ТЕМА 9 Основи програмування мовою асемблер

9.1 Загальні відомості про мову програмування асемблер
9.2 Формат мови асемблер
9.3 Регістри загального призначення (AX, BX, CX, DX)
9.4 Сегментні регістри (CS, DS, SS, ES)

9.1 Загальні відомості про мову програмування асемблер

Зазвичай програмісти використовують для розробки своїх прикладних програм такі алгоритмічні мови, як Pascal, C++, Basic, Java, Prolog.

Однак насправді такі мови, які ще називають мовами високого рівня, є посередниками між програмістом і комп'ютером. Комп'ютер їх безпосередньо не розуміє, тому необхідно перекласти програми на його рідну, машинну мову, тобто компілювати програми.

Мова асемблера також потребує перекладу на машинну мову. Однак вона принципово відрізняється від алгоритмічних мов високого рівня тим, що практично один до одного відповідає машинній мові. За своїм змістом мова асемблера є системою позначень команд процесора(одна команда мови – одна команда процесора). Це єдина мова, яка дає змогу безпосередньо керувати командами процесора. За допомогою мови асемблера можна використовувати усі наявні можливості процесора і будувати максимально ефективні програми. Очевидно, що мова асемблера тісно пов'язана з будовою процесора та комп'ютера загалом. Отже, для програмування мовою асемблера необхідні знання архітектури процесора та основних принципів роботи комп'ютера.

Для професійного програмування необхідно добре знати, що відбувається у комп'ютері під час виконання програми. Вивчити внутрішні засади функціонування комп'ютера найкраще через мову асемблера з відповідним закріпленням матеріалу на практичних заняттях. Якщо ж програміст використовує для роботи лише мови високого рівня, знання асемблера дає змогу у багатьох випадках будувати кращі програми. А комбінування програми мовою високого рівня та мовою асемблера для цілої низки задач є найефективнішим рішенням. Крім того, існує певне коло задач, реалізувати які можна лише мовою асемблера.

Числа у шістнадцятковому записі позначають суфіксом H, а двійкові числа - суфіксом B. Десяткові числа пишуть без суфікса або з суфіксом D. Для зображення даних в асемблерній програмі можна користуватися будь-якою з трьох розглянутих систем: десятковою, двійковою чи шістнадцятковою.

Під час запису 16-кових чисел важливо пересвідчитися, що асемблер сприйме їх як число. Якщо введено FAH, то це може бути або шістнадцяткове число FA, або ім'я змінної FAH. Асемблер передбачає, що запис числа розпочинається з цифри, а імені змінної чи позначки - з букви.

Тому FAH у мові асемблера виявляється змінною. Якщо ж ми маємо на увазі не змінну, а число, то його необхідно записати 0FAH: це число має бажане значення і, відповідно, його запис розпочинається цифрою. Отже, кожному шістнадцятковому числу, запис якого розпочинається зі значень від A до F, повинен передувати 0 (нуль).

Константи

• Двійкова– послідовність цифр 0 і 1, що завершується буквою B (наприклад, 11001010B).

• Десяткова– послідовність цифр від 0 до 9, що може завершуватися
буквою D(наприклад,419 або 419D).

• Шістнадцяткова– послідовність цифр від 0 до 9 і букв від A до F, що
завершується буквою H. Першим символом обов'язково повинна бути
цифра(наприклад, 105H або 0C04BH).

• Літерали– рядок букв, цифр та інших символів у лапках(або між апострофами). Дві форми передбачено для того, щоб за необхідністю
вставляти в текст самі лапки або апострофи. Наприклад: ”Значення функції”, або ’Значення функції’; ”Відповідь на питання ’Ваш вибір’”.

• Від'ємні числа. Якщо число– десяткове, то перед ним ставлять знак
мінус(наприклад, -26). Якщо число двійкове або шістнадцяткове, то його вводять у доповнювальному коді. Число –32: 11100000B або 0E0H.

9.2 Формат мови асемблер

Кожну команду мови асемблера записують в один рядок. Формат рядка:

Поле імені	*Поле операції*	*Поле операндів*	*Поле коментаря*

або:

Поле (рядок) коментаря

Поля відокремлюють між собою одним чи декількома інтервалами (пропусками). Обов'язковими є лише поле операції та поле операндів, якщо операція вимагає операндів. Поле чи рядок коментаря розпочинають знаком ';'. Операнди розділяють між собою комами.

Неприпустимо використовувати пропуски всередині поля імені, поля операції чи поля операндів. Однак пропуск може бути літерою – значенням операнда у лапках.

Вважатимемо, що імена налічують не більше, ніж вісім латинських букв і цифр, а їхній запис обов'язково розпочинається буквою. Великі та малі букви вважатимемо однаковими. Наприклад:

ABc12, AbC12, Abc12, abc12 – це однакові імена.

Бітом називають двійкову цифру, одиничне значення 0 або 1. Групу з8-ми бітів називають байтом. Байт заслужив своє власне ім'я з декількох причин. Елементарний елемент пам'яті (комірка) має довжину8 бітів.

