МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання курсової роботи
з дисципліни «Системне програмування»
для здобувачів освітньо-кваліфікаційного рівня «бакалавр»
напряму підготовки 6.050102 – Комп'ютерна інженерія
галузі знань 12 - Інформаційні технології
усіх форм навчання

Черкаси
2020

УДК 656(075.8)
ББК 39.18я73

Затверджено вченою радою ФІТІС, протокол № 8 від 29.04.2020 р., згідно з рішенням кафедри інформаційної безпеки та комп'ютерної інженерії, протокол № 8 від 22.04.2020 р.

Упорядники: Куницька С.Ю., к.т.н., доцент,
Шувалова Л.А., к.т.н., доцент,

Рецензент Голуб С.В., д.т.н., професор

Д 15 Методичні рекомендації до виконання курсової роботи з дисципліни «Системне програмування» для здобувачів освітньо-кваліфікаційного рівня «бакалавр» напряму підготовки 6.050102 – Комп'ютерна інженерія галузі знань 12 - Інформаційні технології усіх форм навчання [Електронний ресурс] / [упоряд. С.Ю. Куницька, Л.А. Шувалова] ; М-во освіти і науки України, Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2020. – 52 с. – Назва з титульного екрана.

УДК 656(075.8)

Виробничо-практичне
електронне видання
комбінованого використовування

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання курсової роботи
з дисципліни «Системне програмування»
для здобувачів освітньо-кваліфікаційного рівня «бакалавр»
напряму підготовки 6.050102 – Комп'ютерна інженерія
галузі знань 12 - Інформаційні технології
усіх форм навчання

Упорядники: **Куницька** Світлана Юріївна, **Шувалова** Людмила Аркадіївна

В авторській редакції.

ЗМІСТ

ВСТУП

Методичні рекомендації розроблені для здобувачів освітньо-кваліфікаційного рівня бакалавр напряму підготовки 6.050102 – Комп'ютерна інженерія галузі знань 12 - Інформаційні технології усіх форм навчання, призначені для формування навиків роботи мовою програмування ASSEMBLER завдяки вивченню: головних етапів та синтаксису програми, команд та регістрів загального призначення, директив та операндів програми, що в цілому відповідають загальним вимогам по написанню лістингу програми.

Метою навчальної дисципліни «Системне програмування» та розроблених методичних рекомендацій є чітка концепція, щодо вирішення окремих основних задач: вивчення структури програми, вміння використовувати команди різних видів: пересилка даних, логічні та циклічні, команди умовного та безумовного переходу, але головне значення набувають знання щодо використання арифметичних команд зі всіма особливостями відповідно кожної арифметичної дії. Вміння застосовувати прості та складні типи даних, знати особливості застосування регістрів загального призначення та володіти практичними знаннями щодо моделювання складних структур мовою Ассемблер.

Вивчення дисципліни підтверджується написанням курсової роботи, яка є самостійною роботою студента згідно отриманого індивідуального варіанта завдання.

Метою курсової роботи є розробка програми щодо виконання арифметичних операцій додавання, віднімання, множення або ділення двійково-десяткових чисел з фіксованою крапкою і заданою кількістю байтів та певним способом зберігання числа в пам'яті.

Основні напрями:

- вивчити 4 етапи створення програм на мові ASSEMBLER;
- ознайомитись із типами даних та їх цільовим призначенням;
- засвоїти головні команди і директиви мови ASSEMBLER;
- вивчити, як заповнювати об'єм пам'яті та розрядну сітку для запису певної кількості байт відповідно формату зберігання числа: упакованому або неупакованому;
- навчитися правильно застосовувати арифметичні команди щодо конкретної арифметичної операції;
- навчитися корегувати результати арифметичних операцій завдяки командам корекції.

В результаті розробленої курсової роботи студент повинен уміти:

- створювати робочу програму шляхом проходження всіх головних етапів: написання текстового файлу, асемблювання, компоновка та відладка програми;

- розробляти програмні модулі для перетворення двійково-десяткового числа мовою ASSEMBLER;
- розробляти програмні модулі для перемикання обчислювальних процесів, розподілу ресурсів обчислювальної системи з використанням внутрішньої інформаційної бази даних операційної системи з командами та регістрами;
- обґрунтовувати оптимальне рішення поставленої задачі, відображати алгоритм лістингу та заповнення розрядної сітки за певним способом зберігання числа в пам’яті згідно індивідуального завдання;
- застосовувати арифметичні команди, щодо конкретної арифметичної операції та застосовувати команди корекції для корегування арифметичного результату.

1 ВИМОГИ ДО ОФОРМЛЕННЯ ВИКОНАННЯ ТА ЗАХИСТУ КУРСОВОЇ РОБОТИ

Курсова робота, що виконується згідно вимог описаних в методичних рекомендаціях, призначена для вивчення головних етапів та синтаксису програми, арифметичних команд та регістрів загального призначення, директив та операндів програми, особливостей по заповненню відповідного форма числа, що в цілому відповідають загальним вимогам по написанню речень в лістингу програми.

При виконанні курсової роботи студент повинен опрацювати розділ з теоретичними відомостями методичних рекомендацій, вивчити лекційний матеріал щодо введення певної кількості байтів відповідно упакованого або неупакованого формату двійково-десяткового числа, знати команди пересилки даних, вивчити директиви та операнди, знати логічні та арифметичні команди, а також команди корекції арифметичного результату та проаналізувати індивідуальне отримане завдання.

Після виконання курсової роботи студент оформлює звіт згідно ДСТУ 3008-95 *(«Документація. Звіти у сфері науки і техніки. Структура і правила оформлення)* на аркушах формату А4 в текстовому редакторі MSWord і захищає курсову роботу у викладача при наявності електронної версії роботи, що містить всі робочі файли з розширеннями .txt, .asm, .obj, .exe та MAP-файл, де розташована інформація про помилки, що виникають в процесі створення лістингу курсової роботи.

Загальні вимоги до оформлення

1. Поля аркуша: праворуч 10 мм, ліворуч не менше 25 мм, зверху та знизу по 20 мм;
2. Шрифт курсової роботи – Times New Roman;
3. Кегль текстового редактора – 14;
4. Міжрядковий інтервал – 1,5;
5. Абзацний відступ – 1,25 см;
6. Структурні елементи «ЗМІСТ», «ВСТУП», «ВИСНОВКИ», «ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ», «СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ» не нумерують, а їхні назви є заголовками структурних елементів, що розташовують посередині без крапки в кінці;
7. Заголовки підрозділів повинні починатися з абзаційного відступу 14 шрифтом з нумерацією в межах розділу і друкуються маленькими літерами починаючи з великої. Пункти і підпункти також можуть мати заголовки. Номер підрозділу складається з номера розділу і порядкового номера підрозділу, відокремлених крапкою. Після номера підрозділу крапку не ставлять, наприклад 1.1, 1.2 і т. д.;
8. Розділ документа починається з нового аркуша. Якщо заголовок складається з двох і більше речень, їх розділяють крапкою. Перенесення слів у

заголовку розділів не допускається. Відстань між заголовком, приміткою, прикладом і подальшим або попереднім текстом має бути не менше ніж два міжрядкових інтервали;

9. Розділи курсової роботи повинні мати порядкову нумерацію в межах викладення суті роботи і позначатися арабськими цифрами без крапки, наприклад, 1, 2, 3 і т. д.

10. Сторінки курсової роботи, слід нумерувати арабськими цифрами, додержуючись наскрізної нумерації впродовж усього тексту. Титульний лист включають до загальної нумерації сторінок і номер сторінки на титульному листі не проставляють;

10. Ілюстрації слід розміщувати у курсовій роботі безпосередньо після тексту, де вони згадуються вперше, або на наступній сторінці. Ілюстрації й таблиці, розмішені на окремих сторінках, включають до загальної нумерації сторінок курсової роботи Усі графічні матеріали роботи повинні мати однаковий підпис «Рисунок». Назву рисунка друкують з великої літери та розміщують під ним посередині рядка, наприклад, «Рисунок 2.1 - Арифметична операція віднімання»;

11. Цифрові дані роботи треба оформлювати як таблицю відповідно до форми, поданої нижче. Таблицю слід розташовувати безпосередньо після тексту, у якому вона згадується вперше, або на наступній сторінці.

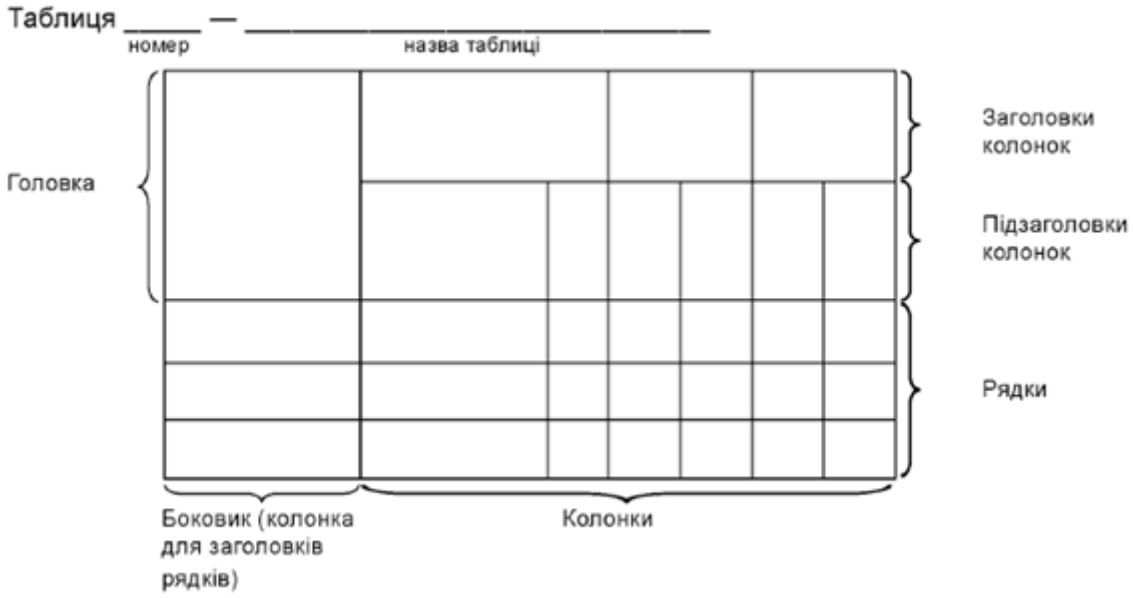


Рисунок 1 – Приклад створення таблиці

Таблиці слід нумерувати арабськими цифрами порядковою нумерацією в межах розділу, за винятком таблиць, що наводяться у додатках. Таблиці кожного додатка нумерують окремо. Номер таблиці додатка складається з позначення додатка та порядкового номера таблиці в додатку, відокремлених крапкою. Наприклад, «Таблиця В.1 — Назва», тобто це перша таблиця Додатка В. Таблиця може мати назву, яку друкують малими літерами (крім першої великої) і вміщують над таблицею з абзацного відступу. Назва має бути

стислою і відбивати зміст таблиці.

Зміст курсової роботи:

- *титульний лист* з темою роботи (Додаток А);
- *форма завдання на курсову роботу* (Додаток Б), що друкується з обох сторін на одному аркуші;
- *зміст (лист 2)* з наступними розділами та відповідними сторінками: вступ, теоретичні відомості, блок-схема головної програми та арифметичних операцій, хід виконання роботи, лістинг програми, результати виконання програми, висновки, список використаних джерел;
- *вступ (лист 3)* включає в себе: назву теми, мету та індивідуальне завдання згідно варіанту;
- *теоретичні відомості (лист 4)* обсягом описання 2-3 сторінки;
- *блок-схеми (лист N);*
- *лістинг програми (лист N);*
- *результати виконання програми (лист N);*
- *висновки (лист N)*;
- *список використаних джерел (лист N)*.

Форма завдання на курсову роботу є стандартною формою № У 6.01, яку заповнює студент в друкованому вигляді згідно прописаних полів (Додаток Б):

- тема курсової роботи – повна назва теми із зазначенням індивідуального варіанту №;
- термін здачі студентом курсової роботи – заповнюється тоді, коли студент захистив роботу у викладача;
- вихідні дані до роботи:

- *Індивідуальне завдання, яке складається з:*

1. Побудова інтерфейсу користувача;

2. Розв'язання поставлених задач згідно варіанту №.

- зміст розрахунково-пояснювальної записки – перелік всіх розділів змісту: вступ, теоретичні відомості, блок-схеми, лістинг програми, хід виконання роботи або результати виконання програми, висновки, список використаних джерел;
- дата видачі завдання – фіксується та дата, коли викладач видає завдання до курсової роботи;
- календарний план – містить таблицю, де необхідно за стовбцями та рядками заповнити всі етапи створення курсової роботи та зафіксувати відповідні дати. Примітка «виконано» вписується після виконання певного етапу та переходу до наступного, а на останньому етапі ставиться дата захисту курсової роботи (таблиця 1).

Таблиця 1 - Приклад заповнення календарного плану

№ п/п	Назва етапів курсової роботи (проекту)	Термін виконання етапів роботи (проекту)	Примітка
1	Отримання теми та індивідуального завдання до курсової роботи	12.03.2020	виконано
2	Розробка загальної структури лістингу програми	20.03.2020	виконано
3	Розробка блок-схем до індивідуального завдання	15.04.2020	виконано
4	Написання лістингу програми	30.04.2020	виконано
5	Відладка програми	01.05.2020	виконано
6	Попередній захист електронної версії курсової роботи	13.05.2020	виконано
7	Оформлення форми завдання № У 6.01	15.05.2020	виконано
8	Оформлення звіту курсової роботи	20.05.2020	виконано
9	Захист курсової роботи	01.06.2020	виконано

Зауваження! Форма завдання на курсову роботу № У 6.01 друкується на одному листі з двох сторін.

Розділ *«Вступ»* курсової роботи повинен мати:

- *тему курсової роботи*: «Модель арифметико-логічного пристрою»;
- *мету:* навчитися складати програму на мові ASSEMBLER, яка використовує арифметичні і логічні операції;
- *завдання:* скласти програму для моделювання операції додавання, віднімання і множення чисел з фіксованою крапкою з заданою кількістю розрядів згідно отриманого варіанту №__ (таблиця 2). Введення операндів виконувати з клавіатури. Результати виводити на екран. Програма повинна мати "дружній" інтерфейс, який складається з пунктів наступного меню:

1. Відомості про студента (П.І.П, № групи, курс, № залікової книжки, тема роботи, варіант завдання);
2. Введення чисел в пам’ять: введіть число А та введіть число В;
3. Робити перевірку правильності введення чисел згідно формату числа;
3. Інформація про переповнення розрядної сітки;
4. Виведення результатів арифметичних операцій згідно індивідуального

завдання;

5. Вихід з програми.

В розділі *«Теоретичні відомості»* студент детально описує процес створення програмного лістингу, тобто як розробляв загальну структуру програми, «дружній інтерфейс», блок-схеми програми, які використовував команди та директиви, операнди та регістри. Поступово описує наступні ключові питання:

- як в програмі заповнюється певна кількість байтів для введення двійково-десяткового числа згідно формату за варіантами;
- як заповнюється розрядна сітка по заповненню певної кількості байтів двійково-десяткового числа по формату упакований або не упакований за індивідуальним варіантом;
- коли відбувається переповнення розрядної сітки;
- які арифметичні команди використані для відповідних арифметичних операцій;
- як відбувається в програмному коді корекція результату, коли це потрібно і завдяки яким командам. Описати методику застосування корекції результату.

В розділі *«Блок-схеми до програми та арифметичних операцій»* курсової роботи потрібно створити блок-схему головної програми та окрему блок-схему на кожну арифметичну операцію (Додаток В). Підписувати необхідно посередині рисунка знизу за вимогами: кожна блок-схема має свою назву та наскрізну нумерацію. Наприклад: *Рисунок 2 – Блок-схема арифметичної команди додавання.*

В розділі «*Лістинг програми»* друкується структурований програмний код згідно отриманого завдання та створений за всіма правилами написання речень на мові ASSEMBLER. Текст програми – 14 шрифтом з коментарями до головних арифметичних операцій або інших важливих модулів програми.

