МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №6

по дисциплине: Архитектура вычислительных систем
тема: «Логические команды и команды сдвига»

Выполнил: ст. группы ВТ-221
Беляков Генрих Сергеевич

Проверили:
ст. пр. Осипов Олег Васильевич

Белгород 2024 г.

Лабораторная работа №6
Логические команды и команды сдвига
Вариант 3

Цель работы: изучение команд поразрядной обработки данных.

Задания для выполнения к работе:

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.
2. Написать подпрограмму для умножения (multiplication) или деления (division) большого целого числа на 2n (в зависимости от варианта задания) с использованием команд сдвига. Подпрограммы должны иметь следующие заголовки:

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

Входные параметры: `a` – адрес первого числа в памяти, `n` – степень двойки. Выходные параметры: res – адрес массива, куда записывается результат. В случае операции умножения, для массива res зарезервировать в два раза больше памяти, чем для множителей a и b. Числа a, b, res вывести на экран в 16-ричном виде. Подобрать набор тестовых данных для проверки правильности работы подпрограммы.

Задание:

3	Вывести все 20-разрядные числа, в 16-ричном представлении которых есть одна цифра **"1"**, остальные – **"E"** или **"F"**. 1: EEEE1 2: EEEF1 3: EEFE1 …	18 байт умножение без знака

1 задание:

Рассмотрим числа с единицей в конкретной позиции. В остальных числах будет использоваться либо цифра F либо цифра E. При заданном условии возможно 2^4 вариантов. Возможных позиций единицы 5, поэтому в сумме получим $5 * 2^4 = 80$ чисел.

Программа:

```
.686
.model flat, stdcall
option casemap: none

include windows.inc
include kernel32.inc
include msvcrt.inc
includelib  kernel32.lib
includelib  msvcrt.lib

.data
    print_number db "%d: ", 0
    print_digit db "%01X", 0
    print_newline db 13, 10, 0
.code

start:
    ; Перебор позиций единицы
    mov ecx, 0
    ; Общий счётчик при выводе
    mov esi, 1

main_loop:
    ; У нас 5 цифр
    cmp ecx, 5
    jge main_loop_end

    ; Маска для остального числа
    mov ebx, 0
subloop:
        cmp ebx, 10000b
        jge subloop_end

        pushad
        push esi
        push offset print_number
        call crt_printf
        add esp, 8
        popad

        inc esi

        ; Копируем маску в eax
        mov eax, ebx
        ; Внутренний счётчик i = 0
        mov edx, 0
subsubloop:
            cmp edx, 5
            jge subsubloop_cycle_end

            ; i == позиции единицы?
            cmp edx, ecx

            je subsubloop_eq
            jmp subsubloop_not_eq
subsubloop_eq:
                pushad
```

```
                push 1
                push offset print_digit
                call crt_printf
                add esp, 8
                popad
                jmp subsubloop_end
subsubloop_not_eq:
                mov edi, eax
                and edi, 1
                shr eax, 1

                cmp edi, 1
                je subsubloop_print_e
                jmp subsubloop_print_f
subsubloop_print_e:
                pushad
                push 0Eh
                push offset print_digit
                call crt_printf
                add esp, 8
                popad
                jmp subsubloop_print_end
subsubloop_print_f:
                pushad
                push 0Fh
                push offset print_digit
                call crt_printf
                add esp, 8
                popad
subsubloop_print_end:

subsubloop_end:
            inc edx
            jmp subsubloop

subsubloop_cycle_end:

        pushad
        push offset print_newline
        call crt_printf
        add esp, 4
        popad

        inc ebx
        jmp subloop
subloop_end:

    inc ecx
    jmp main_loop
main_loop_end:

    call crt__getch     ; Задержка ввода, getch()
    ; Вызов функции ExitProcess(0)
    push 0      ; Поместить аргумент функции в стек
    call ExitProcess    ; Выход из программы
end start
```

Результат выполнения программы:

```
1: 1FFFF
2: 1EFFF
3: 1FEFF
4: 1EEFF
5: 1FFEF
6: 1EFEF
7: 1FEEF
```

8: 1EEEF
9: 1FFFE
10: 1EFFE
11: 1FEFE
12: 1EEFE
13: 1FFEE
14: 1EFEE
15: 1FEEE
16: 1EEEE
17: F1FFF
18: E1FFF
19: F1EFF
20: E1EFF
21: F1FEF
22: E1FEF
23: F1EEF
24: E1EEF
25: F1FFE
26: E1FFE
27: F1EFE
28: E1EFE
29: F1FEE
30: E1FEE
31: F1EEE
32: E1EEE
33: FF1FF
34: EF1FF
35: FE1FF
36: EE1FF
37: FF1EF
38: EF1EF
39: FE1EF
40: EE1EF
41: FF1FE
42: EF1FE
43: FE1FE
44: EE1FE
45: FF1EE
46: EF1EE
47: FE1EE
48: EE1EE
49: FFF1F
50: EFF1F
51: FEF1F
52: EEF1F
53: FFE1F
54: EFE1F
55: FEE1F
56: EEE1F
57: FFF1E
58: EFF1E
59: FEF1E
60: EEF1E
61: FFE1E
62: EFE1E
63: FEE1E
64: EEE1E
65: FFFF1
66: EFFF1
67: FEFF1
68: EEFF1
69: FFEF1
70: EFEF1
71: FEEF1
72: EEEF1
73: FFFE1
74: EFFE1

```
75: FEFE1
76: EEFE1
77: FFEE1
78: EFEE1
79: FEEE1
80: EEEE1
```

Получили 80 строчек, наши вычисления совпали с результатом выполнения программы. Программа корректна.