Упродовж кожного звертання до пам’яті процесор опрацьовує рівно8 бітів інформації. Байт- найменша одиниця інформації, з якою можна маніпулювати безпосередньо. Крім того, байт використовують для зображення одного символу. За допомогою одного байта можна визначити 256 (2^8) окремих символів(межі для цілих чисел: -128 … 127).

Для розміщення певного значення в одному байті пам'яті асемблер має в своєму розпорядженні спеціальний механізм визначення байта (define byte) – псевдокоманду DB формату:

[< ім’я>] DB < число або літерал>

або

[< ім’я>] DB ?

Отже, псевдокоманда DB:

- формує однобайтову константу – число;
- формує у пам’яті заданий рядок символів(літерал);
- просто резервує1 байт пам’яті(за умови наявності ”?”).

Псевдокоманда

R1 DB 23

дає асемблерові завдання зберегти десяткове значення23 у деякому байті пам'яті, зазначеному як R1. Позначку R1 можна трактувати як назву(ім’я) константи(значення байта R1 у програмі змінюватися не буде) або назву змінної(значення байта R1 у програмі змінюватиметься).

Псевдокоманда

R2 DB 1,2,3,4

зберігає значення від1 до4 за чотирма послідовними адресами в пам'яті.

Псевдокоманда

Wer DB ?

повідомляє асемблеру щодо необхідності виокремити (зарезервувати) один байт пам'яті, не визначаючи початково його вміст. У цьому байті може виявитися будь-яке випадкове число, яке там залишатиметься, доки будь-яка команда не вмістить у нього певне значення.

Часом треба виокремити значну кількість байтів, наприклад, щоб зарезервувати область пам'яті для масиву. Це можна зробити так:

Mas1 DB 25 DUP(?)

У такий спосіб виокремлюють25 байтів пам'яті. Ключове слово DUP у цій псевдокоманді означає повторити (duplicate). Число25 вказує, скільки разів асемблер повторить визначення байта у пам'яті. Значення в дужках асемблер використовує з метою ініціалізації цієї області пам'яті. У нашому випадку це значення невідоме. Для ініціалізації області з ідентичним значенням, наприклад 31, записують так:

Mas2 DB 17 DUP(31)

Нарешті,

Mas3 DB 30 DUP(1,2,3,4,5)

виокремлює 30 груп по5 байтів(загалом150 байтів) зі значеннями від1 до 5. Асемблер повторює значення в дужках доти, доки не буде заповнено усі 30 груп байтів.

Іноді необхідно звернутися до групи бітів, меншої від байта. Прийнято розмір4 біти. У 4-х бітах можна зобразити усі10 десяткових цифр. Для значень такого розміру використовують термін півбайт. Термін слово має для програміста значення, відмінне від прийня-того у мові. У застосуванні до ЕОМ слово- це найбільша кількість бітів, з якою машина може працювати як з єдиним елементом. Для IBM/370 слово становить 32 біти, а для Intel 8088 – 16 бітів. Тому термін слово є невизначеним доти, доки не визначено конкретної машини.

9.3 Регістри загального призначення (AX, BX, CX, DX)

Загальні регістри використовують переважно з метою обчислень. Усі вони мають розмір 16 бітів, однак програма може працювати зі старшими чи молодшими 8-ма бітами кожного регістра окремо. Наприклад, регістр AX налічує 16 бітів. Програма може звернутися до старших 8-ми бітів AX як до регістра AH, а молодші 8 бітів утворять регістр AL. Те ж саме стосується регістрів BX, CX і DX. Програма може розглядати цю групу регістрів як чотири 16-бітових, вісім 8-бітових або деяку комбінацію 8- і 16-бітових регістрів.

Головне призначення загальних регістрів – зберігати операнди. Загальні регістри зберігають як слово, так і байт даних. Однак ці регістри під час виконання певних операцій мають спеціальне призначення.

Регістр AX відповідає суматорові(акумуляторові). Безпосередні операції з регістрами AX і AL(16- і 8-бітовий суматори, відповідно) зазвичай вимагають значно коротшої команди, ніж аналогічні операції із залученням інших регістрів загального призначення. А менший розмір команди дає змогу отримати компактніші й швидкодіючіші програми.

Регістр BX є і регістром для обчислень, і адресним регістром. Використвуючи його як 16-бітовий, ним можна визначати адреси операнда. Способи (режими) адресування перелічено в окремому пункті.

Регістр CX використовують як лічильник до деяких команд. Ці команди використовують значення, що розташоване в CX як покажчик числа ітерацій команди або фрагмента програми.

Регістр DX слугує як розширення акумулятора для багаторозрядних операцій множення і ділення. У цих 32-бітових операціях беруть участь одночасно регістри AX і DX.

9.4. Сегментні регістри (CS, DS, SS, ES)

Процесор має чотири сегментних регістри: CS, DS, SS і ES - відповідно для кодового, даних, стекового і додаткового сегментів. Це їхнє звичайне використання, проте застосування цих регістрів може змінюватися відповідно до потреб програми. Процесор використовує регістр сегмента програми (коду) для ідентифікації того сегмента, який містить програму, що виконується в даний момент. У поєднанні з покажчиком команд регістр CS використовують для вказівки поточної команди. Кожна команда, що виконується, перебуває в осередку, на який вказує пара регістрів CS:IP.

