ПРАКТИЧНЕ ЗАНЯТТЯ 7
ВИВЧЕННЯ ЗАСОБІВ ОРГАНІЗАЦІЇ РОЗГАЛУЖЕНЬ І ЦИКЛІВ МОВОЮ АСЕМБЛЕРА

7.1 Мета заняття

Ознайомитись з засобами організації циклів умовних переходів.

7.2 Теоретичні положення

Дуже часто при обробці рядків, масивів даних необхідно здійснювати послідовне порівняння одного або декількох елементів масиву з іншими. Для розв’язання цього завдання на мовах високого рівня застосовують цикли, проте, під час програмування мовою Асемблера доцільно застосовувати операції з рядками.

Операції з рядками – це набір команд, який дозволяє провести циклічну обробку всього рядка (масиву) виконавши одну команду процесора. Перед виконанням такої команди необхідно здійснити ініціалізацію декількох регістрів, вказавши, де знаходиться рядок (елемент) джерело і де знаходиться рядок (елемент) приймач, а також кількість повторень команди.

7.2.1. Операції з рядками

Всі команди для роботи з рядками (масивами) вважають, що рядок-джерело знаходиться за адресою DS: ESI (або DS: SI), тобто у сегменті пам'яті, яка адресується сегментним регістром DS зі зміщенням у ESI, а рядок-приймач – відповідно у ES: EDI (або ES: DI). Зокрема, всі команди роботи з рядками працюють тільки з одним елементом рядка (байтом (BYTE), словом (WORD) або подвійним словом (DWORD)) за один раз. Для того щоб команда виконувалася над усім рядком, необхідний один з префіксів повторення операцій.

Таблиця 1. Перелік префіксів повторення операції.

Команда	Дія
REP	Повторювати
REPE	Повторювати, поки одно
REPNE	Повторювати, поки не дорівнює
REPZ	Повторювати, поки нуль
REPNZ	Повторювати, поки не нуль

Будь-який з префіксів виконує наступну за ним команду обробки рядків стільки разів, скільки вказано у регістрі ECX (або CX, залежно від розрядності адреси), зменшуючи його при кожному виконанні команди на 1. Зокрема, префікси REPZ і REPE припиняють повторення команди, якщо прапор ZF скинутий у 0, а префікси REPNZ і REPNE припиняють повторення, якщо прапор ZF встановлений у 1. Префікс REP, зазвичай, використовується з командами INS, OUTS, MOVS, LODS, STOS, а префікси REPE, REPNE, REPZ і REPNZ – з командами CMPS і SCAS. Поведінка префіксів без команд обробки рядків не визначена.

Для вказівки того, в який бік (зменшення або збільшення) повинен змінитися поточний індекс рядка (масиву) після виконання чергової дії, служить прапор напрямку – DF. Після виконання команди регістри ESI (SI) і EDI (DI) збільшуються на 1, 2 або 4 (якщо копіюються байти, слова або подвійні слова відповідно), якщо прапор DF = 0, і зменшуються, якщо DF = 1. Для установки в 1 прапора напрямку DF служить команда процесора STD, для скидання його у 0 служить команда процесора CLD.

Усі команди рядкової обробки мають кілька варіацій в іменуванні, які вказують на розмірність оброблюваних даних. При використанні базової команди асемблер сам визначає, яку команду поставити на підставі розмірності операндів, зазначених у команді. Перелік всіх команд вказано в таблиці.

Таблиця 2. Перелік команд.

Команда	MOVS	CMPS	SCAS	LODS	STOS	INS	OUTS
Призначення	Копіювання рядка	Порівняння рядків	Сканування рядків	Читання з рядка	Запис до рядка	Читання з рядка порту	Запис рядка у порт
Байти	MOVSB	CMPSB	SCASB	LODSB	STOSB	INSB	OUTSB
Слова	MOVSW	CMPSW	SCASW	LODS	STOSW	INSW	OUTSW
Подвійні слова	MOVSD	CMPSD	SCASD	LODS	STOSD	INSD	OUTSD

