Лабораторна робота №2
Основи побудови програми на асемблері в архітектурі IA-32

Мета:

Ознайомитися з створенням базової програми виключно на мові асемблер для платформи на архітектурі IA-32

Теоретичні відомості:

Розробка програми мовою асемблера потребує знання про те, на якій архітектурі процесора ця програма має бути реалізована, і яка операційна система стоїть на даній апаратній платформі, а також треба знати особливості конкретного асемблера. Детальні відомості про команди, операції та структуру програм для асемблера та технологію програмування описані у теоретичних посібниках та підручниках, в пунктах, пов'язаних з архітектурними особливостями процесорів IA-32, середовищами програмування для асемблера та макроасемблер MASM 32 та NASM. Навіть для зовсім простих задач асемблерні коди будуть відрізнятися в залежності від того в якій операційній системі реалізована програма.

Візьмемо для прикладу загальновідому тестову програму "Hello, world". Її код мовою С дуже простий:

```
#include <stdio.h>
int main()
{
 printf("Hello, World!");
return 0;
}
```

асемблерні коди для цієї задачі теж досить прості, проте вони різні в тому самому асемблері, в залежності від операційної системи.

Так при реалізації асемблера NASM в середовищі Linux асемблерний код має вигляд:

```
section .data
msg:  db      "Hello, world!", 10
.len:  equ    $ - msg

section .text
global _start
_start:
        mov   eax, 4 ; write
        mov   ebx, 1 ; stdout
        mov   ecx, msg
        mov   edx, msg.len
        int     0x80
        mov   eax, 1 ; exit
        mov   ebx, 0
        int     0x80
```

А при реалізації асемблера NASM в середовищі Win 32 асемблерний код має вигляд:

```
.386
.model flat, stdcall
option casemap :none

EXTERN printf :PROC ;
declare printf

.data
   HelloWorld db "Hello Wolrd", 0
.code
start:
  sub esp, 4
  push offset HelloWorld
  call printf
  add esp, 4
  ret
end start
```

Ще більші відмінності помітимо, якщо реалізуємо код в середовищі MASM-32 при виведенні на консоль.

```
.386
.model flat,stdcall
.stack 4096
   EXTRN ExitProcess@4 : PROC
   EXTRN GetStdHandle@4 : PROC

   EXTRN WriteConsoleA@20 : PROC

.data
   msg BYTE "Hello World",0
   bytesWritten DWORD ?
   .code
main PROC
   push -11
   call GetStdHandle@4
   push 0
   push OFFSET bytesWritten
   push LENGTHOF msg - 1
   push OFFSET msg
   push eax
   call WriteConsoleA@20
   push 0
   call ExitProcess@4
   main ENDP
 END main
```

А намагання вивести 'Hello, world!' у звичне для користувачів вікно для Windows, взагалі породжує код, який з кодом асемблера NASM в середовищі Linux має спільного тільки рядок 'Hello, world!'

```
global _start
extern _MessageBoxA@16
extern _ExitProcess@4

section code use32 class=code
_start:
```

```
        push    dword 0 ; UINT uType = MB_OK
        push    dword title ; LPCSTR lpCaption
        push    dword banner ; LPCSTR lpText
        push    dword 0 ; HWND hWnd = NULL
        call    _MessageBoxA@16

        push    dword 0 ; UINT uExitCode
        call    _ExitProcess@4

section data use32 class=data
        banner:         db 'Hello, world!', 0
        title:  db 'Hello', 0
```

Відтак, для успішного оволодіння навичками написання коду мовою асемблер треба, володіючи загальними принципами програмування, враховувати всі особливості процесора і операційної системи платформи, а також специфіку асемблера в середовищі якого пишеться програма.

Для даної лабораторної роботи середовищем є операційна система Linux,встановлена на архітектуру IA-32, а інструментом є асемблер NASM. Деякі додаткові пояснення стосуються засобів вводу/виводу, оскільки ці дії виконує операційна система. Для того, щоб реалізувати операції вводу/виводу потрібно використовувати системні виклики ядра Linux. Ці системні виклики – бібліотека, вбудована в операційну систему, яка забезпечує такі функції, як читання вводу з клавіатури та вівид на екран.

