Ми переходимо до наступної теми – Нелінійні розрахунки та алгоритмі, їх реалізація на мові Assembler

План лекції
Безумовні переходи
Умовні переходи
Команди умовних переходів
Приклад

Приклади команд, які ми розглядали з Вами на попередніх лекціях були лінійними. Тобто команди в них виконувались одна за одною.

Як Ви, маю надію, розумієте – це дуже примітивні програми, в яких, до того ж, не враховані певні «нештатні» ситуації, такі як «переповнення» «ділення на 0» і так далі.

Відповідно сьогодні ми спробуємо розглянути процес створення програм, в яких є певні нелінійні розрахунки, коли потрібно організувати певні переходи між різними блоками програми.

Це може бути варіант, коли нам потрібно перевірити певну умову і отримати розгалуження у роботі програми (аналог оператора IF), або потрібно буде повторити декілька разів тей чи інший набір операцій (аналог циклів).

Відповідно, у загальному випадку, в програмі є місця, у яких потрібно прийняти рішення про те, яка команда буде виконуватись наступною. Це рішення може бути:

– безумовним: із визначеного місця необхідно передати управління не тій команді, яка йде наступною, а іншій, яка розміщується на деякому віддаленні від поточної команди;

– умовним: рішення про те, яка команда буде виконуватись наступною, приймається за результатами аналізу деяких умов або даних.

Слід зазначити, що безумовні переходи в мовах високого рівня зараз не використовуються. І хоча в певних мовах накшталт С або С++ є певні засоби, типу оператора GOTO, але використання таких засобів не рекомендується.

Отже, за принципом дії команди мікропроцесора, які забезпечують організацію переходів у програмі, можна поділити на три основні групи:

1. Команди безумовної передачі управління.
2. Команди умовної передачі управління.
3. Команди управління циклом.

Розглянемо Команди безумовного переходу:

– команди безумовного переходу; (тобто це чіткі вказівки, що виконання програми повинно переміститись з одного рядка до іншого).

– виклик процедур та повернення з процедур; (що таке процедура? Це функція або підпрограма, що дозволяє об’єднувати декілька команд або інструкцій для повторного використання та, відповідно, організації більш структурованого коду. Після виконання процедури програма повертається для продовження виконання наступної команди в коді)

– виклик програмних переривань та повернення з програмних переривань. (Що таке переривання? Це якась подія в процесі виконання програми, яка вимагає певного втручання та рішення – ігнорувати цю проблему і продовжити виконання програми, чи повернути певне повідомлення та спробувати виконати ще раз. Наприклад, коли не вдається записати файл, або підєднується або від’єднується якийсь пристрій, або виникає ділення на нуль, або інші критичні ситуації, які без обробки викликають руйнування ходу виконання програми). Цю тему ми поки що детально розглядати не будемо.

Далі ми розглянемо більш детально команди **безумовного переходу**.

Як вже зазначалось при безумовному переході здійснюється передача управління не наступному рядку, наступній за порядком команді, а якійсь іншій команді. Для організації таких переходів використовуються мітки.

Мітки. Ще раз: місце, у яке необхідно здійснити перехід, визначається за допомогою міток. *Мітка* – це символічне ім'я, яке позначає визначену комірку пам'яті, що призначена для використання як операнда в командах передачі управління. Тобто до якої потрібно перейти для продовження виконання коду програми.

Як і для змінної транслятор *Assembler* привласнює мітці три основні атрибути:

– ім'я сегмента коду, де ця мітка описана;

– зміщення – відстань у байтах від початку сегмента коду, у якому описана мітка;

– тип мітки або атрибут відстані

Атрибут відстані може набувати двох значень:

– near – перехід на цю мітку можливий лише в межах сегмента коду, де ця мітка описана. Фізично це означає, що для переходу на мітку достатньо змінити тільки вміст регістра eip/ip;

– far – перехід на цю мітку можливий тільки внаслідок міжсегментної передачі управління, для здійснення якої потрібна зміна вмісту як регістра eip/ip, так і регістра cs.).

Мітку можна визначити двома способами:

– оператором **:** (двокрапка); (найпростіший і найпоширеніший спосіб і ми будемо з ним працювати)

– директивою label (аналіз літератури показав, що зазвичай цей метод не використовується, окрім деяких асемблерних середовищ, де можна оголошувати мітку саме директивою label.

