Лекція-2. Структура програми на асемблері

- Програма на асемблері являє собою сукупність блоків пам'яті, називаних *сегментами пам'яті*. Програма може складатися з одного чи декількох таких блоків-сегментів. Кожен сегмент містить сукупність речень мови, кожне з який займає окремий рядок коду програми.

Речення асемблера бувають чотирьох типів:

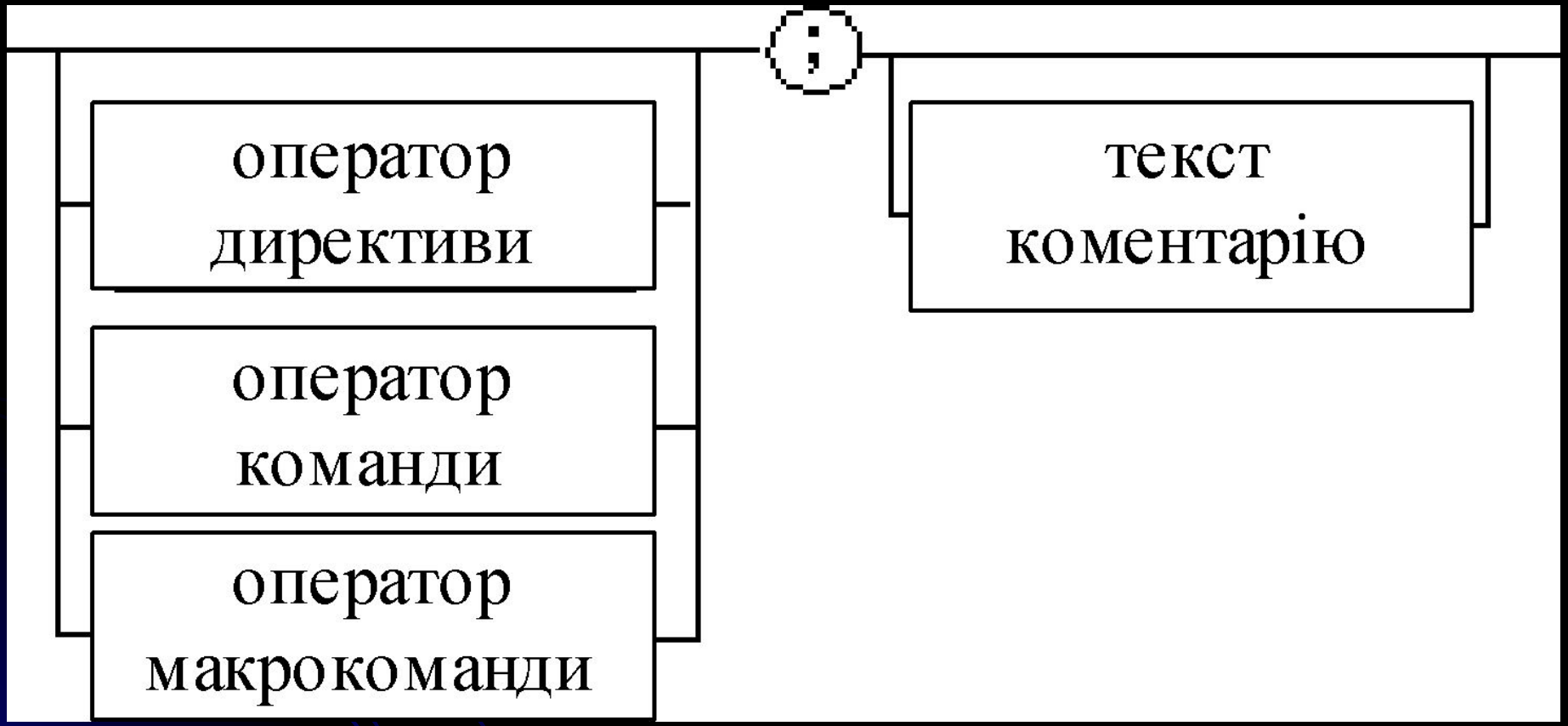
- *команди чи інструкції*, що представляють собою символічні аналоги машинних команд. У процесі трансляції інструкції асемблера перетворяться у відповідні команди системи команд мікропроцесора;
- *макрокоманди* — оформлювані певним чином речення тексту програми, що заміщаються під час трансляції іншими реченнями;
- *директиви*, що є вказівкою транслятору асемблера на виконання деяких дій. У директив немає аналогів у машинному представленні;
- *рядка коментарів*, що містять будь-які символи, у тому числі і букви російського алфавіту. Коментарі ігноруються транслятором.

Синтаксис асемблера

- Речення, що складають програму, можуть являти собою синтаксичну конструкцію, що відповідає команді, макрокоманді, чи директиві коментарю. Для того щоб транслятор асемблера міг розпізнати їх, вони повинні формуватися по визначених синтаксичних правилах. Для цього найкраще використовувати формальний опис синтаксису мови на зразок правил граматики. Найбільш розповсюджені способи подібного опису мови програмування — *синтаксичні діаграми* і *розширені форми Бекуса—Наура*. Для практичного використання більш зручні *синтаксичні діаграми*. Приміром, синтаксис речень асемблера можна описати за допомогою синтаксичних діаграм, показаних на наступних малюнках.

