

Лабораторная работа 6 (0110 = 6)

Целочисленные вычисления. Команды общего назначения

Цель работы: научиться использовать команды общего назначения x86/amd64, в том числе — предназначенные для целочисленных вычислений. Научиться использовать для вычислений *lea* и битовые операции.

Задание Л6.№1

Разработайте на C/C++ функцию *fc16_c(void * p)*, которая принимает адрес 16-битной целочисленной переменной *x* и выполняет беззнаковое округление её значения до кратного *D* (таблица Л6.1) дважды:

- Вниз ($x_1 \leq x$);
- Вверх ($x_2 \geq x$).

Исходное значение *x* и каждый результат *x1* и *x2* печатается *print16()* из Л2.

При *x*, некратном *D*, получится два различных результата: $x_1 < x_2$, но при кратном они должны совпасть: $x_1 = x_2 = x = k \cdot D$.

Как реализовать это без ветвлений, на битовых и арифметических операциях?

Разработайте аналогичную функцию *fc16_asm(void * p)* на ассемблере.

Вариант 2: $D = 32$

```
Введите 16-битное число (0-65535): 100
Исходное: 100
Округление вверх: 128
Округление вниз: 96
```

Рис. 1: Результат выполнения на ассемблере (System V amd64, gcc, linux)

```
Задание №1
=====
SystemInfo
=====
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
=====
Введите число x :100
0064 01100100 100 +100
0060 01100000 96 +96
0080 10000000 128 +128
=====
```

Рис. 2: Результат выполнения в Си (gcc, linux)

Листинг:

Файл task6_1.c:

```
1. #ifndef task4_1_H
2. #define task4_1_H
3.
4. #include <stdio.h>
5. #include <stdint.h>
6.
7. void print16(void *p);
8. void print32(void *p);
9. void print64(void *p);
10.
11. void printSystemInfo();
12.
13. int64_t f1(int64_t x, int64_t y);
14.
15. void run_task4_1()
16. {
17.     printf("\nЗадание №1\n");
18.     printf("=====");
19.     printSystemInfo();
20.     printf("=====\n");
21.
22.     int64_t x = 5;
23.     int64_t y = 3;
24.     int64_t result = f1(x, y);
25.
26.     printf("f1 (\nx= ");
27.     print64(&x);
28.     printf(", \ny= ");
29.     print64(&y);
30.     printf(") =\n");
31.     print64(&result);
32.
33.     printf("\n===== \n");
34. }
35.
36. int64_t f1(int64_t x, int64_t y) {
37.     int64_t result;
38.     __asm__ (
39.         "lea (%1, %2, 8), %0\n\t" // result = x + 8*y
40.         "sub $7, %0"           //
41.         : "=&r" (result)       // output operand
42.         : "r" (x), "r" (y)     // input operands
43.     );
44.     return result;
45. }
46.
47. #endif
```

Файл Ir6_1.S:

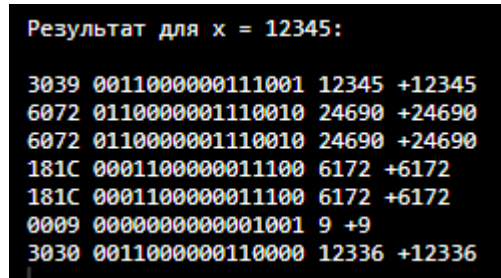
```
1. .section .rodata
2. prompt: .asciz "Введите 16-битное число (0-65535): "
3. scan_fmt: .asciz "%hu"
4. result_fmt: .asciz "Исходное: %u\nОкругление вверх: %u\nОкругление вниз: %u\n"
5.
6. .section .bss
7. .lcomm input, 2 # 16-битная переменная для ввода
8.
9. .section .text
10. .globl main
11. .type main, @function
12.
13. main:
14.     pushq %rbp
15.     movq %rsp, %rbp
16.     subq $16, %rsp # Выравнивание стека
17.
18.     # Вывод приглашения
19.     movq $prompt, %rdi
20.     xorq %rax, %rax
21.     call printf
22.
23.     # Ввод числа
24.     movq $scan_fmt, %rdi
25.     movq $input, %rsi
26.     xorq %rax, %rax
27.     call scanf
28.
29.     # Вызов fc16_asm
30.     movq $input, %rdi
31.     call fc16_asm
32.
33.     # Восстановление и выход
34.     xorq %rax, %rax
35.     leave
36.     ret
37.
38. # Функция округления
39. .globl fc16_asm
40. .type fc16_asm, @function
41. fc16_asm:
42.     pushq %rbp # Сохраняем каллер-сейв регистры
43.     pushq %rbx
44.     pushq %r12
45.
46.     movzwl (%rdi), %ebx # x в ebx (сохраняемое)
47.     movl %ebx, %r12d # копия x в r12d
48.
49.     movl $31, %ecx # D-1 = 31 (volatile регистр)
50.     notl %ecx # mask = ~31
51.
52.     andl %ecx, %r12d # x1 = x & mask
53.
54.     leal 31(%rbx), %eax # x + D-1 (используем lea вместо add)
55.     andl %ecx, %eax # x2 = (x + D-1) & mask
56.
57.     # Подготовка к вызову printf
58.     movzwl %bx, %r8d # исходное x (16 бит, zero-extended)
59.     movl %r12d, %ecx # x1
60.     movl %eax, %edx # x2
61.     movq $result_fmt, %rdi # форматная строка
62.     xorl %eax, %eax # 0 FP args
63.     call printf
64.
65.     # Восстановление регистров
66.     movl %r12d, %eax # возвращаем x1 (или x2 - по условиям задачи)
67.     popq %r12
68.     popq %rbx
69.     popq %rbp
70.     ret
71.
```

