

1.6 Особливості формування машинних команд

За мнемонікою команди транслятор створює машинну команду. Вона може займати від 1 до 6 байт в залежності від того, які режими адресації застосовуються.

Зміщення і власне дані також записуються в команду.

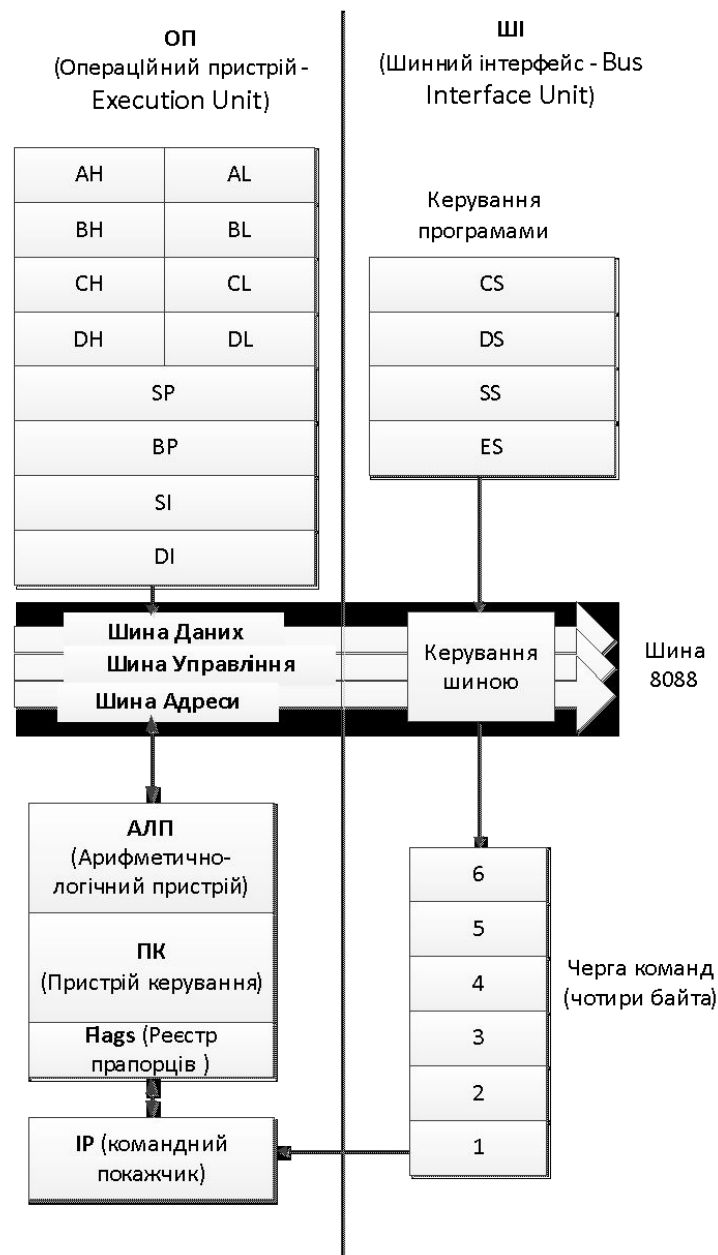
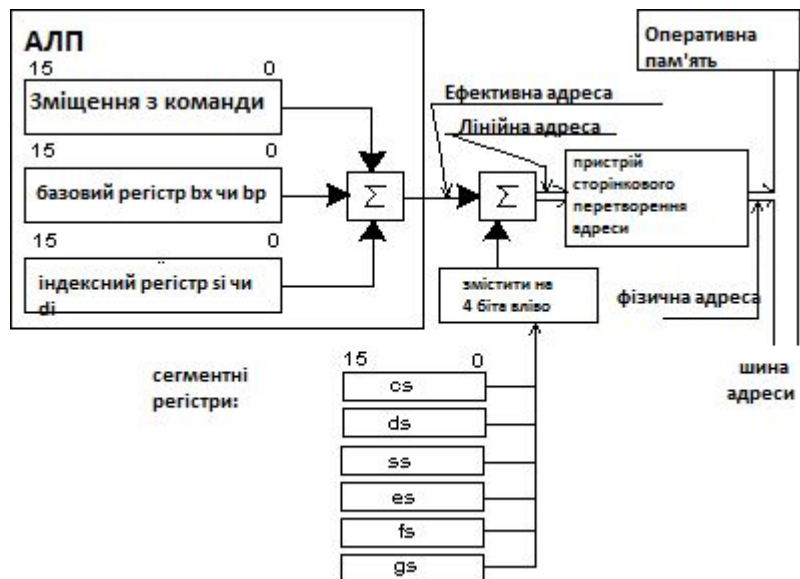
Якщо вони в межах $-128 + 127$, то займають байт, інакше - слово.