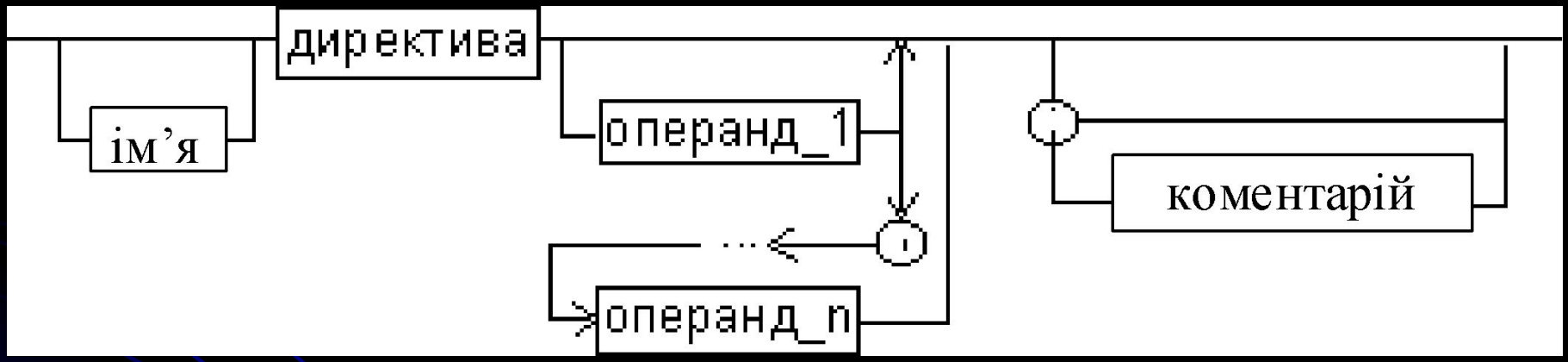
Синтаксис асемблера

Формат речення асемблера



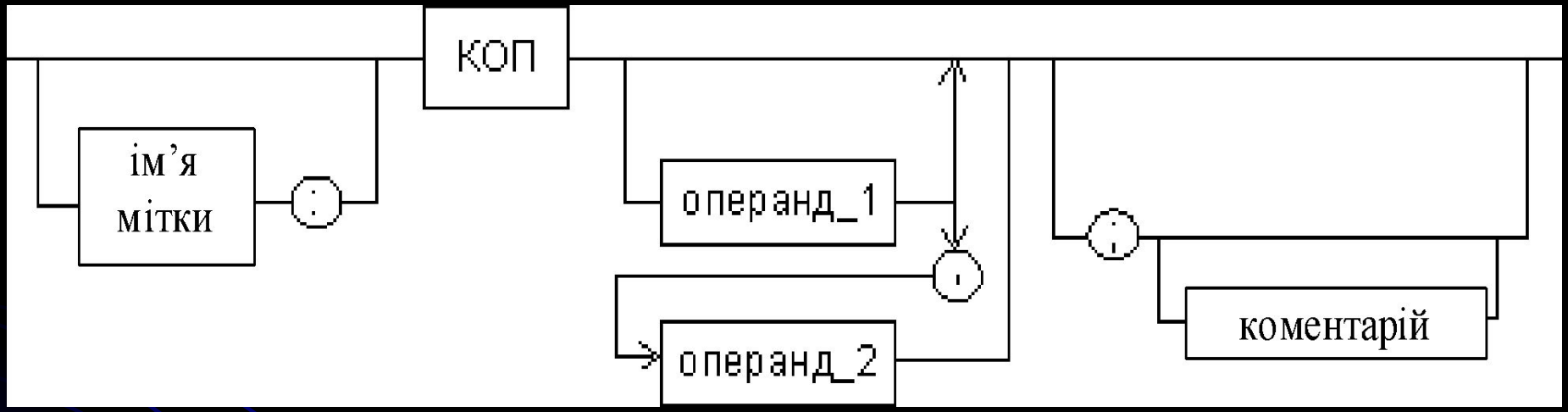
Синтаксис асемблера

Формат директив



Синтаксис асемблера

Формат команд и макрокоманд



Синтаксис асемблера

- *ім'я мітки* — ідентифікатор, значенням якого є адреса першого байта тієї Речення вихідного тексту програми, що він позначає;
- *ім'я* — ідентифікатор, що відрізняє дану директиву від інших однойменних директив. У результаті обробки асемблером визначеної директиви цьому імені можуть бути привласнені визначені характеристики;
- *код операції (КОП) і директива* — це мнемонічні позначення відповідної машинної команди, чи макрокоманди директиви транслятора;
- *операнди* — частини команди, чи макрокоманди директиви асемблера, що позначають об'єкти, над якими виробляються дії. Операнди асемблера описуються виразами з числовими і текстовими константами, мітками й ідентифікаторами перемінних з використанням знаків операцій і деяких зарезервованих слів.

Синтаксис асемблера

Припустимими символами при написанні тексту програм є:

- усі латинські букви: **A—Z**, **a—z**. При цьому заголовні і малі літери вважаються еквівалентними;
- цифри від **0** до **9**;
- знаки **?**, **@**, **$**, **_**, **&**;
- роздільники **, . [ ] ( ) < > { } + / * % ! ' " ? \ = # ^**.

- Речення асемблера формуються з *лексем*, що представляють собою синтаксично нероздільні послідовності припустимих символів мови, що мають зміст для транслятора.

Синтаксис асемблера

- Лексемами є:
- *ідентифікатори* — послідовності припустимих символів, що використовуються для позначення таких об'єктів програми, як коди операцій, імена перемінних і назви міток. Правило запису ідентифікаторів полягає в наступному: ідентифікатор може складатися з одного чи декількох символів. Як символи можна використовувати букви латинського алфавіту, цифри і деякі спеціальні знаки — _, ?, $, @. Ідентифікатор не може починатися символом цифри. Довжина ідентифікатора може бути до 255 символів, хоча транслятор сприймає лише перші 32, а інші ігнорує. Регулювати довжину можливих ідентифікаторів можна з використанням опції командного рядка **mv**. Крім цього існує можливість указати транслятору на те, щоб він розрізняв прописні і малі літери або ігнорував їхнє розходження (що і робиться за замовчуванням). Для цього застосовуються опції командного рядка **/mu, /ml, /mx**;
- *ланцюг символів* — послідовності символів, укладені в одинарних чи подвійних лапок;
- *цілі числа* в одній з наступних систем числення: *двійковій, десятковій, шістнадцятирічній*.

Синтаксис асемблера

Ототожнення чисел при записі їх у програмах на асемблері виробляється за визначеними правилами:

- **Десяткові числа** не вимагають для свого ототожнення вказівки яких-небудь додаткових символів, наприклад 25 чи 139.
- Для ототожнення у вихідному тексті програми **двійкових чисел** необхідно після запису нулів і одиниць, що входять у їхній склад, поставити латинське “**b**”, наприклад 10010101**b**.
- **шістнадцятирічні числа** мають більше умовностей при своєму записі.

