


**Федеральное государственное бюджетное образовательное  
учреждение высшего образования "Белгородский государственный  
технологический университет им. В.Г. Шухова"**

Кафедра программного обеспечения вычислительной техники и  
автоматизированных систем.

**Лабораторная работа № 6**  
Логические команды и команды сдвига.  
Вариант 13

Выполнил:  
Студент группы КБ-211

  
\_\_\_\_\_ Коренев Д.Н.

Принял:

\_\_\_\_\_ Осипов О.В.

*Цель работы:* изучение команд поразрядной обработки данных.

### Задание

1. Написать программу для вывода чисел на экран согласно варианту задания. При выполнении задания №1 все числа считать беззнаковыми. Написать и использовать функцию `output(a)` для вывода числа `a` на экран или в файл. Функция должна удовлетворять соглашению о вызовах. В функцию для вывода `output` передавать в качестве аргумента переменную размерности 32 или 64 бита, которой достаточно для хранения числа. К примеру, если в задании число указано как 15-разрядное, то аргументом функции должно быть число размером двойное слово, если 40-разрядное, то учетверённое слово. Функция должна выводить столько разрядов числа, сколько указано в задании, даже если старшие разряды равны нулю. Не допускается прямой перебор всех чисел с проверкой, удовлетворяет ли оно условию вывода (за исключением вариантов № 8, 12, 13). Числа выводить в порядке, который является удобным. Проверить количество выведенных чисел с помощью формул комбинаторики. В отчёт включить вывод формул и результаты работы программы.

Вариант	Задание №1	Задание №2
13	Вывести все 18-разрядные числа, сумма цифр которых в восьмеричном представлении равна 4. 1: 001111 2: 000112 3: 000022 ...	35 байт умножение со знаком

		103:	00120100
1:	00000004	104:	00121000
2:	00000013	105:	00130000
3:	00000022	106:	00200002
4:	00000031	107:	00200011
5:	00000040	108:	00200020
6:	00000103	110:	00200110
7:	00000112	111:	00200200
8:	00000121	112:	00201001
9:	00000130	113:	00201010
10:	00000202	114:	00201100
11:	00000211	115:	00202000
12:	00000220	116:	00210001
13:	00000301	117:	00210010
14:	00000310	118:	00210100
15:	00000400	119:	00211000
16:	00001003	120:	00220000
17:	00001012	121:	00300001
18:	00001021	122:	00300010
19:	00001030	123:	00300100
20:	00001102	124:	00301000
21:	00001111	125:	00310000
22:	00001120	126:	00400000

```

; =====
; bool check_sum_oct_eq4(uint32_t a)
; Проверка числа на соответствие условию
; =====
check_sum_oct_eq4 proc
    PUSH EBX
    PUSH ECX
    PUSH EDX
    PUSH EDI

    MOV EAX, [ESP+5*4]    ; Взять из стека аргумент 18-битное число (32)
    MOV EDI, 6            ; Количество итераций (18/3 разрядов числа)
    XOR EDX, EDX          ; Сумма цифр
loop_check:
    MOV EBX, EAX          ; EBX = EAX
    AND EBX, 0111b        ; Оставить только младшие 3 бита, остальные обнулить
    ADD EDX, EBX          ; Сложить сумму цифр и младшие биты числа
    JMP continue_check    ; Если EBX == 0F, то продолжить проверку

increment_counter:
    INC EDX                ; Увеличить счетчик
    JMP continue_check

continue_check:
    SHR EAX, 3            ; Сдвиг на 3 бит вправо
    dec EDI                ; для выполнения цикла, содержащего 18 итераций

```

```

    cmp EDI, 0
    jne loop_check

    CMP EDX, 4          ; Если сумма цифр равна 4, то возвращаем 1
    je ret1
    JNE ret0

ret0:
    MOV EAX, 0
    JMP end_check

ret1:
    MOV EAX, 1
    JMP end_check

end_check:              ; вернуть стек в исходное состояние
    POP EDI
    POP EDX
    POP ECX
    POP EBX
    RET 4
check_sum_oct_eq4 endp

```

```

start:
.data
    __main__18bit_number_fmt db "%u:", 9, "0o%06o", 13, 10, 0
.code

    mov EDI, 0
    mov EBX, 0 ; Счетчик
main_loop:
    push EDI
    call check_sum_oct_eq4

    cmp EAX, 1
    jne skip_print

    inc EBX
    push EDI
    push EBX
    push offset __main__18bit_number_fmt
    call crt_printf
    add esp, 8

skip_print:

    inc EDI
    cmp EDI, 1111111111111111b
    JL main_loop
    invoke ExitProcess, 0          ; Выход из программы

```

$$C_n^k = \frac{(n+k-1)!}{k!(n-1)!}$$

$$C_6^4 = \frac{(6+4-1)!}{4!(6-1)!} = \frac{9!}{4!5!} = \frac{362880}{24 \times 120} = 126$$

```
multiplication(char* a, int n, char* res);
```

division(char\* a, int n, char\* res).

