Системне програмування, частина 1

Автор ст.викл. Мархивка В.С.

Практичне заняття № 1

АРХІТЕКТУРА PC

Основними елементами апаратних засобів комп'ютера з ядром 8086/8088 є: системний блок, клавіатура, пристрій відображення, дисководи, друкувальний пристрій (принтер) і різні засоби для асинхронної комунікації і керування ігровими програмами. Системний блок складається із системної плати, блоку живлення і слотів розширення для додаткових плат. На системній платі розміщені:

- мікропроцесор (Intel);
- постійна пам'ять (ROM 40Кбайт);
- оперативна пам'ять (RAM до 512К в залежності від моделі);
- розширена версія бейсик-интерпретатора.

Слоти розширення забезпечують підключення пристроїв відображення, дисководів для гнучких дисків (дискет), каналів телекомунікацій, додаткової пам'яті й ігрових пристроїв.

Клавіатура містить власний мікропроцесор, що забезпечує тестування при включенні пам'яті, сканування клавіатури, придушення "дребезга" клавішею і буферизацію до 20 символів.

"Мозком" комп'ютера є **мікропроцесор**, що виконує обробку всіх команд і даних. Процесор 8088 використовує 16-бітові регістри, що можуть обробляти два байти одночасно. Процесор 8088 схожий на 8086, але з одним розходженням: 8088 обмежений 8-бітовими (замість 16- бітових) шинами, що забезпечують передачу даних між процесором, пам'яттю і зовнішніми пристроями. Це обмеження співвідносить вартість передачі даних і виграш у простоті апаратної реалізації. Процесори 80286 і 80386 є розширеними версіями процесора 8086.

Як показано на рис. 1 процесор розділений на дві частини: операційний пристрій (ОП) і шинний інтерфейс (ШІ). Роль ОП полягає у виконання команд, у той час як ШІ підготовляє команди і дані для виконання. Операційний пристрій містить арифметико-логічний пристрій (АЛП), пристрій керування (ПК) і десять регістрів. Ці пристрої забезпечують виконання команд, арифметичні обчислення і логічні операції (порівняння на більше, чи менше дорівнює).

Три елементи шинного інтерфейсу (ШІ): пристрій керування шиною, черга команд і сегментні регістри здійснюють три важливі функції: по-перше, ШІ керує передачею даних на операційний пристрій, у пам'ять і на зовнішні пристрої вводу-виводу. По-друге, чотири сегментні регістри керують адресацією пам'яті обсягом до 1 Мбайта.

Третя функція ШІ це вибірка команд. Оскільки, усі програмні команди знаходяться в пам'яті, ШІ повинен мати доступ до них для вибірки їх у чергу команд. Оскільки черга має розмір 4 чи більш байт, у залежності від процесора, ШІ повинен "заглядати вперед" і вибирати команди так, щоб завжди існувала непорожня черга команд готових для виконання.

Операційний пристрій і шинний інтерфейс працюють паралельно, причому ШІ випереджає ОП на один крок. Операційний пристрій повідомляє шинному інтерфейсу про необхідність доступу до даних у пам'яті чи на пристрій вводу/виводу. Крім того ОП запитує машинні команди з черги команд. Поки ОП зайнятий виконанням першої в черзі команди, ШІ вибирає наступну команду з пам'яті. Ця вибірка відбувається під час виконання, що підвищує швидкість обробки.

Пам'ять

Зазвичай, мікрокомп'ютер має два типи внутрішньої пам'яті. Перший тип це постійна пам'ять (ПЗП) чи ROM (read-only memory). ROM являє собою спеціальну мікросхему, з якої (як це випливає з назви) можливо тільки читання. Оскільки дані в ROM спеціальним чином "пропалюють ся" вони не можуть бути модифіковані.

Основним призначенням ROM є підтримка процедур початкового завантаження: при включенні живлення комп'ютера ROM виконує різні перевірки і завантажує в оперативну пам'ять (RAM) дані із системної дискети (наприклад, DOS). Для програмування, найбільш важливим елементом ROM є BIOS (Basic Input/Output System) базова система вводу/виводу.

