РОЗРАХУНКОВА РОБОТА
АРИФМЕТИЧНІ ОПЕРАЦІЇ. БІТОВІ КОМАНДИ

Мета роботи: вивчити арифметичні операції мовою Асемблер та їх використання при складанні програм; вивчити бітові команди і їх застосування.

Теоретичні відомості

Команди двійкової арифметики

Усі команди з цього розділу, крім команд ділення і множення, змінюють прапорці OF, SF, ZF, AF, CF, PF відповідно до призначення кожного з них.

Команда `ADD` (від англ. add – складати, додавати; adding – додавання) виконує арифметичне додавання приймача й джерела, поміщає суму в приймач, не змінюючи вміст джерела (табл. 1). *Приймач* може бути регістром або змінною, *джерело* може бути числом, регістром або змінною, але не можна використовувати змінну одночасно і для джерела, і для приймача. Команда `ADD` ніяк не розрізняє числа зі знаком і без знака, але, вживаючи значення прапорців `CF` (перенесення при додаванні чисел без знака), `OF` (перенесення при додаванні чисел зі знаком) і `SF` (знак результату), можна використовувати її для всіх чисел.

Таблиця 1

Основні характеристики команди `ADD`

Команда	Призначення	Процесор
`ADD приймач, джерело`	Додавання	8086

Синтаксис команда `ADD` такий:

```
приймач: = приймач + джерело
```

Команда `ADC` (від англ. add with carry – скласти з перенесенням) в усьому є аналогічною `ADD`, крім того, що вона виконує арифметичне додавання приймача, джерела й прапорця `CF` (табл. 2).

Таблиця 2

Основні характеристики команди `ADC`

Команда	Призначення	Процесор
`ADC приймач, джерело`	Додавання з перенесенням	8086

Пара команд `ADD` і `ADC` використовується для складання чисел підвищеної точності. Синтаксис команди `ADC` є таким:

```
приймач: = приймач + джерело + CF
```

Приклад. Складемо два 64-бітних цілих числа: нехай одне з них знаходиться в парі регістрів `EDX:EAX` (молодше подвійне слово (біти 0–31) – в `EAX` і старше (біти 32–63) – в `EDX`), а друге – в парі регістрів `EBX:ECX`:

```
add eax,ecx
adc edx,ebx
```

Якщо при додаванні молодших подвійних слів сталося перенесення зі старшого розряду (прапорець `CF = 1`), то його буде враховано з допомогою команди `ADC`.

Команда `XADD` (від англ. exchange and add – обміняти й скласти) виконує складання, поміщає вміст приймача в джерело, суму операндів – у приймач (табл. 3). Джерело – завжди регістр, приймач може бути регістром і змінною.

Таблиця 3

Основні характеристики команди `XADD`

Команда	Призначення	Процесор
`XADD приймач, джерело`	Обміняти між собою і скласти	80486

Команда `SUB` (від англ. subtract – віднімати; subtraction – віднімання) віднімає джерело з приймача і поміщає різницю в приймач (табл. 4). Приймач може бути регістром або змінною, джерело може бути числом, регістром або змінною, але не можна використовувати змінну одночасно і для джерела, і для приймача.

Таблиця 4

Основні характеристики команди `SUB`

Команда	Призначення	Процесор
`SUB приймач, джерело`	Віднімання	8086

Так само, як і команда `ADD`, `SUB` не робить різниці між числами зі знаком і без знака, але прапорці дають змогу використовувати її для всіх чисел. Синтаксис команда `SUB` такий:

```
приймач: = приймач – джерело
```

Команда `SBB` (від англ. subtract with borrow – віднімати з

позичанням) в усьому є аналогічною SUB, крім того, що вона віднімає з приймача значення джерела й додатково віднімає значення прапорця CF (табл. 5).

Таблиця 5

Основні характеристики команди SBB

Команда	Призначення	Процесор
SBB приймач, джерело	Віднімання з позичанням	8086

Синтаксис команди SUB є таким:

```
приймач: = приймач – джерело – CF
```

Цю команду можна використовувати для віднімання 64-бітових чисел в EDX:EAX і EBX:ECX аналогічно командам ADD і ADC:

```
sub eax,ecx
sbb edx,ebx
```

Якщо при відніманні молодших подвійних слів відбулося позичання, то його буде враховано при відніманні старших.

