення повинна розрізняти три типи ASCII кодів шістнадцяткових цифр і робити їх коригування на величину 30h, 37h або 57h. Визначення коефіцієнта коригування починають послідовно з 30h до 57h з урахуванням таких міркувань:

1. Якщо байт ASCII коду шістнадцяткової цифри зменшити на 30h і після цього його значення буде менше, ніж 0Ah, то це символ від “0” до “9”.

2. Якщо байт ASCII коду шістнадцяткової цифри зменшити на 37h і після цього його значення буде більше, ніж 09h, і менше, ніж 10h, то це символ від “A” до “F”.

3. Якщо байт ASCII коду шістнадцяткової цифри зменшити на 57h і після цього його значення буде більше, ніж 09h і менше, ніж 10h, то це символ від “a” до “f”.

Зменшивши ASCII код кожної цифри на значення потрібного коефіцієнта коригування, отримаємо послідовність байтів, у якій значення кожного байта являтиме собою двійковий еквівалент відповідної шістнадцяткової цифри.

Подальше використання формули розкладання можливе, але через особливості шістнадцяткової системи числення не доречне. Для отримання машинного подання шістнадцяткового числа потрібно виділити в отриманій послідовності байтів молодші тетради і поєднати їх. Наприклад, 00000000b, 00000100b, 00001101b, 00000010b => 0000 0100 1101b 0010b = 04D2h.

Загальний алгоритм перетворення такий.

1. Зарезервувати місце для зберігання результату. Початкове значення задати рівним 0.

2. Почати зі старшого байта з ASCII кодом шістнадцяткової цифри та обробляти зліва направо.

3. Завантажити в регістр байт з ASCII кодом шістнадцяткової цифри.

4. Зменшити значення в регістрі на величину 30h. Якщо після цього значення в регістрі виявиться меншим ніж 0Ah, перейти до п. 7.

5. Зменшити значення в регістрі ще на величину 07h (разом з попереднім пунктом уже на 30h + 07h = 37h). Якщо після цього значення в регістрі виявиться меншим ніж 10h, перейти до п. 7.

6. Зменшити значення в регістрі ще на величину 20h (разом з попередніми пунктами вже на 30h + 07h + 20h = 57h).

7. Записати молодшу тетраду регістру у молодшу тетраду результату.

8. Якщо оброблено всі символи ASCII-рядка, перейти до п. 10.

9. Виконати лінійне зрушення поточного значення результату на 4 біти вліво й перейти до п. 3.

10. Завершити обробку.

Програмну реалізацію алгоритму перетворення демонструє функція, наведена у прикладі 1.3.

Приклад 1.3. Функція HexASCIIToBin виконує перетворення шістнадцяткових чисел в ASCIIZ форматі на машинне зображення для 32-бітного режиму процесора.

Функція приймає в регістрі ESI адресу початку ASCIIZ-рядка і повертає в регістрі EAX машинне зображення числа.

```
HexASCIIToBin   proc   uses esi ebx
        xor ebx,ebx
        cld
hex:
        lodsb
        or al,al
        jz done
        shl ebx,4
        sub al,30h
        cmp al,9
        jbe @F
        sub al,7h
        cmp al,15
        jbe @F
        sub al,20h
@@:  or bl,al
        jmp hex
done:
        mov eax,ebx
```

```
        ret
HexASCIIToBin endp
```

Перетворення з машинного зображення в шістнадцяткові числа в ASCII

Перетворення двійкових чисел на шістнадцяткові в ASCII форматі полягає в декомпозиції байта, слова або подвійного слова, які містять значення двійкового числа, на окремі тетради й заміні кожної такої тетради ASCII кодом відповідної цифри. Така заміна виконується за такими правилами.

1. Якщо значення двійкового числа, поданого тетрадою, менше, ніж десять, то це шістнадцяткова цифра від “0” до “9” і для отримання байта її ASCII коду потрібно цю тетраду розширити зліва бітами 0011 (додати цю тетраду до байта 30h = 00110000b або занести 30h у старшу тетраду байта командою OR).

2. Якщо значення двійкового числа, поданого тетрадою, більше, ніж десять, то це шістнадцяткова цифра від “a” до “f”, або від “A” до “F” і для отримання байта її ASCII коду потрібно цю тетраду додати до байта 37h = 00110111b, або 57h = 01010111b.

Наприклад : 0000010011010010b → 0000 0100 1101 0010 b →

```
  00000000  00000100  00001101  00000010     – розширена тетрада
+ 00110000  00110000  00110111  00110000     – додаток
  00110000  00110100  01000100  00110010     – двійковий ASCII код
|   30h   |  34h    |  44h    |  32h    |    – шістнадцятковий ASCII код
   “0”        ”4”       ”D”       ”2”        – відповідний символ
```

Примітка. Перетворення двійкових чисел на шістнадцяткові в ASCII форматі можна також виконати за допомогою операцій послідовного ділення, як це було розглянуто раніше для перетворення двійкових чисел на десяткові в ASCII форматі. Ділити потрібно на шістнадцять і проводити коригування залишків на 30h або 37h залежно від їх значення, але використання тут арифметичних операцій звичайно недоцільне.

Програмну реалізацію алгоритму перетворення демонструє функція, наведена у прикладі 1.4.

Приклад 1.4. Функція BinToHexASCII для 32-бітного режиму процесора виконує перетворення машинного подання на еквівалентний ASCIIZ-рядок в шістнадцятковій системі числення.

Функція приймає в регістрі EBX число, у регістрі EDI – адресу початку ASCIIZ-рядка розміром не менше, ніж 11 байтів.

```
BinToHexASCII   proc   uses edi
        mov ecx,8
        cld
next:   rol ebx,4
        mov al,bl
        and al,0Fh
        cmp al,10
        jb @F
        add al,'A'-10-'0'
@@:     add al,'0'
        stosb
        loop next
        mov ax,'h'
        stosw
        ret
BinToHexASCII endp
```

Перетворення з машинного зображення на двійкові числа в ASCII форматі та зворотне перетворення

Це найпростіші з перетворень. Отримання ASCII кодів символів “0” та “1” можна виконувати, починаючи як з молодших, так зі старших цифр. Після аналізу значення старшого (молодшого) біта машинного зображення числа виконується формування відповідного байта ASCII коду. Перевірку значення біта зручно реалізовувати за допомогою команд зрушення, оскільки вони встановлюють значення прапора CF відповідним значенням біта, що зрушувався.

Перетворення з ASCII-рядка в машинне зображення можна виконувати на основі значення молодшого біту ASCII-коду символів. Для символу “0” (код 30h) молодший біт дорівнює 0, а для символу ”1” (код 31h) молодший біт дорівнює 1. Потрібно тільки правильно зберегти послідовність молодших бітів.

