Лабораторная работа №0
Assembly

Выполнил
Зайцев Александр, Б03-305

1. Ассемблерный листинг программы «Hello, world!»

```c
#include "stdio.h"

int main(){
        printf("Hello, World!\n");
}
```

Рис. 1: Программа на языке C

```
        .file   "hello_world.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello, World!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        call    puts@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
        .align 8
4:
~
```

Рис. 1: Ассемблерный листинг программы

2. Ассемблерный листинг программы с простой арифметикой

Напишем код на двух языках, выполняющий одни и те же команды

```c
#include "stdio.h"

int a = 5;

int main() {
        int first = 1;
        int second = 16;

        int result;

        result = first + second;

        long int result_next = first * second;

        double result_next_2 = result / 5;

        char f = 'e';

        printf("%d\n", f);

        first = result - second;
        printf("Hello World");
}
```

Рис. 2: C

```cpp
#include <iostream>

int a = 5;

int main() {
        int first = 1;
        int second = 16;

        int result;

        result = first + second;

        long int result_next = first * second;

        double result_next_2 = result / 5;

        char f = 'e';

        std::cout << f << std::endl;

        first = result - second;
        std::cout << "Hello world";
}
```

Рис.3: C++

Генерируем их ассемблерные листинги.
Сначала программа, которая написана на языке C:

```
        .file   "lab0.c"
        .text
        .globl  a
        .data
        .align 4
        .type   a, @object
        .size   a, 4
a:
        .long   5
        .section        .rodata
.LC0:
        .string "%d\n"
.LC1:
        .string "Hello World"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $32, %rsp
        movl    $1, -28(%rbp)
        movl    $16, -24(%rbp)
        movl    -28(%rbp), %edx
        movl    -24(%rbp), %eax
        addl    %edx, %eax
        movl    %eax, -20(%rbp)
        movl    -28(%rbp), %eax
        imull   -24(%rbp), %eax
        cltq
        movq    %rax, -16(%rbp)
        movl    -20(%rbp), %eax
        movslq  %eax, %rdx
        imulq   $1717986919, %rdx, %rdx
        shrq    $32, %rdx
        sarl    %edx
        sarl    $31, %eax
        movl    %eax, %ecx
        movl    %edx, %eax
        subl    %ecx, %eax
```

```
        pxor    %xmm0, %xmm0
        cvtsi2sdl       %eax, %xmm0
        movsd   %xmm0, -8(%rbp)
        movb    $101, -29(%rbp)
        movsbl  -29(%rbp), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    -20(%rbp), %eax
        subl    -24(%rbp), %eax
        movl    %eax, -28(%rbp)
        leaq    .LC1(%rip), %rax
        movq    %rax, %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
        .align 8
4:
```

Рис. 4: Ассемблерный листинг программы на C

Затем программа на C++:

```
        .file   "lab0.c"
        .text
        .globl  a
        .data
        .align 4
        .type   a, @object
        .size   a, 4
a:
        .long   5
        .section        .rodata
.LC0:
        .string "%d\n"
.LC1:
        .string "Hello World"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $32, %rsp
        movl    $1, -28(%rbp)
        movl    $16, -24(%rbp)
        movl    -28(%rbp), %edx
        movl    -24(%rbp), %eax
        addl    %edx, %eax
        movl    %eax, -20(%rbp)
        movl    -28(%rbp), %eax
        imull   -24(%rbp), %eax
        cltq
        movq    %rax, -16(%rbp)
        movl    -20(%rbp), %eax
        movslq  %eax, %rdx
        imulq   $1717986919, %rdx, %rdx
        shrq    $32, %rdx
        sarl    %edx
        sarl    $31, %eax
        movl    %eax, %ecx
        movl    %edx, %eax
        subl    %ecx, %eax
        pxor    %xmm0, %xmm0
        cvtsi2sdl       %eax, %xmm0
        movsd   %xmm0, -8(%rbp)
        movb    $101, -29(%rbp)
        movsbl  -29(%rbp), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    -20(%rbp), %eax
        subl    -24(%rbp), %eax
        movl    %eax, -28(%rbp)
        leaq    .LC1(%rip), %rax
        movq    %rax, %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
        .align 8
4:
```

Рис. 5: Ассемблерный листинг на C++

3. Модификация листинга

Затем модифицируем ассемблерный листинг программы, написанной на C. Изменим “hello world” на “it was changed by hands”:

```
aobaks@WIN-851G51OH5DP:~/assembler_files$ ./lab0c
101
Hello Worldaobaks@WIN-851G51OH5DP:~/assembler_files$ rm a.out
aobaks@WIN-851G51OH5DP:~/assembler_files$ vim lab0c.s
aobaks@WIN-851G51OH5DP:~/assembler_files$ gcc lab0c.s -o asm_onC
aobaks@WIN-851G51OH5DP:~/assembler_files$ ls
asm_onC  c.s  cpp.s  hello_world.c  hello_world.s  lab0.c  lab0.cpp  lab0c  lab0c.s
aobaks@WIN-851G51OH5DP:~/assembler_files$ ./asm_onC
101
It was changed by handsaobaks@WIN-851G51OH5DP:~/assembler_files$
```

Рис. 6: Hello World изменилось

4. Базовые команды

mov – присвоение
add – сложение
sub – вычитание

Инструкция **add** выполняет сложение двух операндов, а инструкция **sub** - вычитание. Они имеют следующий синтаксис:

```
add destination, source
sub destination, source
```