Команда MUL виконує множення вмісту джерела (реєстру або змінної) і регістра AL, AX, EAX (залежно від розміру джерела) і поміщає результат в AX, DX: AX, EDX: EAX відповідно (табл. 6).

Таблиця 6

Основні характеристики команди MUL

Команда	Призначення	Процесор
MUL джерело	Множення чисел без знака	8086

Якщо старша половина результату (AH, DX, EDX) містить тільки нулі (результат цілком помістився в молодшу половину), то прапорці CF і OF встановлюються в 0, інакше – в 1. Значення інших прапорців (SF, ZF, AF і PF) не визначено.

```
mov ax, 105
mul x          ; AX = AX * x, AX = 26880, CF = OF = 0
mov eax, 500000
mov ebx, 100000
mul ebx        ; EDX:EAX = EAX * EBX,
               ; EDX:EAX = 50000000000,
               ; CF = OF = 1
```

Команда IMUL (від англ. indication multiply – помножити зі знаком)

має три форми, що різняться кількістю операндів (табл. 7):

1. Джерело (регістр або змінна) множиться на AL, AX або EAX (залежно від розміру операнда), і результат розташовується в AX, DX:AX або EDX:EAX відповідно [IMUL джерело].

2. Джерело (число, регістр або змінна) множиться на приймач (реєстр), і результат заноситься в приймач [IMUL приймач, джерело].

3. Джерело1 (реєстр або змінна) множиться на джерело2 (число), і результат заноситься в приймач (реєстр) [IMUL приймач, джерело1, джерело2].

Таблиця 7

Основні характеристики команди IMUL

Команда	Призначення	Процесор
IMUL джерело IMUL приймач, джерело IMUL приймач, історнік1, історнік2	Множення чисел зі знаком	8086 80386 80186

У всіх трьох варіантах уважається, що результат може займати в два рази більше місця, ніж розмір джерела. У першому випадку приймач автоматично виявляється досить великим, але в другому й третьому випадках можуть відбутися переповнення й утрата старших бітів результату, тоді прапорці OF і CF дорівнюватимуть одиниці. Якщо результат множення помістився цілком у приймач (у другому й третьому випадках) або в молодшу половину приймача (у першому випадку), то ці прапорці будуть дорівнювати нулю. Значення прапорців SF, ZF, AF і PF після команди IMUL не визначено.

Команда DIV (від англ. divide – ділити; division – ділення) виконує цілочислове ділення без знака AL, AX або EAX (залежно від розміру джерела) на джерело (регістр або змінну) і поміщає результат в AL, AX або EAX, а залишок – в AH, DX або EDX відповідно (табл. 6.8). Результат завжди округлюється в бік нуля, абсолютне значення залишку завжди є меншим від абсолютного значення дільника. Значення прапорців CF, OF, SF, ZF, AF і PF після цієї команди не визначено, а переповнення або ділення на нуль викликає виняток #DE (помилка під час розподілу) у захищеному режимі й переривання 0 – у реальному.

Таблиця 8

Основні характеристики команди DIV

Команда	Призначення	Процесор
DIV джерело	Цілочислове ділення без знака	8086

Оскільки регістри можуть містити тільки цілочислові значення, результат ділення розбито на частку й остачу. Тепер залежно від розміру джерела частка зберігається в EAX, а остача – в EDX (табл. 9).

Таблиця 9

Результат ділення командою DIV

Розмір джерела	Ділення	Частка в ...	Остача в ...
BYTE (8-bits)	ax / дільник	AL	AH
WORD (16-bits)	dx:ax / дільник	AX	DX
DWORD (32-bits)	edx:eax / дільник	EAX	EDX

Приклад

Якщо DX = 2030h, а AX = 0040h, то DX:AX = 20300040h.

Тут DX:AX – значення dword, де DX відображає старше слово, а AX – молодше.

EDX:EAX – значення quadword (64 біти), де старше dword знаходиться в EDX, а молодше – в EAX.

Джерелом операції ділення може бути:

1. 8-бітовий регістр (AL, AH, CL, ...).
2. 16-бітовий регістр (AX, DX , ...).
3. 32-бітовий регістр (EAX, EDX, ECX ...).
4. 8-бітове значення з пам'яті (byte ptr [xxxx]).
5. 16-бітове значення з пам'яті (word ptr [xxxx]).
6. 32-бітове значення з пам'яті (dword ptr [xxxx]).

