ЛАБОРАТОРНА РОБОТА №6

Складні структури даних.Структури. Об’єднання.

|  |
| --- |
|  |

***Мета:*** Ознайомлення з побудовою структур і об’єднань. Опис шаблона структури. Визначення данных с типом структури. Методи роботи зі структурою.

План:

1. Структури.
2. Методи роботи із структурою.
3. Опис записів.
4. Робота із записами.

Короткі теоретичні відомості

Структури

Розглянуті нами вище масиви є сукупністю однотипних елементів. Але часто в застосуваннях виникає необхідність розглядати деяку сукупність даних різного типу як деякий єдиний тип. Це дуже актуально, наприклад, для програм баз даних, де необхідно пов'язувати сукупність даних різного типу з одним об'єктом. Наприклад, вище ми розглянули лістинг 3, в якому робота проводилася з масивом трьохбайтових елементів. Кожен елемент, у свою чергу, був двома елементами різних типів: однобайтовим полем лічильника і двобайтовим полем, яке могло нести ще якусь потрібну для зберігання і обробки інформацію. Мовами високого рівня такий об'єкт зазвичай описується за допомогою спеціального типу даних – структури. З метою підвищити зручність використання мови асемблера в нього також був введений такий тип даних.

За визначенням **структура**— це тип даних, що складається з фіксованого числа елементів різного типу.

Для використання структур в програмі необхідно виконати три дії:

1. Задати **шаблон**структури. По сенсу це означає визначення нового типу даних, який згодом можна використовувати для визначення змінних цього типу.
2. Визначити **екземпляр**структури. Цей етап має на увазі ініціалізацію конкретної змінної із заздалегідь визначеною (за допомогою шаблону) структурою.
3. Організувати звернення до елементів структури.

Дуже важливо із самого початку з'ясувати, в чому різниця між *описом* структури в програмі і її **визначенням***.* Описати структуру в програмі означає лише вказати її схему або шаблон; пам'ять при цьому не виділяється. Цей шаблон можна розглядати лише як інформацію для транслятора про розташування полів і їх значення за умовчанням. Визначити структуру — означає дати вказівку транслятору виділити пам'ять і присвоїти цій області пам'яті символічне ім'я. Описати структуру в програмі можна тільки один раз, а визначити - будь-яку кількість разів.

Опис шаблону структури

Опис шаблону структури має наступний синтаксис:

ім’я\_структури struc

<опис полів>

ім’я\_структури ends

Тут **<описом полів>**є послідовність директив опису даних db, dw, dd, dq і dt. Їх операнди визначають розмір полів і при необхідності початкові значення. Цими значеннями будуть, можливо, ініціалізуватися відповідні поля при визначенні структури.

При описі шаблону пам'ять не виділяється, оскільки це всього лише інформація для транслятора. Місцеположення шаблону в програмі може бути довільним, але, слідуючи логіці роботи однопрохідного транслятора, він повинен бути розташований до того місця, де визначається змінна з типом даної структури. Тобто при описі в сегменті даних змінної з типом деякої структури її шаблон необхідно помістити на початку сегменту даних або перед ним.

Розглянемо роботу із структурами на прикладі моделювання бази даних про співробітників деякого відділу. Для простоти, щоб піти від проблем перетворення інформації при введенні, умовимося, що всі поля символьні. Опишемо структуру запису цієї бази даних:

Приклад 6.1.: Опис структури

.MODEL SMALL

animal struc ;створюємо структуру animal

cat dw 11 ;поле структури

dog dw 12 ;поле структури

animal ends ;кінець опису структури

.CODE

start:

mov ax, animal.dog ;запис поля dog зі структури animal

mov ds, ax

mov ax, 4c00h

int 21h

END start

Визначення даних з типом структури

Для використання описаної за допомогою шаблону структури в програмі необхідно визначити змінну з типом даної структури. Для цього використовується наступна синтаксична конструкція:

[ім'я змінної] ім'я структури <[список значень]>

Тут:

