

МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Отчёт о практике

студента 2 курса 251 группы  
направления 09.03.04 — Программная инженерия  
факультета КНиИТ  
Толстова Роберта Сергеевича

Проверено:

Старший преподаватель

\_\_\_\_\_

Е. М. Черноусова

Саратов 2024

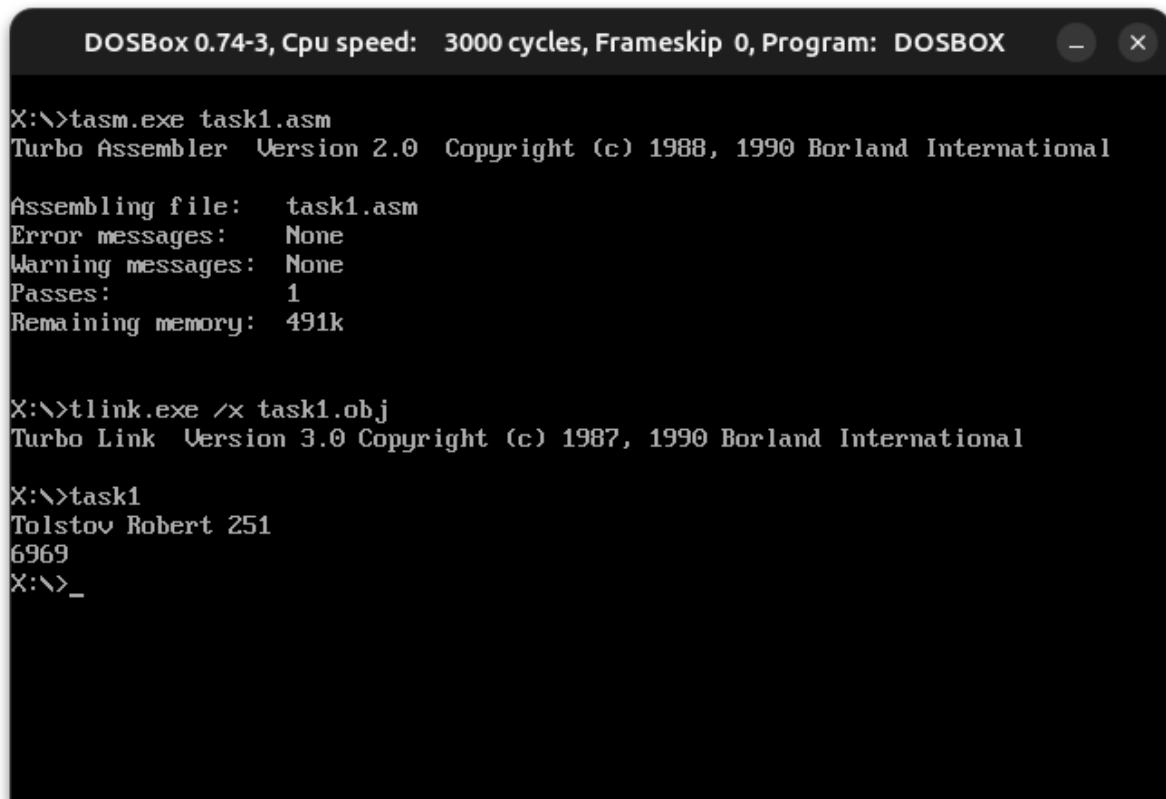
## СОДЕРЖАНИЕ

1 Исходный код программ .....	2
1.1 Задание 1 .....	2
1.2 Задание 2 .....	4
2 Ответы на контрольные вопросы .....	6
2.1 Чем отличается деление на байт от деления на слово? (где должно располагаться делимое, куда попадут частное от деления и остаток от деления) .....	6
2.2 Каков механизм действия команды CMP? .....	6
2.3 На какие флаги реагируют команды условного перехода для чисел со знаком и для чисел без знака? .....	6
2.4 С помощью команд условного и безусловного перехода выполните программную реализацию алгоритма ветвления для определения наименьшего числа из двух заданных. ....	6
2.5 Каков механизм работы команды организации цикла LOOP? .....	7
2.6 Как с помощью команды сдвига можно умножить знаковое число, хранящееся в AX, на 2 в n-ой степени? .....	8
2.7 Как с помощью команды сдвига проверить содержимое регистра BX на четность? .....	8