Джерело не може бути безпосереднім значенням, тому що тоді процесор не зможе визначити розмір вихідного операнду:

```
mov ax, 127
mov bl, 5
div bl              ; AL = 19h = 25, AH = 02h = 2
```

Команда IDIV (від англ. indication divide – поділити зі знаком) виконує цілочислове ділення зі знаком AL, AX або EAX (залежно від розміру джерела) на джерело (регістр або змінну) і поміщає результат в AL, AX або EAX, а остачу – в AH, DX або EDX відповідно (табл. 10).

Таблиця 10

Основні характеристики команди IDIV

Команда	Призначення	Процесор
IDIV джерело	Цілочислове ділення зі знаком	8086

Результат завжди округлюється в бік нуля, знак остачі завжди збігається зі знаком діленого, абсолютне значення остачі завжди менше абсолютного значення дільника. Значення прапорців CF, OF, SF, ZF, AF і PF після цієї команди не визначено, а переповнення або ділення на нуль викликає виняток #DE (помилка під час розподілу) у захищеному режимі й переривання 0 – у реальному:

```
mov ax, 127
mov bl, -5
idiv bl           ; AL = e7h = -25, AH = 02h = 2

mov ax, -127
mov bl, 5
idiv bl           ; AL = e7h = -25, AH = feh = -2

mov ax, -127
mov bl, -5
idiv bl           ; AL = 19h = 25, AH = feh = -2
```

Команда INC (від англ. increment – зростання, збільшення, incrementation – приріст, збільшення на одиницю) збільшує приймач (реєстр або змінну) на одиницю, при цьому прапорець CF не використовується. Інші арифметичні прапорці (OF, SF, ZF, AF, PF) установлюються відповідно до результату складання (табл. 11).

Таблиця 11

Основні характеристики команди INC

Команда	Призначення	Процесор
INC приймач	Інкремент	8086

Приклад:

```
mov al, 15
inc al        ; тепер AL = 16 (еквівалентно add al, 1)
```

Команда DEC (від англ. decrement – зниження, зменшення) зменшує приймач (реєстр або змінну) на одиницю, при цьому прапорець CF не використовується. Інші арифметичні прапорці (OF, SF, ZF, AF, PF) установлюються відповідно до результату віднімання (табл. 12).

Приклад:

```
mov al, 15
dec al      ; тепер AL = 14 (еквівалентно sub al, 1)
```

Таблиця 12

Основні характеристики команди DEC

Команда	Призначення	Процесор
DEC приймач	Декремент	8086

Команда NEG (від англ. negative – від'ємний) виконує над числом, що знаходиться в приймачі (реєстрі або змінній), операцію доповнення до

двох (табл. 13).

Таблиця 13

Основні характеристики команди NEG

Команда	Призначення	Процесор
NEG приймач	Змінення знака	8086

Ця операція є еквівалентною зверненню знака операнда, якщо розглядати його як число зі знаком. Якщо приймач дорівнює нулю, то прапорець CF установлюється в 0, інакше – в 1. Інші прапорці (OF, SF, ZF, AF, PF) установлюються відповідно до результату операції.

Приклад:

```
mov ax, 1
neg ax      ; AX = -1 = ffffh
```

Команда CMP (від англ. compare – порівняти, звірити) порівнює приймач і джерело й установлює прапорці. Порівняння здійснюється шляхом вирахування джерела (числа, регістру або змінної) з приймача (реєстру або змінної; приймач і джерело не можуть бути змінними одночасно), причому результат віднімання нікуди не записується, єдиним результатом роботи цієї команди є змінення прапорців CF, OF, SF, ZF, AF і PF (табл. 14).

Таблиця 14

Основні характеристики команди CMP

Команда	Призначення	Процесор
CMP приймач, джерело	Порівняння	8086

Зазвичай команду CMP використовують разом з командами умовного переходу (J**), умовного пересилання даних (CMOV**) або умовного установлення байтів (SET**), які дають змогу використовувати результат порівняння, не звертаючи уваги на детальне значення кожного прапорця. Так, команди CMOVE, JE і SETE виконують відповідні дії, якщо значення операндів попередньої команди CMP були однаковими.