Задание Л6.№2

Разработайте на ассемблере функцию `ab16_asm(void * p)`, которая принимает адрес 16-битной целочисленной переменной `x` и выполняет над её копиями все уникальные операции, описанные в задании Л2.№9.

Дублирующиеся арифметические/битовые операции необходимо реализовать один раз — как битовые.

Исходное значение `x` и каждый результат печатается `print16()`.



```
Результат для x = 12345:
3039 0011000000111001 12345 +12345
6072 0110000001110010 24690 +24690
6072 0110000001110010 24690 +24690
181C 0001100000011100 6172 +6172
181C 0001100000011100 6172 +6172
0009 000000000001001 9 +9
3030 0011000000110000 12336 +12336
```

Рис. 3: результат выполнения `ab16_asm`

Листинг:

Файл `lr6_2.S`:

```
1. .section .data
2. .align 2
3. mask_15: .word 0x000F      # 15
4. mask_neg16: .word 0xFFFF0  # -16 (0xFFFF0 для 16 бит)
5.
6. .section .rodata
7. # Форматные строки для printf
8. print_format: .asciz "%04X %016b %u %d \n"
9. prompt: .asciz "Результат для x = 12345:\n"
10.
11. .section .text
12. .globl main
13. .globl print16
14. .globl ab16_asm
15.
16. # Функция print16
17. print16:
18.     pushq %rbp
19.     movq %rsp, %rbp
20.
21.     # Загружаем 16-битное значение из памяти
22.     movzwl (%rdi), %esi      # arg2: unsigned short (для %04X)
23.     movzwl (%rdi), %edx      # arg3: unsigned short (для %08b)
24.     movzwl (%rdi), %ecx      # arg4: unsigned short (для %u)
25.     movswl (%rdi), %r8d      # arg5: signed short (для %d)
26.
27.     movq $print_format, %rdi # arg1: форматная строка
28.     call printf
29.
30.     popq %rbp
31.     ret
32.
33. # Основная функция
34. main:
35.     pushq %rbp
36.     movq %rsp, %rbp
37.     subq $16, %rsp          # Выделяем место для переменных
38.
39.     # Выводим приглашение
40.     movq $prompt, %rdi
41.     call puts
42.
43.     # Создаем и инициализируем переменную x = 12345
44.     movw $12345, -2(%rbp)
45.
46.     # Вызываем ab16_asm с адресом x
47.     leaq -2(%rbp), %rdi
48.     call ab16_asm
49.
50.     # Возвращаем 0
```

```

51.    xorl %eax, %eax
52.    leave
53.    ret
54.
55. ab16_asm:
56.    push %rbp
57.    mov %rsp, %rbp
58.
59.    sub $32, %rsp          # выделим 32 байта стека для локальных переменных
60.
61.    # rdi = указатель на x
62.    movzwl (%rdi), %eax    # загрузить x (16 бит) в eax (обнулить старшие)
63.    mov %ax, %bx          # сохранить x в bx для повторного использования
64.
65.    movw %bx, -32(%rbp)    # Сохраняем x для печати
66.
67.    # a1) беззнаковое умножение на 2: x * 2 = x << 1 (беззнаковое)
68.    mov %bx, %cx
69.    shl $1, %cx           # умножение на 2
70.    movw %cx, -30(%rbp)
71.
72.    # a2) знаковое умножение на 2: x * 2 = x << 1 (знаковое)
73.    mov %bx, %cx
