ЛАБОРАТОРНА РОБОТА №3

Робота з командами арифметичних операцій

|  |
| --- |
|  |

***Мета:*** ознайомлення з системами числення та переведення чисел з однієї системи в іншу; застосування арифметичних операцій над двійковими числами для на-писання програм ; застосування арифметичних операцій над десятковими (ВСD) числами для написання програм.

План:

1. Двійкова система числення.
2. Шістнадцяткова система числення.
3. Цілочисельні арифметичні команди: додавання, віднімання, множення, ділення.
4. Оператори зсуву.
5. Логічні оператори.

Короткі теоретичні відомості

Двійкова система числення

Набір цифр для двійкової системи числення:{0, 1}, основа степені *р =* 2.

Кількісний еквівалент деякого цілого n-значного двійкового числа обчислюється відповідно до формули :



Наявність цієї системи числення обумовлено тим, що комп'ютер побудований на логічних схемах, що мають у своєму елементарному виді тільки два стани - увімкнений і вимкнений.

Наприклад, розглянемо двійкове число 10100111.

Обчислимо кількісний еквівалент цього двійкового числа це буде величина, рівна наступній сумі:

1 27 + 0 26 + 1 25 +0 24 + 0 23 + 1 22 + 1 21 + 1 20=167

Додавання і віднімання двійкових чисел виконується так само, як і для інших позиційних систем числення, наприклад десяткової. Точно так само виконуються позика і перенос одиниці з (в) старший розряд.

Наприклад:

11 11111 перенесення

110011011 +

110010101

1100110000

Шістнадцяткова система числення

Дана система числення має наступний набір цифр:

{0, 1, 2, .... 9, А, В, С, D, Е, F}, основа системи *р* = 16.

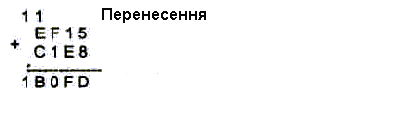
**Таблиця**

|  |  |  |
| --- | --- | --- |
| Десяткове число | Двійкова тетрада | Шістнадцяткове число |
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| 10 | 1010 | А,а |
| 11 | 1011 | B,b |
| 12 | 1100 | С, с |
| 13 | 1101 | D,d |
| 14 | 1110 | Е,е |
| 15 | 1111 | F,f |
| 16 | 10000 | 10 |

Таблиця містить представлення десяткових чисел з діапазону 0-16 у двійковій і шістнадцятковій системах числення. Таблицю зручно використовувати для взаємного перетворення чисел. Шістнадцяткова система числення при проведені обчислень трохи складніша, ніж двійкова, зокрема, у тому, що стосується правил переносів у старші розряди. Головне тут — пам'ятати наступну рівність:

(1+F=10)16

Ці переходи дуже важливі при виконанні додавання і вирахування шістнадцяткових чисел.



Переведення чисел з однієї системи числення в іншу

Цілочисельні арифметичні команди

Двійкова арифметика.

**Команди додавання :**

1. inc *операнд* — операція *інкремента,* тобто збільшення значення операнда на 1;
2. add *операнд\_1,операнд\_2 —* команда додавання з принципом дії:

**операнд\_1 = операнд\_1 + операнд\_2;**

1. adc *операнд\_1,операнд\_2 —* команда додавання з урахуванням прапора переносу cf.

Приклад 3.1.: Використання команди додавання

model small

.stack 100h

.data

a dw 3h ;змінна a = 3

b dw 5h ;змінна b = 5

c dw ? ;змінна c = ?

