**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%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 \* 1]

    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

    movsx eax, word ptr [esp + 8 \* 4 + 13 \* 2 + 4 + 2 + 4 \* 12]

    push eax

    push offset print\_digit

    call crt\_printf

    add esp, 56

    dont\_print:

    ; Восстановление регистров

    popad

    cmp word ptr [esp], 3

    je output\_ret\_true

    jmp output\_ret\_false

    output\_ret\_true:

    mov eax, 1

    jmp output\_ret\_end

    output\_ret\_false:

    mov eax, 0

    jmp output\_ret\_end

    output\_ret\_end:

    add esp, 13 \* 2

    ret 4

output endp

start:

    mov dx, 1

    mov cx, max\_num

    main\_cycle:

    mov ax, cx

    push dx

    push ax

    call output

    cmp eax, 1

    je increase\_counter

    jmp increase\_counter\_end

    increase\_counter:

    add dx, 1

    increase\_counter\_end:

    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:

1: 111000 000000

2: 110100 000000

3: 110010 000000

4: 110001 000000

5: 110000 100000

6: 110000 010000

7: 110000 001000

8: 110000 000100

9: 110000 000010

10: 110000 000001

11: 101100 000000

12: 101010 000000

13: 101001 000000

14: 101000 100000

15: 101000 010000

16: 101000 001000

17: 101000 000100

18: 101000 000010

19: 101000 000001

20: 100110 000000

21: 100101 000000

22: 100100 100000

23: 100100 010000

24: 100100 001000

25: 100100 000100

26: 100100 000010

27: 100100 000001

28: 100011 000000

29: 100010 100000

30: 100010 010000

31: 100010 001000

32: 100010 000100

33: 100010 000010

34: 100010 000001

35: 100001 100000

36: 100001 010000

37: 100001 001000

38: 100001 000100

39: 100001 000010

40: 100001 000001

41: 100000 110000

42: 100000 101000

43: 100000 100100

44: 100000 100010

45: 100000 100001

46: 100000 011000

47: 100000 010100

48: 100000 010010

49: 100000 010001

50: 100000 001100

51: 100000 001010

52: 100000 001001

53: 100000 000110

54: 100000 000101

55: 100000 000011

56: 011100 000000

57: 011010 000000

58: 011001 000000

59: 011000 100000

60: 011000 010000

61: 011000 001000

62: 011000 000100

63: 011000 000010

64: 011000 000001

65: 010110 000000

66: 010101 000000

67: 010100 100000

68: 010100 010000

69: 010100 001000

70: 010100 000100

71: 010100 000010

72: 010100 000001

73: 010011 000000

74: 010010 100000

75: 010010 010000

76: 010010 001000

77: 010010 000100

78: 010010 000010

79: 010010 000001

80: 010001 100000

81: 010001 010000

82: 010001 001000

83: 010001 000100

84: 010001 000010

85: 010001 000001

86: 010000 110000

87: 010000 101000

88: 010000 100100

89: 010000 100010

90: 010000 100001

91: 010000 011000

92: 010000 010100

93: 010000 010010

94: 010000 010001

95: 010000 001100

96: 010000 001010

97: 010000 001001

98: 010000 000110

99: 010000 000101

100: 010000 000011

101: 001110 000000

102: 001101 000000

103: 001100 100000

104: 001100 010000

105: 001100 001000

106: 001100 000100

107: 001100 000010

108: 001100 000001

109: 001011 000000

110: 001010 100000

111: 001010 010000

112: 001010 001000

113: 001010 000100

114: 001010 000010

115: 001010 000001

116: 001001 100000

117: 001001 010000

118: 001001 001000

119: 001001 000100

120: 001001 000010

121: 001001 000001

122: 001000 110000

123: 001000 101000

124: 001000 100100

125: 001000 100010

126: 001000 100001

127: 001000 011000

128: 001000 010100

129: 001000 010010

130: 001000 010001

131: 001000 001100

132: 001000 001010

133: 001000 001001

134: 001000 000110

135: 001000 000101

136: 001000 000011

137: 000111 000000

138: 000110 100000

139: 000110 010000

140: 000110 001000

141: 000110 000100

142: 000110 000010

143: 000110 000001

144: 000101 100000

145: 000101 010000

146: 000101 001000

147: 000101 000100

148: 000101 000010

149: 000101 000001

150: 000100 110000

151: 000100 101000

152: 000100 100100

153: 000100 100010

154: 000100 100001

155: 000100 011000

156: 000100 010100

157: 000100 010010

158: 000100 010001

159: 000100 001100

160: 000100 001010

161: 000100 001001

162: 000100 000110

163: 000100 000101

164: 000100 000011

165: 000011 100000

166: 000011 010000

167: 000011 001000

168: 000011 000100

169: 000011 000010

170: 000011 000001

171: 000010 110000

172: 000010 101000

173: 000010 100100

174: 000010 100010

175: 000010 100001

176: 000010 011000

177: 000010 010100

178: 000010 010010

179: 000010 010001

180: 000010 001100

181: 000010 001010

182: 000010 001001

183: 000010 000110

184: 000010 000101

185: 000010 000011

186: 000001 110000

187: 000001 101000

188: 000001 100100

189: 000001 100010

190: 000001 100001

191: 000001 011000

192: 000001 010100

193: 000001 010010

194: 000001 010001

195: 000001 001100

196: 000001 001010

197: 000001 001001

198: 000001 000110

199: 000001 000101

200: 000001 000011

201: 000000 111000

202: 000000 110100

203: 000000 110010

204: 000000 110001

205: 000000 101100

206: 000000 101010

207: 000000 101001

208: 000000 100110

209: 000000 100101

210: 000000 100011

211: 000000 011100

212: 000000 011010

213: 000000 011001

214: 000000 010110

215: 000000 010101

216: 000000 010011

217: 000000 001110

218: 000000 001101

219: 000000 001011

220: 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.