

Лекція 6. Масиви

- *масив - структурований тип даних, що складає з деякого числа елементів одного типу.*



Масиви

- Для того щоб розібратися в можливостях і особливостях обробки масивів у програмах на асемблері, потрібно відповісти на наступні питання:
- Як описати масив у програмі?
- Як ініціалізувати масив, тобто як задати початкові значення його елементів?
- Як організувати доступ до елементів масиву?
- Як організувати масив с розмірністю більш однієї?
- Як організувати виконання типових операцій з масивами?

Опис і ініціалізація масиву в програмі

- Спеціальних засобів опису масивів у програмах асемблера, звичайно, немає. При необхідності використовувати масив у програмі його потрібно моделювати одним з наступних способів:

- 1. Перерахуванням елементів масиву в полі операндів однієї з директив опису даних.

- При перерахуванні елементи розділяються комами. Наприклад:

;масив з 5 елементів.Розмір кожного елемента 4 байти:

- `mas dd 1,2,3,4,5`

Опис і ініціалізація масиву в програмі

- 1. Перерахуванням елементів масиву в полі операндів однієї з директив опису даних.
- 2. Використовуючи оператор повторення `dup`.
Наприклад:
;масив з 5 нульових елементів.
;Розмір кожного елемента 2 байти:
`mas dw5 dup (0)`
- Такий спосіб визначення використовується для резервування пам'яті з метою розміщення й ініціалізації елементів масиву.

Опис і ініціалізація масиву в програмі

- 1. Перерахуванням елементів масиву в полі операндів однієї з директив опису даних.
- 2. Використовуючи оператор повторення dup.
- 3. Використовуючи директиви label і rept.
- Пара цих директив може полегшити опис великих масивів у пам'яті і підвищити наочність такого опису. Директива rept відноситься до макрозасобів мови асемблера і викликає повторення зазначеного числа разів рядків, розташованих між директивою і рядком endm.
- Прикладом, визначимо масив байт в області пам'яті, позначеної ідентифікатором mas_b. У даному випадку директива label визначає символічне ім'я mas_b, аналогічно тому, як це роблять директиви резервування й ініціалізації пам'яті.

Опис і ініціалізація масиву в програмі

- Перевага директиви `label` у тім, що вона не резервує пам'ять, а лише визначає характеристики об'єкта. У даному випадку об'єкт — це комірка пам'яті. Використовуючи кілька директив `label`, записаних одна за іншою, можна привласнити однієї і тій же області пам'яті різні імена і різний тип, що і зроблено в наступному фрагменті:

```
...  
n=0  
...  
mas_b label byte  
mas_w label word  
rept 4  
  dw 0f1f0h  
endm
```

Опис і ініціалізація масиву в програмі

- У результаті в пам'яті буде створена послідовність з чотирьох слів `f1f0`. Цю послідовність можна трактувати як масив чи байт слів у залежності від того, яке ім'я області ми будемо використовувати в програмі — `mas_b` чи `mas_w`.

Опис і ініціалізація масиву в програмі

- 1. Перерахуванням елементів масиву в полі операндів однієї з директив опису даних.
- 2. Використовуючи оператор повторення dup.
- 3. Використовуючи директиви label і rept.
- 4. Використання циклу для ініціалізації значеннями області пам'яті, яку можна буде згодом трактувати як масив.
- Подивимося на прикладі лістингу, яким чином це робиться.



MODEL small

STACK 256

.data

mesdb 0ah,0dh,'Масив- ','\$'

masdb 10 dup (?) ;вихідний масив

i db 0

.code

main:

mov ax,@data

mov ds,ax

xor ax,ax ;обнулення ax

mov cx,10

;значення лічильника циклу в cx

mov si,0

;індекс початкового елемента в si

go:

;цикл ініціалізації

mov bh,i ;i в bh

mov mas[si],bh ;запис у масив i

inc i ;інкремент i

inc si ;перехід до

; наступного елемента масиву

loop go ;повторити цикл
;виведення на екран
; отриманого масиву

mov cx,10

mov si,0

mov ah,09h

lea dx,mes

int 21h

show:

mov ah,02h

;функція виводу значення

; з al на екран

mov dl,mas[si]

add dl,30h

;перетворення числа в символ

int 21h

inc si

loop show

exit:

mov ax,4c00h ;стандартний вихід

int 21h

end main ;кінець програми

Доступ до елементів масиву

- При роботі з масивами необхідно чітко представляти собі, що всі елементи масиву розташовуються в пам'яті комп'ютера послідовно.
- Саме по собі таке розташування нічого не говорить про призначення і порядок використання цих елементів. І тільки лише програміст за допомогою складеного їм алгоритму обробки визначає, як потрібно трактувати цю послідовність байт, що складають масив.