[illegible]

Korenev lab6 ♥ 02:50

```
; =====  
; void multiplication (void* a, uint32_t n, void* res)  
; Умножение 35-байтного числа на 2^n, записывая результат в 70-байтное число res  
; =====  
multiplication proc  
    PUSH EAX  
    PUSH EBX  
    PUSH ECX  
    PUSH EDX
```

```

MOV EBX, 70                ; EBX = 70 (размер нового массива)

; (!) Заполним новый массив res нулями
MOV ECX, 0                 ; ECX = 0
loop_mult_zero:
MOV EAX, [ESP+5*4+8]       ; EAX = &res
ADD EAX, ECX               ; EAX = &res + ECX
MOV BYTE PTR [EAX], 0      ; *(&res + ECX) = 0

INC ECX                   ; ECX += 1
CMP ECX, EBX
JL loop_mult_zero         ; Цикл пока ECX < EBX

; (!) Заполним новый массив res значениями из a,
; сдвинутый на n / 8 байтов влево, сдвинутый побитово на n % 8
влево
MOV EAX, [ESP+5*4+4]       ; EAX = n
CDQ
MOV EBX, 8
IDIV EBX                  ; EAX = n / 8, EDX = n % 8
MOV CL, DL

MOV EDX, 35-1
ADD EDX, [ESP+5*4+0]       ; EDX = &a+35-1
MOV EBX, 70-1
SUB EBX, EAX               ; EBX = 70-1-EAX
ADD EBX, [ESP+5*4+8]       ; EBX = &res+70-1-(n/8)

MOV AL, 0                 ; AL = 0
loop_mult:
MOV AH, BYTE PTR [EDX]     ; AH = a[i]
SHL AX, CL                ; AH:AL = a[i] <=< CL
MOV BYTE PTR [EBX], AH     ; res[i] = AH

MOV AL, BYTE PTR [EDX]     ; AL = a[i]
DEC EDX                   ; EDX -= 1 (сдвигаем указатели на массивы влево, к их
началу)
DEC EBX                   ; EBX -= 1
CMP EDX, [ESP+5*4+0]
JGE loop_mult             ; Цикл пока EDX != &a+0 (пока указатель на масс. а не
дойдет до начала)

MOV AH, 0                 ; AH = 0
SHL AX, CL                ; AX = 00:CL <=< CL
MOV BYTE PTR [EBX], AH     ; res[i] = AH

POP EDX
POP ECX
POP EBX

```

```

    POP EAX
    RET 12
multiplication endp

; =====
; void print_hex (void* a, uint32_t n);
; Вывод числа a в 16-чном виде размера n байт
; =====
print_hex_memory proc
.data
    __print_hex_memory__start_hex_fmt db "0x:" , 0
    __print_hex_memory__single_hex_fmt db "%02X:" , 0
    __print_hex_memory__single_hex_last_fmt db "%02X" , 0
    __print_hex_memory__new_line_fmt db 13, 10, 0
.code
    PUSH EAX
    PUSH ECX
    PUSH EDX
    PUSH EBX

    invoke crt_printf, offset __print_hex_memory__start_hex_fmt

    XOR EDX, EDX
    MOV ECX, 0
loop_print_hex_memory:
    MOV EAX, DWORD PTR [ESP+5*4] ; EAX = &a
    MOV DL, [EAX+ECX]           ; EAX = a[EAX]

    PUSH ECX
    PUSH EDX ; Выложить байт числа на стек
    PUSH offset __print_hex_memory__single_hex_fmt
    CALL crt_printf
    ADD ESP, 8
    POP ECX

    INC ECX ; ECX += 1
    MOV EBX, [ESP+5*4+4]
    DEC EBX
    CMP ECX, EBX ; сравнить ECX и n-1
    JB loop_print_hex_memory ; Цикл пока ECX < n

    MOV EAX, DWORD PTR [ESP+5*4] ; EAX = &a
    MOV DL, [EAX+ECX]           ; EAX = a[EAX]

    PUSH ECX
    PUSH EDX ; Выложить байт числа на стек
    PUSH offset __print_hex_memory__single_hex_last_fmt
    CALL crt_printf
    ADD ESP, 8
    POP ECX

```

```

    PUSH offset __print_hex_memory__new_line_fmt
    CALL crt_printf ; вывод перехода к новой строке
    ADD ESP, 4

    POP EBX
    POP EDX
    POP ECX
    POP EAX
    RET 8
print_hex_memory endp

```

```

start:
.data
    x DB 35 DUP (0FAh)
    res DB 70 DUP (?)
.code
    push offset res
    push 2
    push offset x
    call multiplication

    push 70
    push offset res
    call print_hex_memory

    invoke ExitProcess, 0 ; Выход из программы
end start

```

**Вывод:** в ходе данной лабораторной работы мы изучили команды поразрядной обработки данных