«Национальный исследовательский университет
«Московский институт электронной техники»

Лабораторная работа №2 по дисциплине «Архитектура вычислительных систем»

Отладка кода

Выполнили студены группы МП-25
Саядян Артём Грачикович
Калинкин Никита Анатольевич
Констандогло Александр Витальевич

Москва 2017

Вариант №2

Задание 1. Разработайте программу на языке C++, вычисляющую три целых выражения от целого аргумента (в соответствии с вариантом).

а) $y(x) = -x - 1$

б) $y(x) = x * 13$

в) $y(x) = \begin{cases} 0, x < 7 \\ x, x \geqslant 7 \end{cases}$

Задание 2. Запустите программу и, используя инструменты отладчика (в частности, дизассемблер), изучите ассемблерный код, соответствующий вычислениям.

Занесите ассемблерный код, соответствующий вычислению *y(x)*, в отчёт (код, не связанный с вычислением *y(x)*, копировать в отчёт не нужно!). Определите и прокомментируйте:

– обращение к переменным *x* и *y*;

– арифметические и логические операции — сложение, вычитание, умножение, деление с остатком, деление на 2^n и т. д. (по возможности);

– сравнения и передачу управления в ветвлениях.

Результат:

Platform: Linux Ubuntu 16.04.3 LTS x86_64
Compiler: GNU GCC 5.3.1
IDE: Qt Creator 4.3.1

а) y = -x-1;

```
mov    -0x14(%rbp),%eax     // eax (32 бита) = *(rbp – 0x14) (bp адресует переменные, хранимые в стеке, 64 бита, так как это длина слова, в данном случае из стека извлекается x)
not    %eax                 // побитовое отрицание x, это и есть -x-1
mov    %eax,-0x4(%rbp)      // *(rbp – 0x4) = eax – запись результата обратно в стек
```

б) y = x*13;

```
mov    -0x14(%rbp),%edx // edx = *(rbp – 0x14) (извлекается значение x)
mov    %edx,%eax        // eax = edx  (x)
add    %eax,%eax        // eax = eax + eax  (2x)
add    %edx,%eax        // eax = eax + edx  (3x)
shl    $0x2,%eax        // eax = eax << 0x2 логический сдвиг влево на 2 (12x)
add    %edx,%eax        // eax = eax + edx  (13x – получили то, что нужно)
mov    %eax,-0x4(%rbp)  // *(rbp – 0x4) = eax – запись результата обратно в стек
```

в) y = (x<7)?0:x;

```
cmpl   $0x6,-0x14(%rbp)        // сраниваются значения 0x6 и *(rbp-0x14)(x)
jle    0x400709 <ex1c(int)+18> // если текущий элемент массива (*(rbp-0x14), то есть значение x) меньше или равен 6, переходим на нужный участок памяти
mov    -0x14(%rbp),%eax        // eax = *(rbp – 0x14)извлекаем x в регистр A (если x<=6, сюда не доходим)
jmp    0x40070e <ex1c(int)+23> // безусловно переходим на нужный участок
```

```
(если x<=6, сюда не доходим)
mov    $0x0,%eax               // 0x400709 eax = 0 (если не(x<=6), сюда не
доходим)
mov    %eax,-0x4(%rbp)         // 0x40070e *(rbp – 0x4) = eax – запись
результата обратно в стек
```

Platform: macOS 10.12.6 (16G29)
Compiler: Apple LLVM version 9.0.0 (clang-900.0.37)
IDE: Xcode Version 9.0 (9A235)

а) *y = -x - 1;*

```
xorl   %eax, %eax          // Обнуляется значение в eax
movl   %edi, -0x4(%rbp)    // *(rdp – 0x4) = edi(x)
subl   -0x4(%rbp), %eax    // Отнимаем x от 0
subl   $0x1, %eax          // отнимаем 1
movl   %eax, -0x8(%rbp)    // Записываем обратно в стек
movl   -0x8(%rbp), %eax
```

б) *y = x * 13;*

```
movl   %edi, -0x4(%rbp)        // *(rdp – 0x4) = edi(x)
imull  $0xd, -0x4(%rbp), %edi  // edi = 0xd * *(rdp – 0x4)
movl   %edi, -0x8(%rbp)        // запись обратно в стек
movl   -0x8(%rbp), %eax
```