При виконанні системного виклику, ядро негайно призупинить виконання поточної програми. Потім відбувається процес завантаження необхідних драйверів апаратного забезпечення для виконання завдання, а після закінчення завдання буде повернуто контроль до програми, яка ініціювала системний виклик. Для реалізації системного виклику необхідно завантажити в регістр EAX номер функції, який ми хочемо

виконати, і заповнивши регістри аргументами, які потрібно передати системному виклику. Запит на виконання програмного переривання ініціюється інструкцією INT, ядро викликає функцію з бібліотеки з відповідними аргументами.

Наприклад, запит на переривання, коли EAX = 1 викликає sys_exit, а запит на переривання, коли EAX = 4, замість цього викликає sys_write. EBX, ECX, EDX передаються як аргументи, якщо функція вимагає їх.

Спочатку створюємо змінну ‘msg’ у розділі .data і призначаємо їй рядок, який потрібно вивести (наприклад, ‘I am a string’). У розділі .text повідомляємо ядру з чого починати виконання, надаючи глобальну мітку _start: для позначення точки входу програм.

Будемо використовувати системний виклик sys_write для виведення повідомлення у консоль. Ця функція призначена номером 4 (називается код операції або OPCODE) у таблиці викликів Linux. Функція також бере 3 аргументи, які послідовно завантажуються в EDX, ECX, EBX, перш ніж викликати переривання, яке буде виконувати завдання.

Передані аргументи такі:

EDX завантажується значенням довжини рядка у байтах.

ECX буде завантажений адресою змінної, створенної у розділі .data

EBX буде завантаженний з файлом, в який ми хочемо записати – STDOUT у нашому випадку.

Тип даних та значення переданих аргументів можна знайти у визначенні функції

Компілюємо та запускаємо програму.

Також важливо точно сказати операційній системі, де вона повинна розпочати виконання і де повинна зупинитися. Виклик sys_exit в кінці всіх програм означатиме, що ядро точно знає, коли завершити процес і повернути пам'ять назад до загального пулу, таким чином уникаючи помилки.

sys_exit має просте визначення функції. У таблиці системних викликів Linux йому призначено OPCODE 1 і передається один аргумент через EBX.

Для того, щоб виконати цю функцію, потрібно

Завантажте EBX з 0, щоб передати нуль функції, що означає нульові помилки

Завантажте EAX з 1 для виклику sys_exit

Потім викликати переривання INT 80h, використовуючи бібліотеку libc

Приклад

```
section .data
msg db 'I am a string', 0xa
len equ $-msg

section .text
global _start
_start:
        mov edx, len
        mov ecx, msg
        mov ebx,1
        mov eax,4
        int 0x80
        mov eax,1
```

```
    int 0x80
```

Приклад

```
section .data
msg db ‘I am a string’, 0xa
len equ $-msg

section .text
global _start
_start:
    mov edx, len
    mov ecx, msg
    mov ebx,1
    mov eax,4
    int 0x80
    mov eax,1
    int 0x80
```

Компіляція

```
nasm -f elf lab.asm
ld -m elf_i386 lab.o -o lab
./lab
I am a string
```

Директива EQU використовується для визначення констант. Синтаксис директиви EQU такий

НАЗВА EQU значення

Приклад

```
section .data

number equ 50
…
mov ecx, number
cmp eax, number
```

Операнд EQU може бути виразом

number1 equ 10
number2 equ 10
number3 equ number1 * number2

Ініцалізація змінних робиться наступним чином

[назва-змнінної] директива значення [, значення]

Директива	Призначення	Кількість байтів
DB	Визначає байт	1
DW	Визначає слово	2
DD	Визначає подвійне слово	4
DQ	Визначає quadword	8
DT	Визначає десять байтів	10

нижче наведено кілька прикладів використання визначення змінних

neg_number dw -12345
big_chungus dq 123456789
number dd 1.2345
yes dw ‘y’

Інструкція CMP порівнює два операнди. Зазвичай він використовується в умовному виконанні. Ця інструкція в основному віднімає один опранд від іншого для порівняння того, рівні операнди чи ні.

Синтаксис
CMP destination, source

CMP порівнює два числові поля даних. Операнд призначення може бути або в регістрі, або в пам’яті. Вихідним операндом може уть бути постійні дані, регістри або пам’ять. CMP часто використовується для порівняння того, чи досягло значення лічильника значення, коли потрібно зупинити цикл. Розлянемо наступний приклад

```
INC EDX
CMP EDX,10
JLE LP1
```

Безумовний перехід

Як зазначалося раніше, це виконується інструкцією JMP. Умовне виконання часто передбачє передачу контролю на адресу інтрукції, яка не відповідає виконуваній в даній час інтрукції. Передача контролю може здійснюватися вперед для виконання новго набору інструкцій або назад, для повторного виконання тих самих кроків. Інструкція JMP поребує ім’я мітки, куди потік контролю передається.

