Системне програмування

Лекція 2

1. Структура програми на асемблері
2. Директиви сегментації
3. Директиви визначення даних
4. Режими адресації даних
5. Команди переміщення даних

Структура програми

Основні частини

Сегмента модель

Сегменти

Фізично сегмент є областю пам'яті, зайняту командами і (або) даними.

Адреси сегментів зберігаються у відповідних сегментних регістрах.

Мікропроцесор має 6 сегментних регістрів, за допомогою яких може одночасно працювати:

- з одним сегментом коду; CS
- з одним сегментом стека; SS
- з одним сегментом даних; DS
- з трьома додатковими сегментами даних ES, FS, GS.

Директиви сегментації

Опис сегмента

Сегменти описуються за допомогою директиви SEGMENT

Спрощений синтаксис опису сегмента на асемблері має наступну структуру:
Ім’я_сегмента SEGMENT [параметри]
Тіло сегменту
Ім’я_сегмента ENDS

де **[параметри]** – [тип комбінування], [клас сегмента], [тип розміру сегмента]

Директиви сегментації

Приклади директив сегментації

Опис сегмента даних

```
data segment
    msg DB "Hello!$"
data ends
```

Опис сегмента коду

```
code segment
    begin:
    mov ax, data
    mov ds, ax
    <команди>
code ends
```

✔Порядок опису сегментів не має значення

Директиви сегментації

Функціональне призначення сегментів

За допомогою директиви ASSUME можна повідомити транслятору, який сегмент до якому сегментному регістру прив'язаний

Формат директиви:
ASSUME <сегментний регістр>: <ім'я сегмента>

приклад:
assume cs: code, ds: data

Директиви сегментації

Структура програми з трьома сегментами

ASSUME CS:code, DS:data	Прив'язка сегментів програми до сегментних регістрів
data SEGMENT msg DB “Hello!$" data ENDS	Сегмент даних
stk SEGMENT STACK DB 256 dup (?) stk ENDS	Сегмент стеку
code SEGMENT BEGIN: <команди> code ENDS	Сегмент коду
END BEGIN	Кінець програми

Директиви сегментації

Спрощений опис директив сегментації

Спрощений опис директив сегментації використовується для опису простих програм, що містять по одному сегменту для коду, даних і стека

Формат директиви	**Призначення**
.CODE [ім'я]	Початок або продовження сегмента коду
.DATA	Початок або продовження сегмента ініціалізації даних.
.STACK [розмір]	Початок або продовження сегмента стека модуля. Параметр [розмір] задає розмір стека

Директиви сегментації

Директива моделі пам'яті Model

- Використовується спільно з спрощеними директивами сегментації
- Частково керує розміщенням сегментів, тобто пов'язує сегменти з сегментними регістрами
- Спрощений формат директиви MODEL:

MODEL [<модифікатор>] <модель пам'яті> [ін. параметри]

Приклади:

```
model small
model tiny
model use 32 small
model compact
```

Директиви сегментації

Параметри директива Model

MODEL [<модифікатор>] <модель пам'яті> [ін. параметри]

Обов'язковим параметром директиви MODEL є **модель пам'яті**, що визначає модель сегментації пам'яті для програмного модуля.

- **TINY** – використовується для створення програм формату .com
- **SMALL** – цю модель зазвичай використовують для більшості програм на асемблері
- **MEDIUM** – код займає кілька сегментів, по одному на кожен програмний модуль, дані об'єднані в одній групі
- **COMPACT** – код в одному сегменті
- **LARGE** – код в декількох сегментах, по одному на кожен програмний модуль

Директиви сегментації

Модифікатори моделі пам'яті

MODEL [<модифікатор>] <модель пам'яті> [ін. параметри]

- **use16** - сегменти обраної моделі 16-бітові
- **use32** - сегменти обраної моделі 32-бітові
- **dos** - програма буде працювати в MS-DOS

Приклад:
model use16 small

Директиви сегментації

Ідентифікатори директиви MODEL

Ім'я ідентифікатора	Значення змінної
@code	Фізична адреса сегмента коду
@data	Фізична адреса сегмента даних типу near
@fardata	Фізична адреса сегмента даних типу far
@fardata?	Фізична адреса сегмента неініціалізованих даних типу far
@curseg	Фізична адреса сегмента неініціалізованих даних типу far
@stack	Фізична адреса сегмента стека

Директиви сегментації

Стандартні та спрощені директиви сегментації

Стандартні і спрощені директиви сегментації **не виключають** одна одну.

Стандартні директиви використовуються, коли програміст бажає отримати повний контроль над розміщенням сегментів у пам'яті і їх комбінуванням з сегментами інших модулів.

Спрощені директиви доцільно використовувати

- для простих програм
- програм, призначених для зв'язування з програмними модулями, написаними на мовах високого рівня.

