

Лабораторна робота № 4 УВЕДЕННЯ В МОВУ АСЕМБЛЕР (EMU8086, DEBUG, TASM, MASM)

Мета роботи: ознайомитися з роботою різних Асемблерів при написанні програми мовою Асемблер.

Звіт має містити:

- таблиця значень регістрів розробленої програми для Emu8086;
- екранні форми виконання створеної програми для Emu8086;
- екранні форми виконання створеної програми для Debug (звичайне й послідовне виконання);
- екранні форми виконання дизасемблювання в Debug;
- екранна форма створеної програми на машинному коді в шістнадцятковому редакторі;
- висновок.

Порядок виконання роботи

Для виконання цієї лабораторної роботи необхідними є такі програми:

1. Emu8086, яка містить редактор вихідного коду й деякі інші корисні речі. Пробну версію можна скачати на сайті <http://www.emu8086.com>
2. Debug, що має невеликі можливості, але великим плюсом є належність до стандартного набору Windows (знаходиться в папках каталогу WINDOWS).
3. Шістнадцятковий редактор – як такий можна використати Волков Командер (VC), Нортон Командер (NC) або McAfee FileInsight v2.1.

Emu8086. Необхідно завантажити й установити емулятор процесора 8086. Запускаємо програму і створюємо новий файл через меню

FILE → NEW → COM TEMPLATE (Файл → Новий → Шаблон файла COM). У редакторі вихідного коду після цього видно, як було додано рядки коду (рис. 4.1).

Тут треба зазначити, що програми, які створюються з допомогою Асемблера для комп'ютерів під керуванням Windows, бувають двох типів: COM і EXE. У цій лабораторній роботі будемо створювати файли з розширенням COM як більш прості.

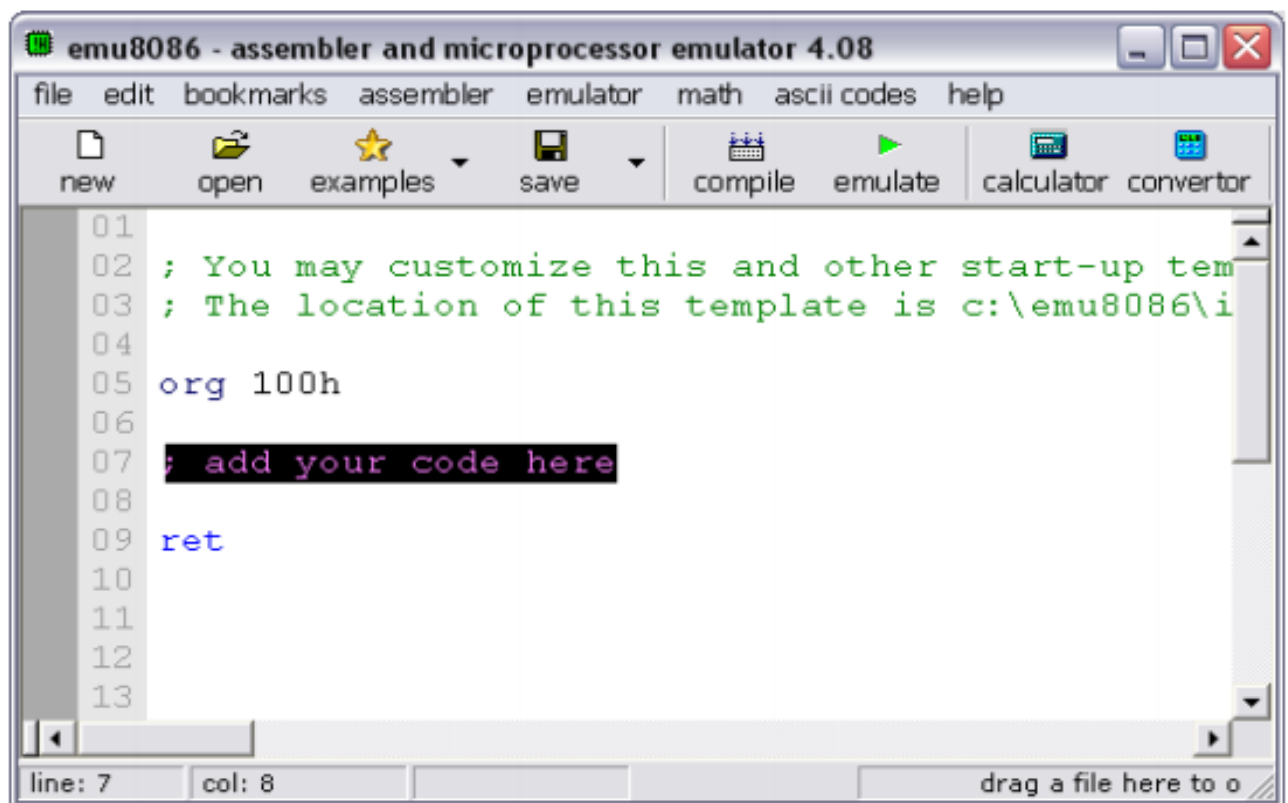


Рис. 4.1. Створення нового файлу в Emu8086

Після створення файлу в Emu8086 описаним вище способом у редакторі вихідного коду виникне рядок `add your code hear` (див. рис. 4.1). Замість цього рядка вставимо такий текст:

```
MOV AH, 02h
MOV DL, 41h
INT 21h
INT 20h
```

Крім цього у верхній частині ще є коментарі (на рис. 4.1. це текст зеленого кольору). Коментар у мові Асемблер починається з символу «`;`» (крапка з комою) і продовжується до кінця рядка. Регістр символів у мові Асемблер значення не має. Можна написати `RET`, `ret` або `Ret` – це буде одна й та сама команда.