Під час використання форми запису MOVS асемблер сам визначає з типу зазначених операндів (прийнято вказувати імена рядків, які потрібно скопіювати, але можна використовувати будь-які два операнди відповідного типу), яку з трьох форм цієї команди (MOVSB, MOVSW або MOVSD) обрати. Копіює один байт (MOVSB), слово (MOVSW) або подвійне слово (MOVSD) з пам'яті за адресою DS: ESI (або DS: SI, залежно від розрядності адреси) у пам'ять за адресою ES: EDI (або ES: DI). Використовуючи MOVS з операндами, можна замінити регістр DS на інший за допомогою префікса заміни сегмента (ES :, GS :, FS :, CS :, SS :), регістр ES замінити не можна. Після виконання команди регістри ESI (SI) і EDI (DI) збільшуються на 1, 2 або 4 (якщо копіюються байти, слова або подвійні слова), якщо прапор DF = 0, і зменшуються, якщо DF = 1. При використанні з префіксом REP команда MOVS виконує копіювання рядка довжиною в ECX (або CX) байт, слів або подвійних слів.

2. Порівняння рядків.

CMPS приймач, джерело

Порівнює один байт (CMPSB), слово (CMPSW) або подвійне слово (CMPSD) з пам'яті за адресою DS: ESI (або DS: SI, залежно від розрядності адреси) з байтом, словом або подвійним словом за адресою ES: EDI (або ES: DI) і встановлює прапори аналогічно команді CMP.

Під час використання форми запису CMPS асемблер сам визначає з типу зазначених операндів (прийнято вказувати імена порівнюваних рядків, але можна використовувати будь-які два операнда відповідного типу), яку з

трьох форм цієї команди (CMPSB, CMPSW або CMPSD) обрати. Використовуючи CMPS з операндами, можна замінити регістр DS на інший, застосовуючи префікс заміни сегмента (ES :, GS :, FS :, CS :, SS :), регістр ES замінити не можна. Після виконання команди регістри ESI (SI) і EDI (DI) збільшуються на 1, 2 або 4 (якщо порівнюються байти, слова або подвійні слова), якщо прапор DF = 0, і зменшуються, якщо DF = 1. При використанні з префіксом REP команда CMPS виконує порівняння рядка довжиною в ECX (або CX) байт, слів або подвійних слів, але частіше її використовують з префіксами REPNE/REPNZ або REPE/REPZ. У першому випадку порівняння триває до першої розбіжності у порівнюваних рядках, а у другому – до першого збігу.

3. Сканування рядка.

SCAS приймач

Порівнює вміст регістра AL (SCASB), AX (SCASW) або Eax (SCASD) з байтом, словом або подвійним словом з пам'яті за адресою ES: EDI (або ES: DI, залежно від розрядності адреси) і встановлює прапори аналогічно команді CMP. При використанні форми запису SCAS асемблер сам визначає з типу зазначеного операнда (прийнято вказувати ім'я сканованого рядка, але можна використовувати будь-який операнд відповідного типу), яку з трьох форм цієї команди (SCASB, SCASW або SCASD) вибрати. Після виконання команди регістр EDI (DI) збільшується на 1, 2 або 4 (якщо скануються байти, слова або подвійні слова), якщо прапор DF = 0, і зменшується, якщо DF = 1. При використанні з префіксом REP команда SCAS виконує сканування рядка довжиною в ECX (або CX) байт, слів або подвійних слів, але частіше її використовують з префіксами REPNE/REPNZ або REPE/REPZ. У першому випадку сканування триває до першого елемента рядка, відмінного від вмісту акумулятора, а в другому – до першого збігається.

4. Читання з рядка

LODS джерело

Копіює один байт (LODSB), слово (LODSW) або подвійне слово (LODSD) з пам'яті за адресою DS: ESI (або DS: SI, залежно від розрядності адреси) в регістр AL, AX або EAX відповідно. При використанні форми записи LODS асемблер сам визначає з типу зазначеного операнда (прийнято вказувати ім'я рядка, але можна використовувати будь-який операнд відповідного типу), яку з трьох форм цієї команди (LODSB, LODSW або LODSD) обрати. Використовуючи LODS з операндом, можна замінити регістр DS на інший за допомогою префікса заміни сегмента (ES :, GS :, FS :, CS :, SS:). Після виконання команди регістр ESI (SI) збільшується на 1, 2 або 4 (якщо зчитується байт, слово або подвійне слово), якщо прапор DF = 0, і зменшується, якщо DF = 1. При використанні з префіксом REP команда LODS виконає копіювання рядка довжиною в ECX (або CX), що призведе до того, що в акумуляторі виявиться останній елемент рядка. Насправді цю команду використовують без префіксів, часто всередині циклу у парі з командою STOS, так що LODS зчитує число, інші команди виконують над

ним будь-які дії, а потім STOS записує змінене число у те саме місце у пам'яті.