Синтаксис інтрукції JMP

JMP label

Приклад

```
mov ax, 00
mov bx, 00
mov cx, 01
L20:
        add ax, 01
        add bx, ax
        shl cx, 1
        jmp L20
```

Умовний перехід

Якщо якась зазначена умова виконується при умовному стрибку, керуючий потік передається цільовий інструкції. Існує багато інструкцій щодо умовного стрибка залежна від стану та даних.

Нижче наведені вказівки умовного переходу, що використовуються для арифметичних операцій

Інструкція	Визначення	Прапорці, які впливають на результат

JE/JZ	Застосувати jump якщо значення рівні/0	ZF
JNE/JNZ	Застосувати jump, якщо значення не рівні/не 0	ZF
JG/JNLE	Застосувати jump, якщо значення більше/не менше або рівне	OF, SF, ZF
JGE/JNL	Застосувати jump, якщо значення більше/рівне або не менше	OF, SF
JL/JNGE	Застосувати jump, якщо значення менше/більше або рівне	OF, SF
JLE/JNG	Застосувати jump, якщо значення менше або рівне/не більше	OF, SF, ZF

Наступні інструкції умовного переходу мають спеціальне використання та перевіряють значення прапорів.

Інструкція	Визначення	Прапорці, які впливають на результат
JXCZ	Застосувати jump якщо значення CX = 0	
JC	Застосувати jump, якщо є значення CARRY	CF
JNC	Застосувати jump, якщо немає значення CARRY	CF
JO	Застосувати jump, якщо є переповнення значення	OF
JNO	Застосувати jump, якщо немає переповнення значення	OF
JP/JPE	Застосувати jump, якщо є парітет біт	PF
JNP/JPO	Застосувати jump, якщо немає парітету біт	PF
JS	Застосувати jump, якщо	SF

	від’ємне значення	
JNS	Застосувати jump, якщо значення не є від’ємним	SF

Підпрограми – це функції, ділянки коду, які можна багаторазово використовувати, і які програми може викликати для виконання різних повторюваних завдань. Підпрограми оголошуються за допомогою міток (наприклад _start:), однак не використовується інструкція JMP, щоб дістатися до них – натомістьвикористовується функція CALL. Для повернення виконанная основної програми використовується функція RET. Стек – особливий тип пам’яті, яки використовує Last In First Out (LIFO). Будь-який регістр, який повинна використовувати функція, повинен мати поточне значення, розміщенне в стеку для безпечного збереження за допомогою інструкції PUSH. Тоді після завершення функції логіки, ці регістри можуть відновити свої початкові значення за допомогою інструкції POP. Це означає, що будь-які значення в регістрах будуть однаковими до і після того, як ви викликали свою функцію.

Зовнішні файли дозволяють перемістити код з програми та помістит його в окремі файли. Цей прийом корисний для написання чистих, простих в обслуговуванні програм. Багаторазові біти коду можна записати як підпрограми та зберігати в окремих файлах, які називаються бібліотеками. Коли буде потрібна частина цієї логіки, можна включити файл у свою програму та використовувати її так, ніби вони є частиною одного файлу

Приклад

%include ‘additional_functions.asm’

SECTION .data

Передати аргументи вашій програмі з командного рядка так само просто, як виштовхувати їх зі стеку в NASM. Коли запускаєте програму, усі передані аргументи завантажуються в стек у зворотньому порядку. Потім ім’я програми завантажується в стек і, нарешті, закальна кількість аргументів завантажується в стек. Отсанніми двома елементами стека для компільованої програми NASM завжди є назва програми та кількість переданих аргументів.

