**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМ**

**ЛАБОРАТОРНА РОБОТА №1.5.**

**Тема: Програмні сегменти**

Виконав:

Студент групи 3СТ Чума Д. І.

Перевірив:

Асист. Ваш Ю. В.

Ужгород 2022

**Мета роботи:** Навчитися розробляти повну програму мовою Асемблер.

1. **Теоретичні відомості**

Дотепер ми вважали, що в ПК використаються 16-розрядні адреси. Але настав час згадати, що обсяг пам'яті ПК дорівнює 220 байтів (1 Мб), а це вимагає 20-розрядних адрес. У даній лабораторній роботі розглядається, як наявність таких 20-розрядних адрес позначається на програмах і як виглядає повна програма на мові Асемблер.

1. **Сегментування адрес у ПК.**

Насамперед відзначимо, що в ПК замість терміна "базування адрес" використовується термін "сегментування адрес", а базові регістри називають сегментними регістрами. Крім того, розмір пам'яті в ПК дорівнює 1 Мб, тобто 220 байтів (до=20), тому абсолютні адреси тут 20-розрядні, а розміри сегментів не повинні перевищувати величину 64 Кб, тобто 216 байтів (m=1б), тому зсуву тут - це 16-розрядні адреси.

Перша особливість сегментування адрес у ПК полягає в тім, що як сегментні регістри можна використати не будь-які регістри, а лише регістри CS, DS, SS і ES. Звичайно, початкові адреси сегментів пам'яті можна зберігати й в інших регістрах (скажемо, в АХ), однак використати ці інші регістри для сегментування адрес не вдасться.

Оскільки в ПК усього чотири сегментних регістри, то одночасно можна працювати із чотирма сегментами пам'яті: установлюємо на початок кожного із цих сегментів свій регістр і далі сегментуємо по ньому всі адреси із цього сегмента. А що робити, якщо треба працювати з більшим числом сегментів пам'яті, скажемо, з п'ятьма? Коли буде потрібно доступ до осередків з п'ятого сегмента, вибираємо один із сегментних регістрів і десь рятуємо (у звичайному регістрі або комірці пам'яті) його вміст, а потім записуємо в цей регістр початок п'ятого сегмента й далі використовуємо цей регістр для доступу до осередків п'ятого сегмента. Пізніше, якщо треба, можна відновити колишнє значення цього регістра, щоб він знову вказував на один з перших 4 сегментів. Міняючи в такий спосіб значення сегментних регістрів, можна працювати з будь-яким числом сегментів пам'яті.

Розміри сегментів також визначаються автором програми, аби тільки ці розміри не перевершували 64 Кб. Установивши сегментний регістр на початок сегмента, ми потенційно можемо по ньому просегментувати 64 Кб, ну а скільки з них реально буде використано, тобто який розмір сегмента, уже визначає автор програми.

Тепер про другу особливість сегментування адрес у ПК. При записі команд на мові Асемблера посилання на сегментний регістр вказується за допомогою наступної конструкції:

**<сегментний регістр>:<адресне вираження>,**

яка називається адресною парою і яка "говорить", що адреса, що є значенням вираження, повинна бути просегментована по зазначеному регістру. Наприклад, у команді MOV AX,ES:X адреса змінної X буде сегментуватись по регістру ES. Запис CS:, DS:, SS: або ES: прийнятий називати префіксом сегментного регістра або просто префіксом (приставкою). У мові А префікс завжди записується перед адресою, що повинна бути просегментована. Однак у машинній мові ситуація трохи інша: тут префікс ставиться перед всією командою. Наприклад, команда MOV AX,ES:X насправді записується у вигляді двох машинних команд:

ES:

MOV АХ,Х

ES: - це спеціальна команда (без операндів), що і називається префіксом. Усього таких команд-префіксів чотири, по одній на кожний сегментний регістр. Сама по собі така команда-префікс нічого не робить, але вона впливає на наступну за нею команду: префікс "говорить", що адресний операнд у наступній команді повинен бути просегментований по відповідному регістру (у нашому прикладі - по регістру ES). Якщо префікс поставлений перед командою, де немає адресного операнда, то він виконується як "порожня" команда.

Отже, у машинних командах сегментні регістри вказуються не усередині команд, а перед ними (у вигляді команд-префіксів). Навіщо це зроблено, буде пояснено трохи пізніше. А поки відзначимо, що в мові Асемблера, записувати префікс поза командою не можна; наприклад, запис ES: MOV АХ,Х уважається помилкової з погляду мови Асемблера. Префікс обов'язково повинен вказуватися усередині команди але перед адресою.

