**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:

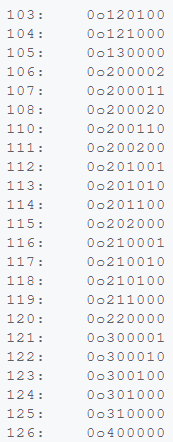
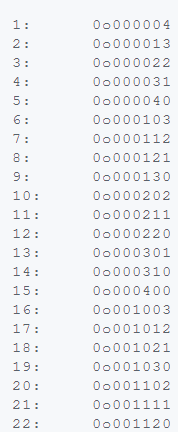
Осипов О.В.

*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.

|  |  |  |
| --- | --- | --- |
| Вари-ант | 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 |



; ==============================================================================

; 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, 111111111111111111b

    JL main\_loop

    invoke ExitProcess, 0                   ; Выход из программы

end start

Чтобы найти количество 18-битных чисел, сумма цифр которых в восьмеричном представлении равна 4, мы можем использовать комбинаторику. Сначала мы переводим 18 бит в восьмеричную систему, получая 6 цифр. Затем мы рассматриваем все возможные способы распределить 4 единицы среди 6 цифр, учитывая, что каждая цифра не может быть больше 7. Это можно сделать с помощью формулы сочетаний с повторениями:

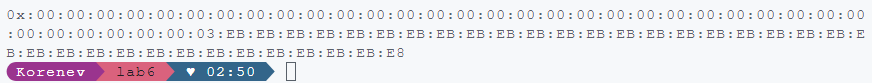
где n - количество цифр, а k - количество единиц. В нашем случае n=6, а k=4, поэтому:

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.



; ==============================================================================

; 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  (смещаем 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 изученили команды поразрядной обработки данных