Програмну реалізацію алгоритмів перетворення демонструють функції, наведені у прикладах 1.5 і 1.6.

Приклад 1.5. Функція BinToBinASCII для 32-бітного режиму процесора виконує перетворення машинного зображення на еквівалентний ASCIIZ-рядок у двійковій системі числення.

Функція приймає в регістрі EBX число, у регістрі EDI адресу початку ASCIIZ-рядка розміром не менше, ніж 34 байти.

```
BinToBinASCII           proc   uses edi ebx
;Вхід: EBX – число, EDI – адреса буфера не менше, ніж 34 байти
        cld
        bsr ecx,ebx
        jnz @F
        mov ax,'0'
        stosw
        ret
@@:  inc ecx
        ror ebx,cl
@@:  rol ebx,1
        setc al
        or al,30h
        stosb
        loop @B
        mov ax,'b'
        stosw
        ret
BinToBinASCII endp
```

Приклад 1.6. Функція BinASCIIToBin виконує перетворення двійкових чисел в ASCIIZ форматі на машинне зображення для 32-бітного режиму процесора.

Функція приймає в регістрі ESI адресу початку ASCIIZ-рядка і повертає в регістрі EAX машинне зображення числа.

```
BinASCIIToBin          proc   uses esi ebx
        xor ebx,ebx
        cld
@@:     lodsb
        or al,al
        jz @F
        rcr al,1
        rcl ebx,1
        jmp @B
@@:     mov eax,ebx
        ret
BinASCIIToBin endp
```

Завдання до практичної роботи

Визначити номер свого варіанта завдання. Для визначення номера варіанта потрібно перевести номер, залікової книжки у двійкову систему числення, виділити чотири молодші біти і скористатися таблицею 1.4.

Розробити програму, яка виконує такі дії.

1. Виводить на екран повідомлення у вигляді

стисле повідомлення про призначення і параметри програми
прізвище, ім’я по батькові, номер групи

2. Розв’язує завдання згідно з номером свого варіанта.

Увага! Під час вводу чисел у вигляді ASCII-рядків програма повинна контролювати коректність вхідних даних (перевищення діапазону подання чисел, неприпустимі символи цифр).

Таблиця 1.4

Біти $b_3b_2b_1b_0$	Розробити програму, яка розв'язує такі завдання
0 0 0 0	Запитує перше число у десятковому вигляді, друге число – у шістнадцятковому вигляді (обидва – у діапазоні від 0 до 255), множить їх і виводить на екран результат у десятковому вигляді
0 0 0 1	Запитує 2 числа у двійковому вигляді в діапазоні від 0 до 255, множить їх і виводить на екран результат у шістнадцятковому вигляді
0 0 1 0	Запитує перше число у десятковому вигляді, друге число – у двійковому вигляді (обидва – у діапазоні від 0 до 65535), додає їх і виводить на екран результат у десятковому вигляді
0 0 1 1	Запитує 2 числа у десятковому вигляді в діапазоні від 0 до 255, множить їх і виводить на екран результат у двійковому вигляді
0 1 0 0	Запитує 2 числа у десятковому вигляді в діапазоні від 0 до 255, множить їх і виводить на екран результат у десятковому вигляді
0 1 0 1	Запитує 2 числа у десятковому вигляді в діапазоні від 0 до 65535, додає їх і виводить на екран результат у десятковому вигляді
0 1 1 0	Запитує 2 числа у десятковому вигляді в діапазоні від 0 до 255, множить їх і виводить на екран результат у шістнадцятковому вигляді
0 1 1 1	Запитує 2 числа у десятковому вигляді в діапазоні від 0 до 65535, додає їх і виводить на екран результат у шістнадцятковому вигляді
1 0 0 0	Запитує 2 числа у шістнадцятковому вигляді в діапазоні від 0h до 0ffh, множить їх і виводить на екран результат у десятковому вигляді
1 0 0 1	Запитує 2 числа у шістнадцятковому вигляді в діапазоні від 0h до 0ffffh, додає їх і виводить на екран результат у десятковому вигляді

Продовження Таблиці 1.4

1 0 1 0	Запитує 2 числа у шістнадцятковому вигляді в діапазоні від 0h до 0ffh, множить їх і виводить на екран результат у шістнадцятковому вигляді
1 0 1 1	Запитує 2 числа у шістнадцятковому вигляді в діапазоні від 0h до 0ffffh, додає їх і виводить на екран результат у шістнадцятковому вигляді
1 1 0 0	Запитує перше число у десятковому вигляді, друге число – у шістнадцятковому вигляді (обидва – у діапазоні від 0 до 255), множить їх і виводить на екран результат у десятковому вигляді
1 1 0 1	Запитує перше число у десятковому вигляді, друге число – у шістнадцятковому вигляді (обидва у діапазоні від 0 до 65535), додає їх і виводить на екран результат у десятковому вигляді
1 1 1 0	Запитує перше число у десятковому вигляді, друге число – у шістнадцятковому вигляді (обидва – у діапазоні від 0 до 255), множить їх і виводить на екран результат у шістнадцятковому вигляді
1 1 1 1	Запитує перше число у десятковому вигляді, друге число – у шістнадцятковому вигляді (обидва – у діапазоні від 0 до 65535), додає їх і виводить на екран результат у шістнадцятковому вигляді

Контрольні питання

1. Чому виникає потреба виконання розглянутих у практичній роботі перетворень?

2. На основі чого можна побудувати алгоритм перетворення із символьного на машинне зображення для позиційної системи числення з будь якою основою?

3. Навести алгоритм і програмну реалізацію перетворення із символьного на машинне зображення для позиційної системи числення основою 5.

4. Навести алгоритм і програмну реалізацію перетворення з машинного на символьне зображення для позиційної системи числення основою 8.

5. Навести алгоритми і програмні реалізації перетворень із символьного на машинне зображення і навпаки – пакованих і непакованих BCD-чисел.

Зміст звіту

Звіт з цієї практичної роботи має містити такі складові.

1. Назва і мета практичної роботи.

2. Повний вихідний текст усіх файлів проекту вирішення завдання практичної роботи з їх детальними коментарями.

3. Відповіді на контрольні питання.

Література: [1, 2, 3, 4, 5].

Практична робота № 2

Тема. Арифметичні операції з багаторозрядними цілими числами

Мета: вивчення і набуття практичних навичок програмної реалізації алгоритмів «довгої» арифметики над багаторозрядними цілими числами.

Додавання та віднімання багаторозрядних чисел

Використання арифметичних команд ADD, ADC, SUB, SBB, MUL, IMUL, DIV, IDIV, INC, DEC було розглянуто в методичних вказівках щодо виконання лабораторних робіт. Іншої «арифметики», окрім деяких специфічних засобів її підтримки в системі команд мікропроцесора (не вбудованого співпроцесора!), не передбачено.