Директиви опису даних

Ініціалізація даних

Директиви опису даних надають вказівки транслятору на виділення певного обсягу пам'яті.

Основні директиви:

- **DB** (define byte, визначити байт) визначаються дані розміром у 1 байт.
- **DW** (define word, визначити слово) описуються змінні розміром у слово (2 байти).
- **DD** (define double word, визначити подвійне слово) описуються змінні, під які відводяться подвійні слова (4 байти).

Директиви опису даних

Ініціалізація даних

Синтаксис цих директив має такий вигляд:

- [Ім'я] **DB** Вираз [, вираз...]
- [Ім'я] **DW** Вираз [, вираз...]
- [Ім'я] **DD** Вираз [, вираз...]

Ім'я – деяке символічне ім'я мітки або комірки пам'яті в сегменті даних, яке використовується в програмі.

Вираз – будь-яке ціле значення в проміжку:

- для **DB** – від -128 до 255;
- для **DW** – від -32768 до 65535;
- для **DD** – від -2^{31} до $2^{31}-1$.

Директиви опису даних

Ініціалізація даних: приклади

1. Ініціалізація простих змінних:

   а) змінна розміром у байт:

```
A DB 254        ; OFEh
B DB -2         ; OFEh (=256-2=254)
C DB 17h        ; 17h
```

   б) змінна розміром у слово:

```
B DW 1234h
C DB ?
D DW C
```

   в) змінна розміром у подвійне слово:

```
B DD 123456h
```

Директиви опису даних

Ініціалізація даних: приклади

2. Ініціалізація масивів:

   а) масив з 8 елементів типу «подвійне слово»:

```
DArray DD 0, 1, 2, 3, 4
       DD 5, 6, 7
```

   б) масив зі ста нулів:

```
WArray DW 100 DUP (0)
```

   в) масив з 50 кодів '0':

```
BArray DB 50 DUP ('0')
```

   г) масив з 19 будь-яких елементів:

```
SArray DW 19 DUP (?)
```

Директиви опису даних

Ініціалізація даних: приклади

3. Ініціалізація рядків:

```
String1 DB ‘A’, ‘B’, ‘C’, ‘D’
String2 DB ‘ABCD’
; String1 = String2

String3 DB ‘Line’, 0Dh, 0Ah, ‘$’
```

Адресація пам'яті

Ефективна та фізична адреса

Адреса, відносно початку сегмента (або зміщення), яку мікропроцесор використовує для доступу до даних у середині сегмента, називається **ефективною**.

Ефективна адреса = база + зміщення + індекс.

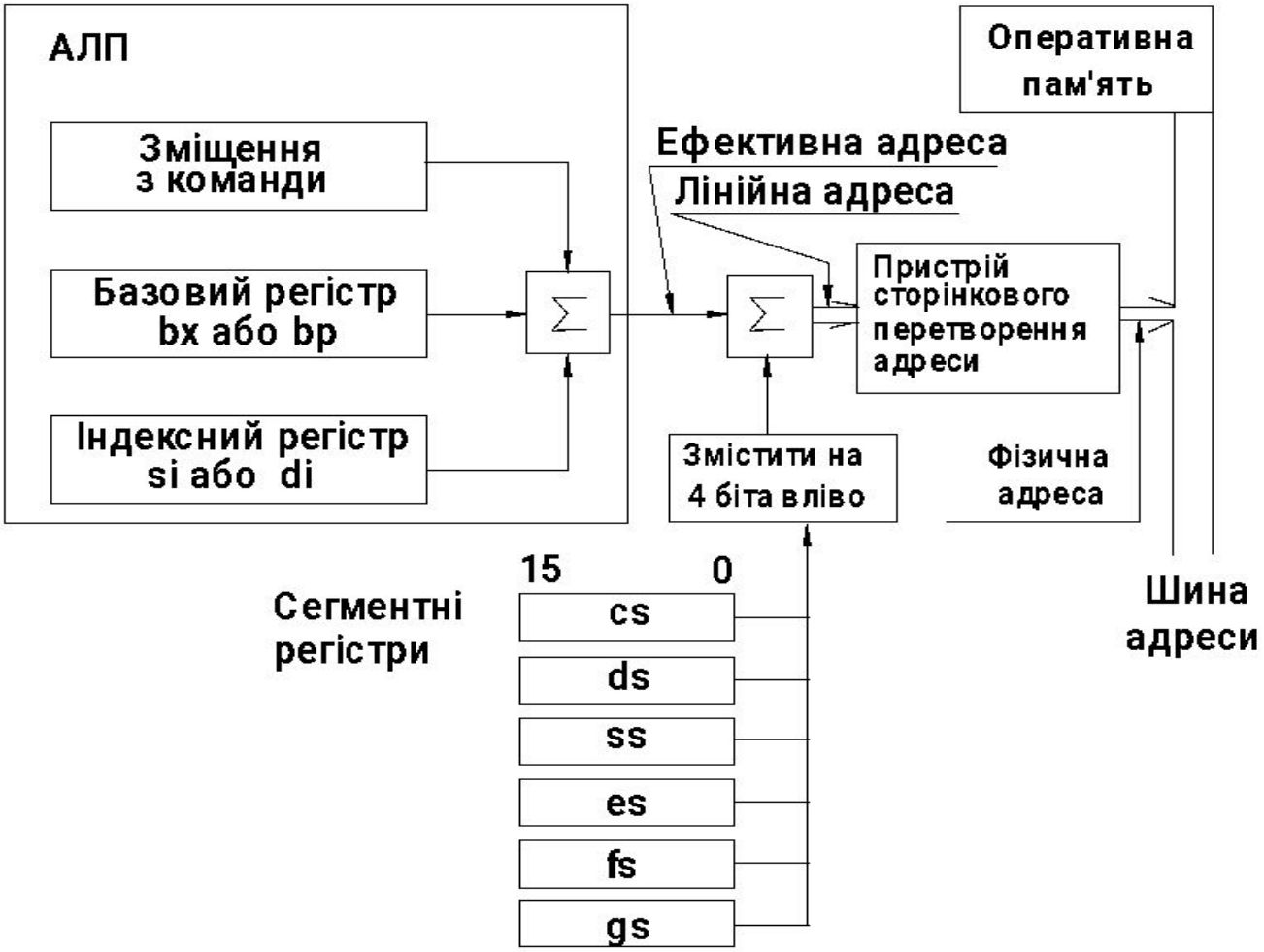
Фізична адреса - адреса пам'яті, яка видається на шину адреси мікропроцесора.