Приклад

%include ‘additional_functions.asm’

```
SECTION .data

msg db ‘I am a string’

SECTION .text

global _start

_start:
        pop ecx
nextArg:
        cmp ecx, 0h
        jz noArgs
        pop eax
        call print_with_linefeed
        dec ecx
        jmp nextArg
noArg:
        call quit
```

Додаток - additional_functions

```
iprint:
   push    eax
```

```
    push    ecx
    push    edx
    push    esi
    mov     ecx, 0

divideLoop:
    inc     ecx
    mov     edx, 0
    mov     esi, 10
    idiv    esi
    add     edx, 48
    push    edx
    cmp     eax, 0
    jnz     divideLoop

printLoop:
    dec     ecx
    mov     eax, esp
    call    print
    pop     eax
    cmp     ecx, 0
    jnz     printLoop

    pop     esi
    pop     edx
    pop     ecx
    pop     eax
    ret

print_number:
    call    iprint
    push    eax
    mov     eax, 0Ah
    push    eax
    mov     eax, esp
    call    print
    pop     eax
    pop     eax
    ret
```

```
slen:
    push ebx
    mov ebx,eax

nextchar:
    cmp byte [eax], 0
    jz  finished
    inc eax
    jmp nextchar

finished:
    sub eax,ebx
    pop ebx
    ret

print:
    push edx
    push ecx
    push ebx
    push eax
    call slen
    mov  edx,eax
    pop  eax
    mov  ecx,eax
    mov  ebx,1
    mov  eax,4
    int  80h
    pop ebx
    pop ecx
    pop edx
    ret

print_with_lifefeed:
    call print
    push eax
    mov  eax, 0Ah
    push eax
    mov  eax,esp
    call print
```

```
    pop  eax
    pop  eax
    ret

quit:
    mov ebx,0
    mov eax,1
    int 80h
    ret
```

Завдання:

1. Встановити на своєму компютері пакет NASM, якщо його не встановлено
2. Ознайомитись з теоретичними положеннями
3. Визначити змінні, занести відповідні значення у регістри та організувати цикл роботи згідно свого варіанту
4. Підготувати звіт для захисту

Варіанти індивідуальних завдань:

Умовні позначення

+	Арифметичне додавання
-	Арифметичне віднімання
*	Арифметичне множення
/	Арифметичне ділення
→	Занести число до регістру або константи
a1,a2,a3,...	Визначити константи як незалежні
A(1), B(2), …	Визначити як елементи масиву
a1&a2	Логічне AND
a1|a2	Логічне OR
(a1)	Логічне NOT
(a1,a2)	Логічне XOR
$a^{\rightarrow n}\ 1$	Логічний зсув a1 на n позицій праворуч
$a^{a\rightarrow n}\ 1$	Арифметичний зсув a1 на n позицій праворуч
$a^{r\rightarrow n}\ 1$	Циклічний зсув a1 на n позицій праворуч
$a^{rc\rightarrow n}\ 1$	Циклічний зсув з переносом a1 на n позицій праворуч

$a^{\leftarrow n}\,1$ Логічний зсув a1 на n позицій ліворуч

$a^{a\leftarrow n}\,1$ Арифметичний зсув a1 на n позицій ліворуч

$a^{r\leftarrow n}\,1$ Циклічний зсув a1 на n позицій ліворуч

$a^{rc\leftarrow n}\,1$ Циклічний зсув з переносом a1 на n позицій ліворуч

Номер Завдання

1	Визначити дані a1 → 10, a2 → 15, b1 → 40, b2 → 25, c1 → 5, c2 → 6 Занести в регістри такі величини AX → a1- a2, BX → b1*b2, CX → c1+c2, DX → $a^{a\leftarrow 1}\,1$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 1. У циклі зменшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, поки значення CX не стане дорівнювати 2
2	Визначити дані a1 → 15H, a2 → 20H, b1 → 4H, b2 → 88H, c1 → 5H, c2 → 6H Занести в регістри такі величини AX → a1+ a2, BX → b2/b1, CX → c1+c2, DX → $a^{rc\leftarrow 2}\,1$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 1. У циклі збільшувати число, що знаходиться у регістрі AX на величину, що знаходиться у регістрі BX, пока значення CX не стане меншим 0
3	Визначити дані a(1) → 1, a(2) → 3, a(3) → 3, a(4) → 5, c1 → 7, c2 → 6 Занести в регістри такі величини AX → (a(1) + a(2))*(a(1) + a(2)), BX → a(4) - a(2), CX → c1+c2, DX →(a(4)| c2) Організувати цикл, послідовно зменшуючи число у регістрі CX на 1. У циклі збільшувати число, що знаходиться у регістрі AX на величину, що знаходиться у регістрі BX, пока значення CX не стане меншим 0
4	Визначити дані a(1) → 8, a(2) → 5, a(3) → 3, c1 → 20, c2 → 6 Занести в регістри такі величини AX → a(1) + a(2) - a(3), BX → a(1) * a(2), CX → c1-c2, DX →((c1&a(2)),a(3)) Організувати цикл, послідовно зменшуючи число у регістрі CX на 1. У циклі зменшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, поки значення CX не стане дорівнювати 0