Ураховуючи опис команд арифметики, можна зробити висновок, що найбільші досягнення реалізовані у мікропроцесорах Intel64/AMD64 – це можливість виконання чотирьох основних арифметичних операцій з цілими 32-розрядними числами в 32-бітному режимі або з цілими 64-розрядними числами в 64-бітному режимі. Використовуючи деякі спеціальні методи, на основі набору команд базової арифметики можна виконувати розрахунки будь-яких чисел з абсолютною точністю. Саме вивченню таких методів присвячено ця практична робота.

Послідовність операції додавання в двійковому вигляді з одиницею переносу, що плавно «пливе» від молодших до старших розрядів, дозволяє

додавати окремі частини багаторозрядних чисел, ураховуючи переноси між ними для корекції результату. Саме такий спосіб може бути використано для додавання багато розряднихцілих двійкових чисел у мікропроцесорах Intel64/AMD64, і саме для цього в системі команд мікропроцесора передбачено дві спеціальні арифметичні команди ADC (додавання з урахуванням переносу) та SBB (віднімання з урахуванням займу).

Програмну реалізацію алгоритму додавання багаторозрядних цілих двійкових чисел для 32-бітного режиму процесора демонструє функція, наведена у прикладі 2.1.

Приклад 2.1. Функція LongAdd передбачає, що доданки і сума подані у вигляді масиву подвійних слів починаючи з молодших «цифр».

```
.data
num1 dd      55667788h, 11223344h  ; 64-х бітне 1122334455667788h
num2 dd      55667788h, 11223344h  ; 64-х бітне 1122334455667788h
num3 dd 2 dup (0); для збереження результату
carry db 0           ; для врахування можливого переносу
```

Доданки мають однакове значення, а отже, додавання еквівалентно множенню на 2, що у свою чергу еквівалентно зрушенню на 1 біт вліво. Це дозволяє перевірити результат, проаналізувавши елементи масиву num3 і змінної carry. Інший варіант для тестування – нулі в старших «цифрах».

Функція приймає такі параметри:

N – розмір доданок у подвійних словах

item, item2, sum – адреси, відповідно, двох доданків і суми

```
LongAdd proc STDCALL uses esi edi ebx N:DWORD,\
                        item1:DWORD, item2:DWORD, sum:DWORD
    mov ecx,N
    xor esi,esi
    mov   ebx,item1
    mov   edx,item2
    mov   edi,sum
```

```
        clc
@@:  mov   eax,[ebx+esi*4]
        adc     eax,dword ptr [edx+esi*4]
        mov   dword ptr [edi+esi*4],eax
        inc esi
        loop   @B
        adc carry,0
        ret
LongAdd endp
```

Для виклику потрібно описати прототип функцій як

```
LongAdd PROTO :DWORD, :DWORD, :DWORD, :DWORD
```

Виклик виконується за допомогою

```
invoke LongAdd, 2, ADDR num1, ADDR num2, ADDR num3
```

Збільшення розрядності чисел, наприклад до 128 бітів, лише «подовжить» значення змінних і кількість ітерацій циклу у два рази, не вносячи нічого принципово нового.

Проаналізувавши приклад 2.1 неважко побачити, що перехід від додавання до віднімання здійснюють простою заміною команд ADD на SUB, а ADC на SBB. Під час віднімання не виникає переносу зі старшої «цифри» і непотрібно контролювати переповнення.

Множення багаторозрядних чисел

Операція множення багаторозрядних чисел виконується звичайним способом «у стовпчик». Це універсальний метод для будь-якої системи числення, хоча тут ми використаємо його, можливо, і в незвичному поки що вигляді.

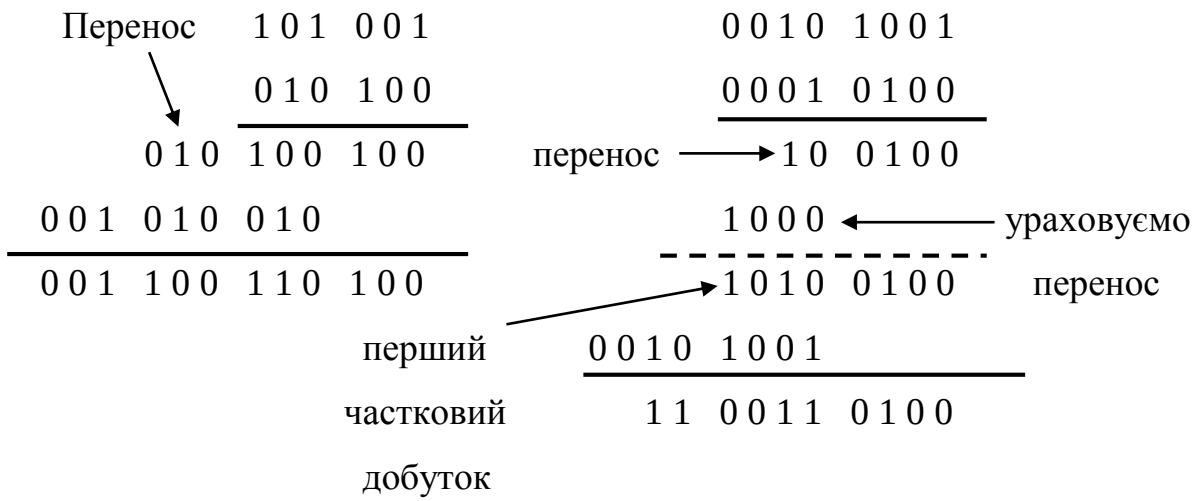
Ми звикли до того, що під час виконання множення «у стовпчик» десяткових, двійкових і навіть шістнадцяткових чисел усі розряди числа зображують однією цифрою. Тепер для зображення однієї «цифри» багаторозрядного числа використовуватимемо 8, 16 або 32 двійкові розряди.

Множення таких «цифр» здійснюватимемо за допомогою команди MUL. Молодші 8-й, 16-й, 32-й розряди результату множення дають відповідну «цифру» часткового добутку, а старші розряди – це та «цифра» переносу, яку потрібно додати до результату множення на наступну цифру множеного. Багаторозрядні часткові добутки додають з використанням розглянутого вище методу.

Приклад 2.2. Виконаємо множення в двійковому вигляді, уважаючи, що кожні два біти – це одна цифра числа.

```
          1 0  1 0  0 1
          0 1  0 1  0 0
        ---------------
          0 0  0 0  0 0
      1 0  1 0  0 1
    1 0  1 0  0 1
  -------------------
  1 1  0 0  1 1  0 1  0 0
```

Збільшимо «цифру» до трьох або до чотирьох бітів



Програмну реалізацію алгоритму множення багаторозрядних цілих двійкових чисел для 32-бітного режиму процесора демонструє функція, наведена у прикладі 2.3.