в) *y = (x<7) ? 0 : x;*

```
cmpl   $0x7, -0x4(%rbp)        // сравниваем
jge    0x100000d7b               ; <+27> at main.cpp:24 // если x больше или
равно, то переходим по адресу
xorl   %eax, %eax              // получаем 0
movl   %eax, -0xc(%rbp)        // и присваиваем 0
jmp    0x100000d81               ; <+33> at main.cpp:24 // безусловный
переход на нужный участок
movl   -0x4(%rbp), %eax        // 0x100000d7b <+27> помещаем x
movl   %eax, -0xc(%rbp)        // Записываем результат обратно в стек
```

Visual C++ 32-bit (Microsoft Visual Studio 2015)

а) *y = -x - 1;*

```
mov         eax,dword ptr [x]  // В регистр A записывается операнд x размером
в 2 слова (2 * 16 бит)
neg         eax                // Знак числа в A меняется на противоположный
sub         eax,1              // eax -= 1
mov         dword ptr [y],eax  // В y записывается значение eax
```

б) *y = x * 13;*

```
imul        eax,dword ptr [x],0Dh  // eax = x * 0xD(13)
mov         dword ptr [y],eax      // В y записывается значение eax
```

в) *y = (x<7) ? 0 : x;*

```
cmp         dword ptr [x],7          // Сравниваются x и 7
jge         ex1c+30h (0CD19A0h)      // Перейти на указанный адрес, если x>=7
mov         dword ptr [ebp-0D0h],0   // *(ebp-0xD0) = 0 (32 бита)
jmp         ex1c+39h (0CD19A9h)      // Безусловный переход по адресу
```

```
mov         eax,dword ptr [x]        // 00CD19A0 eax = x
mov         dword ptr [ebp-0D0h],eax // *(ebp-0xD0) = eax
mov         ecx,dword ptr [ebp-0D0h] // 00CD19A9 ecx = *(ebp-0xD0) (результат)
mov         dword ptr [y],ecx
```

Visual C++ 64-bit (Microsoft Visual Studio 2015)

а) б) Аналогично 32-битной версии.

в) *y = (x<7) ? 0 : x;*

```
cmp         dword ptr [x],7
jge         ex1c+3Bh (07FF69A3C199Bh) // Размер адреса увеличился
mov         dword ptr [rbp+0D4h],0    // *(rbp-0xD4) = 0 (64 бита)
jmp         ex1c+47h (07FF69A3C19A7h)
mov         eax,dword ptr [x]         // 00007FF69A3C199B
mov         dword ptr [rbp+0D4h],eax
mov         eax,dword ptr [rbp+0D4h]  // 00007FF69A3C19A7
mov         dword ptr [y],eax         // Регистр C не используется
```

Задание 3. Внесите в программу из задания 1, а) изменения (либо, что предпочтительнее, добавьте новые фрагменты кода, выполняющие аналогичные вычисления для других переменных, используя макросы препроцессора или шаблоны C++).

- сделайте переменные глобальными;
- измените тип с int на char, short, long и long long;
- измените тип с int на long double.

Опишите в отчёте различия в ассемблерном коде.

Результат:

- сделайте переменные глобальными;

Platform: Linux Ubuntu 16.04.3 LTS x86_64
Compiler: GNU GCC 5.3.1
IDE: Qt Creator 4.3.1

```
Y = -X-1; // X и Y – глобальные переменные
lea    0x2008ae(%rip),%rax        # 0x601040 <X> // rax = *(rip + 0x2008ae)
- глобальные переменные расположены в конкретном участке памяти (адрес-
смещение следующей команды (64 бита))

mov    (%rax),%eax          // eax = rax (сужаем область до размера int
(младшие байты))
not    %eax                 // вычисление
mov    %eax,%edx            // edx = eax
lea    0x2008a9(%rip),%rax        # 0x601048 <Y> // rax = *(rip + 0x2008a9)
mov    %edx,(%rax)          // rax = edx – запись результата
```

Platform: macOS 10.12.6 (16G29)
Compiler: Apple LLVM version 9.0.0 (clang-900.0.37)
IDE: Xcode Version 9.0 (9A235)

```
    Y = -X - 1;
movl   $0x1, %edi                    // Записываем 1
xorl   %ecx, %ecx                    // Создаём 0
subl   0x208(%rip), %ecx         ; X   // Вычитаем X из 0
subl   $0x1, %ecx                    // Вычитаем 1
movl   %ecx, 0x203(%rip)           ; Y // Записываем в Y
```