Найкоротші команди – один байт, для яких операндів визначається самою командою.

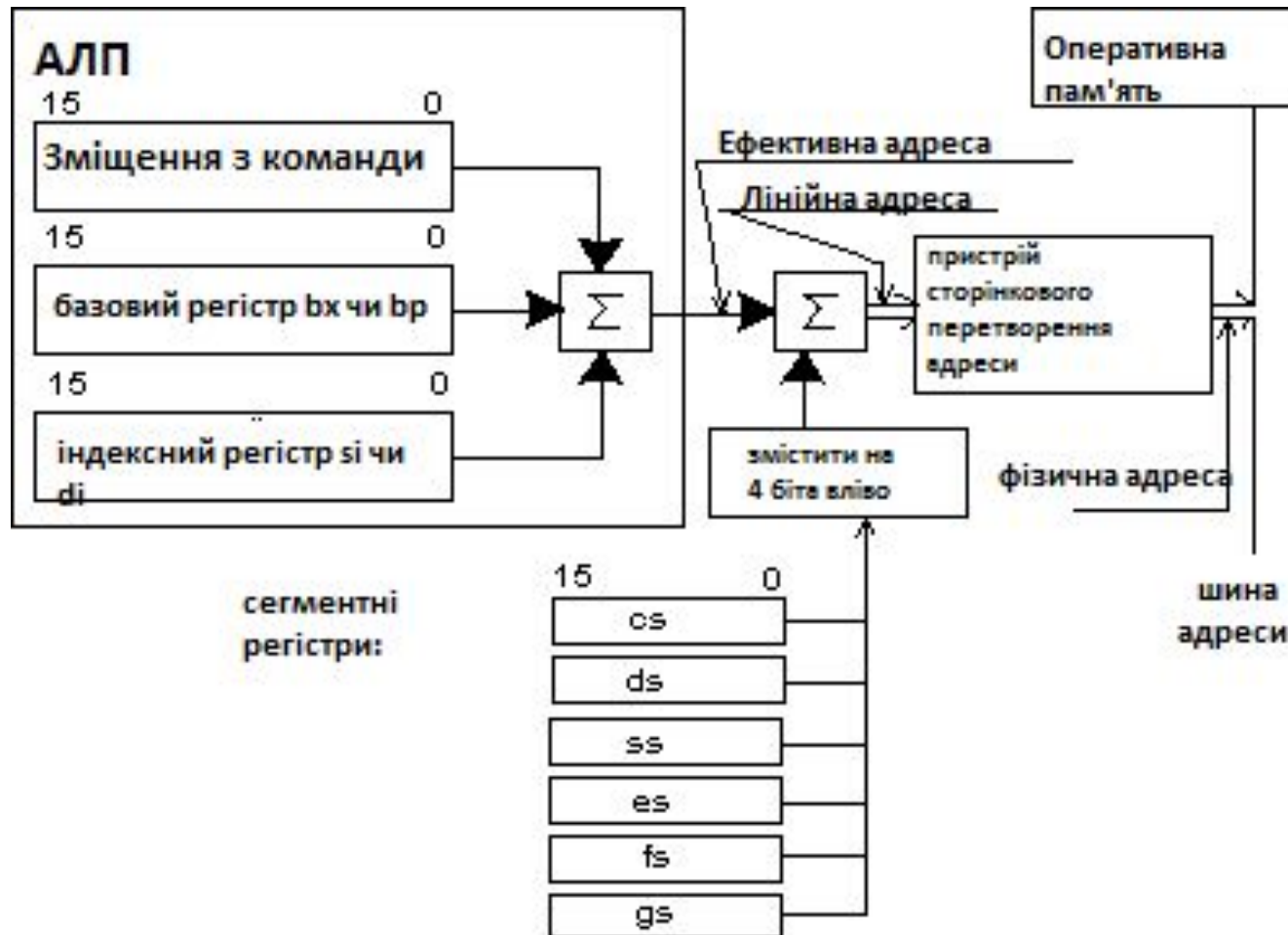
Наприклад команди для роботи з бітами регістра прапорців.

CLC – очистити біт прапорця carry flag.

