Содержание:

РАБОТА 1.
СИСТЕМЫ СЧИСЛЕНИЯ

Цель работы: изучение позиционных систем счисления.
Приборы и принадлежности: компьютер.

1.1. Используемые системы счисления

Система счисления – символический метод записи чисел или способ представления чисел с помощью письменных знаков, именуемых цифрами.

Число – это некоторая абстрактная сущность для описания количества чего-либо.

Цифры – это знаки, используемые для записи чисел.

Поскольку чисел гораздо больше, чем цифр, то для записи числа обычно используется набор (комбинация) цифр.

Существует много способов записи чисел с помощью цифр. Каждый такой способ называется системой счисления. Величина числа может зависеть от порядка цифр в записи, а может и не зависеть. Это свойство определяется системой счисления и служит основанием для простейшей классификации таких систем.

Указанное основание позволяет все системы счисления разделить на три класса (группы): позиционные, непозиционные и смешанные.

Примером «чисто» непозиционной системы счисления является римская система, а смешанной – денежная система единиц.

Позиционные системы счисления – это системы счисления, в которых значение цифры напрямую зависит от ее позиции в числе. Например, число 01 обозначает единицу, 10 – десять. Позиционные системы счисления позволяют легко производить арифметические расчеты.

Представление чисел с помощью арабских цифр – самая распространенная позиционная система счисления, она называется «десятичной системой счисления». Десятичной системой она называется потому, что использует десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8 и 9.

Для составления машинных кодов удобно использовать не де-

сятичную, а двоичную систему счисления, содержащую только две цифры 0 и 1. Программисты для вычислений также пользуются восьмеричной и шестнадцатеричной системами счисления.

Количество цифр, используемых в системе счисления, называется ее «основанием». В десятичной системе основание равно десяти, в двоичной системе – двум, а в восьмеричной и шестнадцатеричной – соответственно, восьми и шестнадцати. В q-ичной системе счисления количество цифр равно q и используются цифры от 0 до $q - 1$.

Для работы **необходимо знать** представления десятичных чисел от нуля до 15 в системах счисления с основаниями $q = 2$, 8, 16 (см. табл. 1.2).

Таблица 1.2

Представления десятичных чисел в разных системах счисления

q = 10	q = 2	q = 8	q = 16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	*A*
11	1011	13	*B*
12	1100	14	*C*
13	1101	15	*D*
14	1110	16	*E*
15	1111	17	F

Кроме этого, полезно знать десятичные значения чисел 2^k от $k = 0$ до $k = 10$ (см. таб. 1.3).

Таблица 1.3

Значения чисел 2^k

k	0	1	2	3	4	5	6	7	8	9	10
2^k	1	2	4	8	16	32	64	128	256	512	1024

Перевод чисел из одной системы счисления в другую

Для перевода целого числа N с q-ичным основанием в десятичное число записывают в виде многочлена, а затем вычисляют его по правилам десятичной арифметики:

$$N = a_n \cdot q^n + a_{n-1} \cdot q^{n-1} \ldots + a_2 \cdot q^1 + a_1 \cdot q^0.$$

Здесь a_n – это цифры числа,

q – основание системы счисления,

n – 0, 1, 2 ... .

Пример:

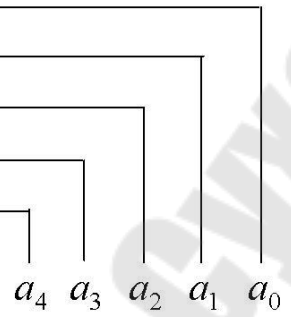
$$(11001)_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$$
$$= 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = (25)_{10}$$
$$(221)_3 = 2 \cdot 3^2 + 2 \cdot 3^1 \cdot 1 \cdot 3^0 = 2 \cdot 9 + 2 \cdot 3 + 1 \cdot 1 = (25)_{10}$$
$$(221)_3 = 2 \cdot 3^2 + 2 \cdot 3^1 \cdot 1 \cdot 3^0 = 2 \cdot 9 + 2 \cdot 3 + 1 \cdot 1 = (25)_{10}$$
$$(31)_8 = 3 \cdot 8^1 + 1 \cdot 8^0 = 3 \cdot 8 + 1 \cdot 1 = (25)_{10}$$
$$(534D)_{16} = 5 \cdot 16^3 + 2 \cdot 16^2 + 4 \cdot 16^1 + 13 \cdot 16^0 =$$
$$= 20480 + 512 + 64 + 13 = (21069)_{10}$$

Примечание: при работе с различными системами счисления число записывают в скобках, а за скобками – основание системы.

Для обратного преобразования целых чисел (из десятичной системы счисления в систему с основанием q) число N делят на q и записывают остатки от деления до тех пор, пока частное от предыдущего деления не станет равным нулю.

Пример: преобразуем число 25 в двоичную систему:

Исходное число	Частное	Остаток
25/2	12	1
12/2	6	0
6/2	3	0
3/2	1	1
1/2	0	1



Результат: $(25)_{10} = (11001)_2$

Когда последнее частное стало равно нулю, записывают все остатки подряд от последнего к первому. Таким образом, получили число в двоичной системе счисления – $(11001)_2$.

Для перевода смешанных чисел в двоичную систему счисления требуется отдельно переводить их целую часть и дробную части. В записи результата целая часть перевода отделяется от дробной запятой в соответствии с формулой:

$$N = \pm a_n \; a_{n-1} \; \ldots \; a_1 \; a_0, \; a_{-1} \; a_{-2} \; \ldots \; a_{-n}$$

Основные системы счисления

Двоичная система счисления. В компьютерной технике в основном используется двоичная система счисления. Такую систему очень легко реализовать в цифровой микроэлектронике, так как для нее требуется всего два устойчивых состояния (0 и 1).

Двоичная система счисления может быть непозиционной и позиционной. Реализовано это может быть присутствием какого-либо физического явления или его отсутствием. Например: есть электрический заряд или его нет, есть напряжение или нет, есть ток или нет, есть сопротивление или нет, отражает свет или нет, намагничено или не намагничено, есть отверстие или нет и т. п.

