Лабораторна робота № 6

Тема: Програми із використанням арифметичних та логічних команд.

Мета: Навчитися використовувати арифметичні та логічні команди у програмах на асемблері.

Короткі теоретичні відомості

Арифметичні та логічні команди процесора широко використовуються при різноманітних перетвореннях даних.

1. Арифметичні команди

До арифметичних команд відносять команди додавання, віднімання, множення, ділення, інкременту та декременту, порівняння, зміни знаку та ін.

Розглянемо для прикладу деякі арифметичні команди.

ADD dest, src ; dest = dest + src

Команда ADD виконує додавання двох операндів (одержувача і джерела) і заносить результат на місце першого операнда (одержувача) та встановлює прапорці. Роль одержувача і джерела можуть виконувати регістри процесора та комірки пам'яті. Розмір операндів повинен бути однаковим. Комірки пам'яті не можуть виконувати роль приймача і джерела одночасно. Джерелом можуть бути безпосередні дані.

SUB dest, src ; dest = dest – src

Команда SUB виконує віднімання джерела від одержувача і заносить результат на місце одержувача та встановлює прапорці. Зауваження щодо операндів аналогічні команді додавання.

Команди ADC, SBB аналогічні попереднім, але в операції додавання (віднімання) додатково приймає участь прапорець переносу CF.

MUL src

Команда MUL виконує множення вмісту акумулятора (першого множника) на беззнаковий операнд (другий множник). В залежності від розміру операнда використовується відповідна частина акумулятора (AL для 8-бітного операнда, AX для 16-бітного операнда). Операндом може бути регістр або комірка пам'яті, розмір якої визначений відповідними директивами асемблера. Розмірність результату у два рази більша за розмірність множників. Результат розміщується в регістр AX або пару регістрів DX_AX. DX_AX позначає пару регістрів (DX – старша частина, AX – молодша).

AX = AL * src8 для 8-бітних операндів

DX_AX = AX * src16 для 16-бітних операндів

Варіант IMUL команди призначений для множення знакових операндів.

DIV src

Команда DIV виконує беззнакове ділення вмісту акумулятора (діленого) на операнд (дільник). В залежності від розміру операнда використовується відповідний акумулятор (AX для 8-бітного операнда, DX_AX для 16-бітного операнда). Операндом може бути регістр або комірка пам'яті, розмір якої визначений відповідними директивами асемблера. Розмірність результату у два рази менша за розмірність діленого. Частка надходить в регістр AL (AX), а остача в регістр AH (DX). При наявності остачі частка дає приблизний результат. Варіант IDIV команди призначений для ділення знакових операндів. Варіанти операндів і результатів команд цілочисельного ділення наведено в таблиці.

Розмір діленого/дільника	Ділене (Dividend)	Дільник (Divisor)	Частка (Quotient)	Остача (Remainder)
Word/byte	AX	r/m8	AL	AH
Doubleword/word	DX:AX	r/m16	AX	DX
Quadword/ doubleword	EDX:EAX	r/m32	EAX	EDX

При виконанні простих цілочисельних операцій на асемблері по можливості враховують наступні рекомендації. Операції множення замінюють на операції додавання або зсуву. Наприклад:

2*A = A+A або зсув A вліво на один розряд (у двійковій системі множення на 2)

3*A = 2*A + A

4*A = 2*A + 2*A або зсув вліво на два розряди (у двійковій системі множення на 4)

5*A = 4*A + A

Такі обчислення виконуються швидше, ніж операція множення MUL (IMUL).

Ділення на 2, 4, 8 і т. д. здійснюється шляхом зсуву вправо на необхідне число розрядів.

Розглянемо приклад створення програми для виконання обчислень із використанням арифметичних операцій.

Приклад. Написати програму для обчислення значення виразу:

X = 3A + (B + 2C) / 2 – 5,

де A, B, C, X – цілі 16-розрядні числа із знаком.

Обчислення проводять із дотриманням порядку виконання арифметичних операцій, звертаючи увагу на наявність дужок.

Розглянемо приклад алгоритму обчислення виразу:

Дія	Коментар	Команда
AX ← A	значення A в регістр AX	`mov ax, a`
AX ← (AX)*2	2A в AX	`sal ax, 1`
AX ← (AX)+A	3A в AX	`add ax, a`
BX ← C	C в BX	`mov bx, c`
BX ← 2*(BX)	2C в BX	`sal bx, 1`
BX ← (BX)+C	(B+2C) в BX	`add bx, b`
BX ← (BX) / 2	(B+2C) / 2 в BX	`sar bx, 1`
AX ← (AX)+(BX)	3A+(B+2C) / 2 в AX	`add ax, bx`
AX ← (AX) – C	3A+(B+5) / 2 – 5 в AX	`sub ax, 5`
X ← (AX)	3A+(B+5)/2 – 5 в X	`mov x, ax`

Приклад тексту програми:

```
include asmio16.inc
.data
     a     dw    10
     b     dw    20
     c     dw    5
     x     dw    ?
.code
.begin
     mov ax, a              ; ax = a
     sal ax, 1              ; ax = ax * 2
     add ax, a              ; ax = ax + a
     mov bx, c              ; bx = c
     sal bx, 1              ; bx = bx *2
     add bx, b              ; bx = bx + b
     sar bx, 1              ; bx = bx / 2
     add ax, bx             ; ax = ax + bx
     sub ax, 5              ; ax = ax - 5
     mov x, ax
     call WriteInt
.end
```

2. Логічні команди

Логічні команди призначені для бітових маніпуляцій з даними шляхом виконання логічних операцій. При програмуванні мовою асемблера використовують логічні операції І (AND), АБО (OR), виключне АБО (Exlusive OR або XOR), НЕ (NOT). Слід пам'ятати, що коли логічні операції застосовуються до багаторозрядних даних, то вони виконуються незалежно над окремими бітами операндів.