Отже, синтаксис першого способу показано на рис. 1.

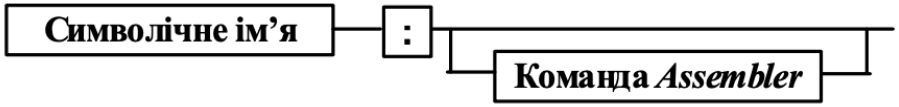


Рис. 1. Синтаксис визначення мітки

За допомогою цього способу можна визначити тільки мітку ближнього типу – near. Команда *Assembler* може розташовуватись як в одному рядку з міткою, так і на наступному рядку. Таку мітку можна використовувати як операнд в командах умовного (jcc) та безумовного (jmp, call) переходів, наприклад:

```
__asm {
m1:     mov ax, bx
        inc bx
        …
        jmp m3
m2:
        mov ax, c
        imul ax
        …
        jmp ext
m3:     imul bx
        jmp m2
ext:
}
```

Можна наочно побачити змінену послідовність виконання команди m1 -> m3 -> m2 -> ext

Якщо в мові С використання оператора безумовних переходів GOTO вважається негативною практикою, то в ассемблері інших варіантів по суті немає.
Більш того, після компіляції програм на мовах високого рівня в машинний код, там будуть використовуватись вже аналогічні безумовні переходи.

Другий спосіб – використання директиви LABEL. Як я вже зазначав, ми його використовувати не будемо, оскільки зазвичай він використовується тільки в певних ассемблерних середовищах для оголошення міток.

Далі розглянемо другий пункт

2. **Команди умовної передачі управління**: (ці команди дають можливість перевірити виконання або не виконання певної умови або даних що сформувались перед виконанням переходу)

– команди переходу за результатом виконання команди порівняння (CMP); (виконання переходу в результаті порівняння числових значень або інших умов). Зазвичай ця команда заміняє команди порівняння більше, менше , відповідності у мовах вищого рівня.

– команди переходу за станом визначеного прапорця FLAGS/EFLAGS; (зазвичай використовуються для усіляких нестандартних операцій, наприклад переповнення, втрати знака або ділення на нуль)

– команди переходу за вмістом регістра ecx/cx. (це, зазвичай ситуація, коли ми працюємо з циклами)

Типова схема умовного переходу (команда if в мовах високого рівня):

- порівняння за допомогою команди CMP
- перехід на мітку за результатами порівняння за допомогою команди Jc/Jcc/Jccc (c/cc/ccc – різні символи для різних порівнюваних значень).

Команда порівняння (cmp). Принцип роботи команди **cmp** (*compare*) такий самий, як і команди віднімання sub *операнд_1*, *операнд_2.* Команда cmp здійснює віднімання операндів і встановлює відповідні прапорці. Однак на відміну від команди sub, команда cmp не записує результат віднімання на місце першого операнда. а генеруються у вигляді відповідних ознак регістра FLAGS/EFLAGS

Операнд 1 – регістр, змінна
Операнд 2 – регістр, змінна, константа (число)

Синтаксис команди:

cmp *операнд_1, операнд_2*
Прапорці, які встановлюються командою cmp, можна аналізувати спеціальними командами переходу. Команди умовного переходу jcc.

Команди умовних переходів **Jccc**
Літери команд «**ccc**» використовуються з наступної мнемоніки:

Позначення	Оригінал	Переклад	Тип операндів
e	**E**qual	дорівнює	операнди будь які
n	**N**ot	не	операнди будь які
g	**G**reater	Більше	Числа зі знаком
l	**L**ess	Менше	Числа зі знаком
a	**A**bove	Вище (більше)	Числа без знаку
b	**B**elow	Нижче (менше)	Числа без знаку

	Знакові	**Беззнакові**
!=	**>=**	**>=**
Не дорівнює	більше або дорівнює	більше або дорівнює
JNE	JGE (JNL)	JAE (JNB)
Jump if **N**ot **E**qual	**J**ump if **G**reater or **E**qual **J**ump if **N**ot **L**ess	**J**ump if **A**bove or **E**qual **J**ump if **N**ot **B**elow

Таблиця **Перелік команд умовного переходу для команди cmp** *оператор_1, оператор_2*