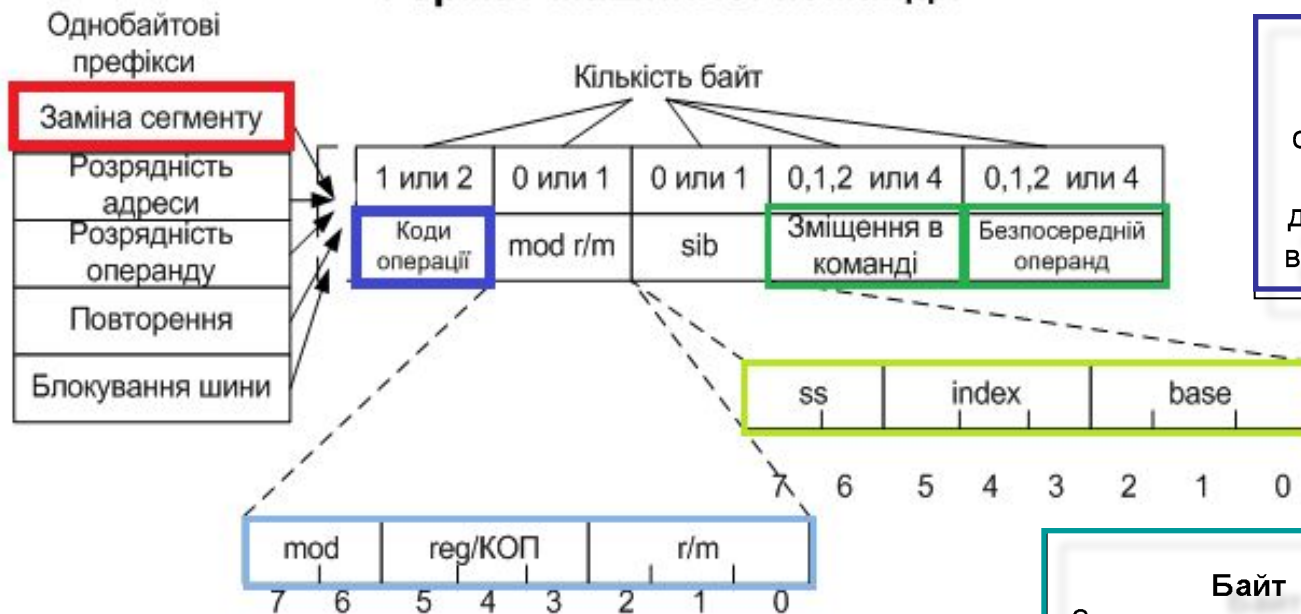
Найдовші команди, коли в них використовується 16-бітове зміщення (DISP) або безпосередньо 16-бітові дані (DATA).



Механізм формування фізичної адреси в реальному режимі



Формат машинної команди



Код операції (КОП)

Обв'язковий елемент, описує операцію, Команду, що виконується. Багатьом командам відповідає декілька кодів операцій, кожна з яких визначає нюанси виконання операції.

Після зміщення в команді

8, 16 или 32-разрядное целое число со знаком, представляющее собой, полностью или частично, значение эффективного адреса операнда

Байт mod r/m - режим адресації

Значення цього байту визначає форму адреси операндів, які використовуються:

mod = 00 - поле зміщення в команді відсутнє

mod = 01 - поле зміщення в команді наявне

mod = 11 - операндів в пам'яті немає: вони знаходяться в реєстрах

reg/КОП - Визначає або *реєстр*, Який знаходиться в команді на місці першого операнда, або можливе *розширення коду операції*

r/m використовується разом з полем *mod* і Визначає або *реєстр*, або базовий і індексний реєстри.

Заміна сегменту:

2eh — заміна сегменту cs;
36h — заміна сегменту ss;
3eh — заміна сегменту ds;
26h — заміна сегменту es;
64h — заміна сегменту fs;
65h — заміна сегменту gs

Префікс повторення:

- **безумовні** (rep — 0f3h), змушуючі повторюватись ланцюгову команду деяку кількість разі;
- **умовні** (repe/repz — 0f3h, repne/repnz — 0f2h), які при зациклюванні перевіряють деякі прапорці і, в результаті перевірки, можливий достроковий вихід з циклу.