.code

start:

mov ax, a ;копіювання змінної в регістр

add ax, b ;a + b

mov c, ax ;c = a + b

mov ax, 4c00h

int 21h

end start

**Команди віднімання:**

1. dec *операнд —* операція декремента, тобто зменшення значення операнда на 1;
2. sub *операнд\_1,операнд\_2 —* команда віднімання; її принцип дії:

**операнд\_1 = операнд\_1 - операнд\_2;**

1. sbb *операнд\_1,операнд\_2 —* команда віднімання з врахуванням позики (прапорця cf):

**операнд\_1 = операнд\_1 - операнд\_2 - значення\_сf.**

Приклад 3.2.: Використання команди віднімання

model small

.stack 100h

.data

a dw 3h ;змінна a = 3

b dw 5h ;змінна b = 5

c dw ? ;змінна c = ?

.code

start:

mov ax, b ;копіювання змінної в регістр ax

sub ax, a ;b - a

mov c, ax ;c = b - a

mov ax, 4c00h

int 21h

end start

**Команди множення**

Для множення чисел без знаку призначена команда

mul **множник\_1**

У команді зазначений усього лише один операнд-множник. Другий *операнд-множник\_2* заданий неявно. Його місце розташування фіксоване і залежить від розміру співмножників. Тому що в загальному випадку результат множення більший, ніж кожний з його співмножників, то його розмір і місце розташування повинні бути теж визначені однозначно. Варіанти розмірів співмножників і розміщення другого операнда і результату приведені в таблиці.

**Таблиця**

|  |  |  |
| --- | --- | --- |
| **співмножник1** | **співмножник2** | **Результат** |
| Байт  Слово  Подвійне слово | аl    ах  еах | 16 біт в ах: аl - молодша частина результату; ah старша частина результату  32 біт у парі dx:ax; ax — молодша частина результату; dx — старша частина результату  64 біт у парі edx:eax: еах- молодша частина результату; edx — старша частина результату |

З таблиці видно, що добуток складається з двох частин і в залежності від розміру операндів розміщується в двох місцях — на місці *множник\_2* (молодша частина) і в додаткових регістрах ah, dx, edx (старша частина).

Для множення чисел зі знаком призначена команда

**imul операнд\_ 1[, операнд\_2, операнд\_3]**

Ця команда виконується так само, як і команда mul*.* Відмінною рисою команди imul є тільки формування знака. Якщо результат малий і вміщається в одному регістрі (тобто якщо cf = of = 0), то вміст іншого регістра (старшої частини) є розширенням знака — усі його біти дорівнюють старшому біту (знаковому розряду) молодшої частини результату. В іншому випадку (якщо cf = of = 1) знаком результату є знаковий біт старшої частини результату, а знаковий біт молодшої частини є значущим бітом двійкового коду результату.

Приклад 3.3.: Використання команди множення

model small

.stack 100h

.data

a dw 3h ;змінна a = 3

b dw 5h ;змінна b = 5

c dw ? ;змінна c = ?

.code

start:

mov ax, a ;копіювання змінної в регістр ax

mul b ;a \* b

mov c, ax ;c = a \* b

mov ax, 4c00h

int 21h

end start

**Ділення**

Для ділення чисел без знака призначена команда

**div *дільник***

Дільник може знаходитися в пам'яті чи в регістрі і мати розмір 8, 16 чи 32 біт. Місцезнаходження діленого фіксоване і так само, як у команді множення, залежить від розміру операндів. Результатом команди ділення є значення частки і залишку. Варіанти місця розташування і розмірів операндів операції ділення показані в таблиці.

**Таблиця 3.1**

Розташування операндів і результату при ділені

|  |  |  |  |
| --- | --- | --- | --- |
| Ділене | Дільник | Частка | Залишок |
| Слово 16 біт  у регістрі ах | Байт -регістр чи  и комірка пам'яті | Байт у регістрі аl | Байт у регістрі ah |
| 32 біт  dx — старша частина  ах — молодша частина | 16 біт  регістр чи комірка  пам'яті | Слово 16 біт  у регістрі ах | Слово 16 біт  у регістрі dx |
| 64 біт  edx — старша частина  еах — молодша частина | Подвійне слово  32 біт  регістр чи комірка  пам'яті | Подвійне слово  32 біт  у регістрі еах | Подвійне слово  32 біт  у регістрі edx |

Після виконання команди ділення вміст прапорців невизначено, але можливе виникнення переривання з номером 0 («ділення на нуль»). Цей вид переривання відноситься до **виключень***.* Він виникає усередині мікропроцесора через аномалії під час обчислювального процесу.