Комбінацію сегментного регістра з регістром зміщення для вказівки фізичної адреси записують у вигляді сегмента:зміщення(наприклад, CS:IP). Значення сегмента стоїть перед двокрапкою, зміщення -після. Таку нотацію використовують і для регістрів, і для абсолютних адрес. Можна записати так: CS:100, DS:BX, 570:100, або 630:DI.

Регістр сегмента даних(DS) процесор використовує з метою звичайного доступу до даних. Схеми адресування для операндів дають 16-бітове зміщення, і переважно для формування виконавчої адреси процесор комбінує це зміщення з регістром DS.

Регістр сегмента стека вказує на системний стек. Команди PUSH, POP, CALL iRET керують даними у стеку за адресою SS:SP. Регістр SP – вказівник стека – слугує для визначення зміщення у стекові.

Крім того, сегмент стека беруть до уваги за домовленістю під час адресування з використанням регістра BP. Це дає доступ до даних у стеку з використанням регістра BP.

Нарешті, регістр додаткового сегмента використовують з метою доступу до даних, коли потрібно більше одного сегмента. Звичайною операцією такого рівня є копіювання даних з однієї ділянки пам'яті в іншу. Між ділянками, розташованими поза одним і тим же блоком пам'яті розміром 64К, неможливо зробити обмін даними, використовуючи єдиний сегментний регістр. Володіючи додатковим сегментним регістром, програма може визначити одночасно початковий і цільовий сегменти.

Регістри адресування (BX, BP, SI, DI)

У процесорі є чотири 16-бітових регістри, які можуть брати участь в адресуванні операндів. Один з них є одночасно регістром загального призначення – регістр бази BX.

Програма являє собою послідовність команд, виконання яких приводить до розв'язку задачі.

Команда визначає операцію, яку має виконати МП над даними. Команда містить у явній або неявній формах інформацію про те, де буде розміщений результат операції, а також про адресу наступної команди. Код команди складається з декількох частин, які називаються *полями.* Склад, призначення і розташування полів називається *форматом команди*. У загальному випадку формат команди містить операційну та адресну частини. Операційна частина містить код операції (наприклад, додавання, множення, передача даних). Адресна частина складається з декількох полів і містить інформацію про адреси операндів, результату операції та наступної команди. Формат команди, у якому адресна частина складається з двох полів (ознаки адресації та адреси операндів) показано на рис. 9.1.

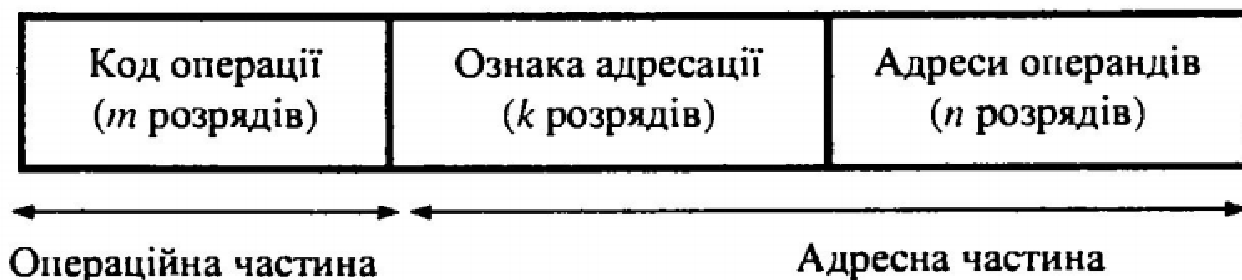


Рис.9.1. Формат команди

Поле «Ознака адресації» визначає спосіб адресації операнда. Біти полів «Ознака адресації» та «Адреси операндів» разом визначають комірки пам'яті, в яких зберігаються операнди.

Розрізняють такі групи команд: 1) команди передачі даних; 2) команди операцій введення-виведення; 3) команди обробки інформації (арифметичні, логічні, зсуву, порівняння операндів, десяткової корекції);

4) команди керування порядком виконання програми (переходів, викликів підпрограм, повернень з підпрограм, переривань); 5) команди задання режимів роботи МП.

Загальна кількість бітів у коді команди називається *довжиною формату.* Кількість двійкових розрядів *m* у полі «Код операції» забезпечує можливість подання всіх операцій, які виконує МП. Якщо МП виконує М різних операцій, то кількість розрядів визначається так:

$$т > \log_2 M.$$

Якщо пам'ять містить *S* комірок, то потрібна для запису адреси одного операнда кількість розрядів у полі «Адреси операндів» становить:

