



НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

Курс: «АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ
СИСТЕМ» (АВС)

ДОМАШНЕЕ ЗАДАНИЕ № 1

Выполнила : Резуник Людмила
Группа БПИ198

<https://github.com/LucyRez/ABC/tree/master/HW1>



ПРОГРАММА №1

«hello.ASM»

flat assembler 1.73.25

```
File Edit Search Run Options Help
format PE console
entry Start
include 'win32a.inc'

section '.data' data readable writable

    nameStr db '%s', 0
    ageStr db '%d', 0

    name rd 2
    age rd 1

    question1 db 'What is your name?', 0
    question2 db 'How old are you?', 0
    hello db 'Hello %s! You are %d years old.', 0

    NULL = 0
```

; указываем, что приложение консольное,
обозначаем точку входа и подключаем
win32a.inc

На данном слайде представлена секция с данными, необходимыми для исполнения программы, а именно:

- Две строки форматирования, с метками nameStr и ageStr
- Метки name и age резервируют данные, для ввода с клавиатуры
- Две строки с вопросами пользователю
- Строка для вывода информации о пользователе на консоль



```
section '.code' code readable executable
```

```
Start:
```

```
    push question1  
    call [printf]
```

```
    push name  
    push nameStr  
    call [scanf]
```

```
    push question2  
    call [printf]
```

```
    push age  
    push ageStr  
    call [scanf]
```

```
    push [age]  
    push name  
    push hello  
    call [printf]
```

```
    call [getch]
```

Заносим в стек строку, на которую указывает метка question1 и выводим ее на консоль при помощи функции printf.

Далее, заносим в стек резервирующую метку name, и строку форматирования. Для выполнения функции scanf нужна строка форматирования, и оператор, куда будут записываться данные с клавиатуры.

После этого, таким же образом запрашиваем возраст пользователя и обрабатываем его ввод.

В завершении, отправляем все полученные данные в стек в обратном порядке, а так же строку, в которую эти данные будут подставляться.

Вызываем функцию printf для вывода информации и getch для того, чтобы окно консоли не закрылось сразу же.



```
section '.idata' import data readable  
  
library kernel, 'kernel32.dll',\  
           msvcrt, 'msvcrt.dll'  
  
import kernel,\  
           ExitProcess, 'ExitProcess'  
  
import msvcrt,\  
           printf, 'printf',\  
           scanf, 'scanf',\  
           getch, '_getch'
```


В секции импорта подключаем все нужные библиотеки и импортируем необходимые функции из них.

Во всех последующих программах секция импорта выглядит аналогично.



ПРОГРАММА №2

«sum.ASM»

 flat assembler 1.73.25

File Edit Search Run Options Help

entry Start

include 'win32a.inc'

section '.data' data readable writable

resStr db 'Sum of 50 and -5 : %d', 0

A dd ?

B dd ?

NULL = 0

; указываем, что приложение консольное,
обозначаем точку входа и подключаем
win32a.inc

На данном слайде представлена секция с данными, необходимыми для исполнения программы, а именно:

- Две переменные A и B необходимые для выполнения арифметических действий
- Строка для вывода результата



```
section '.code' code readable executable
```

```
    Start:
```

```
        mov [A], 50
```

```
        mov [B], -5
```

```
        mov ecx, [A]
```

```
        add ecx, [B]
```

```
        push ecx
```

```
        push resStr
```

```
        call [printf]
```

```
        call [getch]
```

```
        push NULL
```

```
        call [ExitProcess]
```

Заносим в переменную А и В данные. Чтобы выполнить операцию сложения нужно в регистр есх занести значения, которые находятся в ячейке памяти, на которую указывает А. Далее прибавляем данные из В. Полученная сумма будет храниться в регистре есх.

Заносим информацию из есх в стек, а далее и строку для подстановки этого числа. Вызываем функцию printf для вывода результата и getch для того, чтобы окно консоли не закрылось сразу же.



ПРОГРАММА №3

«function_example.ASM»

flat assembler 1.73.25

File Edit Search Run Options Help

_format PE console

include 'win32a.inc'

; указываем, что приложение консольное,
переходим на метку Start и подключаем win32a.inc

jmp Start

section '.data' data readable writable

resStr db 'Composition of -5 and 9 : %d', 0

A dd ?

B dd ?