Вторая программа:

```
.686
.model flat, stdcall
option casemap: none

include windows.inc
include kernel32.inc
include msvcrt.inc
includelib  kernel32.lib
includelib  msvcrt.lib

.data
    value db 0h, 0h, 0h, 0h, 0h, 0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h
    n dd 4
    res db 18 dup(?)
    get_value_fmt db "%04x %08x %08x %08x %08x", 0
    print_value_fmt db "%04x %08x %08x %08x %08x", 13, 10, 0
    get_n_fmt db "%d", 0
.code

; 18 байт
; multiply (char *a, int n, char* res);
multiply proc
    pushad
    mov ebp, dword ptr [esp + 4 + 8 * 4]   ; ebp - адрес a
    mov eax, dword ptr [esp + 12 + 8 * 4]  ; eax - адрес res

    ; Копируем данные в res
    mov ecx, 0
multiply_copy_cycle:
        mov dh, byte ptr [ebp + ecx]
        mov byte ptr [eax + ecx], dh
        inc ecx
        cmp ecx, 18
        jl multiply_copy_cycle

    ; В счётчик пишем n
    mov ecx, dword ptr [esp + 8 + 8 * 4]
    cmp ecx, 0

    jle multiply_immediate_end

multiply_shift_cycle_n:
        ; Сохраняем текущий счётчик в edx
        mov edx, ecx
        mov ecx, 0

        ; Сброс CF флага
        clc
        pushfd
multiply_shift_cycle_shift:
            ; Восстанавливаем CF
            popfd
            jc significant_set
            jmp significant_not_set
```

```
significant_set:
                ; Если CF установлен, тогда будем добавлять перенесённый бит
                shl byte ptr [eax + ecx], 1
                ; Сохраняем флаги
                pushfd
                ; Добавляем перенесённый бит
                add byte ptr [eax + ecx], 1b
                jmp significant_end
significant_not_set:
                ; Если CF не установлен, просто выполняем сдвиг
                shl byte ptr [eax + ecx], 1
                ; Сохраняем флаги
                pushfd
                jmp significant_end
significant_end:

            ; У нас 36 байтов, поэтому проверяем
            inc ecx
            cmp ecx, 18
            jl multiply_shift_cycle_shift

        popfd

        mov ecx, edx
        dec ecx
        jne multiply_shift_cycle_n
multiply_immediate_end:
    popad
    ret 12
multiply endp

start:
    push offset value
    push offset value + 4
    push offset value + 8
    push offset value + 12
    push offset value + 16
    push offset get_value_fmt
    call crt_scanf
    add esp, 24

    push offset n
    push offset get_n_fmt
    call crt_scanf
    add esp, 8

    push offset res
    push n
    push offset value
    call multiply

    lea ebp, res
    mov eax, dword ptr [ebp]
    push eax
    mov eax, dword ptr [ebp + 4]
    push eax
    mov eax, dword ptr [ebp + 8]
    push eax
    mov eax, dword ptr [ebp + 12]
    push eax
    mov eax, 0
    mov ax, word ptr [ebp + 16]
    push eax
    push offset print_value_fmt
    call crt_printf

    call crt__getch     ; Задержка ввода, getch()
```

```
    ; Вызов функции ExitProcess(0)
    push 0      ; Поместить аргумент функции в стек
    call ExitProcess    ; Выход из программы
end start
```

Тестовые данные:

1. Набор 1:
    a. value = ffff ffffffff ffffffff ffffffff ffffffff
    b. n = 1
    c. res = ffff ffffffff ffffffff ffffffff fffffffe
2. Набор 2:
    a. value = ffff ffffffff ffffffff ffffffff ffffffff
    b. n = 0
    c. res = ffff ffffffff ffffffff ffffffff ffffffff
3. Набор 3:
    a. value = 2455 00000000 39172311 AABC1123 74581234
    b. n = 11
    c. res = a800 000001c8 b9188d55 e0891ba2 c091a000

Результаты выполнения программы:

```
ffff ffffffff ffffffff ffffffff ffffffff
1
ffff ffffffff ffffffff ffffffff fffffffe
```

```
ffff ffffffff ffffffff ffffffff ffffffff
0
ffff ffffffff ffffffff ffffffff ffffffff
```

```
2455 00000000 39172311 AABC1123 74581234
11
a800 000001c8 b9188d55 e0891ba2 c091a000
```

Вывод: в ходе лабораторной изучили команды поразрядной обработки данных.