ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
ФАКУЛЬТЕТ АЭРОКОСМИЧЕСКИХ ТЕХНОЛОГИЙ

Лабораторная 2
Продолжение базы

Рогозин Владимир
Группа Б03-106

Пункт 1:

Для начала напишем простейшую вставку и посмотрим как она выглядит в ассемблерном листинге. Видим, что наша вставка выделена специальными пометками, также присутствует нвзвание файла программы со вставкой.

```
asm("movq \tglobint1(%rip), %rax\n"
        "\taddq \t$1, %rax\n"
        "\tmovq \t%rax, globint1(%rip)\n" );
```

Рис. 1: Ассемблерная вставка в C

```
#APP
# 9 "operations.c" 1
        movq    globint1(%rip), %rax
        addq    $1, %rax
        movq    %rax, globint1(%rip)

# 0 "" 2
#NO_APP
```

Рис. 2: Ассемблерная вставка в листинге

Пункты 2, 3, 4:

В этих пунктах сами напишем метки и посмотрим на написанные компилятором, получим команды перехода и оператор сравнения. Для этого напишем вставку с меткой, сравнением, оператором перехода. В программе происходит прибавление единицы к переменной, которая изначально была равна 1, сравнение её значения с 3, выход из цикла если есть равенство. Затем, чтобы проверить корректность работы цикла, выводится значение переменной на экран.

```
int globint1 = 1;
int globint2 = 2;

int main() {
        asm(    "metka1:\n"
                "\tmovl $1, %ecx \n"
                "\taddl globint1(%rip), %ecx \n"
                "\tmovl %ecx, globint1(%rip) \n"
                "\tcmp  $3, globint1(%rip) \n"
                "\tjne  metka1\n");
        if (globint1 > 1)
                printf("%d\n", globint1);
        return 0;
}
```

Рис. 3: Вставка с меткой, оператором сравнения и оператором перехода

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.c -S -o operations.s
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.s  -o operations.out
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ./operations.out
3
```

Рис. 4: Результат работы программы выше

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
#APP
# 7 "operations.c" 1
        metka1:
        movl    $1, %ecx
        addl    globint1(%rip), %ecx
        movl    %ecx, globint1(%rip)
        cmp     $3, globint1(%rip)
        jne     metka1

# 0 "" 2
#NO_APP
        movl    globint1(%rip), %eax
        cmpl    $1, %eax
        jle     .L2
        movl    globint1(%rip), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
.L2:
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

Рис. 5: Ассемблерный листинг программы выше

Пункт 5:

Посмотрим как выглядят массивы данных на языке ассемблера и как происходит обращение к i-ому элементу.

```
int globint1 = 1;
int globint2 = 2;
int globmassive[3] = {0, 1, 2};
```

Рис. 6: Инициализация массива в C

```
        .globl  globmassive
        .align 8
        .type   globmassive, @object
        .size   globmassive, 12
globmassive:
        .long   0
        .long   1
        .long   2
```

Рис. 7: Инициализация массива в ассемблере

```
movl    4+globmassive(%rip), %eax
addl    $1, %eax
movl    %eax, 4+globmassive(%rip)
movl    $0, %eax
```

Рис. 8: Обращение к элементу массива в ассемблере

Видим, что массивы чисел хранятся как набор отдельных чисел единым именем, сразу указан размер, занимаемый массивом(в байтах). В нижнем кусочке кода идет обращение ко второму элементу массива ($i = 1$) и увеличение его на единицу. Получаем, что название массива указывает на нулевой элемент, все элементы массива лежат в памяти друг за другом, и чтобы обратиться к i-ому элементу нужно к адресу нулевого элемента(который совпадает с названием массива) прибавить $i \cdot sizeof(int)$ (int в общем случае надо заменить типом данных элементов массива).

Пункты 6, 7:

В этих пунктах получше разберёмся с командами перехода, рассмотрим условный и безусловный переходы.