Пам'ять, з якою працює програміст, являє собою RAM (Random Access Memory) чи ОЗП, тобто оперативна пам'яті, доступна як для читання, так і для запису. RAM можна розглядати як робочу область для тимчасового збереження програм і даних на час виконання.

Оскільки, вміст RAM губиться при відключенні живлення комп'ютера, необхідна зовнішня пам'ять для збереження програм і даних. Якщо встановлена дискета з операційною системою чи є твердий диск типу вінчестер, то при включенні живлення ROM завантажує програми DOS у RAM. (Завантажується тільки основна частина DOS, а не повний набір програм DOS). Потім необхідно відповісти на запрошення DOS для установки дати і можна вводити запити DOS для виконання конкретних дій. Однією з таких дій може бути завантаження програм з диска в RAM. Оскільки DOS не займає всю пам'ять, то в ній є (звичайно) місце для користувацьких програм. Користувацька програма виконується в RAM і звичайно здійснює вивід на екран, принтер чи диск. По закінченні можна завантажити іншу програму в RAM. Попередня програма зберігається на диску і нова програма при завантаженні може затерти попередню програму в RAM.

Виділення пам'яті. Оскільки будь-який сегмент має обсяг до 64К и є чотири типи сегментів, те це припускає загальну кількість доступної RAM пам'яті: 4 x 64K = 256K. Але можливо будь-яка кількість сегментів. Для того, щоб адресувати інший сегмент, необхідно усього лише змінити адресу сегментного регістра.

RAM містить у собі перші три чверті пам'яті, а ROM - останню чверть. Відповідно до карти фізичної пам'яті мікрокомп'ютера, перші 256К RAM пам'яті знаходяться на системній платі. Оскільки одна область у RAM зарезервована для відеобуфера, то є 640К доступних для використання програмістом, принаймні в поточних версіях DOS. ROM починається за адресою 768К и забезпечує підтримку операцій вводу/виводу на такі пристрої як контролер твердого диска. ROM, що починається за адресою 960До керує базовими функціями комп'ютера, такими як тест при включенні живленні, точкові образи графічних символів і автозавантажувач з дискет.

Усі подальші згадки RAM використовують загальний термін - пам'ять.

Адресація

Усі комірки пам'яті пронумеровані послідовно від 00 - мінімальної адреси пам'яті. Процесор забезпечує доступ до байтів чи слів у пам'яті. Розглянемо десяткове число 1025. Для запису в пам'ять шістнадцяткового представлення цього числа - 0401 потрібно два байти чи одне слово. Воно складається зі старшої частини - 04 і молодшої частини - 01. Система зберігає в пам'яті байти слова в зворотній послідовності: молодша частина по меншій адресі, а старша - по більшій адресі. Припустимо, що процесор записав 0401h з регістра в комірки пам'яті 5612 і 5613, у такий спосіб:

|01| |04|

комірка 5612, комірка 5613

молодший байт старший байт

Процесор думає, що байти числових даних у пам'яті представлені в зворотній послідовності й обробляє їх відповідно. Незважаючи на те, що ця властивість цілком автоматизована, варто завжди пам'ятати про цей факт при програмуванні і відлагодженні асемблерних програм.