Розберемо синтаксис програми:

`ORG 100h` – ця команда встановлює значення програмного лічильника в `100h`, тому що при завантаженні COM-файла в пам'ять DOS виділяє під блок даних PSP перші 256 байтів (десятькове число 256 дорівнює шістнадцятковому 100). Код програми розташовується тільки після цього блока. Усі програми, які компілюються у файли типу `.com`, мають починатися з цієї директиви.

`MOV AH, 02h` – інструкція (або команда) `MOV` поміщає значення другого операнду в перший операнд, тобто значення `02h` поміщається

в регістр AH, а 02h – це ДОСівська функція, яка виводить символ на екран. Записуємо цю функцію (а точніше її номер) саме в регістр AH, тому що переривання 21h використовує саме цей регістр.

MOV DL, 41h – код символу «А» заноситься в регістр DL. Код символу «А» за стандартом ASCII – 41h.

INT 21h – це і є те саме переривання 21h – команда, яка викликає системну функцію DOS, задану в регістрі AH (у цьому прикладі це – функція 02h). Команда INT 21h – основний засіб взаємодії програм з операційною системою (ОС).

INT 20h – переривання, яке повідомляє операційній системі про вихід з програми і передачу керування консольному додатку. Отже, при використанні INT 20h у цьому прикладі управління буде передаватися програмі Emu8086. Якщо ж програма вже є відкомпільованою і її запущено з ОС, то команда INT 20h поверне керування в ОС (наприклад, у DOS). У принципі, для Emu8086 цю команду можна було би пропустити, оскільки цю саму функцію виконує команда RET, яка вставляється у вихідний текст автоматично при створенні нового файлу за шаблоном (див. рис. 4.1). Однак для сумісності з іншими Асемблерами краще використовувати INT 20h.

Можна зберегти цей файл куди-небудь на диск. Щоб виконати програму, натисніть кнопку EMULATE (із зеленим трикутником) або клавішу F5. Відкриються вікна емулятора й вихідного коду (рис. 4.2).

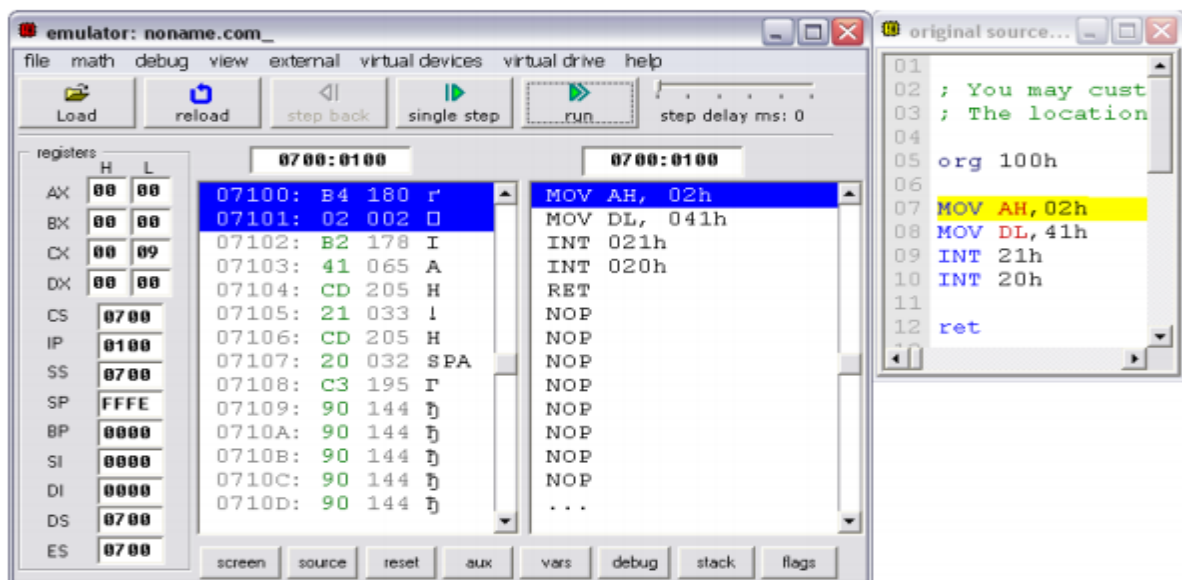


Рис. 4.2. Вікно емулятора Emu8086

У вікні емулятора відображаються регістри й знаходяться кнопки керування програмою. У вікні вихідного коду відображаються вихідний текст програми, де підсвічується рядок, що виконується в поточний момент. Усе це є дуже зручним для вивчення й налаштування програм.