Незважаючи на те, що умовні команди майже завжди застосовуються відразу після CMP, не треба забувати, що так само їх можна використовувати після будь-якої команди, що модифікує прапорці, наприклад: перевірити рівність AX нулю можна коротшою командою

```
test ax,ax
```

а рівність одиниці – однобайтною командою

```
dec ax
```

Команда CMPXCHG (від англ. compare and exchange – порівняти й обміняти) порівнює значення, що міститься в AL, AX, EAX (залежно від розміру операндів), з приймачем (регістром). Якщо вони однакові, то вміст джерела копіюється в приймач і прапорець ZF установлюється в «1», якщо не однакові, то вміст приймача копіюється в AL, AX, EAX і прапорець ZF установлюється в «0». Інші прапорці установлюються за результатом операції порівняння, як після CMP. Джерело завжди є регістром, приймач може бути регістром або змінною (табл. 15).

Таблиця 15

Основні характеристики команди CMPXCHG

Команда	Призначення	Процесор
CMPXCHG приймач, джерело	Порівняти й обміняти між собою	80486

Команда CMPXCHG8B виконує порівняння вмісту регістрів EDX:EAX як 64-бітного числа (молодше подвійне слово в EAX, старше – в EDX) з приймачем (8-байтова змінна в пам'яті). Якщо вони однакові, то вміст регістрів ECX:EBX як 64-бітове число (молодше подвійне слово в EBX, старше – в ECX) копіюється в приймач, якщо інакше – то вміст приймача копіюється в EDX:EAX (табл. 16).

Таблиця 16

Основні характеристики команди CMPXCHG8B

Команда	Призначення	Процесор
CMPXCHG8B приймач	Порівняти й обміняти вісім байтів	P5

Логічні операції

Найпоширеніші варіанти значень, яких може набувати один біт, – це значення «правда» (True) і «неправда» (False), що використовуються в логіці, звідки походять так звані *логічні операції* над бітами. Так, якщо об'єднати «правду» і «правду», то одержимо «правду», а якщо об'єднати «правду» і «неправду» – то «правду» не отримаємо. У мові Асемблер є чотири основні операції – і (AND), або (OR), «виключне або» (XOR) і заперечення (NOT), дію яких наведено в табл. 17.

Таблиця 17

Логічні операції

І (логічне множення)	АБО (логічне складання)	Виключне АБО (сума за модулем 2)	Заперечення
0 AND 0 = 0 0 AND 1 = 0 1 AND 0 = 0 1 AND 1 = 1	0 OR 0 = 0 0 OR 1 = 1 1 OR 0 = 1 1 OR 1 = 1	0 XOR 0 = 0 0 XOR 1 = 1 1 XOR 0 = 1 1 XOR 0 = 0	NOT 0 = 1 NOT 1 = 0

Усі перелічені операції є побітовими, тому для виконання логічної дії над числом треба перевести його в двійковий формат і зробити операцію над кожним бітом, наприклад:

```
96h OR 0Fh = 10010110b
OR 01000000b =
11010110b = D6h
```

Логічні команди

Команда AND (від англ. and – і) виконує побітове «логічне І» над приймачем (регістром або змінною) і джерелом (числом, регістром або змінною) і поміщає результат у приймач. Джерело й приймач не можуть бути змінними одночасно. Будь-який біт результату дорівнює 1, тільки якщо відповідні біти обох операндів дорівнювали 1, і дорівнює 0 в інших випадках (табл. 18).

Таблиця 18

Основні характеристики команди AND

Команда	Призначення	Процесор
AND приймач, джерело	Логічне І	8086

Найбільш часто AND застосовують для вибіркового обнулення окремих бітів, наприклад, команда

```
and al,00001111b
```

обнулить старші чотири біти регістра AL, зберігши незмінними чотири молодших.

Прапорці OF і CF обнуляються, SF, ZF і PF установлюються відповідно до результату, AF не визначено. Для беззнакових і додатних чисел остача від ділення на 2^n – це останні n бітів числа. Тому для отримання остачі від ділення на 2^n потрібно виокремити ці останні n бітів з допомогою операції AND:

```
mov EAX, x
and EAX, 111b              ;EAX = EAX % 2^3
```

Команда OR (від англ. or – або) виконує побітове «логічне АБО» над приймачем (реєстром або змінною) і джерелом (числом, регістром або змінною) і поміщає результат у приймач. Джерело й приймач не можуть бути змінними одночасно. Будь-який біт результату дорівнює нулю, тільки якщо відповідні біти обох операндів дорівнювали нулю, і дорівнює одиниці в інших випадках (табл. 19).