В розділі *«Результати виконання програми»* студент зображує отримані результати роботи у вигляді друкованих робочих вікон з відповідною арифметичною дією або згідно меню інтерфейсу по кожному етапу розробки програми (Додаток Г). Підписувати та нумерувати результати необхідно єдиною нумерацією наскрізь в межах всієї курсової роботи.

У *«Висновках»* студент описує як формував структуру програми, які команди використовував, які директиви та регістри застосовував, як відбувався запис двійково-десяткового числа у певному форматі, як редагував помилки в середині програми, чому при занесенні іншого числа відбувається переповнення розрядної сітки, як працює застосування команд арифметичної корекції результату, як налаштовував створену програму на конкретні робочі

умови і які отримав результати згідно індивідуального варіанту курсової роботи.

Таблиця 2 - Варіанти завдань

Номер варіанта	***Кількість байтів у цілої і дробовій частинах числа***	***Спосіб зберігання числа в пам'яті***	***Арифметична операція***
1	3	упакований	додавання віднімання
2	*4*	*не упакований*	*віднімання* *множення*
3	5	упакований	додавання множення
4	*6*	*не упакований*	*додавання* *віднімання*
5	7	упакований	віднімання множення
6	*8*	*не упакований*	*додавання* *множення*
7	8	упакований	додавання віднімання
8	*3*	*не упакований*	*віднімання* *множення*
9	4	упакований	множення ділення
10	*5*	*не упакований*	*додавання* *віднімання*
11	6	упакований	додавання ділення
12	*7*	*не упакований*	*віднімання* *ділення*
13	2	упакований	множення ділення
14	*2*	*не упакований*	*віднімання* *множення*
15	9	упакований	додавання віднімання

Створення курсової роботи студент повинен планувати таким чином, щоб закінчити її в термін, визначений навчальним планом та отримати оцінку з урахуванням певних складових: терміну та якості виконання і захисту роботи, оформлення звіту, складності написання програмного лістингу, знань при захисті роботи за наступними критеріями оцінок:

- Оцінка «5» (*за шкалою ECTS – A*, тобто 90-100 балів) виставляється студенту, який володіє фактичним матеріалом теми, розділу, курсу. Відповідь повна, побудована послідовно, логічно і підтверджується прикладами або програмами.

- Оцінка «4» (*за шкалою ECTS – B*, тобто 85-89 балів, а також *за шкалою ECTS – C*, тобто 75-84 бали) виставляється студенту, який добре знає фактичний матеріал теми, розділу, курсу, але відповідь не достатньо повна, потребує додаткових питань викладача.
- Оцінка «3» (*за шкалою ECTS –D*, тобто 67-74 бали, а також *за шкалою ECTS – E*, тобто 60-66 бали) виставляється студенту, який володіє лише основними поняттями теми, розділу, курсу, знає основні алгоритми. Відповідає лише на деякі додаткові питання викладача.
- Оцінка «2» (*за шкалою ECTS – FX*, тобто 34-59 – це можливість повторного захисту роботи після доопрацювання, а також *за шкалою ECTS –F*, тобто 1-34 бали – з обов'язковим повторним курсом навчання) виставляється студенту, який не володіє теоретичним матеріалом теми, розділу, курсу, не знає відповідних алгоритмів, не вміє їх застосовувати.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Операційна система DOS

Напевно, десь в запорошених кутах ще можна розшукати комп'ютери IBM PC XT. Багато з них досі справні, тільки навряд чи кому прийде в голову включати їх, адже сучасні операційні системи (такі як Windows або Unix) не можна на них запустити навіть в принципі.

А ще зовсім нещодавно, в кінці 80-х років, ці машини коштували скажених грошей і викликали хвилювання у кожного справжнього програміста. На той момент в світі персональних комп'ютерів панувала операційна система DOS (теж фірми Microsoft), яка керувалася командним рядком, приблизно таким, як в оболонці FAR. Сама ця оболонка теж прийшла до нас з тих часів. Хоч FAR і консольний додаток Windows і не здатен працювати в системі DOS, але він досить точно копіює інтерфейс оболонки Norton Commander, що стояла в ті роки на кожному комп'ютері.

На відміну від Windows, DOS – однозадачна операційна система, що не здатна одночасно виконувати кілька програм. Це означає, що в DOS неможливий звичний для Windows буфер обміну. Адже буфер – це не просто ділянка пам'яті, а програма, яка змінює формат даних, які їй надсилаються і працює одночасно з іншими програмами. Тому доводиться спочатку зберігати дані на диску, потім виходити з однієї програми, завантажувати іншу і читати збережені дані.

Обмін даними в DOS спрощують так звані резидентні програми, які «спливають» при натисканні певної комбінації клавіш, «здирають» з екрану картинку або текст, зберігають їх у файлі і передають управління попередній програмі, яка вже може ці файли прочитати.

Незважаючи на всі ці незручності, DOS володіла і володіє важливою перевагою: вона дає повний контроль над комп'ютером, дозволяє робити з ним і всіма його пристроями все, що завгодно. Програміст (особливо якщо це умілий програміст на асемблері) відчуває, що може по максимуму витиснути з наявного «заліза» все можливе і навіть написати програму, здатну знищити DOS, а слідом за нею і себе саму.

Чудові часи, коли програміст міг володіти цілим комп'ютером, пройшли. Сучасні операційні системи дуже багато беруть на себе: вони не дозволяють вже програмам безпосередньо звертатися до пристроїв комп'ютера, тому що програм кілька, а пристрій – один. Тепер програміста відокремлює від «заліза» товстий шар вати - так званий API (наприклад, вже знайомий нам Windows API).

Але є ще галузі (і чималі), де DOS може стати в нагоді: це різні саморобні прилади, засновані на процесорах Intel. Такі прилади зазвичай збираються по частинах, як конструктор: в спеціальну «корзину» вставляється материнська плата, а в неї - процесор, пам'ять і необхідні плати. Прилад зазвичай керується єдиною програмою, яка повинна взаємодіяти з нестандартними пристроями, тому її простіше написати і зручніше виконувати в системі DOS. Є ще одна

причина, по якій потрібно бути знайомим з пристроями програми для DOS: в світі залишилося дуже багато вихідних текстів на асемблері для цієї операційної системи. І щоб не піддатися паніці, побачивши незрозумілі значки на кшталт int 21h, треба бути обізнаним з DOS ближче. Програмуванню на асемблері для DOS присвячено безліч книг. Тому поговоримо про найголовніше. Тому що, говорячи про DOS, ми дізнаємося багато нового про інструкції процесора і пристрої Windows.

А почнемо з програми для DOS, що виводить на екран вже знайому фразу Не можу мовчати! (лістинг 1).

Лістинг 1 - **Не можу мовчати! (DOS-версія)**

```
.8086
.MODEL small
.stack 100
.data
hello BYTE "Не можу мовчати!",Odh, Oah,`$`
.code
start:
mov dx,@stack          :реєстр стека
mov ss,dx              ;реєстр даних hello
mov dx,@data           :вивести на екран завершити програму
mov ds,dx
mov dx,Offset
mov ah,09
int 21h
mov ah,4ch
int 21h
end start
```

Незважаючи на багато нововведень, вам повинно бути в загальних рисах зрозуміло, що і як вона робить. Так, наприклад, рядки:

```
mov ah, 09h
int 21h
```

виводять на екран монітора слова «**Не можу мовчати !**», а рядки

```
mov ah,4c00h
int 21h
```

завершують програму, виконуючи роль процедури ExitProcess в Windows API.

Програма, показана в лістингу 1 хоч і призначена для системи DOS, спокійно може бути виконана і в Windows. В оболонці FAR вона запускається так само, як і консольний додаток Windows, але якщо дослідити детальніше її запуск і виконання, то виявиться, що Windows чинить з нею зовсім не так, як з «рідним» консольним додатком. Windows емулює виконання DOS-програм, тобто намагається своїми засобами виконати програму так, щоб ніхто не помітив підміни. Часто це вдається.

Як же Windows розпізнає програми для DOS? Так само, як і для Windows – по самій програмі, вірніше, по її заголовку. Адже програма, яка зберігається в файлі з розширенням .exe, містить не тільки інструкції процесора, але і відомості, які потрібні операційній системі для її запуску.

Отже, програмі, призначеній для системи DOS, потрібен особливий заголовок, не такий як у консольного додатку Windows. Такий заголовок створюється спеціальним компонувальником, який в нашій навчальній версії асемблера називається Link.exe, як одного з етапів створення та проходження програми на мові ASSEMBLER.

2.2 Етапи створення програми

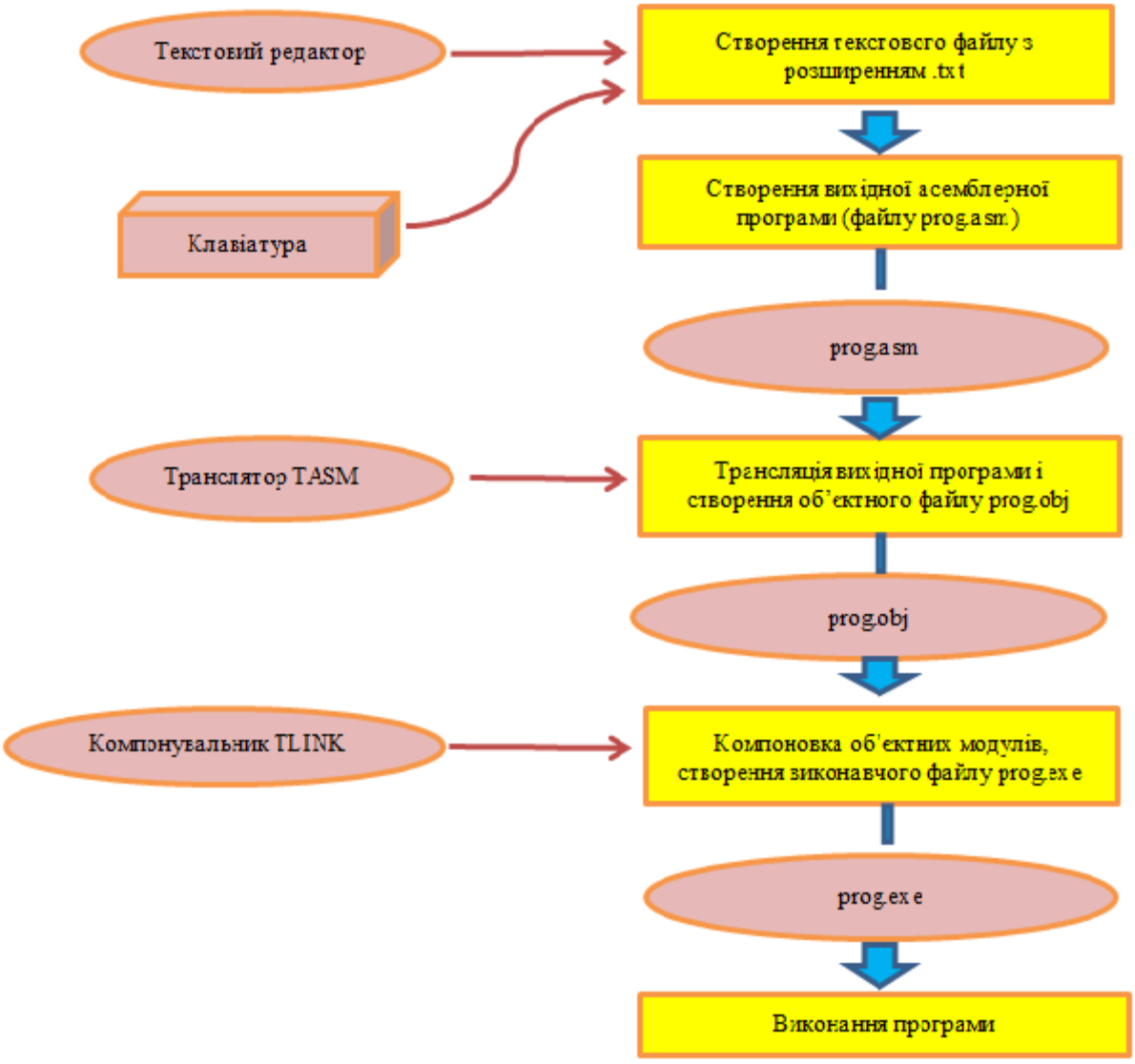


Рисунок 1 – Етапи створення програми

Процес створення програми на мові Асемблер можна розбити на такі структурні етапи:

1. *Написання тексту програми* з розширенням .txt.

Надалі потрібно використати етап асемблювання, тобто змінити попередній файл з розширенням .txt на файл з цією ж назвою, але розширенням .asm;

2. *Створення об'єктного модуля*.

Об'єктний модуль отримується в результаті трансляції програми. Трансляція здійснюється за допомогою наступної команди:

TASM ім'я_файла.asm [ключі] (розширення вказувати не обов'язково).

Якщо даний етап пройшов успішно, то ми отримуємо наступні файли: ім'я_файла.obj, ім'я_файла.lst, ім'я_файла.cfg (перший – обов'язково, інші – в залежності від ключів та директив в тексті програми).

Цей етап передбачає виправлення помилок лістингу в процесі його створення. Відповідно до завантажувальної версії операційної системи та TASM.EXE можливий наступний розвиток:

- можливе створення MAP-файлу, де за розміщенням кожного рядку з помилками прописується тип помилки, тобто синтаксична або логічна;
- безпосередньо помилки прописуються в командному рядку також з посиланням на місцезнаходження та тип помилки.

Виправлення помилок передбачає роботу з файлом .asm, збереженням нового файлу .asm та подальшого переходу на наступний етап.

3. *Створення завантажувального модуля* (компонування програми). Компонування програми здійснюється наступною командою:

TLINK ім'я_файла.obj [ключі] (розширення вказувати не обов'язково)

В результаті роботи даного етапу ми отримуємо виконуємий файл (.exe або .com). Який саме виконуємий файл отримується залежить від структури самої програми та ключів компоновки.

4. *Відладка програми*. Здійснюється за допомогою програми TASM.

Призначення файлів, що отримуються при створенні програми:

- ім'я_файла.asm – початковий асемблерний текст програми;
- ім'я_файла.obj – машинний об'єктний код програми;
- ім'я_файла.lst – лістінг асемблювання програми;
- ім'я_файла.crf – лістінг перехресних посилань;
- ім'я_файла.exe або ім'я_файла.com – виконавчий файл.

Обов'язковим етапом процесу розробки програми є її відладка. Специфіка програми на асемблері полягає в тому, що вона інтенсивно працює з апаратними ресурсами комп'ютера, тому необхідно постійно відслідковувати вміст певних регістрів та ділянок пам'яті. Для тестування програм на асемблері використовують спеціальні відладчики, наприклад, Turbo Debugger (td.exe), що дозволяє виконувати трасування програми і перегляд стану регістрів і ділянок пам'яті. Недоліком td.exe є те, що він не дозволяє вносити виправлення у

вихідний текст програми. Запуск програми-відладчика здійснюється за командою:

td.exe < ім'я виконавчого файлу >

2.3 Формати числових даних

Цілі двійкові числа

Ціле двійкове число з фіксованою крапкою – це число, закодоване в двійковій системі числення. Розмірність цілого двійкового числа може становити 8, 16 або 32 біта. Знак двійкового числа визначається тим, як інтерпретується старший біт в представленні числа. Це 7-й, 15-й або 31-й біти для чисел відповідної розмірності. Серед арифметичних команд є всього дві команди, які дійсно враховують цей старший розряд як знаковий – це команди цілочисельного множення і ділення imul і idiv.

В інших випадках відповідальність за дії зі знаковими числами і, відповідно, зі знаковим розрядом лягає на програміста. Діапазон значень двійкового числа залежить від його розміру і трактування старшого біта або як старшого значущого біта числа, або як біта знака числа.

Числа з фіксованою точкою в програмі описуються з використанням директив db, dw і dd. Діапазон значень двійкових чисел представлений в таблиці 3.