Visual C++ 32-bit (Microsoft Visual Studio 2015)

```
    Y = -X - 1;
mov         eax,dword ptr [X (0CD9000h)]
neg         eax
sub         eax,1
mov         dword ptr [Y (0CD9148h)],eax // Обращение к глобальным переменным
по участку памяти
```

Visual C++ 64-bit (Microsoft Visual Studio 2015)

```
    Y = -X - 1;
mov         eax,dword ptr [X (07FF69A3CC000h)]
neg         eax
dec         eax
mov         dword ptr [Y (07FF69A3CC170h)],eax
// Размеры адресов увеличились
```

- измените тип с int на char, short, long и long long;

Platform: Linux Ubuntu 16.04.3 LTS x86_64
Compiler: GNU GCC 5.3.1
IDE: Qt Creator 4.3.1

```
char:   y = -x-1;
movzbl -0x14(%rbp),%eax  // чтение байта по адресу *(rbp – 0x14) и его
расширение до 32 бит путём добавления нулей в 3 старших байта
not    %eax
mov    %al,-0x1(%rbp)    // *(rbp – 0x1) = al – запись младшего байта
регистра eax обратно в стек

short:   y = -x-1;
movzwl -0x14(%rbp),%eax  // чтение 2 байт по адресу *(rbp – 0x14) и их
расширение до 32 бит путём добавления нулей в 2 старших байта
not    %eax
mov    %ax,-0x2(%rbp)    // *(rbp – 0x2) = ax – запись младших 2 байт
регистра eax обратно в стек

long:   y = -x-1;    // Здесь используются все 64 бита регистра A
mov    -0x18(%rbp),%rax
not    %rax
mov    %rax,-0x8(%rbp)
```

```
long long:   y = -x-1;       // как long
```

Platform: macOS 10.12.6 (16G29)
Compiler: Apple LLVM version 9.0.0 (clang-900.0.37)
IDE: Xcode Version 9.0 (9A235)

```
char:   y = -x-1;
movb   %dil, %al
xorl   %edi, %edi          // 0
movb   %al, -0x1(%rbp)
movsbl -0x1(%rbp), %ecx    // знаковое расширение
subl   %ecx, %edi          // edi = 0 - ecx (= -x)
subl   $0x1, %edi          // - 1
movb   %dil, %al           // в регистр A
movb   %al, -0x2(%rbp)     // в стек

short:   y = -x-1;
xorl   %edi, %edi         // 0
movw   %ax, -0x2(%rbp)
movswl -0x2(%rbp), %ecx   // приведение к int
subl   %ecx, %edi         // -x
subl   $0x1, %edi         // -x - 1
movw   %di, %ax           // берём 16 бит
movw   %ax, -0x4(%rbp)
movswl -0x4(%rbp), %eax

long:   y = -x-1;
xorl   %eax, %eax
movl   %eax, %ecx
movq   %rdi, -0x8(%rbp)       // В отличие от int используется rdi и все 64
бита в регистрах A и C
subq   -0x8(%rbp), %rcx       // -x
subq   $0x1, %rcx             // -x - 1
movq   %rcx, -0x10(%rbp)      // обратно в стек
movq   -0x10(%rbp), %rax

long long:   y = -x-1;    // Почти то же, что и long
xorl   %rcx, %rcx
movq   %rdi, -0x8(%rbp)
subq   -0x8(%rbp), %rcx
subq   $0x1, %rcx
movq   %rcx, -0x10(%rbp)
```

Visual C++ 32-bit (Microsoft Visual Studio 2015)

```
char:   y = -x-1;
movsx       eax,byte ptr [x]  // eax = x, при этом 8-битовый операнд
преобразуется в 32-битный путём знакового расширения
neg         eax
sub         eax,1
mov         byte ptr [y],al  // Младший байт регистра A переходит в y

short:   y = -x-1;
movsx       eax,word ptr [x]
neg         eax
sub         eax,1
```

```
mov         word ptr [y],ax // Младшее «слово» регистра A переходит в y

long:   y = -x-1;    // Как int

long long:   y = -x-1;
mov         eax,dword ptr [x]
neg         eax                         // eax = -(32 младших бита)x
mov         ecx,dword ptr [ebp+0Ch]     // ecx = *(ebp + 12 байт)
adc         ecx,0                       // ecx = ecx + 0 + CF (бит переноса)
neg         ecx                         // ecx = -ecx
sub         eax,1
sbb         ecx,0                       // ecx = ecx – (0 + CF)
mov         dword ptr [y],eax           // Младшие 32 бита
mov         dword ptr [ebp-8],ecx       // Старшие 32 бита
```