У вікні емулятора запускаємо програму на виконання в покроковому режимі (кнопка SINGLE STEP) і дивимось, як змінюються значення регістрів.

У звіт про виконання лабораторної роботи записуємо (у вигляді таблиці) вихідні й змінені значення регістрів з поясненнями, наприклад: регістр CS змінює свої значення під час виконання програми, оскільки відображає сегмент команд, що виконуються.

Після цього (якщо не було помилок в тексті програми) виникне повідомлення про завершення програми (див. рис. 4.3). Тут буде повідомлення про те, що програма передала керування ОС, тобто її було успішно завершено. Результатом роботи програми, написаною мовою Асемблер буде виведення на екран англійської літери «А» (рис. 4.4).

Emu8086 – це емулятор, який емулює роботу комп'ютера з процесором 8086, тому в описаному вище прикладі програма виконується не операційною системою, а емулятором. Emu8086 може створювати й реальні програми, які можуть самостійно виконуватися на комп'ютері.

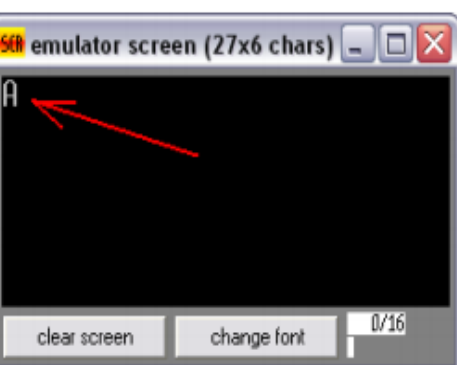
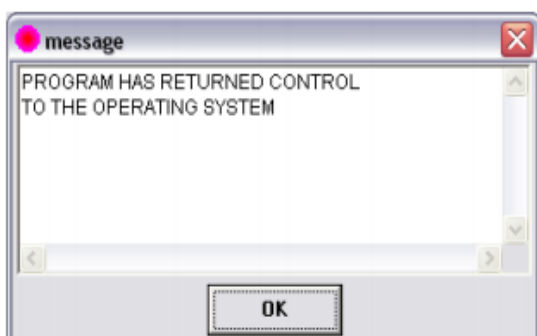


Рис. 4.3. Повідомлення про завершення виконання програми

Рис. 4.4. Результат програми

Debug. Програма Debug входить до складу Windows. Запустити програму Debug можна з командного рядка або безпосередньо з папки, у якій вона знаходиться (C: \ Windows \ System32 \ debug.exe). Щоб запустити програму з командного рядка, виберіть команду з меню

ПУСК → ВЫПОЛНИТЬ або натисніть комбінацію клавіш WIN + R. У вікні напишіть debug і натисніть «Выполнить» (якщо у використовуваній версії Windows все-таки не виявилось цієї програми, скопіюйте виконуваний файл debug.exe з папки з лабораторною роботою до відповідного каталога). Після цього відкриється консольне вікно з порожнім екраном для введення команд програми. Введіть команду а і натисніть клавішу ENTER на клавіатурі. Потім уведіть таку програму, натискаючи клавішу ENTER у кінці кожного рядка:

```
MOV AH, 02
MOV DL, 41
INT 21
INT 20
```

Зверніть увагу, що всі числові значення пишуться без h у кінці. Це тому, що Debug працює тільки з шістнадцятковими числами, і їй не треба пояснювати, у якій системі числення вводяться дані. Результат буде приблизно таким, як показано на рис. 4.5.

Після введення команди «а» виникають символи 0B72:0100. Перші чотири символи можуть бути й іншими, а 0100 – це адреса, з якої починається виконання програми, тобто в пам'ять до цієї адреси заноситься перша команда програми (для файлів COM). Кожна команда займає два байти, тому наступною адресою буде 0102 і т. д.

Програму написано і тепер потрібно перевірити її роботу. Натисніть клавішу ENTER ще раз, потім уведіть команду g (від англ. go) і знову натисніть клавішу ENTER.