Переривання 0 — «ділення на нуль» — при виконанні команди div може виникнути по причині:дільник дорівнює нулю;частка не входить у відведену під нього розрядну сітку, що може відбутися у випадках:

1. при діленні діленого завбільшки слово на дільник завбільшки байт, причому значення діленого в більш ніж 256 разів більше значення дільника;
2. при діленні діленого завбільшки подвійне слово на дільник завбільшки слово, причому значення діленого в більш ніж 65 536 разів більше значення дільника;
3. при діленні діленого завбільшки чотирикратне слово на дільник завбільшки подвійне слово, причому значення діленого в більш ніж 4 294 967 296 разів більше значення дільника.

Для ділення чисел із знаком призначена команда:

**idiv *дільник***

Для цієї команди справедливі всі розглянуті положення, що стосуються команд і чисел зі знаком. Виникнення переривання 0 «ділення на нуль» у випадку чисел зі знаком виникає при виконанні команди idiv по причині:

дільник дорівнює нулю; частка не входить у відведену для нього розрядну сітку:

що у свою чергу, може відбутися:

1. при діленні діленого завбільшки слово зі знаком на *дільник* завбільшки байт зі знаком, причому значення діленого в більш ніж 128 разів більше значення *дільника* (таким чином, частка не повинна знаходитися поза діапазоном від -128 до 127);
2. при діленні діленого завбільшки подвійне слово зі знаком на *дільник* завбільшки слово зі знаком, причому значення діленого в більш ніж 32 768 разів більше значення *дільника* (таким чином, частка не повинна знаходитися поза діапазоном від -32 768 до 32 768);
3. при діленні діленого завбільшки чотирикратне слово зі знаком на *дільник* завбільшки подвійне слово зі знаком, причому значення діленого в більш ніж 2 147 483 648 разів більше значення *дільника* (таким чином, частка не повинна знаходитися поза діапазоном від -2 147 483 648 до 2 147 483 647).

Приклад 3.4.: Використання команди ділення

model small

.stack 100h

.data

a dw 3h ;змінна a = 3

b dw 5h ;змінна b = 5

c dw ? ;змінна c = ?

.code

start:

mov ax, a ;копіювання змінної в регістр ax

div b ;a / b

mov c, ax ;c = a / b

mov ax, 4c00h

int 21h

end start

**Зміна знаку**

**neg операнд**

Десяткова арифметика

**Корекція додавання**

Для корекції операції додавання двох однозначних неупакованих BCD-чисел у системі команд мікропроцесора використовується команда

**ааа (ASCII Adjust for Addition)**

-корекція результату додавання для представлення в символьному виді. Ця команда не має операндів. Вона працює неявно тільки з регістром аl і аналізує значення його молодшої тетради.

Як і для неупакованих BCD-чисел, для упакованих BCD-чисел існує необхідність коректувати результати арифметичних операцій. Мікропроцесор має для цього команду daa:

**daa (Decimal Adjust for Addition)**

- корекція результату додавання для представлення в десятковому виді. '

Команда daa перетворить уміст регістра аl у дві упаковані десяткові цифри.

**Корекція віднімання**

**aas (ASCII Adjust for Substraction)**

- корекція результату віднімання для представлення в символьному виді.

Команда aas не має операндів і працює з регістром аl, аналізуючи його молодшу тетраду.

Корекція результату віднімання BCD-чисел здійснюється командою das:

**das (Decimal Adjust for Substraction)**

- корекція результату вирахування для представлення в десятковому виді.