Таблиця 3 – Діапазон значень двійкових чисел

Розмірність поля	***Ціле число без знака***	***Ціле число зі знаком***
Байт	0...255	–128...+ 127
Слово	0...65 535	–32 768...+32 767
Подвійне слово	0...4 294 967 295	–2 147 483 648... +2 147 483 647

Таблиця 4 – Представлення чисел в двійковому коді

Цифра	Двійковий код	Цифра	Двійковий код
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	0101

Числа, які вводяться з клавіатури, надходять у процесор в ASCII - кодах. ASCII - код кожної десяткової цифри числа дорівнює значенню цієї цифри, збільшеному на 30h.

В свою чергу, процесор дозволяє виконувати арифметичні операції додавання, віднімання, множення і ділення над цілими числами, які можуть зберігатись у пам'яті в одному з трьох форматів: *двійковий формат, двійково-десятковий упакований формат і двійково-десятковий неупакований формат.*

Число в двійковому форматі зберігається в пам'яті у вигляді одно- чи двобайтового числа в додатковому коді. Число в двійково-десятковому упакованому форматі зберігається в пам'яті у вигляді послідовності байтів.

Послідовність цифр у байтах звичайна: в молодших байтах зберігаються старші цифри. В кожному байті зберігаються дві двійково-кодовані десяткові цифри в коді BCD (Binary-Coded Decimal).

Десяткові числа

Десяткові числа – спеціальний вид подання числової інформації, в основу якого покладено принцип кодування кожної десяткової цифри числа групою з чотирьох біт. При цьому кожен байт числа містить одну або дві десяткові цифри в так званому двійково-десятковому коді (BCD - Binary Coded Decimal). Мікропроцесор зберігає BCD-числа в двох форматах – упакований формат – кожен байт містить дві десяткові цифри. Десяткова цифра є двійковим значенням в діапазоні від 0 до 9 розміром 4 біта. При цьому код старшої цифри 7 числа займає старші 4 біти. Отже, діапазон представлення десяткового упакованого числа в одному байті складає від 00 до 99; неупакований формат – кожен байт містить одну десяткову цифру в чотирьох молодших бітах. Старші чотири біта мають нульове значення. Це так звана зона. Отже, діапазон представлення десяткового неупакованого числа в одному байті складає від 0 до 9.

Наведемо приклад запису 6-розрядного десяткового числа 653201 в пам'яті в BCD-коді:

Таблиця 5 – Запис числа

1	байт	2	байт	3	байт	
6	5	3	2	0	1	- десяткове представлення
0110	0101	0011	0010	0000	0001	- двійкове представлення

Тобто упакований формат містить значення кожної десяткової цифри в одному полубайті. Інакше виглядає запис неупакованого формату представлення числа. Число в двійково-десятковому неупакованому форматі зберігається у вигляді послідовності байтів. У кожному байті зберігається одна двійково-десяткова цифра. Її значення міститься в чотирьох молодших бітах байта.

Перед виконанням операції множення таких чисел у старших чотирьох бітах повинні бути записані нулі. Для операції додавання і віднімання вміст цих бітів не є суттєвим.

Приклад запису 3-розрядного десяткового числа 653 у пам`яті в двійково-десятковому неупакованому форматі:

Таблиця 6 – Запис числа в неупакованому форматі

1	байт	2	байт	3	байт	

0	6	0	5	0	3	- десяткове представлення
0000	0110	0000	0101	0000	0011	- двійкове представлення

Представлення двійково-десяткового числа в коді програми

Для опису двійковій-десяткових чисел в програмі можна використовувати тільки дві директиви опису та ініціалізації даних – db і dt (довжиною в 10 байт). Можливість застосування тільки цих директив для опису BCD-чисел обумовлена тим, що до таких чисел можна застосувати принцип «молодший байт за молодшою адресою», що дуже зручно для їх обробки. При використанні BCD-чисел спосіб опису цих чисел в програмі і алгоритм їх обробки – це «справа смаку» програміста.

Лістинг 2 – ***Опис BCD-чисел***

```
masm
model small
stack 256
.data                ;сегмент даних
per_1 db 2,3,4,6,8,2    ;неупаковане BCD-число 286432
                    ;в пам'яті 02 03 04 06 08 02
per_3 dt 9875645        ;упаковане BCD-число 9875645
                    ;в пам'яті 45 56 87 09
.code                ;сегмент кода
main:                ;точка входу в програму
mov ax,@data            ;зв'язуємо регістр dx з сегментом
mov ds,ax            ;даних через регістр ax
mov ax,4c00h            ;стандартний вихід
int 21h
end main            ;кінець програми
```

2.4 Блок-схеми алгоритмів

Розв'язування завдань складається з декількох етапів, частина яких відбувається без участі ЕОМ:

1. Постановка завдання, розробка математичної моделі;
2. Вибір розв'язання;
3. Розробка алгоритму і структури даних;
4. Реалізація алгоритму на алгоритмічній мові;
5. Підготовка завдання для ПК;
6. Відладка програми;
7. Обчислення, обробка і оформлення результатів.

На основі сформульованого завдання вибираються вхідні та вихідні змінні, записуються обмеження і зв'язок між ними.

Для поставленого арифметичного завдання вибирають метод рішення, який би звів його до послідовності арифметичних і логічних операцій. При виборі методу враховуються вимоги постановки завдання і можливості його реалізації.

Алгоритм – послідовності арифметичних і логічних дій над числовими значеннями змінних, що приводить до обчислення результату при зміні вхідних даних в достатньо широких межах. Таким чином, математичне формулювання завдання перетвориться в послідовність арифметичних і логічних дій і зв'язків між ними.

Найбільш точним способом представлення алгоритму є ***блок-схеми***. При цьому алгоритм представляється послідовністю блоків, що виконують певні функції і зв'язків між ними. Усередині блоків указується інформація функцій, які вони виконують. Блоки однієї схеми мають крізну нумерацію.

У схемі алгоритму кожному типу дій (введення, вивід, обчислення виразів, перевірка умов, перевірка певних дій і так далі) відповідає певна геометрична фігура, яка представляється у вигляді блоку (символ дії); вони з'єднуються лініями переходу, які визначають послідовність виконання дій:

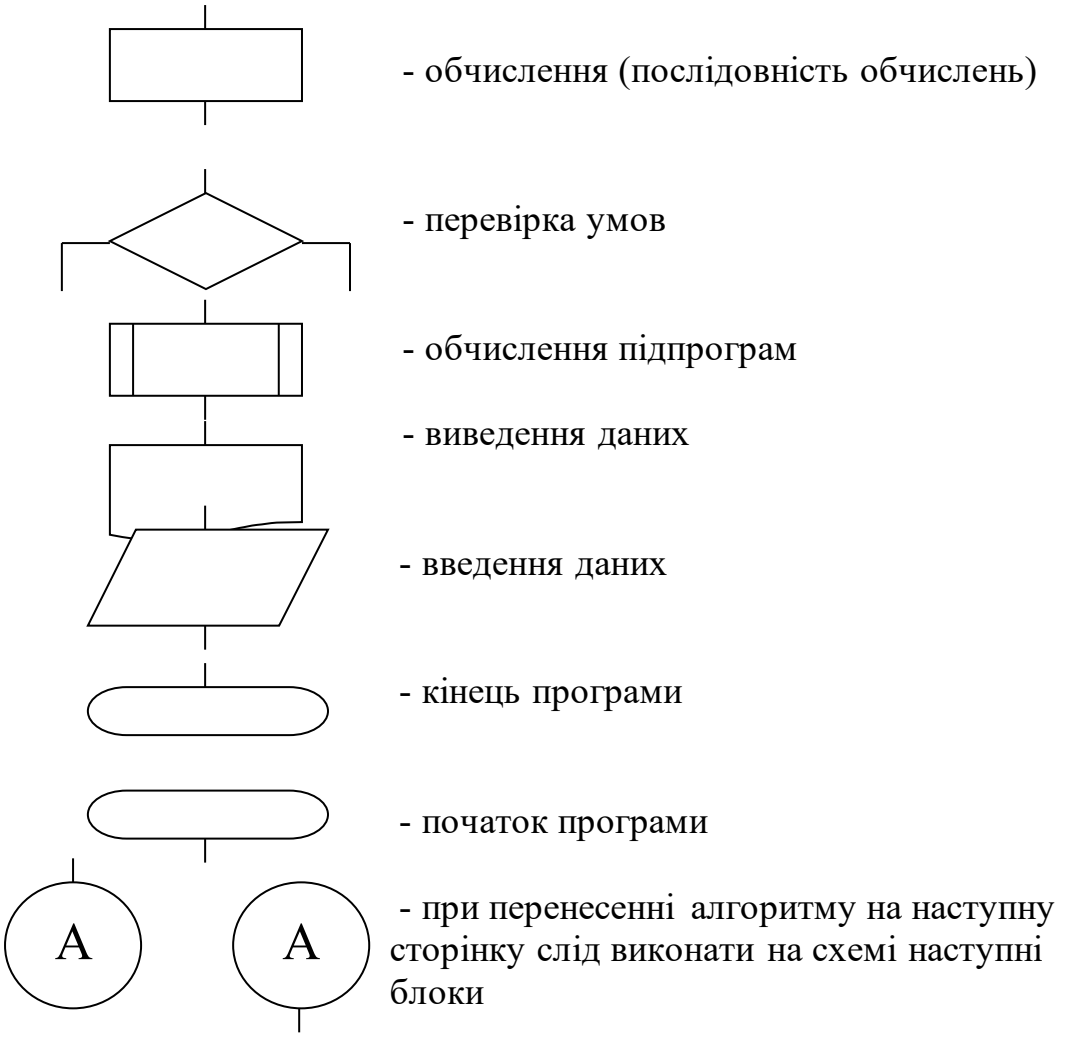


Рисунок 2 – Відповідність блоків та дій

Блокові символи можуть бути згруповані в основні (базові) конструкції, на підставі яких будуються будь-які алгоритми:

1. Лінійні алгоритми.

2. Розгалужені алгоритми.
3. Циклічні алгоритми.

Лінійний алгоритм – алгоритм, в якому блоки виконуються послідовно в порядку, заданому схемою.

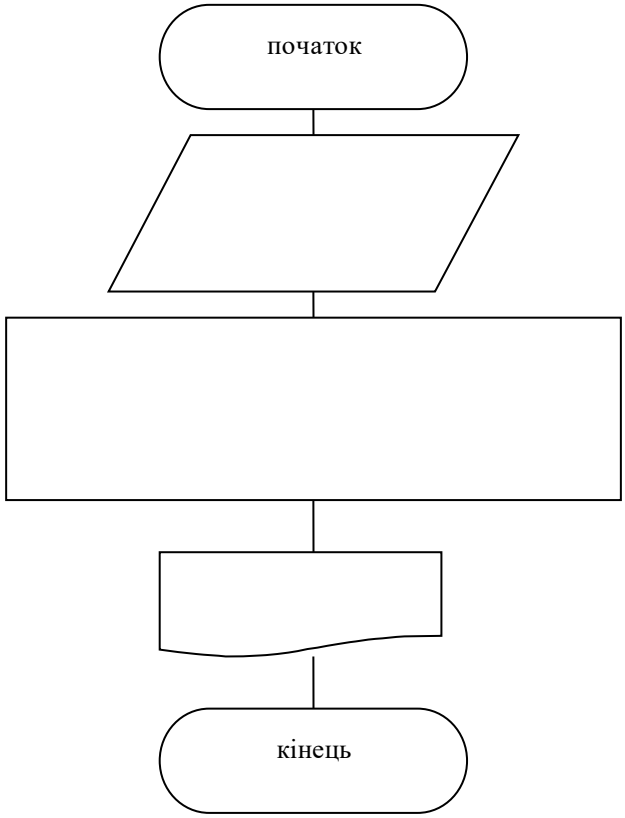


Рисунок 3 – Блок-схема за лінійним алгоритмом

Якщо для отримання результату необхідно застосовувати одну або багато формул, залежно від поставленого завдання, то у такому разі обчислювальний процес розгалужується. Так виникають ***розгалужені алгоритми*** (рисунок 4).

При розв'язанні більшості завдань доводиться багато разів виконувати обчислення за однією і тією ж формулою для різних змінних. Такі фрагменти в обчислювальних процесах називаються ***циклами.***

Для організації циклу треба:

1. До початку циклу задати початкове значення змінної, яка є початковим значенням циклу;
2. Змінити цей параметр перед кожним виконанням циклу;
3. Перевірити умову продовження або завершення циклу;
4. Управляти циклом (виходити або переходити на початок циклу).

Пункти 3-4 виконуються багато разів.

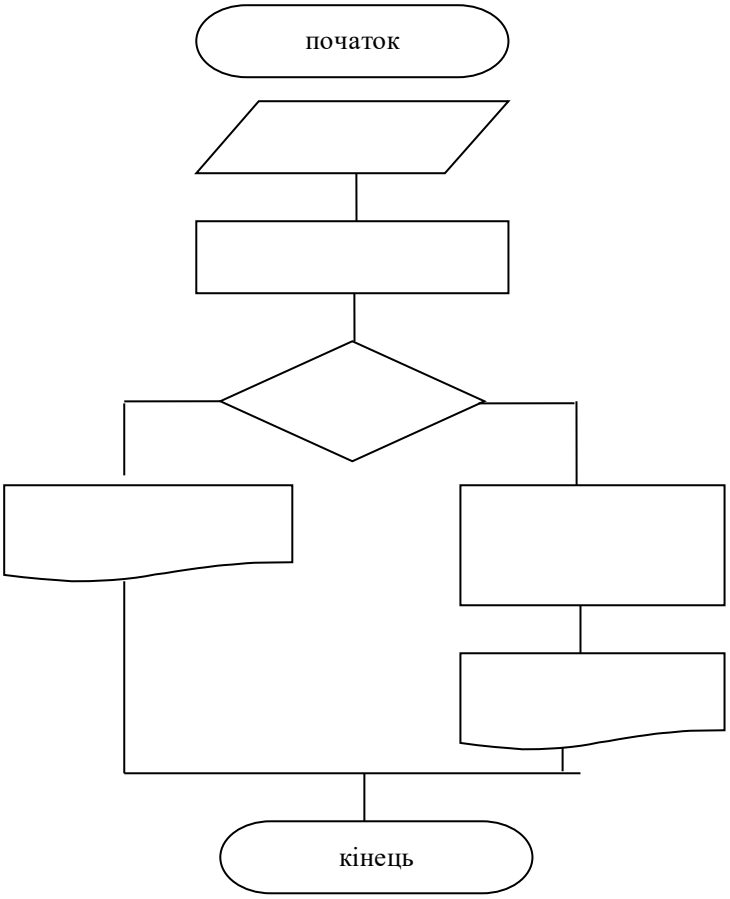


Рисунок 4 – Блок-схема розгалуженого алгоритму

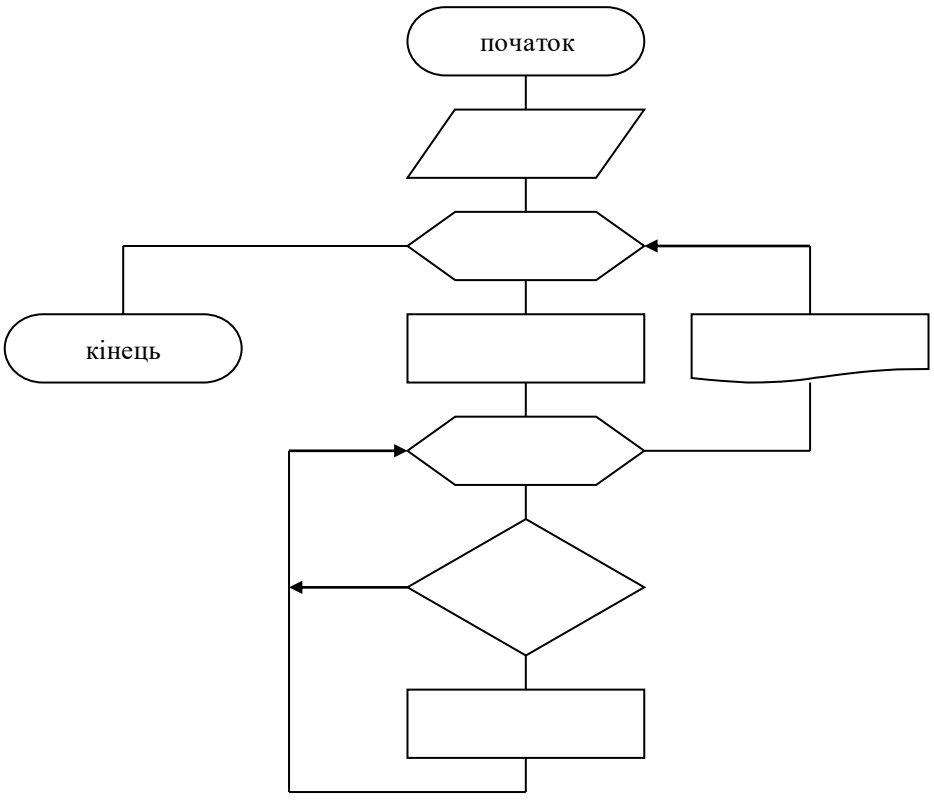


Рисунок 5 – Блок-схема вкладених циклів

Алгоритм структури вкладених циклів. У цикл, який називатимемо

зовнішнім, вкладемо декілька або 1 внутрішній цикл. Назвемо їх ***вкладеними***. Організація зовнішніх і вкладених циклів аналогічна звичайним циклам. Параметри внутрішнього і зовнішнього циклів не одночасні, тобто при певному значенні параметра зовнішнього циклу, параметр внутрішнього циклу по черзі приймає всі свої значення (рисунок 5).