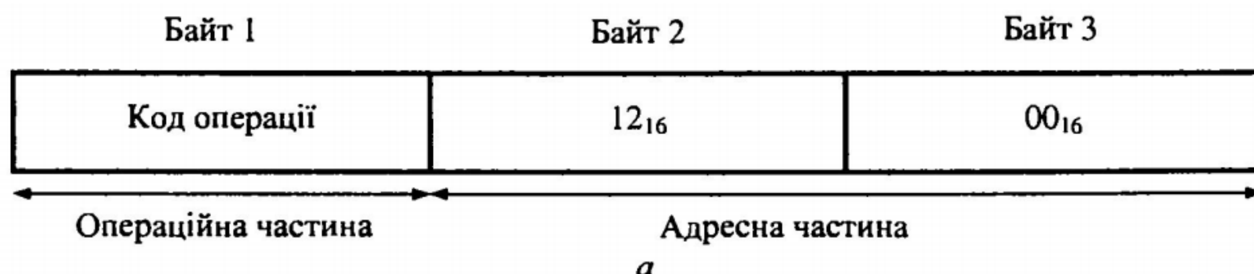
$$n > \log_2 S.$$

Довжина формату команди визначає швидкість виконання команди і залежить від способу адресації операндів. Існують такі способи адресації: пряма, непряма, безпосередня, автоінкрементна (автодекрементна), сторінкова, індексна, відносна.

Пряма адресація. При такій адресації адреса операнда вказана безпосередньо в команді. Як приклад розглянемо команду МП К580ВМ80А (/8080) прямого завантаження акумулятора вмістом комірки пам'яті, розміщеної за адресою 0012_{16}. Формат та схему виконання цієї команди показано на рис. 2.11.

У байті 1 команди (рис. 9.2, а) міститься код операції пересилання даних в акумулятор із комірки пам'яті, а в байтах 2 і 3 - адреса комірки пам'яті. У байті 2 розташований молодший (12іб), а в байті 3 - старший (00іб) байт адреси.

На рис. 9.2, *б* комірка пам'яті з адресою 0012^ має вміст 110101112- Вміст акумулятора до операції становить 00000000_2. Після виконання команди значення вмісту комірки пам'яті копіюється в акумулятор.



а

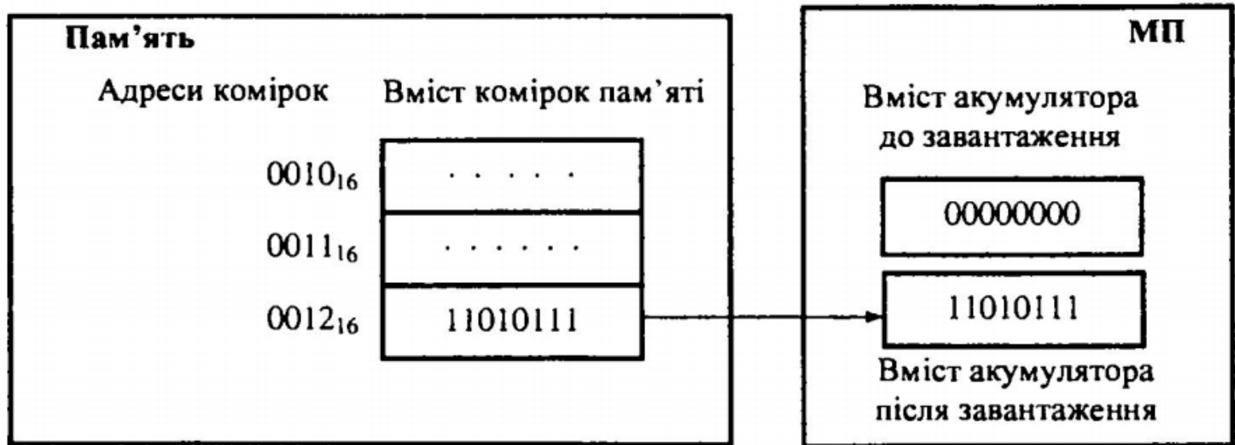


Рис. 9.2. Команда прямого завантаження в акумулятор вмісту комірки пам'яті: *а* - формат команди; *б* - схема виконання

Непряма адресація. При такій адресації у форматі команди вказується номер регістра, у якому зберігається адреса комірки пам'яті, яка містить операнд. Для збереження 16-розрядної адреси у 8-розрядному процесорі 8-розрядні регістри об'єднуються у регістрові пари. У першому регістрі регістрової пари зберігається старший байт адреси, а в другому - молодший. Номер регістрової пари, в якій зберігається адреса, є 2-роз- рядним двійковим числом, тому він розміщується в однобайтовій команді разом з кодом операції.

Як приклад виконання команди МП К580ВМ80А непрямого завантаження в акумулятор вмісту комірки пам'яті з адресою 0012іб, яка зберігається в регістровій парі *DE*, показано на рис. 9.3.