Синтаксис асемблера

- Умовності при записі **шістнадцятирічних чисел** :
  - *По-перше*, вони складаються з цифр **0...9**, рядкових і прописних букв латинського алфавіту **a**, **b**, **c**, **d**, **e**, **f** чи **A**, **B**, **C**, **D**, **E**, **F**.
  - *По-друге*, у транслятора можуть виникнути труднощі з розпізнаванням шістнадцятирічних чисел через те, що вони можуть складатися як з одних цифр 0...9 (наприклад 190845), так і починатися з букви латинського алфавіту (наприклад **ef15**). Для того щоб "пояснити" транслятору, що дана лексема не є десятковим числом чи ідентифікатором, програміст повинний спеціальним чином виділяти шістнадцятирічне число. Для цього на кінці послідовності шістнадцятирічних цифр, що складають шістнадцятирічне число, записують латинську букву “**h**”. Це обов'язкова умова. Якщо шістнадцятирічне число починається з букви, то перед ним записується ведучий нуль: **0**ef15**h**.

Синтаксис асемблера

- Практично кожне речення містить опис об'єкта, над яким чи за допомогою якого виконується деяка дія. Ці об'єкти називаються операндами.
- ***ОПЕРАНДИ*** *— це об'єкти (деякі значення, регістри чи комірки пам'яті), на які діють інструкції чи директиви, або це об'єкти, що чи визначають уточнюють дію інструкцій чи директив.*
- Операнди можуть комбінуватися з арифметичними, логічними, побітовими й атрибутивними операторами для розрахунку деякого значення чи визначення комірки пам'яті, на яку буде впливати дана команда чи директива.

Класифікація операндів

- · постійні, чи безпосередні, операнди
- · адресні операнди
- · переміщувані операнди
- · лічильник адреси
- · реєстровий операнд
- · базовий і індексний операнди
- · структурні операнди

Операнди

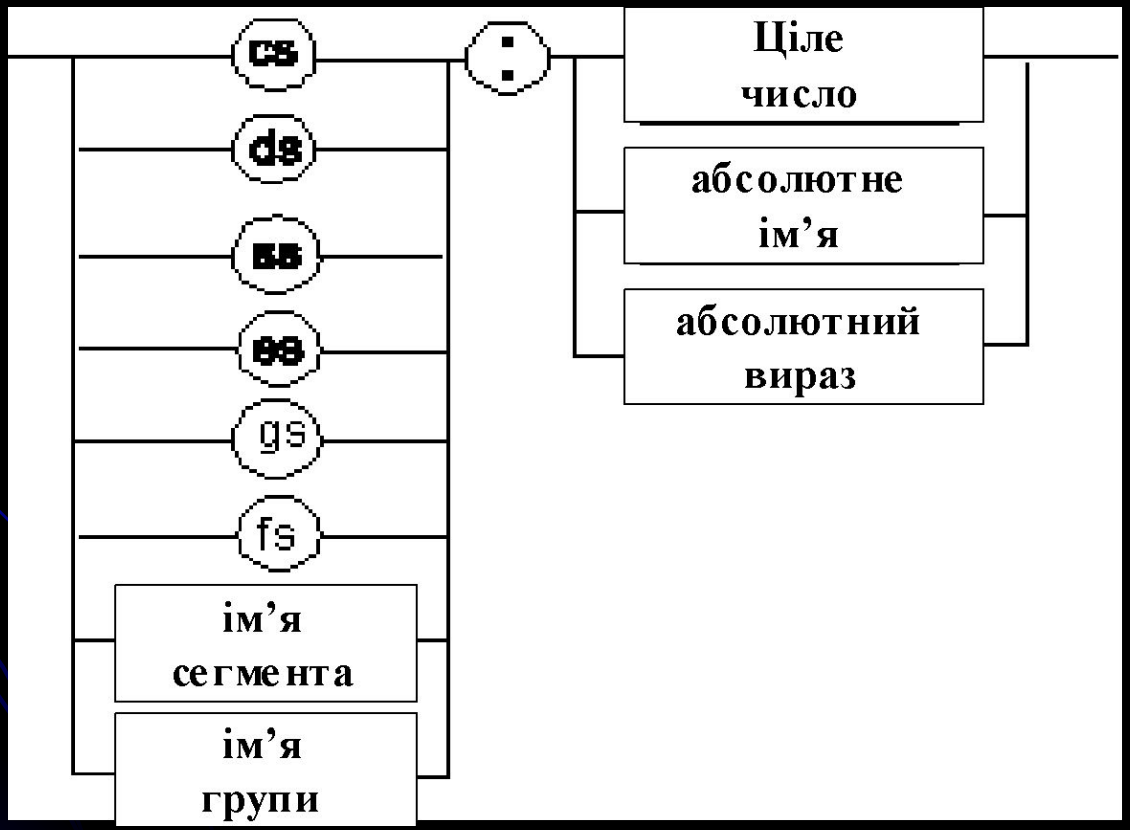
- *Постійні чи безпосередні операнди* — число, рядок, чи ім'я вираження, що мають деяке фіксоване значення. Ім'я не повинне бути переміщуваним, тобто залежати від адреси завантаження програми в пам'ять. Наприклад, воно може бути визначено операторами **equ** чи **=**.

```
num     equ     5
imd = num-2
mov     al,num  ;еквівалентно mov al,5
;5 тут безпосередній операнд
add     [si],imd        ; imd=3 - безпосередній операнд
mov     al,5            ;5 - безпосередній операнд
```

- У даному фрагменті визначаються дві константи, що потім використовуються в якості безпосередніх операндів у командах пересилання mov і додавання add.

Адресні операнди

- *Адресні операнди* — задають фізичне розташування операнда в пам'яті за допомогою вказівки двох складових адрес: *сегмента* і *зсуву*



Адресні операнди

```
mov     ax,0000h
mov     ds,ax
mov     ax,ds:0000h   ;записати слово в ax з області
                      ;пам'яті по фізичній адресі 0000:0000
```

Тут третя команда mov має адресний операнд.

Переміщувані операнди

- *Переміщувані операнди* — будь-які символьні імена, що представляють деякі адреси пам'яті. Ці адреси можуть позначати місце розташування в пам'яті деяких інструкції (якщо операнд — мітка) чи даних (якщо операнд — ім'я області пам'яті в сегменті даних).
- Переміщувані операнди відрізняються від адресних тим, що вони не прив'язані до конкретної адреси фізичної пам'яті. Сегментна складова адреси переміщуваного операнда невідома і буде визначена після завантаження програми в пам'ять для виконання.