Восьмеричная система счисления – позиционная целочисленная система счисления с основанием 8. Для представления чисел в ней используются цифры от 0 до 7.

Восьмеричная система счисления часто используется в областях, связанных с цифровыми устройствами. Характеризуется легким переводом восьмеричных чисел в двоичные и обратно, путем замены восьмеричных чисел на триады двоичных.

Ранее эта система широко использовалась в программировании и компьютерной документации, однако в настоящее время почти полностью вытеснена шестнадцатеричной системой.

Для перевода двоичного числа в восьмеричное исходное число разбивают на триады влево и вправо от запятой; отсутствующие крайние цифры дополняют нулями. Затем каждую триаду записывают восьмеричной цифрой (см. табл. 1.2).

Пример: иллюстрация перевода двоичного числа в восьмеричное число:

$$N = (110011,100010)_2 = \left(\overbrace{110}^{6}\ \overbrace{011}^{3},\overbrace{100}^{4}\ \overbrace{010}^{2}\right)_2 = (63,42)_8 .$$

Шестнадцатеричная система счисления – позиционная система счисления по целочисленному основанию 16. Обычно в качестве шестнадцатеричных цифр используются десятичные цифры от 0 до 9 и латинские буквы от A до F для обозначения цифр от $(1010)_2$ до $(1111)_2$, то есть (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)$_{16}$.

Для перевода двоичного числа в шестнадцатеричное исходное число разбивают на тетрады влево и вправо от запятой; отсутствующие крайние цифры дополняют нулями. Затем каждую тетраду записывают шестнадцатеричной цифрой (см. табл. 1.2).

Пример: иллюстрация перевода двоичного числа в шестнадцатеричное число:

$$N = \left(\overbrace{0111}^{7}\ \overbrace{1010}^{A}\ \overbrace{1011}^{B},\ \overbrace{1110}^{E}\ \overbrace{1111}^{F}\right)_2 = (7AB,EF)_{16}.$$

1.2. Варианты заданий к лабораторной работе (см. табл. 1.4)

Задание 1. Перевести целые числа из десятичной системы счисления:

– в двоичную;

– в восьмеричную;

– в шестнадцатеричную.

Задание 2. *Перевести целые числа из двоичной системы счисления:*

– в восьмеричную;
– в шестнадцатеричную;
– в десятичную.

Задание 3. *Перевести целые числа из шестнадцатеричной системы счисления:*
– в двоичную;
– в восьмеричную;
– в десятичную.

Задание 4. *Сложить*:
– двоичные числа;
– восьмеричные числа;
– шестнадцатеричные числа.

Задание 5. *Найти разность:*
– двоичных чисел;
– восьмеричных чисел;
– шестнадцатеричных чисел.

Задание 6. *Вычислить значение выражения и представить в десятичной системе счисления.*

Таблица 1.4

Варианты заданий к лабораторной работе

	Вариант 1	**Вариант 2**	**Вариант 3**	**Вариант 4**	**Вариант 5**
Задание 1	a) 2515 b) 3084 c) 9042	a) 1052 b) 1387 c) 7634	a) 2042 b) 5548 c) 2372	a) 5911 b) 6321 c) 7629	a) 3988 b) 5147 c) 1123
Задание 2	a) 110101 b) 111010 c) 101111	a) 011001 b) 101010 c) 010101	a) 100110 b) 110011 c) 101111	a) 011001 b) 100001 c) 001001	a) 100010 b) 111000 c) 011111
Задание 3	a) 1F52 b) 5521 c) 1101	a)1A1B b) 2350 c) 3239	a) 5EE2 b) 2682 c) 2461	a) 7B1B b) 3458 c) 6537	a) 1C2D b) 6824 c) 8673
Задание 4	a) 1011 + + 0111 b) 573 + 325 c) F1 + E7	a) 0110 + + 1100 b) 274 + 235 c) 93 + 2C	a) 1010 + + 0101 b) 271 + 123 c) 58 + 79	a) 1101 + + 1101 b) 632++714 c) 51 + 9D	a) 1010 + + 1010 b) 521++623 c) 36 + AB

Продолжение таблицы 1.4

	Вариант 1	**Вариант 2**	**Вариант 3**	**Вариант 4**	**Вариант 5**
Задание 5	a) 1011 – – 0111 b) 573 – 325 c) F1 – E7	a) 1110 – – 1100 b) 274 – 235 c) 93 – 2C	a) 1010 – – 0101 b) 271 – 123 c) A8 – 79	a) 1101 – – 1001 b) 732 – 714 c) B1 – 9D	a) 1010 – – 1000 b) 721 –623 c) C6 – AB
Задание 6	23_8 + $A2_{16}$ * * 1001_2	$B1_{16}$ – – 1011_2 * * 117_8	51_8 * 21_{16} – – 45510	(59_{16} + + 1110_2) * * 456_8	25_8 * * 567_{16} – – 10101_2

Лабораторная работа №2

Логические и арифметические операции над двоичными числами

Цель: получение навыков выполнения арифметических и логических операций над двоичными числами.

Задание 1. Выполнить логическое сложение двоичных чисел, представленных в шестнадцатеричном коде.

№ вар.	1	2	3	4	5	6	7	8
1 оп.	23AD	9FD	3409	FF00	A456	12FF	5123	EF00
2 оп.	5F34	567D	E450	9F1	87FC	BA00	FF03	456

№ вар.	9	10	11	12	13	14	15	16
1 оп.	23AD	AFD0	3509	1F00	A456	12FF	5F34	EF00
2 оп.	E45A	56AD	80FC	9F10	BA00	5123	FF01	4560

Задание 2. Выполнить логическое сложение по модулю 2 двоичных чисел, представленных в шестнадцатеричном коде.