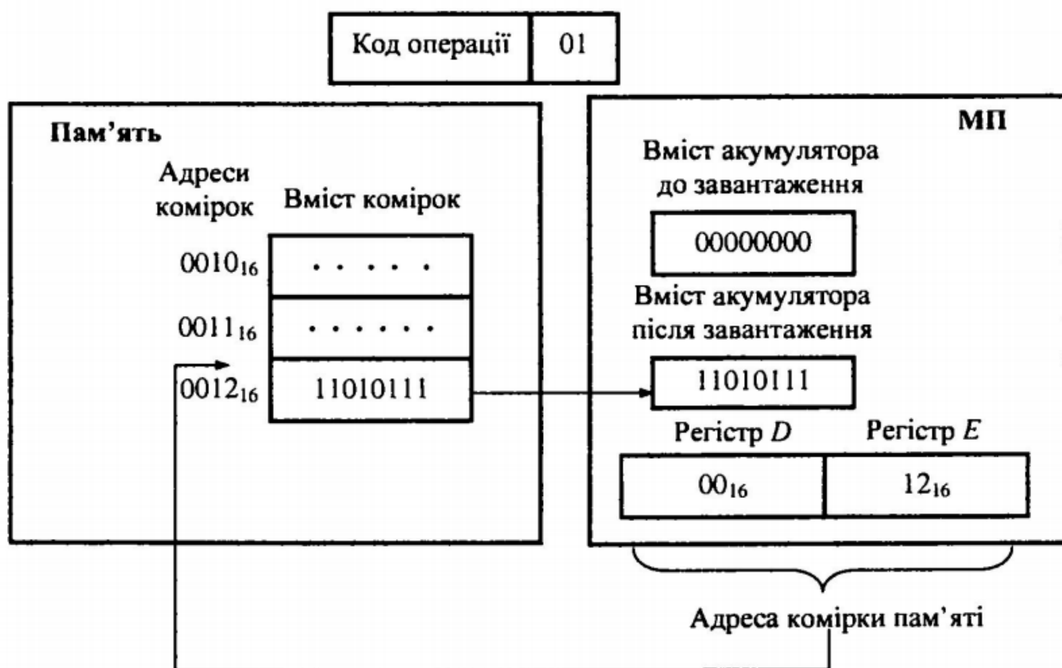


Рис. 9.3. Команда непрямого завантаження акумулятора: *а* - формат команди; *б* - схема виконання

Команда непрямого завантаження акумулятора є однобайтовою і, крім коду операції, містить номер 01 регістрової пари *DE.* Старша частина адреси комірки пам'яті (OOiб) зберігається у регістрі Д а молодша частина ($12i_6$)-у регістрі Е.

Вміст регістрової пари передається в регістр адреси МП, у результаті чого вміст 110101112 комірки з адресою $0012i_6$ копіюється в акумулятор.

Безпосередня адресація. У першому байті команди з безпосередньою адресацією розміщується код операції. Значення операндів заносяться в команду під час програмування і знаходяться у другому або другому і третьому байтах.

Цими значеннями зазвичай є деякі константи, заздалегідь відомі програмісту. У процесі виконання програми значення операндів залишаються незмінними, оскільки вони разом із командою розмішуються в ПЗП. Використання такого способу не потребує адреси операндів. Як приклад на рис. 9.4 зображено формат і схему виконання команди безпосереднього завантаження акумулятора значенням 110101112, яке зберігається у другому байті команди. Після виконання команди це число копіюється в акумулятор. При кожному черговому зверненні до цієї команди в акумулятор записується таке саме число.

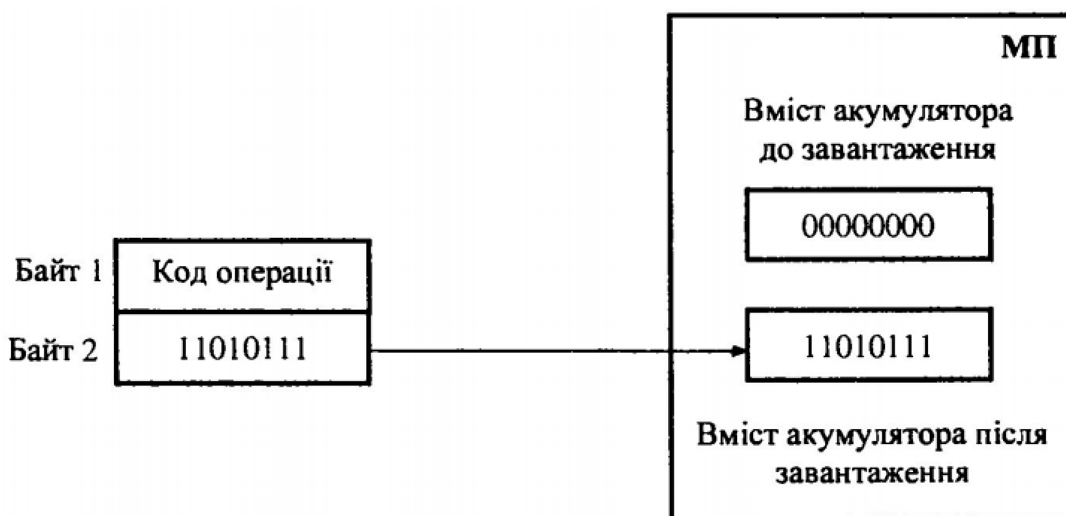


Рис. 9.4. Команда безпосереднього завантаження акумулятора

Автоінкрементна (автодекрементна) адресація. При автоінкремент ній адресації адреса операнда обчислюється так само, як і при непрямій адресації, а потім здійснюється збільшення вмісту регістра: на один - для звернення до наступного байта, на два - для звернення до наступного слова. Розмір операнда визначається кодом операції.