Приклад 2.3. Функція LongMul передбачає, що співмножники і добуток подано у вигляді масиву подвійних слів починаючи з молодших «цифр».

```
.data
num1 dd      12345678h,12345678h
```

```
num2 dd      12345678h,12345678h
num4 dd 4 dup (0); добуток повинен бути розміром N*2
```

Функція приймає такі параметри:

N – розмір у подвійних словах співмножників

multiplier1, multiplier2, product – адреси, відповідно, співмножників і добутку

```
LongMul     proc   STDCALL uses    esi edi ebx N:DWORD,\
                                   multiplier1:DWORD, multiplier2:DWORD,
                                   product:DWORD
        mov   ecx,N
        shl     ecx,1
        mov   edi,product
        xor     eax,eax
        cld
        rep stosd
        mov ecx,N
        xor     esi,esi
next: push ecx
        xor edi,edi
        mov   ecx,N
        clc
@@:  mov   ebx,multiplier2
        mov   eax,dword ptr [ebx+esi]
        mov   ebx,multiplier1
        mul    dword ptr [ebx+edi]
        mov   ebx,product
        lea     ebx,[ebx+esi]
        lea     ebx,[ebx+edi]
        add    dword ptr [ebx],eax
        adc    dword ptr [ebx+4],edx
        adc    dword ptr [ebx+8],0
```

```
        lea     edi,[edi+4]
        loop    @B
        lea     esi,[esi+4]
        pop ecx
        loop next
        ret
LongMul     endp
```

Для виклику потрібно описати прототип функцій як

```
LongMul PROTO :DWORD, :DWORD, :DWORD, :DWORD
```

Виклик виконується за допомогою

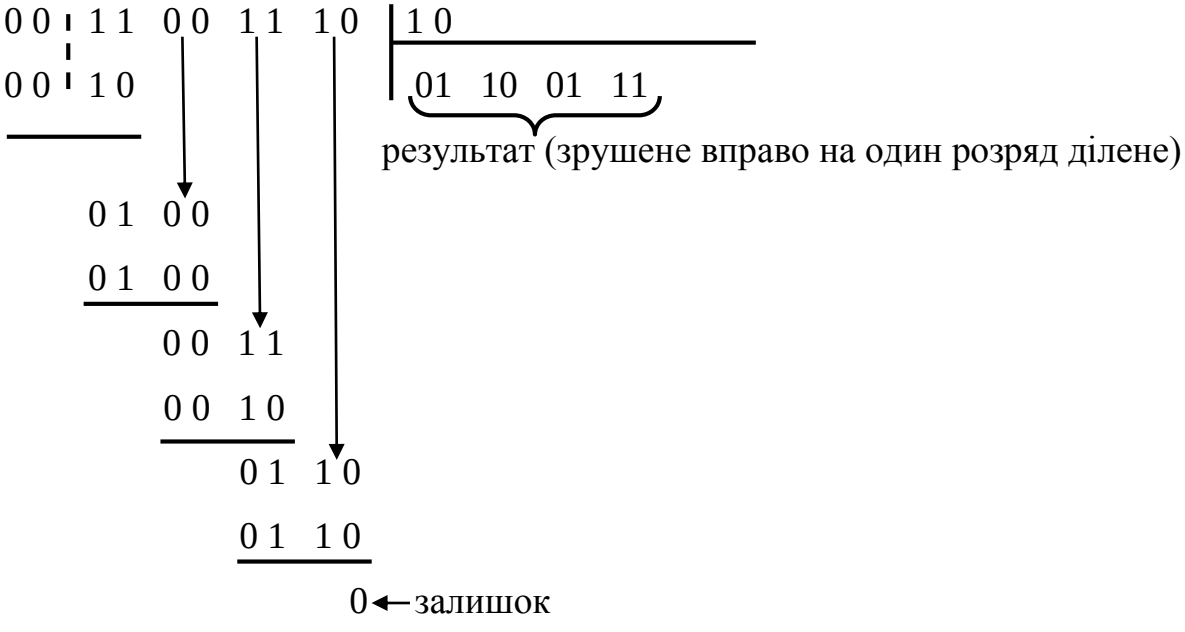
```
invoke LongMul, 2, ADDR num1, ADDR num2, ADDR num4
```

Аналогічно можна реалізувати множення чисел більшої розрядності. Звичайно це збільшує кількість повторень циклу, і до того ж, у нелінійному масштабі.

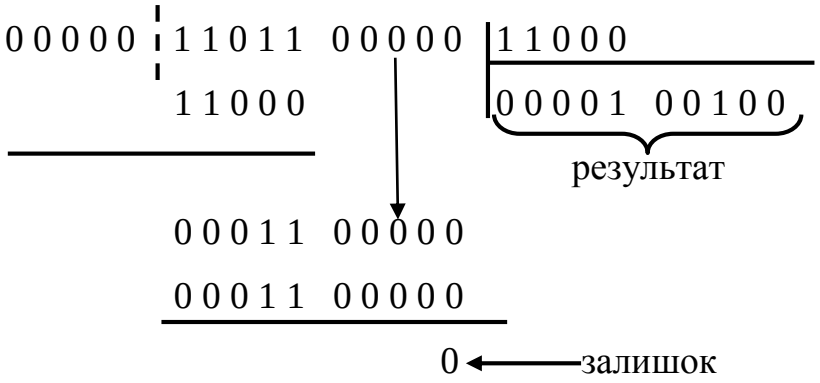
Ділення багаторозрядних чисел

За певних співвідношень діленого та дільника для виконання операції можна пристосувати і команду DIV, використовуючи багаторозрядні «цифри» аналогічно розглянутому вище методу множення багаторозрядних чисел.

Приклад 2.5. Поділимо 11001110b на 10b з використанням дворозрядних «цифр»



Збільшимо «цифру» до п’яти розрядів і поділимо 1101100000b на 11000b



Такий алгоритм, на жаль, позбавлений універсальності, адже існує одне принципове обмеження – неможливо виконати ділення за допомогою DIV більш ніж на 32-розрядні «цифри» в 32-бітному режимі або на 64-розрядні «цифри» в 64-режимі.

Але такий метод дозволяє уникнути іншого обмеження команди DIV – неможливість ділення 2n розрядного числа (n = 8, 16, 32, 64) на n розрядне, якщо результат операції буде більший ніж n-розрядне число. Можна поділити справжнє довге 64-розрядне число, що містить “1” навіть у найстаршому біті найстаршого байта, на байт, розширений аж до подвійного слова.