№ вар.	1	2	3	4	5	6	7	8
1 оп.	FD45	EE2D	3AA9	F4F0	A456	126F	5113	EF30
2 оп.	5784	529D	9810	9FD3	857C	BD00	FF03	45A6

№ вар.	9	10	11	12	13	14	15	16
1 оп.	2CFD	AF60	35E9	1700	A466	122F	5F34	4F00

2 оп.	E4BA	56BD	80EC	9A10	BA50	4123	FF01	4562

Задание 3. Выполнить логическое умножение двоичных чисел, представленных в шестнадцатеричном коде.

№ вар.	1	2	3	4	5	6	7	8
1 оп.	FD45	EE2D	3AA9	F4F0	A456	126F	5113	EF30
2 оп.	E4BA	56BD	80EC	9A10	BA50	4123	FF01	4562

№ вар.	9	10	11	12	13	14	15	16
1 оп.	2CFD	AF60	35E9	1700	A466	122F	5F34	4F00
2 оп.	FD45	EE2D	3AA9	F4F0	A456	126F	5113	EF30

Задание 4. Выполнить сложение целых двоичных чисел без знака, представленных в шестнадцатеричном коде. Все операнды перевести в десятичный код.

№ вар.	1	2	3	4	5	6	7	8
1 сл.	23AD	9FD	3409	FF00	A456	12FF	5123	EF00
2 сл.	5F34	567D	E450	9F1	87FC	BA00	FF0	456

№ вар.	9	10	11	12	13	14	15	16
1 сл.	23AD	AFD0	3509	1F00	A456	12FF	5F34	EF00
2 сл.	E45A	56AD	80FC	9F10	BA00	5123	FF01	4560

Задание 5. Выполнить сложение целых двоичных чисел со знаком (в дополнительном коде), представленных в десятичном коде. Все операнды перевести в десятичный код.

№ вар.	1	2	3	4	5	6	7	8
1 сл.	245	797	-345	878	-1	-5435	5123	-5862
2 сл.	-765	-16000	-5987	-5432	1866	-342	-5634	342

№ вар.	9	10	11	12	13	14	15	16
1 сл.	-24	-654	-54	-52	3333	12553	6733	-543
2 сл.	-1655	-5438	1500	5431	-400	-76	-333	-867

Задание 6. Выполнить умножение целых двоичных чисел без знака, представленных в шестнадцатеричном коде. Все операнды перевести в десятичный код.

№ вар.	1	2	3	4	5	6	7	8
1 мн.	AD	67	78	23	3A	8E	9C	45
2 мн.	34	9A	5B	DF	F0	34	44	AD

№ вар.	9	10	11	12	13	14	15	16
1 мн.	E4	7A	AA	CC	CA	B4	DD	7F
2 мн.	23	FE	10	56	49	C6	87	23

Задание 7. Выполнить деление целых двоичных чисел без знака, представленных в шестнадцатеричном коде. Все операнды перевести в десятичный код.

№ вар.	1	2	3	4	5	6	7	8
Делимое	1AC4	FF67	2F78	CD23	763A	988E	AD9C	4FF5
Делитель	34	9A	5B	DF	F0	34	44	AD

№ вар.	9	10	11	12	13	14	15	16
Делимое	E224	751A	A3A8	CC22	CDFA	11B4	D18D	711F
Делитель	23	FE	10	56	49	C6	87	23

Лабораторная работа №3

Вывод строковых массивов на языке Ассемблера

Цель работы:

– знакомство с Макро Ассемблером MASM32;

– создание минимального Windows-приложения на ассемблере;

– изучение вызова функций API win32 из ассемблерных приложений.

Немного теории…

Далее будем полагать, что имеем дело с 32-битными процессорами, начиная с *Intel* 80386. Эти процессоры имеют 32-битную шину адреса, следовательно, могут адресовать 2^{32} = 4 294 967 296 = 4 Гб памяти.

Каждую Win32 программу Windows запускает в отдельном виртуальном пространстве. Это означает, что каждая Win32 программа будет иметь 4-Гб адресное пространство, но вовсе не означает, что каждая программа имеет 4 Гб физической памяти, а только то, что программа может обращаться по любому адресу в этих пределах.

Под Win32 есть только одна модель память: плоская, без 64-килобайтных сегментов. Память – это большое последовательное 4-гигабайтовое пространство. Можно не париться с сегментными регистрами, зато можно использовать любой сегментный регистр для адресации к любой точке памяти.

При программировании под Win32 необходимо помнить, что Windows использует регистры *esi*, *edi*, *ebp* и *ebx* для своих целей и не ожидает, что вы измените их значение. Если же вы используете какой-либо из этих четырёх регистров в вызываемой функции, то не забудьте восстановить их перед возвращением управления Windows.

ОС Windows предоставляет огромное количество ресурсов Windows-программам через Windows API (Application Programming Interface) - коллекцию полезных функций, располагающихся непосредственно в операционной системе и готовых для использования программами. Эти функции находятся в нескольких динамически подгружаемых библиотек (DLL), таких как *kernel32.dll*, *user32.dll* и *gdi32.dll*. *Kernel32.dll* содержит API функции, взаимодействующие с памятью и управляющие процессами, *user32.dll* контролирует пользовательский интерфейс, *gdi32.dll* отвечает за графические операции. Существуют также другие библиотеки. Код API-функций не включается в исполняемый файл.

Вызов системных функций API Win32 из программы на ассемблере подчиняется набору соглашений *stdcall*. Эти соглашения заключаются в следующем:

- регистр символов в имени функции не имеет значения. Например, функции с именами *ExitProcess* и *exitprocess* – это одна и та же функция;
- **аргументы передаются вызываемой функции через стек**. Если аргумент укладывается в 32-битное значение и не подлежит модификации вызываемой функцией, он обычно записывается в стек непосредственно. В остальных случаях программист должен разместить значение аргумента в памяти, а в стек записать 32-битный указатель на него. Таким образом, все передаваемые функции API параметры представляются 32-битными величинами, и количество байт, занимаемых в стеке для передачи аргументов, кратно четырем;

– вызывающая программа загружает аргументы в стек последовательно, начиная с последнего, указанного в описании функции, и заканчивая первым. После загрузки всех аргументов **программа вызывает функцию командой *call***;

– за возвращение стека в исходное состояние после возврата из функции API отвечает сама эта вызываемая функция. Программисту заботиться о восстановлении указателя стека *esp* нет необходимости;

– вызываемая функция API гарантированно сохраняет регистры общего назначения *ebp*, *esi*, *edi*. Регистр *eax*, как правило, содержит возвращаемое значение. Состояние остальных регистров после возврата из функции API следует считать неопределенным. (Полный набор соглашений *stdcall* регламентирует также сохранение системных регистров *ds* и *ss*. Однако, для *flat*-модели памяти, используемой в Win32, эти регистры значения не имеют).

