

Лабораторная работа 5 (0101 = 5)

Флаги и условные команды. Ветвления и циклы.

Цель работы: ознакомиться с набором флагов состояния регистра *flags*, семействами команд, выполняющихся по-разному в зависимости от флагов (условных команд); научиться реализовывать ветвления и циклы на ассемблере.

Задание Л5.№1

Вычислите сумму двух целых чисел $z = x + y$, используя команду *add*. Сформируйте *w* (таблица Л5.1), используя семейство команд *setCC* и анализируя флаги состояния *CF*, *OF*, *SF*, *ZF*, *AF*, *PF* после вычисления *z*.

Вариант 2: $w = \{ 0, \text{если не было знакового переполнения}, 1, \text{если было знаковое переполнение} \}$

```
Задание №1
=====
SystemInfo
=====
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
=====
Введите два целых числа x и y:
3 6
x = 3
y = 6
z = x + y = 9
w (OF flag) = 0
=====
```

```
Задание №1
=====
SystemInfo
=====
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
=====
Введите два целых числа x и y:
1234567890 1234567890
x = 1234567890
y = 1234567890
z = x + y = -1825831516
w (OF flag) = 1
=====
```

Рис. 1: Вывод флага в результате обычного вычисления и с переполнением

Листинг:

Файл task5_1.c:

```
1. void run_task5_1()
2. {
3.     printf("\nЗадание №1\n");
4.     printf("=====");
5.     printSystemInfo();
6.     printf("===== \n");
7.
8.     int x, y, z;
9.     unsigned char w;
10.
11.     printf("Введите два целых числа x и y:\n");
12.     if (scanf("%d %d", &x, &y) != 2) {
13.         printf("Ошибка ввода\n");
14.         return;
15.     }
16.
17.     // Inline asm: z = x + y; w = OF (знаковое переполнение)
18.     __asm__ volatile (
19.         "movl %[x], %%eax\n\t"
20.         "addl %[y], %%eax\n\t"
21.         "movl %%eax, %%ebx\n\t"
22.         "seto %%al\n\t"
23.         "movb %%al, %[w]\n\t"
24.         "movl %%ebx, %[z]\n\t"
25.         : [z] "=r" (z), [w] "=r" (w)
26.         : [x] "r" (x), [y] "r" (y)
27.         : "eax", "ebx"
28.     );
29.
30.     printf("x = %d\n", x);
31.     printf("y = %d\n", y);
32.     printf("z = x + y = %d\n", z);
33.     printf("w (OF flag) = %u\n", w);
34.
35.     printf("\n===== \n");
36. }
```

Задание Л5.№2

Вычислите z для заданного целого беззнакового x (таблица Л5.2); z принимает значение 1 либо 0, аналогично операторам сравнения C/C++

Вариант 2: $z = (x > -3)$

Задание №2	Задание №2
SystemInfo	SystemInfo
ОС: Linux	ОС: Linux
Архитектура процессора: x86_64 (64-бит)	Архитектура процессора: x86_64 (64-бит)
Compiler: GCC	Compiler: GCC
Version: 13.2.1	Version: 13.2.1
Введите целое число в любой системе x:	Введите целое число x:
4294967293	555555555
$z = (4294967293 > -3) = 0$	$z = (1260588259 > -3) = 0$

Рис. 2:

Листинг:

Файл task5_2.c:

```
1. void run_task5_2()
2. {
3.     printf("\nЗадание №2\n");
4.     printf("=====");
5.     printSystemInfo();
6.     printf("=====\n");
7.
8.     uint32_t x;
9.     unsigned char z;
10.
11.    printf("Введите целое число x:\n");
12.    if (scanf("%u", &x) != 1) {
13.        printf("Ошибка ввода\n");
14.        return;
15.    }
16.
17.    // Сравним младшие 32 бита x с 0xFFFFFFFF
18.    uint32_t x32 = (uint32_t)(x & 0xFFFFFFFF);
19.
20.    __asm__ volatile (
21.        "cmpl $0xFFFFFFFF, %[x32]\n\t" // сравнить x с -3
22.        "seta %[z]\n\t"                // z = 1, если x > -3
23.        : [z] "=r" (z)
24.        : [x32] "r" (x32)
25.        : "cc"
26.    );
27.
28.    printf("z = (%u > -3) = %u\n", x32, z);
29.
30.    printf("=====\n");
31. }
```