Програмну реалізацію алгоритму ділення багаторозрядних цілих двійкових чисел з використанням команди DIV для 32-бітного режиму процесора демонструє функція, наведена у прикладі 2.4.

Приклад 2.4. Функція LongDiv передбачає, що ділене і частку подано у вигляді масиву подвійних слів, починаючи зі старших «цифр».

```
.data
num1 dd      12345678h,12345678h,12345678h,12345678h
.data?
num2 dd 4 dup (?)
```

Функція приймає такі параметри:

N – розмір у подвійних словах діленого dividend;

dividend – адреса діленого;

divisor – 32-розрядний дільник;

result – адреса частки.

Функція повертає в регістрі EAX залишок від ділення

```
LongDiv proc STDCALL uses esi edi ebx ebp     N:DWORD, \
                                              dividend:DWORD, \
                                              divisor:DWORD, \
                                              result:DWORD
        mov edi,dividend
        mov ebx,result
        mov   ecx,N
        mov   ebp,divisor
        xor    esi,esi
        xor    edx,edx
@@:     mov   eax,dword ptr [edi+esi*4]
        div    ebp
        mov   dword ptr [ebx+esi*4],eax
        inc esi
        loop @B
        mov   eax,edx
        ret
      LongDiv       endp
```

Загалом операція ділення для чисел будь-якого розміру на числа будь-якого розміру реалізується найважче, бо вона виконується «у чистий стовпчик», без залучення команди DIV мікропроцесора.

Операцію ділення у стовпчик виконують за допомогою послідовних вирахувань дільника (зрушеного вліво на потрібну кількість розрядів) з діленого, збільшуючи відповідний розряд частки на 1 з кожним відніманням, поки не залишиться число, менше за дільник (залишок). Схему алгоритму демонструє приклад 2.5.

Приклад 2.5. Ділення 64-бітного числа в EDX:EAX на 64-бітне число в ECX:EBX. Частка поміщається в EDX:EAX, залишок – у ESI:EDI.

```
mov ebp,64  ;  Лічильник  бітів.
```

```
        xor esi,esi
        xor edi,edi  ;  Залишок  =  0.
bitloop:
        shl eax,1
        rcl edx,1
        rcl edi,1  ;  Зрушення  на  1  біт  вліво  128-бітного  числа
        rcl esi,1  ;  ESI:EDI:EDX:EAX.
        cmp esi,ecx  ;  Порівняти старші подвійні  слова.
        ja divide
        jb next
        cmp edi,ebx  ;  Порівняти  молодші  подвійні  слова.
        jb next
divide: sub  edi,ebx
        sbb esi,ecx
        inc  eax  ;  Установити молодший  біт в  EAX.
next:  dec  ebp
        jne  bitloop ;  Повторити  цикл  64  рази.
```

Для чисел зі знаком програмна емуляція аналогів команд мікропроцесора IDIV та IMUL для багаторозрядних чисел потребує лише певного доповнення вищерозглянутих алгоритмів.

1. Визначити знак (значення старшого біта) першого та другого числа, після чого отримати прямий код для негативних чисел і встановити в 0 знакові їх біти.

2. Визначити і запам’ятати знак результату за співвідношенням знаків операндів.

3. Виконати операцію ділення чи множення без знакових чисел згідно з вибраним алгоритмом.

4. Виконати перетворення результату на додатковий код відповідно до його знаку, отриманого в п. 2.

Завдання до практичного заняття

Визначити номер свого варіанта завдання. Для визначення номера варіанта потрібно перевести номер залікової книжки у двійкову систему числення, виділити п’ять молодших бітів і скористатися таблицею 2.1.

Таблиця 2.1

Номер варіанта $b_4b_3b_2b_1b_0$	Обчислити значення виразу, уважаючи, що X,Y,Z,U,V,W – це 64-розрядні числа
0 0 0 0 0	$\frac{X^2+X\cdot Y+W}{2\cdot Z}+\frac{X^2+X\cdot Y}{2\cdot Z+Y^2}$
0 0 0 0 1	$\frac{W^2+U}{X^2+Y^2+Z^2}+\frac{X^2+Y^2+Z^2}{X\cdot Y\cdot Z}$
0 0 0 1 0	$\frac{W^2+U}{X^2+Y^2+Z^2}+\frac{X+Y+Z}{V-U}$
0 0 0 1 1	$\frac{X^2+Y^2+Z^2}{X\cdot Y\cdot Z}+\frac{Y^2}{X^2}+\frac{X\ Y}{U}-\frac{V}{Y+Z}$
0 0 1 0 0	$\frac{U^2+V^2+XYZ}{(Z+Y)^2}+\frac{W^2+X\cdot Y+Z}{X^2+Y^2+Z^2}$
0 0 1 0 1	$\frac{W^2+U}{X^2+Y^2+Z^2}+\frac{X^2+Y^2+Z^2}{X\cdot Y\cdot Z}+\frac{Y^2}{X^2}$
0 0 1 1 0	$\frac{W^2+X\cdot Y+Z}{X^2+Y^2+Z^2}+\frac{X\cdot Y^2\cdot Z}{X^2+Y^3+Z^2}$
0 0 1 1 1	$\frac{W^2+X\cdot Y+\frac{U}{X}}{X\cdot Y\cdot Z}+\frac{W+U\ V}{2X^2+3U-8}$

Продовження таблиці 2.1

0 1 0 0 0	$\frac{W^2+X\cdot Y+\frac{U}{X^2}}{X^2\cdot Y\cdot Z}+\frac{Y^3}{Z^2}$
0 1 0 0 1	$\frac{W^2+X\cdot(Y+Z^2)+U}{X^2\cdot Y\cdot Z}+\frac{(X+Y)^2}{2U}+\frac{Y^3}{Z^2}$
0 1 0 1 0	$\frac{U^2+(V+Y^2)X+2}{4}$
0 1 0 1 1	$\frac{X^2+Y^2+Z^2}{U+V}+\frac{Y^2X^2}{2}-\frac{W+UV+4}{XYZ}$
0 1 1 0 0	$\frac{X^2+XY+Y^2}{U+V}+\frac{XYZ}{Z+U}+\frac{W+UV}{2X^2+3U-8}$
0 1 1 0 1	$\frac{UV}{X^2+Y^2+Z^2}+\frac{U^2+(V+Y^2)X+2}{4}-\frac{V-U}{Y^2}$
0 1 1 1 0	$\frac{UV}{X+XY+XYZ}+\frac{U^2+V^2}{2UV+1}-\frac{W+(X+Y+Z)^3}{XYZV}$
0 1 1 1 1	$\frac{(X+Y)^3}{W+X^2}+\frac{W+XYZ}{U^2+V^2}-\frac{X^2+Y^2+Z^2}{12}$
1 0 0 0 0	$\frac{(X+Y)^2}{2U}+\frac{Y^3}{Z^2}-\frac{W}{U^2+V^2}$
1 0 0 0 1	$\frac{X^3+Y^2+Z}{U+X+Z}+\frac{UV}{XYZ}-\frac{W}{V(X+4)}$
1 0 0 1 0	$\frac{X^2Y^2}{X+Y+Z}-\frac{W+ZU}{2U+Y+Z}+\frac{U^2+V^2}{(Z+Y)^2}$