Таблиця 19

Основні характеристики команди OR

Команда	Призначення	Процесор
OR приймач, джерело	Логічне АБО	8086

Команду OR найчастіше використовують для вибіркового установлення в «1» окремих бітів. Наприклад, команда

```
or al, 00001111b
```

установлює молодші чотири біти регістра AL в «1», а старші залишаться такими самими. При виконанні команди OR прапорці OF і CF обнуляються, SF, ZF і PF установлюються відповідно до результату, AF не визначено.

Команда XOR (від англ. exclusionary or – виключне або) виконує побітове «логічне виключне АБО» над приймачем (реєстром або змінною) і джерелом (числом, регістром або змінною) і поміщає результат у приймач. Джерело і приймач не можуть бути змінними одночасно. Будь-який біт результату дорівнює одиниці, якщо відповідні біти операндів є різними, і нулю, якщо однаковими (табл. 20). Команда XOR використовується для різних операцій.

Таблиця 20

Основні характеристики команди XOR

Команда	Призначення	Процесор
XOR приймач, джерело	Логічне виключне АБО	8086

Приклади

При виконанні команд

```
mov AX, 10011010b
mov BX, 01111001b
and AX, BX
```

має місце операція «І» над вмістом регістрів AX і DX. У цьому випадку отримаємо

```
1 0 0 1 1 0 1 0
AND
0 1 1 1 1 0 0 1
———————————————
0 0 0 1 1 0 0 0
```

Результат 00011000b заноситься в акумулятор. Регістр BX свого значення не змінює. При виконанні команд OR і XOR з тими самими значеннями операндів отримаємо

```
1 0 0 1 1 0 1 0
OR
0 1 1 1 1 0 0 1
———————————————
1 1 1 1 1 0 1 1
```

```
1 0 0 1 1 0 1 0
XOR
0 1 1 1 1 0 0 1
———————————————
1 1 1 0 0 0 1 1
```

Команда NOT (від англ. not – не) інвертує (замінює на протилежний) кожен біт приймача (реєстра або змінної): біт, що дорівнює нулю, установлюється в одиницю, а біт, що дорівнює «1», скидається в «0». Прапорці не будуть зачіпатися (табл. 21).

Таблиця 21

Основні характеристики команди NOT

Команда	Призначення	Процесор
NOT приймач	Інверсія	8086

Команда TEST (від англ. test – перевірка, тест) обчислює результат дії побітового «логічного І» над приймачем (регістром або змінною) і джерелом (числом, регістром або змінною) і встановлює прапорці SF, ZF і PF відповідно до отриманого результату без його збереження (прапорці OF і CF обнуляються, значення AF є невизначеним) (табл. 22). Джерело й приймач не можуть бути змінними одночасно. Команда TEST, так само як і CMP, використовується зазвичай у поєднанні з командами умовного переходу (J*), умовного пересилання даних (CMOV*) і умовного установлення байтів (SET*).

Таблиця 22

Основні характеристики команди NOT

Команда	Призначення	Процесор
TEST приймач, джерело	Логічне порівняння	8086

Приклад:

```
test EAX, 100b
jnz зміщення
```

Команда JNZ виконає перехід, якщо в регістрі EAX третій біт праворуч є установленим. Дуже часто команду TEST використовують для перевірки рівності нулю регістра:

```
test ECX, ECX
jz зміщення
```

Команда JZ виконає перехід, якщо ECX = 0.

Зсувні операції

Операції *зсуву вправо* й *уліво* зрушують біти в змінній на задану кількість позицій. Кожна команда зсуву має два види:

```
<мнемокод> <операнд>, <безпосередній операнд>

<мнемокод> <операнд>, CL
```

Перший операнд має бути регістром або коміркою пам'яті. Саме в ньому здійснюється зсув. Другий операнд визначає кількість позицій для зсуву, яке задається безпосереднім операндом або зберігається в регістрі CL (і тільки CL). З цього регістра враховуються тільки молодші п'ять бітів, що можуть набувати значень від 0 до 31.