5. Запис у рядок

STOS приймач

Копіює регістр AL (STOSB), AX (STOSW) або Eax (STOSD) у пам'ять за адресою ES: EDI (або ES: DI, залежно від розрядності адреси). При використанні форми запису STOS асемблер сам визначає з типу зазначеного операнда (прийнято вказувати ім'я рядка, але можна використовувати будь-який операнд відповідного типу), яку з трьох форм цієї команди (STOSB, STOSW або STOSD) вибрати. Після виконання команди регістр EDI (DI) збільшується на 1, 2 або 4 (якщо копіюється байт, слово або подвійне слово), якщо прапор DF = 0, і зменшується, якщо DF = 1. При використанні з префіксом REP команда STOS заповнить рядок довжиною в ECX (або CX) числом, що знаходиться в акумуляторі.

6. Читання рядка з порту

INS джерело, DX

Зчитує з порту введення-виведення, номер якого вказаний у регістрі DX, байт (INSB), слово (INSW) або подвійне слово (INSD) в пам'ять за адресою ES: EDI (або ES: DI, в залежності від розрядності адреси). При використанні форми записи INS асемблер визначає з типу зазначеного операнда, яку з трьох форм цієї команди (INSB, INSW або INSD) вжити. Після виконання команди регістр EDI (DI) збільшується на 1, 2 або 4 (якщо зчитується байт, слово або подвійне слово), якщо прапор DF = 0, і зменшується, якщо DF = 1. При використанні з префіксом REP команда INS зчитує блок даних з порту довжиною в ECX (або CX) байт, слів або подвійних слів.

7. Запис рядка в порт

OUTS DX, приймач

Записує до порту введення-виведення, номер якого вказаний у регістрі DX, байт (OUTSB), слово (OUTSW) або подвійне слово (OUTSD) з пам'яті за адресою DS: ESI (або DS: SI, залежно від розрядності адреси). При використанні форми записи OUTS асемблер визначає з типу зазначеного операнда, яку з трьох форм цієї команди (OUTSB, OUTSW або OUTSD) вжити. Використовуючи OUTS з операндами, також можна замінити регістр DS на інший за допомогою префікса заміни сегмента (ES :, GS :, FS :, CS :, SS:). Після виконання команди регістр ESI (SI) збільшується на 1, 2 або 4 (якщо зчитується байт, слово або подвійне слово), якщо прапор DF = 0, і зменшується, якщо DF = 1. При використанні з префіксом REP команда OUTS записує блок даних розміром в ECX (або CX) байт, слів або подвійних слів до зазначеного порту. Всі процесори аж до Pentium не перевіряли готовність порту прийняти нові дані протягом виконання команди REP OUTS, таким чином, що якщо порт не встигав обробляти інформацію з тією швидкістю, з якою її поставляла ця команда, частина даних губилася.

7.2.2 Адресація

Режими адресації – це форми запису операнда у команді.

Існує два основні режими адресації: пряма та непряма.

Пряма адресація – це найпростіший вид адресації операнда в пам'яті, так як ефективна адреса міститься в самій команді і для її формування не використовуються ніякі додаткові джерела або регістри. Ефективна адреса береться безпосередньо з поля зміщення машинної команди, яке може мати розмір 8, 16, 32 біти. Це значення однозначно визначає байт, слово або подвійне слово у сегменті даних. Пряма адресація може бути двох типів.

1. Відносна пряма адресація використовується в командах умовних переходів для вказівки відносної адреси переходу. Відносність такого переходу полягає у тому, що у полі зсуву машинної команди міститься 8-, 16- або 32-розрядне значення, яке в результаті роботи команди буде складатися з вмістом регістра покажчика команд IP/EIP. У результаті такого складання виходить адреса, за якою і здійснюється перехід. Наприклад:

```
jc m1
mov al,2  ; перехід на мітку m1, якщо cf=1
……
m1
```

Хоча в команді вказана конкретна мітка, асемблер обчислює зсув цієї мітки щодо наступної команди (mov al, 2) і підставляє його до команди JC.