Продовження таблиці 2.1

1 0 0 1 1	$\frac{W-UV}{XYZ}+\frac{U^2+V^2}{W-U}-\frac{W}{U^2+ZU}$
1 0 1 0 0	$\frac{(X^2+Y)^3}{W+UV}+\frac{X\ Y\ Z}{U^2}-\frac{W}{3U^2}$
1 0 1 0 1	$\frac{U^2+X\ Y\ Z+V^2}{U\ (X+Z)}+\frac{4X^2Y^2}{U}-\frac{W-UV}{U^2}$
1 0 1 1 0	$\frac{X^3+3X\ Y\ Z^2+2}{2U\ +V}-\frac{XYZ}{4}+\frac{W-U(X+V)}{U}$
1 0 1 1 1	$\frac{X^3+Y^2+Z}{U+X+Z}+\frac{U\ V}{XYZ^2}+\frac{W}{V\ (XY+4)}$
1 1 0 0 0	$\frac{X^2Y^2}{X+Y+Z}+\frac{W\ +ZU}{U+Y+Z^2}+\frac{U^2+V^2+XYZ}{(Z+Y)^2}$
1 1 0 0 1	$\frac{U\ V}{X\ +X\ Y\ +X\ Y\ Z}+\frac{W}{U^2+ZU}$
1 1 0 1 0	$\frac{X^3+X\ Y\ Z^2+2}{X\ +V+W}+\frac{XYZ}{X+V}+\frac{W-U(X+V)}{U}$
1 1 0 1 1	$\frac{W-UV}{XYZ}+\frac{U^2+V^2}{W-U}+\frac{U^2+V^2+XYZ}{(Z+Y)^2}$
1 1 1 0 0	$\frac{X\ Y}{X\ +X\ Y\ +X\ Y\ Z}+\frac{U^2+V^2}{XY+1}+\frac{W+(X+Y)^3}{XYZV}$
1 1 1 0 1	$\frac{X^3+Y^2+Z}{U+X+Z}+\frac{U\ V}{XYZ^2}+\frac{XYZ}{X+V}$
1 1 1 1 0	$\frac{(X+Y)^2}{U+V}+\frac{Y^3}{Z^2}+\frac{W^2}{U^2+V^2}$

Продовження таблиці 2.1

1 1 1 1 1	$\frac{X^3+X\ Y\ Z^2+2}{X\ +V+W}+\frac{W-U(X+Y)^2}{U^2}$

Контрольні питання

1. Навести алгоритми множення залежно від способу формування суми часткових добутків для

а) множення, починаючи з молодших розрядів множника, зі зрушенням суми часткових добутків вправо за непорушного множеного;

б) множення, починаючи з молодших розрядів множника зі зрушенням множеного вліво і непорушній сумі часткових добутків;

в) множення, починаючи зі старших розрядів множника зі зрушенням суми часткових добутків вліво за непорушного множеного;

г) множення, починаючи зі старших розрядів множника зі зрушенням управо множеного та непорушній сумі часткових добутків.

2. Навести алгоритми ділення без відновлення залишку для:

а) ділення з непорушним діленим і дільником, що зрушується вправо;

б) ділення з непорушним дільником і діленим, що зрушується вліво.

Зміст звіту

Звіт з цієї практичної роботи має містити такі складові.

1. Назва і мета практичної роботи.

2. Повний вихідний текст усіх файлів проекту вирішення завдання практичної роботи з їх детальними коментарями.

3. Відповіді на контрольні питання.

Література: [2, 4, 5].

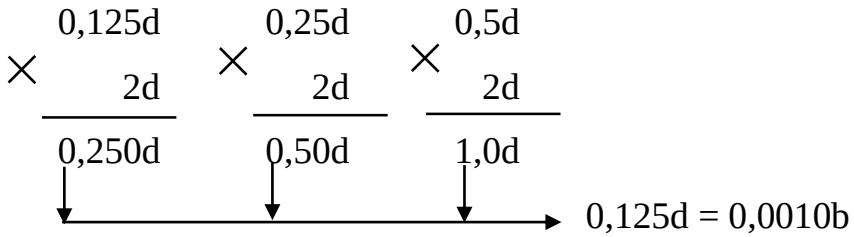
Практична робота № 3

Тема. Операції з дійсними числами з фіксованою комою

Мета: вивчення і набуття практичних навичок програмної реалізації алгоритмів операцій з дійсними числами з фіксованою комою.

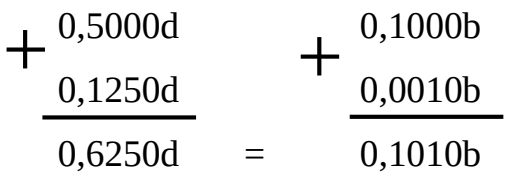
Під час розгляду базових арифметичних команд ADD/ADC, SUB/SBB, MUL/IMUL, DIV/IDIV, INC/DEC дотепер уважалося, що операндами таких операцій є цілі числа зі знаком або без нього. Насправді це не зовсім так, або навіть зовсім не так. Уся раніше розглянута арифметика без перешкод узагальнюється для чисел, поданих у форматі з фіксованою комою. Якщо кома фіксується після молодшого розряду числа, то це ціле число (дробова частина дорівнює 0, і для неї не виділяють розрядів), якщо ж кома фіксується перед старшим розрядом числа, то це правильний дріб (ціла частина дорівнює 0, і для неї не виділяється розрядів). Знак числа в обох випадках кодується в самому лівому розряді.

Приклад 3.1. Розглянемо базові арифметичні команди на прикладі 4-розрядних двійкових чисел. Виконаємо перетворення чисел 0.125d, 0.25d 0.5d на двійкову систему за допомогою алгоритму множення.