1. **ім'я змінної** *–* ідентифікатор змінної даного структурного типу. Завдання імені змінної необов'язкове. Якщо його не вказати, буде просто виділена область пам'яті розміром в суму довжин всіх елементів структури;
2. **список значень**– укладений в кутові дужки список початкових значень елементів структури, розділених комами. Його завдання також необов'язково. Якщо список вказаний не повністю, то всі поля структури для даної змінної ініціалізувалися значеннями з шаблону, якщо такі задані. Допускається ініціалізація окремих полів, але в цьому випадку пропущені поля повинні відділятися комами. Пропущені поля ініціалізують значеннями з шаблону структури. Якщо при визначенні нової змінної з типом даної структури ми згодні зі всіма значеннями полів в її шаблоні (тобто заданими за умовчанням), то потрібно просто написати кутові дужки. Наприклад: victor worker <>.

Для прикладу визначимо декілька змінних з типом описаної вище структури:

Приклад 6.2.: Вивести прізвище першого співробітника

model small

stack 256

.data

;описуємо шаблон структури

worker struc

nam db 30 dup (' ') ;прізвище, ім'я, по-батькові

sex db 'm' ;стать (по замовчуванню м - чоловіча)

position db 30 dup (' ') ;посада

age db 2 dup(' ') ;вік

standing db 2 dup(' ') ;стаж

salary db 4 dup(' ') ;зарплатня

birthdate db 8 dup(' ') ;дата народження

worker ends

;визначаємо данні для структури

sotrl worker <'Гурко Андрій Олегович',,'художник','33','15','1800','26.01.64'>

sotr2 worker <'Михайлова Наталія Генадіївна','ж','програміст','30','10','1680','27.10.58'>

sotr3 worker <'Степанов Юрій Сергійович',,'бухгалтер','38','20','1750','01.01.58'>

sotr4worker <> ;тут всі значення по замовуванню

.code

main:

mov ax,@data ;визначили сегмент даних

mov ds,ax ; і налаштували на нього DS

xor ax,ax ; обнулення ax

;виводимо прізвище першого співробітника

mov ah,40h

mov bx,1

mov dx,offset sotrl.nam

mov cx,30

int 21h

;виводимо його стать

mov ah,40h

mov bx,1

mov dX,offset sotrl.sex

mov cx,1

int 21h

;змінимо прізвище першого співробітника

mov sotrl.nam,'Д'

; і виведемо її

mov ah,40h

mov bx,1

mov dX,offset sotrl.nam

mov cx,30

int 21h

mov ax,4c00h

int 21h

end main

Методи роботи зі структурою

Ідея введення структурного типу в будь-яку мову програмування полягає в об'єднанні різнотипних змінних в один об'єкт. У мові мають бути засоби доступу до цих змінних всередині конкретного екземпляра структури. Для того, щоб посилатися в команді на поле деякої структури, використовується спеціальний оператор — символ «.» (крапка). Він використовується в наступній синтаксичній конструкції:

адресний\_вираз.ім’я\_поля\_структури

Тут:

1. **адресний вираз** – ідентифікатор змінної деякого структурного типу або вираз в дужках відповідно до вказаних нижче синтаксичних правил
2. **ім’я\_поля\_структури** – ім'я поля з шаблону структури. Це насправді теж адреса, а точніше, зсув поля від початку структури. Таким чином оператор **.** обчислює вираз: (адресний\_вираз) + (ім’я\_поля\_структури)

Продемонструємо на прикладі визначеної нами структури worker деякі прийоми роботи зі структурами. Наприклад, потрібно витягувати в ax значення поля з віком. Оскільки навряд чи вік працездатної людини буде більше величини 99 років, то після поміщення вмісту цього символьного поля в регістр ax буде зручно перетворити його в двійкове представлення командою aad. Будьте уважні, оскільки із-за принципу зберігання даних «молодший байт за молодшою адресою» старша цифра віку буде поміщена в al, а молодша – в ah.

Давайте уявимо, що співробітників не четверо, а набагато більше, і до того ж їх число і інформація про них постійно міняються. В цьому випадку втрачається сенс явного визначення змінних з типом worker для конкретних осіб. Мова асемблера дозволяє визначати не тільки окрему змінну з типом структури, але і масив структур. Наприклад, визначимо масив з 10 структур типу worker:

mas\_ robitnici worker 10 dup (<>).