Третя особливість сегментування адрес у ПК пов'язана з розміром сегментних регістрів. Загальна схема базування адрес припускає, що розміри базових регістрів досить більші й у них може бути розміщена будь-яка абсолютна адреса, будь-яка база. Однак у ПК ця умова не виконується: у ПК абсолютні адреси - 20-розрядні, а всі регістри, у тому числі й сегментні, 16-розрядні. Природно, виникає питання: як в 16-розрядних сегментних регістрах вдається розмістити 20- розрядні базові адреси?

У ПК ця проблема вирішена в такий спосіб. Ми дотепер уважали, що як базу можна використати будь-яку адресу. У ПК же на початкові адреси сегментів накладається обмеження: як базу можна використати будь-яку адресу, але кратну 16. Особливість цих адрес у тім, що в них останні 4 біти нульові, або, що те ж саме, у шістнадцятирічному записі цих адрес остання цифра завжди нульова, тобто вони мають вигляд XXXX0h, де X - будь-яка цифра. А раз так, то цю нульову цифру можна явно не вказувати, а лише мати на увазі. Так і роблять: у сегментних регістрах зберігають тільки перші 16 бітів початкової адреси сегмента, тобто перші чотири шістнадцятирічні цифри. Наприклад, якщо початком сегмента є адреса 12340h, то в сегментному регістрі буде зберігатися величина 1234Ь. Початкова адреса сегмента без останнього шістнадцятирічного 0 називають номером сегмента й позначають як seg.

Центральний процесор, природно, враховує цю особливість сегментних регістрів і при сегментуванні адреси насамперед до вмісту сегментного регістра приписує праворуч цей незазначений 0 і тільки потім додає зсув, зазначений у команді. Умовно це можна зобразити так:

**СР=ХХХХ ХХХХ0 - база**

**КІП ... CP:YYYY ... ==> + YYYY - зсув**

**ZZZZZ - абсолютна адреса**

Оскільки приписування праворуч нуля до шістнадцятирічного числа еквівалентно множенню цього числа на 16, то формулу обчислення абсолютної адреси по адресній парі CP:ofs можна виразити так:

**Аабс = 16\*[CP] + ofs**

Наприклад, якщо ES=1234h, тоді адресна пара ES:53h задає абсолютну адресу 16\*1234h+53h = 12340h+53h = 12387h.

Тепер уточнимо одну річ. Як ми знаємо, адреси, що вказують у командах, можна модифікувати по регістрах ВХ, ВР, SI і DI. Як сполучається ця модифікація із сегментуванням? Правила тут наступні. Спочатку виконується модифікація адреси по регістрах-модифікаторах, у результаті чого виходить адреса, яку ми називаємо виконавчою. При цьому, нагадаємо, обчислення ведеться по модулю 216, тобто виконавча адреса - це завжди 16-розрядна адреса. Потім ця адреса розглядається як зсув і саме він сегментується, тобто саме до нього додається вміст сегментного регістра, помножене на 16. Причому дане підсумовування ведеться по модулі 220, щоб не вийшла адреса, більше максимально припустимого. Таким чином, більш точна формула обчислення абсолютної адреси така:

**Аабс = (Авик + 16\*[СР]) mod 220**

Відзначимо, що сегментування виконавчої адреси відбувається, тільки якщо команда здійснює доступ до пам'яті. Якщо ж у команді не передбачене звертання до пам'яті, то сегментний регістр не враховується. Скажемо, команда LEA r16,А не звертається до пам'яті, тому адреса А не буде сегментуватися, тобто по цій команді виконується операція r16:=Авик, а не операція r16:=Аабс.

1. **Програмні сегменти.**

Розглянуті угоди про сегментні регістри змушують нас у такий спосіб будувати програму: в одному сегменті пам'яті треба розмістити всі команди програми й установити регістр CS на початок цього сегмента, в іншому сегменті треба розмістити всі дані програми й на його початок установити регістр DS, а 33 третій сегмент треба відвести під стек і встановити на його початок регістр SS. Але пропозиції нашої програми на мові А розміщає в пам'яті асемблер, а не ми. Як же асемблер визначає, які пропозиції в якому сегменті розміщати? Відповідь такої: він цього не знає, це повинні повідомити йому ми самі. І робиться це так: всі пропозиції, які ми хочемо розмістити в одному сегменті пам'яті, ми зобов'язані в тексті програми об'єднати в так званий програмний сегмент. Це аналог сегмента пам'яті, але в термінах мови А. Програмний сегмент має наступну структуру:

**<ім'я сегмента> SEGMENT <параметри> <пропозиція>**

**<пропозиція> <ім'я сегмента> ENDS**

Програмному сегменту дається ім'я, що повинне бути повторене двічі -у директиві SEGMENT, що відкриває сегмент, і в директиві ENDS (end of segment), що закриває його. Між цими директивами може бути зазначене будь-яке число будь-яких пропозицій. Зміст всієї цієї конструкції такий: ми заявляємо асемблеру, що всі пропозиції між директивами SEGMENT і ENDS він повинен розмістити в одному сегменті пам'яті. Але при цьому треба пам'ятати, що розмір сегмента пам'яті не повинен перевищувати 64 Кб, тому зазначених пропозицій повинне бути стільки, щоб у сукупності вони займали не більше 64 Кб, інакше асемблер зафіксує помилку.

Відзначимо, що в програмі може бути кілька сегментів з тим самим ім'ям. Уважається, що це один сегмент, просто він з якоїсь причини описаний вроздріб; всі пропозиції цих частин асемблер з'єднає разом.

Параметри директиви SEGMENT будуть розглянуті пізніше, а поки лише відзначимо, що параметри в основному потрібні тоді, коли програма більша й зберігається в декількох файлах, і параметри вказують, як поєднувати ці файли. Якщо ж програма невелика, уміщається в одному файлі, тоді параметри не потрібні.

Приклад програмних сегментів:

A SEGMENT

Al DB 400h DUP(?)

А2 DH 8

A ENDS

В SEGMENT

34

B1 DW A2

В2 DD A2

В ENDS

З SEGMENT

ASSUME ES:A, DS:B, CS:C

L: MOV AX,A2

MOV BX,B2

З ENDS

Асемблер приписує імені кожного програмного сегмента певне, значення. Їм є номер відповідного сегмента пам'яті, тобто перші 16 бітів (перші 4 шістнадцятирічні цифри) початкової адреси даного сегмента. У нашому прикладі імені А буде приписане значення l000h, імені В - значення 1041h, а імені З - значення 1042h. Зустрічаючи в тексті програми ім'я сегмента, асемблер буде заміняти його на дану величину. Наприклад, команда |

mov ах,b

буде сприйматися асемблером як команда

MOV AX,1041h ; АХ:<=номер сегмента В

Відзначимо особливо, що в мові А імена сегментів віднесені до константних виражень, а не до адресного. Тому наша команда записує в регістр АХ число 1041h, а не вміст осередку по такій адресі. (По цій же причині команда LEA АХ,У буде сприйматися як помилкова, тому що в LEA не можна вказувати константу.) Оператори OFFSET і SEG.

При трансляції пропозицій сегмента асемблер ставить у відповідність описаним у ньому іменам змінних і міткам адреси - зсуву відповідних осередків, відлічені від початку сегмента. У нашому прикладі ім'я А1 одержить зсув 0, ім'я А2 - зсув 400h, ім'я В1 - зсув 0, ім'я В2 - зсув 2, ім'я L - зсув 0. Саме на ці адреси (зсуву) асемблер і заміняє імена, коли зустрічає їх у тексті програми.

При цьому, нагадаємо, імена змінних і мітки розглядаються в мові А як адресні вираження. Але в мові є можливість розглядати їх і як константні вираження, для чого використається оператор OFFSET (зсув):

**offset <ім'я>**

Значенням оператора є зсув зазначеного імені, відлічене від початку того сегмента, у якому воно описано. Тому в нашому прикладі:

**OFFSET A1 = 0, OFFSET A2 - 400h, OFFSET B1 = 0**

При цьому значення даного оператора розглядається як константа, а не адреса. Тому маємо:

MOV АХ,А2; АХ:=уміст осередку А2 (АХ:=8)

MOV АХ,OFFSET А2; АХ:“зсув А2 (AX:=400h) = LEA AX,A2

Отже, А2 - це адреса, a OFFSET A2 - це константа, хоча по величині вони збігаються.