2.5 Основні команди обміну даними

До основних команд обміну даних відносяться ***mov, xchg, lea***, а також група команд для роботи зі стеком – ***push, pop, pusha, popa, pushf, popf***.

Команда ***xchg*** призначена для обміну двох операндів, причому не допускається обмін «пам'ять-пам'ять», наприклад:

```
xchg dx,cx                ;обмін вмісту dx и cx
xchg ax,word ptr[di]      ;обмін між ax та словом за адресою в di
```

Команда ***lea*** призначена для завантаження адреси пам'яті в регістр, наприклад:

```
lea bx,mas                ;завантаження адреси масива mas в bx
```

Розглянемо команди для роботи зі стеком. Стек – це ділянка пам'яті, яка виділяється для тимчасового зберігання даних програми. Для роботи зі стеком призначені сегментний регістр *ss*, регістр вказівника стека *esp* (*sp*) та регістр бази стека *ebp* (*bp*). Запис та читання у стеку виконується у відповідності з принципом *Last In First Out*, та по мірі запису даних в стек він росте в сторону молодших адрес. Регістр *esp*(*sp*) завжди вказує на останній записаний в стек елемент. Якщо стек порожній, то значення цього регістра дорівнює адресі останнього байта сегмента, виділеного під стек. При занесенні элемента в стек спочатку зменшується значення *esp* (*sp*), а потім за адресою в цьому регістрі записується элемент. Для запису в стек використовується команда ***push елемент***.

При отриманні даних зі стека спочатку копіюється елемент за адресою з *esp* (*sp*), а потім значення цього регістра збільшується. Для читання зі стека використовується команда ***pop элемент***.

Наприклад, для запису в регістр *es* значення із *ds* можна використовувати команди:

```
push ds
pop es
```

Якщо необхідно отримати доступ до елементів не на вершині стека, а всередині, то використовують регістр *ebp* (*bp*), в який записувается вміст

регістра *esp* (*sp*), а для регістра *ebp* (*bp*) можна використовувати адресацію зі зміщенням.

Команди ***pusha*** та ***popa*** служать для збереження в стеку групи регістрів загального призначення та не мають операндів. Команда ***pushf*** також не має операндів та зберігає в стеку регістр прапорів. Команда ***popf*** відновлює регістр прапорів зі стека.

2.6 Арифметичні команди та операції

До найбільш часто використовуваним цілочисельних арифметичних команди відносяться команди двійкової арифметики (додавання – ***add, inc***; віднімання – ***sub, dec***; множення – ***imul, mul***; ділення – ***idiv, div***; зміна знака – ***neg***) та команди перетворенняя типів (***cbw, cwd*** і т.д.). Команди *add* та *sub* мають два операнда, над якими виконується арифметична операція (відповідно додавання чи віднімання), результат якої записується в перший операнд. Команди *inc* та *dec* мають один операнд, значення якого відповідно збільшується чи зменшується на одиницю.

При виконанні множення та ділення можуть бути використані команди, які враховують чи не враховують знак операндів. Команди *mul* та *div* працюють з беззнаковими операндами, а команди *imul* та *idiv* враховують знак. Розглянемо виконання множення чисел зі знаком. Команда *imul* синтаксично может мати один, два чи три операнда. Форма команди з одним операндом потребує дійсної вказівки розташування тільки першого співмножника, який може бути розташований в комірці пам’яті чи регістрі. Розташування другого співмножника фіксовано та залежить від розміру першого співмножника:

- якщо операнд в команді – байт, то другий співмножник повинен розміщуватись в al;
- якщо операнд – слово, то другий співмножник повинен розміщуватись в ax;
- якщо операнд в команді – подвійне слово, то другий співмножник розміщується в регістрі eax.

Результат множення для команды з одним операндом розсташовується також в точно визначеному місці, яке визначається розміром співмножників:

- при множенні байтів добуток розміщується в ax;
- при множенні слів результат розміщується в парі регістрів dx:ax (в dx – старша частина добутку);
- при множенні подвійних слів результат розміщується в парі edx:eax.

В команді з двома операндами перший та другий операнди визначають місце розташування першого та другого співмножників, результат буде записаний на місце першого співмножника. В команді з трьома операндами перший операнд визначає розташування результата, другий операнд – розташування першого співмножника, третій операнд може бути безпосередньо заданим значенням розміром байт, слово чи подвійне слово. Команди множення

впливають на прапори переносу та переповнення в регістрі прапорів (CF та OF). Якщо результат малий і поміщається в молодшу частину, відведену для зберігання добутку, то CF=OF=0, і вміст старшої частини є розширенням знака. В іншому випадку факт переповнення або перенесення повинен бути оброблений програмно при подальшій обробці результату множення.

Рисунок 6 – Класифікація арифметичних команд

При діленні чисел, так само, як і при множенні, може враховуватися або не враховуватися знак. Розглянемо команду знакового ділення *idiv*, що має один операнд, який є дільником і може знаходитися в пам'яті або в регістрі і мати розмір в байт, слово або подвійне слово. Місцезнаходження діленого задається неявно за такими правилами:

- якщо дільник розміром в байт, то ділене має бути розташоване в регістрі ax. Після операції частка поміщається в al, а залишок – в ah;
- якщо дільник розміром слово, то ділене має бути розташоване в парі регістрів dx:ax, причому молодша частина діленого повинна знаходитися в ax. Після ділення частка поміщається в ax, а залишок – в dx.

- якщо дільник розміром подвійне слово, то ділене має бути розташоване в парі регістрів edx:eax. Після ділення частка поміщається в eax, а залишок в edx.

Помилка ділення виникає при діленні на нуль або якщо частка занадто велика для розміщення в регістрі eax/ax/al.

Для того щоб ділене розміщувати в парі регістрів або щоб розширити байт до слова, використовуються команди розширення знака *cbw* (розширення al до ax) і *cwd* (розширення ax до dx:ax).

Арифметичні команди над двійковими цілими числами

Для виконання додавання можуть використовуватись команди:

ADD a,b – додавання без урахування перенесення. Команда додає значення двох операндів a та b і розміщує результат замість першого операнда. Операнд може знаходитися в пам'яті, в регістрі або бути безпосереднім параметром у команді. Не допускається додавання двох операндів, які знаходяться в пам'яті. Першим операндом не може бути безпосередній параметр. Встановлюються прапорці AF, CF, OF, FF, SF, ZF.

ADC a,b – додавання з перенесенням. Команда додає значення двох операндів, а також значення прапорця перенесення CF (a+b+CF), і розміщує результат замість першого операнда. Обмеження на операнді такі ж, як і в команді ADD. Установлюються прапорці AF, CF, OF, FF, SF, ZF. Ця команда використовується для програмування арифметичних операцій над довгими цілими числами.

INC a – збільшення на 1. Команда додає 1 до значення операнда a. Операнд може знаходитися в пам'яті або в регістрі. Установлюються прапорці AF, OF, PF, SF, ZF.

Для виконання віднімання можуть використовуватись команди:

SUB a,b – віднімання. Команда віднімає вид значення першого операнда a значення другого операнда b і розміщує результат замість першого операнда. Операнд може знаходитися в пам'яті, в регістрі або бути безпосереднім параметром у команді. Не допускається віднімання, коли обидва операнди знаходяться в пам'яті. Першим операндом не може бути безпосередній параметр. Установлюються прапорці AF, CF, OF, PF, SF, ZF.

SBB a,b – віднімання з позикою. Команда віднімає із значення першого операнда a значення другого операнда b, зменшує результат на значення прапорця CF (a-b-CF) і розміщує результат замість першого операнда. Обмеження операндів такі ж, як і в команді SUB. Установлюються прапорці AF, CF, OF, PF, SF, ZF.

DEC a – зменшування на 1. Команда зменшує значення операнда a на 1. Операнд може знаходитися в пам'яті або в регістрі. Установлюються прапори AF, OF, PF, SF, ZF.

Для виконання множення цілих двійкових чисел можуть

використовуватись команди:

MUL a – множення. Команда виконує множення вмісту акумулятора (AL або AX) на значення операнда a. Результат розміщується відповідно в акумуляторі AX або в парі регістрів DX:AX. Множники розглядаються як числа без знаків. Установлюються прапорці CF і OF. Прапорці AF, PF, SF і ZF не визначені.

IMUL a – множення із знаком. Команда виконує множення вмісту акумулятора (AL або AX) на значення операнда a. Результат розміщується відповідно в акумуляторі AX або в парі регістрів DX:AX. Множники розглядаються як числа із знаками. Прапорці CF і OF скидаються в 0, коли старша частина добутку (регістр AH або DX) є поширенням знака, який знаходиться в молодшій частині добутку (AL або AX), інакше ці прапорці установлюються в 1.

Для виконання ділення цілих двійкових чисел можуть використовуватися команди:

DIV a – ділення. Команда виконує ділення вмісту акумулятора на операнд a. Коли операнд a – байт, тоді ділене розміщується в AX, ціла частина в AL, а залишок – в AH. Коли операнд a – слово, тоді ділене розміщується в парі DX:AX (подвійне слово), ціла частина розміщується в AX, а залишок в DX. Операнди розглядаються як цілі числа без знака. При діленні на 0 виконується переривання типу 0.

IDIV a – ділення із знаком. Команда виконує ділення операндів так само, як у команді DIV. Операнди розглядаються як числа із знаками. Прапорці не визначені. При діленні на 0 виконується переривання типу 0.

Арифметичні команди над двійково-десятковими неупакованими числами

Арифметичні операції над двійково-десятковими неупакованими числами виконуються порозрядно, починаючи з молодших розрядів операндів. Слід нагадати, що для зберігання однієї цифри такого числа потрібен один байт.

Виконання операції над черговими розрядами операндів, тобто над черговими байтами, починається з того, що над цими байтами виконується відповідна двійкова команда додавання (ADD, ADC), віднімання (SUB, SBB), множення (MUL) або ділення (DIV). Після того виконується корекція отриманого результату, щоб отримати правильне значення неупакованої десяткової цифри результату:

AAA – корекція результату додавання. Команда корегує результат операції ADD. У чотирьох молодших бітах регістра AL записується значення десяткової цифри, а в чотирьох старших бітах записуються нулі. Коли в AL результат попередньої операції перевищує 9, тоді CF=1 і AH=AH+1.

AAS – корекція результату віднімання. Команда корегує результат операції SUB. У чотирьох молодших бітах регістра AL записується значення десяткової цифри, а в чотирьох старших бітах записуються нулі. Коли потрібно

мати позику із старшого розряду, тоді CF=1 і AH=AH+1.

AAM – корекція в AX результату множення. Команда корегує результат операції MUL. У регістрі AH записується старша цифра добутку, а в AL – молодша цифра.

AAD – корекція ділення. Команда корегує ділене в регістрі AL перед виконанням ділення так, що наступне ділення DIV дає неупаковану десяткову частку: в AL – результат, в AH – нуль.

Арифметичні команди над двійково-десятковими упакованими числами

Арифметичні операції над двійково-десятковими упакованими числами виконуються по байтах, тобто задана операція додавання або віднімання виконується одночасно над двома розрядами числа, які знаходяться в одному байті, починаючи з молодших розрядів.

Виконання операції додавання або віднімання здійснюється в два етапи. Спочатку над цим байтом виконується команда додавання (ADD, ADC) або віднімання (SUB, SBB), а потім виконується корекція результату, щоб отримати правильні значення упакованих десяткових цифр.

Для корекції результату використовуються такі команди:

DAA – корекція результату додавання. Команда корегує в AL результат операції ADD, ADC. У регістрі AL створюються дві правильні двійково-десяткові цифри результату. Коли виникає перенесення в наступний байт, тоді CF=1.

DAS – корекція результату віднімання. Команда корегує в AL результат операції SUB, SBB. У регістрі AL створюються дві правильні двійково – десяткові цифри результату. Коли виникає запозичення із наступного байта, тоді CF=1.

У процесорі IBM PC не передбачені команди корекції множення і ділення упакованих двійково-десяткових чисел. Тому для виконання цих операцій треба спочатку розпакувати черговий байт, виконати відповідну операцію над неупакованими цифрами, а потім упакувати результат.

Використання знака

До цього часу ми вважали, що додаємо додатні числа. Насправді, будь-яке число для процесора – всього лише послідовність двійкових розрядів – бітів. Тому в регістрі, що складається з 32 біт, можна закодувати 232 різних числа, а якими вони будуть: додатніми, від’ємними, цілими або дробовими, вирішуємо самі.

Спробуємо зрозуміти, як в комп'ютері кодуються від’ємні числа, для чого перейдемо від справжніх 32-бітових чисел до «іграшкових» 4-бітових. Такими числами значно простіше оперувати, а те, що вдалося зрозуміти на їх прикладі, легко замінити на будь-яке число двійкових розрядів.

І так, регістр, що складається з чотирьох бітів, може перебувати в 2^4=16

різноманітних станах, і логічно поділити його навпіл: 8 станів (включаючи 0) будуть «додатніми», 8 – «від'ємними». Щоб від'ємне число відразу можна було відрізнити від додатного, зробимо старший (крайній лівий) біт знаковим; нехай він буде дорівнювати нулю для додатніх чисел і одиниці – для від'ємних.

З урахуванням викладеного матеріалу, в чотирьох бітах можуть поміститися (разом з нулем) такі додатні числа:

0→0000
1→0001
2→0010
3→0011
4→0100
5→0101
6→0110
7→0111

Від'ємні числа легко отримати з додатніх, якщо врахувати, що сума додатного числа і такого ж по абсолютній величині від'ємного дорівнює нулю. Якщо просто звернути всі біти додатного числа, записавши замість нуля 1, а замість одиниці 0, то всі біти суми будуть дорівнювати одиниці, наприклад,

0001+1110=1111.

В даному випадку використовується нехитре правило додавання двійкових розрядів 1+0=0+1= 1.

А тепер уявимо, що до суми, що складається з одиниць, додається ще одна одиниця. Згідно з правилами двійкової арифметики, 1+1 = «нуль пишемо, один маємо на увазі». Адже 1+1 – це два – основа двійкової системи числення, тому в молодшому розряді пишеться 0, а одиниця переноситься в старший розряд (в десятковій системі цьому відповідає сума 5+5, яка теж дорівнює «нуль пишемо, один маємо на увазі»).