5	Визначити дані a1 → 11, a2 → 7, a3 → 9, c1 → 25, c2 → 16 Занести в регістри такі величини AX → a1 + a2 - a3, BX → a1 * a2, CX → c1-c2, DX → $c^{rc\to 1}_{\ 1}$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 3. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, у тому випадку, коли значення у регістрі CX – парне число, поки значення CX не стане менше 3.
6	Визначити дані a(1) → 12, a(2) → 6, a(3) → 17, c1 → 23, c2 → 16 Занести в регістри такі величини AX → a(1) + a(3) - a(2), BX → a(1)/a(2), CX → c1+c2, DX → $a^{rc\to 4}_{\ 1}$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 5. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, та зменшувати на величину, що знаходиться у регістрі CX, поки значення CX не стане менше 5.
7	Визначити дані a(1) → 16H, a(2) → 8H, a(3) → 27H, c1 → 2FH, c2 → 1AH Занести в регістри такі величини AX → (a(3) - a(1)) * a(2), BX → a(1) + a(2), CX → c1-c2, DX → $a^{a\leftarrow 2}(3)$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 5H. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, та зменшувати на величину, що знаходиться у регістрі CX, поки значення CX не стане менше -7H.
8	Визначити дані a1 → 11, a2 → 7, a3 → 9, c1 → 25, c2 → 16 Занести в регістри такі величини 8 AX → a1 + a2 - a3, BX → a1 * a2, CX → c1-c2, DX → $c^{rc\to 1}_{\ 1}$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 3. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, у тому випадку, коли значення у регістрі CX – парне число, поки значення CX не стане менше 3.
9	Визначити дані a(1) → 12, a(2) → 17, a(3) → 19, c1 → 15, c2 → 26 Занести в регістри такі величини AX → (a(2) - a(1)) * a(3), BX → a(1) + a(2), CX → c1 + c2, DX → $a^{a\leftarrow 4}(2)$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 3. У

	циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, у тому випадку, коли значення у регістрі CX – непарне число, поки значення CX не стане менше -5.
10	Визначити дані a1 → 17H, a2 → 25H, b1 → 21H, b2 → 3H, c1 → 28H, c2 → 42H Занести в регістри такі величини AX → a1 - a2 , BX → b1 * b2, CX → c2 - c1, DX → ((a1&c2), c2) Організувати цикл, послідовно зменшуючи число у регістрі CX на 3H. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, у тому випадку, коли значення у регістрі CX – парне число, поки значення CX не стане менше 5H.
11	Визначити дані a(1) → 21, a(2) → 8, a(3) → 27, c1 → 25, c2 → 18 Занести в регістри такі величини AX → (a(1) + a(2)) - a(3), BX → (a(3) - a(1))*a(2), CX → c1 + c2, DX → ((a(1)&a(2),a(1)) Організувати цикл, послідовно зменшуючи число у регістрі CX на 5. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, та зменшувати на величину, що знаходиться у регістрі CX, поки значення CX не стане менше 5.
12	Визначити дані a(1) → 8, a(2) → 5, a(3) → 3, c1 → 20, c2 → 6 Занести в регістри такі величини AX → a(1) + a(3) - a(2), BX → a(1)/a(2), CX → c1+c2, DX → $a\ ^{rc\rightarrow 4}_{1}$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 5. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, та зменшувати на величину, що знаходиться у регістрі CX, поки значення CX не стане менше 5.
13	Визначити дані a1 → 17H, a2 → 25H, b1 → 21H, b2 → 3H, c1 → 28H, c2 → 42H Занести в регістри такі величини AX → a1+ a2, BX → b2/b1, CX → c1+c2, DX → $a\ ^{rc\leftarrow 2}_{1}$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 1. У циклі зменшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, поки значення CX не стане дорівнювати 2
14	Визначити дані a(1) → 16H, a(2) → 8H, a(3) → 27H, c1 → 2FH, c2 → 1AH Занести в регістри такі величини AX → (a(1) + a(2)) - a(3), BX → (a(3) - a(1))*a(2), CX → c1 + c2, DX