Подальша робота з масивом структур проводиться так само, як і з одновимірним масивом. Тут виникає декілька питань. Як бути з розміром і як організувати індексацію елементів масиву?

Аналогічно іншим ідентифікаторам, визначеним в програмі, транслятор призначає імені типу структури і імені змінної з типом структури атрибут типу. Значенням цього атрибуту є розмір в байтах, займаний полями цієї структури. Витягувати це значення можна за допомогою оператора type. Після того, як став відомий розмір екземпляра структури, організувати індексацію в масиві структур не представляє особливої складності. Наприклад:

worker struc

- - -

worker ends

- - -

mas\_ robitnici worker 10 dup (<>)

- - -

mov bx,type worker ;bx=77

lea di,mas\_ robitnici

;витягти і вивести на екран ПІБ всіх співробітників:

mov сх,10

cycl:

mov dl[di].sex

... ;виведення на екран вмісту поля sex структури worker

add di,bx ;до наступної структури в масиві mas\_sort

loop cycl

Як виконати копіювання поля з однієї структури у відповідне поле іншої структури? Або як виконати копіювання всієї структури? Давайте виконаємо копіювання поля nam третього співробітника в поле nam п'ятого співробітника:

worker struc

worker ends

mas\_spivr worker 10 dup (<>)

mov bx,offset mas\_spivr

mov si(type worker)\*2 ;si=88\*2

add si,bx

mov di(type worker)\*4 ;si=88\*4

add di,bx

mov ex,30

repmovsb

Об’єднання

Представимо ситуацію, коли ми використовуємо певну область пам'яті для розміщення якого-небудь об'єкту програми (змінної, масиву або структури). Раптом після деякого етапу роботи у нас відпала потреба у використанні цих даних. Зазвичай пам'ять залишається зайнятою до кінця роботи програми.

Звичайно, в принципі, її можна було б використовувати для зберігання інших змінних, але при цьому без вживання спеціальних заходів не можна змінити тип і ім'я. Непогано було б мати можливість перевизначити цю область пам'яті для об'єкту з іншими типом і ім'ям. Мова асемблера надає таку можливість у вигляді спеціального типу даних, званого об'єднанням. **Об'єднання**– тип даних, що дозволяє трактувати одну і ту ж область пам'яті як що має різні типи і імена.

Опис об'єднань в програмі нагадує опис структур, тобто спочатку описується шаблон, в якому за допомогою директив опису даних перераховуються імена і типи полів:

ім’я\_об’єднання union

<опис полів>

ім'я об'єднання ends

Відмінність об'єднань від структур полягає, зокрема, в тому, що при визначенні змінної типу об'єднання пам'ять виділяється відповідно до розміру максимального елементу. Звернення до елементів об'єднання відбувається по їх іменах, але при цьому потрібно, звичайно, пам'ятати про те, що всі поля в об'єднанні накладаються одне на одного. Одночасна робота з елементами об'єднання виключена. В якості елементів об'єднання можна використовувати і структури.

Приклад, який ми зараз розглянемо, примітний тим, що, окрім демонстрації використання власне типу даних «об'єднання», в нім показується можливість взаємного вкладення структур і об'єднань. Основна ідея тут в тому, що покажчик на пам'ять, формований програмою, може бути представлений у вигляді:

1. 16-бітового зсуву;
2. 32-бітового зсуву;
3. пари з 16-бітового зсуву і 16-бітової сегментної складової адреси;
4. пари з 32-бітового зсуву і 16-бітового селектора.

Котрі з цих покажчиків можна застосовувати в конкретній ситуації, залежить від режиму адресації (use16 або use32) і режиму роботи мікропроцесора. Так от, описаний в прикладі шаблон об'єднання, дозволяє нам полегшити формування і використання покажчиків різних типів.