Переміщувані операнди

```
data    segment
mas_w   dw      25 dup (0)
…
code    segment
…
lea     si,mas_w        ;mas_w - переміщуваний операнд
```

- У цьому фрагменті *mas_w* — символьне ім'я, значенням якого є початкова адреса області пам'яті розміром 25 слів. Повна фізична адреса цієї області пам'яті буде відома тільки після завантаження програми в пам'ять для виконання.

Лічильник адреси

Лічильник адреси — специфічний вид операнда. Він позначається знаком **$**. Специфіка цього операнда в тім, що коли транслятор замість нього поточне значення лічильника адреси. Значення лічильника адреси, чи, як його іноді називають, *лічильника розміщення*, являє собою зсув поточної машинної команди відносно початку сегмента коду. У форматі лістінгу лічильнику адреси відповідає другий чи третій стовпчик (у залежності від того, є присутнім чи ні в лістінгу стовпчик з рівнем вкладеності). Якщо взяти в якості приклад будь-який лістінг, то видно, що при обробці транслятором чергової команди асемблера лічильник адреси збільшується на довжину сформованої машинної команди. Важливо правильно розуміти цей момент.

Домашнє завдання

Проаналізувати використання наступних операндів:

Базовий і індексний

Структурний

Запису

Оператори

Типи операторів асемблера і синтаксичні правила формування виразів асемблера.

- · Арифметичні оператори
- · Оператори зсуву
- · Оператори порівняння
- · Логічні оператори
- · Індексний оператор
- · Оператор перевизначення типу
- · Оператор перевизначення сегмента
- · Оператор іменування типу структури
- · Оператор одержання сегментної складової адреси виразу
- · Оператор одержання зсуву виразу

Оператори і їхній пріоритет

Оператор	Пріоритет
length, size, width, mask, (,), [,], <, >	1
.	2
:	3
ptr, offset, seg, type, this	4
high, low	5
+, - (унарні)	6
*, /, mod, shl, shr	7
+, -, (бінарні)	8
eq, ne, lt, le, gt, ge	9
not	10
and	11
or, xor	12
short, type	13

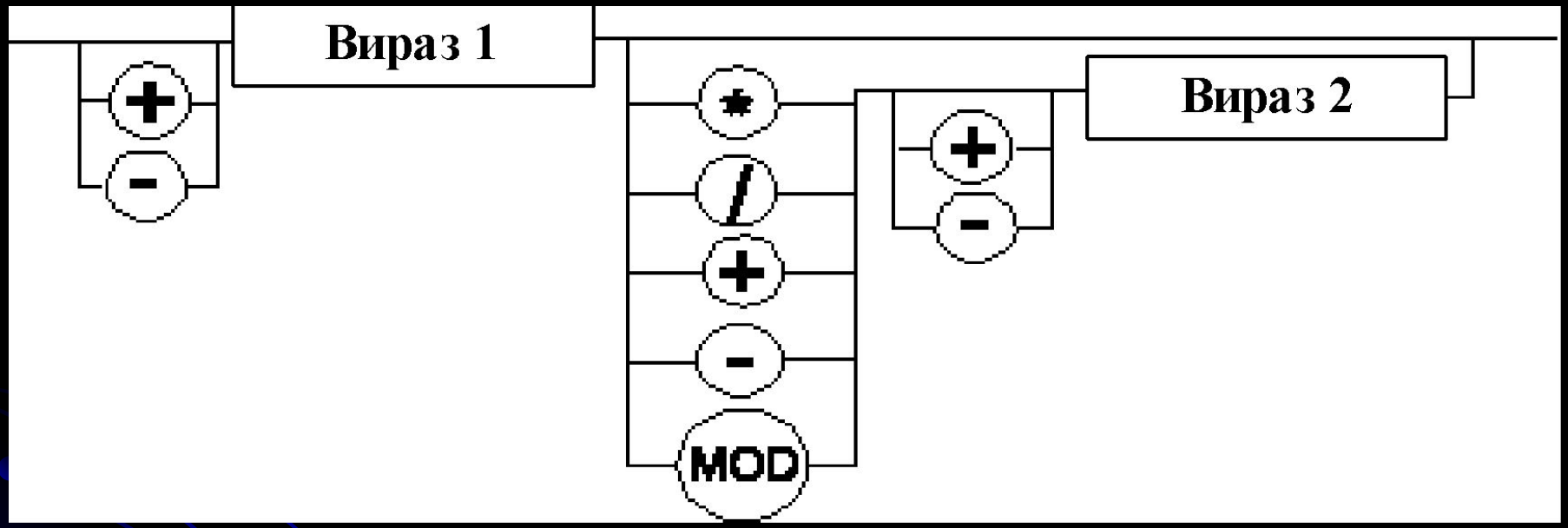
Арифметичні оператори

- *Арифметичні оператори*. До них відносяться:
  - унарні “**+**” і “**–**”;
  - бінарні “**+**” і “**–**”;
  - множення “*****”;
  - цілочисельного ділення “**/**”;
  - одержання залишку від ділення “**mod**”.

Арифметичні оператори

```
tab_size equ 50 ;розмір масиву в байтах

size_el equ 2 ;розмір елементів

…

;обчислюється число елементів масиву і заноситься врегістр cx

mov cx,tab_size / size_el ;оператор “/”
```