Итак, вот шаблон программы на ассемблере MASM32 для Win32

```
.386
.model flat, stdcall
option casemap: none
include C:\masm32\include\windows.inc
include C:\masm32\include\kernel32.inc
includelib C:\masm32\lib\kernel32.lib
.data
<инициализированные данные>
...
.data?
<неинициализированные данные>
...
.const
<константы>
...
.code
<метка>:
     <текст программы>
     ...
     push NULL
     call ExitProcess
end <метка>
```

.386 – директива ассемблера, определяющая набор команд процессора, который может быть использован в программе. Для приложений win32 необходимо использовать эту директиву или выше, в зависимости от того, собираетесь ли вы использовать возможности, предоставляемые процессорами последующих поколений;

.model – директива ассемблера, определяющая сегментную модель памяти приложения как плоскую (**flat**). Именно такая сегментная модель

должна всегда использоваться при написании приложений для win32;

stdcall – говорит MASM32 о порядке передачи параметров. Согласно ему, данные передаются справа налево, но вызываемый ответственен за выравнивание стека. Платформа Win32 использует исключительно STDCALL;

option casemap: none – говорит MASM сделать метки чувствительными к регистрам, то есть ExitProcess и exitprocess - это различные имена;

include – директива ассемблера, добавляющая содержимое указанного после директивы файла в то место, где расположена директива. При использовании API-функций win32 в программе должны быть описаны их прототипы. Это можно сделать вручную или путём подключения соответствующего *include*-файла. Подключаемые файлы находятся в директории *c:\masm32\include*. Файлы подключения имеют расширение *.inc* и прототипы функций DLL находятся в *.inc* файле с таким же именем, как и у этой DLL. Например, *ExitProcess* экспортируется *kernel32.lib*, так что прототип *ExitProcess* находится в *kernel32.inc*.

includelib – директива, сообщающая ассемблеру, какие библиотеки использует программа.

Например, сервис функции *ExitProcess* предоставляется *dll*-библиотекой *kernel32.dll*, следовательно, при сборке приложения необходимо подключить библиотеку импорта *kernel32.lib*.

Также можно указать имена библиотек импорта не с помощью директивы *.includelib*, а в командной строке при запуске линкера;

.data – директива, определяющая секцию инициализированных данных программы.

.data? – директива, определяющая секцию неинициализированных данных программы. Если нужно предварительно выделить некоторое количество памяти, но не инициализировать ее, то данная секция предназначается для этого. Преимущество неинициализированных данных следующее: они не занимают места в исполняемом файле. Например, если вы выделите 10 кб в секции *.data?*, *exe*-файл не увеличиться на 10 кб. Вы всего лишь говорите компилятору, сколько места будет нужно, когда программа загрузится в память.

.const – директива, определяющая секцию констант. Константы не могут быть изменены программой.

Не обязательно задействовать все три секции. Объявляйте только те, которые хотите использовать.

.code – директива, определяющая секцию текста программы.

Все четыре директивы (секции) могут поделить адресное пространство на логические секции. Начало одной секции отмечает конец предыдущей. По факту есть две группы секций: данных и кода.

<метка>: – произвольная метка, устанавливающая границы кода. Обе метки должны быть идентичны. Весь текст программы должен располагаться

между **<метка>:** и **end <метка>**.

push NULL – задание параметра функции ExitProcess.

call ExitProcess – вызов функции завершения приложения. Здесь код выхода NULL, но он, может быть любым в пределах 32-разрядного целого числа. В полноценных приложениях нормой считается определение кода выхода в цикле обработки сообщений главного окна. Именно с помощью функции *ExitProcess* должно завершаться приложение win32, написанное на ассемблере.

Программирование процессоров x86 на языке Ассемблера для архитектуры Win32.

Программу на языке Ассемблера для Win32 можно писать как в специализированной среде разработки (SASM, RadASM и др.), так и в обычном блокноте.

Для программирования в блокноте скачайте и установите программу Notepad++ (https://notepad-plus-plus.org/downloads/).

Скачиваем и распаковываем архив masm32v11r.zip по ссылке http://www.masm32.com/download.htm. Запускаем файл установщика *install.exe*. Выбираем для установки диск C. После установки на диске C появится папка *masm32*.

Для быстрого доступа из любой директории можно добавить пути в переменные среды Windows.

Для этого переходим: Этот компьютер -> Свойства -> Дополнительные параметры системы -> Переменные среды -> Системные переменные… переменная Path… Изменить…

В конец списка добавляем строки:

```
C:\masm32
C:\masm32\bin
C:\masm32\help
C:\masm32\include
C:\masm32\lib
```

… сохраняем.

Создаем текстовый файл и переименовываем его в Test.asm.