Сторінкова адресація. Під час використання сторінкової адресації пам'ять поділяється на ряд сторінок однакової довжини. Адресація сторінок здійснюється або з програмного лічильника, або з окремого

регістра сторінок. Адресація пам'яті всередині сторінок здійснюється адресою, що міститься в команді.

Індексна адресація. Для утворення адреси операнда до знаення адресного поля команди додається значення вмісту індексного регістра, яке називається індексом.

Відносна адресація. При відносній адресації адреса операнда визначається додаванням вмісту програмного лічильника або іншого регістра із зазначеним у команді числом. Вміст програмного лічильника або іншого регістра називається *базовою адресою*. Для збереження базових адрес у МП можуть бути передбачені базові регістри або спеціально виділені комірки пам'яті. Тоді в адресному полі команди вказується номер базового регістра.

У МПС використовується програмування мовою асемблер. *Асемблером* називається і мова програмування у мнемокодах команд, і спеціальна програма-транслятор, що переводить (транслює) мнемокоди у машинні коди, які зчитуються мікропроцесором із пам'яті програм, дешифруються і виконуються. Процес переведення у машинні коди називається *асемблюванням.*

Програма на мові асемблер містить два типи виразів:

1. команди, що транслюються в машинні коди;
2. директиви, що керують ходом трансляції.

Вираз має вигляд:

{<мітка>}: <мнемокод> {<операнд>}{,}{< операнд >}{; коментар}.

У фігурних дужках наведено елементи виразу, яких може не бути у деяких командах. Мітка, мнемокод і операнди відокремлюються хоча б одним пробілом або табуляцією. Максимальна довжина рядка становить 132 символи, проте найчастіше використовуються рядки з 80 символів, що відповідає довжині екрана.

Прикладами команд асемблера є:

Мітка	Мнемокод	Операнд(и)	Коментар
	MOV	AX, 0	; Команда, два операнди
MI:	ADD	AX, BX	; Мітка, команда, два операнди
DELAY:	MOV	CX, 1234	; Мітка, команда, два операнди

Прикладом директиви є: Мітка Мнемокод

Операнд(и) Коментар
COUNT: DB 1 ; Мітка, команда, один операнд

Мітка у мові асемблера є символічною адресою команди. Мітками позначаються не всі команди, а лише ті, до яких треба виконувати перехід за допомогою команд переходів або викликів підпрограм. У командах переходів або викликів підпрограм позначення мітки використовується як операнд - символічна адреса переходу, наприклад:

Мітка Мнемокод Операнд(и) Коментар

```
JMP     MI      ; Перехід до команди з міткою MI
CALL    DELAY   ; Виклик підпрограми з міткою
                ; DELAY
```

Після мітки ставиться двокрапка. Першим символом у мітці має бути літера або один із спеціальних символів: знак питання «?»; крапка «.»; знак амперсанд «@»; підкреслювання «_»; знак долара «$». Знак питання і крапка можуть займати тільки перше місце. Максимальна довжина мітки - 31 символ. Приклади міток: *COUNT, PAGE25*, $£10. Рекомендується використовувати описові та смислові мітки. Усі мітки у програмі мають бути унікальними, тобто не може бути декількох команд з однаковими мітками. Як мітки не можна використовувати зарезервовані асемблером слова, до яких належать коди команд, директиви, імена регістрів. Наприклад, імена *AX*, *DI* та *AL* є зарезервованими і використовуються тільки для зазначення відповідних регістрів.

Мнемокод ідентифікує команду асемблера. Для мнемокодів використовують скорочені або повні англійські слова, які передають значення основної функції команди: *ADD* - додати, *SUB (SUBtract)* - відняти, *XCHG (eXCHanGe)* - поміняти.

Операнди відокремлюються комами. Якщо задано два операнди, то перший з них завжди є приймачем, а другий - джерелом інформації. Команда може містити різну кількість операндів різних типів, наприклад:

```
Мітка Мнемокод Операнд(и) Коментар
        RET     MI          ;  Повернутися (операнди не вказані)
        INC     CX          ;  Збільшити CX (один операнд)
        ADD     AX, 12H ;      Додати 12H до вмісту AX
                            ;  (два операнди)
        MOV     BX, [SI] ;     Занести до регістра BX число
                            ;  з комірки пам'яті з адресою
                            ;  DS:SI (два операнди)
```

Коментарі ігноруються у процесі трансляції і використовуються для документування і кращого розуміння змісту програми. Коментар завжди починається із символу «;» і може містити будь-які символи. Коментар може займати увесь рядок або бути розташованим за командою в одному рядку, наприклад:

```
Мітка Мнемокод Операнд(и) Коментар
                            ; Цей рядок є коментарем ADD  AX,
        BX          ; Команда і коментар в одному рядку
```

Оскільки коментарі не транслюються у машинні коди, то їх кількість не впливає на ефективність виконання програми.

