ЗМІСТ

ВСТУП

Курсовий робота - це самостійно виконана і відповідно оформлена творча робота студента з вирішення конкретного практичного завдання з однієї або декількох загально-технічних чи спеціальних дисциплін на основі набутих теоретичних знань та умінь.

Основною метою курсової роботи є закріплення, поглиблення та систематизація отриманих студентами в процесі навчання теоретичних знань з різних дисциплін та на вчитись користуватись різною літературою та документацію для дослідження та розроблення програмних засобів та документації до неї.

Під час дослідження тематики курсової роботи для побудови програмних засобів було розглянуті методи та різні мови і середовище програмування які можуть краще представити набуті з дисципліни. Найбільше увага була приділена мові низького рівня Асемблер.

Мова Асемблера - мова низького рівня. Дані, якими він оперує, це елементи пам’яті і числа, що зберігаються в їх невеликій розрядності. Програмою на Асемблері є послідовність машинних команд, записаних в символічному вигляді. Мова Асемблера будь-якого процесора суттєво складніша будь-якої високорівневої мови. Щоб скористатись всіма можливостями мови Асемблера, треба знати команди мікропроцесора, і їх кількість.

Завданням до курсової роботи є розробити автоматизовану систему для роботи з матрицями з додатковими умова та функціями для полегшення обрахунків. Дані необхідно оформити у вигляді матриць цілих чисел. В програмі повинні бути реалізовано операції над матрицями такі як порівняння, множення та додавання. Головним завданням програмного засобу щоб була присутність перевірка на не парність.

Отже наша програма повинна бути зручною у використанні і забезпечувати ряд безперебійних операцій по обробці вхідних даних і відображення вихідних. Алгоритм

програми повинен мати чітку послідовність, забезпечуючи прямий порядок дій. Інтерфейс програми має бути простим і зрозумілим простому користувачу.

1. АНАЛІЗ ВІДОМИХ РІШЕНЬ РОЗРОБКИ ПРОГРАМНИХ ЗАСОБІВ

1.1 Загальні відомості про матриці

Ма́триця — математичний об'єкт, записаний у вигляді прямокутної таблиці чисел (чи елементів кільця), він допускає операції (додавання, віднімання, множення та множення на скаляр).

Зазвичай матриці представляються двовимірними (прямокутними) таблицями. Іноді розглядають багатовимірні матриці або матриці непрямокутної форми. В цій статті вони розглядатися не будуть.

Матриці є корисними для запису даних, що залежать від двох категорій, наприклад: для коефіцієнтів систем лінійних рівнянь та лінійних перетворень.

Матриці мають довготривалу історію застосування при розв'язуванні систем лінійних рівнянь. Китайський текст «Математика в дев'яти книгах» (написаний ще до нашої ери) містить приклади використання матриць для розв'язання системи рівнянь, включаючи поняття визначника, ще задовго до введення визначників японським математиком Такакадзу Секі та німецьким математиком Лейбніцем.

Поняття «матриці», яке вже не було похідним від поняття «визначник» з'явилось тільки в 1858 році в праці англійського математика Артура Келі. Термін «матриця» першим став вживати Джеймс Джозеф Сильвестр, який розглядав матрицю, як об’єкт, що породжує сімейство мінорів (визначників менших матриць, утворених викреслюванням рядків та стовпців з початкової матриці).

Вивчення визначників відбувалось в різних галузях математики:

– Карл Фрідріх Гаус першим встановив зв’язок між квадратичними формами, лінійними відображеннями та матрицями.

– Коші розглядав визначники як многочлени та в 1829 довів, що власні значення симетричних матриць є дійсними числами.

Для обчислення системи рівнянь часто доцільно використовувати метод

Крамера. Він передбачає обчислення визначників матриць, іншого застосування визначникам важко придумати.

Означення визначника: Визначником (детермінантом) будь-якої квадратної матриці A=(ai,j) називається алгебраїчна сума всіх можливих добутків елементів матриці ai,j, взятих по одному з кожного рядка і стовпця з певним знаком. Цей знак рівний мінус одиниці (-1) в степені кількості інверсій номерів других індексів, коли перші впорядковані в порядку зростання.

Таке правило незручне для сприйняття, тому на практиці користуються простими формулами. Визначник другого порядку рівний різницю добутків елементів головної та бічної діагоналі:

$$\det A = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}$$

Визначник третього порядку знаходять за правилом трикутників:

$$\Delta = \det A = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} =$$

$$= a_{11} \cdot a_{22} \cdot a_{33} + a_{12} \cdot a_{23} \cdot a_{31} + a_{13} \cdot a_{21} \cdot a_{32} - a_{13} \cdot a_{22} \cdot a_{31} - a_{12} \cdot a_{21} \cdot a_{33} - a_{11} \cdot a_{23} \cdot a_{32}$$

Визначник 3 порядку має 6 доданків з трьох множників у кожному. Формула для визначника четвертого порядку ще складніша і містить 4!=4*3*2=24 доданки, визначник п'ятого порядку утворюється сумуванням 5!=120 доданків, кожен з яких є добутком 5 елементів матриці, взятих відповідно до означення.

Але на практиці визначники 4, 5 порядку теж не обчислюють за загальними формулами, а розкладають через мінори або спрощують, застосовуючи елементарні перетворення над визначниками.

Останні базуються на властивостях визначників, які продемонструємо на прикладах.

Властивості визначників матриць

1. Визначник залишається незмінним при його транспонуванні (тобто заміна рядків на стовпці і навпаки не змінює результуюче значення). Доведення продемонструємо на прикладі визначника 2 порядку :

$$A=\begin{pmatrix}2 & 1\\ 4 & 3\end{pmatrix}, A^T=\begin{pmatrix}2 & 4\\ 1 & 3\end{pmatrix}.$$

$$\det A=\begin{vmatrix}2 & 1\\ 4 & 3\end{vmatrix}=2\cdot 3-1\cdot 4=6-4=2;$$

$$\det A^T=\begin{vmatrix}2 & 4\\ 1 & 3\end{vmatrix}=2\cdot 3-4\cdot 1=6-4=2;$$

$$\det A^T \equiv \det A$$

2. Якщо всі елементи деякого рядка чи стовпця визначника дорівнюють нулю, то і сам визначник дорівнює нулю.

Це доведемо без наведення прикладів – справа в тому, що кожен доданок у визначнику має елементи з кожного рядка і стовпця. Звідси слідує, що у кожній сумі буде множник рівний нулю, і відповідно сам визначник теж.

3. Якщо у визначнику замінити місцями два сусідні рядки чи стовпці, то визначник змінить знак на протилежний. Важливим тут є слово сусідні, оскільки якщо змінити 1 і 2 рядки то матимемо –det(A), якщо 1 і 3 то знак зміниться двічі, а два рази – (–) рівний (+), матимемо det(A).

1.2. Вибір методу та програмних засобів

В даній курсовій роботі було вибрано для перевірки елементів матриці метод не парного порівняння. Так, як він є одним з найперспективніших методів, що дозволяє отримувати дані з матриці і працювати з ними дальше.

Запропонована методика є похідною від методу порівняльного аналізу. Заснована вона на відомому в матричної алгебри методі попарних порівнянь. Як і будь-який продукт, вона має недоліки і достоїнства. До недоліків можна віднести її відносну трудомісткість, до достоїнств - некритичність до суворого добору аналогів, тому що дозволяє отримати досить точні результати навіть при відсутності близьких за своїми характеристиками аналогів оцінюваного об'єкта. Для полегшення розрахунків застосовується комп'ютерна версія даної методики.