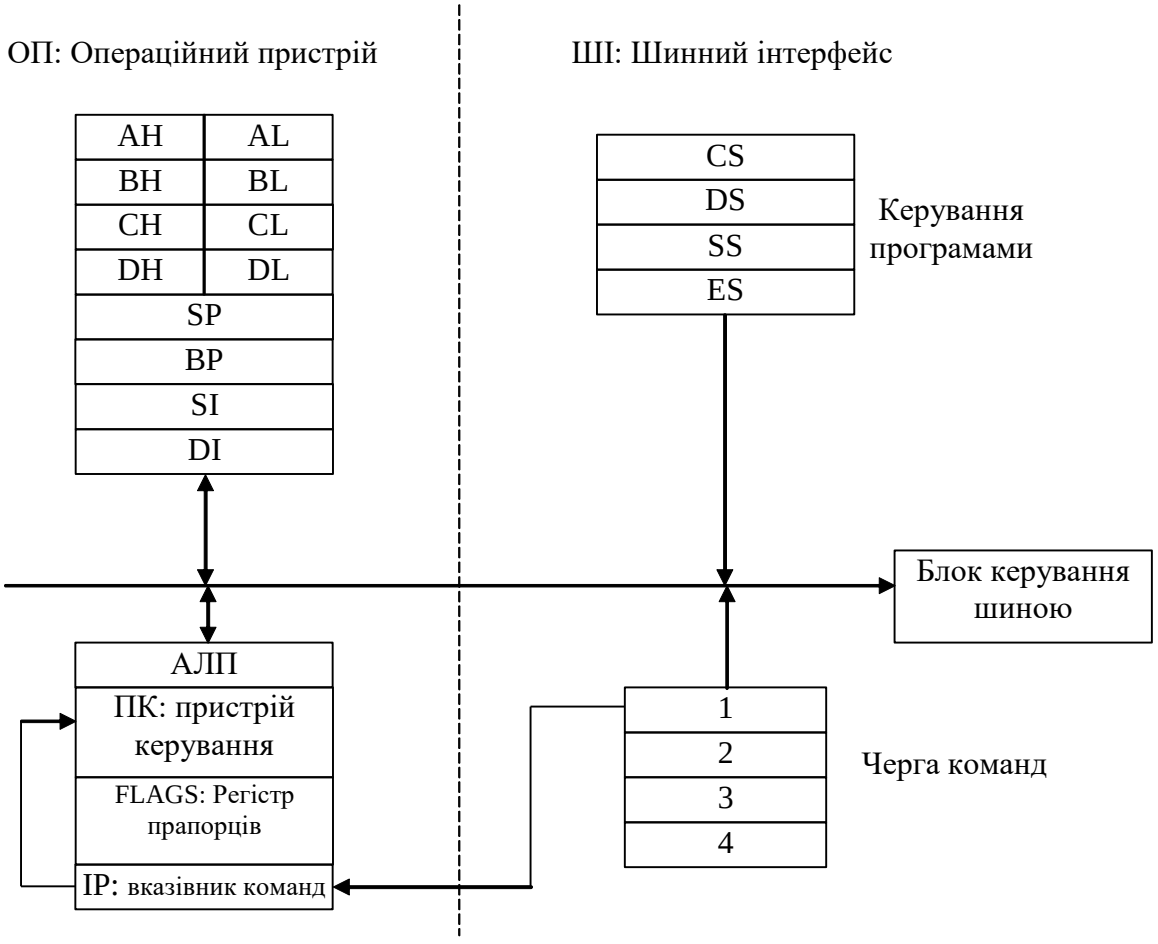


Рис.1. Логічна структура процесора.

Сегменти

Сегментом називається область, яка починається на границі параграфу, тобто за будь-якою адресою, що ділиться на 16 без остачі. Хоча сегмент може розташовуватися в будь-якому місці пам'яті та мати розмір до 64 Кбайт, він потребує стільки пам'яті, скільки треба для виконання програми. Є три основні сегменти:

1. Сегмент коду. Сегмент коду містить машинні команди, які будуть виконуватися. Зазвичай, команда, що має виконуватися першою, розташовується на початку цього сегменту і операційна система передає керування за адресою даного сегменту для виконання програми. Регістр сегменту коду (CS) адресує даний сегмент.

2. Сегмент даних. Сегмент даних містить визначені дані, константи і робочі області, необхідні програмі. Регістр сегмента даних (DS) адресує даний сегмент.

3. Сегмент стека. Стік містить адреси повернення як для програми для повернення в операційну систему, так і для викликів підпрограм для повернення в головну програму. Регістр сегмента стека (SS) адресує даний сегмент.

Ще один сегментний регістр, регістр **додаткового сегмента** (ES), призначений для спеціального використання. Три сегментних регістри містять початкові адреси відповідних сегментів і кожен сегмент починається на границі параграфа.