Тип операндів	Мнемокод команди	Критерій умовного переходу	Значення прапорців для здійснення переходу
Будь-які	je	операнд_1=операнд_2	zf = 1
Будь-які	je	операнд_1< >операнд_2	zf = 0
Зі знаком	jl / jnge	операнд_1<операнд_2	sf< > of
Зі знаком	jle / jng	операнд_1<=операнд_2	sf< > of or zf = 0
Зі знаком	jg / jnle	операнд_1>операнд_2	sf = of and zf = 0
Зі знаком	jge / jnl	операнд_1=>операнд_2	sf = of
Без знака	jb / jnae	операнд_1<операнд_2	cf = 1
Без знака	jbe / jna	операнд_1<=операнд_2	cf = 1 or zf = 1
Без знака	ja / jnbe	операнд_1>операнд_2	cf = 0 and zf = 0
Без знака	jae / jnb	операнд_1=>операнд_2	cf = 0

Команди умовного переходу не змінюють прапорці, тому після команди **cmp** може бути декілька команд умовного переходу. Це може бути використано для того, щоб дослідити кожну з альтернатив: “більше”, “менше” або “дорівнює”.

```
.....
cmp ax, 5 ;(порівняти вміст ax з 5)
je eql ;(перехід, якщо ax = 5)
jl low ;(перехід, якщо ax<5)
jg grt ;(перехід, якщо ax>5)
egl: .....
low: .....
grt: .....
```

Команди умовного переходу та регістр ecx/cx. Архітектура мікропроцесора передбачає спеціальне використання деяких регістрів. Регістр ecx/cx виконує функцію лічильника в командах управління циклами й при роботі з ланцюгами символів. Можливо, що функціонально, команду умовного переходу, пов’язану з регістром ecx/cx, правильніше було б віднести до цієї групи команд. Синтаксис цієї команди умовного переходу такий:

jcxz *мітка_переходу* (Jump if CX is Zero) – *перехід, якщо* cx *нуль*; jecxz *мітка_переходу* (Jump if ECX is Zero) – *перехід, якщо* ecx *нуль*.

Ці команди зручно використовувати при організації циклів і при роботі з ланцюгами символів.

Слід зазначити, що команди умовної передачі управління jcxz/jecxz можуть адресувати тільки короткі переходи – на –128 *байт* або на +127 *байт* від наступної за нею командою.

Приклад. Розрахувати значення 16-бітного знакового виразу

$$y = \begin{cases} \mathbf{1}, \mathbf{a} > \mathbf{b} \\ \mathbf{0}, \mathbf{a} = \mathbf{b} \\ \mathbf{1}, \mathbf{a} < \mathbf{b} \end{cases}$$

__asm {

; варіант 1
 	
MOV AX, a;
MOV BX, b;
CMP AX,BX ;
JG m1; Перехід, якщо a>b
JE m2 ; Перехід, якщо a=b
JL m3 ; Перехід, якщо a<b
m1: MOV y,1
JMP end
m2: MOV y, 0
JMP end
m3: MOV y, -1
JMP end
end:
	; варіант 1 виділене червоним можна відкинути
	
	MOV AX, a;
	MOV BX, b;
	CMP AX,BX ;
	JG m1; Перехід, якщо a>b
	JE m2 ; Перехід, якщо a=b
	JL m3 ; Перехід, якщо a<b
	m1: MOV y,1
	JMP end
	m2: MOV y, 0
	JMP end
	m3: MOV y, -1
	JMP end
	end:

}

Розглянемо приклад

$$y = \begin{cases} (a + b - 5) * c, a > b \\ 0, a = b \\ \dfrac{a + b - 5}{c}, a < b \end{cases}$$

```
short int a, b, c, y, err;
scanf_s("%hd", &a);
scanf_s("%hd", &b);
scanf_s("%hd", &c);
```

```
__asm {
        CMP AX, BX;
        JG M1;
        JL M2;
        MOV y, 0;
        JMP EXT;
M1:     MOV AX, a;
        ADD AX, b;
        SUB AX, 5;
        MOV BX, C;
        IMUL BX;
        MOV BX, 1;
        IDIV BX;
        MOV y, AX;
        JMP EXT;
M2:     MOV AX, a;
        ADD AX, b;
        SUB AX, 5;
        CWD;
        CMP c, 0;
        JE ERR1;
        MOV BX, c;
        IDIV BX;
        MOV y, AX
        JMP EXT;
ERR1:   MOV err, 1;
EXT:
}
```

```
If( err == 1) {
        printf("помилка, ділення на 0 \n")
        } else{
        printf("y=%hd\n", y)
        }
```

Нестандартні ситуації при виконанні арифметичних операцій та їх опрацювання в Ассемблер.

