**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 ПВ-223

Пахомов Владислав Андреевич

Проверили:

ст. пр. Осипов Олег Васильевич

Белгород 2024 г.

**Лабораторная работа №6  
Логические команды и команды сдвига  
Вариант 8**

**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:**

|  |  |  |
| --- | --- | --- |
| 8 | Вывести все 12-разрядные числа, в двоичном представлении которых есть три единицы, остальные нули.  1: 000000 000111  2: 000000 001011  3: 000000 010011  … | 36 байт  деление  без знака |

1 задание:

Обозначим позиции, на которых стоит единица как 1,2,3…12. Будем формировать сочетания из 12 по 3 для получения количества подходящих чисел:

Значит, нужно получить 220 строчек.

Программа:

.686

.model flat, stdcall

option casemap: none

include windows.inc

include kernel32.inc

include msvcrt.inc

includelib  kernel32.lib

includelib  msvcrt.lib

.data

    max\_num dw 4095

    print\_digit db "%d%d%d%d%d%d %d%d%d%d%d%d", 13, 10, 0

.code

; output (short a)

output proc

    ; short sum = 0;

    ; short left[12] = {};

    sub esp, 13 \* 2

    mov dword ptr [esp], 0

    mov dword ptr [esp + 4], 0

    mov dword ptr [esp + 8], 0

    mov dword ptr [esp + 12], 0

    mov dword ptr [esp + 16], 0

    mov dword ptr [esp + 20], 0

    mov word ptr [esp + 24], 0

    ; Сохранение регистров

    pushad

    ; + 4 - адрес возврата

    ; + 13 \* 2 - локальные переменные

    ; + 8 \* 4 - сохранённые регистры

    mov ax, word ptr [esp + 4 + 13 \* 2 + 8 \* 4]

    ; Для 12 цифр в цикле

    mov ecx, 12

cycle:

    mov dx, ax

    ; Заносим маску

    mov bx, 1

    ; Выполняем побитовое и над последней цифрой числа

    and ax, bx

    ; Добавляем количество единиц в переменную sum

    add word ptr [esp + 8 \* 4], ax

    ; В ebp записываем эффективный адрес

    lea ebp, [esp + 8 \* 4 + 13 \* 2]

    ; После чего вычитаем счётчик

    sub ebp, ecx

    sub ebp, ecx

    ; И записываем в left[ecx] результат выполнения

    mov word ptr [ebp], ax

    ; Выполняем побитовый сдвиг вправо

    mov ax, dx

    shr ax, 1

    dec ecx

    jne cycle

    ; Считем сумму

    movsx ecx, word ptr [esp + 8 \* 4]

    mov eax, 3

    cmp ecx, eax

    je print\_val

    jmp dont\_print

    print\_val:

    movsx eax, word ptr [esp + 8 \* 4 + 1 \* 2]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 2 \* 2 + 4]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 3 \* 2 + 4 \* 2]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 4 \* 2 + 4 \* 3]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 5 \* 2 + 4 \* 4]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 6 \* 2 + 4 \* 5]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 7 \* 2 + 4 \* 6]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 8 \* 2 + 4 \* 7]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 9 \* 2 + 4 \* 8]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 10 \* 2 + 4 \* 9]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 11 \* 2 + 4 \* 10]

    push eax

    movsx eax, word ptr [esp + 8 \* 4 + 12 \* 2 + 4 \* 11]

    push eax

    push offset print\_digit

    call crt\_printf

    add esp, 52

    dont\_print:

    ; Восстановление регистров

    popad

    add esp, 13 \* 2

    ret 2

output endp

start:

    mov cx, max\_num

    main\_cycle:

    mov ax, cx

    push ax

    call output

    dec cx

    jge main\_cycle

    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:

111000 000000

110100 000000

110010 000000

110001 000000

110000 100000

110000 010000

110000 001000

110000 000100

110000 000010

110000 000001

101100 000000

101010 000000

101001 000000

101000 100000

101000 010000

101000 001000

101000 000100

101000 000010

101000 000001

100110 000000

100101 000000

100100 100000

100100 010000

100100 001000

100100 000100

100100 000010

100100 000001

100011 000000

100010 100000

100010 010000

100010 001000

100010 000100

100010 000010

100010 000001

100001 100000

100001 010000

100001 001000

100001 000100

100001 000010

100001 000001

100000 110000

100000 101000

100000 100100

100000 100010

100000 100001

100000 011000

100000 010100

100000 010010

100000 010001

100000 001100

100000 001010

100000 001001

100000 000110

100000 000101

100000 000011

011100 000000

011010 000000

011001 000000

011000 100000

011000 010000

011000 001000

011000 000100

011000 000010

011000 000001

010110 000000

010101 000000

010100 100000

010100 010000

010100 001000

010100 000100

010100 000010

010100 000001

010011 000000

010010 100000

010010 010000

010010 001000

010010 000100

010010 000010

010010 000001

010001 100000

010001 010000

010001 001000

010001 000100

010001 000010

010001 000001

010000 110000

010000 101000

010000 100100

010000 100010

010000 100001

010000 011000

010000 010100

010000 010010

010000 010001

010000 001100

010000 001010

010000 001001

010000 000110

010000 000101

010000 000011

001110 000000

001101 000000

001100 100000

001100 010000

001100 001000

001100 000100

001100 000010

001100 000001

001011 000000

001010 100000

001010 010000

001010 001000

001010 000100

001010 000010

001010 000001

001001 100000

001001 010000

001001 001000

001001 000100

001001 000010

001001 000001

001000 110000

001000 101000

001000 100100

001000 100010

001000 100001

001000 011000

001000 010100

001000 010010

001000 010001

001000 001100

001000 001010

001000 001001

001000 000110

001000 000101

001000 000011

000111 000000

000110 100000

000110 010000

000110 001000

000110 000100

000110 000010

000110 000001

000101 100000

000101 010000

000101 001000

000101 000100

000101 000010

000101 000001

000100 110000

000100 101000

000100 100100

000100 100010

000100 100001

000100 011000

000100 010100

000100 010010

000100 010001

000100 001100

000100 001010

000100 001001

000100 000110

000100 000101

000100 000011

000011 100000

000011 010000

000011 001000

000011 000100

000011 000010

000011 000001

000010 110000

000010 101000

000010 100100

000010 100010

000010 100001

000010 011000

000010 010100

000010 010010

000010 010001

000010 001100

000010 001010

000010 001001

000010 000110

000010 000101

000010 000011

000001 110000

000001 101000

000001 100100

000001 100010

000001 100001

000001 011000

000001 010100

000001 010010

000001 010001

000001 001100

000001 001010

000001 001001

000001 000110

000001 000101

000001 000011

000000 111000

000000 110100

000000 110010

000000 110001

000000 101100

000000 101010

000000 101001

000000 100110

000000 100101

000000 100011

000000 011100

000000 011010

000000 011001

000000 010110

000000 010101

000000 010011

000000 001110

000000 001101

000000 001011

000000 000111

Получили 220 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

; Здесь Бога нет тем более

.data

    value db 00h, 00h, 00h, 00h, 0h, 0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h,0h, 0h, 0h

    n dd 4

    res db 36 dup(?)

    get\_value\_fmt db "%08x %08x %08x %08x %08x %08x %08x %08x %08x", 0

    print\_value\_fmt db "%08x %08x %08x %08x %08x %08x %08x %08x %08x", 13, 10, 0

    get\_n\_fmt db "%d", 0

.code

; 36 байт

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

division 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

    division\_copy\_cycle:

    mov dh, byte ptr [ebp + ecx]

    mov byte ptr [eax + ecx], dh

    inc ecx

    cmp ecx, 36

    jl division\_copy\_cycle

    ; В счётчик пишем n

    mov ecx, dword ptr [esp + 8 + 8 \* 4]

    cmp ecx, 0

    jle division\_immediate\_end

    division\_shift\_cycle\_n:

    ; Сохраняем текущий счётчик в edx

    mov edx, ecx

    mov ecx, 35

    ; Сброс CF флага

    clc

    pushfd

    division\_shift\_cycle\_shift:

    ; Восстанавливаем CF

    popfd

    jc significant\_set

    jmp significant\_not\_set

    significant\_set:

    ; Если CF установлен, тогда будем добавлять перенесённый бит

    shr byte ptr [eax + ecx], 1

    ; Сохраняем флаги

    pushfd

    ; Добавляем перенесённый бит

    add byte ptr [eax + ecx], 10000000b

    jmp significant\_end

    significant\_not\_set:

    ; Если CF не установлен, просто выполняем сдвиг

    shr byte ptr [eax + ecx], 1

    ; Сохраняем флаги

    pushfd

    jmp significant\_end

    significant\_end:

    ; У нас 36 байтов, поэтому проверяем

    dec ecx

    jge division\_shift\_cycle\_shift

    popfd

    mov ecx, edx

    dec ecx

    jne division\_shift\_cycle\_n

    division\_immediate\_end:

    popad

    ret 12

division endp

start:

    push offset value

    push offset value + 4

    push offset value + 8

    push offset value + 12

    push offset value + 16

    push offset value + 20

    push offset value + 24

    push offset value + 28

    push offset value + 32

    push offset get\_value\_fmt

    call crt\_scanf

    add esp, 40

    push offset n

    push offset get\_n\_fmt

    call crt\_scanf

    add esp, 8

    push offset res

    push n

    push offset value

    call division

    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, dword ptr [ebp + 16]

    push eax

    mov eax, dword ptr [ebp + 20]

    push eax

    mov eax, dword ptr [ebp + 24]

    push eax

    mov eax, dword ptr [ebp + 28]

    push eax

    mov eax, dword ptr [ebp + 32]

    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:
   1. value = ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   2. n = 1
   3. res = 7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
2. Набор 2:
   1. value = ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
   2. n = 0
   3. res = ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff
3. Набор 3:
   1. value = 26461723 ABC42123 89321320 11111111 FF412455 00000000 39172311 AABC1123 74581234
   2. n = 11
   3. res = 0004c8c2 e4757884 24712642 64022222 223fe824 8aa00000 000722e4 62355782 246e8b02

Результаты выполнения программы:

C:\Users\vladi\Workspace\Assembler\computing\_systems\_architecture\lab6>task2.exe

ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff

1

7fffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff

C:\Users\vladi\Workspace\Assembler\computing\_systems\_architecture\lab6>task2.exe

ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff

0

ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff

C:\Users\vladi\Workspace\Assembler\computing\_systems\_architecture\lab6>task2.exe

26461723 ABC42123 89321320 11111111 FF412455 00000000 39172311 AABC1123 74581234

11

0004c8c2 e4757884 24712642 64022222 223fe824 8aa00000 000722e4 62355782 246e8b02

**Вывод:** в ходе лабораторной изучили команды поразрядной обработки данных.