**Лабораторная работа № 4**

Тема: Основные логические операции над целыми числами.

Интерфейс с языками высокого уровня.

1. Цель работы

Ознакомиться с логическими операциями над целочисленными данными, ознакомиться с правилами оформления ассемблерных процедур, в том числе вызываемых из Си-программ.

2. Краткие теоретические сведения

Следующий пример исходного кода иллюстрирует использование команд x86-64 and (логическое И), or (логическое включающее ИЛИ) и xor (логическое исключающее ИЛИ). Он также показывает, как получить доступ к глобальной переменной C++ из функции на языке ассемблера.

; Вычисление (((a & b) | c ) ^ d) + g\_Val1

and ecx,edx ;ecx = a & b

or ecx,r8d ;ecx = (a & b) | c

xor ecx,r9d ;ecx = ((a & b) | c) ^ d

add ecx,[g\_Val1] ;ecx = (((a & b) | c) ^ d) + g\_Val1

//------------------------------------------------

// Ch02\_02.cpp

//------------------------------------------------

#include "stdafx.h"

#include <iostream>

#include <iomanip>

using namespace std;

extern "C" unsigned int IntegerLogical\_(unsigned int a, unsigned int b, unsigned int c,

unsigned int d);

extern "C" unsigned int g\_Val1 = 0;

unsigned int IntegerLogicalCpp(unsigned int a, unsigned int b, unsigned int c, unsigned int

d)

{

// Вычисление (((a & b) | c ) ^ d) + g\_Val1

unsigned int t1 = a & b;

unsigned int t2 = t1 | c;

unsigned int t3 = t2 ^ d;

unsigned int result = t3 + g\_Val1;

return result;

}

void PrintResult(const char\* s, unsigned int a, unsigned int b, unsigned int c, unsigned

int

d, unsigned val1, unsigned int r1, unsigned int r2)

{

const int w = 8;

const char nl = '\n';

cout << s << nl;

cout << setfill('0');

cout << "a = 0x" << hex << setw(w) << a << " (" << dec << a << ")" << nl;

cout << "b = 0x" << hex << setw(w) << b << " (" << dec << b << ")" << nl;

cout << "c = 0x" << hex << setw(w) << c << " (" << dec << c << ")" << nl;

cout << "d = 0x" << hex << setw(w) << d << " (" << dec << d << ")" << nl;

cout << "val1 = 0x" << hex << setw(w) << val1 << " (" << dec << val1<< ")" << nl;

cout << "r1 = 0x" << hex << setw(w) << r1 << " (" << dec << r1 << ")" << nl;

cout << "r2 = 0x" << hex << setw(w) << r2 << " (" << dec << r2 << ")" << nl;

cout << nl;

if (r1 != r2)

cout << "Ошибка сравнения" << nl;

}

int main()

{

unsigned int a, b, c, d, r1, r2 = 0;

a = 0x00223344;

b = 0x00775544;

c = 0x00555555;

d = 0x00998877;

g\_Val1 = 7;

r1 = IntegerLogicalCpp(a, b, c, d);

r2 = IntegerLogical\_(a, b, c, d);

PrintResult("Пример 1", a, b, c, d, g\_Val1, r1, r2);

a = 0x70987655;

b = 0x55555555;

c = 0xAAAAAAAA;

d = 0x12345678;

g\_Val1 = 23;

r1 = IntegerLogicalCpp(a, b, c, d);

r2 = IntegerLogical\_(a, b, c, d);

PrintResult("Пример 2", a, b, c, d, g\_Val1, r1, r2);

return 0;

}

;-------------------------------------------------

; Ch02\_02.asm

;-------------------------------------------------

; extern "C" unsigned int IntegerLogical\_(unsigned int a, unsigned int b, unsigned int c,

unsigned int d);

2.1. Простая целочисленная арифметика  47

extern g\_Val1:dword ;внешнее значение doubleword (32-бит)