У процесі перетворення попутно було отримано і двійкові значення двох інших чисел: 0,25d = 0,0100b та 0,5d = 0,1000b (незначущі нулі зліва записано для вирівнювання на межу 4-х бітів)

Зафіксувавши кому перед старшим розрядом числа виконаємо додавання та множення:



```
  0,25d          0,0100b
×                ×
  0,50d          0,1000b
 -------        --------
  0 0 0          0 0000
   1 2 5         00000
 -------        00000
0,1 2 5         00100
                --------
                0,00100000b = 0,125d
```

Зафіксувавши кому після молодшого розряду, отримуємо:

```
  1000,0b = 8d          0100,0b = 4d
+               ×
  0010,0b = 2d          1000,0b = 8d
 ------------          ------------
  1010,0b = 10d          0000 0
                        00000
                       00000
                      00000
                     01000
                    ----------
                    00100000,00b = 32d
```

«Забувши» про існування коми, адже про неї не здогадується і мікропроцесор, бачимо, що числові розряди результатів операцій однакові, і виникає питання, як їх інтерпретувати. Під час множення чисел за модулем, менших ніж одиниця, результат не може перевищити одиницю, збільшення довжини результату в два рази означає збільшення кількості цифр дробової частини. Переповнення під час додавання чисел з фіксованою перед старшим розрядом комою означає, що результат уже буде більший ніж 1. Така ситуація фіксується мікропроцесором установленням прапорця CF і потребує обробки.

Існує певний клас задач, у яких потрібно виконувати обчислення дійсних чисел, до того ж, бажано робити це дуже швидко, а висока точність результатів непотрібна. Використання для таких розрахунків команд арифметичного співпроцесора недоцільне, або може бути й зовсім неприйнятне.

Ураховуючи те, що в таких задачах діапазон можливих значень чиселзазвичай відомий, для спрощення розрахунків використовують формат з

фіксованою комою, за якого розряди відводяться і для цілої, і для дробової частин числа. Оскільки такий формат явно не підтримується системою команд процесора, то потрібно створити для нього власну обробку.

Для цього відводять певну кількість байтів для розміщення цілої частини числа й певну кількість байтів для розміщення його дробової частини. Операції виконують окремо з обома частинами таких чисел, але при цьому враховують можливий вплив результатів оброблення однієї частини на іншу.

Найзручнішими форматами чисел з фіксованою комою щодо подальшої реалізації арифметичних операцій є формати, що узгоджуються з розрядністю регістрів процесора – 8:8, 16:16 та 32:32. Додавання і віднімання для чисел з фіксованою комою нічим не відрізняється від додавання і віднімання цілих чисел:

```
mov     ax,1080h
mov     bx,1240h
add     ax,bx
sub     ax,bx
```

Тепер, якщо вважати, що операції були виконані з числами з фіксованою комою в форматі 8:8, тобто AH:AL – ціла:дробова частина першого числа, BH:BL – ціла:дробова частина другого числа, то результати можна інтерпретувати так:

```
  AX = 1080h = 16,5d
+ BX = 1240h = 18,25d
-------------------
  AX = 22C0h = 34,75d
- BX = 1240h = 18,25d
-------------------
  AX = 1080h = 16,5d
```

Під час множення потрібно пам’ятати, що воно призводить до збільшення довжини результату у два рази. Наприклад, нехай EAX і EBX містять числа з фіксованою комою у форматі 16:16, тоді після виконання команди mul ebx EDX:EAX міститиме 64-бітний результат, де EDX містить усю цілу частину, а EAX – усю дробову), а після команди shrd eax,edx,16 EAX міститиме кінцеву

відповідь, якщо не відбулося переповнювання (тобто якщо результат не перевищив 65 535).

Реалізація ділення потребує дещо більших зусиль. Число, записане з фіксованою комою у форматі 16:16, можна подати як число, помножене на 2^{16}. Якщо розділити такі числа одне на одне, то отримаємо: $(A * 2^{16})/(B * 2^{16}) = A/B$. Щоб результат мав потрібний вигляд $(A/B) * 2^{16}$, потрібно наперед помножити ділене на 2^{16}

```
xor     edx,edx
ror     eax,16
xchg    ax,dx       ; EDX:EAX = EAX * 2^16
div     ebx         ; EAX = результат
```

Саме завдяки використанню аналогічних методів перші комп'ютери через відсутність арифметичного співпроцесора могли виконувати операції над числами з плаваючою комою. Компілятори мов програмування високого рівня виконували емуляцію арифметики з плаваючою комою, замінюючи команди арифметичного співпроцесора еквівалентними послідовностями команд арифметики з фіксованою комою.

Завдання до практичного заняття

Визначити номер свого варіанта завдання. Для визначення номера варіанта потрібно перевести номер, залікової книжки у двійкову систему числення, виділити п'ять молодших бітів і скористатися таблицею 3.1.

Таблиця 3.1

Номер варіанта $b_4b_3b_2b_1b_0$	Обчислити значення виразу, уважаючи, що X, Y, Z, U, V, W – це числа з фіксованою комою у форматі 16:16
0 0 0 0 0	$\frac{X^2+X\cdot Y+W}{2\cdot Z}+\frac{X^2+X\cdot Y}{2\cdot Z+Y^2}$
0 0 0 0 1	$\frac{W^2+U}{X^2+Y^2+Z^2}+\frac{X^2+Y^2+Z^2}{X\cdot Y\cdot Z}$

Продовження таблиці 3.1

0 0 0 1 0	$\frac{W^2+U}{X^2+Y^2+Z^2}+\frac{X+Y+Z}{V-U}$
0 0 0 1 1	$\frac{X^2+Y^2+Z^2}{X\cdot Y\cdot Z}+\frac{Y^2}{X^2}+\frac{XY}{U}-\frac{V}{Y+Z}$
0 0 1 0 0	$\frac{U^2+V^2+XYZ}{(Z+Y)^2}+\frac{W^2+X\cdot Y+Z}{X^2+Y^2+Z^2}$
0 0 1 0 1	$\frac{W^2+U}{X^2+Y^2+Z^2}+\frac{X^2+Y^2+Z^2}{X\cdot Y\cdot Z}+\frac{Y^2}{X^2}$
0 0 1 1 0	$\frac{W^2+X\cdot Y+Z}{X^2+Y^2+Z^2}+\frac{X\cdot Y^2\cdot Z}{X^2+Y^3+Z^2}$
0 0 1 1 1	$\frac{W^2+X\cdot Y+\frac{U}{X}}{X\cdot Y\cdot Z}+\frac{W+UV}{2X^2+3U-8}$
0 1 0 0 0	$\frac{W^2+X\cdot Y+\frac{U}{X^2}}{X^2\cdot Y\cdot Z}+\frac{Y^3}{Z^2}$
0 1 0 0 1	$\frac{W^2+X\cdot(Y+Z^2)+U}{X^2\cdot Y\cdot Z}+\frac{(X+Y)^2}{2U}+\frac{Y^3}{Z^2}$
0 1 0 1 0	$\frac{U^2+(V+Y^2)X+2}{4}$
0 1 0 1 1	$\frac{X^2+Y^2+Z^2}{U+V}+\frac{Y^2X^2}{2}-\frac{W+UV+4}{XYZ}$
0 1 1 0 0	$\frac{X^2+XY+Y^2}{U+V}+\frac{XYZ}{Z+U}+\frac{W+UV}{2X^2+3U-8}$