Открываем файл в Notepad++ и пишем программу на языке Ассемблера, производящую вывод текстовой строки на экран консоли (не для MS DOS).

```
;-----------------------------------------
; Программа на masm32 для консоли
; (вывод сообщения)
;-----------------------------------------

.386
.model flat, stdcall
option casemap:none

; Библиотеки и подключаемые файлы проекта
```

```
;-----------------------------------------
include C:\masm32\include\windows.inc
include C:\masm32\include\kernel32.inc
includelib C:\masm32\lib\kernel32.lib

; Сегмент данных
;-----------------------------------------
.data
string          db "Hello World!", 0Ah, 0h
sConsoleTitle   db "My first project", 0

; Сегмент кода
;-----------------------------------------
.code
start:
;-----------------------------------------
; Функция SetConsoleTitle устанавливает строку для заголовка текущей
; консоли
;
; ==== Аргументы ====
; lpConsoleTitle - указатель на строку для заголовка консоли

      push offset sConsoleTitle
      call SetConsoleTitle


; ----------------------------------------
; Функция GetStdHandle извлекает дескриптор для указанного
; стандартного устройства (стандартный ввод, стандартный вывод или
; стандартная ошибка).
; Функция возвращает дескриптор для указанного устройства.
;
; ==== Аргументы ====
; STD_INPUT_HANDLE - значение -10 стандартное устройство ввода
; STD_OUTPUT_HANDLE - значение -11 стандартное выходное устройство
; STD_ERROR_HANDLE - значение -12 устройство стандартных ошибок

      push STD_OUTPUT_HANDLE
      call GetStdHandle

; ----------------------------------------
; Функция WriteConsole записывает символьную строку в экранный буфер
; консоли, начинающийся с текущей позиции курсора.
;
; ==== Аргументы ====
; hConsoleOutput - дескриптор экранного буфера
; lpBuffer - указатель на массив символов (строку), которые будут
; записаны в экранный буфер консоли
; nNumberOfCharsToWrite  - число символов для записи (размер строки)
```

```
; lpNumberOfCharsWritten - указатель на переменную, которая принимает
; число фактических записей
; lpReserved - зарезервировано. MSDN рекомендует просто передать NULL

      push NULL
      push sizeof string
      push offset string
      push eax
      call WriteConsole

;-------------------------------------------
; Функция Sleep приостанавливает работу по выполнению текущего потока
; на заданный промежуток времени.
;
; ==== Аргументы ====
; dwMilliseconds - длительность задержки [мс]. Значение БЕСКОНЕЧНО
; (INFINITE) вызывает бесконечную задержку

      push INFINITE
      call Sleep

;-------------------------------------------
; Функция ExitProcess заканчивает работу процесса и всех его потоков
;
; ==== Аргументы ====
; uExitCode - возвращаемое значение

      push NULL
      call ExitProcess
end start
```

В папке с файлом *Test.asm* создаем текстовый файл *Run.bat* и добавляем в него в блокноте строку:

```
C:\masm32\bin\ml.exe /c /coff /Fl Test.asm
```

… сохраняем изменения.

Со значением ключей *ml.exe* можно ознакомиться здесь: https://wasm.in/blogs/kompiljacija-fajlov-asm-s-pomoschju-kompiljatora-ml-exe.74/.

После выполнения этой строки в папке появятся ещё два файла *Test.lst* и *Test.obj*.

Добавим в файл *Run.bat* ещё одну строку:

```
C:\masm32\bin\link.exe /SUBSYSTEM:CONSOLE /LIBPATH:C:\masm32\lib\
Test.obj
```

Подробную информацию о ключах link.exe можно посмотреть здесь http://bitfry.narod.ru/link.htm.

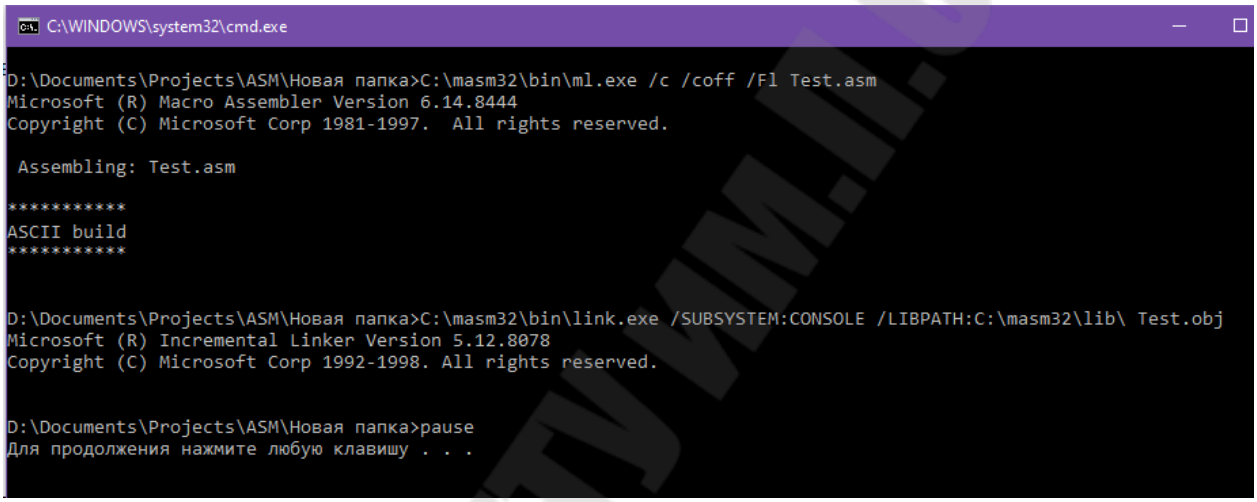
После выполнения этой строки появится файл *Test.exe*.

Также в файл Run.bat добавьте строки:

```
pause
del Test.obj
start Test.exe
```

```
1  C:\masm32\bin\ml.exe /c /coff /Fl Test.asm
2  C:\masm32\bin\link.exe /SUBSYSTEM:CONSOLE /LIBPATH:C:\masm32\lib\ Test.obj
3  pause
4  del Test.obj
5  start Test.exe
```

Сохраните изменения в файлах и запустите *Run.bat*. В результате



В результате выполнения файла Test.exe откроется консоль с заголовком «*My first project*» и содержанием «*Hello, World!*».

Задание 1.

Набрать текст приведеной выше программы, произвести компиляцию и линковку, проверить работоспособность приложения.

Задание 2.

Переписать приведенный выше текст программы с использованием макрокоманды ***invoke***.