Інша назва - **лінійна адреса**.

Лінійна адреса = сегмент * 16 + Ефективна адреса

Адресація пам'яті

Механізм формування фізичної адреси



Режими адресації даних

Режими адресації

Адресації	Формат операнда	Приклад	Примітка
1. Регістрова	Регістр	mov ax, bx	Найбільш швидке виконання
2. Безпосередня	Дані	mov ax, 2	Застосовується в операціях з константам
3. Пряма	Лінійна адреса	mov ax, es:0001 mov ax, word_var	Застосовується для одноразового звернення до пам'яті

4. Непряма регістрова	[BX], [SI],
	[DI],
	[BP]
		mov ax, [bx]
		inc [di]
		mov cl, [bp]
			Застосовується при роботі з одновимірними масивам

Режими адресації даних

Режими адресації відносно бази

Адресації	Формат операнда	Приклад	Примітка

5. Базова зі зміщенням	[BX+зміщення],
	[SI+зміщення],
	[DI+зміщення],
	[BP+зміщення]
		mov ax, [bx+2]
		mov ax, [bp]+2
		mov ax, 2[bp]
			Застосовується для звернення до елементу структури, поч. адреса якої в BP або BX

6. Базова індексна	[BX+SI] [BX+DI] [BP+SI] [BP+DI]	mov [bx+di], dx	Застосовується при роботі с одновимірними масивами, зміщення - поч. адреса масиву
7. Базова індексна зі зміщенням	[BX+SI+зміщення] [BX+DI+зміщення] [BP+SI+зміщення] [BP+DI+зміщення]	mov ax, [bx+si+2] mov ax, [bx][si]+2	Застосовується при роботі з двовимірними масивами

Режими адресації даних

Приклади використання

- Регістрова:
  MOV DX, AX
  SUB CX, AX
- Безпосередня
  MOV AX, 5
  ADD BL, 0Ah
- Пряма
  MAS DB ‘HELLO’
  MOV AL, MAS ;AL=’H’
- Непряма регістрова
  ARR DB 1, 2, 3
  LEA BX, ARR
  MOV AL, [BX] ;AL=1

Режими адресації даних

Приклади використання

- Базова адресація зі зміщенням:

```
ARR DB 1, 2, 3
LEA BX, ARR
MOV AL, [BX+2]      ;AL=3
```

- Базова індексна адресація

```
ARR DB 1, 2, 3
LEA BX, ARR
MOV SI, 0
MOV AL, [BX][SI]          ;AL=1
```

- Базова індексна адресація зі зміщенням

```
MOV AL, [BX][SI]+2      ;AL=3
```

Команди пересилання даних

Загального призначення

Основна команда пересилання даних:

mov <операнд призначення>, <операнд-джерело>

Для двонаправленного пересилання даних

xchg <операнд1>,<операнд2>

Команди пересилання даних

Команда mov

Схема команди:

mov призначення, джерело

Призначення:

пересилання даних між регістрами або регістрами і пам'яттю.