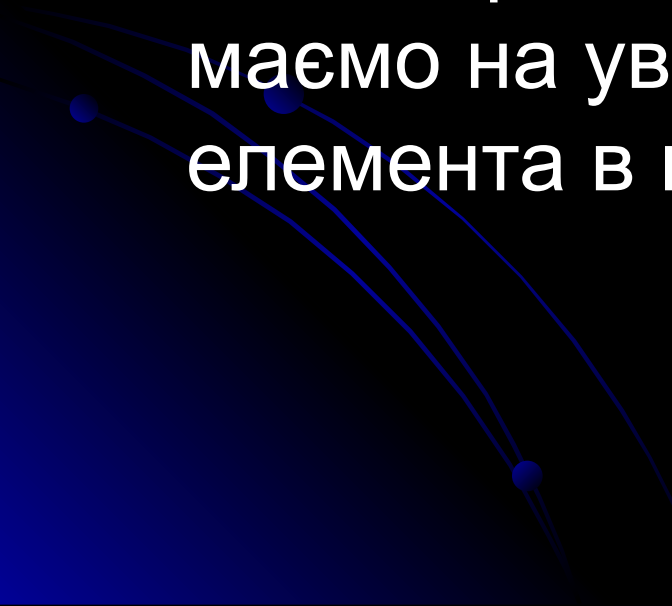
Доступ до елементів масиву

- Так, ту саму область пам'яті можна трактувати як одномірний масив, і одночасно ті ж самі дані можуть трактуватися як двомірний масив. Усе залежить тільки від алгоритму обробки цих даних у конкретній програмі. Самі по собі дані не несуть ніякої інформації про своєму “значенню”, чи логічному типу. Пам'ятайте про цей принциповий момент.

Доступ до елементів масиву

- Це розуміння можна також поширити і на *індекси елементів масиву*. Асемблер не підозрює про їхнє існування і йому абсолютно все рівно, які їх чисельні значення.
- Для того щоб локалізувати визначений елемент масиву, до його імені потрібно додати *індекс*. Тому що ми моделюємо масив, то повинні подбати і про моделювання індексу.

Доступ до елементів масиву

- У мові асемблера індекси масивів — це звичайні адреси, але з ними працюють особливим образом. Іншими словами, коли при програмуванні на асемблері ми говоримо про індекс, то скоріше маємо на увазі під цим не номер елемента в масиві, а деяку адресу.
- 

Доступ до елементів масиву

- Звернемося до опису масиву.
- У програмі визначена послідовність даних:

mas dw 0,1,2,3,4,5



Доступ до елементів масиву

- Нехай ця послідовність чисел трактується як одномірний масив. Розмірність кожного елемента визначається директивою `dw`, тобто вона дорівнює 2 байти. Щоб одержати доступ до третього елемента, потрібно до адреси масиву додати 6. Нумерація елементів масиву в асемблері починається з нуля.
- Тобто в нашому випадку мова, фактично, йде про 4-й елемент масиву — 3, про це знає тільки програміст; мікропроцесору в даному випадку все рівно — йому потрібна тільки адреса.

Доступ до елементів масиву

- У загальному випадку для одержання адреси елемента в масиві необхідно початкова (базова) адреса масиву скласти з добутком індексу (номер елемента мінус одиниця) цього елемента на розмір елемента масиву:

база + (індекс*розмір елемента)

Доступ до елементів масиву

- Архітектура мікропроцесора надає досить зручні програмно-апаратні засоби для роботи з масивами. До них відносяться базові й індексні реєстри, що дозволяють реалізувати кілька режимів адресації даних. Використовуючи дані режими адресації, можна організувати ефективну роботу з масивами в пам'яті. Згадаємо ці режими:
- *індексна адресація зі зсувом* — режим адресації, при якому ефективна адреса формується з двох компонентів:
 - *постійного (базового)* — вказівкою прямої адреси масиву у виді імені ідентифікатора, що позначає початок масиву;
 - *змінного (індексного)* — вказівкою імені індексного реєстра.

Доступ до елементів масиву

- Наприклад:

```
mas dw 0,1,2,3,4,5
```

```
...
```

```
mov si,4
```

;помістити 3-й елемент масиву mas у регістр ax:

```
mov ax,mas[si]
```

Доступ до елементів масиву

- **базова індексна адресація зі зсувом** — режим адресації, при якому ефективна адреса формується максимум із трьох компонентів:
 - постійного (необов'язковий компонент), у якості якої може виступати пряма адреса масиву у виді імені ідентифікатора, що позначає початок масиву, чи безпосереднє значення;
 - змінного (базового) — вказівкою імені базового регістра;
 - змінного (індексного) — вказівкою імені індексного регістра.
- Цей вид адресації зручно використовувати при обробці двомірних масивів.