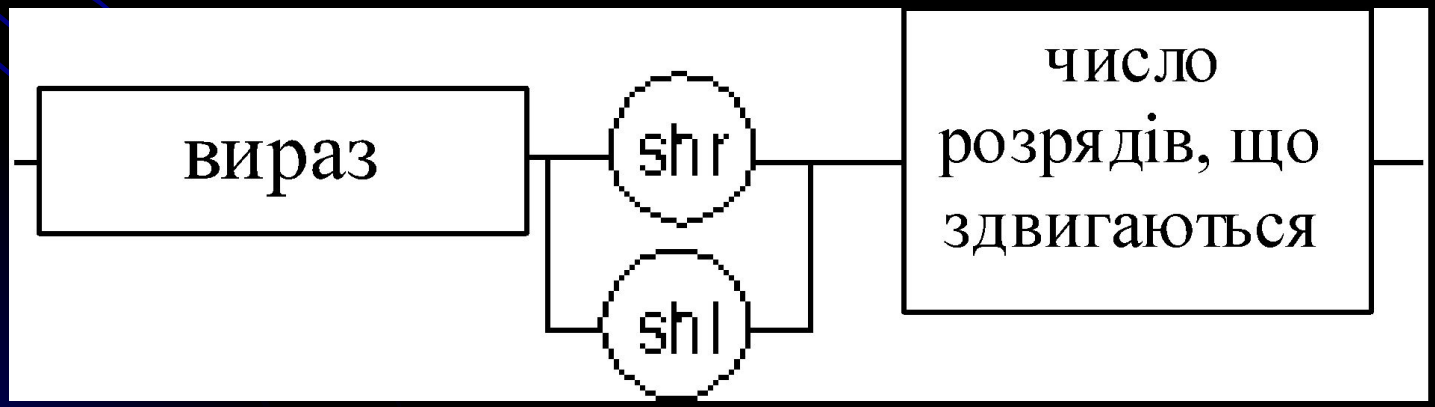
Арифметичні оператори



Оператори зсуву

- *Оператори зсуву* виконують зсув виразу на зазначену кількість розрядів

```
mask_b equ 10111011

…

mov al,mask_b shr 3 ;al=00010111
```



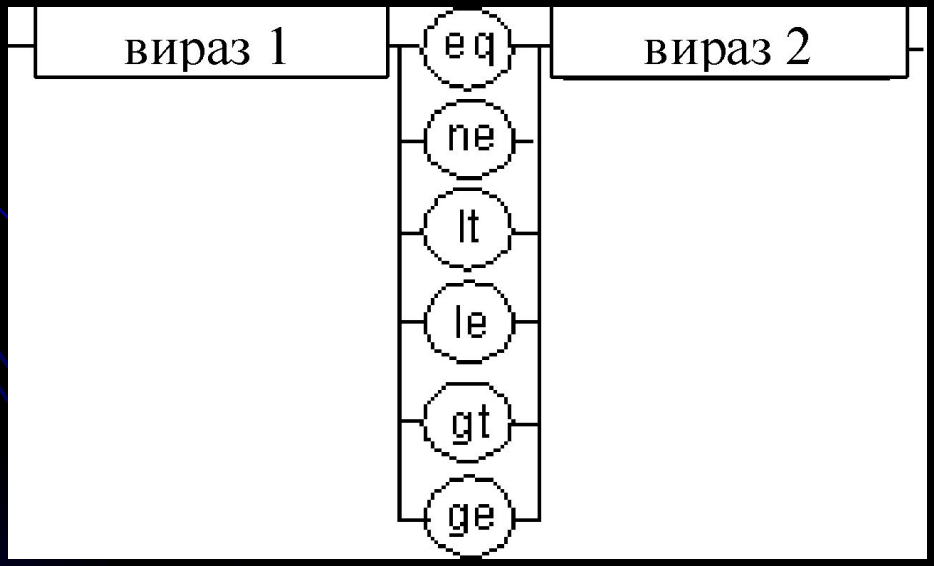
Оператори порівняння

- *Оператори порівняння* (повертають значення “істина” чи “неправда”) призначені для формування логічних виразів. Логічне значення “істина” відповідає цифровій одиниці, а “неправда” — нулю. Наприклад,

```
tab_size equ 30 ;розмір таблиці
…
mov al,tab_size ge 50 ;завантаження розміру таблиці в al cmp al,0
                      ;якщо tab_size < 50, то
je m1                 ;перехід на m1
…
m1:
 …
```

Оператори порівняння

- У цьому прикладі якщо значення tab_size більше чи дорівнює 50, то результат у al дорівнює 0ffh, а якщо tab_size менше 50, то al дорівнює 00h. Команда cmp порівнює значення al з нулем і установлює відповідні прапори в flags/eflags. Команда je на основі аналізу цих прапорів передає чи не передає керування на мітку m1.



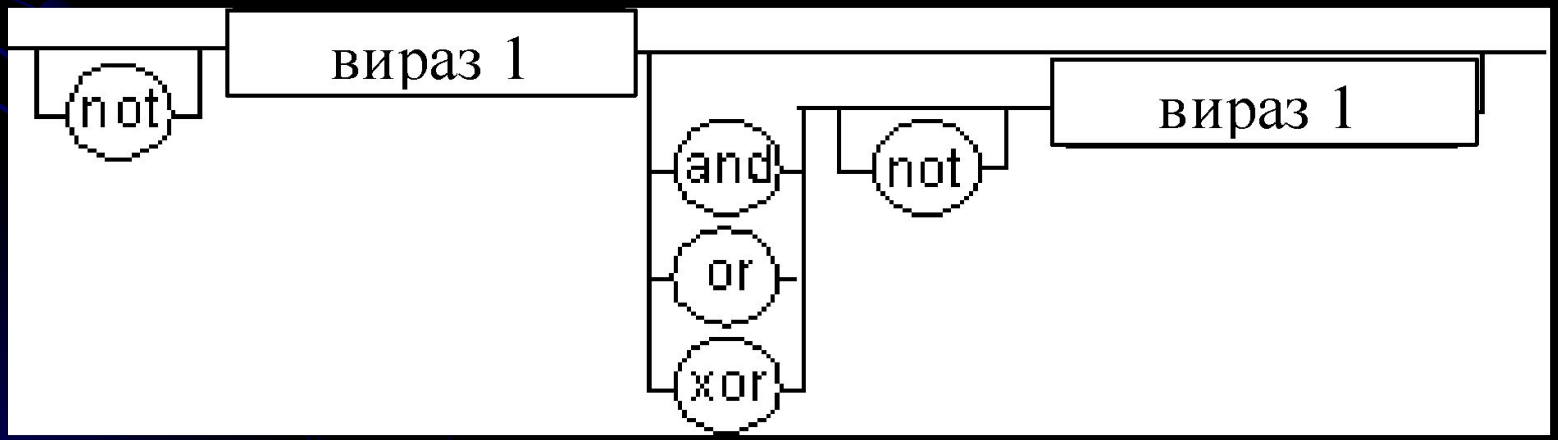
Оператори порівняння

Оператор	Значення
eq	ІСТИНА, якщо вираз_1 дорівнює вираз_2
ne	ІСТИНА, якщо вираз_1 не дорівнює вираз_2
lt	ІСТИНА, якщо вираз_1 менше вираз_2>ІСТИНА, якщо вираз_1 не дорівнює вираз_2
le	ІСТИНА, якщо вираз_1 менше чи дорівнює вираз_2
gt	ІСТИНА, якщо вираз_1 більше вираз_2
ge	ІСТИНА, якщо вираз_1 більше чи дорівнює вираз_2