У середині програми всі адреси пам'яті відносні до початку сегмента. Такі адреси називаються зміщенням від початку сегмента. Двобайтове зміщення (16-бітне) може бути в межах від 0000h до FFFFh чи від 0 до 65535. Для звертання до будь-якої адреси в програмі, комп'ютер додає адресу в регістрі сегмента і зміщення. Наприклад, перший байт у сегменті кодів має зміщення 0, другий байт - 01 і так далі до зміщення 65535.

Як приклад адресації, допустимо, що регістр сегмента даних містить тичину. 045F і деяка команда звертається до комірки пам'яті усередині сегмента даних зі зміщенням 0032. Незважаючи на те, що регістр сегмента даних містить 045F, він указує на адресу 045F0, тобто на границі параграфа. Дійсна адреса пам'яті тому буде наступний:

Адреса в DS:	045F0
Зміщення:	0032
Реальна адреса:	04622

Яким чином процесори 8086/8088 адресують пам'ять в один мільйон байт? У регістрі міститься 16 біт. Оскільки адреса сегмента завжди на границі параграфа, молодші чотири біти адреси дорівнюють нулю. FFF0h дозволяє адресувати до 65520 (плюс зміщення) байт. Але фахівці вирішили, що нема сенсу зберігати біти, що завжди дорівнюють нулю. Тому адреса зберігається в сегментному регістрі як шіст. nnnn, а комп'ютер думає, що є ще чотири нульових молодших біти (одна шіст. цифра), тобто шіст. nnnn0. Таким чином, шіст. FFFF0 дозволяє адресувати до 1048560 байт. Якщо ви сумніваєтеся, то декодуйте кожне шіст. F як двійкове 1111, врахуйте нульові біти і додайте значення для одиничних біт.

Процесор 80286 використовує 24 біти для адресації так, що FFFFF0 дозволяє адресувати до 16 мільйонів байт, а процесор 80386 може адресувати до чотирьох мільярдів байт.

Регістри

Процесори 8086/8088 мають 14 регістрів, використовуваних для керування програмою, що виконується, для адресації пам'яті і для забезпечення арифметичних обчислень. Кожен регістр має довжину в одне слово (16 біт) і адресується по імені. Біти регістра прийнято нумерувати зліва направо:

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Процесори 80286 і 80386 мають ряд додаткових регістрів, деякі з них 16-бітові. Ці регістри тут не розглядаються.

Сегментні регістри CS, DS, SS і ES

Кожен сегментний регістр забезпечує адресацію 64Кб пам'яті, що називається поточним сегментом. Як показано раніше, сегмент вирівняний на границю параграфа і його адреса в сегментному регістрі припускає наявність праворуч чотирьох нульових бітів.

1. Регістр CS. Регістр сегменту коду містить початкова адреса сегмента коду. Ця адреса плюс величина зміщення у вказівнику команд (IP) визначає адресу команди, що повинна бути обрана для виконання. Для звичайних програм немає необхідності робити посилання на регістр CS.

2. Регістр DS. Регістр сегменту даних містить початкова адреса сегмента даних. Ця адреса плюс величина зміщення, визначена в команді, вказують на конкретну комірку у сегменті даних.

3. Регістр SS. Регістр сегменту стека містить початкову адресу в сегменті стека.

4. Регістр ES. Деякі операції над рядками використовують додатковий сегментний регістр для керування адресацією пам'яті. У даному контексті регістр ES зв'язаний з індексним регістром DI. Якщо необхідно використовувати регістр ES, асемблерна програма поовинна його ініціалізувати.