Команди зсуву змінюють прапорці CF, OF, PF, SF і ZF.

Існує кілька видів зсуву, які відрізняються тим, як заповнюються вивільнювані біти (табл. 23).

Таблиця 23

Основні характеристики команд зсуву

Команда	Призначення
SAR приймач, лічильник	Арифметичний зсув управо
SAL приймач, лічильник	Арифметичний зсув уліво
SHR приймач, лічильник	Логічний зсув управо
SHL приймач, лічильник	Логічний зсув уліво

Ці чотири команди виконують двійковий зсув приймача (регістра або

змінної) управо (у сторону молодшого біта) або уліво (у сторону старшого біта) на значення лічильника (число або регістр CL).

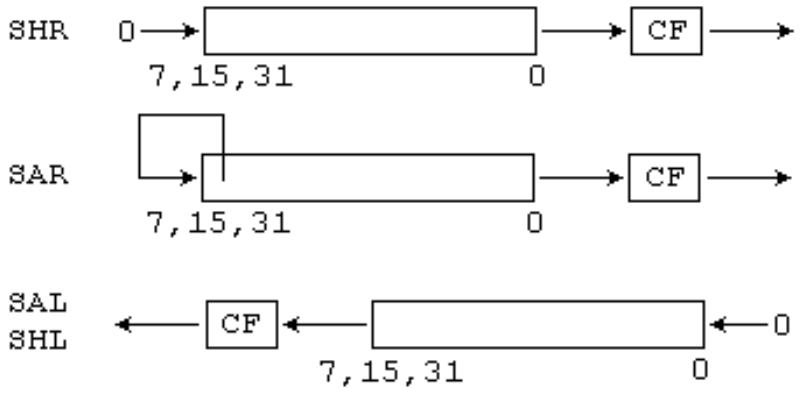
Операція зсуву на одиницю еквівалентна множенню (зсуву уліво) або діленню (зсуву управо) на два. Так, число 0010b (2d) після зсуву на одиницю вліво перетворюється на 0100b (4d).

***Команди* SAL** і **SHL** виконують одну й ту саму операцію (насправді, це – одна й та сама команда) – на кожен крок зсуву старший біт заноситься в CF, усі біти зсуваються уліво на одну позицію,а молодший біт обнуляється.

Команда SHR виконує протилежну операцію: молодший біт заноситься в CF, усі біти зсуваються на одиницю вправо, старший біт обнуляється. Ця команда є еквівалентною беззнаковому цілочисловому діленню на два.

Команда SAR діє за аналогією з SHR, тільки старший біт не обнуляється, а зберігає попереднє значення, так що, наприклад, число 11111100b (–4d) перейде в 11111110b (–2d). Команда SAR, таким чином, є еквівалентною знаковому діленню на два, але на відміну від команди IDIV округлення відбувається не в бік нуля, а в бік від'ємної нескінченності.

Так, якщо поділити –9d на 4d з допомогою команди IDIV, то результат буде дорівнювати –2d (остача –1d), а якщо виконати арифметичний зсув управо на 2d числа –9d, то результат буде дорівнювати –3d. Зсуви більш ніж на одиницю є еквівалентними відповідним зсувам на одиницю, виконаним послідовно. Схему всіх зсувних операцій зображено на рис. 1.



$AX / 2^N = SAR\ AX, N$

$AX * 2^N = SAL\ AX, N$

Рис. 1. Зсувні операції

При *логічному зсуві* біти, що вивільняються, заповнюються нулями. Останній біт, що вивільнився, зберігається у прапорці CF:

```
SHL <операнд>, <кількість>     ;Логічний зсув уліво

SHR <операнд>, <кількість>     ;Логічний зсув управо
```

Арифметичний зсув уліво є еквівалентним логічному зсуву вліво (це одна й та сама команда): біти, що вивільняються заповнюються нулями. При *арифметичному зсуві вправо* біти, що вивільняються, заповнюються знаковим бітом. Останній біт, що вивільнився, зберігається у прапорці `CF`:

```
SAL<операнд>,<кількість>  ;Арифметичний зсув уліво

SAR <операнд>,<кількість> ;Арифметичний зсув управо
```

Для множення використовується зсув уліво. Незважаючи на наявність двох команд, по суті зсув уліво – один. Його використовують для множення як знакових, так і беззнакових чисел. Однак результат буде правильним тільки в тому випадку, якщо він уміщається в регістр або елемент пам'яті.