```
int globint1 = 1;
int globint2 = 2;


int main() {
        asm(    "jmp    metkaTest\n"
                "\tmovl $1, %ecx \n"
                "\taddl globint1(%rip), %ecx \n"
                "\tmovl %ecx, globint1(%rip) \n"
                "metkaTest: \n");
        printf("%d\n", globint1);
        return 0;
}
```

(a) Безусловный переход

```
int globint1 = 1;
int globint2 = 2;


int main() {
        asm(    "cmp    $1, globint1(%rip)\n"
                "\tje   metkaTest\n"
                "\tmovl $10, %ecx \n"
                "\taddl globint1(%rip), %ecx \n"
                "\tmovl %ecx, globint1(%rip) \n"
                "metkaTest: \n");
        printf("%d\n", globint1);
        return 0;
}
```

(b) Условный переход

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.s  -o operations.out
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ./operations.out
1
```

Рис. 10: Результат выполнения программ с условным и безусловным переходами

В каждой из программ пропускается часть кода, где увеличивается значение переменной, вывод в командную строку показывает, что переменная осталась прежней, значит переходы сработали корректно.

Теперь, с помощью команд перехода, реализуем условный оператор(if) и операторы цикла($for, while, do--while$). Сначала посмотрим на if

```
int globint1 = 1;
int globint2 = 2;


int main()
{/*тоже самое, что и
        if (globint <= 1) {
                globint1++;
        } */
        asm(    "movl   $1, %eax \n"
                "\tcmpl globint1(%rip), %eax\n"
                "\tjle  if \n"
                "\tjmp  endOfIf \n"
                "if: \n"
                "\tincl globint1(%rip) \n"
                "endOfIf: \n"
                "\tmovl $0, %eax \n");
        printf("%d\n", globint1);
        return 0;
}
```

Рис. 11: Условный оператор с помощью вставки

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.c -S -o operations.s
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.s  -o operations.out
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ./operations.out
2
```

Рис. 12: Результат программы

Далее напишем цикл *for*

```
int globint1 = 1;
int globint2 = 2;


int main()
{/*тоже самое, что и
        for(int i = 0; i < 10; ++i) {
                globint1++;
        } */
        asm(    "movl   $0, %eax \n"
                "\tmovl globint1(%rip), %ecx \n"
                "cycle: \n"
                "\tcmpl $10, %eax\n"
                "\tjge  endOfCycle \n"
                "\tinc  %eax \n"
                "\tinc  %ecx \n"
                "\tjmp  cycle \n"
                "endOfCycle: \n"
                "\tmovl %ecx, globint1(%rip) \n"
                "\tmovl $0, %eax \n"
                "\tmovl $0, %ecx \n");
        printf("%d\n", globint1);
        return 0;
}
```

Рис. 13: Цикл *for* с помощью вставки

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.c -S -o operations.s
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.s  -o operations.out
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ./operations.out
11
```

Рис. 14: Результат программы

И, наконец, сделаем цикл $do - while$

```
int globint1 = 1;
int globint2 = 2;


int main()
{/*тоже самое, что и
        do {
                globint1++;
        } while ( globint1 < 5 )
        */
        asm(    "movl   globint1(%rip), %ecx \n"
                "do: \n"
                "\tincl %ecx \n"
                "\tcmpl $5, %ecx \n"
                "\tjl   do \n"
                "\tmovl %ecx, globint1(%rip) \n"
                "\tmovl $0, %ecx \n");
        printf("%d\n", globint1);
        return 0;
}
```

Рис. 15: Цикл $do - while$ с помощью вставки

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.c -S -o operations.s
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.s  -o operations.out
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ./operations.out
5
```

Рис. 16: Результат программы

Пункт 8:

В этом пункте рассмотрим оператор цикла *loop*, который смотрит на значение в регистре ecx, на каждой итерации уменьшает это значение на 1, и выходит из цикла при $\%ecx == 0$