74.    sal $1, %cx           # знаковый сдвиг влево на 1
75.    movw %cx, -28(%rbp)
76.
77.    # a3) беззнаковое деление на 2: x / 2 = x >> 1 (беззнаковое)
78.    mov %bx, %cx
79.    shr $1, %cx
80.    movw %cx, -26(%rbp)
81.
82.    # a4) знаковое деление на 2: x / 2 = x >> 1 (знаковое)
83.    movswx %bx, %cx       # расширяем bx в cx знаково
84.    sar $1, %cx
85.    movw %cx, -24(%rbp)
86.
87.    # a5) остаток от беззнакового деления на 16: x % 16 = x & 15
88.    mov %bx, %cx
89.    andw $0x000F, %cx
90.    movw %cx, -22(%rbp)
91.
92.    # a6) округление вниз до кратного 16 (беззнаковое): x & (~15)
93.    mov %bx, %cx
94.    andw $0xFFF0, %cx
95.    movw %cx, -20(%rbp)
96.
97.    # 61) беззнаковый сдвиг влево на 1 бит: x << 1
98.    # Уже сделано в a1), используем -30(%rbp)
99.    # 62) знаковый сдвиг влево на 1 бит: x << 1
100.    # Уже сделано в a2), используем -28(%rbp)
101.    # 63) беззнаковый сдвиг вправо на 1 бит: x >> 1
102.    # Уже сделано в a3), используем -26(%rbp)
103.    # 64) знаковый сдвиг вправо на 1 бит: x >> 1
104.    # Уже сделано в a4), используем -24(%rbp)
105.    # 65) x & 15
106.    # Уже сделано в a5), используем -22(%rbp)
107.    # 66) x & -16
108.    # Уже сделано в a6), используем -20(%rbp)
109.
110.    # Печать результатов
111.    lea -32(%rbp), %rdi    # Исходное значение
112.    call print16
113.    lea -30(%rbp), %rdi    # a1
114.    call print16
115.    lea -28(%rbp), %rdi    # a2
116.    call print16
117.    lea -26(%rbp), %rdi    # a3
118.    call print16
119.    lea -24(%rbp), %rdi    # a4
120.    call print16
121.    lea -22(%rbp), %rdi    # a5
122.    call print16
123.    lea -20(%rbp), %rdi    # a6
124.    call print16
125.
126.    leave
127.    ret
128.

```

Задание Л6.№3

Вычислите целочисленное выражение $z = 12 - x - y^2$ для заданных целых x и y .
Разрядность x , y , z совпадает.

```
Задание №3
=====
SystemInfo
=====
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
=====
z = 12 - 5 - 3^2 = -2
=====
```

Рис. 4: результат выполнения calc_z

Листинг:

Файл task6_3.c:

```
1. // Функция округления вниз и вверх до кратного 32 без ветвлений
2. int calc_z(int x, int y) {
3.     int z;
4.     int y_sq;
5.
6.     // Вычисляем y^2
7.     __asm__ (
8.         "imull %1, %1;" // y = y * y
9.         : "+r" (y)
10.    );
11.
12.    y_sq = y;
13.
14.    // Вычисляем z = 12 - x - y_sq
15.    __asm__ (
16.        "movl $12, %0;"
17.        "subl %1, %0;"
18.        "subl %2, %0;"
19.        : "=&r" (z)
20.        : "r" (x), "r" (y_sq)
21.    );
22.
23.    return z;
24. }
```

Задание Л6.№4.

Вычислите беззнаковое и знаковое целочисленное выражение $\{ z = (x + 2)/y, w = (x + 2)\%y$ для заданных целых x и y .