2. Абсолютна пряма адресація – в цьому випадку ефективна адреса є частиною машинної команди, але формується ця адреса тільки з значення поля зсуву в команді. Для формування фізичної адреси операнда в пам'яті процесор складає це поле зі зсунутим на чотири біти значенням сегментного регістра. У команді асемблера можна використовувати кілька форм такої адресації. Наприклад:

```
.data
perem1 db "Hello"
…….
mov al, perem1
```

У процесі трансляції асемблер обчислює і підставляє значення зсувів символічних імен змінних в поле зсуву формованої їм машинної команди. У підсумку виходить, що машинна команда прямо адресує свій операнд, маючи, фактично, в одному зі своїх полів значення ефективної адреси.

Решта видів адресації відносяться до непрямих. Слово «непрямий» в назві цих видів адресації означає, що в самій команді може перебувати лише частина ефективної адреси, а решта її компонентів знаходяться в регістрах, на які вказують своїм вмістом байт mod r/m і, можливо, байт sib (у багатьох командах процесора байти mrm (mod,reg,r/m) і sib слідують за байтом (байтами) коду операції. Містять наступну інформацію: тип індексації або номер регістра, використаного в команді; використаний регістр або додаткову інформацію про код операції; інформацію про базу, індекси і масштабі.).

Непряма адресація має такі різновиди:

1. Непряма базова, або регістрова, адресація

Ефективна адреса операнда може перебувати в будь-якому з регістрів загального призначення, крім SP/ESP і BP/EBP (це спеціальні регістри для роботи з сегментом стека).

```
.data
perem1 dd 1, 2, 3
lea ebx, perem1
mov eax, [ebx]
```

Наприклад, команда mov eax, [ebx] поміщає до регістра eax вміст подвійного слова за адресою сегмента даних зі зсувом, що зберігається в регістрі ebx. Так як вміст регістра легко змінити в ході роботи програми, даний спосіб адресації дозволяє динамічно призначити адресу операнда. Це корисно, наприклад, для організації циклічних обчислень і для роботи з різними структурами даних типу таблиць або масивів.

2. Непряма базова адресація зі зсувом

Доповнення попереднього виду адресації і призначена для доступу до даних з відомим зсувом щодо деякої базової адреси. Цей вид адресації зручний для доступу до елементів структур даних, коли зсув елементів відомий заздалегідь на стадії розробки програми, а базова (початкова) адреса структури повинна обчислюватися динамічно на стадії виконання програми.

Модифікація вмісту базового регістра дозволяє звертатися до однойменних елементів різних примірників однотипних структур даних. Наприклад, команда mov eax, [ebx+4] пересилає до регістру eax подвійне слово з області пам'яті за адресою, яка визначається вмістом ebx+4. Команда mov ax, mas [dx] пересилає до регістру ax слово за адресою, яка визначається вмістом dx плюс значення ідентифікатора mas (не забувайте, що транслятор присвоює кожному ідентифікатору значення, рівне зсуву цього ідентифікатора щодо початку сегмента даних).

3. Непряма індексна адресація зі зсувом

Тут також для формування ефективної адреси використовується один з регістрів загального призначення. Крім того, існує можливістю так званого масштабування вмісту індексного регістра. До структури байту sib входить поле масштабу (ss), на значення якого збільшується вміст індексного регістра. Наприклад, в команді mov eax, mas[esi*2] значення ефективної адреси другого операнда визначається виразом mas+(esi)*2. У зв'язку з тим, що в асемблері немає засобів індексації масивів, програмістові доводиться організовувати її своїми силами. Наявність можливості масштабування істотно допомагає в рішенні цієї проблеми, але за умови, що розмір елементів масиву становить 1, 2, 4 або 8 байт.

4. Непряма базова індексна адресація

Ефективна адреса формується як сума вмісту двох регістрів загального призначення: базового і індексного. У якості цих регістрів можуть застосовуватися будь-які регістри загального призначення, при цьому часто вміст індексного регістра масштабується. Наприклад:

```
.data
```

perem1 dd 1, 2, 3
lea ebx, perem1
mov esi, 2
mov eax, [esi] [edx] ; al=3

У даному прикладі ефективна адреса другого операнда формується з двох компонентів, (esi)+(edx).