1. Регістр ознак
2. Нестандартні ситуації
3. Команди умовних переходів для нестандартних ситуацій
4. Приклад.

Нестандартні ситуації

Для цілочисельних арифметичних виразів

1. Переповнення
2. Ділення на нуль
3. Вихід за межі діапазону

Переповнення

a, b, y – 8 бітні беззнакові (0 … 255)
y = a+b+1
a = 150 – в діапазон потрапляє 0 … 255
b = 180 – в діапазон потрапляє 0 … 255
y = 150 + 180 !!! y = 330 переповнення

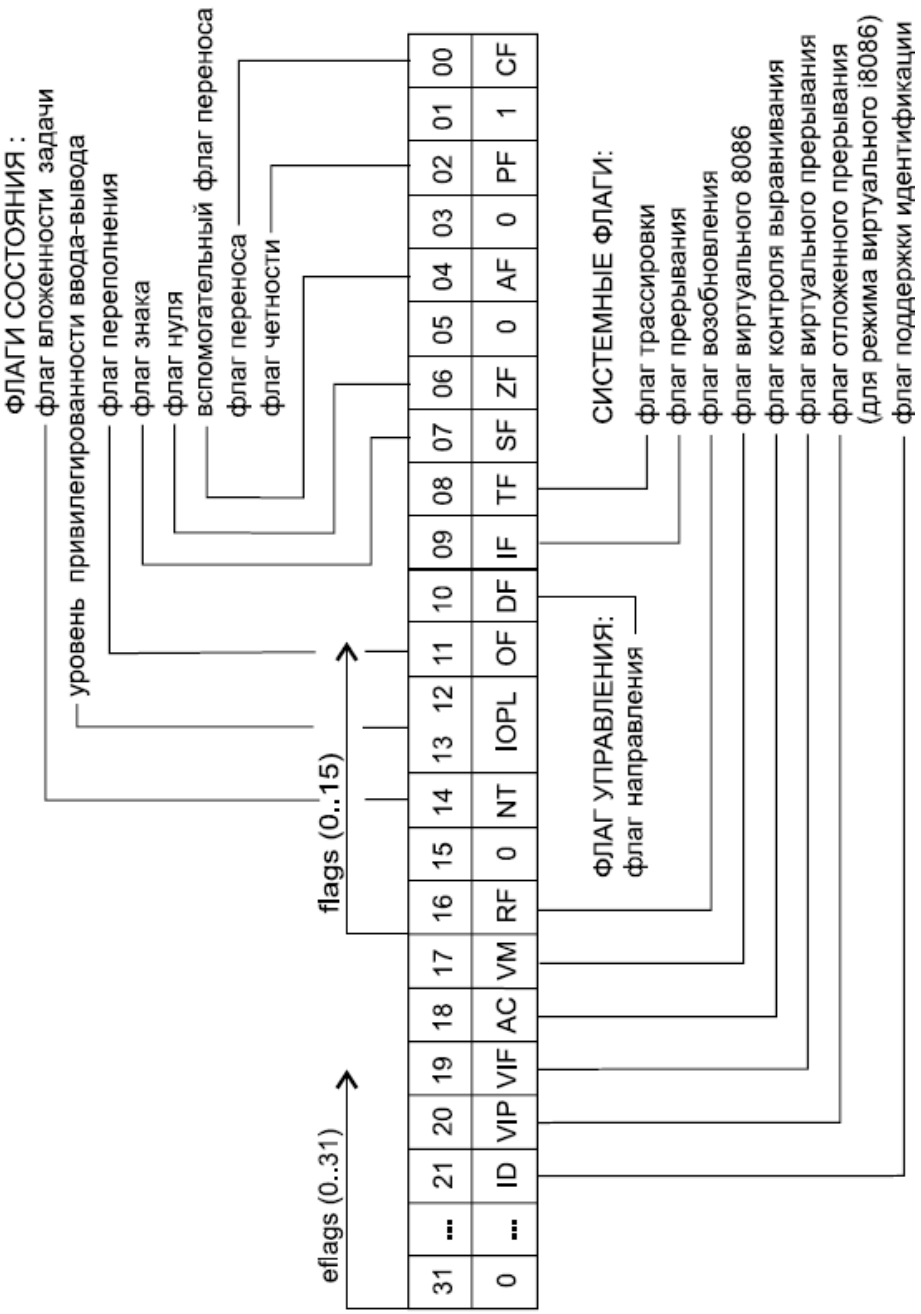
Ділення на нуль

a, b, y – 8 бітні беззнакові (0 … 255)
y = a/(b-1)
a = 50 – в діапазон потрапляє 0 … 255
b = 1 – в діапазон потрапляє 0 … 255
y = 50/(1-1) !!! Ділення на нуль

Вихід за межі діапазону

a, b, y – 8 бітні беззнакові (0 … 255)
y = a – b – 1
a = 5 – в діапазон потрапляє 0 … 255
b = 6 – в діапазон потрапляє 0 … 255
y = 5 – 6 !!! вихід за межі діапазону

Регістр ознак FLAGS/EFLAGS



Мнемоніка	Назва прапорця	Номер біта	Вміст та призначення
CF	Прапорець переносу (*Carry Flag*)	0	Встановлюється в 1, якщо арифметична операція зробила перенесення зі старшого біта результату. Старшим є 7, 15 або 31-й біт залежно від розмірності операнда; 0 – перенесення не було
PF	Прапорець паритету (*Parity Flag*)	2	Встановлюється в 1, якщо вісім молодших розрядів результату містять парну кількість одиниць (цей прапорець тільки для 8 молодших розрядів операндів будь-якого розміру). 0 – вісім молодших розрядів результату містять непарну кількість одиниць
AF	Допоміжний прапорець переносу (*Auxiliary Flag*)	4	Тільки для команд, працюючих з BCD-числами. Фіксує факт позики з молодшої тетради результату: 1 – в результаті операції додавання було зроблено перенесення з розряду 3 в старший розряд або при відніманні була позика в розряд 3 молодших тетради зі значення в старшій тетраді; 0 – перенесень і позик в(з) 3 розряд(а) молодшої тетради результату не було