Розрядність адреси:

уточнює розрядність адреси (32 або 16-розрядний)

Розрядність операнди:

66h — 16-розрядний;
67h — 32-розрядний

Байт масштаб-індекс-база (байт sib)

Розширює можливості адресації операндів:

ss — в ньому розміщується масштабний множник для індексного компоненту *index*

index — використовується для зберігання номеру індексного реєстру

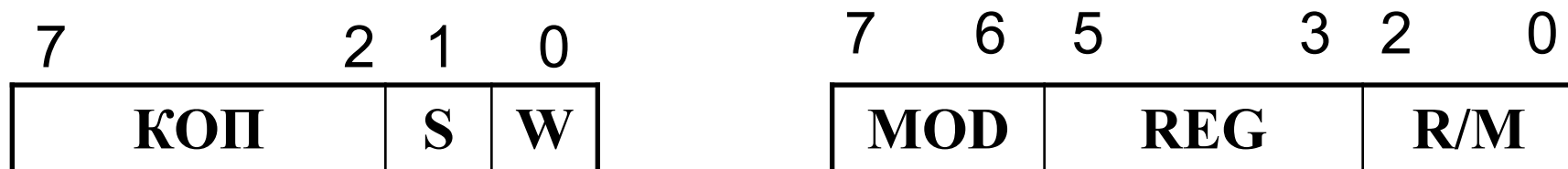
base — використовується для зберігання номеру базового реєстру

Поле безпосереднього операнда

Необов'язкове поле, що являє собою 8,16 або 32-розрядний безпосередній операнд. Наявність цього поля, необов'язкове поле, що являє собою 8, 16 або 32-розрядний операнд. Наявність цього поля відображається на значенні байту *mod r/m*.

Розглянемо формат двооперандної команди. В першому байті записується код операції, в другому – режим адресації.

Інші байти виділяються під зміщення (DISP) чи безпосередньо дані (DATA).



В першому байті крім коду операції є два однобітові індикатори:

W – визначає, над якою одиницею даних виконується команда.

W = 1 – над словом, **W** = 0 - над байтом.

Бітів **S** показує чим являється даний регістр.

S = 0 – операндом-джерелом

S = 1 – операндом-приймачем

Регістри кодуються таким чином:

КОП	S	W
------------	----------	----------

КОП	W = 1 (над словом)	W = 0 (над байтом)
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

Сегментні регістри кодуються так:

00	DS
01	CS
10	SS
11	ES

MOD	REG	R/M
------------	------------	------------

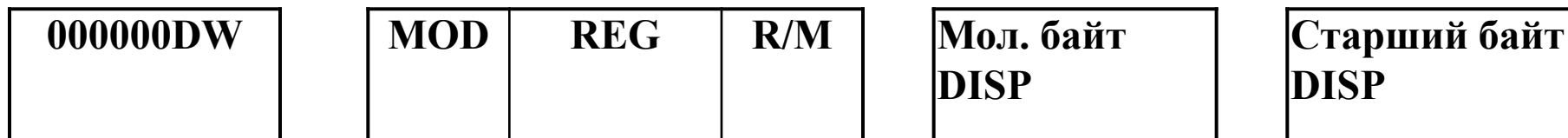
Режим адресації другого операнда визначається кодами в полях **MOD** і **R/M** (register-memory).

Поле **MOD** визначає, що саме закодовано в полі слово **R/M**. Коли ж **MOD** = 11, це значить, що в поле **R/M** записано код другого регістру.

R/M	MOD- 00 (зміщення НЕМАЄ)	MOD-01 (зміщення БАЙТ)	MOD-10 (зміщення СЛОВО)
000	[BX] + [SI]	[BX + SI] + DISP 8	[BX + SI] + DISP 16
001	[BX] + [DI]	[BX + DI] + DISP 8	[BX + DI] + DISP 16
010	[BP + SI]	[BP + SI] + DISP 8	[BP + SI] + DISP 16
011	[BP + DI]	[BP + DI] + DISP 8	[BP + DI] + DISP 16
100	[SI]	[SI] + DISP 8	[SI] + DISP 16
101	[DI]	[DI] + DISP 8	[DI] + DISP 16
110	disp 16	[BP] + DISP 8	[BP] + DISP 16
111	[BX]	[BX] + DISP 8	[BX] + DISP 16

Особливістю МП є те, що для однієї команди може бути **кілька кодів** та **форматів** в залежності від **режиму адресації** чи операндів, які там використовуються.

Наприклад, для команди **ADD** (додати):
Додати операнди в регістрах чи пам'яті



Необов'язкова частина, частіше залежить від
поля **MOD**

Додати безпосередній операнд до регістру чи пам'яті



Необов'язкова частина, частіше залежить від поля **MOD**

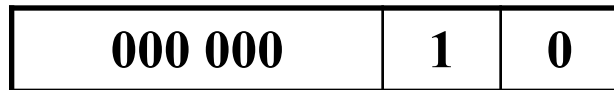
Додати безпосередньо з AX (AL). Спеціальний випадок для акумулятора:



Тобто, команда **ADD** має 3 варіанти коду і різні формати. Як видно, в форматі є лише одне поле для кодування комірки пам'яті. Тому в **двоадресних** командах може бути лише **один операнд** – комірка пам'яті, а не дві. Отже, додати вміст однієї команди до іншої за одну команду не можна.