Приклад 6.1. Приклад використання об'єднання

masm

model small

stack 256

.586P

str struc ;структура str, що містить вкладене об'єднання

union ;опис вкладеного в структуру об'єднання

offs\_16 dw ?

offs\_32 dd ?

ends ;кінець опису об'єднання

segm dw ?

ends ;кінець опису структури

.data

point union ;визначення об'єднання, що містить

;вкладену структуру

off\_16 dw ?

off\_32 dd ?

point\_16 str <>

point\_32 str <>

point ends

tst db "Рядок для тестування"

adr\_data point <> ;визначення екземпляра об'єднання

.code

main:

mov ax @data

mov ds, ax

mov ax, seg tst

;записати адресу сегменту рядка tst в полі структури adr\_data

mov adr\_data.point\_16.segm, ax

;коли знадобиться можна витягувати значення з цього поля назад,

;наприклад, в регістр bx:

mov bx, adr\_data.point\_16.segm

;формуємо зсув в полі структури adr\_data

mov ax, offset tst ;зміщення рядка в ax

mov adr\_data.point\_16.offs\_16, ax

;аналогічно, коли знадобиться, можна витягувати значення з цього ;поля:

mov bx, adr\_data.point\_16.offs\_16

exit:

mov ax, 4c00h

int 21h

end main

Коли ви працюватимете в захищеному режимі мікропроцесора і використовуватимете 32-розрядні адреси, то аналогічним чином зможете заповнювати і використовувати описане вище об'єднання.

Опис запису

Опис шаблону запису має наступний синтаксис:

ім'я запису record <опис елементів>

Тут **<опис элементів>** є послідовністю описів окремих елементів запису згідно синтаксичній діаграмі.

При описі шаблону пам'ять не виділяється, оскільки це всього лише інформація для транслятора асемблера про структуру запису. Так само як і для структур, місцеположення шаблону в програмі може бути будь-яким, але при цьому необхідно враховувати логіку роботи однопрохідного транслятора.

Визначення екземпляра запису

Для використання шаблону запису в програмі необхідно визначити змінну з типом даного запису, для чого застосовується наступна синтаксична конструкція.

Якщо ініціалізувати поля не потрібно, то досить вказати ? при визначенні екземпляра запису:

- - -

iotest record i1:1,i2:2=11,i3:1,i4:2=11,i5:2=00

flag iotest?

- - -

Якщо ви складете і дослідите у відладчику тестовий приклад з даним визначенням запису, то побачите, що всі поля змінної flag типу запис обнуляються. Це відбувається не дивлячись на те, що у визначенні запису задані початкові значення полів.

Якщо потрібна часткова ініціалізація елементів, то вони покладаються в кутові (< і >) або фігурні ({ і }) дужки. Відмінність тут в тому, що в кутових дужках елементи мають бути задані в тому ж порядку, що і у визначенні запису. Якщо значення деякого елементу співпадає з початковим, то його можна не указувати, але обов'язково позначити його комою. Для останніх елементів, що йдуть підряд, коми можна опустити.

Наприклад, погодитися із значеннями за умовчанням можна так:

iotest record i1:l,і2:2=11,i3:1,i4:2=11,і5:2=00

- - -

flag iotest <> ;погодились зі значенням за умовчанням

Змінити значення поля i2 можна так:

iotest record i1:1,i2:2=11,i3:1,i4:2=11,i5:2=00

- - -

flag iotest <,10,> ; перевизначили i2

Застосовуючи фігурні дужки, також можна вказати вибіркову ініціалізацію полів, але при цьому необов'язково позначати комами поля, зі значеннями за умовчанням яких ми згодні:

iotest record i1:1,i2:2=11,i3:l,i4:2=11,i5:2=00

flag iotest {i2=10} ; перевизначили i2, не звертаючи уваги на ;порядок проходження інших компонентів запису

Робота з записами

Як організувати роботу з окремими елементами запису? Звичайні механізми адресації тут безсилі, оскільки вони працюють на рівні елементів пам'яті, тобто байтів, а не окремих бітів. Тут програмістові потрібно докласти деякі зусилля. Перш за все для розуміння проблеми потрібно засвоїти декілька моментів.