5. Непряма базова індексна адресація зі зсувом.

Ефективна адреса формується як сума трьох складових: вмісту базового регістра, вмісту індексного регістра і значення поля зсуву в команді. Наприклад, команда

mov eax, [esi+2][edx]

пересилає в регістр eax подвійне слово за адресою: (esi)+2+(edx). Команда add eax, array [esi][ebx]

складає вміст регістра eax з вмістом подвійного слова за адресою, отриманою за значенням ідентифікатора array+(esi)+(ebx).

Для підтримки операцій по роботі з адресами операндов, які знаходяться в пам'яті, є спеціальна група команд – команди пересилання адрес. Формат команди:

LEA операнд1 (реєстр), операнд2

Поміщає в операнд1 зсув змінної, зазначеної в якості другого операнда. Команда LEA схожа на команду MOV тим, що вона також виконує пересилання, проте, команда LEA виконує пересилання не даних, а ефективної адреси даних (тобто зсув даних щодо початку сегмента даних) до регістру.

LDS – завантаження покажчика в регістр сегмента даних ds;

LES – завантаження покажчика в регістр сегмента даних es;

LGS – завантаження покажчика в регістр сегмента даних gs;

LFS – завантаження покажчика в регістр сегмента даних fs;

Приймач – регістр, джерело – поле пам'яті довжиною у подвійне слово. Молодше слово містить зсув об'єкта, а старше – сегментну складову. Тоді молодше слово джерела поміщається в приймач, а старше – до відповідного сегментного регістру.

7.2.3 Робота з масивами на асемблері.

Масив – структурований тип даних, який складається з певної кількості елементів одного типу.

Всі елементи масиву розташовуються в пам'яті комп'ютера послідовно. Інтерпретація структури даних в масиві залежить тільки від алгоритму обробки цих даних у конкретній програмі, а не від особливостей розташування в пам'яті.

Нумерація елементів масиву в асемблері починається з нуля.

Для отримання адреси елемента в масиві необхідно початкову (базову) адресу масиву скласти з добутком індексу (номер елемента мінус одиниця) цього елемента на розмір елемента масиву: база+(індекс*розмір елемента)

В якості базового регістра може використовуватися будь-який з восьми регістрів загального призначення. У якості індексного регістру також можна використовувати будь-який регістр загального призначення, за винятком ESP/SP.

Іноді зручно використовувати ECX у якості номеру елемента масиву в циклі, організованому за допомогою команди loop.

7.3 Контрольні завдання.

1 рівень складності:

1. Вивести на екран заданий масив чисел (1,2,5,7,8,9) у зворотному порядку.
2. Вивести на екран заданий масив (1,2,5,7,8,9) чисел з середини до кінця масиву.
3. Вивести на екран заданий масив (1,2,5,7,8,9) чисел з початку до середини масиву.
4. Вивести на екран 1-й та останній елементи заданого масиву (1,2,5,7,8,9) чисел.
5. Вивести на екран 2-й та передостанній елементи заданого масиву (1,2,5,7,8,9) чисел.
6. Вивести на екран відсортований за зростанням заданий масив (8,2,3,7,8,9) чисел.
7. Вивести на екран відсортований за зменшенням заданий масив (8,2,3,7,8,9) чисел.
8. Вивести на екран заданий масив (8,v,q,2,c,7,a,9) елементів відсортувавши його наступним чином: числа за зростанням з початку масиву, а літери за зростанням – у кінці.
9. Вивести на екран заданий масив (8,v,q,2,c,7,a,9) елементів відсортувавши його наступним чином: числа за зменшенням з початку масиву, а літери за зменшенням – у кінці.
10. Вивести на екран заданий масив (8,v,q,2,c,7,a,9) елементів відсортувавши його наступним чином: числа за зростанням з початку масиву, а літери за зменшенням – у кінці.
11. Вивести на екран заданий масив (8,v,q,2,c,7,a,9) елементів відсортувавши його наступним чином: числа за зменшенням з початку масиву, а літери за зростанням – у кінці.
12. Вивести на екран заданий масив (8,v,q,2,c,7,a,9) елементів відсортувавши його наступним чином: числа за зростанням у кінці масиву, а літери за зростанням – з початку.
13. Вивести на екран заданий масив (8,v,q,2,c,7,a,9) елементів відсортувавши його наступним чином: числа за зменшенням у кінці масиву, а літери за зменшенням – з початку.
14. Вивести на екран заданий масив (8,v,q,2,c,7,a,9) елементів відсортувавши його наступним чином: числа за зростанням у кінці масиву, а літери за зменшенням – з початку.