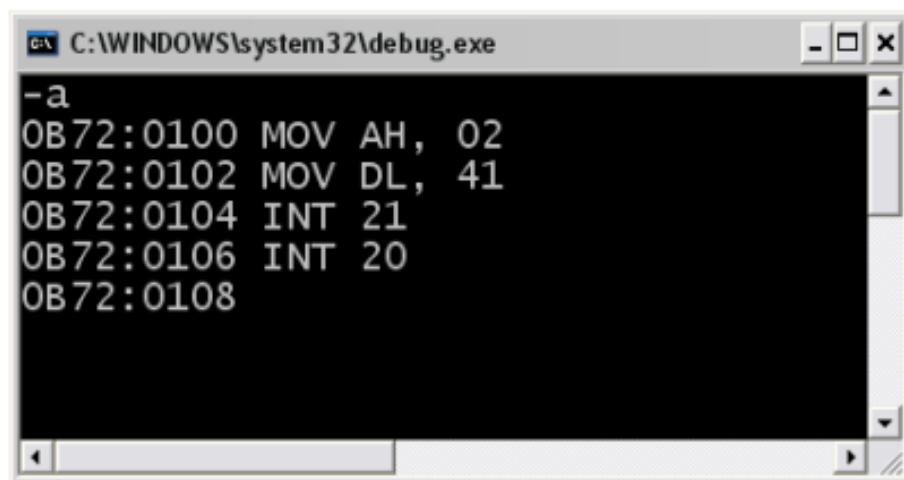


Рис. 4.5. Створення програми в Debug

На екрані виникне такий запис:

```
-g
A
Програма завершилась нормально
-
```

Тут А – та сама буква, яка виводиться на екран унаслідок роботи програми. Потім іде повідомлення про нормальне завершення програми (яке може бути різним залежно від версії Debug).

Запустіть програму в покроковому режимі. Для цього заново наберіть текст програми (спочатку введіть команду а), потім введіть команду `t` і натисніть клавішу ENTER. На екрані побачите таке:

```
AX=0200  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B72  ES=0B72  SS=0B72  CS=0B72  IP=0102  NV UP EI PL NZ NA PO NC
0B72:0102 B241          MOV    DL,41
```

Це – відображення стану регістрів процесора після виконання першого рядка програми. Як можна бачити, у регістр `AX` записалося число 02. У нижньому рядку знаходиться адреса команди і сама команда, що буде виконуватися наступною.

Знову введіть команду `t` і натисніть клавішу ENTER. На екрані виникне таке:

```
AX=0200  BX=0000  CX=0000  DX=0041  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B72  ES=0B72  SS=0B72  CS=0B72  IP=0104  NV UP EI PL NZ NA PO NC
0B72:0104 CD21          INT     21
```

Команда `MOV DL, 41` записала в регістр `DL` число 41. Знову введіть команду `t` і натисніть клавішу ENTER. На екрані виникне такий запис:

```
AX=0200  BX=0000  CX=0000  DX=0041  SP=FFE8  BP=0000  SI=0000  DI=0000
DS=0B72  ES=0B72  SS=0B72  CS=0347  IP=0225  NV UP EI PL NZ NA PO NC
0347:0225 80FC4B  CMP    AH,4B
```

Зверніть увагу, що команди `CMP AH, 4B` немає в програмі. Програма вже завершила свою роботу, але можна ще довго вводити команду `t`, при цьому будуть видаватися стани регістрів. Щоб остаточно виконати програму, уведіть команду `g` і натисніть клавішу ENTER. Програму написано й перевірено.

Створить виконуваний файл програми (`.com`). Для цього визначіть, якого розміру буде файл. Виконання програми почнеться з адреси `0100h`, останній рядок у програмі містить адресу `0108h`. Це означає, що розмір файла буде становити вісім байтів ($108h - 100h = 8$).

Тепер виконайте такі дії:

1. Знову напишіть програму.
 2. Запишіть у регістр `CX` розмір файла. Для цього введіть команду `r cx` і натисніть клавішу ENTER. Потім введіть розмір файла (8 байт) і натисніть клавішу ENTER.
 3. Введіть команду `n`, потім один пробіл та ім'я файла. Натисніть клавішу ENTER.
 4. Введіть команду `w` і натисніть клавішу ENTER.
- Унаслідок усіх цих дій на екрані виникне інформація, показана на рис. 4.6.



Рис. 4.6. Створення виконуваного файла `.com` з допомогою Debug

Якщо робота виконується в режимі емуляції DOS з-під WINDOWS, то файл `debug_1.com` збережеться на робочий стіл або в папку поточного користувача. Це залежить від версії і (або) налаштувань WINDOWS. Тепер його можна запустити як звичайну програму. Якщо в зазначених папках немає цього файла, то знайдіть його через пошук файлів. Вихід з Debug здійснюється командою `q`.

Дизасемблювання з Debug. Дизасемблювання – це перетворення виконуваного файла на вихідний код мовою Асемблер. Для дизасемблювання наберіть у командному рядку

```
debug debug_1.com
```

Тут `debug_1.com` – ім'я файла, який необхідно дизасемблювати. Натисніть клавішу ENTER. Якщо програма не відкривається, то потрібно вказати повний напрямок до неї, наприклад:

```
C:\WINDOWS\COMMAND\debug debug_1.com
```

Якщо ж програма запустилася, але видала помилку (наприклад, «Ошибка 1282» або «Файл не найден»), то потрібно вказати повний шлях до файла, наприклад:


```
C:\WINDOWS\COMMAND\debug C:\MYPROG\debug_1.com
```