Регістри загального призначення: AX, BX, CX і DX

При програмуванні на асемблері регістри загального призначення є "робочими конячками". Особливість цих регістрів полягає в тому, що можлива адресація їх як одного цілого слова чи як однобайтової частини. Лівий байт є старшою частиною (high), а правий - молодшою частиною (low). Наприклад, двобайтовий регістр CX складається з двох однобайтових: CH і CL, і посилання на регістр можливі по кожному з цих трьох імен. Наступні три асемблерні команди засилають нулі в регістри CX, CH і CL, відповідно:

```
MOV  CX,00
MOV  CH,00
MOV  CL,00
```

1. Регістр AX. Регістр AX є основним суматором і застосовується для всіх операцій вводу-виводу, деяких операцій над рядками і деяких арифметичних операцій. Наприклад, команди множення, ділення і зсуву припускають використання регістра AX. Деякі команди генерують більш ефективний код, якщо вони мають посилання на регістр AX.

AX: | AH | AL |

2. Регістр BX. Регістр BX є базовим регістром. Це єдиний регістр загального призначення, що може використовуватися в якості "індексу" для розширеної адресації. Інше загальне застосування його - обчислення.

BX: | BH | BL |

3. Регістр CX. Регістр CX є лічильником. Він необхідний для керування кількістю повторень циклів і для операцій зсуву вліво чи вправо. Регістр CX використовується також для обчислень.

CX: | CH | CL |

4. Регістр DX. Регістр DX є регістром даних. Він застосовується для деяких операцій вводу-виводу і тих операцій множення і ділення над великими числами, які використовують регістрову пару DX і AX.

DX: | DH | DL |

Будь-які регістри загального призначення можуть використовуватися для додавання і віднімання як 8-ми, так і 16-ти бітових значень.

Регістрові вказівники покажчики: SP і BP

Регістрові вказівники SP і BP забезпечують системі доступ до даних у сегменті стека. Рідше вони використовуються для операцій додавання і віднімання.

1. Регістр SP. Покажчик стека забезпечує використання стека в пам'яті, дозволяє тимчасово зберігати адреси та іноді дані. Цей регістр зв'язаний з регістром SS для адресації стеку.

2. Регістр BP. Покажчик бази полегшує доступ до параметрів: даних і адрес переданих через стек.

Індексні регістри: SI і DI

Обоє індексних регістра можливі для розширеної адресації і для використання в операціях додавання і вирахування.

1. Регістр SI. Цей регістр є індексом джерела і застосовується для деяких операцій над рядками. У даному контексті регістр SI зв'язаний з регістром DS.

2. Регістр DI. Цей регістр є індексом призначення і застосовується також для рядкових операцій. У даному контексті регістр DI зв'язаний з регістром ES.

Регістр вказівника команд: IP

Регістр IP містить зміщення на команду, що повинна бути виконана. Зазвичай, цей регістр у програмі не використовується, але він може змінювати своє значення при використанні відлагоджувача DOS DEBUG для тестування програми.

Регістр прапорців

Дев'ять з 16 бітів регістра прапорців є активними і визначають поточний стан машини і результатів виконання. Багато арифметичних команд і команди порівняння змінюють стан прапорів. Призначення прапорців:

Прапор	Призначення
O (Переповнення)	Указує на переповнення старшого біта при арифметичних командах.
D (Напрямок)	Позначає лівий чи правий напрямок пересилання порівняння строкових даних (даних у пам'яті перевищуючих довжину одного слова).
I (Переривання)	Указує на можливість зовнішніх переривань
T (Покроковий режим)	Забезпечує можливість роботи процесора в покроковому режимі. Наприклад, програма DOS DEBUG установлює даний прапор так, що віз можна покрокове виконання кожної команди для перевірки зміни вмісту регістрів і пам'яті.
S (Знак)	Містить результуючий знак після арифметичних операцій (0 - плюс1 - мінус).
Z (Нуль)	Показує результат арифметичних операцій і операцій порівняння (0 - ненульовий, 1 – нульовий результат)
A (Зовнішній перенос)	Містить перенос з 3-го біта для 8-бітних даних, використовується для спеціальних арифметичних операцій.
P (Контроль парності)	Показує парність молодших 8-бітових даних (1 - парне і 0 - непарне число).
C (Перенос)	Містить перенос зі старшого біта, після арифметичних операцій, а також останній біт при зсувах і циклічних зсувах.