Написання програми на асемблері - вкрай важкий і витратний процес. Щоб створити ефективний алгоритм, необхідно глибоке розуміння роботи ЕОМ, знання деталей команд, а також підвищена увага і акуратність. Ефективність - це критичний параметр для програмування асемблер (див. рис. 1.1).

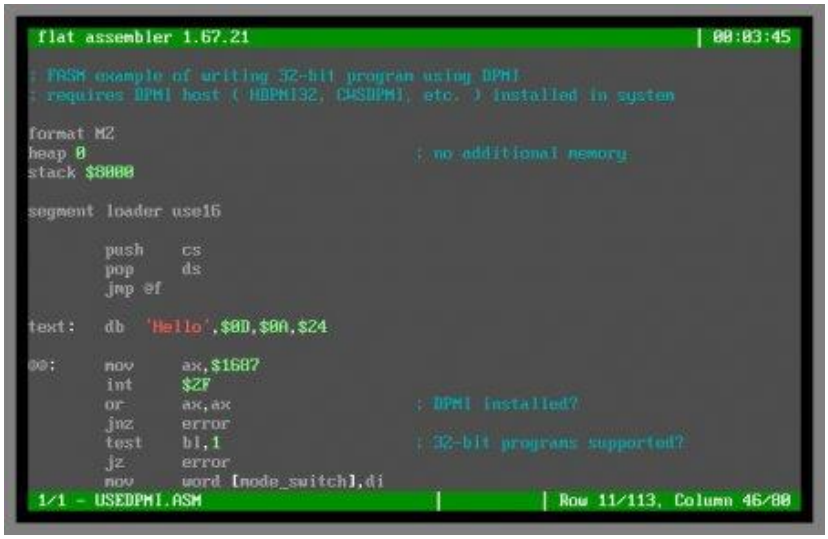

Рис. 1.1 – Вікно середовища асемблер

Головна перевага мови асемблер в тому, що він дозволяє створювати короткі і швидкі програми. Тому використовується, як правило, для вирішення вузькоспеціалізованих завдань. Необхідний код, який працює ефективно з апаратними компонентами, чи потрібна програма, вимоглива до пам'яті або часу виконання.є

1.2.1 Регістри

Регістрами в мові асемблер називають комірки пам'яті, розташовані безпосередньо на кристалі з АЛУ (процесор). Особливістю цього типу пам'яті є швидкість обігу до неї, яка значно швидше оперативної пам'яті ЕОМ. Вона також називається надшвидкої оперативною пам'яттю (СОЗУ або SRAM).

Існують наступні види регістрів:

– Регістри загального призначення (РОН).
– Прапори.
– Покажчик команд.
– Регістри сегментів (див. рис. 1.2).

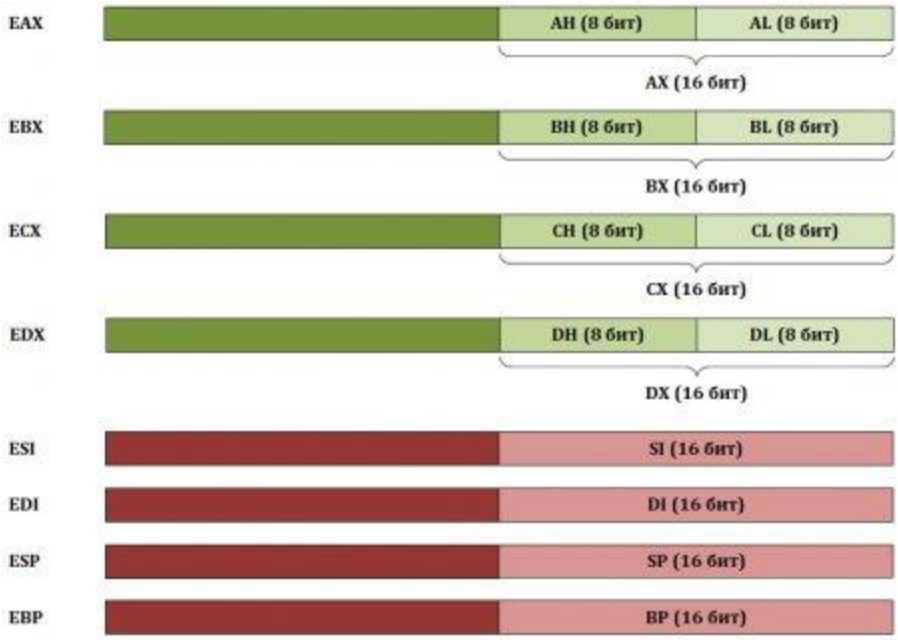

Рис 1.2. – Види регістрів

Є 8 регістрів загального призначення, кожен розміром в 32 біта. Доступ до регістрів EAX, ECX, EDX, EBX може здійснюватися в 32-бітовому режимі, 16-бітовому - AX, BX, CX, DX, а також 8-бітовому - AH і AL, BH і BL і т. д. Буква "E" у назвах регістрів означає Extended (розширений). Самі імена пов'язані з їх назвами англійською:

- Accumulator register (AX) - для арифметичних операцій.
- Counter register (CX) - для зрушень і циклів.
- Data register (DX) - для арифметичних операцій і операцій вводу/виводу.
- Base register (BX) - для покажчика на дані.
- Stack Pointer register (SP) - для покажчик вершини стека.
- Stack Base Pointer register (BP) - для індикатора підстави стека.
- Source Index register (SI) - для покажчика відправника (джерела).
- Destination Index register (DI) - для одержувача.

Спеціалізація РОН мови асемблер є умовною. Їх можна використовувати в будь-яких операціях. Однак деякі команди здатні застосовувати тільки певні регістри. Наприклад, команди циклу використовують ESX для зберігання значення лічильника. Регістр прапорів. Під цим мається на увазі байт, який може приймати значення 0 і 1. Сукупність всіх прапорів (їх близько 30) показують стан процесора. Приклади прапорів: Carry Flag (CF) - Прапор переносу, Overflow Flag (OF) - переповнення, Nested Flag (NT) - прапор вкладеності завдань та багато інших. Прапори поділяються на 3 групи: стан, управління і системні (див. рис. 1.3).