Макрокоманда *invoke*, позволяет записывать вызовы в более удобном виде. Например, следующий вызов:

```
push MB_OKCANCEL
push offset hello_title
push offset hello_mess
push 0
call MessageBox
```

с помощью макрокоманды *invoke* будет записан следующим образом:

```
invoke MessageBox, 0, addr hello_mess, addr hello_title, MB_OKCANCEL
```

Содержание отчёта.

1. Название работы.
2. Цель работы.
3. Постановка задачи.
4. Текст программ с комментариями.
5. Анализ результатов.
6. Выводы.

Лабораторная работа №4

Типы данных Ассемблера. Пересылка данных

Цель работы:

– знакомство с типами переменных Ассемблера и их размещением в памяти;

– изучение команд пересылки данных Ассемблера;

– получение навыков работы с отладчиком.

Немного теории…

Самое главное в процессоре это регистры. Регистры состоят из триггеров. Триггер может иметь два значения «0» или «1». Регистры бывают 8-и, 16-и, 32-х и 64-х разрядные. Если регистр 8-разрядный, то в нем 8 триггеров.

Регистр используется для промежуточного хранения информации, некоторые регистры хранят только определённую информацию. Также есть порты ввода вывода. Доступ к внешним устройствам происходит через порты ввода вывода, с помощью контроллера ввода-вывода. Не путайте порты ввода-вывода с портами LPT, COM и т.д.

Директивы определения данных в Ассемблере.

В общем случае все директивы объявления данных имеют такой синтаксис:

[имя] директива <значение>

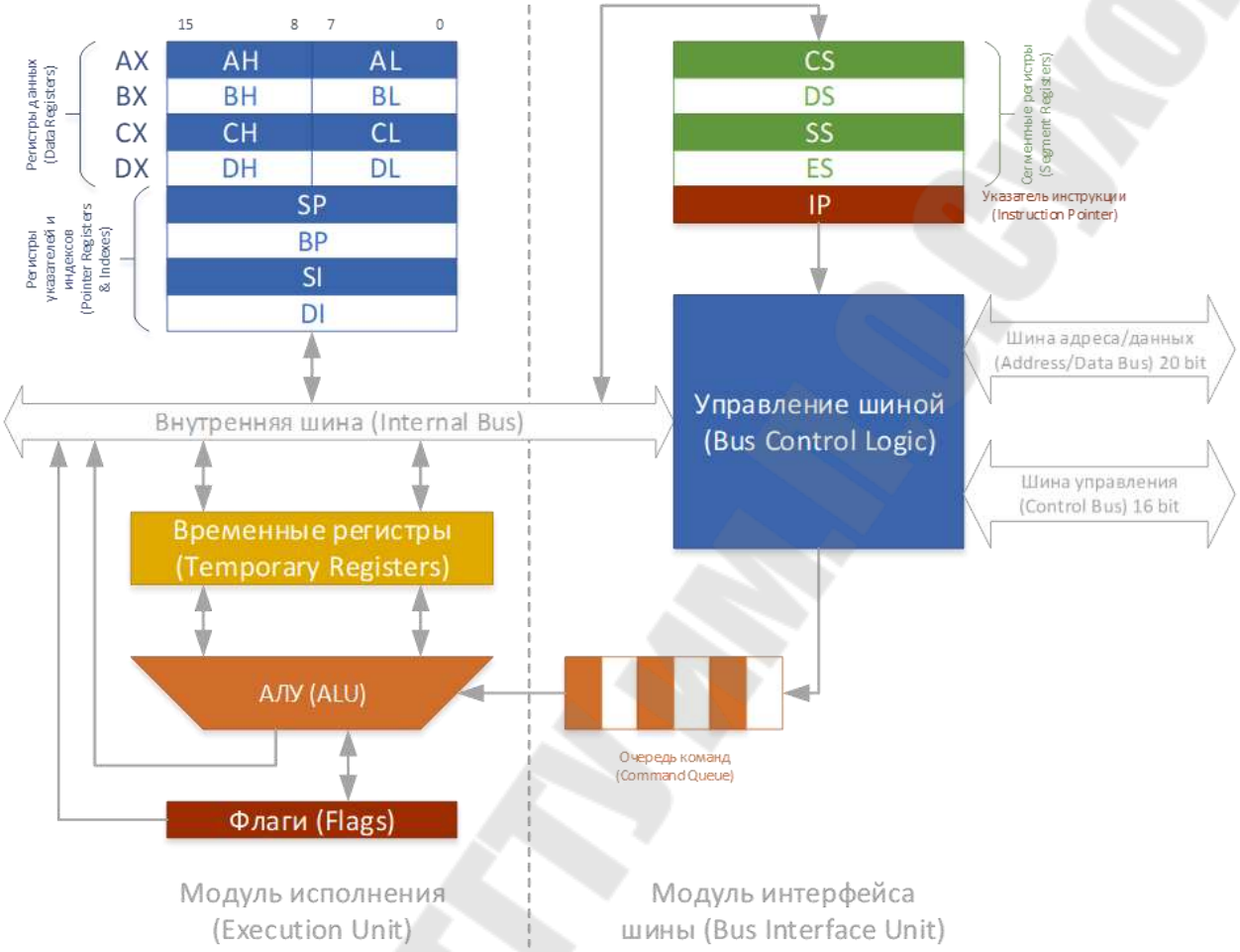


Рис. 4.1. Архитектура МП i8086

Синтаксис параметра <*значение*> может быть следующим:

– *?* (неинициализированные данные);

– *значение* (значение элемента данных);

– *DUP* (dup_выражение) объявление и инициализация массивов.