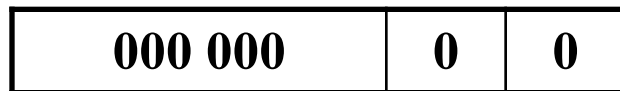
Інколи, одна і та ж команда може бути закодована транслятором по-різному.

Наприклад, **ADD CL, BH**.

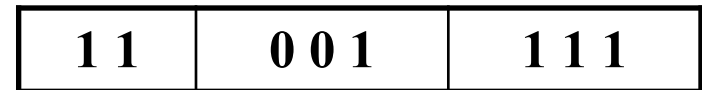


|
приймач

Цей код 02CFH



|
джерело



p/p CL BH



00F9H

Поля із 6-ти бітів для кодування всіх команд не достатньо. Тому деякі команди об'єднуються в групу і в першому байті кодується група команд.

Наприклад, формати команд, з безпосередньою адресацією.

Регістр – безпосередній операнд

КОП	S	W
-----	---	---

1 1	КОП	REG
-----	-----	-----

Необов'язкова
частина

Пам'ять – безпосередній операнд

КОП	S	W
-----	---	---

1 1	КОП	MEM
-----	-----	-----

Крім відмічених частин може бути кілька префіксів, що може збільшити розмір до 15 байт.

Оператори

Програма на асемблері складається з окремих рядків-операторів, які описують виконувані операції.

Оператором може бути **команда** чи **псевдооператор** (директива).

Команда – це умовне (символьне) позначення машинних команд.

[Мітка:] Мнемокод [Операнд] [, Операнд] [; Коментар]

Із них обов'язкове є лише поле мнемокоду, всі інші частково або повністю можуть бути відсутніми.

[Мітка:] Мнемокод [Операнд] [, Операнд] [; Коментар]

до 31 символу:

A...Z

a...z

0...9

?

.

@

—

\$

мнемоніка
команди
від 3 до 6
символів

приймач

джерело

Справа від;. Може бути в рядку,
а також може займати цілий
рядок

Наприклад:

COUNT: MOV AX,DI; переслати DI в акумулятор

Псевдооператори – керують роботою транслятора, а **не мікропроцесора**. З їх допомогою визначають сегменти і процедури (підпрограми), дають імена командам та елементам даних, резервують робочі місця в пам'яті.

Псевдооператори можуть мати **4** поля:

**[Ідентифікатор] Псевдооператор [операнд] [;
коментар]**

Поля в дужках можуть бути відсутніми. В Макроасемблері нараховують близько **60 псевдооператорів**.

Псевдооператори даних

Їх можна розділити на групи:

1. *Визначення ідентифікаторів;*
2. *Визначення даних;*
3. *Псевдооператори визначення сегменту і процедури.*

1. Визначення ідентифікаторів

Дозволяють присвоїти символічне ім'я виразу, константі, адресі, іншому символічному імені. Після цього можна використовувати це ім'я.

Наприклад:

Операндом в цьому псевдооператорі в загальному випадку може бути вираз:

K	EQU	1024; константа
TABLE	EQU	DS: [BP] [SI]; адреса
SPEED	EQU	RATE ; синонім
COUNT	EQU	CX; ім'я регістру

DBL_SPEED EQU 2*SPEED

Тобто, вираз може включати деякі простіші арифметичні та логічні дії. Якщо вираз включає константу, то за замовчуванням вона рахується десятковою.

Шіснадцяткові константи – справа літера **H** (2FH). Коли така константа починається з літери, то ліворуч треба поставити 0 - 0FH, а не FH;

Вісімкові константи – з літерою **Q**: 256Q;

Двійкові – з літерою **B** - 01101B.

Очевидно, що на відмінну від мов високого рівня, тут вираз виконується під час трансляції.

Крім псевдооператора EQU можна використати знак “ = ”.

Таке ім'я може задавати лише числову константу, а не символну чи якусь іншу.

Це ім'я можна перевизначити, а визначену через **EQU** – НЕ МОЖНА.

```
CONST1 = 59;  
CONST2 = 98;  
CONST3 = CONST1 + 2.
```

Визначення даних

Коли комірка використовується для збереження даних, їй можна присвоїти ім'я за допомогою псевдооператорів **DB**, **DW**, **DD** (Define Byte, Word, Double Word).