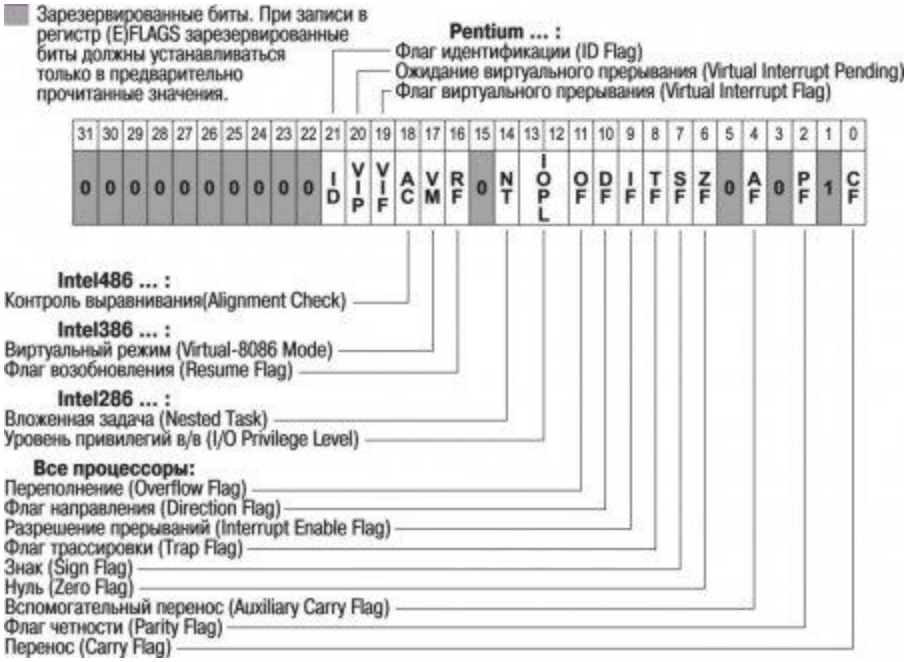

Рис. 1.3 – Призначення регістрів

Покажчик команд (EIP - Instruction Pointer). Цей регістр містить адресу інструкції, яка повинна бути виконана наступній, якщо немає інших умов. Регістри сегментів (CS, DS, SS, ES, FS, GS). Їх наявність в асемблері продиктовано особливим управлінням оперативною пам'яттю, щоб збільшити її використання в програмах. Завдяки ним можна було керувати пам'яттю розміром до 4 Гб. В архітектурі Win32 необхідність в сегментах відпала, але назви регістрів збереглися і використовуються по-іншому.

1.2.2 Стек

Це область пам'яті, що виділена для роботи процедур. Особливість стека полягає в тому, що останні дані, записані в нього, доступні для читання першими. Або іншими словами: перші записи стека витягуються останніми. Уявити собі цей процес можна як вежі з шашок. Щоб дістати шашку (нижню шашку в основу вежі або будь-яку в середині) потрібно спочатку зняти все, що лежать зверху. І, відповідно, остання покладена на вежу шашка, при розборі вежі знімається першої. Такий принцип

організації пам'яті і роботи з нею продиктований її економією. Стек постійно очищається і в кожен момент часу одна процедура використовує його (див. рис. 1.4).

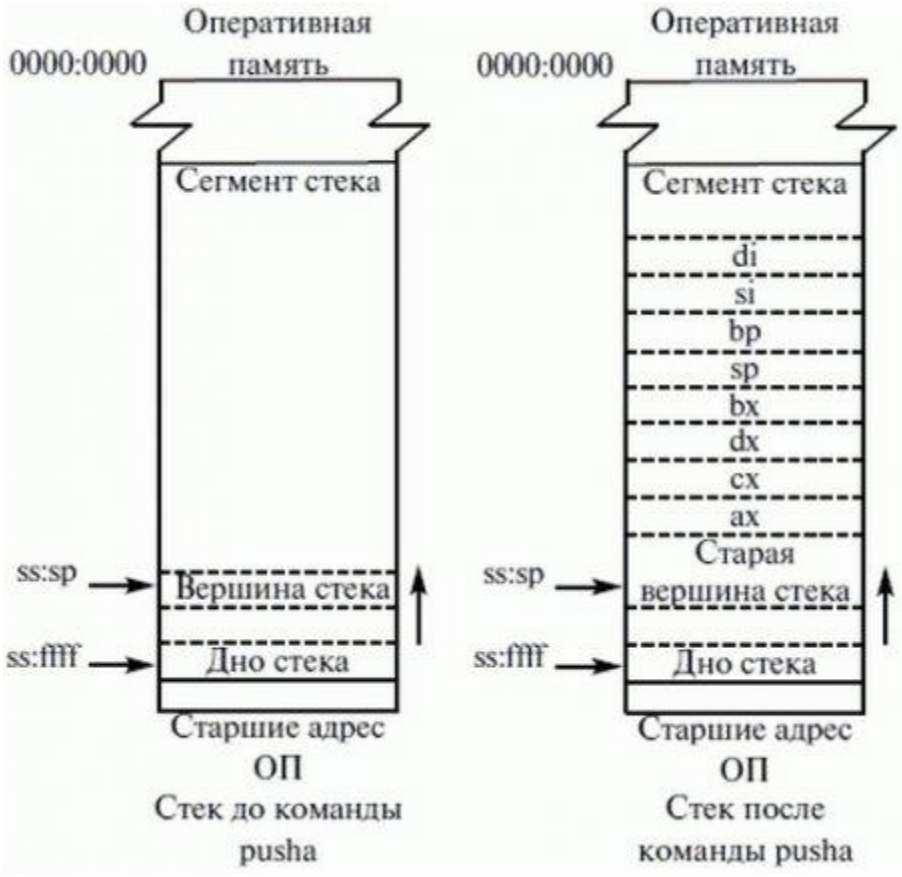

Рис. 1.4 – Виділення стеку пам’яті для регістрів

1.3. Структура розробки програмного засобу

Програма на асемблері являє собою сукупність блоків пам'яті, званих *сегментами пам'яті* . Програма може складатися з одного або декількох таких блоків-сегментів. Кожен сегмент містить сукупність речень мови, кожне з яких займає окремий рядок коду програми.

Речення асемблера бувають чотирьох типів:

- *команди або інструкції* , що представляють собою символічні аналоги машинних команд.

У процесі трансляції інструкції асемблера перетворяться у відповідні команди системи команд мікропроцесора;

- *макрокоманди* - оформляються певним чином пропозиції тексту програми, що заміщаються під час трансляції іншими пропозиціями;
- *директиви* , які є зазначенням транслятору асемблера на виконання деяких дій. У директив немає аналогів в машинному поданні;
- *рядки коментарів* , що містять будь-які символи, в тому числі і літери російського алфавіту. Коментарі ігноруються транслятором.

1.3.1 Синтаксис асемблера

Пропозиції, складові програму, можуть являти собою синтаксичну конструкцію, що відповідає команді, макрокоманді, директиві або коментарю. Для того щоб транслятор асемблера міг розпізнати їх, вони повинні формуватися за певними синтаксичними правилами. Для цього найкраще використовувати формальний опис синтаксису мови на зразок правил граматики. Найбільш поширені способи подібного опису мови програмування - *синтаксичні діаграми* і *розширені форми Бекуса-Наура* . Для практичного використання більш зручні *синтаксичні діаграми* . Наприклад, синтаксис пропозицій асемблера можна описати за допомогою синтаксичних діаграм, показаних на наступних малюнках (див. рис. 1.5-1.7).