Задание Л5.№3

Реализуйте Л5.№2 для целого знакового x

Задание №2	Задание №2
===== SystemInfo =====	===== SystemInfo =====
ОС: Linux Архитектура процессора: x86_64 (64-бит) Compiler: GCC Version: 13.2.1 =====	ОС: Linux Архитектура процессора: x86_64 (64-бит) Compiler: GCC Version: 13.2.1 =====
Введите целое число в любой системе x : -3 $z = (-3 > -3) = 0$ =====	Введите целое число в любой системе x : 1 $z = (1 > -3) = 1$ =====

Рис. 3: результат выполнения сравнения

Листинг

Файл task5_3.c:

```
1. void run_task5_3()
2. {
3.     printf("\nЗадание №2\n");
4.     printf("=====\n");
5.     printSystemInfo();
6.     printf("=====\n");
7.
8.     uint32_t x;
9.     unsigned char z;
10.
11.    printf("Введите целое число в любой системе  $x$ :\n");
12.    if (scanf("%i", &x) != 1) {
13.        printf("Ошибка ввода\n");
14.        return;
15.    }
16.
17.    __asm__ volatile (
18.        "cml $0xFFFFFFFF, %[x]\n\t" // сравнить  $x$  с -3
19.        "setg %[z]\n\t"           //  $z = 1$ , если  $x > -3$  (знаковое сравнение)
20.        : [z] "=r" (z)
21.        : [x] "r" (x)
22.        : "cc"
23.    );
24.
25.    printf("z = ( $d > -3$ ) =  $u$ \n", x, z);
26.
27.    printf("=====\n");
28. }
```

Задание Л5.№4.

Реализуйте Л5.№2 для x с плавающей запятой (таблица Л5.3), используя AVX-команды сравнения `vcomisd/vcomiss` (или их SSE-аналоги).

Вариант 2: Одинарной точности (*float*)

Задание №4	Задание №4
===== SystemInfo ===== ОС: Linux Архитектура процессора: x86_64 (64-бит) Compiler: GCC Version: 13.2.1 ===== Введите число с плавающей точкой x: -3.1 z = (-3.100 > -3) = 1 =====	===== SystemInfo ===== ОС: Linux Архитектура процессора: x86_64 (64-бит) Compiler: GCC Version: 13.2.1 ===== Введите число с плавающей точкой x: 1 z = (1.000 > -3) = 1 =====

Рис. 4: результат выполнения сравнения

Листинг:

Файл task5_4.c:

```
1. void run_task5_4()
2. {
3.     printf("\nЗадание №4\n");
4.     printf("=====\n");
5.     printSystemInfo();
6.     printf("=====\n");
7.
8.     float x;
9.     unsigned char z;
10.    float cmp_val = -3.0f;
11.
12.    printf("Введите число с плавающей точкой x:\n");
13.    if (scanf("%f", &x) != 1) {
14.        printf("Ошибка ввода\n");
15.        return;
16.    }
17.
18.    __asm__ volatile (
19.        "vmovss %[x], %%xmm0\n\t"           // загрузить x в xmm0
20.        "vmovss %[cmp_val], %%xmm1\n\t"     // загрузить -3.0 в xmm1
21.        "vcomiss %%xmm1, %%xmm0\n\t"        // сравнить xmm0 с xmm1 (x с -3.0)
22.        "setg %[z]\n\t"                     // z = 1, если x > -3.0, иначе 0
23.        : [z] "=r" (z)
24.        : [x] "m" (x), [cmp_val] "m" (cmp_val)
25.        : "xmm0", "xmm1", "cc"
26.    );
27.
28.    printf("z = (%0.3f > -3) = %u\n", x, z);
29.
30.    printf("=====\n");
31. }
32.
```

Задание Л5.№5.

Вычислите z для заданного целого беззнакового x (таблица Л5.4), используя семейство условных команд *stovcc* и выставляя флаги состояния при помощи команды *str*.

Бонус +1 балл, если вычисление линейной комбинации производится одной командой *lea*.