Приклади:

```
mov ax, 250   ;AX = 00fah = 250
sal ax, 4     ;Множення на 24 = 16, AX = 0fa0h = 4000

mov ax, 1     ;AX = 1
sal ax, 10    ;Множення на 210, AX = 0400h = 1024

mov ax, -48   ;AX = ffd0h = -48 (у додатковому коді)
sal ax, 2     ;AX = ff40h = -192 (у додатковому коді)

mov ax, 26812     ;AX = 68bch = 26812
sal ax, 1         ;AX = d178h = -11912
                ;Знакове додатне число стало від'ємним
mov ax, 32943     ;AX = 80afh = 32943
sal ax, 2         ;AX = 02bch = 700
     ; Більше беззнакове число стало набагато меншим
```

Для ділення використовується зсув управо. При діленні немає проблем з переповненням, але для знакових і беззнакових чисел треба використовувати різні механізми.

Для ділення беззнакових чисел слід використовувати логічний зсув управо:

```
mov ax, 43013     ;AX = a805h = 43013
shr ax, 1         ;AX = 5402h = 21506
```

Циклічні зсуви

Ці команди здійснюють циклічний зсув приймача (реєстру або змінної) на кількість бітів, зазначену в лічильнику (число або регістр `CL`, з якого враховуються тільки молодші п'ять бітів, що набувають значення від 0 до 31) (табл. 24).

Таблиця 24

Основні характеристики команд, що здійснюють циклічний зсув

Команда	Призначення
ROR `приймач, лічильник`	Циклічний зсув управо
ROL `приймач, лічильник`	Циклічний зсув уліво
RCR `приймач, лічильник`	Циклічний зсув управо з прапорцем `CF`
RCL `приймач, лічильник`	Циклічний зсув уліво з прапорцем `CF`

При виконанні циклічного зсуву на одиницю команди `ROR` (`ROL`) зрушують кожен біт приймача вправо (уліво) на одну позицію, за винятком наймолодшого (старшого), який записується в позицію найстаршого (молодшого) біта. Команди `RCR` і `RCL` виконують аналогічну дію, але включають прапорець `CF` у цикл, нібито він є додатковим бітом у приймачі (рис. 2).

При *циклічному зсуві* біти, що «вивільняються», заповнюються вивільненими бітами. Останній біт, що вивільнився, зберігається в прапорці `CF`.

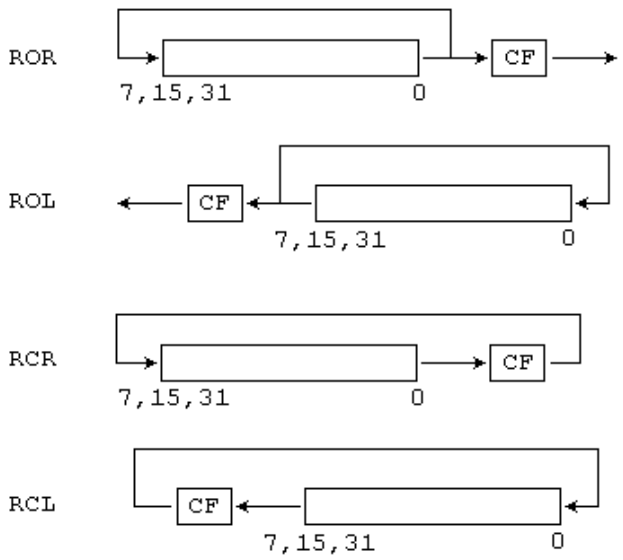


Рис. 2. Циклічні зсуви

```
;Циклічний зсув уліво
ROL <операнд>, < кількість зсувів>
```

```
;Циклічний зсув управо
ROR <операнд>, < кількість зсувів>
```

Після виконання команд циклічного зсуву прапорець CF завжди дорівнює останньому біту, що вийшов за межі приймача, прапорець OF визначено тільки для зсувів на одиницю. Його встановлюють, якщо змінилося значення найстаршого біта, і скидають, якщо старший біт не змінився. Прапорці SF, ZF, AF і PF не змінюються.

