Лабораторна робота № 5

Тема: Створення та відлагодження програм на асемблері

Мета: Одержати навички підготовки асемблерних програм, їх трансляції, компонування та відлагодження

Короткі теоретичні відомості

Кожна програма, написана на мові асемблера, зазвичай складається з кількох сегментів. Це так звані логічні сегменти – іменовані області пам’яті, призначені для зберігання даних певного функціонального призначення. Більшість програм мають сегменти коду, даних і стеку.

Існують також фізичні сегменти – області пам’яті, до яких має доступ процесор через свої шини за допомогою спеціальних сегментних регістрів. Процесори сімейства Intel мають чотири сегментних регістри, які призначені для адресації сегментів коду (CS), даних (DS, ES) і стеку (SS). У 32-розрядних процесорів з’явилися ще два сегментних регістри (FS, GS), хоча ними користуються значно рідше.

При написанні програм для процесорів Intel потрібно визначити, в якому режимі буде працювати процесор при виконанні коду і яка модель пам'яті буде використовуватись. 16-розрядні програми, які створювались для перших процесорів Intel 8086 (8088), використовували режим реальної адресації і механізм сегментації пам’яті. У цьому режимі процесор підтримує фізичний адресний простір розміром 1 МБайт. Цей адресний простір ділиться на сегменти, кожен з яких може бути до 64 кілобайт в довжину. Фізичний сегмент має базу (початок), яка визначається 16-бітним сегментним селектором. Операнд в межах сегмента адресується 16-бітним зміщенням відносно початку сегмента.

Мова асемблера має спеціальні директиви для позначення сегментів програми.

Стандартні директиви сегментації явно визначають початок і кінець кожного сегмента за допомогою ключових слів SEGMENT і ENDS, відповідно. У програмі може міститися будь-яка кількість сегментів, головне, щоб вони мали унікальні імена. Сегменти можуть об'єднуватися в групи. Приклад використання директиви для опису сегменту коду:

```
CODE SEGMENT
     ...   ; оператори коду
CODE ENDS
```

Для того, щоб встановити відповідність між сегментними регістрами процесора і логічними сегментами програми необхідна директива ASSUME. Наприклад:

```
ASSUME CS:CODE, DS:DATA, SS:STACK
```

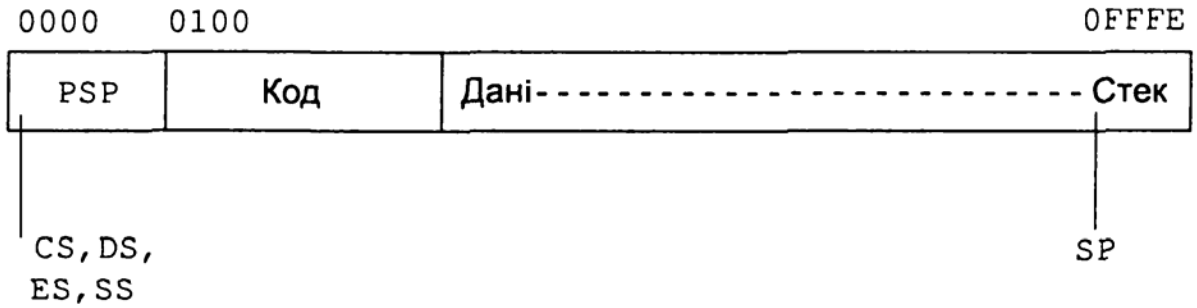
Ця директива не ініціалізує сегментні регістри, а лише вказує асемблеру, що адреси команд та даних слід відраховувати відносно вмісту заданих регістрів.

Існують також **спрощені директиви сегментації**. Для їх використання спочатку оголошують певну модель пам’яті. Модель пам’яті описує найбільш типові випадки використання сегментів. Наприклад, у крихітній моделі пам’яті (TINY) код, дані і стек розташовуються в одному сегменті розміром до 64 Кб. Мала модель (SMALL) має два сегменти: один для коду, другий для даних і стеку. Плоска модель пам’яті (FLAT) схожа на крихітну тим, що використовує єдиний сегмент, але максимальний його розмір може сягати 4 Гб. У нашому циклі лабораторних робіт ми будемо використовувати малу модель пам’яті для 16-бітних програм. Модель пам’яті оголошується директивою .MODEL:

```
.model small
```

Після оголошення моделі пам’яті можна використовувати спрощені директиви сегментації. До них відносяться директиви .DATA, .CODE і .STACK, які позначають початок сегментів коду, стеку і даних відповідно. Кінець сегментів позначати не потрібно, кожна нова директива припиняє дію попередньої.