Нагадаємо, що логічні операції означають наступне:

X	Y	X AND Y	X OR Y	X XOR Y	NOT X
0	0	0	0	0	1
0	1	0	1	1	
1	0	0	1	1	0
1	1	1	1	0	

Серед команд процесора є команда TEST, призначена для виконання порозрядної команди AND над операндами. Але ця команда, на відміну від AND лише встановлює прапорці, а результат операції нікуди не заносить. Цю команду зазвичай використовують для перевірки значень окремих бітів.

Логічні операції у мові асемблера використовують для керування окремими бітами даних. При цьому перший операнд називають даними, а другий маскою. Маска показує з якими бітами будуть виконуватися дії. Маска може мати активні "одиниці" або "нулі". Якщо необхідно деякі біти даних перевести в нуль, то на відповідних місцях маски ставлять нулі і використовують команду AND. Якщо необхідно деякі біти даних перевести в одиницю, то в масці використовують одиниці і застосовують команду OR. Команда XOR використовується для інверсії бітів (маска – одиниці) або для знищення (обнулення) бітів (маскою є самі дані, наприклад XOR AX, AX).

Приклади логічних операцій над даними:

	AND		OR		XOR			
					Інверсія бітів		Очистка	
X (дані)	00110111	37h	00000111	07h	00110111	37h	00110111	37h
Y (маска)	**0000**1111	0Fh	00**11**0000	30h	0000**1111**	0Fh	**00110111**	37h
Результат	00000111	07h	00110111	37h	00111000	38h	00000000	00h

Розглянемо для прикладу алгоритм програми, яка виводить на екран байт даних у шістнадцятковому вигляді (дві цифри). Для побудови програми слід детально продумати алгоритм перетворення. Порівняємо цифри шістнадцяткової системи числення із їх кодами ASCII. Коди ASCII необхідні для виводу символа на екран.

Цифра	Код ASCII	Цифра	Код ASCII
0	30h	A	41h
1	31h	B	42h
2	32h	C	43h
3	33h	D	44h
4	34h	E	45h
5	35h	F	46h
6	36h		
7	37h		
8	38h		
9	39h		

З таблиці видно, що для цифр 0…9 код цифри більший від самої цифри на 30h, а для цифр A…F код цифри більший на 37h.

Один із алгоритмів виведення чисел передбачає перевірку кожної шістнадцяткової цифри на належність до однієї із множин {0…9} або {A…F} і формування коду ASCII шляхом додавання 30h або 37h.

Інший алгоритм реалізується за допомогою табличного перетворення. Для цього в пам'яті створюється таблиця ASCII-кодів шістнадцяткових цифр. Наприклад:

```
HexTab db '0123456789ABCDEF'    ; 30h, 31h, 32h,.. 39h,.. 41h,.. 46h
```

У цій таблиці код кожної цифри n знаходиться за адресою HexTab+n. Дані з таблиці дістаються командою MOV після обчислення необхідної адреси, або використовується спеціальна команда для роботи з таблицями XLAT.

Алгоритм перетворення може бути приблизно такий.

1. Дістати з пам'яті байт даних.
2. Виділити з нього старшу тетраду (зсув вправо на 4 розряди).
3. Знайти за таблицею ASCII-код символа (початок таблиці + цифра).
4. Вивести на екран.
5. Виділити з байта молодшу тетраду (маска 0Fh).
6. Знайти за таблицею ASCII-код символа.
7. Вивести на екран.

Порядок виконання роботи

1. Написати програму для обчислення значення арифметичного виразу X= f (A, B, C).

2. Розробити алгоритм і написати програму перетворення 2-байтового цілого числа в рядок символів, який представляє запис числа в шістнадцятковій системі (4 знаки).

3. Написати програму переведення 2-байтового числа із знаком у 5-розрядне десяткове число, представлене як рядок символів (6 байтів).

Завдання для пунктів 1–3 обрати згідно варіанту за таблицею.

Номер варіанту	Вираз	Число
1	X = A – 4 (B + 2C) + 7	1BD8h
2	X = (A + 2B) / 4 – C + 27	C81Fh
3	X = 3A · (B – 2C) – 17	2E8Bh
4	X = A + (B + 7) / 2 – 3C	B37Dh
5	X = 4 (A + 5) – (B + 5C)	A2B7h
6	X = (6A + B) / (4C + 12)	3EB7h
7	X = 4 (A +2B) / 3 – 5C +1	26B7h
8	X = 5 (A– 2B) + C / 4 + 12	E89Ah
9	X = 2A – B (A + B) / C	8BC1h
10	X = 3B + 5 – (A + 4C) / 2	B90Eh
11	X = (2A – B) / 5 + C / 2 + 16	DA33h
12	X = (5A + B + C / 4) + 10	8BA3h
13	X = 6C + (B – C + 1) / 2	F05Ch
14	X = 6 (A – 2B + C / 4) + 10	DF35h
15	X = 3 (A – 2B) + 30 – C / 2	07CFh

5. Скомпілювати програму у форматі EXE-файлу.

6. Простежити хід виконання програми за допомогою відлагоджувальника Turbo Debugger (td.exe). При необхідності відлагодити програму.

7. Запустити програму на виконання.

Контрольні питання

1. Назвіть відомі вам арифметичні та логічні команди.
2. Поясніть різницю між командами ADD і ADC, SUB і SBB?
3. Поясніть різницю між командами MUL та IMUL, DIV та IDIV?
4. Як реалізувати на асемблері операцію Z = Y MOD X?
5. Як поміняти місцями старшу і молодшу тетради у байті?
6. Якою командою можна округлити число до найближчого парного?