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

Лабораторная 1
База + фундамент

Рогозин Владимир
Группа Б03-106

Пункты 3, 4, 5:

Сначала посмотрим как выглядит простейшая программа, которая выводит строку "Hello world"на языке ассемблера, сравним ассемблерный листинг простейшей программы на языке C с точно такой же программой на языке C++.

```
        .file   "test.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), %rdi
        call    puts@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 9.3.0-10ubuntu2) 9.3.0"
        .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:
```

(a) "Hello world"на C

```
        .file   "test.cpp"
        .text
        .section        .rodata
        .type   _ZStL19piecewise_construct, @object
        .size   _ZStL19piecewise_construct, 1
_ZStL19piecewise_construct:
        .zero   1
        .local  _ZStL8__ioinit
        .comm   _ZStL8__ioinit,1,1
.LC0:
        .string "Hello world"
        .text
        .globl  main
        .type   main, @function
main:
.LFB1522:
        .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), %rsi
        leaq    _ZSt4cout(%rip), %rdi
        call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@PLT
        movq    %rax, %rdx
        movq    _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_@GOTPCREL(%rip), %rax
        movq    %rax, %rsi
        movq    %rdx, %rdi
        call    _ZNSolsEPFRSoS_E@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1522:
        .size   main, .-main
        .type   _Z41__static_initialization_and_destruction_0ii, @function
_Z41__static_initialization_and_destruction_0ii:
.LFB2011:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movl    %esi, -8(%rbp)
"testcpp.s" 111L, 2297C
```

(b) "Hello world"на C++

Не трудно заметить, что имеются небольшие различия – на C++ добавилась дополнительная информация, программа стала объёмнее. Однако прослеживается схожесть во многих командах, к примеру строка "Hello world"объявляется в обоих языках одинаково. Ниже представлен код программы, которая создаёт три глобальных целочисленных переменных, двум из них присваиваются значения, а третьей присваивается произведение значений первых двух. Опять же, видны различия в объявлении переменных, но те части кода, где происходит присваивание и умножение абсолютно идентичны на C и C++.

```
        .file   "operations.c"
        .text
        .comm   glob1,4,4
        .comm   glob2,4,4
        .comm   glob3,4,4
        .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
        movl    $4, glob1(%rip)
        movl    $5, glob2(%rip)
        movl    glob1(%rip), %edx
        movl    glob2(%rip), %eax
        imull   %edx, %eax
        movl    %eax, glob3(%rip)
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 9.3.0-10ubuntu2) 9.3.0"
        .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:
```

(a) Умножение двух целых чисел на C

```
        .file   "operations.cpp"
        .text
        .section        .rodata
        .type   _ZStL19piecewise_construct, @object
        .size   _ZStL19piecewise_construct, 1
_ZStL19piecewise_construct:
        .zero   1
        .local  _ZStL8__ioinit
        .comm   _ZStL8__ioinit,1,1
        .globl  glob1
        .bss
        .align 4
        .type   glob1, @object
        .size   glob1, 4
glob1:
        .zero   4
        .globl  glob2
        .align 4
        .type   glob2, @object
        .size   glob2, 4
glob2:
        .zero   4
        .globl  glob3
        .align 4
        .type   glob3, @object
        .size   glob3, 4
glob3:
        .zero   4
        .text
        .globl  main
        .type   main, @function
main:
.LFB1522:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $4, glob1(%rip)
        movl    $5, glob2(%rip)
        movl    glob1(%rip), %edx
        movl    glob2(%rip), %eax
        imull   %edx, %eax
        movl    %eax, glob3(%rip)
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

(b) Умножение двух целых чисел на C++

Далее, добавим вывод на экран(чтобы проверять корректность работы программы), сложим два числа, запишем результат в третье, выведем его на экран. После этого попробуем изменить программу непосредственно через ассемблерный листинг, сравним результаты работы обеих программ. Ниже приведены скриншоты различий программ и вывода в консоль до/после.

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $4, glob1(%rip)
        movl    $5, glob2(%rip)
        movl    glob1(%rip), %edx
        movl    glob2(%rip), %eax
        addl    %edx, %eax
        movl    %eax, glob3(%rip)
        movl    glob3(%rip), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

(a) Сложение 4 с 5

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %r`p
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $44, glob1(%rip)
        movl    $5, glob2(%rip)
        movl    glob1(%rip), %edx
        movl    glob2(%rip), %eax
        addl    %edx, %eax
        movl    %eax, glob3(%rip)
        movl    glob3(%rip), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

(b) Сложение 44 с 5

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.s -o operations.out
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ./operations.out
9
```