Розглянемо приклад, де корисний цей оператор. Нехай у регістрі ВХ перебуває адреса деякого елемента масиву А1, тобто BX=Al+i, і треба у ВХ записати індекс цього елемента, тобто i. Зробити це командою SUB BX,A1 не можна, т. до вона відніме із ВХ уміст початкового елемента масиву А1, а не адреса цього елемента (до того ж тут буде вирахування байта зі слова). Але от команда SUB BX,OFFSET А1 правильно вирішить нашу проблему, тому що із ВХ віднімемо саме адресу А1: ВХ:=Вх-адреса(А1)=1.

У мові А є також оператор, що дозволяє довідатися, у якому сегменті описане ім'я змінної або мітка. Це оператор SEG (сегмент):

**SEG <ім'я>**

Його значенням є ім'я того програмного сегмента, у якому описане ім'я, зазначене як операнд, а точніше - значення імені цього сегмента, тобто номер відповідного сегмента пам'яті. Тому в нашому прикладі:

**SEG A1 = SEG A2 = А = l000h SEG B1 = SEG B2 = В = 1041h SEG L = З = 1042h**

При цьому значення оператора SEG уважається константою:

**MOV BX,SEG B1; BX:=B=1041h (але не ВХ:=0)**

**2.2. Адресні змінні**

Тепер уточнимо, як асемблер обробляє опис змінних, як початкові значення яких зазначені адресні вираження. Такі змінні описуються за допомогою директив DW і DD. У нашому прикладі як початкове значення змінних В1 і В2 зазначене ім'я 36 змінної А2, тобто адреса цієї змінної. Але яка адреса - абсолютна або відносна тут мається на увазі? Правила тут такі: якщо або змінної або мітки (або адресне вираження, що містить таке ім'я або мітку) зазначено як операнд директиви DW, те асемблер заміняє ім'я на його зсув, але якщо ім'я зазначене в директиві DD, тоді асемблер заміняє його на адресну пару, що складається з номера сегмента, де описане це ім'я, і зі зсуву імені усередині даного сегмента (ця адресна пара визначає абсолютну адресу імені):

**Bl DW А2 еквівалентно Bl DW offset А2**

**В2 DD А2 еквівалентно В2 DD вод А2 : offset A2**

Таким чином, хоча в директиві DD ми вказуємо тільки ім'я й не вказуємо ніякого сегмента, асемблер сам визначає цей сегмент і приєднує його до зсуву. При цьому, у силу "переверненого" подання подвійних слів у пам'яті ПК, частина seg цієї пари виявляється в другому слові даного подвійного слова, а частина offset - у першому слові (В2: offset, B2+2: seg).

Відзначимо також, що в директиві DD як операнд можна вказати адресну пару наступного виду:

**<ім'я сегмента>:<адресне вираження> Наприклад:**

**DD А:В2**

У цьому випадку асемблер заміняє ім'я сегмента на його номер, а от ім'я змінної заміняє на його зсув, але відлічене не від початку того сегмента, де воно описано, а від початку зазначеного сегмента. Наприклад, у нашій директиві ім'я А буде замінене на l000h, а ім'я В2 - на 412h, тому що абсолютна адреса змінної В дорівнює 10412h, а абсолютна адреса початку сегмента А дорівнює l0000h: 10412h10000h=412h. І хоча подібні операнди припустимі, ними треба користуватися обережно, тому що відстань від початку сегмента А до змінної В2 з іншого сегмента може бути занадто більшим і перевершити максимально припустиму величину (64 Кб) для зсувів. Крім того при трансляції програми можлива перестановка її сегментів, і може трапитися так, що сегмент А виявиться розташованим у пам'яті після сегмента В, де описана змінна В2, і тоді зсув вийде негативним, що, звичайно, неприпустимо.

1. **Текст програми**

Варіант 17:

17. f = (11y + 7x - 2z) / (y + 1);

.386

INCLUDE Irvine32.inc

.model small

.stack 100h

.data

msg1 db "Input X: ", 0

msg2 db "Input Y: ", 0

msg3 db "Input Z: ", 0

x DWORD 0

y DWORD 0

z DWORD 0

f DWORD 0

.code

main proc

mov edx, offset msg1

call writestring

call readint

mov x, eax

mov edx, offset msg2

call writestring

call readint

mov y, eax

mov edx, offset msg3

call writestring

call readint

mov z, eax

; 17. f = (11y + 7x - 2z) / (y + 1);

mov eax, 11

imul y

xchg eax, ebx

mov eax, 7

imul x

xchg eax, ebx

add eax, ebx

mov ebx, 2

xchg eax, ebx

imul z

sub ebx, eax

mov ecx, y

mov eax, 1

add eax, ecx

xchg eax, ebx

idiv ebx

mov f, eax

call writeint

invoke exitprocess, 0

main endp

end main

1. **Результати роботи програми**

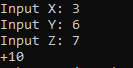


Рисунок 1. Результат роботи при значеннях x = 3, y = 6, z = 7

Перевіримо значення:

f = (11y + 7x – 2z) / (y + 1);

f = (11 \* 6 + 7 \* 3 – 2 \* 7) / (6 + 1);

f = (66 + 21 – 14) / 7;

f = 73 / 7 = 10.42;