При програмуванні на асемблері найчастіше використовуються прапори O, S, Z, і C для арифметичних операцій і операцій порівняння, а прапор D для позначення напрямку в операціях над рядками.

Структура стандартної програми типу .EXE

Структура стандартної програми типу .EXE має наступний вигляд:

```
TITLE   Програма типу .EXE
STACK SEGMENT       PARA   STACK 'STACK'
                    db 200h DUP (?)
STACK ENDS

DATA  SEGMENT       WORD  'DATA'
      HelloMessage db 'Hello, world',13,10,'$'
DATA  ENDS

CODE  SEGMENT       WORD  'CODE'
      ASSUME        cs:CODE, ds:DATA
ProgramStart:
      push    ds                          ;initialize stack segment
      sub     ax,ax                       ;initialize stack segment
      push    ax                          ;initialize stack segment

      mov     ax,Data               ;initialize data segment
      mov     ds,ax                       ;initialize data segment

      mov     dx,OFFSET HelloMessage      ;DS:DX points to the HelloMessage
      mov     ah,09                       ;DOS string print function
      int     21h                         ;print the HelloMessage
      mov     ah,4Ch                ;DOS terminate program function
      int     21h                         ;end of the program
CODE  ENDS
      END     ProgramStart
```

Порядок запису сегментів у програмі на мові Turbo-Assembler може бути довільним.

Образ програми у пам'яті починається з префікса програмного сегмента (Program Segment Prefix, PSP), що утворюється та заповнюється системою. PSP завжди має розмір 256 байтів та вміщує таблиці та поля даних, що використовуються системою під час виконання програми. Після PSP розташовуються сегменти програми. Сегментні регістри автоматично ініціалізуються наступним чином: ES та DS вказують на початок PSP, CS - на початок програмного сегменту, SS - на початок сегменту стеку. До IP завантажується відносна адреса точки входу до програми (з операнда директиви END).

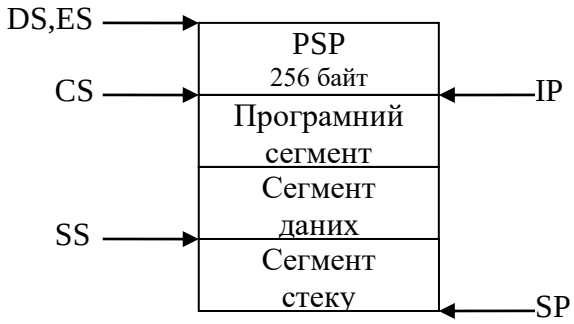
Додаткова ініціалізація та завершення:

Оскільки системний завантажувач використовує DS для встановлення початкової точки PSP, програма користувача повинна зберегти цю адресу, записавши значення DS до стеку. Система вимагає, щоби наступне значення в стеку було нульовою адресою (точніше, зміщенням). Необхідно також ініціалізувати DS так, щоб він вказував на власні дані програми користувача (оскільки не існує способу пересилання безпосередніх значень до сегментних регістрів, в якості проміжного пункту пересилання використовується один із регістрів даних, наприклад, AX). Послідовність операцій:

```
      push    ds                          ;store DS in stack
      sub     ax,ax                       ;zero AX
      push    ax                          ;store zero in stack

      mov     ax,Data               ;store Data segment address in DS
      mov     ds,ax
```

Образ пам'яті програми типу .EXE наведено на рисунку:



Розмір кожного сегменту не може перевищувати 64KB.

Команда RET використовує значення DS, що було збережено в стеку на початку програми, для повернення в DOS.

Розрізняють два стилі оформлення (запису) асемблерних програм - із стандартними та спрощеними сегментними директивами. Попередній приклад ілюструє застосування стандартних сегментних директив. Нижче наведено приклад із спрощеними сегментними директивами.