**Корекція множення**

Для корекції результату після множення застосовується команда

**ааm (ASCII Adjust for Multiplication)**

- корекція результату множення для представлення в символьному виді.

Вона не має операндів і працює з регістром ах у наступним чином:

- ділить аl на 10;

-результат ділення записується так: частка — в аl, залишок — в ah.

У результаті після виконання команди ааm у регістрах аl і ah знаходяться правильні двоїчно-десяткові цифри добутку двох цифр.

**Корекція ділення**

**aad (ASCII Adjust for Division)**

- корекція розподілу для представлення в символьному виді.

Команда не має операндів і перетворить двозначне неупаковане BCD-число в регістрі ах у двійкове число. Це двійкове число згодом буде відігравати роль діленого в операції розподілу. Крім перетворення, команда aad поміщає отримане двійкове число в регістр а1.

**Перетворення типів**

Команди без операндів — ці команди працюють з фіксованими регістрами:

1. **cbw (Convert Byte to Word)** — команда перетворення байта (у регістрі а1) у слово (у регістрі ах) шляхом поширення значення старшого біта а1 на всі біти регістра ah;
2. **cwd (Convert Word to Double)** — команда перетворення слова (у регістрі ах) у подвійне слово (у регістрах dx:ax) шляхом поширення значення старшого біта ах на всі біти регістра dx;
3. **cwde (Convert Word to Double)** — команда перетворення слова (у регістрі ах) у подвійне слово (у регістрі еах) шляхом поширення значення старшого біта ах на всі біти старшої половини регістра еах;
4. **cdq (Convert Double Word to Quarter Word)** — команда перетворення подвійного слова (у регістрі еах) в чотирикратне слово (у регістрах edx:eax) шляхом поширення значення старшого біта еах на всі біти регістра edx.

Приклад 3.5.:Використання арифметичних операцій

.model small

.stack 100h

.data

a db 0ffh, 0ffh ; 255

sum dw 0

LOCAL @@

.code

start:

mov ax, @data ;завантажити усе з data в ax

mov ds, ax

mov es, ax

mov al, a

add byte ptr sum, al

adc byte ptr sum+1, 0

mov al, byte ptr a+1

add byte ptr sum, al

adc byte ptr sum+1, 0

mov ax, sum ;result = 1FEh

exit:

mov ax, 4c00h

int 21h

end start

Приклад 3.6. Використання арифметичних операцій

.386

.model small

.stack 100h

.data

X dw 250

Y dw 5

F dd ?

.code

start: mov ax,@data

mov ds,ax

;F = (12563h - 78) / ? + (Y \* 2852)

mov dx,1

mov ax,2563h

sub ax,78 ;2563h - 78

sbb dx,0

div X

mov cx,ax

mov ax, 2852

mul Y

add ax,cx

adc dx,0

mov word ptr F,ax ;F=(12563h - 78) / X +

;(Y \* 2852)

mov word ptr F+2, dx

mov ah,4Ch

int 21h

end start

Оператори зсуву

Оператори зсуву виконують зсув виразу на зазначену кількість розрядів.

Зсув логічного операнда праворуч.

Алгоритм роботи:

1. зсув всіх бітів операнда вправо на один розряд; при цьому висунутий праворуч біт стає значенням прапора перенесення CF;
2. одночасно зліва в операнд всовується нульовий біт;
3. зазначені вище дві дії повторюються кількість разів, рівне значенню другого операнда.

Команда SHR використовується для логічного зсуву розрядів операнда вправо. Так само, як і для інших зрушень, значення другого операнда (лічильника зсуву) обмежено діапазоном 0...31. Це пояснюється тим, що мікропроцесор використовує тільки п'ять молодших розрядів операнда кількість\_розрядів. На відміну від інших команд зсуву, прапор із завжди скидається в нуль в операціях зсуву на один розряд.  
Команду SHR можна використовувати для поділу цілочисельних операндів без знаку на ступені 2.

Зсув логічного операнда ліворуч.