Инструкция add складывает операнды destination и source и результат помещает в операнд destination. Инструкция sub вычитает из destination операнд source и результат вычитания помещает в операнд destination.

```
; для add
destination = destination + source
; для sub
destination = destination - source
```

Рис. 7: Базовые команды

5. Глобальные переменные

Глобальные переменные объявляются в самом начале ассемблерного листинга. В зависимости от типа переменной ей присваивается соответствующий размер.

```
        .file   "lab0.c"
        .text
        .globl  a
        .data
        .align 4
        .type   a, @object
        .size   a, 4
a:
        .long   5
        .globl  b
        .align 8
        .type   b, @object
        .size   b, 8
b:
        .quad   12
        .section        .rodata
```

Рис. 8: Объявление глобальной переменной int a и long int b

Long int переменная b добавлена для наглядности: int = 4, long int = 8.

6. Регистры процессора

Ключевую роль в обработке данных в процессоре играют специальные ячейки, известные как **регистры**. Регистры в процессоре x86-64 можно разделить на четыре категории: регистры общего назначения, специальные регистры для приложений, сегментные регистры и специальные регистры режима ядра. Здесь нас будут интересовать прежде всего **регистры общего назначения** (general-purpose registers), которые в основном и используются в приложениях на ассемблере.

Начнем с того, что процессор архитектуры x86 имел восемь 32-битных регистров общего назначения, регистр флагов и указатель инструкций. Регистры общего назначения:

- **EAX** (Accumulator): для арифметических операций
- **ECX** (Counter): для хранения счетчика цикла
- **EDX** (Data): для арифметических операций и операций ввода-вывода
- **EBX** (Base): указатель на данные
- **ESP** (Stack pointer): указатель на верхушку стека
- **EBP** (Base pointer): указатель на базу стека внутри функции
- **ESI** (Source index): указатель на источник при операциях с массивом
- **EDI** (Destination index): указатель на место назначения в операциях с массивами
- **EIP**: указатель адреса следующей инструкции для выполнения
- **EFLAGS**: регистр флагов, содержит биты состояния процессора

Можно получить доступ к частям 32-битных регистров с меньшей разрядностью. Например, младшие 16 бит 32-битного регистра EAX обозначаются как AX. К регистру AX можно обращаться как к отдельным байтам, используя имена AH (старший байт) и AL (младший байт).

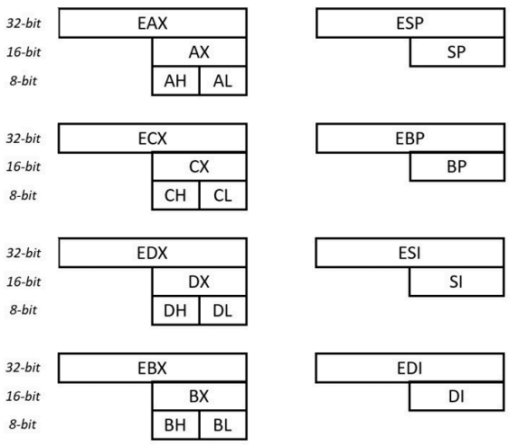

Рис. 9: К регистрам процессора

7. Функции mul, div, imul, idiv

Напишем программу на языке C++:

```cpp
#include <iostream>

int main(){
        int a = 14;
        int b = 7;
        int mul;
        int div;

        mul = a * b;
        div = a / b;

        std::cout << mul << div << std::endl;

        unsigned int ua = 21;
        unsigned int ub = 3;
        unsigned int umul;
        unsigned int udiv;

        umul = ua * ub;
        udiv = ua / ub;

        std::cout << umul << udiv << std::endl;

}
```

Рис. 10: Код программы

Затем откроем ее ассемблерный листинг:

```
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $32, %rsp
        movl    $14, -32(%rbp)
        movl    $7, -28(%rbp)
        movl    -32(%rbp), %eax
        imull   -28(%rbp), %eax
        movl    %eax, -24(%rbp)
        movl    -32(%rbp), %eax
        cltd
        idivl   -28(%rbp)
        movl    %eax, -20(%rbp)
        movl    -24(%rbp), %eax
        movl    %eax, %esi
        leaq    _ZSt4cout(%rip), %rax
        movq    %rax, %rdi
        call    _ZNSolsEi@PLT
        movq    %rax, %rdx
        movl    -20(%rbp), %eax
        movl    %eax, %esi
        movq    %rdx, %rdi
        call    _ZNSolsEi@PLT
        movq    _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_@GOTPCREL(%rip), %rdx
        movq    %rdx, %rsi
        movq    %rax, %rdi
        call    _ZNSolsEPFRSoS_E@PLT
        movl    $21, -16(%rbp)
        movl    $3, -12(%rbp)
        movl    -16(%rbp), %eax
        imull   -12(%rbp), %eax
        movl    %eax, -8(%rbp)
        movl    -16(%rbp), %eax
        movl    $0, %edx
        divl    -12(%rbp)
        movl    %eax, -4(%rbp)
        movl    -8(%rbp), %eax
        movl    %eax, %esi
        leaq    _ZSt4cout(%rip), %rax
        movq    %rax, %rdi
        call    _ZNSolsEj@PLT
        movq    %rax, %rdx
        movl    -4(%rbp), %eax
        movl    %eax, %esi
        movq    %rdx, %rdi
```

Рис. 11: Фрагмент ассемблерного листинга