Алгоритм роботи:

копіювання другого операнда в перший операнд.

Стан флагів після виконання команди:

виконання команди не впливає на флаги

Команди пересилання даних

Приклади

```
Par1  dw 100          ; 0000 – адрес Par1
Par2  dw 200          ; 0002 – адрес Par2
Par3  dd 10257h       ; 0004 – адрес Par3
```

```
mov  cx, Par1              cx      00 64
```

4_{16}

```
mov  bx, offset Par1       bx      00 00
```

```
mov  bx, offset Par2       bx      00 02
```

```
mov  ax, bx          ax      00 02          00 02
```

Команди пересилання даних

Приклади

```
Par1  dw 100        ; 0000 – адрес Par1
Par2  dw 200        ; 0002 – адрес Par2
Par3  dd 10257h     ; 0004 – адрес Par3
```

```
mov  cx, Par2
```

cx | 00 c8 | $0_{10}=c8_{16}$

```
mov  ch, 20h
```

cx | 20 c8

ch cl

Команди пересилання даних

Приклади

```
Par1  dw 100          ; 0000 – адрес Par1
Par2  dw 200          ; 0002 – адрес Par2
Par3  dd 10257h       ; 0004 – адрес Par3
```

```
mov  ax, offset Par2      ax
```

00 02

```
mov  al,5                 ax
                          ah   al
```

00 05

```
mov  ah,10+15             ax
```

19 05 19_{16}

```
mov  ax, -1               ax          -1
```

FF FF

Команди пересилання даних

Приклади помилок

```
Par1  dw 100        ; 0000 – адрес Par1
Par2  dw 200        ; 0002 – адрес Par2
Par3  dd 10257h     ; 0004 – адрес Par3
```

~~mov dh, Par1~~ *- constant too large*

⇒ **mov dx, Par1**

~~mov dh, 1254h~~ *- constant too large*

⇒ **mov dx, 1254h**

~~mov ah, Fh~~ *- Undefined symbol* (Fh – ідентифікатор, якого немає)

⇒ **mov ah, 0Fh**

Команди пересилання даних

Особливості застосування команди mov

1. **Не можна** здійснити пересилку з однієї області пам'яті в іншу

mov Par1, Par2

illegal memory reference
need register in expression

⇒ потрібно використовувати в якості проміжного буфера будь-який доступний в даний момент регістр загального призначення

mov ax, Par2
mov Par1, ax

Команди пересилання даних

Особливості застосування команди mov

фрагмент програми:

```
model   small
.data
x     db      5
y     db      ?
.code
start:
...
        mov     al,x
        mov     y,al
        ...
end     start
```

Команди пересилання даних

Особливості застосування команди mov

2. **Не можна** завантажити в сегментний регістр значення безпосередньо з пам'яті

mov ds, Param

⇒ для виконання такого завантаження потрібно використовувати проміжний об'єкт.

Це може бути регістр загального призначення або стек.

mov ax, Param
mov ds, ax

Команди пересилання даних

Особливості застосування команди mov

3. **Не можна** переслати вміст одного сегментного регістра в інший сегментний регістр.
   (В системі команд немає відповідної операції)

mov es, ds

⇒ використовувати в якості проміжних все ті ж регістри загального призначення

```
mov     ax, ds
mov     es, ax
```

Команди пересилання даних

Особливості застосування команди mov

4. **Не можна** використовувати сегментний регістр cs як операнд призначення.

mov cs, ax

mov cs, 100

Пара cs: ip завжди містить адресу команди, яка повинна виконуватися наступній

⇒ зміна командою mov вмісту регістра cs фактично означало б операцію переходу, а не пересилання, що неприпустимо

Команди пересилання даних

Особливості застосування команди mov

Рекомендовано використовувати в якості одного з операндів регістр al / ax / eax.

```
mov     al,5
mov     bl,al
```

в цьому випадку TASM генерує швидшу форму команди mov

Команди пересилання даних

Команда XCHG (eXCHanGe)

Схема команди:

xchg ax, bx

Призначення:

обміняти вміст регістрів ax і bx.

Фактично замінює 3 команди mov:

```
mov dx, ax
mov ax, bx
mov bx, dx
```

Команди пересилання даних

Особливості команди xchg

Операнди повинні мати один тип!

```
xchg    ax, bl
```

Не допускається (як і для всіх команд асемблера) обмінювати між собою вміст двох комірок пам'яті

```
xchg    Par1, Par2
```

Команди пересилання даних

Приклади

; змінити порядок слідування байт у слові

```
    ch1   dw     0f85ch          5c f8 ; [chl]=5c, [chl+1]=f8
...
          mov    al,ch1          al = 5c
          xchg   ch1+1,al        al = f8, [chl+1]=5c
          mov    ch1,al          [ch1]=f8
```

f8 5c