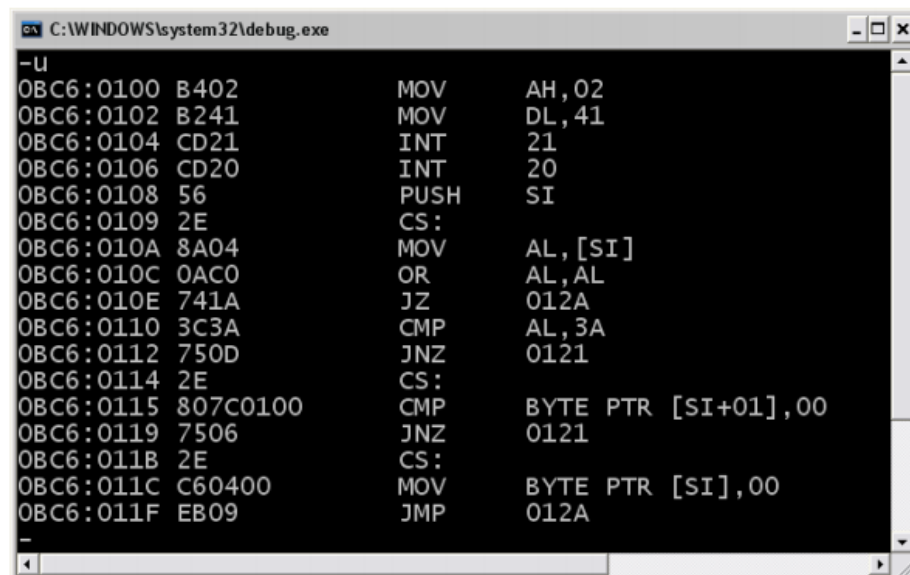
Якщо і це не допомогло, то, можливо, десь було допущено помилку в напрямку або напрямом не відповідає вимогам DOS. У такому випадку краще помістити програму в корінь диска C, звідки вона гарантовано завантажиться, наприклад:

```
C:\debug_1.com
```

Якщо Debug запусився без повідомлень про помилки, то введіть команду `u` і натисніть клавішу ENTER (рис. 4.7).

Перші чотири рядки – це програма, яку було введено. Інші рядки – це інструкції, що залишилися від програм або даних, які відпрацювали до запуску Debug. Постає питання: якщо розглядати незнайомий файл, то як дізнатися, де закінчується програма й починається «сміття»? Орієнтовно це можна зробити за розміром файла (для дуже маленьких програм). Розмір можна подивитись у властивостях файла. Тільки слід урахувувати, що у властивостях файла розмір наведено в десятковій формі, а Debug видає шістнадцяткові адреси. Тому доведеться перевести десяткове число в шістнадцяткове.

Існує ще один варіант (який теж не завжди є прийнятним) – знайти в отриманому списку рядок, що містить команду виходу з програми (INT 20). Якщо програма велика, то список її команд не поміститься на екран. Тоді знову введіть команду `u` і натисніть клавішу ENTER, і так до кінця програми.



```
C:\WINDOWS\system32\debug.exe
-u
0BC6:0100 B402      MOV     AH,02
0BC6:0102 B241      MOV     DL,41
0BC6:0104 CD21      INT     21
0BC6:0106 CD20      INT     20
0BC6:0108 56       PUSH    SI
0BC6:0109 2E       CS:
0BC6:010A 8A04      MOV     AL,[SI]
0BC6:010C 0AC0      OR      AL,AL
0BC6:010E 741A      JZ      012A
0BC6:0110 3C3A      CMP     AL,3A
0BC6:0112 750D      JNZ     0121
0BC6:0114 2E       CS:
0BC6:0115 807C0100 CMP     BYTE PTR [SI+01],00
0BC6:0119 7506      JNZ     0121
0BC6:011B 2E       CS:
0BC6:011C C60400    MOV     BYTE PTR [SI],00
0BC6:011F EB09      JMP     012A
```

Рис. 4.7. Дизасемблер Debug

Якщо ж на екрані буде інша програма, то це сталося, можливо, або через те, що програма чомусь не завантажилася, або через невідповідність адрес. Тому необхідно звертати увагу на адреси пам'яті,

які вказано в лівій колонці. Програма починається з адреси 0100. Якщо адреса є іншою, то й програма буде іншою.

Шістнадцятковий редактор. Для початку необхідно установити шістнадцятковий редактор. Для прикладу роботи взято редактор McAfee FileInsight v2.1. Шістнадцяткові редактори й дизасемблери використовуються, наприклад, для налагодження або вивчення машинних кодів і т. ін. Запускаємо шістнадцятковий редактор, клацаємо по кнопці Открыть, знаходимо один зі створених СОМ-файлів, наприклад debug_1.com, і завантажуюмо його в редактор. Коли файл завантажено, в редакторі відкриється вікно, зображене на рис. 4.8.

Якщо відкриється інше вікно, то, можливо, переглядається файл у текстовому режимі. У такому випадку натисніть кнопку View as Hex на панелі інструментів (див. рис. 4.8).

Розглянемо машинний код програми (правий верхній кут). Нулями позначено першу комірку пам'яті, до якої записано число B4. Це число потім буде записано за адресою 0100h (для СОМ-файла). У рядку має бути 16 чисел, кожне з яких складається з двох цифр. Числа записуються в шістнадцятковій формі. Оскільки розмір програми усього вісім байтів, то й чисел буде вісім.