Алгоритм роботи:

1. зсув всіх бітів операнда вліво на один розряд, при цьому висунутий зліва біт стає значенням прапора перенесення cf;
2. одночасно зліва в операнд всовується нульовий біт;
3. зазначені вище дві дії повторюються кількість разів, рівне значенню другого операнда.

Команда SHR використовується для зсуву розрядів операнда вліво. Так само, як і для інших зрушень, значення другого операнда (лічильник зсуву) обмежено діапазоном 0...31. Аналогічно іншим командам зсуву зберігається ефект, пов'язаний з поведінкою прапора of, значення якого має сенс тільки в операціях зсуву на один розряд:

1. якщо of = 1, то поточне значення прапора cf і висунутого зліва біта операнда різні;
2. якщо of = 0, то поточне значення прапора cf і висунутого зліва біта операнда збігаються.

Приклад 3.7.: Використання операторів зсуву

model small

.stack 100h

.data

a dw 3h

b dw 5h

e dw 7h

c dw ?

.code

start:

mov ax, a

;Коли виконується логічний зсув, "звільнені"

;біти заповнюються нулями.

;Останній біт збережений в прапорі CF.

shl ax, 2 ;логічний зсув ліворуч

shr ax, 3 ;логічний зсув праворуч

mov ax, 4c00h

int 21h

end start

Логічні оператори

Логічні оператори виконують над виразами побітові операції. Вирази повинні бути абсолютними, тобто такими, числове значення яких може бути обчислено транслятором.

**Логічний оператор NOT:**

Інвертування операнда - not операнд.

Виконання цієї команди не впливає на значення прапорців. Команду також можна використовувати для зміни байта, що виконує роль деякого прапора, з метою відслідковування логічних умов в програмі, але такий спосіб не є оптимальним.

flag db 0ffh ;значення прапорця — істина

...

cycl:

...

cmp flag, 0

je m1

...

m1: not flag ;встановити прапорець в істину.

**Логічний оператор OR:**

Логічне АБО, операція логічного АБО над бітами операнда призначення - or джерело, маска.

Алгоритм роботи:

1. виконати операція логічного АБО над бітами операнда призначення, використовуючи в якості маски другий операнд - маска. При цьому біт результату рівний 0, якщо відповідні біти операндів маска і призначення рівні 0, інакше - біт рівний 1;
2. записати результат операції в джерело (операнд маска залишається незмінним);
3. встановити прапорці.

Команду можна використовувати для роботи з операндами на бітовому рівні. Типове використання команди - встановлення визначених розрядів першого операнда в одиницю.

mov al, 01h

or bl, al ;встановити нулевий біт в одиницю.

**Логічний оператор XOR:**

Операція логічного виключаючого АБО (викл.АБО) над двома операндами розмірністю байт, слово або подвійне слово - xor приймач, джерело.

Алгоритм роботи:

1. виконати операцію логічного викл.АБО над операндами: біт результату рівний 1, якщо значення відповідних бітів операндів різняться, інакше - біт результату рівний 0;
2. записати результат додавання в приймач;
3. встановити прапорці.

Команда xor використовується для виконання операції логічного викл.АБО двох операндів. Результат операції поміщається в перший операнд. Цю операцію зручно використовувати для інвертування чи порівняння визначених бітів операндів.

;змінити значення біта 0 регістра al на протилежне

xor al, 01h

**Логічний оператор AND:**

Операція логічного множення для операндів приймач і джерело розмірністю байт, слово або подвійне слово - and приймач, джерело.

Алгоритм роботи:

1. - виконати операцію логічного множення над операндами джерело та приймач, кожен біт результату рівний 1, якщо відповідні біти операндів рівні 1, інакше - біт результату рівний 0;
2. - записати результат операції в приймач;
3. - встановити прапорці.