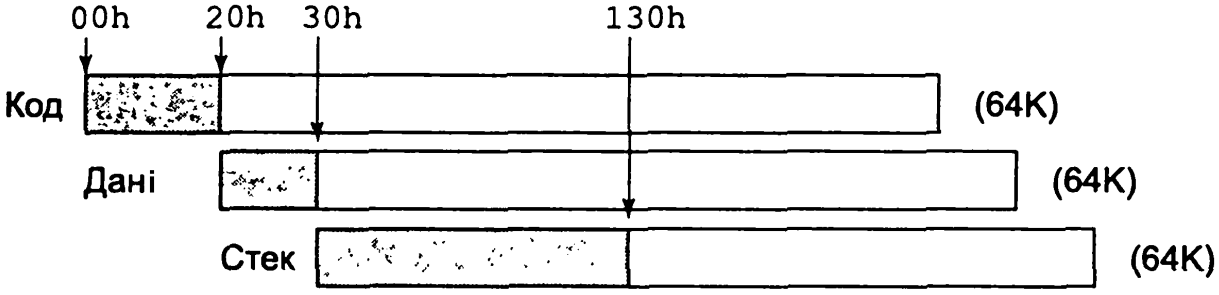
Існує **два типи асемблерних програм**, призначених для 16-бітного програмування, які розрізняються розширенням виконуваного файлу (.COM і .EXE). Файл з розширенням .COM являє собою двійковий образ машинного коду. Він завантажується в пам'ять системою MS-DOS за найменшою доступною сегментною адресою зі зміщенням 100h. Перших 256 байтів сегменту призначені для розташування префіксу програмного сегменту (PSP). Таким чином, сумарна довжина .COM-програми (з урахуванням довжини сегмента даних і стеку) не може перевищувати 64 Кбайт – 256 байт – 2 байти стеку, оскільки сегменти коду, даних і стеку об'єднані в один фізичний сегмент пам'яті. При запуску .COM-програми у всіх її сегментних регістрах знаходиться базова адреса PSP. Область коду починається із зміщення 100h відносно сегменту PSP, а відразу за областю коду розташовується область даних. Область стеку розташовується в кінці фізичного сегменту пам'яті, оскільки в процесорах Intel стек "росте" зверху вниз. Тому при запуску .COM-програми в регістр SP завантажується зміщення 0FFFEh.



Файл програми з розширенням .EXE складається із заголовка, за яким власне записаний завантажувальний модуль програми. У заголовку програми зберігається службова інформація, завдяки якій операційна система може завантажити в пам'ять і запустити на виконання цю програми.

При завантаженні .EXE-файлу в пам'ять операційна система MS-DOS спочатку створює для нього префікс програмного сегменту (PSP), який розташовується починаючи з першого вільної ділянки пам'яті, а слідом за ним у пам'ять завантажується сама програма. Після розшифровки заголовка програми і завантаження її в пам'ять, в регістри DS і ES завантажується сегментна адреса PSP, а в регістри CS і IP - адреса точки входу програми.

У регістр SS завантажується адреса початку сегменту стеку, а в регістр SP – адреса його вершини. Програмні сегменти можуть бути розташовані в окремих ділянках пам’яті, частково перекриватися, або співпадати.



Програма типу .COM складається всього з одного сегменту обсягом до 64 КБайт. Вона не має окремого сегменту для розміщення даних, тому зазвичай дані розміщують після коду, в кінці програми.

```
TITLE Програма helcom.asm
.model tiny
.code
org 100h
start:
     mov ah,9                    ; вивести на екран
     lea dx, message             ; текстове повідомлення
     int 21h

     mov ax,4C00h                ; завершити
     int 21h                     ; програму

message DB "Hello World!",0dh,0ah,"$"
END start
```

Програма типу .EXE складається з кількох сегментів і може використовувати стандартні або спрощені директиви сегментації.