Логічні оператори

- *Логічні оператори* виконують над виразами побітові операції. Вирази повинні бути абсолютними, тобто такими, чисельне значення яких може бути обчислено транслятором. Наприклад:

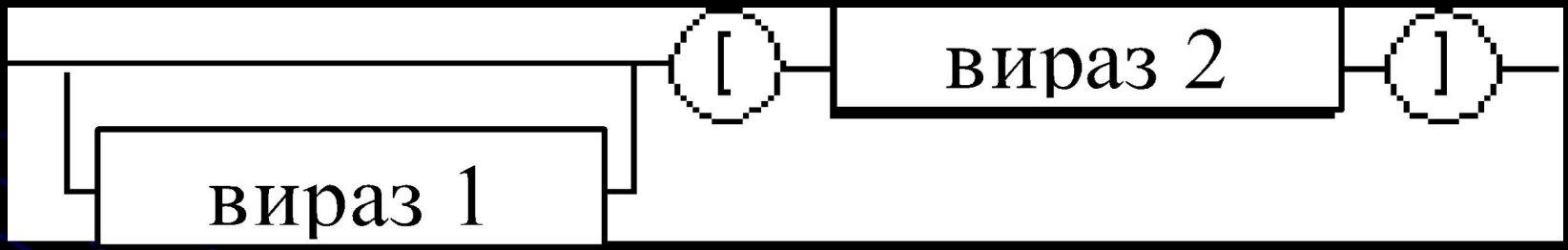
```
flags   equ     10010011
mov     al,flags xor 01h;al=10010010;пересилання в al поле flags з
                        ;інвертованим правим бітом
```



Індексний оператор

- *Індексний оператор* **[ ]**. Не дивуйтеся, але дужки теж є оператором, і транслятор їхню наявність сприймає як вказівку скласти значення **вираження_1** за цими дужками з **вираження_2**, взятим у дужки Наприклад,

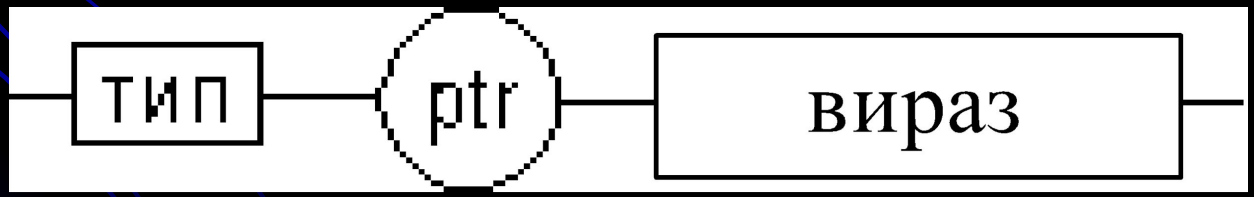
```
mov     ax,mas[si];пересилання слова за адресою mas+(si) у регістр ax
```



Оператор перевизначення типу

Оператор перевизначення типу ptr застосовується для перевизначення чи уточнення типу чи мітки змінної, обумовленим виразом.

```
d_wrd   dd      0
...
mov     al,byte ptr d_wrd+1
;пересилання     другого байта з подвійного слова
```