NULL = 0

На данном слайде представлена секция с данными, необходимыми для исполнения программы, а именно:

- Две переменные A и B для выполнения арифметических действий
- Строка для вывода результата



```
function:
    imul ecx, ebx ;multiplication
ret
Start:
    mov ecx, -5 ;we put numbers in ecx and ebx
    mov ebx, 9

    call function

    push ecx
    push resStr
    call [printf]

exit:
    call [getch]

    push NULL
    call [ExitProcess]
```

Чтобы выполнить операцию умножения нужно в регистр ecx занести первое значение, а в ebx второе. Вызываем простую функцию для умножения.

Результат умножения будет находиться в регистре ecx.

Команда ret позволяет нам вернуться в то же самое место, откуда мы запустили функцию изначально, без использования дополнительных меток.

Осталось внести в стек значение из регистра, а так же строку для вывода результата и вызвать printf. Вызываем getch для того, чтобы окно консоли не закрылось сразу же.



ПРОГРАММА №4

«cycle_shift.ASM»

 flat assembler 1.73.25

```
File Edit Search Run Options Help
format PE console

entry Start                ; указываем, что приложение консольное,
                           ; переходим на метку Start и подключаем win32a.inc

include 'win32a.inc'

section '.data' data readable writable

    resStr db 'Result for 200: %d', 0
    A dw ?
    B dw ?

    NULL = 0
```

На данном слайде представлена секция с данными, необходимыми для исполнения программы, а именно:

- Две переменные A и B для выполнения арифметических действий (в этом случае они не нужны)
- Строка для вывода результата



```
section '.code' code readable executable
```

```
Start:
```

```
    xor eax, eax
    mov al, 11001000b
    rol al, 1
```

```
    push eax
    push resStr
    call [printf]
```

```
    call [getch]
```

```
    push NULL
    call [ExitProcess]
```

Сначала обнуляем регистр, с которым мы будем работать при помощи xor. Заносим в al число, записанное в двоичной системе счисления. Далее, производим циклический сдвиг влево на одну позицию при помощи команды rol.

Осталось внести в стек значение из регистра eax, а так же строку для вывода результата и вызвать printf. Вызываем getch для того, чтобы окно консоли не закрылось сразу же.



ПРОГРАММА №5

«is_prime.ASM»

```
section '.data' data readable
    szEnterNum db 'Please enter a number: ',0
    szNumFormat db '%d',0
    szPrime db '%d is prime',0
    szNotPrime db '%d is not prime',0

section '.bss' data readable writeable
    ddNum rd 1
    ddSqrt rd 1
```

На данном слайде представлена секция с данными, необходимыми для исполнения программы, а именно:

- Строка, запрашивающая у пользователя ввод числа
- Строка форматирования для введённого числа
- Две строки для вывода одного из результатов
- Две зарезервированные метки



```
section '.text' code readable executable
```

```
start:
```

```
push szEnterNum
call [printf]
add esp, 4
```

```
push ddNum
push szNumFormat
call [scanf]
add esp, 8
```

```
; Handle special cases "1", "2", "3"
```

```
cmp [ddNum], 1
je notPrime
cmp [ddNum], 2
je isPrime
cmp [ddNum], 3
je isPrime
```

```
; Check if number is even
```

```
clc ; Clear the carry flag
mov ebx, [ddNum]
shr ebx, 1
jb checkIfPrime ; Jump if carry flag is set -> number is odd
jmp notPrime ; Number is even -> not a prime
```

```
checkIfPrime:
```

```
fild [ddNum] ; Integer Load from memory -> int to float
fsqrt ; Compute sqrt of value in ST(0) and store result in ST(0)
fist [ddSqrt] ; Store integer number to memory
```

```
mov ebx, 3
```

```
loopPrime:
```

```
mov eax, [ddNum]
cdq ; Convert signed value in EAX in signed value in EDX:EAX
idiv ebx ; Signed division divides 64 bit value in EDX:EAX through the given operand. Result in EAX, remainder in EDX
cmp edx, 0 ; Check if the remainder is 0
je notPrime ; Not a prime if the remainder is 0
cmp ebx, [ddSqrt]
jg isPrime ; If ebx is bigger than the sqrt, stop the loop. The number is prime,
inc ebx
jmp loopPrime
```

Сначала заносим в стек вопрос и выводим его на консоль. Далее считываем значения с консоли при помощи scanf (как в программе №1) и записываем его в ранее зарезервированное место.

После, обрабатываем особые случаи для 1, 2 и 3, и если нужно отправляем на метку isPrime или notPrime.

Далее, проверка на чётность. Если число нечетное, проверяем дальше при помощи цикла. В результате, если число, находящееся в регистре окажется больше квадратного корня, то это число простое.



НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

P.S. Все скриншоты консолей выложены на GitHub

<https://github.com/LucyRez/ABC/tree/master/HW1>