.code

IntegerLogical\_ proc

; Вычисление (((a & b) | c ) ^ d) + g\_Val1

and ecx,edx ;ecx = a & b

or ecx,r8d ;ecx = (a & b) | c

xor ecx,r9d ;ecx = ((a & b) | c) ^ d

add ecx,[g\_Val1] ;ecx = (((a & b) | c) ^ d) + g\_Val1

mov eax,ecx ;eax = окончательный результат

ret ;возвращает результат вызывающей функции

IntegerLogical\_ endp

end

Здесь объявление функции на языке ассемблера

IntegerLogical\_ использует модификатор "C", чтобы указать компилятору C++

не генерировать расширенное имя для этой функции. Отсутствие данного модификатора приведет к ошибке линкера во время сборки программы. (Если

модификатор "C" в текущем примере убрать, Visual C++ 2017 сгенерирует расширенное имя функции ?IntegerLogical \_@@YAIIIII@Z вместо IntegerLogical\_.

Расширенные имена выводятся с использованием типов аргументов функции, и эти имена зависят от компилятора.) Функция IntegerLogical\_ ожидает

четыре аргумента unsigned int и возвращает один результат unsigned int. Сразу

после объявления функции IntegerLogical\_ следует определение глобальной

переменной типа unsigned int с именем g\_Val1. Эта переменная определена для

демонстрации доступа к глобальному значению из функции на языке ассемблера. Как и объявления функций, использование модификатора "C" для g\_Val1

указывает компилятору использовать именование в стиле C вместо расширенного имени C++.

Далее в исходном коде C++ следует определение функции IntegerLogicalCpp.

Назначение этой функции заключается в проверке того, вычисляет ли соответствующая функция на языке ассемблера x86-64 IntegerLogical\_ правильный

результат. Хотя в этом конкретном примере такая проверка на самом деле

не нужна, параллельное кодирование сложных функций с использованием как

C++, так и ассемблера часто бывает полезным для тестирования и отладки программного обеспечения. Функция main в листинге 2.2 содержит код, который

вызывает как IntegerLogicalCpp, так и IntegerLogical\_. Он также вызывает функцию PrintResult для отображения результатов.

В листинге код на языке ассемблера x86-64 следует сразу после функции C++ main. Первый оператор исходного кода на языке ассемблера,

extern g\_Val1:dword,

является эквивалентом MASM соответствующего объявления для g\_Val1, которое используется в коде C++. В этом случае директива extern уведомляет ассемблер о том, что место для хранения переменной g\_Val1 определено в другом модуле, а директива dword указывает, что g\_Val1 является двойным

(или 32-битным) словом без знака.

Аргументы a, b, c и d передаются функции IntegerLogical\_ через регистры ECX, EDX, R8D и R9D. Команда and ecx,edx выполняет побитовую операцию И над значениями в регистрах ECX и EDX и сохраняет результат в регистре ECX. Команды or ecx,r8d и xor ecx,r9d

выполняют побитовые операции включающего ИЛИ и исключающего ИЛИ соответственно. Команда add ecx,[g\_Val1] складывает содержимое регистра ECX и значение глобальной переменной g\_Val1 и сохраняет полученную сумму в регистре ECX. Команда mov eax,ecx копирует окончательный результат в регистр EAX, чтобы его можно было передать обратно вызывающей функции. Вот результат выполнения примера Ch02\_02:

Пример 1

a = 0x00223344 (2241348)

b = 0x00775544 (7820612)

c = 0x00555555 (5592405)

d = 0x00998877 (10061943)

val1 = 0x00000007 (7)

r1 = 0x00eedd29 (15654185)

r2 = 0x00eedd29 (15654185)

Пример 2

a = 0x70987655 (1889039957)

b = 0x55555555 (1431655765)

c = 0xaaaaaaaa (2863311530)

d = 0x12345678 (305419896)

val1 = 0x00000017 (23)

r1 = 0xe88ea89e (3901663390)

r2 = 0xe88ea89e (3901663390)