Команда B4 (MOV AH, 02) означає «ввести значення 02 (наступне в рядку число) у регістр AH».

Команда B2 (MOV DL, 41) означає «ввести значення 41 у регістр DL».

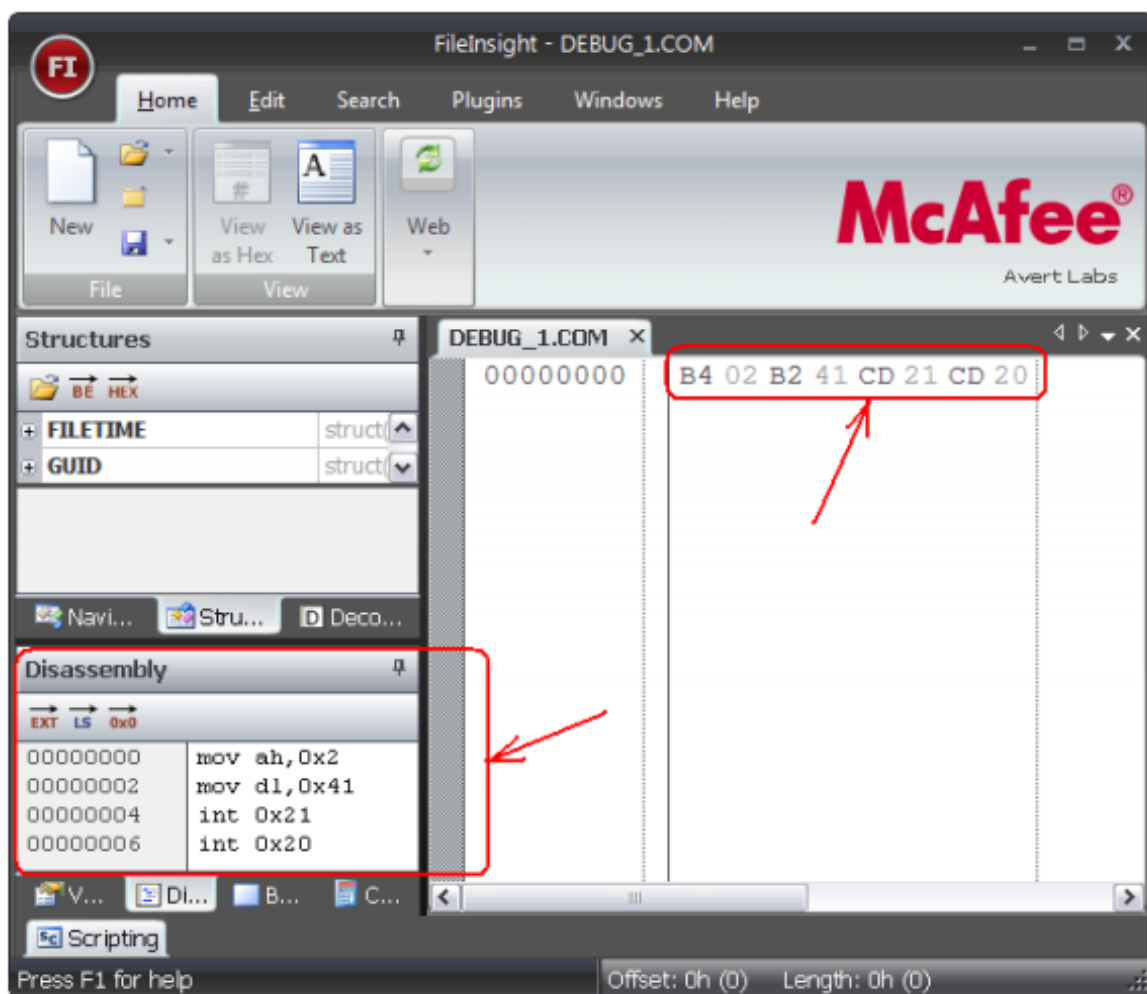


Рис. 4.8. Файл DEBUG_1.COM у шістнадцятковому редакторі

У програмі Debug після введення команди `t` на екрані виникне таке:

```
AX=0200  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B72  ES=0B72  SS=0B72  CS=0B72  IP=0102  NV UP EI PL NZ NA PO NC
0B72:0102 B241          MOV     DL, 41
```

В останньому рядку B241 – машинний код команди `MOV DL, 41`.

Машинний код команди `CD` – це код виклику переривання, отже, `CD 21` – код команди `INT 21`. Аналогічно `CD 20` – код команди `INT 20`.

Тепер напишемо й створимо програму без асемблерів і компонувальників: відкриємо редактор, створимо новий файл. Для цього клацнемо кнопку **NEW** на панелі інструментів, потім – кнопку **View as Hex** і введемо дані

```
00000000 B4 02 B2 41 CD 21 CD 20
```

Зберігаємо файл під ім'ям, наприклад, `hex_1.com`. Програма готова, тепер її можна запустити – результат буде таким самим, що й у всіх

попередніх випадках. Крім того, редактор McAfee FileInsight v2.1 має свій дизасемблер. Якщо завантажити в редактор виконуваний файл, а в лівому нижньому кутку вибрати вкладку DISASSEMBLY, то можна подивитися вихідний код завантаженої програми мовою Асемблер (див. рис. 4.8).