Це означає, що поодинокі біти від додавання ще однієї одиниці «поваляться», перетворяться в нулі, і результатом додавання буде одиниця, яка вийшла за межі регістра:

-1+1=1111+0001=10000

і чотири нулі, як і повинно бути. Отже, згідно з правилом звернення бітів і додавання одиниці, від'ємні числа будуть кодуватися так:

-1→1111
-2→1110
-3→1101
-4→1100
-5→1011
-6→1010
-7→1001

Тепер у нас є коди 15 чисел (0, 7 додатніх і 7 від'ємних). Всього в 4 бітах розміщується 16 чисел, тому додамо до них код для -8. Його не можна отримати звернувши біти, тому що немає відповідного додатного числа. Але можна скористатися тим, що сума додатного числа і відповідного від'ємного

для регістра з чотирьох бітів завжди дорівнює 10000 або, в десяткового запису, -16.

Щоб, наприклад, отримати двійковий код для -7, необхідно представити у вигляді двійкового числа різницю 16–7=910=10012. Перевірка показує, що 7+(-7)=0111+1001=10000=1610, що і необхідно. Проробивши те ж саме з вісімкою, знайдемо двійкове представлення для -8:

-8=16–8=8=1000.

Отриманий таким чином код називається додатковим, тому що від'ємне число виходить в ньому доповненням додатного до 16. Відомі й інші коди для від'ємних чисел, але додатковий використовується частіше інших завдяки своїм чудовим властивостям.

По-перше, в ньому існує тільки один нуль, адже -0=1111+1=10000=0, тому що старший одиничний біт не вміщується в 4-бітовому регістрі і зникає.

По-друге, знак числа можна міняти нескінченну кількість разів без будь-яких змін і втрат. Щоб змінити знак числа -5, потрібно звернути всі біти числа 1011 і додати одиницю: $0100+1=0101=5_{10}$. Потім можна ще раз отримати -5, знову звернувши біти і додавши одиницю, і так до нескінченності.

Нарешті, по-третє, додатковий код дозволяє звести віднімання до додавання. Щоб, наприклад, відняти від п'яти три, достатньо записати в регістр 3, потім звернути (інвертувати) всі біти і додати одиницю, після чого в регістрі з'явиться число 3 в додатковому коді, і потім вже додати п'ять. Отримана сума буде дорівнює двом, тому що п'ятірку можна уявити сумою двійки і трійки. Але сума звичайної трійки і трійки в додатковому коді дорівнює, як ми вже зрозуміли, нулю. Отже, 5+3 (в додатковому коді) = 2. Віднімання шляхом додавання числа в додатковому коді зручне процесору, тому що інвертування і складання бітів для нього дуже прості.

Переповнення розмірної сітки

Додатні та від'ємні числа, з якими ми познайомилися в попередньому розділі, виходять за межі чотирьох бітів, тому багато результатів дій над ними виявляються неправильними. Зрозуміло, що в 32, 16 і навіть 8 біт місця набагато більше, але і там потрібно вміти визначати, коли результат операції вірний, а коли ні. Спробуємо дізнатися межі дозволеного для 4 біт, з метою застосувати отримані знання до «реальних» чисел, що існують не в тісних 4-бітових комірках, а в просторих, але все ж кінцевих 32-бітових «вольєрах».

Насамперед зазначимо, що процесор нічого не знає ні про додатні, ні про від'ємні числа. Він здатний лише додавати двійкові розряди за правилом: 0+0=0, 0+1=1+0=1, 1+1=«0 пишемо, один маємо на увазі». Про те, що 1111_2 – додатне число 15, або -1 в додатковому коді, – знає лише програміст. Тому «місткість» будь-якого числа бітів залежить від типів доданків і очікуваного результату. Якщо ми вважаємо, що в 4 бітах зберігаються тільки додатні числа, то зрозуміло, що результат їх додавання буде вірним, поки сума не перевищить 15 (1111_2). Наприклад, 7 (0111_2) і 5 (0101_2) дадуть в сумі 12 (1100_2) – число, яке легко поміститься в 4 біти.

Але якщо скласти, скажімо, дві вісімки, то трапиться переповнення, тобто сума 16 не поміститься в 4 біти, і програміст побачить замість шістнадцяти чотири нулі і піднятий прапор переносу C.

Якщо ж вважати, що додаються числа зі знаком, то прапор C вже не буде ознакою переповнення, бо він підніметься при додаванні будь-яких від'ємних чисел. Наприклад, додавання -5 (1011_2) та -2 (1110_2) не призведе до переповнення, але прапор C при цьому все одно буде піднятий через перенос, що виник при додаванні двох знакових бітів.

Тому процесор інакше визначає переповнення при додаванні чисел зі знаком і піднімає в цьому випадку зовсім інший прапор O (від слова «overflow» – переповнення), показаний на рисунку 7.

Щоб краще звикнути до двійкової арифметики, спробуємо здогадатися, як процесор дізнається про переповнення, що виникло при додаванні чисел зі знаком. Для нас, людей, переповнення при додаванні 4-бітових чисел виникає, коли результат менше -8 або більше 7.

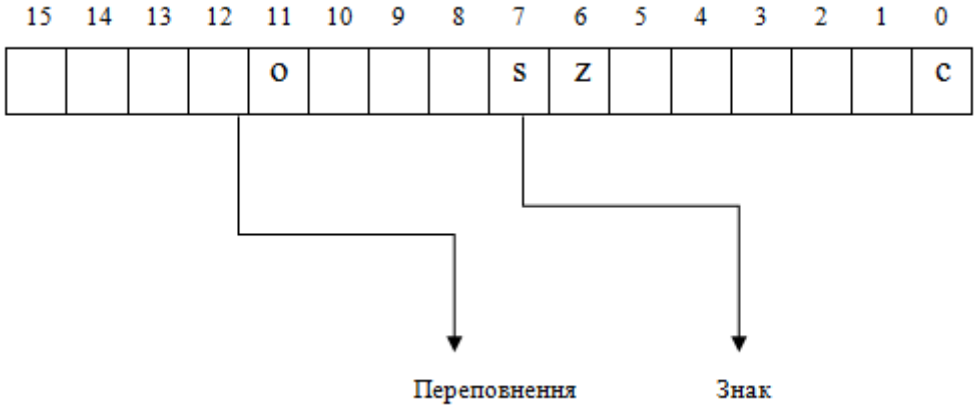


Рисунок 7 – Прапори переповнення та знаку

Але процесор оперує тільки двійковими розрядами, і йому ніколи переводити числа з двійкової системи в десяткову. Тому він може використовувати тільки окремі біти доданків і результату. Щоб здогадатися, як він це робить, подивимося на «гранично допустиму» суму двох додатних чисел -7 (0111_2). Якщо до цієї суми додати одиницю, вийде число 1000_2, в якому «зіпсований» знаковий біт: сума додатних чисел (4+4, 3+5, 2+6, 1+7) дорівнює 8, але якщо ці числа зберігаються в чотирьох бітах, їх сума дорівнюватиме -8 (1000_2): складові додатні, а сума від'ємна.

Ця відмінність знака суми від знака доданків, очевидно, збережеться і для іншого, невірного додавання додатних чисел, тому що їх сума буде коливатись від 1000_2 (4+4, 3+5, 2+6, 1+7) до 1110_2 (7+7). І всюди знаковий біт буде дорівнювати одиниці, а доданки додатні.

Тепер подивимося, що станеться при додаванні від'ємних чисел. Тут остання вірна сума буде дорівнювати -8 (1000_2). Якщо скласти числа, сума яких дорівнює -9, наприклад -2 (1110_2) і -7 (1001_2), то вийде 0111 – додатне число 7. І

тут, як видно, знак суми відрізняється від знака доданків. Ця закономірність збережеться і для інших від'ємних доданків, тільки їх сума буде зменшуватися, а не збільшуватися, як при додаванні додатних чисел зі знаком. Останнє значення суми вийде при додаванні максимально можливих від'ємних чисел - $8+(-8)=1000_2+1000_2=0000_2$, і всюди знак суми буде не таким, як у доданків. Перевірити відмінність знака суми від знаків доданків процесору досить легко. Для цього він повинен порівняти знакові біти чисел, що беруть участь у додаванні та знаковий біт результату S (від слова «sign», знак), який зберігається в регістрі прапорів на сьомій позиції. Цей прапор піднімається, коли результат операції від'ємний, і опускається, коли той додатний. Зауважимо, що умова переповнення (відміна знака суми від знака доданків) не пов'язана з розміром регістрів і тому може бути застосована до будь-якого числа бітів: 8, 16, 32.

Практичне пояснення операції додавання та віднімання

Як змусити число «переїхати» з одного регістра eax в два – edx:eax? Ми побачимо, що перехід двійкового числа з регістра в регістр не складніше, ніж перехід десяткового числа з одного десяткового розряду в два.

Дійсно, помістимо нуль в один десятковий розряд і будемо додавати туди по одиниці. Все буде добре, поки розряд не стане дорівнювати 9. Якщо додати до нього одиницю, виникне переповнення, тобто розряд обернеться в нуль і з'явиться «один в уяві» – перенесення в наступний десятковий розряд. Очевидно, регістр eax нічим не відрізняється від десяткового розряду.

Додаючи до нього одну одиницю за одною, потрібно стежити за прапором перенесення. І кожен раз, коли він піднятий, додавати одиницю до регістру edx.

Стежити за перенесенням здатна спеціальна команда ***adc***, яка додає число, як звичайна команда ***add***, а потім додає до суми вміст прапора переносу. За допомогою команди ***adc*** формування числа для перевірки на «простоту» буде виглядати так:

```
add eax, 1      ;збільшимо
adc edx, 0      ; число
```

Перша інструкція збільшує на одиницю eax, а друга додає до edx прапор C. Замість *add eax, 1* не можна використовувати інструкцію *inc eax*, тому що вона не впливає на прапор переносу. Отже, комбінацію ***add***, ***adc*** можна використовувати не тільки для додавання до «довгого» числа одиниці, але і для додавання дуже великих чисел, що не вміщуються ні в двох, ні в чотирьох байтах. Нехай, наприклад, одне число зберігається в парі регістрів edx:eax, а друге в парі ecx:ebx. Для складання таких чисел знадобляться інструкції:

```
add eax,ebx:eax          ;eax+ebx
adc edx,ecx:edx          ;edx+ecx+C
```

Перша інструкція виконує звичайне додавання, а друга додає регістри edx і ecx і додає до їх суми прапор переносу C. «Довгі» числа можна не тільки додавати, а й віднімати за допомогою команд ***sub***, ***sbb***. Зрозуміти їх роботу неможливо без уміння розрізняти додавання і віднімання.

Ми вважали, що віднімання еквівалентне збільшенню числа, записаного в додатковому коді. Нехай, наприклад, числа 1 і 3 зберігаються в 4-бітових регістрах. Тоді ми думали, що для віднімання від одиниці три, достатньо скласти 1 (0001_2) і 3 в додатковому коді (1101_2). Дійсно, склавши ці числа, отримаємо 1110_2, тобто, -2 в додатковому коді.

Здавалося б, все вірно. Але крім числа треба ще правильно встановити прапори, а при додаванні одиниці та додаткового коду для -3 не виникає ні переповнення, ні переносу зі старшого розряду.

Між тим, віднімання від меншого числа більшого, як в нашому прикладі, має супроводжуватися позикою зі старшого розряду. Щоб зрозуміти, що це таке, спробуємо відняти від одиниці три, не вдаючись до двійкового додаткового коду, а керуючись звичайними правилами віднімання «стовпчиком».

Віднімаючи самі молодші розряди, отримаємо 0. Переходячи на крок вліво зіткнемося з необхідністю віднімати від нуля одиницю. Це неможливо, тому, як і при відніманні десяткових чисел, позичимо одиницю старшого розряду (візьмемо першу «позику»). Вона має вдвічі більшу вагу, тому віднімання від одиниці старшого розряду, одиниця молодшого дає одиницю. Далі переходимо до віднімання від нуля, з якого вже взята одиниця, тобто «звичайного нуля». Знову позичаємо одиницю старшого розряду (беремо другу «позику», з якого «покриваємо» перший розряд) і знову отримуємо одиницю молодшого. Продовжуючи у тому ж руслі, одержимо ряд одиниць, але останню одиницю ми отримаємо, якщо запозичити одиницю неіснуючого дев'ятого розряду. Ось це і є «позика», про яку процесор повинен повідомляти підняттям якогось прапора. В процесорах Intel для цього вибрано прапор переносу C.

Для більшого розуміння повернемося до нашого прикладу, але тепер займемося числами, які зберігаються в реальних 8-бітових регістрах. Якщо додавати за допомогою команди add два числа 0000001_2 (1) і 11111101_2 ((-3) в додатковому коді), вийде 11111110_2 ((-2) в додатковому коді). При цьому прапор C виявиться опущеним (рівним нулю). Якщо ж для віднімання від одиниці трійку, використати команду ***sub***:

```
mov al,1
mov ah,3
sub al,ah
```

то в регістрі al знову з'явиться число -2 (11111110_2); точно таким же, як при додаванні 1+(-3), буде й прапор переповнення O, але прапор C тепер підніметься, що скаже нам про «позику» зі старшого розряду.

Тепер ми, нарешті, можемо повернутися до віднімання «довгих» чисел за допомогою команд ***sub*** і ***sbb***. Припустимо, що в регістрах edx:eax записаний

нуль. Тоді відняти від такого довгого числа одиницю можна командами:

```
sub eax,1
sbb edx,0
```

Перша команда призводить до того, що в eax всі біти встановлюються в одиницю і виникає запозичення зі старшого розряду (піднімає прапор переносу C). Друга команда віднімає прапор C, який дорівнює, у нашому випадку, одиниці, від числа, записаного в регістрі edx. Результат зрозумілий: в edx і eax всі біти перетворяться в одиницю, тобто в парі регістрів з'являється -1 в додатковому коді.

Послідовність команд ***sub, sbb, sbb***... можна застосувати при відніманні будь-яких «довгих» чисел. Нехай, наприклад, перше число знаходиться в парі регістрів edx:eax, а друге — в парі eax:ebx. Тоді фрагмент програми віднімання від першого числа другого, буде виглядати так:

```
sub eax,ebx             ;віднімаємо «молодші» біти
sbb edx,ecx:edx                 ;edx - ecx – C
```

Перша інструкція sub віднімає від eax вміст ebx і записує результат в eax, вона займається «молодшими» бітами. Друга інструкція посилає в edx результат операції edx - ecx - C.

Операції множення та ділення

Ми вже говорили про те, що множення майже на будь-яке число можна представити комбінацією зсувів і додавання. Але було б дивно ділити числа інструкцією div, а множити, створюючи кожен раз нові хитрі комбінації за допомогою інструкцій add і shl.

Тому в процесорі передбачена інструкція універсального множення mul. Як і div, інструкція єдина в трьох діях: операнд, який зберігається в байті, множиться на al, а результат опиняється в регістрі ax:

mul <операнд>

Якщо операнд – слово, процесор множить його на ax, а результат опиняється в eax. Нарешті, операнд, який зберігається в подвійному слові, множиться на eax, а результат опиняється в парі регістрів edx:eax.

Як правило, нове знання породжує нову проблему: результат множення, що зберігається в двох регістрах і займає 64 біт, необхідно виводити на екран, а для цього процедура wsprintf не підійде, тому що вона оперує тільки 32-бітовими значеннями.

Приклад застосування команди mul:

```
MOV AL,02H;   завантаження першого співмножника (02H = 02D)
          до регістру AL
MOV BL,80H;   завантаження другого співмножника (80H = 128D)
```

до регістру BL
MUL BL; множення операндів, результат знаходиться в регістрі AX (0100H = 256D)

Множення двійкових чисел без знака

Для множення чисел без знака призначена команда

mul співмножник_1

У команді вказано тільки один операнд-співмножник. Другий операнд-співмножник_2 заданий неявно. Його місце розташування фіксоване і залежить від розміру співмножників. Добуток складається з двох частин і в залежності від розміру операндів розміщується в двох місцях – на місці співмножника 2 (молодша частина) і в додаткових регістрах ah, dx, edx (старша частина). Щоб дізнатися, що результат досить малий і розмістився в одному регістрі, або перевищив розмірність регістра, і старша частина виявилася в іншому регістрі, потрібно проаналізувати прапори перенесення cf і переповнення of:

- якщо старша частина результату 0, то після операції множення прапори cf = 0 і of = 0;

- якщо ж ці прапори НЕ 0, то це означає, що результат вийшов за межі молодшої частини добутку і складається з двох частин, що і потрібно враховувати при подальшій роботі.