Visual C++ 64-bit (Microsoft Visual Studio 2015)

```
char:   y = -x-1;
movsx       eax,byte ptr [x]
neg         eax
dec         eax               // Декремент вместо sub
mov         byte ptr [y],al

short: и long:   Различие только в использовании декремента вместо sub

long long:   y = -x-1;
y = -x - 1;
mov         rax,qword ptr [x]
neg         rax
dec         rax                   // Декремент вместо sub
mov         qword ptr [y],rax
// В связи с наличием 64-битных регистров упростились операции над 64-битными
целыми числами
```

– измените тип с int на long double.

Platform: Linux Ubuntu 16.04.3 LTS x86_64
Compiler: GNU GCC 5.3.1
IDE: Qt Creator 4.3.1

```
long double:   y = -x-1;
fldt   0x10(%rbp)  // загрузить временное вещественное (или, как его ещё
называют, long double) в вершину стека из *(rbp + 0x10)
fchs               // изменить знак вершины стека
fld1               // загрузка 1 в вершину стека (1 является константой)
fsubrp %st,%st(1)  // st(1) = st(0) - st(1)
fstpt  -0x10(%rbp) // Переместить значение из st(0) в *(rdp – 0x10) ???
(должен получиться 0) ???
```

Platform: macOS 10.12.6 (16G29)
Compiler: Apple LLVM version 9.0.0 (clang-900.0.37)
IDE: Xcode Version 9.0 (9A235)

```
long double:   y = -x-1;
movq   %rsp, %rbp
```

```
fldt   0x10(%rbp)     // Загрузка long double в вершину стека
fstpt  -0x10(%rbp)     // Сохранение вершины стека в память с выталкиванием
fldt   -0x10(%rbp)     // Загрузка операнда в вершину стека
fld1                   // Загрузка 1 в вершину стека (1 является константой)
fchs                   // Изменение знака
fsubp  %st(1)          // Вычитание вещественное реверсивное с выталкиванием
fstpt  -0x20(%rbp)     // Сохранить вершину стека
fldt   -0x20(%rbp)     // загрузить long double в вершину
```

Visual C++ 32-bit (Microsoft Visual Studio 2015)

```
long double:   y = -x-1;
movsd       xmm0,mmword ptr [x]   // xmm0 = x (запись двойного слова)
xorps       xmm0,xmmword ptr [__xmm@80000000000000008000000000000000
(0CD6B50h)]                       // меняется знак числа путём применения
операции xor, котрая влияет только на знаковый бит
subsd       xmm0,mmword ptr [__real@3ff0000000000000 (0CD6B40h)] //
вычисление разности
movsd       mmword ptr [y],xmm0  // Запись результата
```

Visual C++ 64-bit (Microsoft Visual Studio 2015)

```
long double:   Размеры адресов увеличились, принцип вычисления не изменился
```

Задание 4. Оформите вычисления из задания 1, а) как целую функцию от целого аргумента. Опишите в отчёте код вызова функции. Как передаётся аргумент? Как возвращается значение?

Результат:

Platform: Linux Ubuntu 16.04.3 LTS x86_64
Compiler: GNU GCC 5.3.1
IDE: Qt Creator 4.3.1

```
    ex1a(1); // Вызов
mov    $0x1,%edi               // Передача аргумента: edi = 0x1
callq  0x4006c6 <ex1a(int)>    // Передача управления функции с запоминанием
в стеке адреса точки возврата

    return y; // Возврат значения
mov    -0x4(%rbp),%eax         // В регистр общего назначения A записываются
данные из стека
```

Platform: macOS 10.12.6 (16G29)
Compiler: Apple LLVM version 9.0.0 (clang-900.0.37)
IDE: Xcode Version 9.0 (9A235)

```
    ex1a(1);
movl   $0x0, -0x4(%rbp)
callq  0x100000d20               ; ex1a at main.cpp:8 // Передача управления
функции с запоминанием в стеке адреса точки возврата

    return y;
movl   -0x8(%rbp), %eax
```

```
popq   %rbp         // Достаём значение из стека
retq                // Переброс к адресу возвращения
```