MASM і TASM. Асемблери MASM і TASM відрізняються один від одного. Однак створення простих програм для них є майже однаковим, за винятком самого асемблювання й компонування. Отже, перша програма для MASM і TASM, яка виводить англійську букву «А» в поточній позиції курсора, тобто в лівому верхньому кутку екрана, буде такою:

```
.model tiny
.code
ORG 100h
start: MOV AH,2
MOV DL,41h
INT 21h
INT 20h
END start
```

Цей текст можна набрати в будь-якому простому текстовому редакторі, потім зберегти цей файл з розширенням `.asm`, наприклад, у папці MYPROG.

Назвемо файл `atest`. Отже, отримаємо `C:\MYPROG\atest.asm`

Зверніть увагу, що в першій команді було записано `2` замість `02h`. У MASM, TASM, як і в Emu8086, допускається такий запис, хоча можна написати `02h` – помилки не буде.

Пояснення до програми:

`.model tiny` – перший рядок. Директива `.model` визначає модель пам'яті для конкретного типу файлів. У цій лабораторній роботі це – файл з розширенням `COM`, тому вибираємо модель `tiny`, у якій об'єднано сегменти коду, даних і стеку. Модель `tiny` призначено для створення файлів типу `COM`;

`.code` – другий рядок. Ця директива починає сегмент коду;

`ORG 100h` – третій рядок. Ця команда встановлює значення програмного лічильника в `100h`, тому що при завантаженні `COM`-файла в пам'ять, DOS виділяє під блок даних PSP перші 256 байтів (десятькове число `256` дорівнює шістнадцятковому `100h`). Код програми розташовується тільки після цього блоку. Усі програми, які компілюються у файли типу `COM`, повинні починатися з цієї директиви;

`Start: MOV AH, 02h` – четвертий рядок. Мітка `start` розташовується перед першою командою в програмі і буде

використовуватися в директиві `END`, щоб указати, з якої команди починається програма. Інструкція `MOV` поміщає значення другого операнда в перший, тобто значення `02h` поміщається в регістр `АH`;

`MOV DL, 41h` – п'ятий рядок. Код символу «А» заноситься в регістр `DL`. За стандартом ASCII – це число `41h`;

`INT 21h` – шостий рядок. Це і є те саме переривання `21h` – команда, яка викликає системну функцію DOS, задану в регістрі `АH` (у цьому прикладі це – функція `02h`). Команда `INT 21h` – основний засіб взаємодії програм з ОС;

`INT 20h` – сьомий рядок. Це переривання, яке повідомляє операційній системі про вихід з програми і передачу керування консольному додатку. У тому випадку, якщо програму вже відкомпільовано й запущено з ОС, команда `INT 20h` поверне керування в ОС;

`END start` – восьмий рядок. Директива `END` завершує програму, одночасно вказуючи, з якою мітки має починатися її виконання.

Для того щоб подивитися, як працює програма, необхідно спочатку викликати Асемблер і скомпілювати її в об'єктний файл, потім викликати компоновник, який з об'єктного файла створить виконуваний файл, тобто програму типу `COM`. Для різних Асемблерів доведеться виконувати ці дії по-різному.

Асемблювання в TASM. Для TASM створюємо об'єктний файл з ім'ям `atest.obj`, набравши в командному рядку команду

```
tasm atest.asm
```

Якщо програма `tasm` знаходиться не в кореновому каталозі диска `C:\` і в файл `autoexec.bat` не внесено відповідних змін, то слід вказувати повний шлях до цієї програми. Це стосується і файла `atest.asm`. У цьому випадку команда може мати, наприклад, такий вигляд:

```
C:\TASM\BIN\tasm C:\MYPROG\atest.asm
```

Далі слід скомпонувати отриманий файл `atest.obj` у виконуваний файл `atest.com`. Для цього набираємо команду

```
tlink /t /x atest.obj
```

або повний шлях

```
C:\TASM\BIN\tlink /t /x atest.obj
```

Зверніть увагу, що для файла `atest.obj` не потрібно вказувати повний шлях, оскільки він записується в папку `C:\TASM\BIN\`. Туди ж за замовчуванням записується й готова до виконання програма `atest.com`. Тепер можна її запустити, і на екрані буде відображено букву «А» в лівому верхньому кутку.

Якщо застосовується режим емуляції DOS з-під WINDOWS, то файли `atest.obj` і `atest.com` за замовчуванням зберігаються на робочий стіл або в папку користувача. Це залежить від версії і (або) налаштувань WINDOWS.