Таблиця 7 – Розміщення операндів та результату при множенні

Співмножник_1	***Співмножник _2***	***Результат***
Байт	al	16 бит в ax: al – молодша частина результата; ah – старша частина результата.
Слово	ax	32 бит в парі dx:ax: ax – молодша частина результата; dx – старша частина результата.
Подвійне слово	eax	64 бит в паре edx:eax: eax – молодша частина результата; edx – старша частина результата.

Множення двійкових чисел зі знаком

Для множення чисел зі знаком призначена команда

imul операнд_1

Ця команда виконується так само, як і команда mul. Відмінною особливістю команди imul є тільки формування знака. Якщо результат малий і розміщується в одному регістрі (тобто якщо cf = of = 0), то вміст іншого регістра (старшої частини) є розширенням знака – всі його біти дорівнюють старшому біту (знаковому розряду) молодшої частини результату. В іншому випадку (якщо cf = of = 1) знаком результату є знаковий біт старшої частини результату, а знаковий біт молодшої частини є значущим бітом двійкового коду результату.

Ділення двійкових чисел без знака

Для ділення чисел без знака призначена команда **div.** Дільник може знаходитися в пам'яті або в регістрі і мати розмір 8, 16 або 32 біт. Місцезнаходження діленого фіксоване і так само, як в команді множення, залежить від розміру операндів. Результатом команди ділення є значення частки та залишку.

div <операнд>

Після виконання команди ділення вміст прапорів невизначений, але може виникнути переривання з номером 0, названого «ділення на нуль». Цей вид переривання відноситься до так званих винятків. Цей різновид переривань виникає всередині мікропроцесора через деякі аномалії під час обчислювального процесу. Переривання 0 – «ділення на нуль» – при виконанні команди div може виникнути з наступних причин:

- дільник дорівнює нулю;

- частка не входить до відведеної під неї розрядної сітки, що може виникнути в наступних випадках:

- 16 – при розподілі діленого величиною «слово» на дільник величиною «байт», причому значення діленого в більш ніж 256 разів більше значення дільника;

- при діленні діленого величиною «подвійне слово» на дільник величиною «слово», причому значення діленого в більш ніж 65 536 разів більше значення дільника;

- при діленні діленого величиною «четверне слово» на дільник величиною «подвійне слово», причому значення діленого в більш ніж 4 294 967 296 разів більше значення дільника.

Приклад застосування команди div:

```
MOV AX, 1111H; завантаження діленого (1111H = 4369D) до
        регістру AX
MOV bx, 20H;   завантаження дільника (20H = 32D) до регістру BL
CWD;           формування необхідного розміру діленого
DIV bx;        ділення, частина в регістрі AX дорівнює 0088H,
        залишок в регістрі DX дорівнює 0011H
```

Таблиця 8 – Розташування операндів та результата при діленні

Ділене	***Дільник***	***Частка***	***Залишок***
Слово 16 біт в регістрі ax	Байт – регістр чи комірка пам’яті	Байт в регістрі al	Байт в регістрі ah
32 біт: dx – старша частина; ax – молодша частина	16 біт регістр чи комірка пам’яті	Слово 16 біт в регістрі ax	Слово 16 біт в регістрі dx
64 біт: edx – старша частина; eax – молодша частина	Подвійне слово 32 біти регистр чи комірка пам’яті	Подвійне слово 32 біти в регістрі eax	Подвійне слово 32 біти в регістрі edx

Ділення двійкових чисел зі знаком

Для ділення чисел зі знаком призначена команда

idiv дільник

Для цієї команди справедливі всі розглянуті положення, що стосуються команд і чисел зі знаком. Відзначимо лише особливості виникнення виключення 0 «ділення на нуль» у випадку чисел зі знаком. Воно виникає при виконанні команди idiv по одній з наступних причин:

- дільник дорівнює нулю;

- частка не входить в відведену для неї розрядну сітку. Це може виникнути:

- при діленні діленого величиною «слово» зі знаком на дільник величиною «байт зі знаком», причому значення діленого в більш ніж 128 разів більше значення дільника (таким чином, частка не повинна виходити за межі -128 +127);

- при діленні діленого величиною «подвійне слово зі знаком» на дільник величиною «слово зі знаком», причому значення діленого в більш ніж 32 768 разів більше значення дільника (таким чином, частка не повинна виходити за межі -32 768 +32 768);

- при діленні діленого величиною «четверне слово зі знаком» на дільник величиною в «подвійне слово зі знаком», причому значення діленого в більш ніж 2 147 483 648 разів більше значення дільника (таким чином, частка не повинна виходити поза межі ~ 2 147 483 648 + 2147483647).

Допоміжні команди для цілочисельних операцій

Ці команди розширюють байти в слова, слова – в подвійні слова і подвійні слова – в четверні слова (64-розрядні значення). Команди перетворення типу використовуються при перетворенні цілих зі знаком, так як вони автоматично заповнюють старші біти знову сформованого операнда значеннями знакового біта старого об'єкта. Ця операція призводить до цілих

значень того ж знака і тієї ж величини, що і вихідна, але вже в більш довгому форматі. Подібне перетворення називається операцією розповсюдження знака. Існують два види команд перетворення типу:

1. Команди без операндів – ці команди працюють з фіксованими регістрами:

- cbw (Convert Byte to Word) – команда перетворення байта (в регістрі al) в слово (в регістрі ax) шляхом розповсюдження значення старшого біта al на всі біти регістра ah;

- cwd (Convert Word to Double) – команда перетворення слова (в регістрі ax) в подвійне слово (в регістрах dx:ax) шляхом розповсюдження значення старшого біта ax на все біти регістра dx;

- cwde (Convert Word to Double) – команда перетворення слова (в регістрі ax) в подвійне слово (в регістрі eax) шляхом розповсюдження значення старшого біта ax на всі біти старшої половини регістра eax;

- cdq (Convert Double Word to Quarter Word) – команда перетворення подвійного слова (в регістрі eax) в четверне слово (в регістрах edx:eax) шляхом поширення значення старшого біта eax на всі біти регістра edx;

2. Команди *movsx* і *movzx*, що відносяться до команд обробки рядків. Ці команди мають корисну властивість:

- movsx операнд_1, операнд_2 – пересилання з розповсюдженням знака. Розширює 8- або 16-розрядне значення операнда_2, яке може бути регістром або операндом в пам'яті, до 16- або 32-розрядної значення в одному з регістрів, використовуючи значення знакового біта для заповнення старших позицій операнда_1. Дану команду зручно використовувати для підготовки операндів зі знаками до виконання арифметичних дій;

- movzx операнд_1, операнд_2 – переслати з розширенням нулем. Розширює 8- або 16-розрядне значення операнда_2 до 16- або 32-розрядної з очищенням (заповненням) нулями старших позицій операнда. Дану команду зручно використовувати для підготовки операндів без знака до виконання арифметичних дій.

3. У системі команд мікропроцесора є ще дві корисні команди:

- xadd призначення, джерело – обмін місцями та додавання. Команда дозволяє виконати послідовно дві дії: обміняти значення операндів призначення та джерела, помістити на місце операнда призначення суму: *призначення = призначення + джерело*.

- neg операнд – заперечення з додаванням до двох. Команда виконує інвертування значення операндів. Фізично команда виконує одну дію:

операнд = 0 – операнд, тобто віднімає операнд від нуля. Команду *neg операнд* можна застосовувати для зміни знака; виконання віднімання від константи. Команди sub і sbb не дозволяють відняти що-небудь від константи, так як константа не може бути операндом-приймальником в цих операціях. Тому дану операцію можна виконати за допомогою двох команд:

```
neg ax     ; зміна знака (ax)
add ax, 340; фактичне віднімання:(ax) = 340 - (ax)
```

Зробимо висновки по роботі з арифметичними командами

До цілочисельних арифметичних команд, які найбільш часто використовуються, відносяться команди двійкової арифметики (додавання – add, inc; віднімання – sub, dec; множення – imul, mul; розподіл – idiv, div; зміна знака – neg) і команди перетворення типів (cbw, cwd і т.п.). Команди add і sub мають два операнди, над якими виконується арифметична операція (відповідно додавання чи віднімання), результат якої записується в перший операнд. Команди inc і dec мають один операнд, значення якого відповідно збільшується або зменшується на одиницю.

При виконанні множення і ділення можуть бути використані команди, що враховують або які не враховують знак операндів. Команди mul і div працюють з беззнаковими операндами, а команди imul і idiv враховують знак. Розглянемо виконання множення чисел зі знаком. Команда imul по синтаксису може мати один, два або три операнда. Форма команди з одним операндом вимагає вказівки на місцезнаходження тільки першого співмножника, який може бути розташований в комірці пам'яті або регістрі. Розташування другого співмножника фіксоване і залежить від розміру першого співмножника:

- якщо операнд в команді – байт, то другий співмножник повинен розташовуватися в al;
- якщо операнд – слово, то другий співмножник повинен розташовуватися в ax;
- якщо операнд в команді – подвійне слово, то другий співмножник розташовується в регістрі eax.

Результат множення для команди з одним операндом знаходиться також в спеціально відведеному місці, яке визначається розміром співмножників:

- при множенні байтів добуток розміщується в ax;
- при множенні слів результат поміщається в пару регістрів dx:ax (в dx – старша частина добутку);
- при множенні подвійних слів результат поміщається в пару edx:eax.

В команді з двома операндами перший і другий операнди визначають місцезнаходження першого і другого співмножників, результат буде записаний на місце першого співмножника. В команді з трьома операндами перший операнд визначає місцерозташування результату, другий операнд – місцерозташування першого співмножника, третій операнд може бути безпосередньо заданим значенням розміром в байт, слово або подвійне слово. Команди множення впливають на прапори перенесення і переповнення в регістрі прапорів (CF і OF). Якщо результат малий і поміщається в молодшу частину, відведену під зберігання добутку, то CF=OF=0, і вміст старшої частини є розширенням знака. В іншому випадку факт переповнення або перенесення повинен бути оброблений програмно при подальшій обробці результату множення.

При діленні чисел, так само, як і при множенні, може враховуватися або не враховуватися знак. Розглянемо команду знакового розподілу idiv, що має

один операнд, який є дільником і може перебувати в пам'яті або в регістрі і мати розмір в байт, слово або подвійне слово. Місцезнаходження діленого задається неявно за такими правилами:

- якщо дільник розміром в байт, то ділене повинне розташовуватися в регістрі ax. Після операції приватне поміщається в al, а залишок – в ah;

- якщо дільник розміром в слово, то ділене повинно бути розташоване в парі регістрів dx: ax, причому молодша частина діленого повинна знаходитися в ax. Після ділення частка поміщається в ax, а решта – в dx.

- якщо дільник розміром з подвійне слово, то ділене повинно бути розташоване в парі регістрів edx:eax. Після поділу частка поміщається в eax, а залишок в edx.

Помилка поділу виникає при діленні на нуль або якщо частка занадто велика для розміщення в регістрі eax/ax/al.

Для того щоб ділене розміщувати в парі регістрів або щоб розширювати байт до слова, використовуються команди розширення знака cbw (розширення al до ax) і cwd (розширення ax до dx:ax).

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ІНФОРМАЦІЙНОЇ БЕЗПЕКИ ТА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КУРСОВА РОБОТА

з дисципліни «Системне програмування»
на тему:
«МОДЕЛЬ АРИФМЕТИКО-ЛОГІЧНОГО ПРИСТРОЮ»

студента _____
групи _____
освітньо-кваліфікаційного рівня бакалавр
напряму підготовки 6.050102 – Комп'ютерна інженерія
Керівник к.т.н., доцент Куницька С. Ю._____

Національна шкала _____
Кількість балів: _____Оцінка: ECTS _____

Члени комісії:

_____ Куницька С. Ю._____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

_____ _____
(підпис) (прізвище та ініціали)

Черкаси 2020

ДОДАТОК Б

Форма № У 6.01

Черкаський державний технологічний університет

Кафедра _____.
Дисципліна _____.
Напрям підготовки _____.
Курс _____ Група _____ Семестр _____.

ЗАВДАННЯ
на курсову роботу студента

_____

(прізвище, ім‘я, по батькові студента)

1. Тема курсової роботи: _____
_____
_____

2. Термін здачі студентом курсової роботи: _____

3. Вихідні дані до роботи: _____
_____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці):
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

5. Дата видачі завдання: _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Примітка
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

Студент:

_____. _____.

(підпис) (прізвище, ім‘я, по батькові)

Керівник:

_____. _____.

(підпис) (прізвище, ім‘я, по батькові)

«___» _____ 20___ р.