ZF	Прапорець нуля (*Zero Flag*)	6	Встановлюється в 1, якщо результат операції дорівнює 0, і встановлюється в 0, якщо результат не нульовий
SF	Прапорець знака (*Sign Flag*)	7	Відбиває стан старшого біта результату (біти 7, 15 або 31), тобто знак результату операції, при від'ємному результаті SF = 1

Закінчення табл. 2.1

Мнемоніка	Назва прапорця	Номер біта	Вміст та призначення
OF	Прапорець переповнення (*Overflow Flag*)	11	Фіксує факт втрати значущого біта при арифметичних операціях, тобто ситуацію переповнення (вихід результату арифметичної операції за межі припустимого діапазону значень), OF = 1, якщо внаслідок операції виникає перенос у старший знаковий біт результату або позика із старшого знакового біта результату, OF=0, якщо внаслідок операції не виникло переносу у старший знаковий біт результату або позиці зі старшого знакового біта результату
IOPL	Рівень привілейованості вводу-виводу (*Input/ Output privilege level*)	12, 13	Використовується в захищеному режимі роботи процесора для контролю доступу до команд вводу-виводу залежно від привілейованості команд

Команди умовних переходів для нестандартних ситуацій

Код команди	Стан прапорців	Код команди	Стан прапорців
JZ/JE	**ZF=1**	**JNZ/JNE**	**ZF=0**
JS	**SF=1**	**JNS**	**SF=0**
JC/JB/JNAE	**CF=1**	**JNC/JAE/JNB**	**CF=0**
JO	**OF=1**	**JNO**	**OF=0**
JP	**PF=1**	**JNP**	**PF=0**

Команди реагують на значення певних ознак (Flags).
Виконання команди порівняння перед цими командами не обо'язкове, достатньо будь якої команди, що змінює ознаку.