# 1 Исходный код программ

## 1.1 Задание 1

В регистре AX задано число от 0 до 65535. Выведите это число на экран.  
(Проверить программу для числа более 2600)



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

X:\>tasm.exe task1.asm
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland International

Assembling file: task1.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 491k

X:\>tlink.exe /x task1.obj
Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland International

X:\>task1
Tolstov Robert 251
6969
X:\>_
```

Фото запуска первой программы

```
.model small
.stack 100h

.data
Names db 'Tolstov Robert 251', 0Dh, 0Ah, '$'

.code

start:
mov AX, @data ;помещение указателя на сегмент данных в AX
mov DS, AX ;помещение указателя на сегмент данных в DS

;адрес начала строки
mov DX, offset Names
mov ah, 09h
```

```

int 21h

mov AX, 6969 ; занесение числа
mov BX, 10 ; занесение делителя
mov CX, 0 ; обнуляем счётчик

loop_first: ; заносим в стек цифры
inc cx ; увеличение счётчика
mov dx, 0 ; очищение значения DX
div bx ; деление AX на BX
push dx ; занесение остатка от деления в стек
cmp ax, 0 ; сравнение частного с нулем
jne loop_first ; если AX != 0, то возвращаемся к loop_first

mov ah, 02h

loop_second: ; достаём из стека
pop dx
call cout
loop loop_second

mov ax, 4C00h ; завершаем программу
int 21h

cout proc; вывод на экран одной цифры
add dx, 30h ; с помощью табл ASCII получаем цифру
int 21h ; возврат к основной программе
ret конец процедуры
cout endp

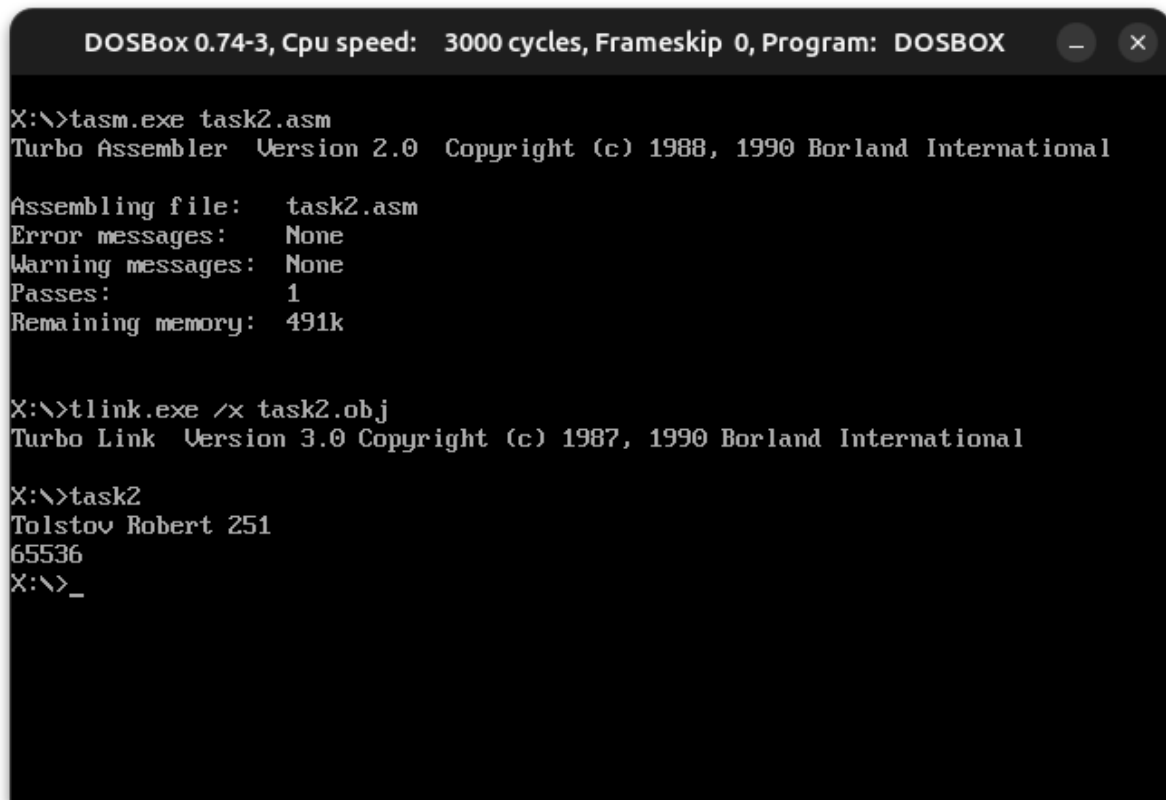
end start

```

Код первой программы

## 1.2 Задание 2

Используя 32-битные регистры процессора (EAX, EBX, EDX), напишите программу, выводящую на экран число 65536. Число 65536 изначально поместить в регистр EAX.



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

X:\>tasm.exe task2.asm
Turbo Assembler Version 2.0 Copyright (c) 1988, 1990 Borland International

Assembling file: task2.asm
Error messages: None
Warning messages: None
Passes: 1
Remaining memory: 491k

X:\>tlink.exe /x task2.obj
Turbo Link Version 3.0 Copyright (c) 1987, 1990 Borland International

X:\>task2
Tolstov Robert 251
65536
X:\>_
```