Приклад оформлення асемблерної програми у стилі .EXE - файла із спрощеними сегментними директивами:

```
DOSSEG
.MODEL SMALL
.STACK 100h
.DATA
        HelloMessage    db 'Hello, world', 13, 10, '$'
.CODE
        push    ds                              ;initialize stack segment
        sub     ax,ax
        push    ax
        mov     ax,@data
        mov     ds,ax                           ;set DS to point to the data segment
        mov     ah,9                            ;DOS print string function
        mov     dx,OFFSET HelloMessage          ;point to "Hello, world"
        int     21h                             ;display "Hello, world"
        mov     ah,4ch                  ;DOS terminate program function
        int     21h                             ;terminate the program
END
```

Система Turbo-Assembler продукує .EXE-файл в наступний спосіб:

C:\>TASM\tasm filename.asm <Enter>

Результат - файл filename.obj. Необов'язкові параметри /l або /la призводять до додаткової генерації файлу filename.lst, що містить лістинг асемблювання програми.

C:\>TASM\tlink filename.obj <Enter>

Результат - файл filename.exe, що вже можна запускати.

Структура стандартної програми типу .COM

Ще один вид виконуваного файлу – це програма, типу .**com.** Програма типу .COM вміщує лише один сегмент, де розташовані усі компоненти програми:

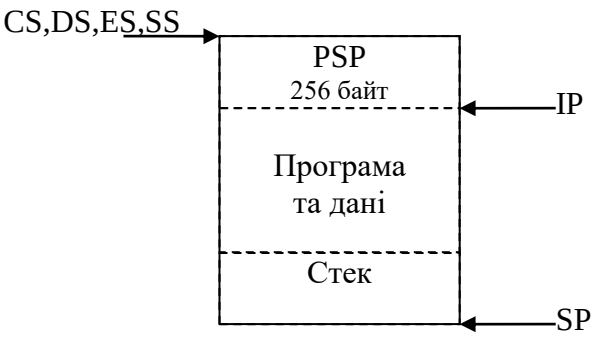
- префікс програмного сегмента;
- програмний код;
- дані;
- стек.

Структура типової програми має вигляд:

```
title    Програма типу .COM
text     segment 'code'
         assume  CS:text, DS:text, ES:text, SS:text
         org     100h
myproc   proc
         ...                ; текст прграми
myproc   endp
         ...                ; означення даних
text     ends
         end     myproc
```

Програма містить лише один сегмент text з класом 'CODE'. Оператор ASSUME затверджує, що усі чотири сегментні регістри вказують на цей єдиний сегмент. Оператор ORG 100h резервує 256 байтів для PSP (префікса програмного сегмента). Наповнювати PSP буде система. У програмі нема потреби ініціалізувати регістр DS, бо його, як всі інші сегментні регістри, ініціалізує система. Система у випадку .COM - файлу завжди задає IP=100h. Через це негайно за оператором ORG 100h повинен знаходитись перший виконуваний рядок програми. Якщо власні дані бажано розмістити на початку програми, тоді першою командою програми має бути jmp через ці дані.

Образ пам'яті програми типу .COM наведено нижче



До покажчика стека автоматично записується число FFFEh. Незалежно від реального розміру програми їй надається 64 K адресного простору, тобто один і тільки один сегмент. Стек зростає в напрямку зменшення адрес та за умови поганого планування програмістом може "наїхати (тобто затерти)" на власні дані та програму.

;Нижче наведено приклад із **спрощеними сегментними директивами**.

```
.MODEL TINY
.CODE
         org     100h
start:
         jmp prog
;власні дані, що потребують перестрибування
         HelloMessage DB 'Hello, world', 13, 10, '$'
prog:
         mov     ah,09
         mov     dx,OFFSET HelloMessage
         int     21h
         mov     ah,4ch
         int     21h
END start
```

Система Turbo-Assembler виробляє .COM-файл в наступний спосіб:

C:\>TASM\tasm filename.asm <Enter>

Результат - файл filename.obj. Необов'язкові параметри /l або /la призводять до додаткової генерації файлу filename.lst, що міститть лістинг асемблювання програми.

C:\>TASM\tlink /t filename.obj <Enter>

Результат - файл filename.com, що вже можна запускати.