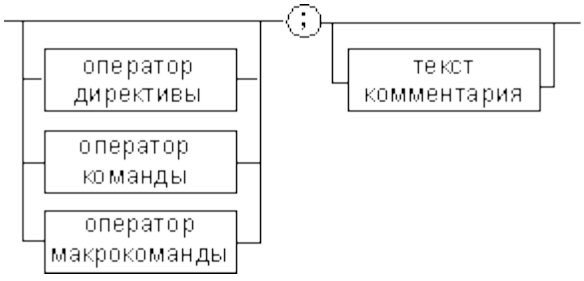

Рис. 1.5 – Формат пропозиції ассемблера

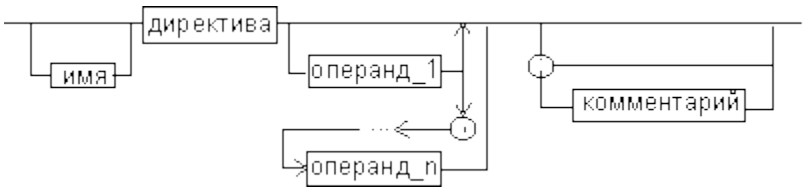

Рис 1.6 –Формат директив

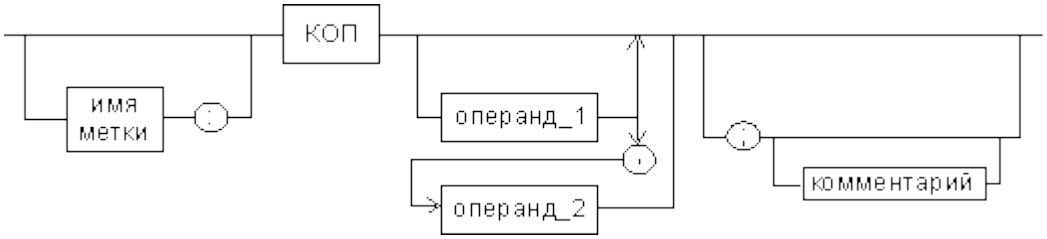

Рис. 1.7 – Формат команд і макрокоманд

На цих рисунках зображено:

- *ім'я мітки* - ідентифікатор, значенням якого є адреса першого байта того пропозиції вихідного тексту програми, яке він позначає;
- *ім'я* - ідентифікатор, що відрізняє дану директиву від інших однойменних директив. В результаті обробки асемблером певної директиви цього імені можуть бути присвоєні певні характеристики;
- *код операції (КОП) і директива* - це мнемонічні позначення відповідної машинної команди, макрокоманди або директиви транслятора;
- *операнди* - частини команди, макрокоманди або директиви асемблера, що позначають об'єкти, над якими виробляються дії. Операнди асемблера описуються виразами з числовими і текстовими константами, мітками і ідентифікаторами змінних з використанням знаків операцій і деяких зарезервованих слів.

1.3.2. Лексеми

- *ідентифікатори* - послідовності припустимих символів, що використовуються для позначення таких об'єктів програми, як коди операцій, імена змінних і назви міток. Правило запису ідентифікаторів полягає в наступному: ідентифікатор може складатися з одного або декількох символів. В якості символів

можна використовувати букви латинського алфавіту, цифри і деякі спеціальні знаки - _,?, $, @. Ідентифікатор не може починатися символом цифри. Довжина ідентифікатора може бути до 255 символів, хоча транслятор сприймає лише перші 32, а інші ігнорує. Регулювати довжину можливих ідентифікаторів можна з використанням опції командного рядка **mv** . Крім цього існує можливість вказати транслятору на те, щоб він розрізняв великі та малі літери або ігнорував їх відмінність (що і робиться за замовчуванням). Для цього застосовуються опції командного рядка **/ mu, / ml, / mx** ;

- *ланцюжка символів* - послідовності символів, укладені в одинарні або подвійні лапки;
- *цілі числа* в одній з наступних систем числення: *двійковій, десятковій, шістнадцятковій* . Ототожнення чисел при записі їх в програмах на асемблері виробляється за певними правилами:
  - **Десяткові числа** не вимагають для свого ототожнення вказівки будь-яких додаткових символів, наприклад 25 або 139.
  - Для ототожнення у вихідному тексті програми **двійкових чисел** необхідно після запису нулів і одиниць, що входять до їх складу, поставити латинське " **b** ", наприклад 10010101 **b** .
  - **Шістнадцятиричні числа** мають більше умовностей при своєму записі:
    - *По-перше* , вони складаються з цифр **0 ... 9** , малих і великих літер латинського алфавіту **a** , **b** , **c** , **d** , **e** , **f** або **A** , **B** , **C** , **D** , **E** , **F** .
    - *По-друге* , у транслятора можуть виникнути труднощі з розпізнаванням шістнадцяткових чисел через те, що вони можуть складатися як з одних цифр 0 ... 9 (наприклад 190845), так і починатися з літери латинського алфавіту (наприклад **ef15**). Для того щоб "пояснити" транслятору, що дана лексема не є десятковим числом або ідентифікатором, програміст повинен спеціальним чином виділяти шістнадцяткове число. Для цього на кінці послідовності шістнадцяткових цифр, складових шістнадцяткове число, записують латинську букву " **h** ". Це

обов'язкова умова. Якщо шістнадцяткове число починається з букви, то перед ним записується провідний нуль: **0** ef15 **h** .

2. ХАРАКТЕРИСТИКА ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Загальне представлення

Для налаштування програмного засобу використовуються регістри загального призначення. Регістрова структура процесора включає в себе 14 16-розрядних програмно-доступних регістрів і може бути представлена в наступному вигляді (див. рис.2.1).

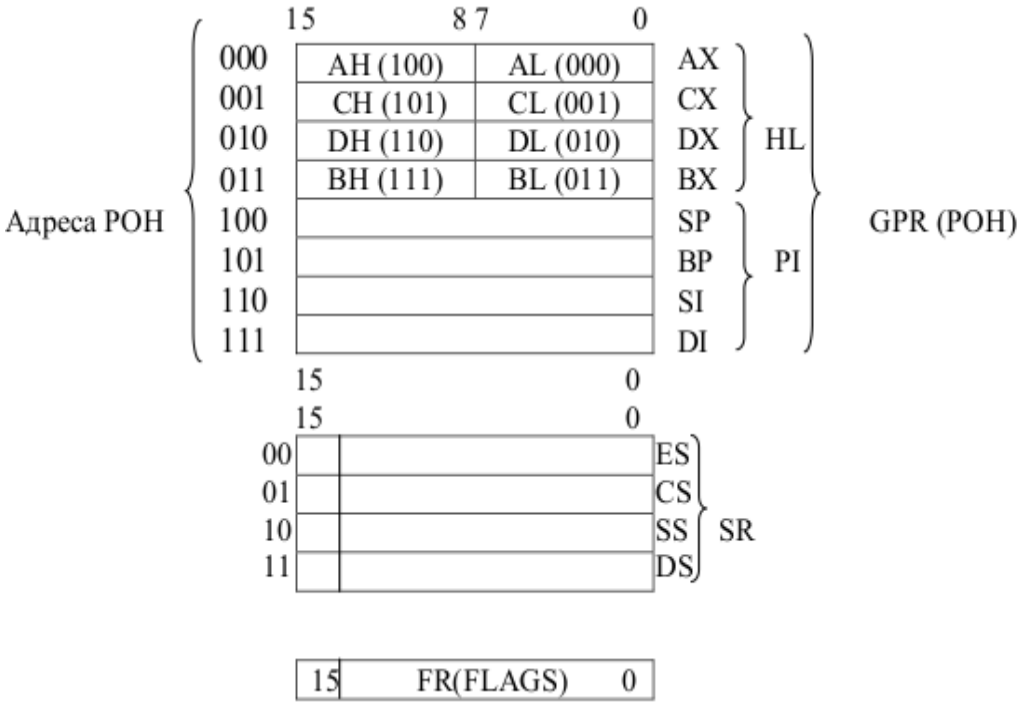

Рис.2.1. – Регістрова структура процесора

Програмно-доступні регістри поділяються на наступні:

– регістри загального призначення (GPR - General Purpose Registers), група включає вісім регістрів;

– сегментні регістри (SR - Segment Registers), група включає чотири регістри;

– регістр прапорів (Flags);

– вказівник команд (Instruction Pointer).