Фото запуска первой программы

```
.model small
.stack 100h

.data
Names db 'Tolstov Robert 251', 0Dh, 0Ah, '$'

.386 ; разрешаем транслировать команды процессорам 386
.code

start:
mov AX, @data ;помещение указателя на сегмент данных в AX
mov DS, AX ;помещение указателя на сегмент данных в DS

mov DX, offset Names ;помещение ссылки на строку с именем в dx
mov AH, 09h ;код команды прерывания для вывода строки
int 21h ;команда прерывания
```

```

mov EAX, 65536 ;занесение в AX числа 65536
mov EBX, 10 ;занесение в BX делителя
mov CX, 0;обнуление счётчика

loop_first: ; заносим в стек
inc CX ;увеличение счётчика
mov EDX, 0 ;очищение значения DX
div EBX ;деление AX на BX
push EDX ;занесение остатка от деления в стек
cmp EAX, 0 ;сравнение частного с нулем
jne loop_first ;если AX != 0, то возвращаемся к loop_first

mov AH, 02h ;код прерывания для вывода единичного вывода

loop_second: ;2я метка, дост содерж стека + cout
pop EDX ;команда прерывания для зачисления прог-мы
call cout ;команда прерывания
loop loop_second

mov EAX, 4C00h
int 21h

cout proc ;вывод на экран одной цифры
add EDX, 30h ;с помощью табл ASCII получаем цифру
int 21h ;возврат к основной программе
ret ;конец процедуры
cout endp

end start

```

Код второй программы

## **2 Ответы на контрольные вопросы**

### **2.1 Чем отличается деление на байт от деления на слово? (где должно располагаться делимое, куда попадут частное от деления и остаток от деления)**

- Деление на байт: Делимое должно располагаться в регистре AX (для 16-битных операций) или AL (для 8-битных операций). Частное от деления помещается в AL, а остаток — в AH. Например, если мы делим 8-битное число, то результат будет в AL, а остаток в AH.
- Деление на слово: Делимое должно располагаться в регистре DX:AX (для 32-битных операций) или в AX (для 16-битных операций). Частное помещается в AX, а остаток — в DX. Это позволяет делить большие числа, так как регистр DX используется для хранения остатка от деления.