Завдання

Завдання полягає в обчисленні результату виконання арифметичного виразу, у якому деякі числа є постійними, а інші – змінними. Формула для обчислень має вигляд

```
X = (A * 2 + B * C) / (D – 3 )
```

Наведена програма спочатку резервує комірки пам'яті під змінні, потім виконує множення однобайтних чисел (A * 2), результат множення – двобайтне число в регістрі AX, яке зберігається в регістрі CX. Далі виконується множення однобайтних чисел (B * C), результат – двобайтне число, яке зберігається в акумуляторі AX. Після складання двох співмножників і обчислення знаменника (D – 3) виконується ділення. Результат присвоюється змінній X.

1. Наберіть наведену програму 1, запишіть вихідний файл з розширенням *.ASM, отримайте файл з розширенням *.EXE.

2. Завантажте програму в налагоджувач TURBODEBUG (команда TURBODEBUG *.exe). Виконайте програму по кроках і подивіться, як вона працює.

3. Виконайте програму з п’ятьма варіантами різних початкових значень змінних A, B, C, D по кроках (табл. 25) і запишіть результат виконання до табл. 25 (у регістрі AL – частка, AH – остача). Переведіть результат у десяткову систему.

Таблиця 25

Варіанти початкових значень змінних A, B, C, D

Варіант		1	2	3	4	5	6	7	8	9	10
Значення змінних	A	3	0AH	30	60	3	0AH	0BH	6	20	20H
	B	4	5H	4	16	4	5H	4H	16	9	9H
	C	2	8H	15	5	2	8H	5H	5	4	4H
	D	5	9H	6	18	5	9H	2H	8	1	1CH
Частка AL											
Остача AH											

Продовження таблиці 25

Варіант		11	12	13	14	15	16	17	18	19	20
Значення змінних	A	3	15	18	20	3	0AH	30	60	18	20H
	B	4	8	4	9	4	5H	4	16	9	9H
	C	2	4	6	4	2	8H	15	5	4	4H
	D	5	3	9	1	5	9H	6	18	8	1CH
Частка AL											
Остача AH											

```
;Програма 1
;x =(a*2+b*c)/(d-3)
.model small
.stack 100h
.data
      a   db   ?
      b   db   ?
      c   db   ?      ;Резервуємо пам'ять для змінних
      d   db   ?      ;A,B,C,D,X
      x   dw   ?
.code
start:
      mov  ax,@data
      mov  ds,ax
      mov  a,3
      mov  b,4
      mov  c,2
      mov  d,5
      mov  al,2
      mul   a
      mov  cx,ax
      mov  al,b
      mul  c
      add  ax,cx
      mov  cl,d
      sub  cl,3
      div  cl
      mov  x,ax
      mov  ah,4ch
      int  21h
end  start
```

4. Наберіть наведену програму 2, запишіть вихідний файл з розширенням *.ASM, отримайте файл з розширенням *.EXE.

5. Завантажте програму в налагоджувач TURBODEBUG (команда TURBODEBUG *.exe). Виконайте програму по кроках і подивіться, як вона працює. Зверніть увагу на прапорець CF.

6. Виконайте програму 2 згідно варіанту початкових значень регістра AL. Результат запишіть в табл. 26 у двійковій і десятковій системах.

Таблиця 26

Результат виконання програми 2

№	Команди	SHL	SHR	SAR	ROR	ROL
1	01001101b					
2	01101010b					
3	10111101b					
4	11011011b					
5	10111100b					
6	01001101b					
7	01101010b					
8	10101111b					
9	11011011b					
10	10101100b					
11	01001111b					
12	01111010b					
13	10101101b					
14	11011011b					
15	10101100b					
16	10111101b					
17	11011011b					
18	10111100b					
19	11011111b					
20	10111100b					

```
;Програма 2
.model tiny
.stack 100h
.code
start:
     mov ah, 01001101b
     shr ah,1
     mov ah, 01001101b
     shl ah,1
     mov ah, 01001101b
     sar ah,1
     mov ah, 01001101b
     ror ah,1
```

```
     mov ah, 01001101b
     rol ah,1
     mov ax,4c00h
     int 21h
end start
```

7. Зробить звіт з виконання розрахункової роботи.