2.2 Регістри загального призначення

У відношенні функціонального призначення регістрів, що утворюють внутрішню регістрову пам'ять процесорів, існують два протилежні підходи, що реалізуються в архітектурі ЕОМ:

– повна спеціалізація регістрів, тобто кожен регістр використовується тільки за одним конкретном призначенням;

– повна універсалізація регістрів, тобто кожен регістр може використовуватися за будь-яким призначенням.

У процесорах фірми Intel використовується проміжний підхід, що поєднує в собі часткову спеціалізацію і часткову універсалізацію регістрів. Це означає, що за замовчуванням будь-кий регістр використовується як спеціалізований для певної мети, і в той же час його можна використовувати і для інших цілей як універсальний регістр.

Використання будь-якого регістра за його прямим призначенням скорочує довжину об'єктного коду програми, так як використання регістра за призначенням, як правило, передбачає його неявну адресацію (адреса регістра не задається, але мається на увазі).

Функціональна спеціалізація регістрів загального призначення відбивається в їх найменуваннях:

- AX - accumulator (регістр-акумулятор) - за замовчуванням використовується для завдання одного з операндів команди і для представлення результату;

- CX - counter (лічильник) - за замовчуванням використовується, по-перше, як лічильник числа повтореннь циклів в команді "організація циклу" (LOOP); подруге, для завдання кількості розрядів для команд здвигів (його молодший байт - CL), по-третє, для завдання числа елементів оброблюваних рядків в командах обробки рядків (MOVS, CMPS і т. д.);

- DX - Data (регістр даних) - за замовчуванням використовується як розширення акумулятора з боку старших розрядів у командах множення і ділення;

- BX - Base (базовий регістр) - за замовчуванням використовується як базова компонента ефективної адреси операнда, що знаходится у пам'яті (у термінології фірми Intel під ефективним адресою - Effective Address (EA) - розуміється адреса операнда, формований програмою).

Для отримання фізичної адреси комірки пам'яті, в якій знаходиться операнд, здійснюється перетворення EA на основі найпростішої моделі сегментованої пам'яті (механізмусегментації);

- SP - Stack Pointer (покажчик стека) - за замовчуванням використовується для адресації вершини стека. Вершина стека вказує на адресу останнього елемента, записаного в стек.

Стек являє собою сегмент пам'яті. Стек зростає в область молодших адрес. Це означає, що при записі (включенні в стек), наприклад, по команде PUSH, спочатку здійснює декремент SP на два, а потім запис в пам'ять за новим значенням SP як адреси. При читанні зі стека, наприклад, по команді POP, спочатку проводиться читання слова за адресою з SP, а потім інкремент вмісту SP на два. Робота зі стеком реалізується на рівні слів, але не байтів;

- BP - Base Pointer (покажчик бази) - за замовчуванням використовується як базова компонента ефективного адреси операнда в пам'яті за аналогією з BX.

Відмінність у використанні вмісту регістрів BX і BP як базових компонент EA полягає в тому, що при використанні BX мається на увазі звернення до сегменту даних, а при використанні BP - до сегменту стека (але не через його вершину).

Подібний спосіб роботи зі стеком не через його вершину використовується в програмах на ASSEMBLER для звернення до параметрів процедури, переданим через стек;

- SI - Source Index (індекс джерела) - за замовчуванням використовується для завдання індексної компоненти EA, а також для адресації елементів строк в командах обробки рядків;

- DI - Distination Index (індекс приймача) - за замовчуванням використовується

аналогічно SI для завдання індексної компоненти EA, а також для адресації елементів рядка-приймача в командах обробки рядків.

Групу з восьми РОН прийнято ділити на дві частини:

- група HL (High - Low);

- група PI (Pointer - Index).

Групу HL іноді називають регістрами даних, маючи на увазі її переважне використання для операндів і результатів команд.

Регістри групи HL можуть використовуватися в командах в двухбайтному (повному) та байтному (неповному) варіантах.

Окремі байти цих регістрів використовують ту ж назву, що і повний регістр (A, C, D, B) з додаванням приставки L - молодший, H - старший, для відповідних байтів регістру.

Група PI або група покажчиків-індексів може використовуватися тільки в двухбайтному варіанті.

Для адресації РЗП, як повних, так і не повних, в машинній команді використовуються три двійкових розряди.

Двійкові адреси повних регістрів загального призначення наведені на рисунку 2.1 зліва, а їх окремих байт - в дужках команд, вказує на черговий байт команди, вибраний з пам'яті і поміщається в спеціальний буфер, який називається чергою команд (IQ - Instruction Queue). Не дивлячись на цей факт, для виконуваної програми IP містить адресу наступної команди. Фактично, на апаратному рівні при необхідності використання IP (наприклад, для його збереження як адреси повернення) здійснюється відповідна корекція його вмісту з урахуванням числа байт, обраних в IQ.

Конвеєр команд служить одним з найважливіших засобів збільшення продуктивності комп'ютера. З його допомогою реалізується паралелізм на рівні машинних команд. Це означає, що в будь-який момент часу в процесорі на стадії одночасного виконання знаходиться декілька послідовних машинних команд (по зростанню адрес). Блоки конвеєра можуть функціонувати паралельно в часі незалежно

один від одного. При використанні класичного шестиступінчастого конвеєра команд (по числу фаз виконання команди) та умови, що кожна фаза вимагає однаковий час реалізації, повне завантаження конвеєра команд в принципі забезпечує шестиразове збільшення продуктивності в порівнянні з послідовним (бесконвейерним) процесором.

2.3 Алгоритм функціонування програмного засобу

Програма повинна виконувати наступні функції за алгоритмами, наведеними нижче:

– Заповнення комірок матриці.

– Пошук парних та не парних елементів матриці.

– Можливість обрахунку комірок матриці (додавання, множення).

– Можливість збереження даних з комірок.

– Для запуску часто використовуваних програм вибирати відповідні команди з головного меню або натискаючи кнопки на панелі інструментів.

Програма повинна також надавати можливість користувачу працювати як з мишею, так і з клавіатурою.

2.4 Обґрунтований вибір апаратно-технічних засобів, операційної системи і мови програмування

Згідно завдання на курсову роботу програма повинна забезпечувати підтримку графічного інтерфейсу та працювати з ОС Windows 8/10. Тому в якості операційної системи, в якій буде функціонувати розроблюваний програмний продукт, необхідно обрати Windows 8/10. Так як файловий менеджер не є програмою, вибагливою до ресурсів, то в якості мінімальної конфігурації ПК для роботи розроблюваного програмного продукту пропонується обрати таку конфігурацію, в якій ОС Windows

8/10 працюють без помітних часових затримок, а саме:

- ОП 1 Гб;
- тактова частота процесору – 1,5 Ггц;
- Наявність вільного дискового простору 500 Мб.

Для створення програмного продукту обрана мова програмування у складі середовища програмування який підтримує мову програмування Асемблер. Основною причиною обрання саме цього середовища програмування було того що воно могло показати набуті навички на практиці і було простим у користуванні.

3. РОЗРОБКА АЛГОРИТМУ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАСОБУ

3.1 Опис використання програмного засобу

Програма починається з визначення необхідних даних. Змінні tasm використовуються для представлення зручного для читання виводу на екран.

Змінні mkt використовуються для зручного виведення змінних циклу. Змінні tpax, tpbx, tpcx, tpdx використовуються для збереження значень відповідних регістрів в програмі. Програма містить процедуру printdec, що виводить в десятковому вигляді значення регістра ax зі знаком, оскільки в циклічній частині програми в з'являються негативні числа. Головна програма розділена на ділянки, що відповідають пунктам завдання.