### **2.2 Каков механизм действия команды CMP?**

Команда CMP выполняет сравнение двух операндов путем вычитания одного из другого. Результаты вычитания не сохраняются, но устанавливаются соответствующие флаги в регистре EFLAGS:

- ZF (Zero Flag): устанавливается, если операнды равны.
- CF (Carry Flag): устанавливается, если первый операнд меньше второго (для беззнаковых чисел).
- SF (Sign Flag): устанавливается, если результат отрицательный (для знаковых чисел).
- OF (Overflow Flag): устанавливается при переполнении знакового результата.

Часто команда CMP используется перед условными переходами (JE, JNE, JG, JL и т.д.), которые принимают решения на основе установленных флагов.

### **2.3 На какие флаги реагируют команды условного перехода для чисел со знаком и для чисел без знака?**

Для чисел со знаком:

- JG (Jump if Greater):  $SF = OF$  и  $ZF = 0$
- JL (Jump if Less):  $SF \neq OF$
- JE или JZ (Jump if Equal):  $ZF = 1$

Для чисел без знака:

- JA (Jump if Above): CF = 0 и ZF = 0
- JB (Jump if Below): CF = 1
- JE или JZ: ZF = 1

**2.4 С помощью команд условного и безусловного перехода выполните программную реализацию алгоритма ветвления для определения наименьшего числа из двух заданных.**

```
.model small
.stack 100h

.data
    R1 db ?          ; Первое число
    R2 db ?          ; Второе число
    result db ?      ; Наименьшее число

.code
main proc
    mov ax, @data
    mov ds, ax

    ; Задаем значения R1 и R2
    mov R1, 5        ; Пример первого числа
    mov R2, 3        ; Пример второго числа

    mov al, R1        ; Загружаем первое число в AL
    cmp al, R2        ; Сравниваем с вторым числом
    jle R2_is_less    ; Если R1 <= R2, переход к метке

    ; Если R1 больше R2
    mov result, R1    ; Сохраняем R1 как результат
    jmp end          ; Переход к завершению

R2_is_less:
    mov result, R2    ; Сохраняем R2 как результат

end:
    ; Завершение программы
    mov ax, 4C00h
    int 21h

main endp
end main
```



## 2.5 Каков механизм работы команды организации цикла LOOP?

Команда LOOP уменьшает значение регистра CX на единицу и выполняет переход по указанной метке, если CX не равен нулю. Это позволяет организовать циклы с фиксированным количеством итераций. Пример:

```
mov cx, 5          ; Устанавливаем количество итераций
loop_start:
    ; Ваш код здесь
    loop loop_start; Переход к loop_start, пока CX не станет равным
нулю.
```

## 2.6 Как с помощью команды сдвига можно умножить знаковое число, хранящееся в AX, на 2 в n-ой степени?

Чтобы умножить число в регистре AX на  $2^n$ , можно использовать команду сдвига влево (SHL). Например:

```
mov ax, 3          ; Пример числа (3)
shl ax, 1          ; Умножаем на 2 (3 * 2 = 6)
shl ax, 2          ; Умножаем на 4 (6 * 4 = 24)
```

Каждый сдвиг влево увеличивает степень двойки.

## 2.7 Как с помощью команды сдвига проверить содержимое регистра BX на четность?

Чтобы проверить четность числа в регистре BX, можно использовать команду сдвига вправо (SHR) и проверять младший бит:

```
mov bx, some_value ; Значение для проверки четности
shr bx, 1           ; Сдвигаем вправо на один бит
jnc is_even         ; Если нет переноса, число четное
```

```
is_odd:
; Код для обработки нечетного числа
```

```
is_even:
; Код для обработки четного числа
```

Если после сдвига младший бит равен нулю (нет переноса), то число четное.