Загальна структура:

[ім'я] псевдооператор вираз [,]

[,] – один чи кілька виразів через кому.

```
MAX DB 57
```

```
WU_MAX DW 5692
```

```
A_TABLE DW 25, -592, 643, 954; - масив
```

Коли значення змінних завчасно невідоме, але буде використане для результатів, то в полі виразу треба поставити «?».

LAMBDA DW ?

Для тексту можна використовувати псевдооператор **DB**.

POLITE DB 'Введіть дані знову \$'

Якщо записуються однакові дані, то можна використовувати команду **DUP** (duplicate) - повторити.

BETA DW 15 DUP(0) или GAMA DW 3 DUP (4DUP(0))

Зрозуміло, що цю операцію можна використовувати для резервування пам'яті:

ALPHA DW 20 DUP (?)

Змінні можна використовувати для збереження не тільки даних, але й адрес. Змінна має назву THERE:

THERE DB 123

.....

NEAR_THERE DW THERE

NEAR_THERE DD THERE

Під іменем **NEAR_THERE** записують 16-бітову адресу **THERE**, тобто, її зміщення, а під іменем **FAR_THERE** – 32-бітову адресу **THERE**, тобто, сегмент + зміщення.

Псевдооператори визначення сегменту і процедури

Як зазначалося, програма може складатися з декількох сегментів: коду, даних, стеку, додаткового сегменту.

Для поділу програми на сегменти використовуються псевдооператори **SEGMENT** та **ENDS**.

Їх структура:

Ім'я SEGMENT [атрибути]

.....

Ім'я ENDS

Наприклад:

```
DATASEG SEGMENT
  A DW 500
  B DW -258
  SQUARE DB 54, 61, 95, 17
DATASEG ENDS
```

В сегменті даних визначаються імена даних та резервується пам'ять для результатів.

У псевдооператора **SEGMENT** можуть бути 4 атрибути:

Имя **SEGMENT** [вирівнювання] [об'єднання][клас] [розмір сегменту]

Вирівнювання визначає, з яких адрес можна розміщувати сегмент.

BYTE — вирівнювання **не виконується**. Сегмент може починатися з будь-якої адреси пам'яті;

WORD — сегмент починається з адреси, кратної **двом**;

DWORD — сегмент починається з адреси, кратної **чотирьом**, тобто є два останніх (молодших) значущих біти, які рівні 0;

PARA — сегмент починається з адреси, що кратна 16, то остання шістнадцяткова цифра адреси повинна бути 0h;

PAGE — сегмент починається з адреси, що кратна 256, то дві останні шістнадцяткові цифри адреси повинні бути 00h (вирівнювання на межі 256-байтної сторінки);

MEMPAGE — сегмент починається з адреси, що кратна 4 Кбайтам, тобто три останні шістнадцяткові цифри адреси повинні бути 000h (адреса наступної 4-Кбайтної сторінки пам'яті).

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

Об'єднання визначає спосіб опрацювання сегменту при компонуванні.

PRIVATE — за замовчуванням, сегмент повинен бути відділений від інших сегментів.

PUBLIC — всі сегменти з однаковим іменем та класом завантажуються в суміжні області. Всі вони будуть мати *одну початкову адресу*.

STACK — для компонувальника аналогічно **PUBLIC**.

COMMON — розміщує всі сегменти з одним і тим самим ім'ям за однією вдресою. Розмір отриманого в результаті сегмента буде рівне розміру найбільшого сегмента;

AT xxxx — розміщує сегмент за абсолютною адресою (xxxx) параграфа (параграф — об'єм пам'яті, кратний 16; тому остання шістнадцяткова цифра адреси параграфа рівна 0). За замовчуванням атрибут комбінування приймає значення **PRIVATE**.

В будь-якій програмі повинен бути один сегмент з атрибутом **STACK**.

Клас – цей атрибут може мати будь-яке коректне ім'я, не обмежене апострофами .

Атрибут використовується компонувальником для обробки сегментів з однаковими іменами та класами.

Часто використовуються імена 'CODE' та 'STACK'.

```
STACK SEG SEGMENT PARA STACK 'STACK'  
    MAS DW 20 DUP (?)  
STACK SEG ENDS.
```

Як зазначалося, процесор використовує регістр **CS** для адресації сегменту коду, **SS** – сегменту стеку, **DS** - даних, **ES** - додатковий.