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc operations.s -o operations.out
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ./operations.out
49
```

Таким образом, изменив в ассемблерном листинге значение, присваиваемое одному из чисел, поменяли значение третьей переменной и результат работы программы. Исходный файл operations.c при этом не изменился.

Пункт 6:
Теперь выясним какие команды отвечают за операции сложения, вычитания, присваивания.

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $4, glob1(%rip)
        movl    $5, glob2(%rip)
        movl    glob1(%rip), %edx
        movl    glob2(%rip), %eax
        addl    %edx, %eax
        movl    %eax, glob3(%rip)
        movl    glob3(%rip), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

(a) Команда addl складывает числа

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $4, glob1(%rip)
        movl    $5, glob2(%rip)
        movl    glob1(%rip), %edx
        movl    glob2(%rip), %eax
        subl    %eax, %edx
        movl    %edx, %eax
        movl    %eax, glob3(%rip)
        movl    glob3(%rip), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

(b) Команда subl вычитает числа

Не трудно увидеть, что за сложение отвечает команда ***addl***, за вычитание – ***subl***. Присваиваются значения с помощью команды ***movl***.

Пункт 7:
Создадим несколько глобальных переменных различного типа, посмотрим как они объявляются, проделаем некоторые операции с ними. Видно, что глобальные переменные объявляются в самом верху, рядом с названием указан размер переменной в байтах и ещё какое-то число. В данной простейшей программе присваиваются значения глобальным переменным, затем эти значения изменяются: переменной типа *char* присваивается другая буква, *int* складывается с *long* и результат записывается во *float*.

```
        .file   "operations.c"
        .text
        .comm   globint,4,4
        .comm   globchr,1,1
        .comm   globfloat,4,4
        .comm   globLong,8,8
        .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
        movq    $90000, globLong(%rip)
        movl    $1, globint(%rip)
        movb    $99, globchr(%rip)
        movss   .LC0(%rip), %xmm0
        movss   %xmm0, globfloat(%rip)
        movb    $97, globchr(%rip)
        movl    globint(%rip), %eax
        movslq  %eax, %rdx
        movq    globLong(%rip), %rax
        addq    %rdx, %rax
        cvtsi2ssq       %rax, %xmm0
        movss   %xmm0, globfloat(%rip)
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

Четыре глобальные переменные различных типов и действия с ними

Пункт 8:

Из предыдущей программы видно, что при выполнении арифметических действий значения различных по типу переменных копируются в различные регистры в зависимости от размера памяти, занимаемого переменной. Изначально переменная типа *int* кладётся в регистр *eax*, размер которого 32 бита, что совпадает с размером *int*'а, а переменная типа *long* в регистр *rax*, который содержит в себе 64 бита(как и размер переменной типа *long*). Перед тем как сложить *int* с *long* переменная типа *int* копируется в 64-битный регистр *rdx*, затем складывается с *long*'ом, потом результат сложения записывается в *rax*, после этого это число записывается в знаяение переменной типа *float*. Также стоит отметить, что результат арифметических операций записывается во второй регистр. Таким образом, обращение к регистру происходит через знак %, а именно %eax, также, если в регистре лежит адрес переменной, то к значению переменной можно обратиться сразу, добавив скобочки вот так вот (%eax)(эту инфу нагуглил). К тому же, для хранения различных данных, в зависимости от размера, используются различные регистры.