К директивам объявления и инициализации простых данных относятся:

db *(Define Byte)* – определить байт. Директивой *db* можно задавать следующие значения:

– выражение или константу, принимающую значение из диапазона -128...+127 для чисел со знаком; 0...255 для чисел без знака;

– 8-битовое относительное выражение, использующее операции *HIGH* и *LOW*;

– символьную строку из одного или более символов, заключенную в кавычки.

dw *(Define Word)* – определить слово. Директивой *dw* можно задавать следующие значения:

– выражение или константу, принимающую значение из диапазона -32768...32767 для чисел со знаком; 0...65535 для чисел без знака;

– 1- или 2-байтовую строку, заключенную в кавычки.

dd *(Define Double word)* - определить двойное слово. Директивой *dd* можно задавать следующие значения:

– выражение или константу, принимающую значение из диапазона -2147483648...+2147483647 для чисел со знаком; для чисел 0...4 294 967 295 без знака;

– относительное или адресное выражение, состоящее из 16-битового адреса сегмента и 16-битового смещения;

– строку длиной до 4 символов, заключенную в кавычки;

dq *(Define Quarter word)* – определить учетверенное слово;

df *(Define Far word)* – определить указатель дальнего слова;

dp *(Define Pointer)* – определить указатель 48 бит;

dt *(Define Ten Bytes)* – определить 10 байт.

Для резервирования памяти под массивы используется директива *dup:*

```
area   dw  10 dup(?)      ; резервируется  память объемом 10 слов
string db  20 dup('*')    ; строка заполняется кодом символа '*'
array  dw  3 dup(8)       ; массив из 3 слов инициализир. числом 8
t      db  4 dup(5 dup(8))      ; 20 восьмерок
```

Пример применения директив определения данных

```
; определение байта
      v1     db   ?              ; не инициализировано
      v2     db   'ИП21'         ; символьная строка
      v3     db   6              ; десятичная константа
      v4     db   0afh           ; шестнадцатеричная константа
      v5     db   0110100b       ; двоичная константа
;определение слова
      w1     dw   bff3h          ; шестнадцатеричная константа
      w2     dw   11101111b      ; двоичная константа
      w3     dw   2,24,5,7       ; 4 константы
      w4     dw   23 dup(*)      ; 23 звездочки
;определение двойного слова
      d1     dd   ?              ; не определено
```

```
       d2      dd    'uAudf'          ; символьная строка
       d3      dd    08734            ; десятичная константа
       d4      dd    087h, 85fh       ; две константы
;определение учетверенного слова
       v1      dq    ?                ; не определено
       v2      dq    a83dh            ; константа
; определение массива слов
       mass    dw    100, -675, 54, 4556, 1
```

Пересылка данных.

Синтаксис команды MOV Ассемблера:

MOV [приёмник], [источник]

С помощью этой команды можно переместить значение из источника в приёмник. То есть, команда *MOV* копирует содержимое источника и помещает это содержимое в приёмник.

Никакие флаги при этом не изменяются.

При использовании этой команды следует учитывать, что имеются некоторые ограничения. А именно, инструкция `MOV` не может:

– записывать данные в регистры CS и IP;

– копировать данные из одного сегментного регистра в другой сегментный регистр (сначала нужно скопировать данные в регистр общего назначения);

– копировать непосредственное значение в сегментный регистр (сначала нужно скопировать данные в регистр общего назначения).

Источником может быть один из следующих:

– область памяти (*mem*);

– регистр общего назначения (*reg*);

– непосредственное значение (например, число) (*imm*);

– сегментный регистр (*sreg*).

Приёмником может быть один из следующих:

– область памяти (*mem*);

– регистр общего назначения (*reg*);

– сегментный регистр (*sreg*).

Пример использования инструкции *MOV*:

```
    mov ax, 0b800h       ; установить ax = b800h
    mov ds, ax           ; копировать значение из ax в ds
    mov cl, 'a'          ; cl = 41h (ascii-код)
```

Просмотр выполнения программы в отладчике.

Для отладки написанной программы можно воспользоваться отладчиком, например, *OllyDbg*. Программа *OllyDbg* — 32-битный отладчик (*debugger*), предназначенный для анализа и модификации откомпилированных исполняемых файлов и библиотек, работающих в режиме пользователя. *OllyDbg* отличается интуитивно понятным интерфейсом, подсветкой специфических структур кода, простотой в установке и запуске.

Исполняемый файл программы загружается в отладчик через меню: *File -> Open*.