Продовження таблиці 3.1

0 1 1 0 1	$\frac{U\ V}{X^2+Y^2+Z^2}+\frac{U^2+(V+Y^2)X+2}{4}-\frac{V-U}{Y^2}$
0 1 1 1 0	$\frac{U\ V}{X+X\ Y+X\ Y\ Z}+\frac{U^2+V^2}{2UV+1}-\frac{W+(X+Y+Z)^3}{XYZV}$
0 1 1 1 1	$\frac{(X+Y)^3}{W+X^2}+\frac{W+X\ Y\ Z}{U^2+V^2}-\frac{X^2+Y^2+Z^2}{12}$
1 0 0 0 0	$\frac{(X+Y)^2}{2U}+\frac{Y^3}{Z^2}-\frac{W}{U^2+V^2}$
1 0 0 0 1	$\frac{X^3+Y^2+Z}{U+X+Z}+\frac{U\ V}{XYZ}-\frac{W}{V\ (X+4)}$
1 0 0 1 0	$\frac{X^2Y^2}{X+Y+Z}-\frac{W+ZU}{2U+Y+Z}+\frac{U^2+V^2}{(Z+Y)^2}$
1 0 0 1 1	$\frac{W-UV}{XYZ}+\frac{U^2+V^2}{W-U}-\frac{W}{U^2+ZU}$
1 0 1 0 0	$\frac{(X^2+Y)^3}{W+UV}+\frac{X\ Y\ Z}{U^2}-\frac{W}{3U^2}$
1 0 1 0 1	$\frac{U^2+X\ Y\ Z+V^2}{U\ (X+Z)}+\frac{4X^2Y^2}{U}-\frac{W-UV}{U^2}$
1 0 1 1 0	$\frac{X^3+3X\ Y\ Z^2+2}{2U+V}-\frac{XYZ}{4}+\frac{W-U(X+V)}{U}$
1 0 1 1 1	$\frac{X^3+Y^2+Z}{U+X+Z}+\frac{U\ V}{XYZ^2}+\frac{W}{V\ (XY+4)}$

Продовження таблиці 3.1

1 1 0 0 0	$\frac{X^2Y^2}{X+Y+Z}+\frac{W+ZU}{U+Y+Z^2}+\frac{U^2+V^2+XYZ}{(Z+Y)^2}$
1 1 0 0 1	$\frac{U\ V}{X+X\ Y+X\ Y\ Z}+\frac{W}{U^2+ZU}$
1 1 0 1 0	$\frac{X^3+X\ Y\ Z^2+2}{X+V+W}+\frac{XYZ}{X+V}+\frac{W-U(X+V)}{U}$
1 1 0 1 1	$\frac{W-UV}{XYZ}+\frac{U^2+V^2}{W-U}+\frac{U^2+V^2+XYZ}{(Z+Y)^2}$
1 1 1 0 0	$\frac{X\ Y}{X+X\ Y+X\ Y\ Z}+\frac{U^2+V^2}{XY+1}+\frac{W+(X+Y)^3}{XYZV}$
1 1 1 0 1	$\frac{X^3+Y^2+Z}{U+X+Z}+\frac{U\ V}{XYZ^2}+\frac{XYZ}{X+V}$
1 1 1 1 0	$\frac{(X+Y)^2}{U+V}+\frac{Y^3}{Z^2}+\frac{W^2}{U^2+V^2}$
1 1 1 1 1	$\frac{X^3+X\ Y\ Z^2+2}{X+V+W}+\frac{W-U(X+Y)^2}{U^2}$

Контрольні питання

1. У чому полягає обмеженість основної системи команд процесора для обчислювальних задач?

2. Якими способами можна реалізувати обчислення елементарних функцій з використанням основної системи команд процесора?

Зміст звіту

Звіт з цієї практичної роботи має містити такі складові.

1. Назва і мета практичної роботи.

2. Повний вихідний текст усіх файлів проекту вирішення завдання практичної роботи з їх детальними коментарями.

3. Відповіді на контрольні питання.

Література: [2, 4, 5].

2 КРИТЕРІЇ ОЦІНЮВАННЯ ЯКОСТІ ВИКОНАННЯ ЗАВДАНЬ ПРАКТИЧНИХ ЗАНЯТЬ СТУДЕНТАМИ

Оцінювання роботи студентів з виконання практичних робіт з навчальної дисципліни здійснюється на підставі оцінювання кожної практичної роботи з переліку цих методичних вказівок. Максимальна кількість балів за кожну практичну роботу дорівнює 6 балів. Кожна практична робота оцінюється за такими складовими:

1. Звіт з виконання практичної роботи – 1 бал.

2. Захист практичної роботи – 5 балів.

Підсумкова оцінка виконання практичних робіт з навчальної дисципліни розраховується з цілої частини від попередньо обчисленого числа S*M/K/6+0,5, де S – сума оцінок усіх складових усіх практичних робіт, K – загальна кількість практичних робіт, M – максимальна кількість балів на цей вид контролю, що передбачена в робочій навчальній програмі дисципліни.

СПИСОК ЛІТЕРАТУРИ

1. Голубь Н. Г. Искусство программирования на Ассемблере. Лекции и упражнения. 2-е изд., испр. и доп. С.Пб.: ДиаСофтЮП, 2002. 656 с.
2. Зубков С. В. Assembler для DOS, Windows и UNIX. 3-е изд., стер. М.: ДМК Пресс; С.Пб.: Питер, 2004. 608 с.
3. Калашников О. А. Ассемблер? Это просто. Учимся программировать. С.Пб.: БХВ-Петербург, 2006. 384 с.
4. Кип Р. Ирвин. Язык ассемблера для процессоров Intel. 4-е изд., пер. с англ. М.: Издательский дом «Вильямс», 2005. 912 с.
5. Юров В. И. Assembler. Практикум. 3-е изд. С.Пб.: Питер, 2006. 399 с.

Методичні вказівки щодо практичних робіт з навчальної дисципліни «Системне програмування» для студентів денної та заочної форм навчання зі спеціальності 123 – «Комп'ютерна інженерія».

Укладач старш. викл. Ю. В. Зілінський

Відповідальний за випуск в.о. зав. кафедри КІС к. т. н. В. М. Сидоренко

Підп. до др. _____Формат 60х84 1/16. Папір тип. Друк ризографія.
Ум. друк. арк. ___. Наклад _____ прим. Зам. № _____. Безкоштовно.

Редакційно-видавничий відділ
Кременчуцького національного університету
імені Михайла Остроградського
вул. Першотравнева, 20, м. Кременчук, 39600