Сравните результаты беззнакового и знакового деления в случае, когда делимое равно -3 , а делитель $+2$.

```
Задание №4
=====
SystemInfo
=====
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
=====
x= -3, y= 2
Unsigned division: z = 2147483647, w = 1
Signed division: z = 0, w = -1
=====
```

Рис. 5: выполнение unsigned_div_mod и signed_div_mod

Листинг:

Файл task6_4.c:

```
1. // Беззнаковое деление и остаток
2. void unsigned_div_mod(uint32_t x, uint32_t y, uint32_t *z, uint32_t *w) {
3.     uint32_t num = x + 2;
4.
5.     __asm__ (
6.         "divl %4"
7.         : "=a" (*z), "=d" (*w)
8.         : "a" (num), "d" (0), "r" (y)
9.     );
10. }
11.
12. // Знаковое деление и остаток
13. void signed_div_mod(int32_t x, int32_t y, int32_t *z, int32_t *w) {
14.     int32_t num = x + 2;
15.
16.     __asm__ (
17.         "cld\n\t"           // Расширить eax в edx:eax (знаковое расширение)
18.         "idivl %4"
19.         : "=a" (*z), "=d" (*w)
20.         : "a" (num), "d" (0), "r" (y)
21.     );
22. }
23.
```

Задание Л6.№5.

Вычислите целочисленное выражение $z = 16x$ для заданного целого x , используя одну команду. Разрядность x и z совпадает.

Штраф -2 балла, если умножение выполняется командами *mul/imul*.

```
Задание №5
=====
SystemInfo
=====
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
=====
16 * 7 = 112
=====
```

Рис. 7: выполнение mul16

Листинг:

Файл task6_5.c:

```
1. int mul16(int x);
2.
3. void run_task6_5()
4. {
5.     printf("\nЗадание №5\n");
6.     printf("=====");
7.     printSystemInfo();
8.     printf("=====\\n");
9.
10.    int x = 7;
11.    int z = mul16(x);
12.    printf("16 * %d = %d\\n", x, z);
13.
14.    printf("=====\\n");
15. }
16.
17. int mul16(int x) {
18.     int z;
19.     __asm__ (
20.         "sal $4, %1\\n\\t"
21.         "movl %1, %0"
22.         : "=r"(z)
23.         : "r"(x)
24.     );
25.     return z;
26. }
```


Задание Л6.№6.

Разработайте функцию `int mce_si(void *p, size_t N)`, которая принимает адрес массива из N элементов типа `int/unsigned` и рассчитывает для него значение по варианту.

Вариант 2: произведение.

```
Задание №6
=====
SystemInfo
=====
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
=====
Array:
00000002 00000010 2 +2 +0x1p-148 +2.802597e-45 +0.00
00000003 00000011 3 +3 +0x1.8p-148 +4.203895e-45 +0.00
00000004 00000100 4 +4 +0x1p-147 +5.605194e-45 +0.00
00000005 00000101 5 +5 +0x1.4p-147 +7.006492e-45 +0.00
00000006 00000110 6 +6 +0x1.8p-147 +8.407791e-45 +0.00
Product of array elements =
000002D0 1011010000 720 +720 +0x1.68p-140 +1.008935e-42 +0.00
=====
```

Рис. 7: выполнение `mce_si`

Листинг:

Файл `task6_6.c`:

```
1. // Макрос для вывода массива с параметризацией массива и делегата вывода
2. #define PRINT_ARRAY(arr, printer) \
3. do { \
4.     size_t n = sizeof(arr) / sizeof(arr[0]); \
5.     for (int i = 0; i < n; i++) { \
6.         printer(&arr[i]); \
7.     } \
8. } while(0)
9.
10. int mce_si(void *p, size_t N);
11.
12. void run_task6_6()
13. {
14.     printf("\nЗадание №6\n");
15.     printf("=====");
16.     printSystemInfo();
17.     printf("=====\\n");
18.     int arr[] = {2, 3, 4, 5, 6};
19.     size_t N = sizeof(arr) / sizeof(arr[0]);
20.
21.     int prod = mce_si(arr, N);
22.     printf("Array:\\n");
23.     PRINT_ARRAY(arr, print32);
24.     printf("Product of array elements = \\n");
25.     print32(&prod);
26.     printf("=====\\n");
27. }
28.
29. int mce_si(void *p, size_t N) {
30.     int *arr = (int *)p;
31.     int result = 1;
32.     size_t i;
33.     for (i = 0; i < N; i++) {
34.         int val = arr[i];
35.         __asm__ (
36.             "imull %1, %0"
37.             : "+r" (result)
38.             : "r" (val)
39.         );
40.     }
41.     return result;
42. }
```