Програма мовою асемблер називається *початковою програмою* або *початковим програмним модулем*. Асемблювання або переведення початкової програми у машинні коди виконує програма-транслятор, наприклад *TASM.COM*. Залежно від установок, які задає користувач,

програма переводить початковий модуль в один із двох програмних модулів: командний модуль (файл з розширенням .COM) або об'єктний модуль (файл з розширенням *.OBJ).*

Командний модуль містить машинні коди команд з абсолютними адресами і виконується МП. Командний модуль доцільно використовувати у тих випадках, коли ємність програми не перевищує розміру одного сегмента (64 кбайт). Першим оператором командного модуля є директива *ORG* 100Я *(ІORIGIN* - початок), яка розміщує першу команду програми у сегменті кодів зі зміщенням 100Я. Закінчуватися програма має або командою *RET,* або стандартною процедурою коректного виходу до *MS DOS:*

Мітка Мнемокод Операнд(и) Коментар

```
        MOV         AH, 4CH ; Занести у AH число 4CH
                              ; (значення параметра переривання ;
                              INT 21H)
        I NT        21H       ; Викликати стандартну процедуру
                              ; переривання 21H - коректного ;
                              виходу до MS DOS
```

Останнім записом програми має бути директива *END.*

Об'єктний модуль містить машинні коди команд з відносними адресами. Об'єктний модуль виконується МП після заміни відносних адрес на абсолютні за допомогою програми-укладача, наприклад *LINK.EXE*, яка генерує модуль з розширенням .EXE (ЕАЕ-файл або EXE-програму); EXE- файл, на відміну від командного модуля, може перевищувати обсяг одного сегмента. Однак у цьому разі обов'язковим є визначення сегментів за допомогою директив асемблера. Закінчується EXE-^шл стандартною процедурою коректного виходу до *MS DOS.*

Програма-укладач має ще одне призначення - вона об'єднує об'єктний модуль з бібліотечними модулями або кілька окремих об'єктних модулів в один *EXE*-файл. *Бібліотечними модулями* називають об'єктні файли, які містять найбільш поширені підпрограми. Бібліотечні модулі розміщуються у спеціальному системному файлі - бібліотеці *(LIBRARY).*

При асемблюванні програма-транслятор генерує лістинг і файл лістингу програми. *Лістинг* - це відображення на дисплеї або папері текстів початкового програмного модуля, програмного модуля *(.COM* або *.OBJ)* та повідомлень, які вказують на помилки програмування, зумовлені порушенням правил запису виразів (наприклад, немає операнда або неправильний мнемокод команди).

Директиви призначені для керування процесом асемблювання і формування лістингу. Вони діють тільки у процесі асемблювання програми і не переводяться в машинні коди. Мова асемблер містить такі основні директиви:

1. початку і кінця сегмента *SEGMENT* та *ENDS;*
2. початку і кінця процедури *PROC* та *ENDP;*

3. призначення сегментів *ASSUME;*
4. початку *ORG*
5. розподілу та ініціювання пам'яті *DB, DW, DD*
6. завершення програми *END;*
7. відзначення *LABEL.*

Директиви початку і кінця сегмента SEGMENT та ENDS призначені для опису сегментів, які використовує програма. Директиви початку і кінця сегмента використовуються разом, наприклад:

```
Назва   Мнемокод  Операнд(и)
DATASG  SEGMENT   {<параметри>}
          . }
          . }     Інші команди або директиви сегмента
          . }
DATASG  ENDS
```

Обидві директиви SEGMENT і ENDS повинні мати однакові назви. Директива SEGMENT може містити три типи параметрів: вирівнювання, об'єднання і класу.

Параметр вирівнювання визначає початкову адресу або межу сегмента, наприклад:

PAGE = *xxx*00,
PARA = xxxx00 (межа за замовчуванням),
WORD = xxxx0 (парна межа),
BYTE = xxxxx,

де *x* – будь-яка шістнадцяткова цифра; *e* - парна шістнадцяткова цифра. Якщо немає параметра вирівнювання за замовчуванням, береться параметр PARA, який вказує на те, що сегмент розміщується на початку параграфа, а початкова адреса сегмента є кратною 16. Параграфом називається область пам'яті розміром 16 байт, початкова адреса якої кратна 16, тобто має праворуч чотири нульові розряди.

Параметр об'єднання вказує на спосіб обробки сегмента при компонуванні декількох програмних модулів:

NONE: значення за замовчуванням. Сегмент має бути логічно відокремленим від інших сегментів, хоча фізично він може розміщуватися поряд. Передбачається, що сегмент має власну базову адресу.

PUBLIC: усі PUBLIC - сегменти з однаковими назвою та класом завантажуються у суміжні області та мають одну базову адресу.

STACK: призначення аналогічне параметру PUBLIC. У будь-якій програмі має бути визначений принаймні один сегмент STACK. Якщо визначено більше одного стека, то вказівник стека SP (Stack Pointer) встановлюється на початок першого стека.

COMMON: для сегментів COMMON з однаковими назвами та класами встановлюється одна спільна базова адреса. Під час виконання

програми здійснюється накладання другого сегмента на перший. Розмір загальної області визначається найдовшим сегментом.

АТ-параграф: параграф слід визначати заздалегідь. Цей параметр забезпечує визначення міток та змінних за фіксованими адресами у фіксованих областях пам'яті.