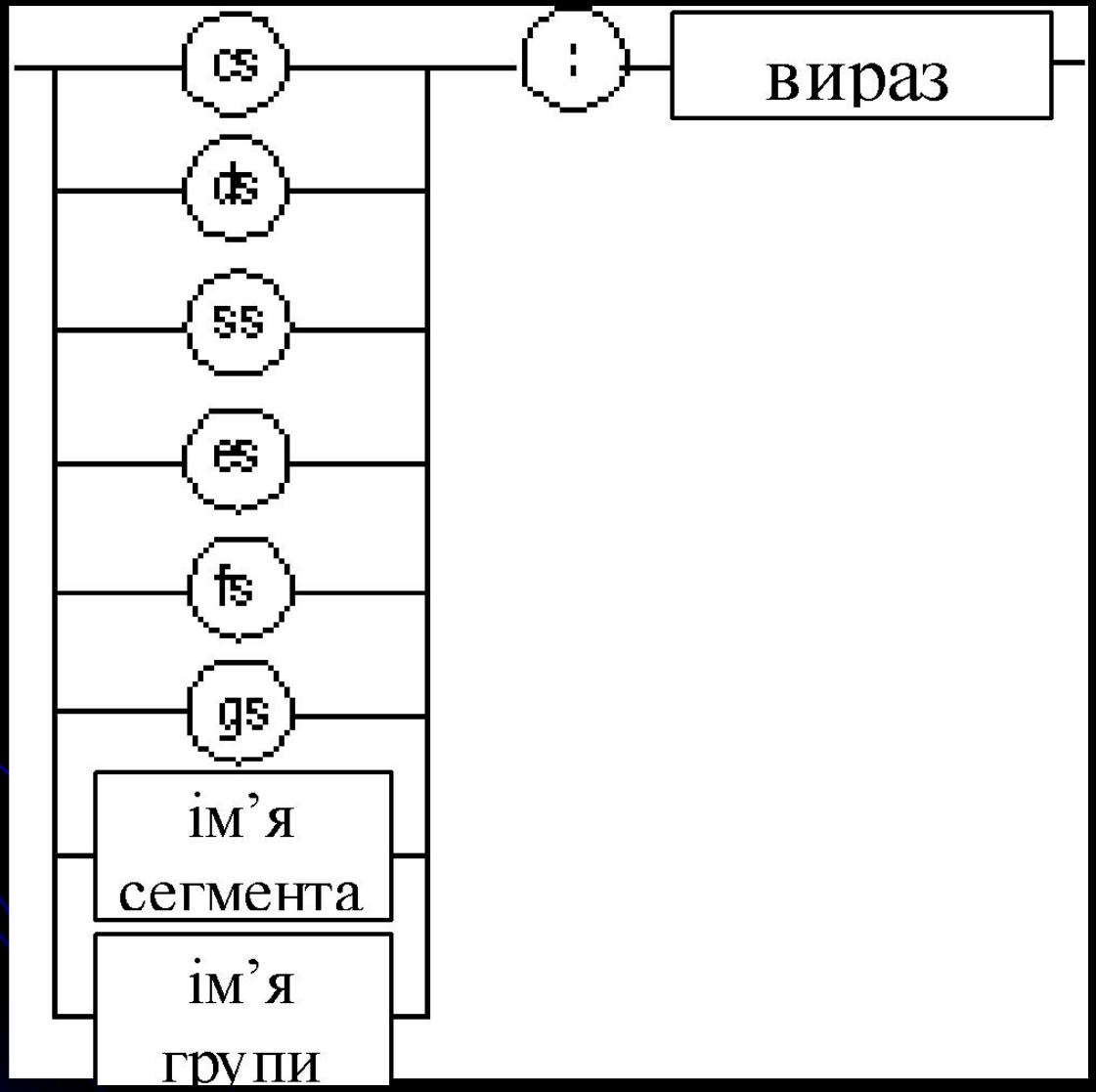


Оператор перевизначення сегмента

Змушує обчислювати фізичну адресу відносно конкретно заданої сегментної складової: "ім'я сегментного регістра", "ім'я сегмента" з відповідної директиви SEGMENT чи "ім'я групи"

```
    code
...
jmp     met1        ;обхід обов'язковий, інакше поле ind буде трактуватися
                    ;як чергова команда
ind     db      5           ;опис поля даних у сегменті команд
met1:
...
mov al,cs:ind       ;перевизначення сегмента дозволяє працювати з
                    ;даними, визначеними всередині сегмента коду
```

Оператор перевизначення сегмента



Оператори іменування типу структури одержання сегментної складової адреси виразу

- *Оператор іменування типу структури* **. (крапка)** також змушує транслятор робити визначені обчислення, якщо він зустрічається у виразі.
- *Оператор одержання сегментної складової адреси виразу* **seg** повертає фізичну адресу сегмента для виразу, у якості якого можуть виступати мітка, змінна, ім'я сегмента, ім'я групи чи деяке символічне ім'я.

Оператор одержання зсуву виразу offset

Оператор одержання зсуву виразу offset дозволяє одержати значення зсуву виразу у байтах відносно початку того сегмента, у якому вираз визначений.

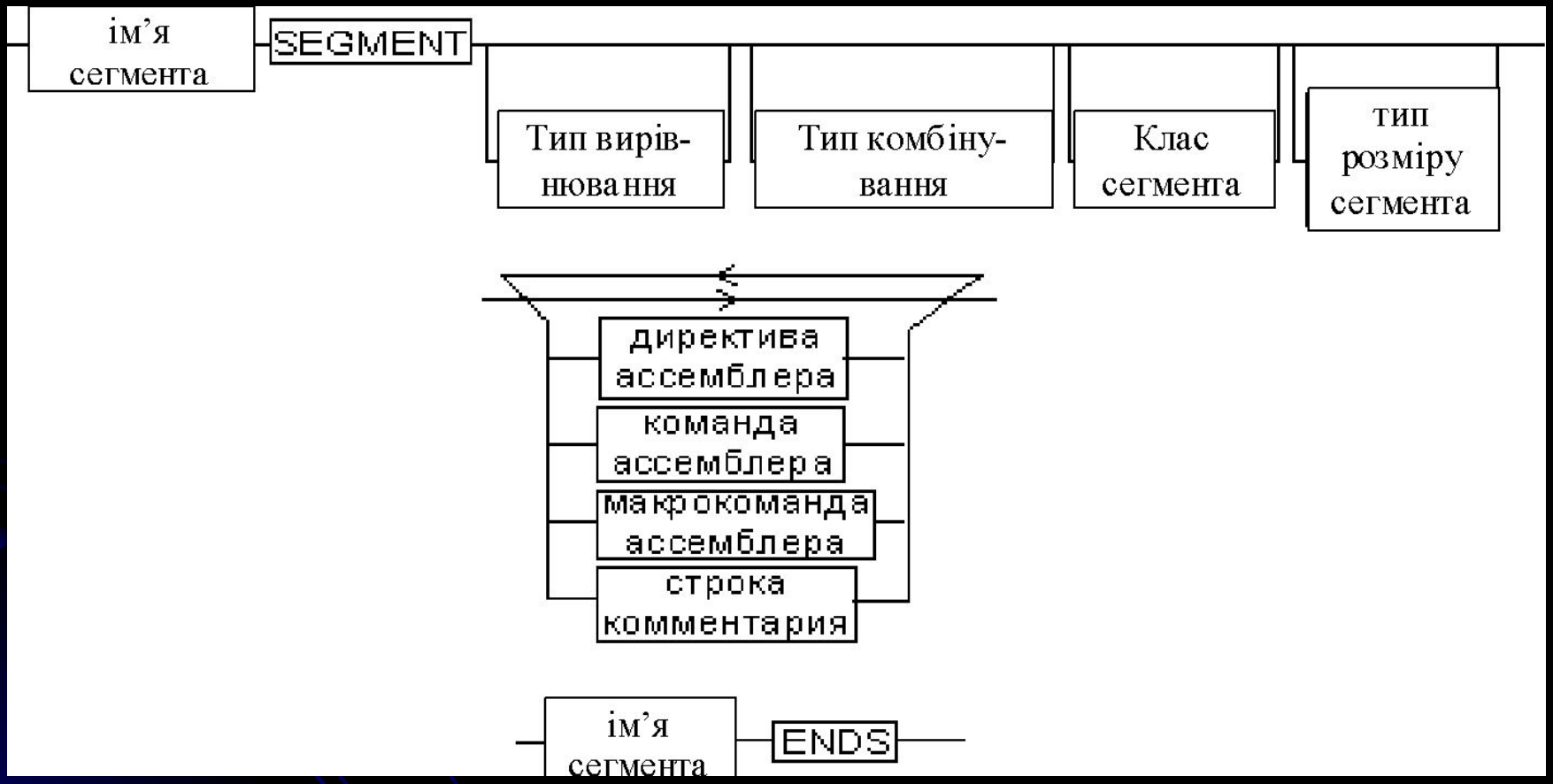
```
. data
pole    dw      5
...
.code
...
mov     ax,seg  pole
mov     es,ax
mov     dx,offset pole     ;тепер у парі es:dx повна адреса pole
```

Директиви сегментації

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

- с одним сегментом коду;
- с одним сегментом стека;
- с одним сегментом даних;
- с трьома додатковими сегментами даних.