```c
int globint1 = 1;
int globint2 = 2;


int main()
{/*Проверим оператор loop,
   программа должна посчитать сумму чисел от 1 до 20
        */
        asm(    "movl   $19, %ecx \n"
                "\tmovl $0, %eax \n"
                "sum: \n"
                "\taddl %ecx, %eax \n"
                "\tloop sum \n"
                "\tmovl %eax, globint1(%rip) \n"
                "\tmovl $0, %ecx \n"
                "\tmovl $0, %eax \n");
        printf("%d\n", globint1);
        return 0;
}
```

Рис. 17: Пробуем *loop*

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.c -S -o operations.s
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.s  -o operations.out
clear@DESKTOP-FOMMSSB:~/assembler_3sem$./operations.out
190
```

Рис. 18: Результат программы

Пункт 9:

С помощью условного опрератора найдём максимум из двух чисел. Всё сработало корректно, программа вывела действительно наибольшее из двух число.

```c
int globint1 = 123;
int globint2 = 34;
int globMaxElem;

int main()
{
        asm(    "movl   globint1(%rip), %ecx \n"
                "\tcmpl globint2(%rip), %ecx \n"
                "\tjg   firstBigger \n"
                "\tmovl globint2(%rip), %eax \n"
                "\tjmp  end \n"
                "firstBigger: \n"
                "\tmovl %ecx, %eax \n"
                "end: \n"
                "\tmovl %eax, globMaxElem(%rip) \n"
                "\tmovl $0, %ecx \n"
                "\tmovl $0, %eax \n");
        printf("%d\n", globMaxElem);
        return 0;
}
```

Рис. 19: Наибольшее из двух чисел

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.c -S -o operations.s
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.s  -o operations.out
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ./operations.out
123
```

Рис. 20: Наибольшее число из двух

Пункты 10, 11:

В этом пункте поработаем с массивом глобальных переменных, а точнее напишем вставку, находящую сумму элементов массива, затем найдем наибольший элемент массива.

```
int globMassive[5] = {11, 3, 4, -5, 89};
int globSum = 0;

int main()
{
        asm(    "movl   $0, %eax \n"
                "\tmovl globSum(%rip), %ecx \n"
                "\tleaq globMassive(%rip), %rbx \n"
                "Cycle: \n"
                "\tcmpl $5, %eax \n"
                "\tjge  EndOfCycle \n"
                "\taddl (%rbx), %ecx \n"
                "\taddq $4, %rbx \n"
                "\tincl %eax \n"
                "\tjmp  Cycle \n"
                "EndOfCycle: \n"
                "\tmovl %ecx, globSum(%rip) \n"
                "\tmovl $0, %ecx \n"
                "\tmovl $0, %eax \n");
        printf("%d\n", globSum);
        return 0;
}
```

Рис. 21: Нахождение суммы элементов в массиве

```
int globMassive[5] = {11, 3, 4, -5, 89};
int globMax;

int main()
{
        asm(    "movl   $0, %edx \n"
                "\tmovl globMassive(%rip), %ecx \n"
                "\tleaq globMassive(%rip), %rbx \n"
                "Cycle: \n"
                "\tcmpl $5, %edx \n"
                "\tjge  EndOfCycle \n"
                "\tcmpl (%rbx), %ecx \n"
                "\tjge  Less \n"
                "\tmovl (%rbx), %ecx \n"
                "Less: \n"
                "\taddq $4, %rbx \n"
                "\tincl %edx \n"
                "\tmovl %ecx, globMax(%rip) \n");
        asm(
                "jmp    Cycle \n"
                "EndOfCycle: \n"
                "\tmovl %ecx, globMax(%rip) \n"
                "\tmovl $0, %ecx \n"
                "\tmovl $0, %edx \n");
        printf("%d\n", globMax);
        return 0;
}
```

Рис. 22: Нахождение наибольшего элемента в массиве