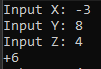


Рисунок 2. Результат роботи при значеннях x = -3, y = 8, z = 4

Перевіримо значення:

f = (11y + 7x – 2z) / (y + 1);

f = (88 – 21 – 8) / 9;

f = 59 / 9 = 6.55;

1. **Висновки**

На даній лабораторній роботі було ознайомлено з розробкою повної програми мовою Асемблер.

1. **Відповіді на контрольні питання**
2. Опишіть схему базування адреси в ПК.

Насамперед відзначимо, що в ПК замість терміна "базування адрес" використовується термін "сегментування адрес", а базові регістри називають сегментними регістрами. Крім того, розмір пам'яті в ПК дорівнює 1 Мб, тобто 220 байтів (до=20), тому абсолютні адреси тут 20-розрядні, а розміри сегментів не повинні перевищувати величину 64 Кб, тобто 216 байтів (m=1б), тому зсуву тут - це 16-розрядні адреси.

1. Вкажіть яку структуру має програмний сегмент.

Програмний сегмент має наступну структуру:

**<ім'я сегмента> SEGMENT <параметри> <пропозиція>**

**<пропозиція> <ім'я сегмента> ENDS**

Програмному сегменту дається ім'я, що повинне бути повторене двічі -у директиві SEGMENT, що відкриває сегмент, і в директиві ENDS (end of segment), що закриває його. Між цими директивами може бути зазначене будь-яке число будь-яких пропозицій. Зміст всієї цієї конструкції такий: ми заявляємо асемблеру, що всі пропозиції між директивами SEGMENT і ENDS він повинен розмістити в одному сегменті пам'яті. Але при цьому треба пам'ятати, що розмір сегмента пам'яті не повинен перевищувати 64 Кб, тому зазначених пропозицій повинне бути стільки, щоб у сукупності вони займали не більше 64 Кб, інакше асемблер зафіксує помилку.

1. Вкажіть для чого використовується оператор OFFSET.

**offset <ім'я>**

Значенням оператора є зсув зазначеного імені, відлічене від початку того сегмента, у якому воно описано. Тому в нашому прикладі:

**OFFSET A1 = 0, OFFSET A2 - 400h, OFFSET B1 = 0**

При цьому значення даного оператора розглядається як константа, а не адреса. Тому маємо:

MOV АХ,А2; АХ:=уміст осередку А2 (АХ:=8)

MOV АХ,OFFSET А2; АХ:“зсув А2 (AX:=400h) = LEA AX,A2

Отже, А2 - це адреса, a OFFSET A2 - це константа, хоча по величині вони збігаються.

Розглянемо приклад, де корисний цей оператор. Нехай у регістрі ВХ перебуває адреса деякого елемента масиву А1, тобто BX=Al+i, і треба у ВХ записати індекс цього елемента, тобто i. Зробити це командою SUB BX,A1 не можна, т. до вона відніме із ВХ уміст початкового елемента масиву А1, а не адреса цього елемента (до того ж тут буде вирахування байта зі слова). Але от команда SUB BX,OFFSET А1 правильно вирішить нашу проблему, тому що із ВХ віднімемо саме адресу А1: ВХ:=Вх-адреса(А1)=1.

1. Вкажіть для чого в використовується оператор SEG.

У мові є оператор, що дозволяє довідатися, у якому сегменті описане ім'я змінної або мітка. Це оператор SEG (сегмент):

**SEG <ім'я>**

Його значенням є ім'я того програмного сегмента, у якому описане ім'я, зазначене як операнд, а точніше - значення імені цього сегмента, тобто номер відповідного сегмента пам'яті. Тому в нашому прикладі:

**SEG A1 = SEG A2 = А = l000h SEG B1 = SEG B2 = В = 1041h SEG L = З = 1042h**

При цьому значення оператора SEG уважається константою:

**MOV BX,SEG B1; BX:=B=1041h (але не ВХ:=0)**