Директиви сегментації



Операнди в директиві **SEGMENT**

Атрибут вирівнювання сегмента (тип вирівнювання) повідомляє розроблювачу про те, що потрібно забезпечити розміщення початку сегмента на заданій границі.

- **BYTE** — вирівнювання не виконується.
- **WORD** — сегмент починається за адресою, що кратна двом
- **DWORD** — сегмент починається за адресою, що кратна чотирьом
- **PARA** — сегмент починається за адресою, що кратна 16
- **PAGE** — сегмент починається за адресою, що кратна 256
- **MEMPAGE** — сегмент починається за адресою, що кратна 4 КБ

- За замовчуванням тип вирівнювання має значення **PARA**.

Операнди в директиві **SEGMENT**

Атрибут комбінування сегментів (комбінаторний тип) повідомляє розроблювачу , як потрібно комбінувати сегменти різних модулів, що мають те саме ім'я.

- **PRIVATE** — сегмент не буде поєднуватися з іншими сегментами з тим же ім'ям поза даним модулем;
- **PUBLIC** — розроблювач змушує з'єднати всі сегменти з однаковими іменами.
- **COMMON** — розташовує всі сегменти із тим самим ім'ям за однією адресою.
- **AT xxxx** — розташовує сегмент по абсолютній адресі параграфа
- **STACK** — визначення сегмента стека. Змушує розроблювача з'єднати всі однойменні сегменти й обчислювати адреси в цих сегментах щодо регістра ss.

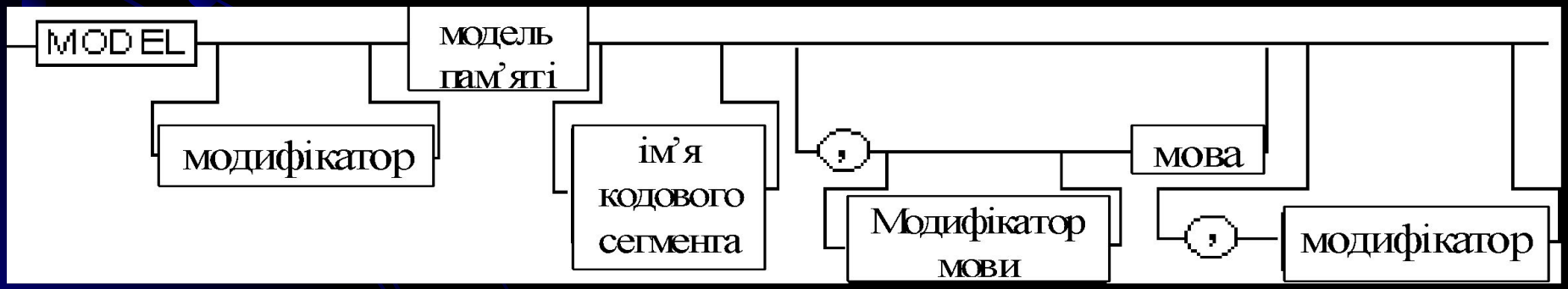
- За замовчуванням атрибут комбінування приймає значення **PRIVATE**.

Операнди в директиві **SEGMENT**

- ***Атрибут класу сегмента*** (тип класу) — це укладений в лапки рядок, що допомагає розроблювачу визначити відповідний порядок проходження сегментів при збиранні програми із сегментів декількох модулів.
- ***Атрибут розміру сегмента***. Для процесорів сегменти можуть бути 16, 32 та 64-розрядними. Це впливає, насамперед, на розмір сегмента і порядок формування фізичної адреси усередині нього. Атрибут може приймати наступні значення:
  - **USE16** — це означає, що сегмент допускає 16-розрядну адресацію. При формуванні фізичної адреси може використовуватися тільки 16-розрядний зсув. Відповідно, такий сегмент може містити до 64 Кбайт коду чи даних;
  - **USE32** — сегмент буде 32-розрядним. При формування фізичної адреси може використовуватися 32-розрядний зсув. Тому такий сегмент може містити до 4 Гбайт коди чи даних.

Директива MODEL

Обов'язковим параметром директиви MODEL є *модель пам'яті*. Цей параметр визначає модель сегментації пам'яті для програмного модуля. Передбачається, що програмний модуль може мати тільки визначені типи сегментів, що визначаються згаданими нами раніше *спрощеними директивами опису сегментів*.



Спрощені директиви визначення сегмента

Формат директиви (режим MASM)	Формат директиви (режим IDEAL)	Призначення
.CODE [ім'я]	CODESEG[ім'я]	Початок чи продовження сегмента коду
.DATA	DATASEG	Початок чи продовження сегмента ініціалізованих даних. Також використовується для визначення даних типу near
.CONST	CONST	Початок чи продовження сегмента постійних даних (констант) модуля
.DATA?	UDATASEG	Початок чи продовження сегмента неініціалізованих даних. Також використовується для визначення даних типу near
.STACK [розмір]	STACK [розмір]	Початок чи продовження сегмента стека модуля. Параметр [розмір] задає розмір стека
.FARDATA [ім'я]	FARDATA [ім'я]	Початок чи продовження сегмента ініціалізованих даних типу far
.FARDATA? [ім'я]	UFARDATA [ім'я]	Початок чи продовження сегмента неініціалізованих даних типу far

Директива MODEL

При використанні директиви **MODEL** транслятор робить доступними кілька ідентифікаторів, до яких можна звертатися під час роботи програми, для того, щоб одержати інформацію про ті чи інші характеристики даної моделі пам'яті

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

Директива MODEL Моделі пам'яті

Модель	Тип коду	Тип даних	Призначення моделі
TINY	near	near	Код і дані об'єднані в одну групу з ім'ям DGROUP. Використовується для створення програм формату .com.
SMALL	near	near	Код займає один сегмент, дані об'єднані в одну групу з ім'ям DGROUP. Цю модель звичайно використовують для більшості програм на асемблері
MEDIUM	far	near	Код займає кілька сегментів, по одному на кожен з'єднувальний програмний модуль. Усі посилання на передачу керування — типу far. Дані об'єднані в одній групі; усі посилання на них — типу near
COMPACT	near	far	Код в одному сегменті; посилання на дані — типу far
LARGE	far	far	Код у декількох сегментах, по одному на кожен з'єднувальний програмний модуль