ДОДАТОК В

БЛОК-СХЕМИ

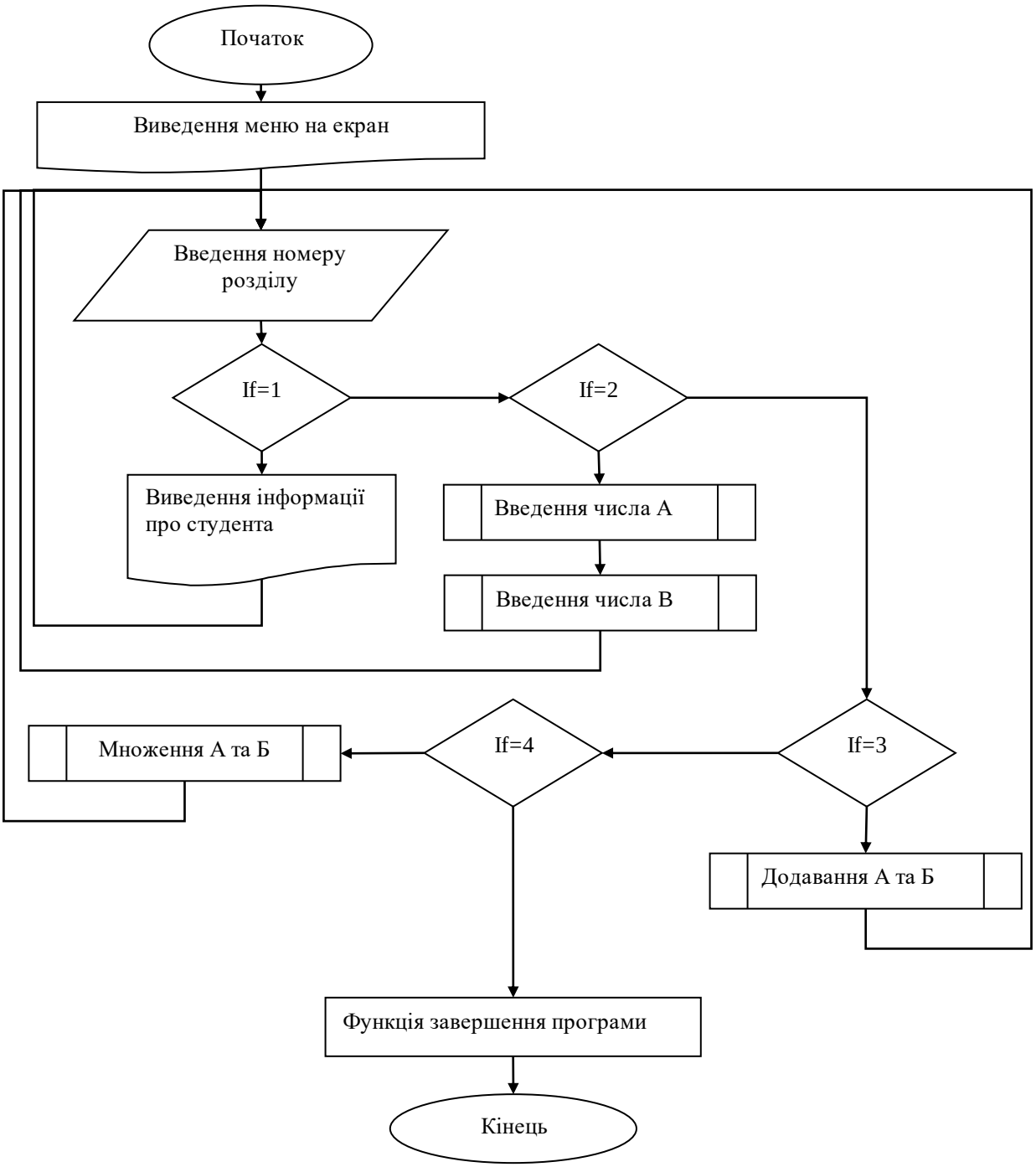
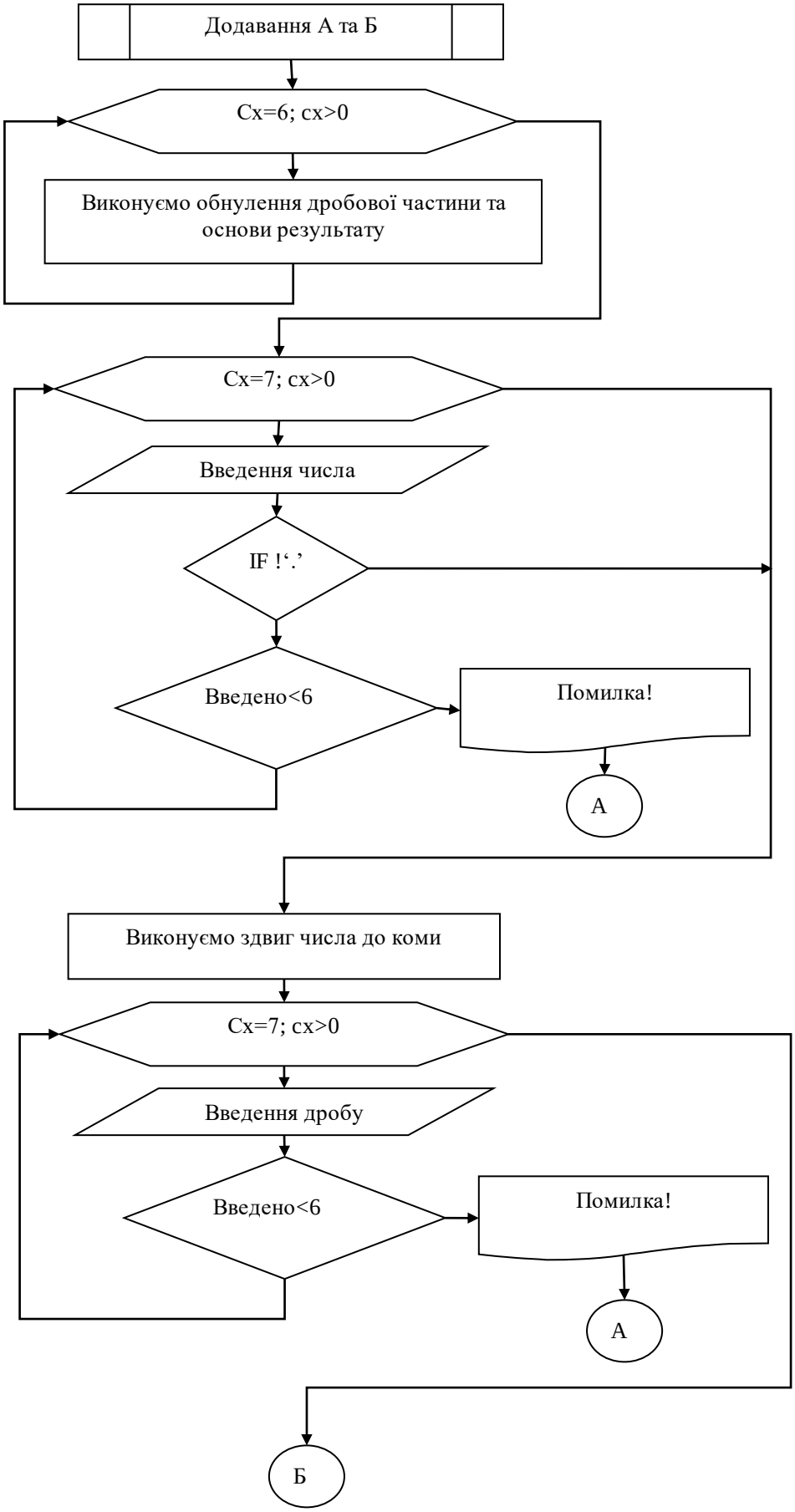


Рисунок В.1 – Блок-схема головної програми


Додавання А та Б
Cx=6; cx>0
Виконуємо обнулення дробової частини та основи результату
Cx=7; cx>0
Введення числа
IF !‘.’
Введено<6
Помилка!
A
Виконуємо здвиг числа до коми
Cx=7; cx>0
Введення дробу
Введено<6
Помилка!
A
Б