Команда and використовується для логічного множення двох операндів. Результат операції поміщається по адресі першого операнда. Цю команду зручно використовувати для примусової установки чи скидання визначених бітів операнда. Наприклад, перетворимо двозначне упаковане BCD-число в його символьний еквівалент:

u\_BCD db 25h ;упаковане BCD-число

s\_ch dw 0 ;місце для результату

...

xor ax, ax ;очистка ax

mov al, u\_BCD

shl ax, 4 ;ax=0250

mov al, u\_BCD ;ax=0225

;перетворення в символьне представлення:

and ax, 3f3fh ;ax=3235h

mov s\_ch, ax

Приклад 3.8. Використання логічних операцій

model small

.stack 100h

.data

a dw 3h ;змінна a = 3

b dw 5h ;змінна b = 5

e dw 7h ;змінна e = 7

c dw ? ;змінна c = ?

.code

start:

mov ax, a

mov dx, b

mov bx, e

not ax ;Заперечення операції змінює значення всіх бітів ;змінної, і навпаки.

;Операція один операнд, який може бути регістром ;або комірка пам'яті.

; Операція не змінює прапори.

and ax, dx ;Побітове "і" виконує логічне множення всіх пар

;бітових ;операндів.

or ax, e ;Побітове "або" виконує ;функцію логічного додавання ;всіх пар бітових операндів.

xor dx, bx ;Операція порозрядне ;виключаюче "або" виконує ;додавання по модулю 2 всіх ;пар бітових операндів.

mov ax, 4c00h

int 21h

end start

Приклад 3.9. Введення з клавіатури символу та його виведення в коді ascii

model small

.stack 100h

.data

.code

start:

MOV AH,1 ; code for "read a character"

INT 21H ; character gets put in AL

MOV AH,2 ; code for "write a character"

MOV DL,'A' ; ascii code goes in DL

INT 21H

mov ax,4c00h

int 21h

end start

Приклад 3.10. Програма попросить вас ввести рядок і завершує всоє виконання виходи. Рядок зберігається в пам'яті, починаючи з адреси оголошеної змінної. Примітка: в даному випадку, повернення каретки (код ASCII 13) або LINE-FEED (код ASCII 10) означає кінець рядка.

model small

.stack 100h

.data

value DB ?

.code

start: MOV AX, @data

MOV DS, AX

MOV DX, OFFSET value

MOV AH, 3FH

INT 21H ;виклик MS-DOS для введення рядка

MOV ax, 4c00h INT 21H

END start

**Завдання до лабораторної роботи №3**

**Перший рівень**

ЗАВДАННЯ

1.1 Наберіть наведену нижче **програму 2.1** зі значеннями:

1.2.Завантажте програму в відладчик

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Варіант** |  | **1** | **2** | **3** | **4** | **5** |
| Значення | A | 3 | 0AH | 20 | 60 | 20H |
| B | 4 | 5H | 4 | 16 | 9H |
| C | 2 | 8H | 15 | 5 | 4H |
| D | 5 | 9H | 6 | 18 | 1CH |
| Частка | AL |  |  |  |  |  |
| Залишок | AH |  |  |  |  |  |

1.3. Виконайте програму з 5 варіантами різних початкових значень змінних по крокам та запишіть результат виконання в таблицю (в регістрі AL - частка, AH - залишок). Переведіть результат в десяткову систему.

1.4. Виконайте програму по крокам. При виконанні програми заповніть таблицю

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер  кроку | Команда | AX | BX | CX | DX | CS | IP | DS |
|  |  |  |  |  |  |  |  |  |

**Пpогpамма 2.1**

;X =(A•2+B•C)/(D-3)

;prog1

.MODEL SMALL

.STACK 100H

.DATA

;Резервуємо память для змінних

;A,B,C,D,X

A DB ?

B DB ?

C DB ?

D DB ?

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

**Другий рівень**

Виконати завдання 1.

**Третій рівень**

Виконати завдання 1 та 2.