Кожному імені елементу запису асемблер присвоює числове значення, рівне кількості зсувів вправо, які потрібно провести для того, щоб цей елемент виявився «притиснутим» на початок комірки. Це дає нам можливість локалізувати його і працювати з ним. Але для цього потрібно знати довжину елемента в бітах. Зсув вправо проводиться за допомогою команди зсуву shr. Асемблер містить оператор width, який дозволяє дізнатися розмір елементу запису в бітах або повністю розмір запису. Варіанти застосування оператора width:

1. width **ім’я\_елемента\_запису** *–* значенням оператора буде розмір елементу в бітах;
2. width **ім’я\_экземпляра\_запису**або width і*м’я\_типу\_запису –* значенням оператора буде розмір всього запису в бітах.

mov al,width i2

- - -

mov ax,width iotest

Асемблер містить оператор mask, який дозволяє локалізувати біти потрібного елементу запису. Ця локалізація проводиться шляхом створення маски, розмір якої співпадає з розміром запису. У цій масці обнулені біти у всіх позиціях, за винятком тих, які займає елемент в запису.

Самі дії з перетворення елементів запису проводяться за допомогою логічних команд.

Тепер у нас є вся інформація про засоби асемблера для роботи із записами. Ви також зрозуміли, що безпосередньо звернутися до елементу запису неможливо. Щоб провести обробку елементу, що цікавить нас, потрібно спочатку виділити його, зсунути при необхідності до молодших розрядів, виконати необхідні дії і помістити назад на своє місце в запису. Тому, щоб не винаходити кожного разу велосипед, опишемо типові алгоритми здійснення цих операцій над елементами запису. Hаше завдання — закодувати ці алгоритми тим або іншим способом відповідно до вимог завдання.

Для виділення елементу запису належить:

1. помістити запис в тимчасову пам'ять — регістр (8-, 16- або 32-бітовий, залежно від розміру запису).
2. отримати бітову маску, відповідну елементу запису, за допомогою оператора mask.
3. локалізувати біти в регістрі за допомогою маски і команди and.
4. зсунути біти елементу до молодших розрядів регістра командою shr. Число розрядів для зсуву отримати з використанням імені елементу запису.

В результаті цих дій елемент запису буде локалізований на початку робочого регістра і далі з ним можна проводити будь-які дії (як, див. нижче).

Як ми вже з'ясували, з елементами запису проводяться будь-які дії, як над звичайною двійковою інформацією. Єдине, що потрібно відстежувати, – це розмір бітового поля. Якщо, наприклад, розмір поля збільшиться, то згодом може відбутися випадкова зміна сусідніх полів бітів. Тому бажано виключити зміну розміру поля.

Щоб помістити змінений елемент на його місце в запис, належить:

1. використовуючи ім'я елементу запису в якості лічильника зсувів, зсунути вліво біти елементу запису.
2. якщо ви не упевнені в тому, що розрядність результату перетворень не перевищила початкову, можна виконати «обрізання» зайвих бітів, використовуючи команду and і маску елементу.
3. підготувати початковий запис до вставки зміненого елементу шляхом обнулення бітів в запису на місці цього елементу. Це можна зробити шляхом накладення командою and інвертованої маски елементу запису на початковий запис.
4. за допомогою команди оr накласти значення в регістрі на початковий запис.

В якості прикладу розглянемо лістинг 1, який обнуляє поле i2 в запису iotest.

Приклад 6.2. Робота з полем запису

masm

model small

stack 256

iotest record i1:1,i2:2=11,i3:1,i4:2=11,i5:2=00

.data

flag iotest <>

.code

main:

mov ax,@data

mov ds,ax

mov al,mask i2

shr al,i2 ;біти i2 на початку ax

and al,0fch ;обнулили i2

;поміщається i2 на місце

shl al,i2

mov bl[flag]

xor bl,mask i2 ;скинули i2

or bl,al ;наклали

exit:

mov ax,4c00h ;стандартний вихід

int 21h

end main ;кінець програми

На закінчення ще раз проаналізуйте тип запису і особливості роботи з ним. При цьому зверніть увагу на ту обставину, що ми ніде явно не прораховуємо розташування бітів. Тому, якщо знадобиться змінити розмір елементу або його початкове значення, досить внести зміни в екземпляр запису або в опис його типу; функціональну частину програми, що працює з цим записом, чіпати не потрібно.