Приклад програми із використанням стандартних директив (універсальний скелет).

```
TITLE Програма helstd.asm

data segment
message DB "Hello World!",0dh,0ah,"$"
data ends

code segment
assume cs:code, ds:data

start:     ; точка входу. Тут починається виконання
     mov ax, data                ; налаштувати сегментний регістр
     mov ds, ax                  ; DS на початок сегменту даних

     mov ah,9                    ; вивести на екран
     lea dx, message             ; текстове повідомлення
     int 21h

     mov ax,4C00h                ; завершити
     int 21h                     ; програму

code ends

end start                        ; вказує на точку входу
```

Приклад програми із використанням спрощених директив сегментації.

```
TITLE Програма helsim.asm
.model small

.data
message DB "Hello World!",0dh,0ah,"$"

.code
start:
     mov ax, @data               ; налаштувати сегментний
     mov ds, ax                  ; регістр DS
```

```
     mov ah,9                    ; вивести на екран
     lea dx, message             ; текстове повідомлення
     int 21h

     mov ax,4C00h                ; завершити
     int 21h                     ; програму

.stack 100h

end start                        ; точка входу
```

Асемеблер не має практично жодних засобів введення-виведення. Саме тому у вищенаведених програмах для виведення тексту на екран використовується програмне переривання DOS (функція 9), яке викликається командою `int 21h`.

Бібліотека asmio16

Для полегшення програмування використовують бібліотеки асемблерних підпрограм. Однією з таких бібліотек є `asmio16.lib`. Опис функцій бібліотеки знаходиться у файлі довідки (asmio16 Library.pdf).

Програма з використанням бібліотеки має вигляд.

```
TITLE Програма hellib.asm
include asmio16.inc
.stack 100h

.data
mes db "Hello, world!",0dh,0ah,0

.code
.begin                           ; макрос початку - визначає точку
                                 ; входу start і налаштовує DS, ES
     lea dx, mes                 ; адреса текстового рядка
     call WriteString            ; вивести текстовий рядок
.end                             ; макрос – завершити програму
```

Порядок виконання роботи

1. Підготуйте робоче середовище для створення програм на асемблері. Створіть каталог `D:\Students\1_kurs\TASM`, а в ньому підкаталог `PROGS`. В каталог `TASM` скопіюйте файли із вказаної викладачем папки. В каталозі `TASM` розміщуйте програму, призначену для компіляції, а в каталог `PROGS` складайте готові відлагоджені програми та їх тексти на асемблері. Програма на асемблері повинна мати розширення `.ASM`, наприклад: `hello.asm`.

Компіляцію асемблерних програм здійснюють у два етапи:

```
tasm hello
tlink hello
```

Можна використовувати командний файл `asm.bat`, тоді компіляція здійснюється однією командою `asm hello`.

Усі файли, які залишаються після створення і відлагодження вашої програми видаляйте або переносьте на зберігання в іншу папку, якщо вони вам потрібні. У разі використання бібліотеки `asmio16.lib`, скопіюйте у папку `TASM` файли `asmio16.lib` та `asmio16.inc`.

2. Наберіть і скомпілюйте EXE-програму на основі скелету зі спрощеними директивами сегментації helsim.asm.

3. Наберіть і скомпілюйте програму `lab1.asm` з використанням бібліотеки `asmio16.lib`.

4. Доповніть програму виведенням на екран цілого числа, шістнадцяткового числа.

Контрольні питання

1. Дайте визначення логічного та фізичного сегментів програми.
2. Які режими роботи підтримують процесори сімейства 80x86 з архітектурою IA-32?
3. Як виділяються сегменти у програмі на асемблері?
4. Чим відрізняються стандартні і спрощені директиви сегментації?
5. Що означає поняття «модель пам’яті» і які ви знаєте моделі пам’яті?
6. Чим відрізняються виконувані модулі програм COM та EXE?
7. Яка процедура одержання виконуваного модуля програми, написаної на мові асемблера?
8. Які ключі використовує програма TASM і яке їх призначення?
9. Які ключі використовує програма TLINK і яке їх призначення?
10. Що таке відлагодження програми і які засоби для цього існують?