Visual C++ 32-bit (Microsoft Visual Studio 2015)

```
    ex1a(1);
push        1                 // Разместить значение в стеке
call        ex1a (0FA1271h)   // Передача управления функции с запоминанием в
стеке адреса точки возврата
add         esp,4             // Сдвиг вершины стека на 4 байта

      return y;
mov         eax,dword ptr [y]
```

Visual C++ 64-bit (Microsoft Visual Studio 2015)

```
    ex1a(1);
mov         ecx,1                 // Передача аргумента в регистр C
call        ex1a (07FF69A3C1267h) // Передача управления функции с
запоминанием в стеке адреса точки возврата

      return y; // Изменений нет
```

Задание 5. Измените тип аргумента и результата на вещественный. Опишите в отчёте код вызова функции. Как передаётся аргумент? Как возвращается значение?

Результат:

Platform: Linux Ubuntu 16.04.3 LTS x86_64
Compiler: GNU GCC 5.3.1
IDE: Qt Creator 4.3.1

```
     ex5(999E-8);  // Вызов
movabs $0x3ee4f357252adccd,%rax // Запись 64-битового значения в регистр A
mov    %rax,-0x18(%rbp)         // *(rbp – 0x18) = rax
movsd  -0x18(%rbp),%xmm0        // Пересылка данных из *(rbp – 0x18) в 128-
битный SSE регистр xmm0
callq  0x400716 <ex5(double)>   // Передача управления функции с запоминанием
в стеке адреса точки возврата

    return y;       // Возврат значения
movsd  -0x8(%rbp),%xmm0            // В регистр xmm0 записываются данные из
стека
```

Platform: macOS 10.12.6 (16G29)
Compiler: Apple LLVM version 9.0.0 (clang-900.0.37)
IDE: Xcode Version 9.0 (9A235)

```
      ex5(999E-8);
fstpt  -0x3c(%rbp)
callq  0x100000d90              ; ex5 at main.cpp:44  // Передача управления
функции с запоминанием в стеке адреса точки возврата
```

```
      return y;
movsd  -0x10(%rbp), %xmm0           // В регистр xmm0 записываются данные из
стека, но смещение в 2 раза больше, чем в предыдущем случае
```

Visual C++ 32-bit (Microsoft Visual Studio 2015)

```
      ex5(999E-8);
sub         esp,8
movsd       xmm0,mmword ptr [__real@3ee4f357252adccd (0A66B30h)]
movsd       mmword ptr [esp],xmm0  // Помещение значения регистра с
аргументом в вершину стека
call        ex5 (0A61023h)         // Вызов функции
fstp        st(0)                  // Сохранить вещественное значение в
вершине стека
add         esp,8

      return y;
fld         qword ptr [y]          // загрузить вещественное значение в стек
```

Visual C++ 64-bit (Microsoft Visual Studio 2015)

```
      ex5(999E-8);
movsd       xmm0,mmword ptr [__real@3ee4f357252adccd (07FF69A3C9BB0h)]
call        ex5 (07FF69A3C1019h) // Значительное сокращение числа операций
при вызове функции

      return y;
movsd       xmm0,mmword ptr [y]  // Помещение значения y в регистр
```

Задание 6. Используйте в функции статическую переменную. Как выглядит обращение к ней?

Результат:

Platform: Linux Ubuntu 16.04.3 LTS x86_64
Compiler: GNU GCC 5.3.1
IDE: Qt Creator 4.3.1

```
   y = -x-1; // y – статическая переменная
mov    %eax,0x2008f6(%rip)        # 0x601050 <_ZZ3ex6iE1y>
// К статическим переменным обращение происходит как к глобальным, так как
это и есть глобальные переменные для данной функции
```

Platform: macOS 10.12.6 (16G29)
Compiler: Apple LLVM version 9.0.0 (clang-900.0.37)
IDE: Xcode Version 9.0 (9A235)

```
      y = -x - 1;
movl   %eax, 0x25b(%rip)
```

Visual C++ 32-bit (Microsoft Visual Studio 2015)

```
      y = -x - 1;
```

```
mov         dword ptr [y (0A6914Ch)],eax
// Разница только в синтаксисе, принцип обращения такой же
```

Visual C++ 64-bit (Microsoft Visual Studio 2015)

```
      y = -x - 1;
mov         dword ptr [y (07FF7AB15C174h)],eax
// Разница только в длине адреса, принцип обращения такой же
```

Задание 7. Запустите тестовую программу (программы), используя платформу и/или компилятор, отличные от GNU/Linux и GCC. Результаты с пояснениями внести в конспект.