Пункт 9:

В этом пункте посмотрим на умножение и деление беззнаковых и знаковых чисел различных размеров. За знаковое деление и умножение отвечают ***imul*** и ***idiv*** соответственно. Внизу представлена программа с двумя числами типа *int*, в которой сначала во вторую переменную записывается результат их перемножения, а потом результат их деления:

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $1, globint1(%rip)
        movl    $4, globint2(%rip)
        movl    globint2(%rip), %edx
        movl    globint1(%rip), %eax
        imull   %edx, %eax
        movl    %eax, globint2(%rip)
        movl    globint2(%rip), %eax
        movl    globint1(%rip), %ecx
        cltd
        idivl   %ecx
        movl    %eax, globint2(%rip)
        movl    globint2(%rip), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

Умножение и деление целых знаковых чисел типа *int*

Видим, что у каждой из команд появилась буква *l* в конце, которая указывает на размер операндов. Обе переменных 32-х битные, поэтому используются регистры *edx*, *eax* и *ecx*. У ***imul*** два аргумента, результат записывается во второй регистр(*eax* в данном случае). У команды ***idiv*** всего один аргумент и, судя по действиям перед и после деления, команда ***idiv*** берёт делимое из регистра *eax* по умолчанию, а делитель – из единственного аргумента, и сохраняет результат в *eax*. Теперь сделаем те же арифметические операции с беззнаковыми числами:

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $1, globint1(%rip)
        movl    $4, globint2(%rip)
        movl    globint2(%rip), %edx
        movl    globint1(%rip), %eax
        imull   %edx, %eax
        movl    %eax, globint2(%rip)
        movl    globint2(%rip), %eax
        movl    globint1(%rip), %ecx
        movl    $0, %edx
        divl    %ecx
        movl    %eax, globint2(%rip)
        movl    globint2(%rip), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

Умножение и деление целых беззнаковых чисел типа *int*

Можем видеть, что вместо команды ***idivl*** появилась ***divl***, которая отвечает за деление беззнаковых чисел, при этом умножаются числа по-прежнему командой ***imull***, результат умножения все так же записывается во второй аргумент(регистр), операция ***divl*** по-прежнему берет делимое из регистра *eax*, и туда же записывает результат деления. Делитель берётся из регистра *ecx*.

Теперь в качестве типов переменных возьмём *long long* и *unsigned long long*. Видно, что поменялись только последние буквы в командах ***imulq***, ***idivq*** и ***divq***. Также, так как изменился размер переменной, то изменились и регистры, в которых хранятся результаты операций и значения самих переменных. Команду ***mul*** так и не удалось вытащить, но работает она схожим с ***div*** и ***idiv*** образом(в том смысле, что принимает один аргумент, а второй сомножитель заведомо находится в определенном регистре).

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    $1, globint1(%rip)
        movq    $4, globint2(%rip)
        movq    globint2(%rip), %rdx
        movq    globint1(%rip), %rax
        imulq   %rdx, %rax
        movq    %rax, globint2(%rip)
        movq    globint2(%rip), %rax
        movq    globint1(%rip), %rcx
        cqto
        idivq   %rcx
        movq    %rax, globint2(%rip)
        movq    globint2(%rip), %rax
        movq    %rax, %rsi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

(a) Умножение и деление целых знаковых чисел типа *long long*

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    $1, globint1(%rip)
        movq    $4, globint2(%rip)
        movq    globint2(%rip), %rdx
        movq    globint1(%rip), %rax
        imulq   %rdx, %rax
        movq    %rax, globint2(%rip)
        movq    globint2(%rip), %rax
        movq    globint1(%rip), %rcx
        movl    $0, %edx
        divq    %rcx
        movq    %rax, globint2(%rip)
        movq    globint2(%rip), %rax
        movq    %rax, %rsi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

(b) Умножение и деление целых беззнаковых чисел типа *long long*