	→ ((a(1)&a(2),a(1)) Організувати цикл, послідовно зменшуючи число у регістрі CX на 5. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, та зменшувати на величину, що знаходиться у регістрі CX, поки значення CX не стане менше 5.
15	Визначити дані a1 → 10, a2 → 15, b1 → 40, b2 → 25, c1 → 5, c2 → 6 Занести в регістри такі величини AX → a1 + a2 - a3, BX → a1 * a2, CX → c1-c2, DX → $c^{rc\to 1}_{1}$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 1. У циклі збільшувати число, що знаходиться у регістрі AX на величину, що знаходиться у регістрі BX, пока значення CX не стане меншим 0
16	Визначити дані a(1) → 12, a(2) → 6, a(3) → 17, c1 → 23, c2 → 16 Занести в регістри такі величини AX → (a(2) - a(1)) * a(3), BX → a(1) + a(2), CX → c1 + c2, DX → $a^{a\leftarrow 4}(2)$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 5. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, та зменшувати на величину, що знаходиться у регістрі CX, поки значення CX не стане менше 5.
17	Визначити дані a(1) → 1, a(2) → 3, a(3) → 3, a(4) → 5, c1 → 7, c2 → 6 **анести в регістри такі величини** AX → (a(3) - a(1)) * a(2), BX → a(1) + a(2), CX → c1-c2, DX → $a^{a\leftarrow 2}(3)$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 1. У циклі збільшувати число, що знаходиться у регістрі AX на величину, що знаходиться у регістрі BX, пока значення CX не стане меншим 0
18	Визначити дані a1 → 10, a2 → 15, b1 → 40, b2 → 25, c1 → 5, c2 → 6 **Занести в регістри такі величини** AX → a1 - a2 , BX → b1 * b2, CX → c2 - c1, DX → ((a1&c2), c2) Організувати цикл, послідовно зменшуючи число у регістрі CX на 3. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, у тому випадку, коли значення у регістрі CX – непарне число, поки значення CX не стане менше -5.
19	Визначити дані a(1) → 12, a(2) → 17, a(3) → 19, c1 → 15, c2 → 26 Занести в регістри такі величини

	AX → (a(1) + a(2)) - a(3), BX → (a(3) - a(1))*a(2), CX → c1 + c2, DX →((a(1)&a(2),a(1)) Організувати цикл, послідовно зменшуючи число у регістрі CX на 5. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, та зменшувати на величину, що знаходиться у регістрі CX, поки значення CX не стане менше 5.
20	Визначити дані a(1) → 8, a(2) → 5, a(3) → 3, c1 → 20, c2 → 6 Занести в регістри такі величини AX → a1 + a2 - a3, BX → a1 * a2, CX → c1-c2, DX → $c^{rc\to 1}$ 1 Організувати цикл, послідовно зменшуючи число у регістрі CX на 1. У циклі збільшувати число, що знаходиться у регістрі AX на величину, що знаходиться у регістрі BX, пока значення CX не стане меншим 0
21	Визначити дані a(1) → 8, a(2) → 5, a(3) → 3, c1 → 20, c2 → 6 Занести в регістри такі величини AX → (a(2) - a(1)) * a(3), BX → a(1) + a(2), CX → c1 + c2, DX → $a^{a\leftarrow 4}(2)$ Організувати цикл, послідовно зменшуючи число у регістрі CX на 5. У циклі збільшувати число, що знаходиться у регістрі BX на величину, що знаходиться у регістрі AX, та зменшувати на величину, що знаходиться у регістрі CX, поки значення CX не стане менше 5.

Контрольні запитання:

1. Поясніть структуру програми у асемблері

2. Особливості умовного переходу

3. Як працюють прапорці ?

4. В чому особливість представленням від’ємного значення ?

5. Які призначення основних регістрів ?

6. Особливості реалізації циклу у мові асемблер

7. Особливість структури IA-32

8. Що таке циклічний зсув праворуч на N позицій ?

9. В чому особливість зсуву з використанням кері ?

10. Що таке логічний зсув ліворуч ? ?

11. Поясніть як працює опкод CMP

12. Поясніть особливості типів даних у мові асемблер