'Клас': цей параметр може мати будь-яку правильну назву, яка розміщується в одинарних лапках. Параметр використовується для обробки сегментів, які мають однакові назви та класи. Типовими є класи '*STACK** та [4]*CODE*\ наприклад:

Назва Мнемокод Операнд
STACKSG SEGMENT PARA STACK [4] STACK'

У випадку, якщо програма не має об'єднуватися з іншими програмами, параметр об'єднання не вказується.

Директиви початку і кінця процедури *PROC* та *ENDP* використовуються для визначення підпрограм у сегменті кодів і мають такий формат:

<Назва> *PROC* {<тип процедури;»}.

Можливі два типи процедур:

- *NEAR* - процедура знаходиться в тому самому сегменті, що і команди, які її викликають;
- *FAR* - процедура знаходиться за межами сегмента. За замовчуванням береться тип процедури *NEAR.*

Сегмент кодів може містити декілька процедур. Описання сегмента кодів, що містить тільки одну процедуру, має такий вигляд:

```
Назва                  Мнемокод Операнд
ім'я _ сегмента SEGMENT PARA
ім'я _ процедури PROC         FAR
                              RET
ім'я_процедури ENDP ім'я_сегмента ENDS
```

Ім'я процедури має бути обов'язково і збігатися з іменем у директиві *ENDP*, яка визначає кінець процедури.

Директива призначення сегментів *ASSUME* використовується для встановлення відповідності між сегментами та сегментними регістрами і має такий формат:

ASSUME <сегментний регістр> : <ім'я> {,}{...} .

Наприклад, запис 55:ім_стек вказує, що ім'я стека визначається вмістом регістра *SS*. Одна директива *ASSUME* може призначати до чотирьох сегментних регістрів у будь-якій послідовності, наприклад:

Мнемокод Операнд(и)
ASSUME SS:ім_стек/ DS:iM_flaHi, C5:ім_код, ES:ім_додаткові_дані

Для скасування будь-якого призначеного раніше у директиві *ASSUME* сегментного регістра треба використовувати слово *NOTHING:*

Мнемокод Операнд(и)

ASSUME ES: NOTHING

Якщо програма не використовує якийсь сегмент, то відповідний йому операнд можна пропустити або вказати слово *NOTHING.*

Директива *ORG* використовується для зміни вмісту програмного лічильника без команд умовного чи безумовного переходу. Найчастіше цю директиву використовують для встановлення початкової адреси програми, наприклад, директива *ORG* 100// встановлює програмний лічильник на зміщення 100// відносно початку сегмента кодів. Операнд зі знаком долара «$» має поточне значення програмного лічильника. Наприклад, директива *ORG* $ + 10// збільшує адресу, завантажену у програмний лічильник, на 10//.

Директиви розподілу та ініціювання пам'яті використовуються для визначення вмісту та резервування комірок пам'яті.
Директива має формат:
{ <ім'я> } *Dn* {кількість повторень *DUP*}<вираз>,

де мнемокод $Dn = \begin{Bmatrix} DB \\ DW \\ DD \\ DQ \\ DT \end{Bmatrix}$ вказує на довжину даних: *DB* – байт; *DW* – слово (2 байт); *DD* - подвійне слово; *DQ* - чотири слова; *DT*- 10 байт. Якщо у форматі наявне ім'я, то далі у програмі воно може використовуватися для позначення комірки пам'яті.

<Вираз> у форматі директиви містить одну або кілька констант для задання початкових значень вмісту комірок пам'яті або знак «?» для невизначеного значення вмісту. Наприклад, директива *ALPHA DB* 34 означає, що комірка пам'яті з іменем *ALPHA* містить число 34. У ході виконання програми вміст комірки може бути змінений. Директива *BETA DW* ? визначає, що комірка з іменем *BETA* має розрядність 16, але вміст комірки є невизначеним. Директива може містити декілька констант, розділених комами й обмежених лише довжиною рядка. Наприклад, вираз

ARRAYDB 01,02, 11, 12,21,22

визначає 6 констант у вигляді послідовності суміжних байтів. Посилання на комірку з іменем *ARRAY* вказує на першу константу (01), з іменем *ARRA Y*+ 1 - на другу (02), з іменем *ARRA*К + 2 - на третю (11) і т. д. Запис

MOVAL, ARRAY + 4

завантажує у регістр *AL* значення 21.

Одна директива може визначити декілька комірок пам'яті. У цьому разі директива має вигляд:
{ <ім'я> } *Dn* (кількість повторень} *DUP* <вираз>.

Наприклад, директива, що визначає 5 байт, які містять число 21, записується так:

DB 5 DUP (21).

Директива завершення програми *END* є останньою у програмі та має такий формат:

END (<стартова адреса>}.

Параметр директиви стартова адреса> використовується лише при створенні EXE-файлів.

Директива мітки *LABEL* призначена для встановлення відповідності між іменем і типом змінних. Вона має такий формат:

<ім'я> LABEL {<тип>}.

*Я*к тип можна використовувати слова *BYTE, WORD, DWORD*, що визначають довжину даних: байт, слово або подвійне слово. Директива *LABEL* перевизначає параметри процедур *NEAR* або *FAR*. Наприклад, директива

TOS LABEL WORD

присвоює комірці пам'яті ім'я *TOS* і зазначає, що її вміст є словом.