ЗАВДАННЯ 1

1.1. Реалізувати програму мовою асемблера згідно варіанту.

1.2.Завантажте програму в відладчик. Виконайте програму по крокам. При виконанні програми заповніть таблицю:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер  кроку | Команда | AX | BX | CX | DX | CS | IP | DS |
|  |  |  |  |  |  |  |  |  |

1.3. Виконайте програму з 5 варіантами різних початкових значень змінних по крокам та запишіть результат виконання в таблицю (в регістрі AL - частка, AH - залишок). Переведіть результат в десяткову систему.

Варіанти завдань:

|  |  |  |
| --- | --- | --- |
| № | Результат | Формула |
| **1** | S= | A+B/3+C-D |
| **2** | Q= | (A-B\*D)/P |
| **3** | Y= | (A+B)\*(C-D) |
| **4** | J= | (G-R/F)modK |
| **5** | S= | (H/R-F)\*A |
| **6** | J= | D+(F/(G-P)) |
| **7** | M= | (H-U+V)modN |
| **8** | L= | K\*R\*(K-Z) |
| **9** | P= | Y-J/(E-W) |
| **10** | R= | D+A\*(T-C) |
| **11** | N= | J/(E+A)\*G |
| **12** | U= | (A-B-C)modZ |
| **13** | A= | (K/D+E)\*V |
| **14** | B= | S/R\*(G-L) |
| **15** | C= | Q-F/(A\*B) |

ЗАВДАННЯ 2

2.1. Реалізувати програму мовою асемблера згідно варіанту.

2.2.Завантажте програму в відладчик. Виконайте програму по крокам. При виконанні програми заповніть таблицю:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер  кроку | Команда | AX | BX | CX | DX | CS | IP | DS |
|  |  |  |  |  |  |  |  |  |

2.3. Виконайте програму з 5 варіантами різних початкових значень змінних по крокам та запишіть результат виконання в таблицю (в регістрі AL - частка, AH - залишок). Переведіть результат в десяткову систему.

**Варіанти завдань:**

|  |  |  |
| --- | --- | --- |
| № | Результат | Формула |
| **1** | S= | (A+3B+C-D) / 3D+(F/(G-2P)) |
| **2** | Q= | 4D+2A\*(3T-C) + (A-B\*D)/P |
| **3** | Y= | ((G-R/2F)modK )-(2A+B)\*(5C-D) |
| **4** | J= | (6G-5R/3F)modK- K\*R\*(K-Z) |
| **5** | S= | (2H/R-4F)\*A-(K/3D+E)\*2V |
| **6** | J= | 4Q-F/(A\*5B)+ 2D+(F/(4G-P)) |
| **7** | M= | ((H-U+3V)modN) / (A-B-3C)modZ |
| **8** | L= | 2K\*3R\*(4K-5Z)- J/(E+A)\*2G |
| **9** | P= | Y-J/(5E-2W)- D+A\*(4T-C) |
| **10** | R= | D+A\*(4T-C)+ J/(E+A)\*5G |
| **11** | N= | J/(E+A)\*G+(K/D+E)\*V |
| **12** | U= | (A-B/C)modZ+(A\*B-C)modZ |
| **13** | A= | (K/D+E)\*V- S/R\*(G+L) |
| **14** | B= | S/R\*(G-3L)- 2S/R\*(G+L) |
| **15** | C= | 5Q-F/(A\*B)/ S+2R\*(G-L) |

**Контрольні запитання**

1. Команди основних арифметичних операцій.
2. По якому біту регістра флагів можна встановити, що віднімання привело до від’ємного результату?
3. Алгоритми виконання основних арифметичних операцій для двійкових чисел.
4. Алгоритми виконання основних арифметичних операцій для ВСD-чисел.
5. Призначення команди ADD.
6. Призначення команди ADC.
7. Призначення команди SUB.
8. Призначення команди DIV.
9. Призначення команди XOR.
10. Призначення команди NOT.