У другій частині програми, де необхідно реалізувати циклічну структуру, порівняння cx. з п`ятьома відбувається в циклі після виведення значень регістрів і відновлення їх значень. Якщо cx становить значення менш п`яти, цикл завершується. Інакше до регістра bx додається значення регістра ax і відіймається значення регістру cx, а регістр cx зменшується на п`ять. Після цих маніпуляцій регістри зберігаються і цикл продовжується. Після завершення циклу організовується затримка результату на екрані за допомогою функції 1 переривання MS-DOS 21h.

Виведення на екран здійснено із застосуванням переривання MS-DOS 21h і функції переривання 9. Ця функція по перериванню виводить блок ASCII-тексту на екран, поки у вихідних даних не зустрінеться символ "$", що позначає в даному випадку кінець рядка.

Виведення символів в функції виведення числа printdec реалізований за допомогою функції 2 переривання MS-DOS 21h.Результат виконання програми наведений на рисунку 3.1.

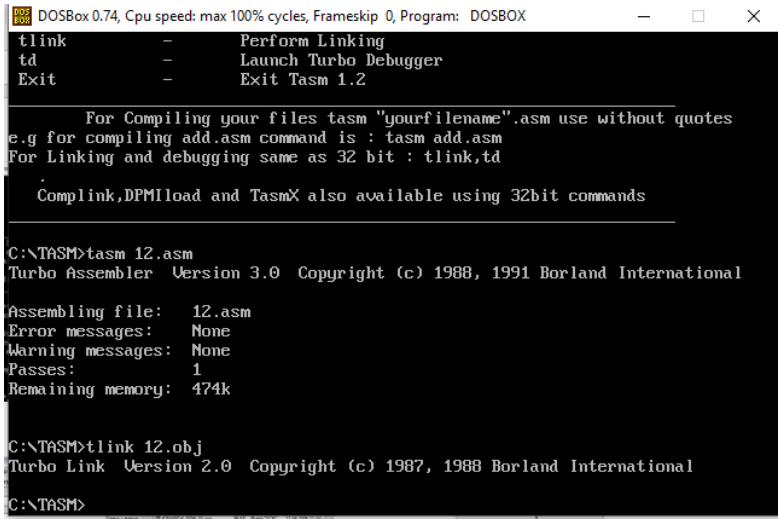

Рис 1.1 – Головне вікно програми в середовищі асемблер

3.2 Опис головного алгоритму програми

На початку роботи задаються усі параметри роботи та змінні, необхідні для роботи системи. Далі відбувається виклик підпрограми, що слугує для виведення початкового привітання у вікно терміналу. Після завершення виконання даної підпрограми відбувається виклик наступної команди, яка запускає процедуру зчитування керуючого символу. Після зчитування відбувається перевірка на відповідність отриманих даних певним заданим значенням. Спочатку отриманий символ порівнюється із символом «C» і якщо вони співпадають, то відбувається виклик підпрограми очистки дисплею. У іншому випадку відбувається порівняння отриманого символу із символом «N» та у випадку їх не співпадіння відбувається виведення у вікно терміналу повідомлення про помилку. Якщо ж символи співпали, то відбувається виклик підпрограми ініціалізації дисплею, після завершення якої викликається інша підпрограма, що зчитує символ із послідовного порту та виводить його на LCD-дисплей. Коли у кожному з рядків буде виведено по 16 символів відбудеться перехід на початок програми та її повторне виконання.

3.3 Опис програмних функцій та модулів

3.3.1 Ініціалізація змінних програми

Дана процедура реалізована наступними командами:

```
STACK        equ R6
counter      equ R5
ADDR_LINE_1  equ 33h

mov 77h,#0
RS           bit P3.2; 0 - Команда. 1 - дані
E            bit P3.3 ; строб
LCD_buff     equ 21h  ; дані для запису
MOV SCON, #050H  ;8 бітний UART режим 1
MOV TMOD, #020H ;режим автоперезагрузки таймера
MOV TH1, #0F3H ;автозагружаємі значення для отримання швидкості 2400 бод на частоті МК 12 МГц
SETB TCON.6  ;пуск таймера (TR1)
```

Дана процедура застосовується на початку програми для задання необхідних режимів роботи та оголошення, необхідних для подальшої робти програми, змінних.

3.3.2 Підпрограма заповнення матриці

Дана підпрограма призначена для початкової ініціалізації програми та роботи з головною функцією програми заповнення матриці та пошуку в неї парних та не праних значень для обрахунку (див. лістинг 3.1).

Лістинг 3.1:

```
unparity:
    test ax,ax
    inc si
    jnp kvadrat
    jp par
```

Після перевірки комірок кожне значення яке було знайдено потрібно записати у регітр щоб його запам’ятати, тобто створення масиву з комірками в пам’яті (див. лістинг 3.2).

Лістинг 3.2:

```
kvadrat:
    dec si
    mov ax,[matrix+si]
    add summa,ax
    mov bx,[matrix+si]
    mul bx
    mov [matrix+si],ax
    inc kv_amount
```

Коли в стеку пам’яті збережені необхідні комірки з цифрами потрібно очистити регістри для подальших дій при яких бде використовуватись регістри (див. лістинг 3.3).

Лістинг 3.3:

```
clrscr:
    mov ax,0600h
    mov bh,07
    mov cx,0000
```

```
    mov dx,184fh
    int 10h
    mov ah,09h
    lea dx,message4
    int 21h
```

Для виведення елементів на екран із розташуванням їх у тих самих положення як вони були використовується підпрограма «output», після виконання якої на екран симулятора асемблера з’являють парні або не парні елементи які вибрали на початку пошуку (див. лістинг 3.4).

Лістинг 3.4:

```
output:
    lea dx,message3
    mov ah,09h
    int 21h
   mov cx,12
    xor si,si
    mov bh,00
    mov dh,24
    mov bl,7
    mov di,3
```

Для виводу елементів які не були внесенні в пошук вони будуть винесені на екран нижче (див. лістинг 3.5).

Лістинг 3.5:

```
output_matrix2:
```

```
mov ah,02h
mov dl,bl
int 10h
    mov ah,02h  ;
mov dl,byte ptr [matrix+si
int 21h
inc bl
inc si
dec di
jnz next2
inc dh
mov di,3
mov bl,7
```

4. НАЛАГОДЖЕНЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Етап налагодження

Основним етапом розроблення мої курсової роботи є розробка автоматизованої системи «Обчислення матричних елементів за умов не парності».

Програма виконує наступні операції на вибір користувача: заповненя матриці, пошук елементів в матриці та виведення на положення які вони були окремі від інших, можливість обчислення матриці як всіх елементів так і по стовпчику так і по діагоналі, пояснення в програмі як працювати з нею.

Щоб перевірити роботу програми на потрібно зайти в симулятор мови Асемблер в даному випадку «DosBox» (див. рис 4.1.).