Вариант 2:

$$z = \begin{cases} -2 + x, & -2 + x \geq -2, \\ 25, & -2 + x < -2 \end{cases}$$

Задание №5	Задание №5
SystemInfo	SystemInfo
ОС: Linux	ОС: Linux
Архитектура процессора: x86_64 (64-бит)	Архитектура процессора: x86_64 (64-бит)
Compiler: GCC	Compiler: GCC
Version: 13.2.1	Version: 13.2.1
Введите целое беззнаковое число x: 1	Введите целое беззнаковое число x: 3
Результат: z = 25	Результат: z = 1

Рис. 5: результат выполнения вставки

Листинг:

Файл task5_5.c:

```
1. void run_task5_5()
2. {
3.     printf("\nЗадание №5\n");
4.     printf("=====");
5.     printSystemInfo();
6.     printf("=====\n");
7.
8.     unsigned int x;
9.     unsigned int z;
10.
11.     printf("Введите целое беззнаковое число x: ");
12.     if (scanf("%u", &x) != 1) {
13.         printf("Ошибка ввода\n");
14.         return ;
15.     }
16.
17.     __asm__ volatile (
18.         "mov %[x], %%eax;" // eax = x
19.         "lea -2(%%eax), %%ecx;" // ecx = x - 2
20.         "mov $25, %%edx;" // edx = 25
21.         "cmp $2, %%eax;" // сравниваем x с 2
22.         "cmovb %%edx, %%ecx;" // если x < 2, то ecx = 25
23.         "mov %%ecx, %[z];" // z = ecx
24.         : [z] "=r" (z)
25.         : [x] "r" (x)
26.         : "eax", "ecx", "edx"
27.     );
28.
29.     printf("Результат: z = %u\n", z);
30.
31.     printf("=====\n");
32. }
```

Задание Л5.№6.

Заполните массив из N целочисленных элементов первыми N членами последовательности (таблица Л5.5). Выделение памяти под массив может быть выполнено на языке C/C++, в этом случае в ассемблерную функцию передаётся адрес начала массива и длина N .

Вариант 2: Нечётные неотрицательные: 1, 3, 5, 7, 9...

```
Задание №6
=====
SystemInfo
=====
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
=====
Введите длину массива N: 6
Результат (первые 6 нечётных неотрицательных чисел):
00000001 00000001 1 +1 +0x1p-149 +1.401298e-45 +0.00
00000003 00000011 3 +3 +0x1.8p-148 +4.203895e-45 +0.00
00000005 00000101 5 +5 +0x1.4p-147 +7.006492e-45 +0.00
00000007 00000111 7 +7 +0x1.cp-147 +9.809089e-45 +0.00
00000009 00001001 9 +9 +0x1.2p-146 +1.261169e-44 +0.00
0000000B 00001011 11 +11 +0x1.6p-146 +1.541428e-44 +0.00
=====
```

Рис. 6: результат выполнения вставки

Листинг:

Файл task5_6.c:

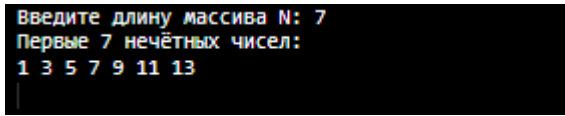
```
1. void run_task5_6()
2. {
3.     printf("\nЗадание №6\n");
4.     printf("=====");
5.     printSystemInfo();
6.     printf("=====\\n");
7.
8.
9.     int N;
10.    printf("Введите длину массива N: ");
11.    if (scanf("%d", &N) != 1 || N <= 0) {
12.        fprintf(stderr, "Ошибка ввода N\\n");
13.        return;
14.    }
15.
16.
17.    int* arr = (int*)malloc(N * sizeof(int));
18.    if (!arr) {
19.        fprintf(stderr, "Ошибка выделения памяти\\n");
20.        return ;
21.    }
22.    int i = 0;
23.    __asm__ volatile (
24.        "mov %[arr], %%rsi;"           // rsi = arr
25.        "xor %%rax, %%rax;"           // i = 0 (rax - 64-bit)
26.        "test %[N], %[N];"
27.        "jz 2f;"                       // если N == 0, выйти
28.        "1:"
29.        "lea (%%rax, %%rax, 1), %%edx;" // edx = 2*i (edx - 32-bit)
30.        "add $1, %%edx;"               // edx = 2*i + 1
31.        "movl %%edx, (%%rsi, %%rax, 4);" // arr[i] = 2*i + 1
32.        "inc %%rax;"                  // i++
33.        "cmp %[N], %%rax;"
34.        "jl 1b;"
35.        "2:"
36.        :
37.        : [arr] "r" (arr), [N] "r" ((unsigned long)N)
38.        : "rax", "rdx", "rsi", "memory"
39.    );
40.
41.    printf("Результат (первые %d нечётных неотрицательных чисел):\\n", N);
42.    PRINT_ARRAY(arr, print32);
43.
44.    free(arr);
45.
46.    printf("=====\\n");
47. }
```