Доступ до елементів масиву

- Мікропроцесор дозволяє масштабувати індекс. Це означає, що якщо вказати після імені індексного регістра знак множення “*” з наступною цифрою 2, 4 чи 8, то вміст індексного регістра буде збільшуватися у 2, 4 чи 8, тобто масштабуватися.
- Застосування масштабування полегшує роботу з масивами, які мають розмір елементів, рівний 2, 4 чи 8 байт, тому що мікропроцесор сам робить корекцію індексу для одержання адреси чергового елемента масиву. Нам потрібно лише завантажити в індексний регістр значення необхідного індексу (починаючи від 0).

```

MODEL    small
STACK256

.data    ;початок сегмента даних
;тексти повідомлень
mes1    db 'not equal 0!$',0ah,0dh
mes2    db ' equal 0!$',0ah,0dh
mes3    db 0ah,0dh,'Element $'
mas     dw 2,7,0,0,1,9,3,6,0,8;
        вихідний масив

.code

.486    ;це обов'язково
main:
    mov ax,@data
    mov ds,ax
;зв'язування ds із сегментом даних
    xor ax,ax ;обнулення ax
prepare:
    mov cx,10
;значення лічильника циклу в cx
    mov esi,0 ;індекс у esi

```

```

compare:
    mov dx,mass[esi*2]
;перший елемент масиву в dx
    cmp dx,0 ;порівняння dx с 0
    je equal
;перехід, якщо дорівнює
not_equal: ;не дорівнює
    mov ah,09h
;виведення повідомлення на екран
    lea dx,mes3
    int 21h
    mov ah,02h
; виведення № елемента масиву
    mov dx,si
    add dl,30h
    int 21h
    mov ah,09h
    lea dx,mes1
    int 21h
    inc esi
;на наступний елемент

```

```
dec cx
```

```
;умова для виходу з циклу
```

```
jcxz exit
```

```
;cx=0? Якщо так — на вихід
```

```
jmp compare
```

```
;ні — повторити цикл
```

```
equal:
```

```
;дорівнює 0
```

```
mov ah,09h
```

```
;виведення повідомлення mes3
```

```
lea dx,mes3
```

```
int 21h
```

```
mov ah,02h
```

```
mov dx,si
```

```
add dl,30h
```

```
int 21h
```

```
mov ah,09h
```

```
;виведення повідомлення mes2
```

```
lea dx,mes2
```

```
int 21h
```

```
inc esi
```

```
;на наступний елемент
```

```
dec cx
```

```
;всі елементи оброблені?
```

```
jcxz exit
```

```
jmp compare
```

```
exit:
```

```
mov ax,4c00h
```

```
;стандартний вихід
```

```
int 21h
```

```
end main
```

```
;кінець програми
```

Домовленості при використанні масивів

- Якщо для опису адреси використовується тільки один регістр, то мова йде про *базову адресацію* і цей регістр розглядається як *базовий*:

;переслати байт з області даних,

;адреса якої знаходиться в регістрі ebx:

`mov al,[ebx]`



Домовленості при використанні масивів

- Якщо для завдання адреси в команді використовується *пряма адресація* (у виді ідентифікатора) у сполученні з одним регістром, то мова йде про *індексну адресацію*. Регістр вважається *індексним*, і тому можна використовувати масштабування для одержання адреси потрібного елемента масиву:

```
add eax,mas[ebx*4]
```

;скласти вміст eax з подвійним словом у пам'яті

;за адресою mas + (ebx)*4

Домовленості при використанні масивів

- Якщо для опису адреси використовуються два регістри, то мова йде про *базово-індексну адресацію*. Лівий регістр розглядається як *базовий*, а правий — як *індексний*. У загальному випадку це не принципово, але якщо ми використовуємо масштабування з одним із регістрів, то він завжди є *індексним*.
- Пам'ятайте, що застосування регістрів **ebp/bp** і **esp/sp** за умовчанням має на увазі, що сегментна складова адреси знаходиться в регістрі **ss**.

Домовленості при використанні масивів

- Відмітимо, що *базово-індексну адресацію* не забороняється сполучити з *прямою адресацією* чи *вказівкою безпосереднього значення*. Адреса тоді буде формуватися як сума всіх компонентів.
- Наприклад:

```
mov ax,mas[ebx][ecx*2]
```

;адреса операнда дорівнює $[mas+(ebx)+(ecx)*2]$

...

```
sub dx,[ebx+8][ecx*4]
```

;адреса операнда дорівнює $[(ebx)+8+(ecx)*4]$

- Але майте на увазі, що масштабування ефективно лише тоді, коли розмірність елементів масиву дорівнює 2, 4 чи 8 байт. Якщо ж розмірність елементів інша, то організувати звертання до елементів масиву потрібно звичайним способом.

```
MODEL    small
STACK   256
.data
N=5
;кількість елементів масиву
mas    db 5 dup (3 dup (0))
.code
main:
    mov    ax,@data
    mov    ds,ax
    xor    ax,ax ;обнулення ax
    mov    si,0    ;0 у si
    mov    cx,N    ;N у cx
go: mov    dl,mass[si]
    ;перший байт в dl
    inc    dl
    ;збільшення dl на 1 (за умовою)
    mov    mass[si],dl
    ;повернути значення у масив
```

```
    add    si,3
    ;перехід на наступний елемент масиву
    loop   go
    ;повтор циклу
    mov    si,0
    ;підготовка до виводу на екран
    mov    cx,N
show:
    ;виведення на екран перших байт полів
    mov    dl,mass[si]
    add    dl,30h
    mov    ah,02h
    int    21h
    loop   show
exit:
    mov    ax,4c00h
    int    21h
end    main
```