Додаткові можливості обробки

Розуміючи важливість для ефективного програмування такого типу даних, як запис, розробники транслятора TASM, починаючи з версії 3.0, включили в систему його команд дві додаткові команди на правах директив. Останнє означає, що ці команди зовні мають формат звичайних команд асемблера, але після трансляції вони приводяться до однієї або декількох машинних команд. Введення цих команд в мову TASM підвищує наочність роботи із записами, оптимізує код і зменшує розмір програми. Ці команди дозволяють приховати від програміста дії з виділення і установки окремих полів запису (ми їх обговорювали вище).

Для установки значення деякого поля запису використовується команда setfield з синтаксисом

Setfield ім’я\_елемента\_данного\_запису призначення, регістр\_джерело

Для вибірки значення деякого поля запису використовується команда getfield з синтаксисом

getfield ім’я\_елемента\_данного\_запису регістр\_призначення, джерело

Робота команди setfield полягає в наступному. Місцеположення запису визначається операндом **призначення***,* який може бути ім'ям регістра або адресою пам'яті. Операнд **ім’я\_елемента\_запису**визначає елемент запису, з яким ведеться робота (по суті, якщо ви були уважні, він визначає зсув елемента в запису щодо молодшого розряду). Нове значення, в яке необхідно встановити вказаний елемент запису, повинне міститися в операнді **регістр\_джерело***.* Обробляючи дану команду, транслятор генерує послідовність команд, які виконують наступні дії:

1. зсув вмісту **регістр\_джерело** вліво на кількість розрядів, відповідну розташуванню елементу в запису;
2. логічну операцію оr над операндами **призначення і регістр\_джерело***.* Результат операції поміщається в операнд **призначення***.*

Важливо відзначити, що setfield не проводить попереднього очищення елементу, в результаті після логічного додавання командою ог можливе накладення старого вмісту елементу і нового встановлюваного значення. Тому потрібно заздалегідь підготувати поле в запису шляхом його обнулення.

Команда getfield діє обернено setfield. В якості операнда **джерело**може бути вказаний або регістр, або адреса пам'яті. У регістр, вказаний операндом **регістр\_призначення***,* поміщається результат роботи команди — значення елементу запису. Цікава особливість пов'язана з регістром **регістр\_призначення***.* Команда getfield завжди використовує 16-бітовий регістр, навіть якщо ви вкажете в цій команді ім'я 8-бітового регістра.

В якості прикладу застосування команд setfield і getfield приклад

Приклад6.3.Робота з полями запису

masm

model small

stack 256

iotest record i1:1,i2:2=11,i3:1,i4:2=11,i5:2=00

.data

flag iotest <>

.code

main:

mov ax,@data

mov ds,ax

mov al,flag

mov bl,3

setfield i5 al,bl

xor bl,bl

getfield i5 bl,al

mov bl,1

setfield i4 al,bl

setfield i5 al,bl

exit:

mov ax,4c00h ;стандартний вихід

int 21h

end main ;кінець програми

У лістингу 2 демонструється порядок витягання і установки деяких полів запису. Результат роботи команд setfield і getfield найзручніше вивчати у відладчику. При установці значень полів не проводиться їх попереднє очищення. Це зроблено спеціально. Для такого роду операцій краще використовувати деякі універсальні механізми, інакше великий ризик внесення помилок, які важко виявити і виправити.

**Контрольні запитання**

* 1. Дайте визначення *запису* як складному типу даних.
  2. Які дії необхідно виконати для використання запису в програмі?
  3. Як виділяється пам'ять при описі шаблону запису?
  4. Які можливості дає використання оператора width?
  5. Який знак треба вказати, якщо не потрібно ініціалізувати поля при визначенні екземпляра запису?
  6. Які особливості визначення змінної з типом даного запису.
  7. Що треба зробити для виділення елементу запису?
  8. Для яких цілей використовуються команди setfield і getfield ?