Задание Л5.№7.

Бонус +2 балла для пар, обязательное для троек.

Напечатайте первые N членов последовательности (таблица Л5.5), не сохраняя их в массиве.

Так как вызов функций из ассемблерной вставки является неопределённым поведением, это задание может быть выполнено только как функция, целиком выполненная на ассемблере (или фрагмент функции `main()`, целиком выполненной на ассемблере).



```
Введите длину массива N: 7
Первые 7 нечётных чисел:
1 3 5 7 9 11 13
```

Рис. 7: результат выполнения без использования массивов

Листинг:

Файл `lr5_7.S`:

```
1. .section .rodata
2. prompt: .asciz "Введите длину массива N: "
3. fmt: .asciz "%d"
4. result_msg: .asciz "Первые %d нечётных чисел:\n"
5. num_fmt: .asciz "%d "
6. newline: .asciz "\n"
7. error_msg: .asciz "Ошибка: N должно быть положительным числом\n"
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           # Выделяем место для переменной N
17.
18.     # Выводим приглашение
19.     movq $prompt, %rdi
20.     xorq %rax, %rax
21.     call printf
22.
23.     # Считываем N
24.     leaq -4(%rbp), %rsi      # Указатель на переменную N в стеке
25.     movq $fmt, %rdi
26.     xorq %rax, %rax
27.     call scanf
28.
29.     # Проверяем корректность ввода
30.     cmpl $0, -4(%rbp)
31.     jle .error
32.
33.     # Выводим сообщение о результате
34.     movq $result_msg, %rdi
35.     movl -4(%rbp), %esi
36.     xorq %rax, %rax
37.     call printf
38.
39.     # Генерируем и выводим нечетные числа
40.     xorl %ebx, %ebx          # Счетчик i = 0
41.     movl -4(%rbp), %r12d     # N
42.
43. .generate_loop:
44.     cmpl %r12d, %ebx
45.     jge .end_loop
46.
47.     # Вычисляем 2*i + 1
48.     leal 1(%ebx, %ebx), %esi # esi = 2*i + 1 (корректный расчет)
49.
50.     # Выводим число
51.     movq $num_fmt, %rdi      # Первый аргумент - формат
52.     # %esi уже содержит число (второй аргумент)
53.     xorq %rax, %rax
54.     call printf
55.
56.     incl %ebx
```

```
57.     jmp .generate_loop
58.
59. .end_loop:
60.     # Выводим перевод строки
61.     movq $newline, %rdi
62.     xorq %rax, %rax
63.     call printf
64.
65.     xorq %rax, %rax          # Возвращаем 0
66.     jmp .exit
67.
68. .error:
69.     movq stderr(%rip), %rdi
70.     movq $error_msg, %rsi
71.     xorq %rax, %rax
72.     call fprintf
73.     movq $1, %rax          # Возвращаем 1
74.
75. .exit:
76.     movq %rbp, %rsp
77.     popq %rbp
78.     ret
79.
```


Задание Л5.№8.

Разработайте функцию, которая принимает адрес pM матрицы M из $R \times C$ или $N \times N$ элементов типа $int/unsigned$ и заменяет в ней часть элементов по варианту на (-1) .

Выделение памяти, заполнение и печать M до и после изменения — на языке C/C++. Элементы матрицы M печатаются как матрица: элементы одной строки на одной строке и разделяются пробелами; младшая цифра $M_{i+1,j}$ под младшей цифрой $M_{i,j}$.