Асемблювання в MASM. Для MASM дії є аналогічними, тільки виклики асемблера й компоновника дещо різняться. Створення об'єктного файла:

```
C:\MASM611\ml /c atest.asm
```

Компонування об'єктного файла:

```
C:\MASM611\BINR\link /TINY atest.obj,,NUL,,
```

Зверніть увагу, що асемблер і компоновник знаходяться в різних каталогах. Тут викликається 16-розрядний компоновник, який створює програми для реального режиму DOS. Це має місце для версії MASM 6.11, для інших версій процес створення може дещо відрізнятися.

Файл `atest.com` за замовчуванням створюється в тому самому каталозі, де знаходяться вихідний і об'єктний файли програми.

Виконання програми. Якщо все зроблено правильно, то створиться файл `atest.com`. Тепер програму з цього файла можна запустити на виконання. Робиться це звичайним способом для операційної системи: для DOS – з командного рядка, для Windows – подвійним клацанням по значку програми або також з командного рядка.

Однак якщо запускати програму в Windows, то результат роботи програми можна не встигнути побачити, оскільки вікно відразу ж закриється після її виконання. Щоб цього не сталося, необхідно клацнути правою кнопкою миші по файлу `atest.com` і в контекстному меню вибрати «Властивості». У вікні, що виникне, перейти на вкладку «Программа» і зняти галочку «Закривать окно после окончания программы» (залежно від версії Windows ця процедура може трохи візнитися).

Після виконання програми на екрані виникне англійська буква «А» (рис. 4.9). Можна трохи поекспериментувати і замість коду літери «А» (41h) записати інший код (42h тощо аж до 0FFh).



Рис. 4.9. Результат програми (асемблер MASM)

Використання BAT-файлів. BAT-файл (або пакетний файл) – це звичайний текстовий файл з розширенням BAT, у якому записуються команди для виконання операційною системою, тобто так само, як це робиться в командному рядку. Тільки у BAT-файлі можна записати відразу кілька команд, і всі ці команди потім можна виконати клацанням миші.

BAT-файл дасть змогу швидко асемблювати й компоувати вихідні тексти мовою Асемблер.


Створимо найпростіший BAT-файл, з допомогою якого здійснемо асемблювання й компоувку, потім – виконуваний файл типу COM з допомогою Асемблера MASM. Отже, відкриємо текстовий редактор, створимо новий файл і напишемо там такий текст:

```
C:\MASM611\BIN\ml /c atest.asm  
PAUSE  
C:\MASM611\BINR\link /TINY atest.obj,,NUL,,,  
PAUSE
```

Тут команда `PAUSE` призупиняє виконання команд BAT-файла й виводить повідомлення «Для продовження натисніть ENTER...». Команди продовжать виконуватися після натиснення на клавішу `ENTER` на клавіатурі. Збережемо цей файл з розширенням BAT у тому самому каталозі, де знаходиться вихідний файл `atest.asm`. Назвемо його, наприклад, `com_create.bat`.

Тепер виконаємо цей BAT-файл звичайним для Windows способом, тобто двічі клацнемо по ньому лівою кнопкою миші. При цьому ОС почне по черзі виконувати команди, записані в пакетному файлі. Спочатку виконається асемблювання (створення об'єктного файла), потім – команда `PAUSE`. Ця команда тут для того, щоб можна було подивитися результат асемблювання. Після натискання клавіші `ENTER`

виконається компонування (створення виконуваного файлу типу COM, тобто готової програми). Потім знову буде пауза, щоб можна було побачити результат. На екрані це буде мати приблизно такий вигляд, як показано на рис. 4.10.



```
C:\WINDOWS\system32\cmd.exe

E:\LITERATURE\MY_BOOK\ASSEMBLER\SOURCE\CH_01\MASM>C:\MASM611\BIN\ml /c atest.asm

Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993. All rights reserved.

Assembling: atest.asm

E:\LITERATURE\MY_BOOK\ASSEMBLER\SOURCE\CH_01\MASM>PAUSE
Для продолжения нажмите любую клавишу . . .

E:\LITERATURE\MY_BOOK\ASSEMBLER\SOURCE\CH_01\MASM>C:\MASM611\BIN\link /TINY atest.obj,,NUL,,,

Microsoft (R) Segmented Executable Linker Version 5.31.009 Jul 13 1992
Copyright (C) Microsoft Corp 1984-1992. All rights reserved.

LINK : warning L4045: name of output file is 'atest.com'

E:\LITERATURE\MY_BOOK\ASSEMBLER\SOURCE\CH_01\MASM>PAUSE
Для продолжения нажмите любую клавишу . . .
```

Рис. 4.10 – Створення програми на MASM з допомогою BAT-файла

Звичайно, напрямки можуть різнитися. Як бачимо, спочатку виконується асемблювання:

```
Assembling: atest.asm
```

Потім виконується команда PAUSE:

Напрямок до BAT файла > PAUSE

Після натискання клавіші ENTER виконується компонування:

```
LINK: warning L4045: name of output file is 'atest.com'
```

Після цього надійде повідомлення, що компоновник створив вихідний файл atest.com, у чому й можна переконатися, подивившись у каталог з вихідними файлами.