15. Вивести на екран заданий масив (8,v,q,2,c,7,a,9) елементів відсортувавши його наступним чином: числа за зменшенням у кінці масиву, а літери за зростанням – з початку.

16. Вивести на екран заданий масив (v,q,t,c,s,a,i) елементів відсортувавши його за зростанням.

17. Вивести на екран заданий масив (v,q,t,c,s,a,i) елементів відсортувавши його за зменшенням.

18. Вивести на екран заданий масив (v,q,t,c,s,a,i) елементів з початку до середини масиву попередньо відсортувавши його за зменшенням.

19. Вивести на екран заданий масив (v,q,t,c,s,a,i) елементів з початку до середини масиву попередньо відсортувавши його за зростанням.

20. Вивести на екран заданий масив (v,q,t,c,s,a,i) елементів з середини до кінця масиву попередньо відсортувавши його за зменшенням.

21. Вивести на екран заданий масив (v,q,t,c,s,a,i) елементів з середини до кінця масиву попередньо відсортувавши його за зростанням.

22. Вивести на екран 3-й та останній елемент заданого масиву (v,q,t,c,s,a,i).

23. Вивести на екран 2-й та передостанній елемент заданого масиву (v,q,t,c,s,a,i).

24. Вивести на екран Зостанні елементи заданого масиву (v,q,t,c,s,a,i).

2 рівень складності:

1. У заданому рядку з нульовим завершителем визначити позицію заданого підрядка.

2. У заданому рядку з нульовим завершителем визначити символи, які входять до іншого рядка.

3. У заданому рядку з нульовим завершителем визначити символи, які не входять до іншого рядка.

4. У заданому рядку з нульовим завершителем визначити номери цифр.

5. У заданому рядку з нульовим завершителем прибрати пробіли, якщо їх більше одного поспіль. Визначити кількість символів у новому рядку.

6. У заданому рядку з нульовим завершителем поміняти символи кожної пари місцями.

7. У заданому рядку з нульовим завершителем переписати символи рядка у зворотному порядку.

8. У заданої рядку з нульовим завершителем виділити окремі слова, якщо в якості роздільників використовуються роздільники української мови.

9. До заданого рядка з нульовим завершителем вставити заданий підрядок, починаючи з вказаної позиції.

10. У заданому рядку з нульовим завершителем видалити вказаний підрядок, скільки б разів він не зустрічався у рядку.

11. Задані 2 рядки, у яких символи впорядковані за порядком зростання кодів. Об'єднати їх у один рядок, зберігши впорядкованість.

12. Задані 2 рядки, в яких символи впорядковані за порядком зменшення кодів. Об'єднати їх у один рядок, зберігши впорядкованість.

13. Для заданого масиву чисел підрахувати кількість позитивних чисел і занести їх у зворотному порядку до іншого масиву.

14. Для заданого масиву чисел визначити номери позитивних чисел.

15. У заданому масиві чисел визначити номер елемента, з якого починається інший масив.

16. У заданому масиві чисел визначити номери чисел, які входять до іншого масиву.

17. У заданому масиві чисел визначити номери чисел, які не входять до іншого масиву.

18. У заданому масиві чисел визначити номери парних чисел.

19. У заданому масиві чисел прибрати числа, які входять до заданого діапазону [a, b] (Прибрати числа, які не менше, але і не більше b). Визначити кількість чисел у новому масиві.

20. У заданому масиві чисел поміняти числа кожної пари місцями.

21. У заданому масиві чисел визначити номери непарних чисел.

22. У заданий масив чисел вставити інший масив, починаючи з обраної позиції.

23. У заданому масиві чисел видалити задану послідовність чисел, скільки б раз вона не зустрічалася у масиві. Наприклад, для масиву 01234512345124 видалити послідовність 123. У результаті повинні отримати масив: 04545124.

24. Задані 2 упорядкованих за зростанням масивів чисел. Об'єднати їх у один масив, зберігши впорядкованість.