**Завдання до лабораторної роботи №6**

**Перший рівень**

1.1.Набрати **програму 5.1**.

1.2.Завантажити програму в відладчик Виконати програмупо кроках.

Програма 6.1.

masm

model small

stack 256

.586P

str struc ;структура str, що містить вкладене об'єднання

union ;опис вкладеного в структуру об'єднання

offs\_16 dw ?

offs\_32 dd ?

ends ;кінець опису об'єднання

segm dw ?

ends ;кінець опису структури

.data

point union ;визначення об'єднання, що містить

;вкладену структуру

off\_16 dw ?

off\_32 dd ?

point\_16 str <>

point\_32 str <>

point ends

tst db "Рядок для тестування"

adr\_data point <> ;визначення екземпляра об'єднання

.code

main:

mov ax @data

mov ds, ax

mov ax, seg tst

;записати адресу сегменту рядка tst в полі структури adr\_data

mov adr\_data.point\_16.segm, ax

;коли знадобиться можна витягувати значення з цього поля назад,

;наприклад, в регістр bx:

mov bx, adr\_data.point\_16.segm

;формуємо зсув в полі структури adr\_data

mov ax, offset tst ;зміщення рядка в ax

mov adr\_data.point\_16.offs\_16, ax

;аналогічно, коли знадобиться, можна витягувати значення з цього ;поля:

mov bx, adr\_data.point\_16.offs\_16

exit:

mov ax, 4c00h

int 21h

end main

**Другий рівень**

Виконати пункт 1 із завдання згідно варіанту.

**Третій рівень**

Виконати пункти 1 та 2 із завдання. згідно варіанту

ЗАВДАННЯ. Реалізувати програму мовою асемблера.

Варіанти завдань:

1. 1.1 Сформувати записи, що містить інформацію про N поїздів, що відправляються з вокзалу станції м.Хмельницький (номер поїзду, станція призначення, час відправлення, час прибуття).

1.2. Надрукувати інформацію про поїзди, час в дорозі яких не перевищує 17 годин.

1. 2.1. Сформувати записи, що містить інформацію про N телевізорів (марка телевізора, його ціна, де випускається).

2.2. Надрукувати інформацію про найдорожчий телевізор.

1. 3.1 Сформувати записи, що містить прізвища N студентів та оцінки кожного студента по результатам 5 іспитів.

3.2 Надрукувати прізвища студентів, що мають найвищий бал.

1. 4.1 Сформувати записи, що містить прізвища N студентів та оцінки кожного студента по результатам 5 іспитів.

4.2 Надрукувати прізвища студентів, що здали всі іспити з оцінками «5».

1. 5.1 Сформувати записи-протокол лижних гонок, що містить прізвища N учасників, час старту, час фінішу для кожного учасника (години, хвилини, секунди).

5.2 Надрукувати прізвище учасника, що посів перше місце.

1. 6.1 Сформувати записи, що містить прізвища, стать, рік народження та групу крові N донорів.

6.2 Надрукувати прізвища донорів жіночої статі, мають 4-ту групу крові.

1. 7.1Сформувати записи, що містить прізвища, стать, групу крові та резус-фактори N донорів.

7.2 Надрукувати прізвища донорів, що мають першу групу крові та резус-фактор «-».

1. 8.1 Сформувати записи, що містить дані про N книг по програмуванню Вашої особистої бібліотеки(прізвище автора та його ініціали, назва книги, назва видавництва).
   1. Надрукувати прізвища авторів та назви книг видавництва «Наука».
2. 9.1 Сформувати записи, що містить дані про N книг по ЕОМ Вашої особистої бібліотеки (прізвище автора та його ініціали, назва книги, назва видавництва).

9.2 Надрукувати прізвища авторів та назви книг, видані не пізніше 2010р.

1. 10.1 Сформувати записи, що містить інформацію про N побутових магнітофонів (марка, де виробляється, ціна).

10.2 Надрукувати інформацію про магнітофони, що зроблено в Україні і з цною у заданих межах (межі вибираються самостійно).