Оскільки, транслятор не розуміє тексту, то йому необхідно повідомити призначення кожного сегменту. Для цього існує псевдооператор **ASSUME**, який має вигляд:

ASSUME SS:ім'я стеку, DS:ім'я даних, CS:ім'я коду.

Наприклад:

```
ASSUME SS:STACK SEG, DS:DATA SEG, CS:CODE SEG
```

Якщо певний сегмент не використовується, то можна його упустити або записати:

DS:NOTHING

Ця директива лише повідомляє *транслятору* що з чим зв'язувати, але **не завантажує** відповідні **адреси** в регістри.

Це повинен зробити сам програміст в програмі.

Сегмент коду може вміщати одну або кілька процедур.
Окрема процедура починається псевдоопертором PROC:

ім'я PROC [атрибут]

ім'я ENDP

Якщо процедура передбачає повернення до точки виклику, то перед **ENDP** повинна стояти директива **RET** (Return From Procedure). Тоді процедура стає підпрограмою.

Атрибутом процедури може бути **NEAR** (близький) и **FAR** (далекий).
NEAR-процедура може бути викликана лише з цього сегменту.
Процедуру з атрибутом **FAR** можна викликати з будь-якого сегменту команд. Основна процедура повинна мати атрибут **FAR**.

```
CALC PROC  
.....  
RET  
CALC ENDP
```

Підпрограмні сегменти

В програмі може бути кілька сегментів з однаковим іменем. Рахується, що це **один сегмент**, який з певних причин записаний **частинами**.

Параметри директиви **SEGMENT** потрібні для великих програм, які складаються з декількох файлів. Для невеликої програми, яка складається з одного файлу, ці параметри не потрібні, крім деяких випадків.

Наприклад, маємо такий
ряд сегментів:

```
A SEGMENT
    A1 DB 400h DUP(?)
    A2 DW 8
A ENDS
;
B SEGMENT
    B1 DW A2
    B2 DD A2
B ENDS
;
C SEGMENT
ASSUME ES:A, DS:B, CS:C
L: MOV AX, A2
    MOV BX, B2
.....
C ENDS
```

Сегменти, розміщені на межі параграфу, тобто, адреса кратна 16.

Якщо **A** розміщено 1000h, то він займе місце до 1402h.

Наступна адреса - 1403h не кратна 16, тому сегмент **B** розміститься з адреси 1410h.

Під сегмент **B** буде відведено 6 байт, а сегмент **C** розміститься з адреси 1420h.

Значення імені сегменту

Значення імені сегменту являється номером, який відповідає сегменту пам'яті, тобто, перші 16 бітів початкової адреси заданого сегменту. Тобто, **A** набуде значення 1000h, **B** - 1410h, **C** - 1420h. Зберігаючи в тексті ім'я сегменту, асемблер буде замінити його на відповідну величину. Наприклад:

MOV BX, B відповідає MOV BX, 1410h

Порівняйте MOV BX, A2.

Отже, в мові асемблер імена сегментів – це константні вирази, а не адресні.

Тому, команда запише до BX **адресу** 1410h, а **не вміст** слова за цією адресою.

Початкове завантаження сегментних регістрів

Директива **ASSUME** відмічає, з якими сегментними регістрами пов'язувати сегменти. Регістри **DS** і **ES** потрібно завантажити початковими адресами самому програмісту.

Нехай, регістр **DS** потрібно встановити на початок сегменту **B**.

Оскільки, ім'я сегменту – це константа, то безпосередньо в **DS** її відправити не можна, а треба через регістр загального значення:

```
MOV AX, B  
MOV DS, AX
```

Аналогічно завантажуються і регістр **ES**. Регістр **CS** завантажувати не потрібно. Це зробить сама операційна система перед тим, як передати управління програмі. Відносно регістру **SS** існує дві можливості:

- По-перше, це можна зробити так як для регістрів **DS** і **ES**. Але тут потрібно записати початкову адресу в регістр **SS**, а в покажчик стеку **SP** – кількість байт під стек.
- По-друге – це можна поручити операційній системі. Для того в відповідній директиві **SEGMENT** треба відмітити атрибут **STACK**.

Структура програми

Відносно сегменту стеку **SS**, навіть якщо програма і **не використовує** його, то створити такий сегмент в програмі **необхідно**.

Тому що, стек програми використовується операційною системою при опрацюванні переривань, наприклад, при натисканні кнопок.

Рекомендований розмір стеку - 128 байт.