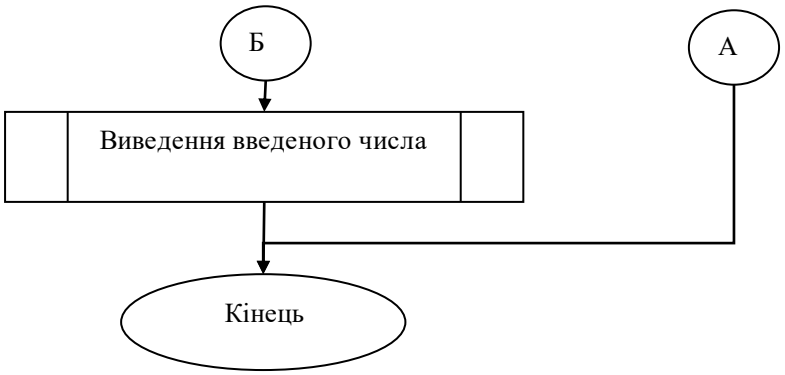


Рисунок В.2 – Підпрограма введення чисел в пам’ять

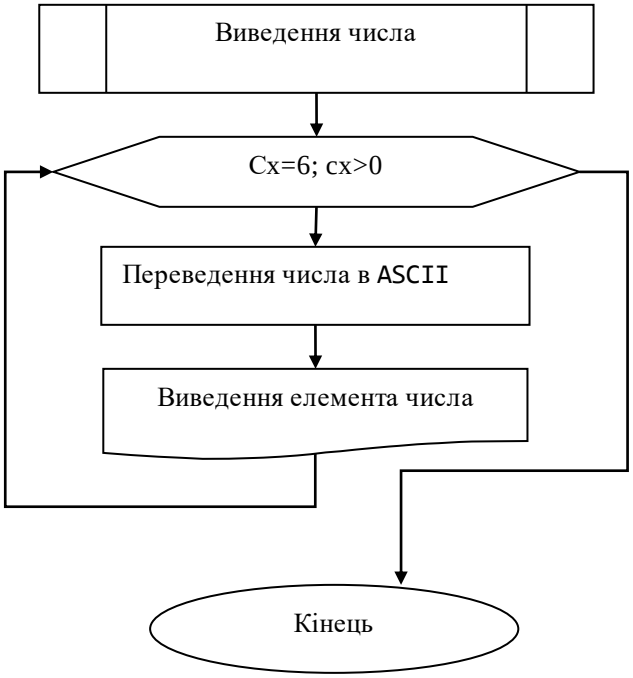


Рисунок В.3 – Підпрограма виведення неупакованого числа на екран

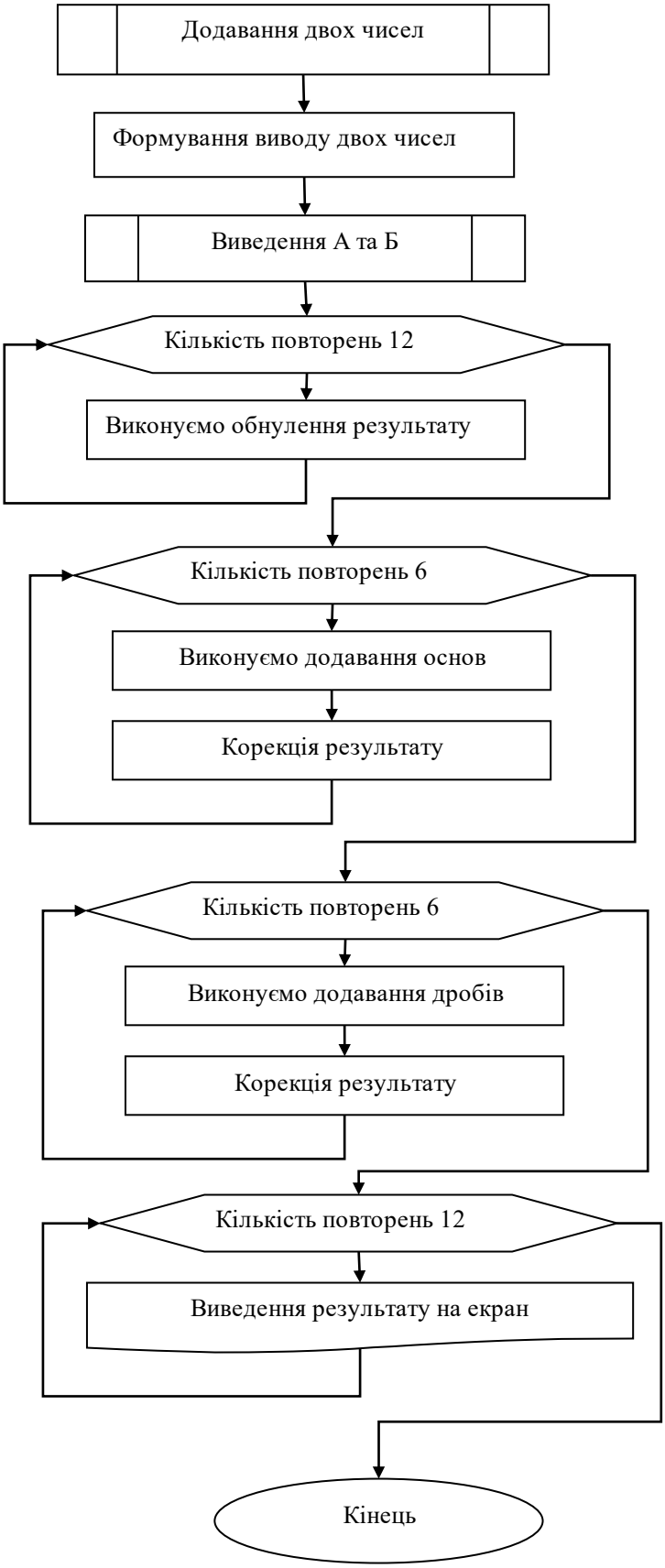


Рисунок В.4 – Підпрограма додавання двійково-десяткових чисел

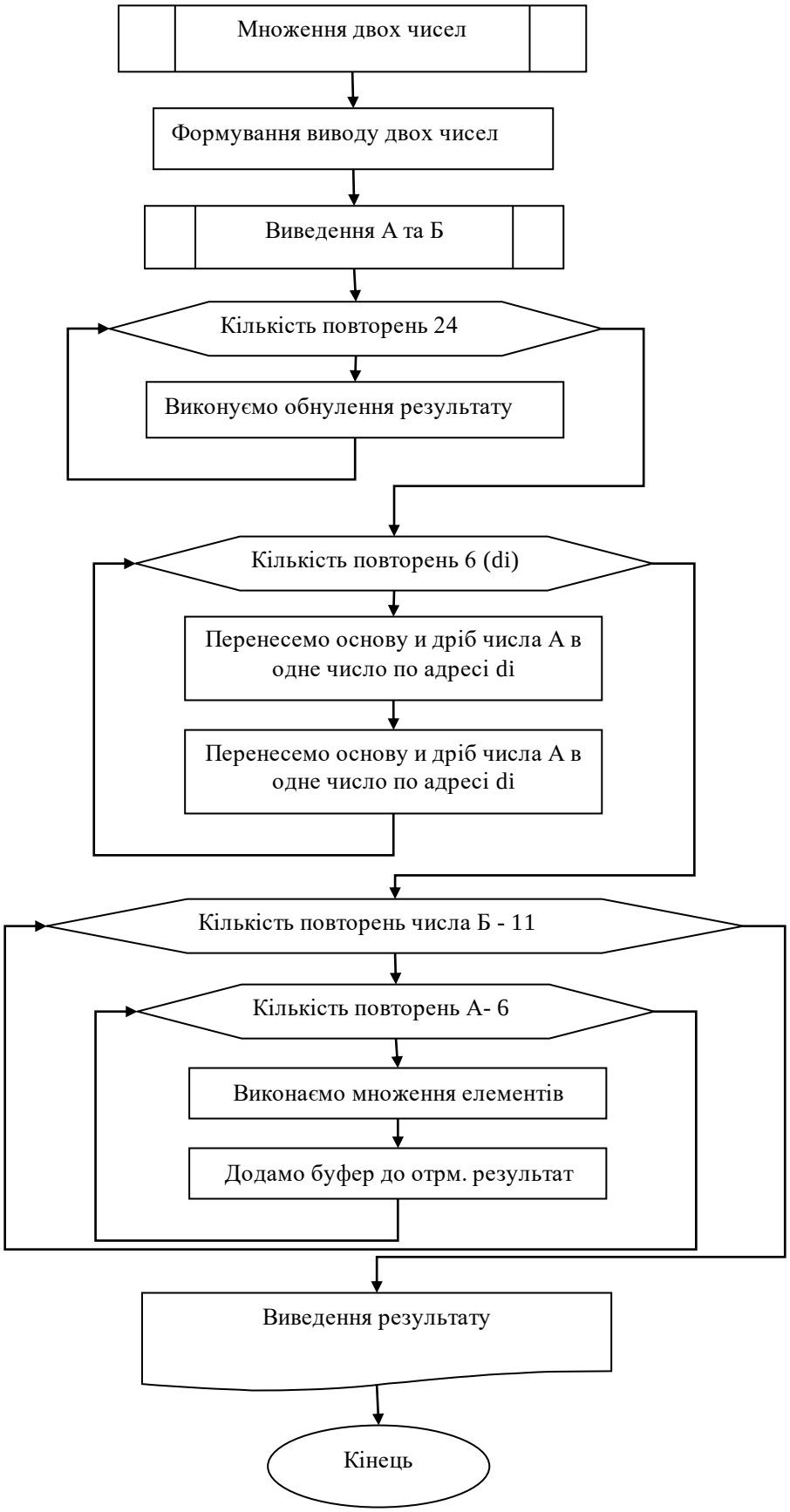


Рисунок В.5 – Підпрограма множення двійково-десяткових чисел

ДОДАТОК Г

РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ

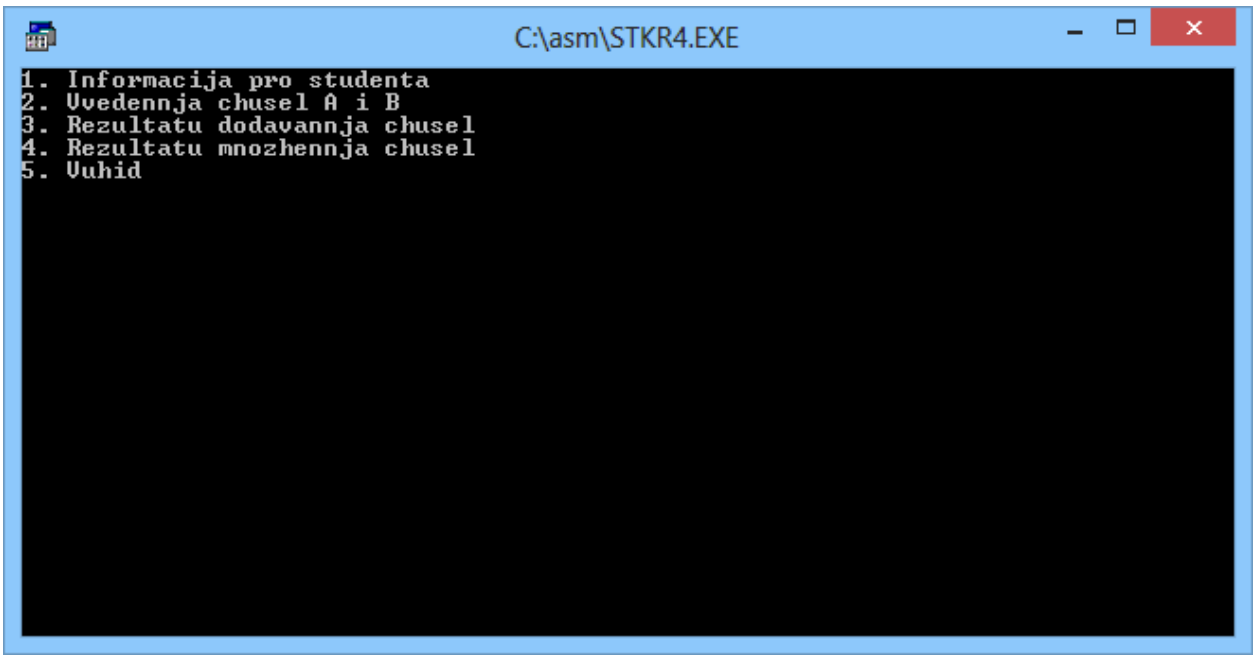


Рисунок Г.1 – Головне меню програми

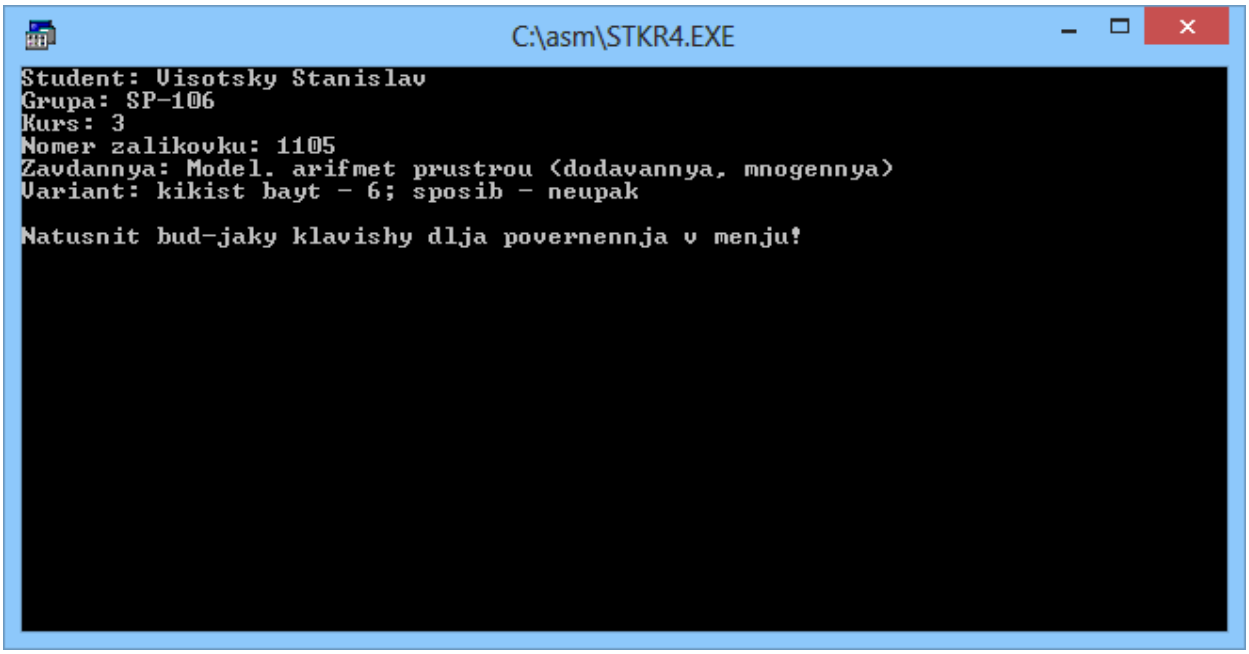


Рисунок Г.2 – Виведення інформації про студента

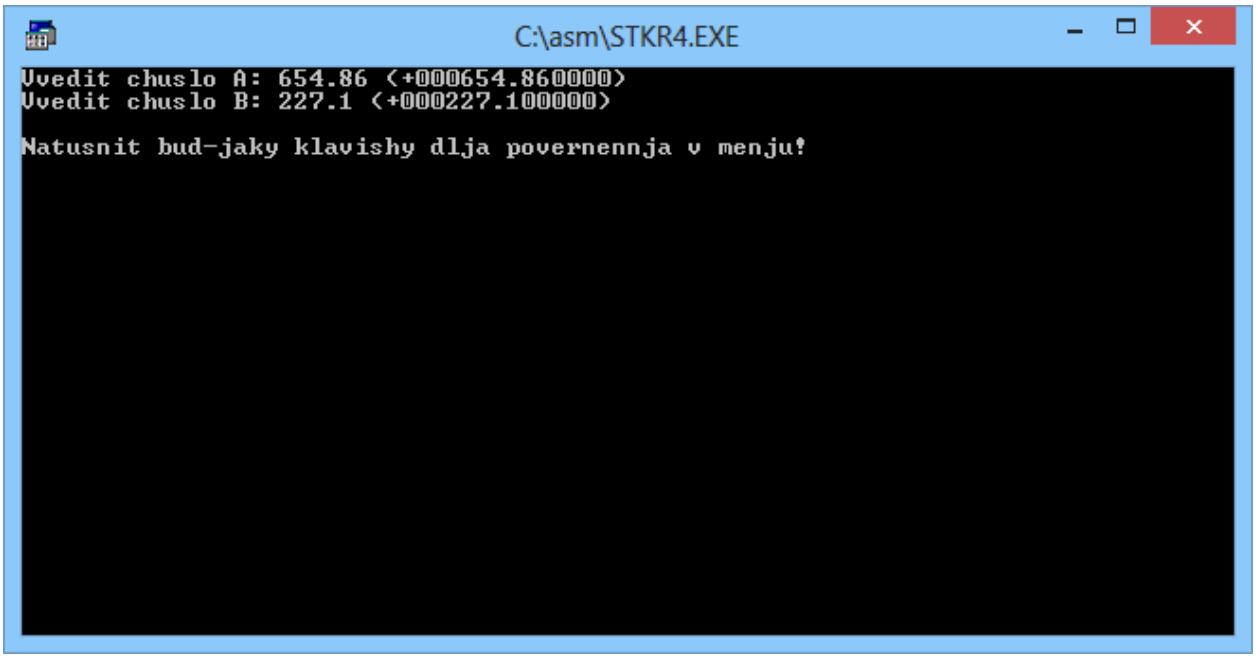


Рисунок Г.3 – Введення чисел в пам’ять

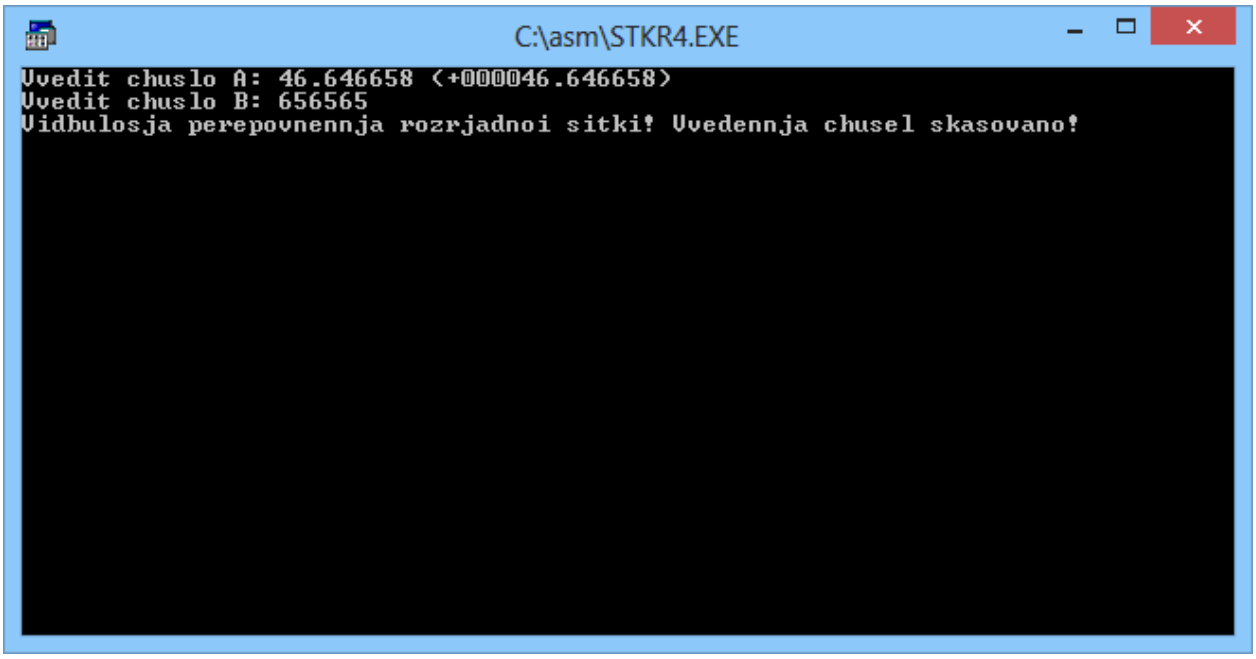


Рисунок Г.4 – Інформація при переповненні розрядної сітки

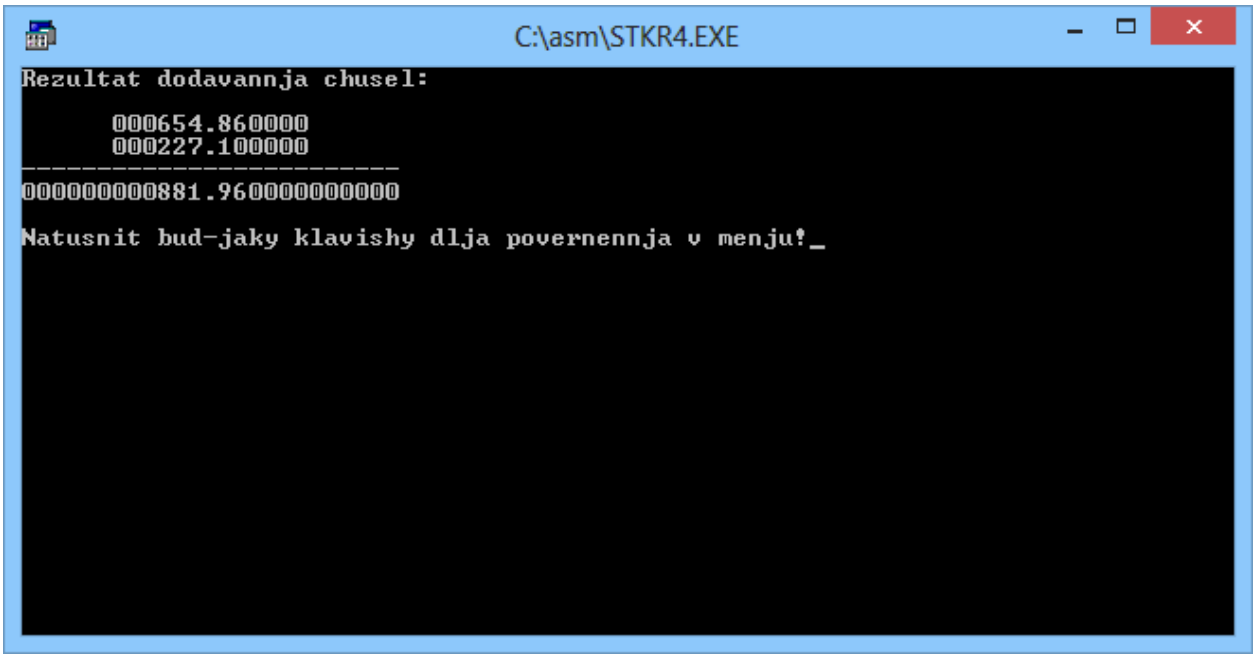


Рисунок Г.5 – Виведення результатів додавання введених чисел

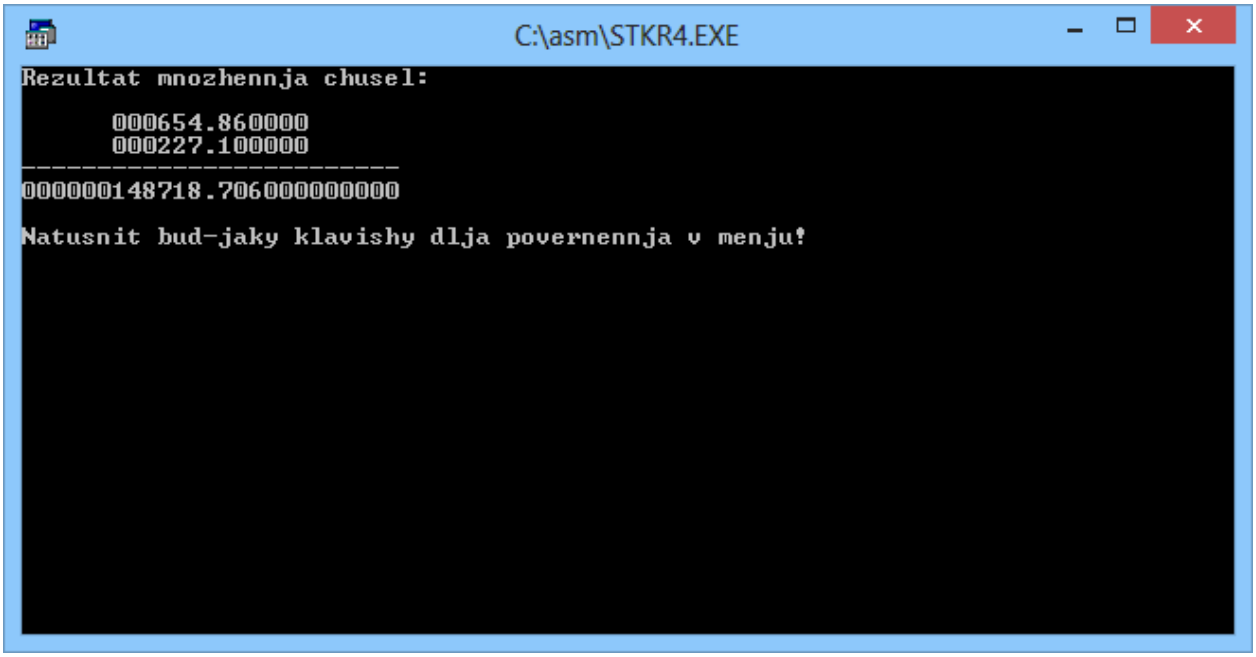


Рисунок Г.6 – Виведення результатів множення введених чисел

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rowohlt Taschenbuch Verlag, 2003. – 77 с.
2. Искусство дизассемблирования (Касперски К., Рокко Е.) БХВ-Петербург 2008. – 892с.
3. Крупник А. Ассемблер. Самоучитель. – СПб.: Питер, 2005. – 235 с.
4. Юров В. И. Assembler. Учебник для вузов. 2-е изд. / В. И. Юров – СПб.: Питер, 2003. – 637 с.
5. Фельдман С. К. Системное программирование на персональном компьютере./С. К. Фельдман. – 2е изд. – М.: Букпресс, 2006. – 512 с.
6. Пирогов В.Ю. ASSEMBLER. Учебный курс. – М.: Издатель Молгачева С.В., Издательство Нолидж, 2001. - 848 с.
7. Ружицкая, Е. А. Программирование на языке Assembler : BCD-числа, цепочечные команды: практическое пособие/Е. А. Ружицкая, Е. Ю. Кузьменкова; М-во образования Республики Беларусь, Гомельский гос. ун-т им. Ф. Скорины. – Гомель: ГГУ им. Ф. Скорины, 2017. – 47 с.