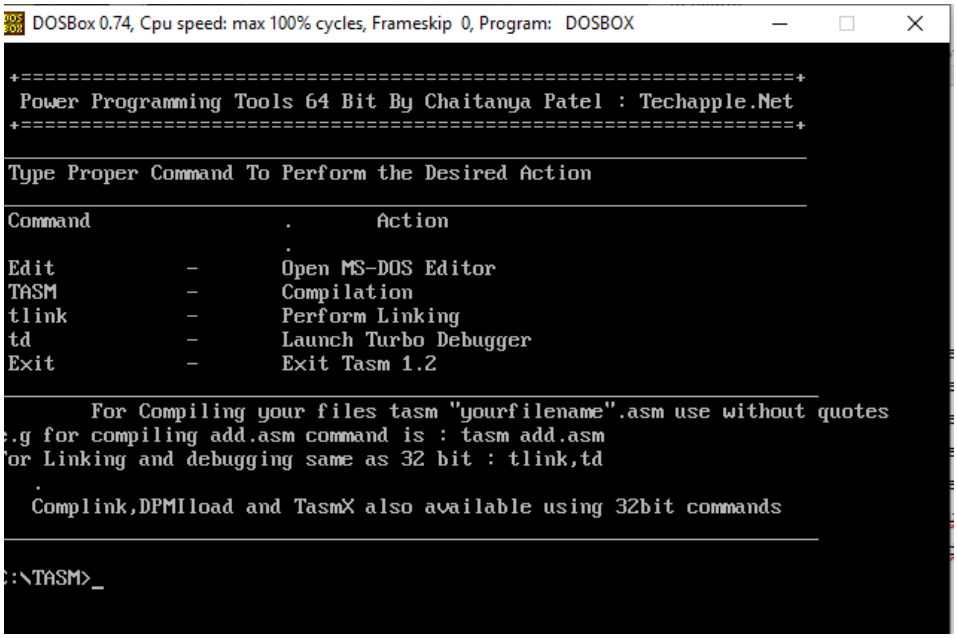

Рис. 4.1 – Головне вікно симулятора Асемблера

Після цього нам потрібно запустити компіляцію програми щоб створився об’єктний файл для створення .exe файлу (див. рис. 4.2).

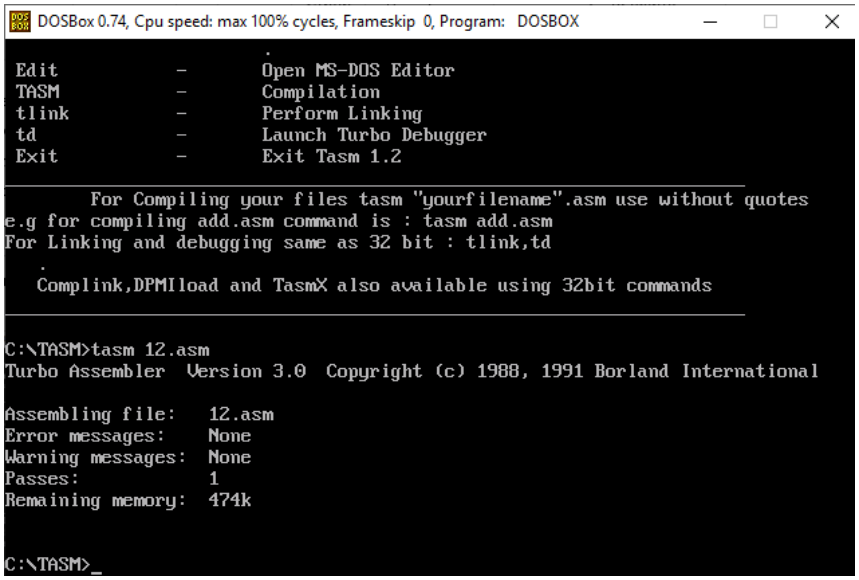

Рис. 4.2 – Компіляція файлу

Щоб створити .exe файл необхідно здійснити перевірку об’єктного файлу за допомогою команди «tlink» в даному випадку в нас помилок не виявлено про що свідчить повідомлення (див. рис. 4.3).

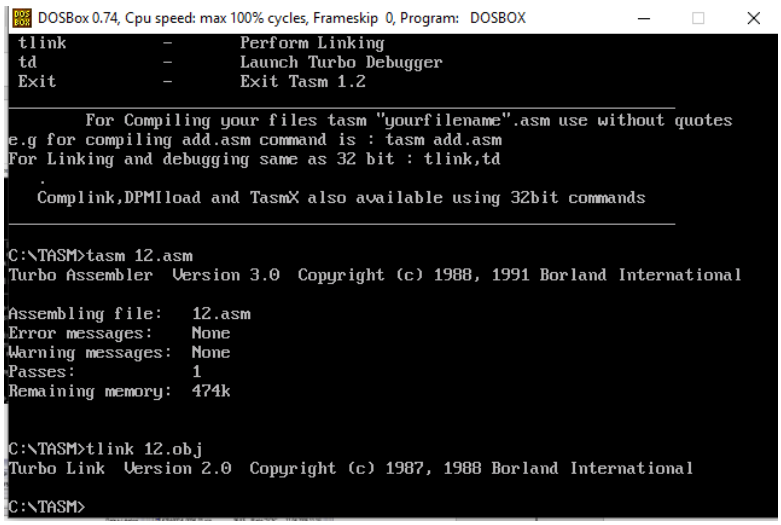

Рис. 4.3 – Створення .exe файлу

Якщо дії виконано правильно то ми можимо здійснити запуск програми для роботи з матрицею (див. рис. 4.4).

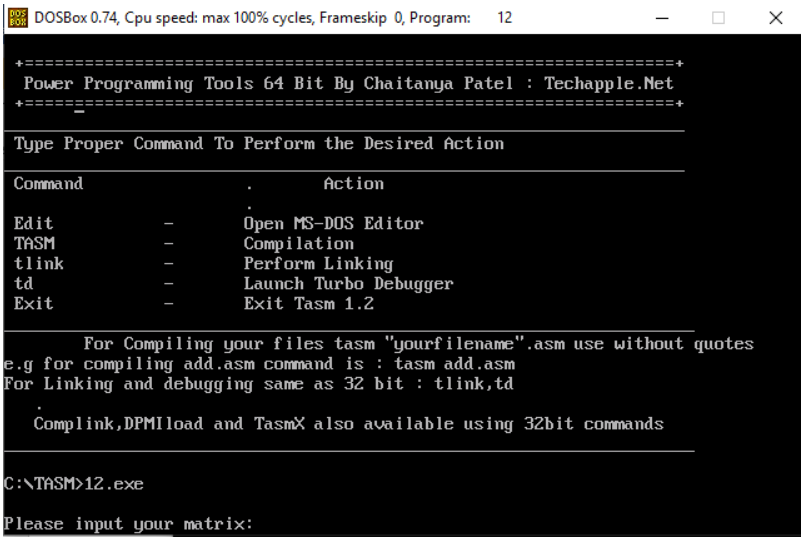

Рис. 4.4 – Запуск програми

4.2. Типи помилок

Програм без помилок не існує. Практика доводить, що винуватцями помилок у програмах найчастіше бувають самі програмісти. Один із загальних законів практичного програмування полягає в тому, що жодна програма не дає бажаних результатів при першій спробі трансляції та виконання.

Існують два типи програмних помилок:

- *синтаксичні помилки* - виникають через порушення правил мови програмування. Такі помилки зазвичай виявляються під час компіляції. Можуть бути виключені порівняно легко. Навіть якщо не переглядати текст програми можна бути впевненим, що компілятор на стадії трансляції знайде помилки і видасть відповідні попередження. Фактично пошук помилок здійснює компілятор, а їхнє виправлення - програміст;
- *семантичні (логічні) помилки* - ті, що призводять до некоректних обчислень або помилок під час виконання (run-time error). Семантичні помилки усувають зазвичай

за допомогою виконання програми з ретельно підібраними перевірочними даними, для яких відома правильна відповідь.