Тому, програма на асемблері має таку структуру:

```
Title EXAMPLE ;заголовок
dat segment ; сегмент даних
    mas dw 1-3,56,91
    res dw 10 dup(?)
dat ends
st segment stack 'stack' ; сегмент стеку
    dw 128 dup(?)
st ends
cod segment 'code' ; сегмент коду
    ASSUME DS:dat, SS:st, CS:cod
    beg proc far
        <операторы>
        ret
    beg endp
cod ends
    end beg
```

Структура програми

В загальному випадку в сегменті даних можна розміщувати і команди, а в сегменті коду - дані. Але краще цього **не робити**, бо виникнуть проблеми з сегментацією.

END – кінець програми.

Якщо програма з одного файлу, то в цьому рядку додається ім'я початкової виконуваної адреси - **beg**.

Якщо з кількох файлів, то тільки в одній програмі відмічається ця адреса. В інших - лише **END**.

Порівняння асемблерних трансляторів

MASM (Macro Assembler) - стандарт де-факто при програмуванні під Windows 9x/NT;

TASM (Turbo Assembler) – стандарт при програмуванні під MS-DOS;

Lazy Assembler - реінкарнація TASMа з підтримкою нових команд процесора;

FASM (Flat Assembler) – неординарний, але «іграшковий» асемблер;

NASM (Netwide Assembler) – гарний асемблер під LINUX/BSD з Intel-синтаксисом;

YASM (Yet another assembler) – вдосконалений варіант NASM'а;

HLA (High Level Assembly Language) – дуже високорівневий асемблер.

Спрощений опис сегментів

Є декілька види трансляторів асемблеру:

MASM фірми Microsoft

TASM фірми Borland, може працювати в режимі MASM і IDEAL.

Для простих програм, які складаються з одного сегменту даних, одного сегменту стеку та одного сегменту коду є можливість спростити директиви опису сегменту.

Для цього спочатку наводиться директива **masm** чи **ideal** режиму роботи транслятора **TASM**.

Далі відмічається модель пам'яті **model small**, яка частково виконує функції директиви **ASSUME**.

Спрощений опис сегментів

Наприклад:

```
Masm ; режим роботи транслятора TASM
model small ; модель пам'яті
.data ; заголовок сегменту даних
mas dw 10 DUP (?)
.stack ; заголовок сегменту стеку
db 256 dup (?)
.code ; заголовок сегменту коду
main proc
    mov ax, @data ; адреса сегменту стеку до AX
    mov ds, ax
; текст програми
    mov ax, 4c00h ; те саме, що і RET
    int 21h
main endp
end main
```

Спрощені директиви визначення сегменту

Режим MASM	Режим IDEAL	Описание
.code [ім'я] .data	Codeseg[ім'я] Dataseg	Початок чи продовження сегменту коду Початок чи продовження сегменту ініціалізації даних
.const .data?	Const Udataseg	Початок чи продовження сегменту констант Початок чи продовження сегменту неініціалізованих даних
.stack [розмір] .fardata[ім'я] .fardata?[ім'я]	Stack[розмір] Fardata[ім'я] Ufardata[ім'я]	Початок сегменту стеку Ініціалізовані дані типу FAR Неініціалізовані дані типу FAR

Ідентифікатори, які створює директива MODEL:

@ code – фізична адреса (зміщення) сегменту коду

@ data – фізична адреса (зміщення) сегменту даних

@ fardata – фізична адреса (зміщення) сегменту даних типу far

@ fardata? – фізична адреса (зміщення) сегменту ініціалізованих даних типу far

@ curseg – фізична адреса (зміщення) сегменту неініціалізованих даних типу far

@ stack – фізична адреса (зміщення) сегменту стеку

Моделі пам'яті

Модель	Тип коду	Тип даних	Призначення моделі
TINY	Near	Near	Код і данні в одній групі DGROUP для створення .com-програм
SMALL	Near	Near	1 сегмент коду. Дані в одну групу DGROUP
MEDIUM	Far	Near	Код займає <i>n</i> сегментів, по одному в кожному модулі. Всі передачі управління типу far . Дані в одній групі, всі посилання на них типу near
COMPACT	Near	Far	Код в 1 сегменті, посилання на данні типу far
LARGE	Far	Far	Код в <i>n</i> сегментах, по одному на кожен об'єднаний модуль

Модифікатор директиви **model** дозволяє визначити деякі особливості вибраної моделі пам'яті: **Use 16** - 16-бітові сегменти; **Use 32** - 32-бітові сегменти; **DOS** - програма в MS DOS. Повний і спрощений опис не виключає один одного, але в повному більше можливостей.



- Команда обміну з портами
- Команда LEA – завантаження ефективної адреси
- Команда пересилання прапорців