Штраф –1 балл, если цикл по элементам строки/столбца/диагонали содержит вложенные ветвления для расчёта i и j . Используйте арифметические операции.

Вариант 2: `mre(void * pM, size_t R, size_t C, size_t j)` заменяет столбец j

```
Задание №8
=====
SystemInfo
=====
ОС: Linux
Архитектура процессора: x86_64 (64-бит)
Compiler: GCC
Version: 13.2.1
=====
Исходная матрица:
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
Матрица после замены столбца 2 на -1:
0 1 -1 3 4
5 6 -1 8 9
10 11 -1 13 14
15 16 -1 18 19
=====
```

Рис. 8: результат замены столбца

Листинг:

Файл task5_8.c:

```
1. #ifndef task5_8_H
2. #define task5_8_H
3.
4. #include <stdio.h>
5. #include <stdint.h>
6. #include <stdlib.h>
7.
8. void print_matrix(int *M, size_t R, size_t C);
9. void mre(void *pM, size_t R, size_t C, size_t j);
10.
11. void print16(void *p);
12. void print32(void *p);
13. void print64(void *p);
14.
15. void printSystemInfo();
16.
17. void run_task5_8()
18. {
19.     printf("\nЗадание №8\n");
20.     printf("=====");
21.     printSystemInfo();
22.     printf("===== \n");
23.
24.     size_t R = 4, C = 5;
25.     int *M = (int*)malloc(R * C * sizeof(int));
26.     if (!M) {
27.         fprintf(stderr, "Ошибка выделения памяти\n");
28.         return;
29.     }
30.
31.     // Заполнение матрицы числами от 0 до R*C-1
32.     for (size_t i = 0; i < R; i++) {
33.         for (size_t k = 0; k < C; k++) {
34.             M[i*C + k] = (int)(i*C + k);
35.         }
36.     }
37.
38.     printf("Исходная матрица:\n");
```

```

39.     print_matrix(M, R, C);
40.
41.     size_t j = 2; // например, заменяем 3-й столбец (индексация с 0)
42.     mre(M, R, C, j);
43.
44.     printf("Матрица после замены столбца %zu на -1:\n", j);
45.     print_matrix(M, R, C);
46.
47.     free(M);
48.
49.     printf("=====\n");
50. }
51.
52. void print_matrix(int *M, size_t R, size_t C) {
53.     for (size_t i = 0; i < R; i++) {
54.         for (size_t k = 0; k < C; k++) {
55.             printf("%d ", M[i*C + k]);
56.         }
57.         printf("\n");
58.     }
59. }
60.
61. void mre(void *pM, size_t R, size_t C, size_t j) {
62.     __asm__ volatile (
63.         // Загружаем параметры в регистры
64.         "mov %[pM], %%rdi\n\t"      // rdi = указатель на матрицу (M)
65.         "mov %[R], %%rcx\n\t"      // rcx = количество строк (R)
66.         "mov %[C], %%r8\n\t"       // r8 = количество столбцов (C)
67.         "mov %[j], %%r9\n\t"       // r9 = индекс столбца (j)
68.         "xor %%r10, %%r10\n\t"     // r10 = счётчик строк (i) = 0
69.
70.         // Проверка на нулевые размеры
71.         "test %%rcx, %%rcx\n\t"
72.         "jz 2f\n\t"
73.
74.         "1:\n\t"
75.         // Вычисляем адрес элемента M[i][j] = M + (i * C + j) * sizeof(int)
76.         "mov %%r10, %%rax\n\t"     // rax = i
77.         "mul %%r8\n\t"             // rax = i * C
78.         "add %%r9, %%rax\n\t"     // rax = i * C + j
79.         "movl $-1, (%%rdi, %%rax, 4)\n\t" // M[i][j] = -1
80.
81.         // Увеличиваем счётчик и проверяем условие
82.         "inc %%r10\n\t"
83.         "cmp %%rcx, %%r10\n\t"
84.         "jl 1b\n\t"
85.
86.         "2:\n\t"
87.         : // Нет выходных операндов
88.         : [pM] "r" (pM), [R] "r" (R), [C] "r" (C), [j] "r" (j)
89.         : "rax", "rcx", "rdi", "r8", "r9", "r10", "memory", "cc"
90.     );
91. }
92.
93. #endif

```