При тестуванні готового програмного засобу були виявлені в роботі синтаксичні помилки що не давало змогу в подальшому працювати програмі. Для запуску програми потрібно виправити всі недоліки (див. рис. 4.3.).

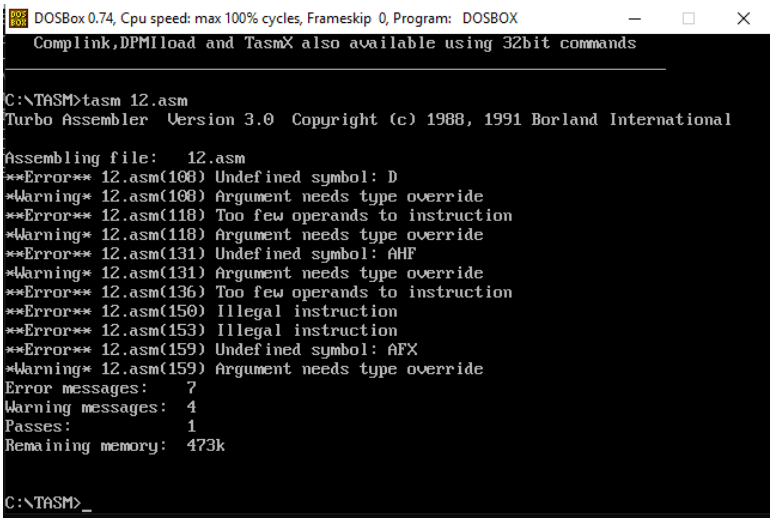

Рис. 4.3. – Повідомлення про помилки

В даному повідомені вказані рядки де були допущені помилки потірбно вирішувати зверху до низу тому що можливо допущена одна помилка яка заважає працювати всій програмі.

ВИСНОВКИ

В ході виконання даної курсової роботи було проведено аналіз та огляд різних середовищ програмування для мов С++ та трансляторів мови Асемблер. Відібраний матеріал було систематизовано, проаналізовано, узагальнено та відображено в першому розділі пояснювальної записки.

На основі аналізу проведеного пошуку визначено, що даний програмний засіб буде полегшить роботу – при обрахунку математичних матриць з пошуком парних та не парних чисел для того щоб в подальшому виконати операцію множення та додавання і виведення їх на екран. Складено технічне завдання, в якому вказано мету розробки і її призначення, основні вимоги до програми, алгоритм її функціонування, вибір апаратно-технічних засобів та мови програмування.

Розроблений автоматизована система що виконує усі функції, притаманні обрахунковим програмним засобам. Для використання даного програмного продукту необхідно мати ПК, що має конфігурацію, не нижчу за наступну:

- ОП 1 Гб;
- тактова частота процесору – 1,5 Ггц;
- Наявність вільного дискового простору 500 Мб.

Крім того, на ПК повинна бути установлена ОС Windows 8/10.

Проведені випробування засвідчили, що програма має достатню працездатність та може експлуатуватись як автоматизована система в операційних системах Windows 8/8,1/10.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Матриця (математика) [Електронний ресурс] // Вікіпедія. – 2018. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0).

2. Теорія матриць [Електронний ресурс] // Вікіпедія. – 2018. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8C.

3. Мова асемблер. Команди та основи асемблера [Електронний ресурс] // HI-NEWS. – 2017. – Режим доступу до ресурсу: http://hi-news.pp.ua/kompyuteri/12225-mova-asembler-komandi-ta-osnovi-asemblera.html.

4. Структура программы на ассемблере [Електронний ресурс] // AGPU. – 2016. – Режим доступу до ресурсу: http://www.agpu.net/fakult/ipimif/fpiit/kafinf/umk/el_lib/calc_system/Assembler/guide/Text/Structur.htm.

5. В.Семотюк. Програмування в середовищі TURBO PASCAL.-Львів:БаК,2000.-248с.

6. Чернов Б.И. Програмирование на алгоритмических языках. -М.:Просвещение,1991.-189с.

7. Борис П. C/C++ и MS Visual C++ 2008 для начинающих / Борис Пахомов. – Санкт-Петербург: БХВ-Петербург, 2015. – 608 с. – (БХВ-Петербург). – (БХВ; кн. 1)

8. Прайс Р. / Програмирование на языке Паскаль. / Прайс Р. – М.:Мир.1986 р.

9. Салтыков А.И., Семашко Г.Л. /Програмирование для всех./ Салтыков А.И -М.: Наука,1986 р.

Додаток А

Лістинг програми

```
.model small
.386
.stack 256

.data
matrix dw 16 DUP (?)
message1 db 10,13,"Please input your matrix: $"
message2 db 10,13,"Your matrix: $"
message3 db 10,13,"New matrix: $"
message4 db 10,13,"Your screen has been cleaned... $"
message5 db 10,13,"Sum of unpare elements: $"
kv_amount db 0
summa dw 0

.code
assume ds:@data, es:@data
start:
    mov ax,@data
    mov ds,ax
    mov es,ax

    lea dx,message1
    mov ah,09h
    int 21h
    mov cx,12 ;
    mov si,0  ;
```

```
    mov bh,0
    mov dh,3
    mov dl,7
    mov di,3

    input_matrix:
    mov ah,02h
    int 10h

    mov ah,01h ;
    int 21h ;
    mov byte ptr [matrix+si],al
    inc si
    inc dl
    dec di
    jnz next1
    inc dh
    mov di,3
    mov dl,7
next1:
    loop input_matrix
    lea dx,message2
    mov ah,09h
    int 21h

    mov cx,12
    xor si,si     ;
    mov bh,00
```

```
    mov dh,12
    mov bl,7
    mov di,3
output_matrix:
    mov ah,02h
    mov dl,bl
    int 10h

    mov ah,02h  ;
    mov dl,byte ptr [matrix+si];
    int 21h
    inc bl
    inc si
    dec di
    jnz next
    inc dh
    mov di,3
    mov bl,7
next:
    loop output_matrix
    mov cx,12
    mov si,0
    mov ax,[matrix+si]
unparity:
    test ax,ax
    inc si
    jnp kvadrat
    jp par
```

```
kvadrat:
    dec si
    mov ax,[matrix+si]
    add summa,ax
    mov bx,[matrix+si]
    mul bx
    mov [matrix+si],ax
    inc kv_amount ;
 par:
    loop unparity



    mov ah,09h
    lea dx,message5
    int 21h
    lea dx,summa
    int 21h

    cmp kv_amount,0  ;
    je clrscr ;
    jg output ;

clrscr:
    mov ax,0600h
    mov bh,07
    mov cx,0000
    mov dx,184fh
```

```
    int 10h

    mov ah,09h
    lea dx,message4
    int 21h

output:
    ;
    lea dx,message3  ;
            mov ah,09h
    int 21h

    mov cx,12    ;
    xor si,si     ;
    mov bh,00
    mov dh,24
    mov bl,7
    mov di,3
output_matrix2:
    mov ah,02h
    mov dl,bl
    int 10h

    mov ah,02h  ;
    mov dl,byte ptr [matrix+si];
    int 21h
    inc bl
    inc si
```

```
    dec di
    jnz next2
    inc dh
    mov di,3
    mov bl,7
next2:
    loop output_matrix2

exit:
    mov ax,4c00h
    int 21h

end start
```