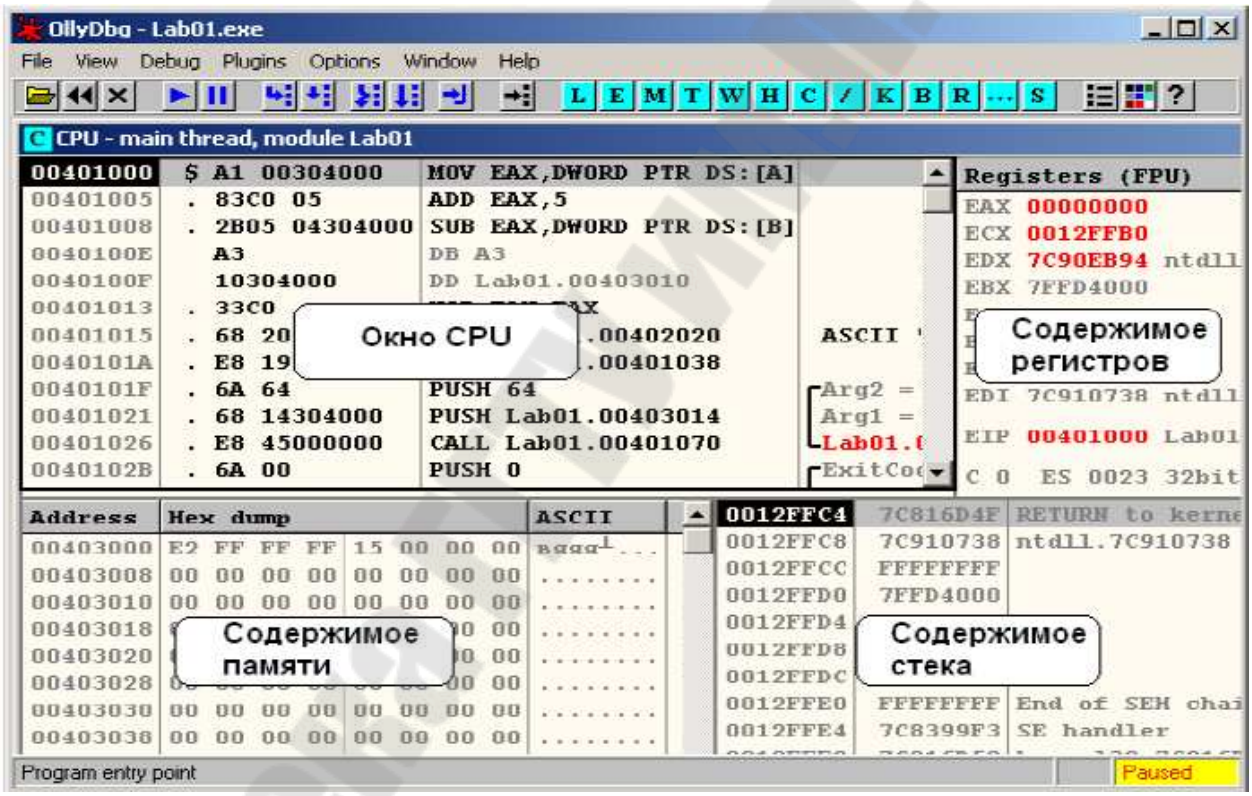


Рис. 4.2. Окно программы *OllyDbg*

В окне программы (рис.4.2) можно выделить четыре области:

– окно CPU, в котором высвечиваются адреса команд исполняемой программы, шестнадцатеричные коды команд программы, результаты дисассемблирования и результаты выделения параметров процедур;

– окно регистров, в котором отображается содержимое регистров процессора, включая регистр флагов;

– окно памяти – окно, в котором высвечиваются адреса памяти (*Address*) и ее шестнадцатеричный (*Hex dump*) и символьный (ASCII) дампы;

– окно стека, в котором отображается содержимое области памяти, отведенной под стек.

Стек – область памяти, в которой хранится информация о каждом обращении к функции. Кроме того, в стеке могут временно храниться данные, например, содержимое регистров.

Стек увеличивается в сторону уменьшения адресов памяти. Когда вы добавляете что-то в стек, то эти данные помещаются по адресу памяти младшему, чем адрес предыдущего элемента. То есть, по мере роста стека адрес вершины стека уменьшается.

Инструкция *push* заносит что-нибудь на верх стека, а *pop* уносит данные оттуда. Например, *push EAX* выделяет место наверху стека и помещает туда значение из регистра *EAX*, а *pop EAX* переносит любые данные из верхней части стека в *EAX* и освобождает эту область памяти.

Регистр *ESP* — указатель стека. Он хранит адрес вершины стека и изменяет своё значение при записи или чтении из стека.

В исходный момент времени (см. рис.4.2) курсор находится в окне CPU, т.е. программа готова к выполнению. Для выполнения команд программы в пошаговом режиме используют следующие функциональные клавиши отладчика:

– F7 (*step into*) – выполнить шаг с заходом в тело процедуры;

– F8 (*step over*) – выполнить шаг, не заходя в тело процедуры.

Адрес начала программы *00401000h*.

Адрес начала раздела инициированных данных – *00403000h*. Каждое значение занимает столько байт, сколько резервируется соответствующей директивой. В отладчике каждый байт представлен двумя шестнадцатеричными цифрами. Кроме того, используется обратный порядок байт, т.е. младший байт числа находится в младших адресах памяти (перед старшим). Если шестнадцатеричная комбинация соответствует коду символа, то он высвечивается в следующем столбце (ASCII), иначе в нем высвечивается точка.

Не инициированные данные располагаются после инициированных с адреса, кратного 16.

При каждом выполнении команды мы можем наблюдать изменение данных в памяти и/или в регистрах, отслеживая процесс выполнения программы и контролируя правильность промежуточных результатов.

Так после выполнения первой команды число *A* копируется в регистр *EAX*, который при этом подсвечивается красным. При записи в регистр порядок байт меняется на прямой, при котором первым записан старший байт.

Задания для лабораторной работы.

1. В соответствии с вариантом задания (табл. 4.1) написать программу, где в сегменте данных будут созданы переменные a, b, массив данных и строка. Произвести пересылку данных в указанное место. Произвести обмен значений переменных через стек.

Табл.4.1. Задание к лабораторной работе №4

№	Переменные				Место пересылки	
	a	b	m	s	a	b
1	-34	65465	5,5,5,5,5,5,5	Привет	EAX	EDX
2	-5	0x56	5,89,65,9	Яблоко	EBX	EAX
3	45	765765	0,0,0,0,0,0,1	Книга	ECX	EBX
4	0b1011	0x76A	765765, -1,0,0, 99999, 56678945	Солнце	EDX	ECX
5	-56	-67565765	-675, 765, -5643,789, -765, 777	Дом	EAX	EDX
6	0b111	-39879798	9,8,7,9,8,7	Музей	EBX	ECX
7	87687	-76	34,34,34,34	Квартира	ECX	EAX
8	67	0b1001	56,78,65,0,1	Медведь	EDX	EBX
9	0b110011	-5765	543543, 7675, 675476904, 56	Собака	EAX	ECX
10	0x45	11111b	654, 98, -432, -4322	Ручка	EBX	EDX
11	-2	7765765	-9,-2,8,67,5	Радуга	ECX	EAX
12	-1	337865765	0x543, 0x34, 0x7647FFF, 0x5633	Звезда	EDX	EAX
13	0xFF095	87687	-9,-9,-9,-9,-9	Часы	EAX	EDX
14	0b100011	6754	44,44,44,44	Минуты	EBX	EAX
15	0x5F	0xAB54	0xADFFFF, 0x34, 0xDF 0x45, 0x67	Планета	ECX	EBX
16	0x34F	0b1111111	56,78,689, 342436, 6	Дождь	EDX	ECX
17	0x78653	-4	3,3,3,3,3,3,3	Чашка	EAX	EDX

18	99	-2376575	0,0,0,0,0,0,0	Футбол	EBX	EAX
19	909	547659099	0,1,2,3,0,1, 2,3	Чемодан	ECX	EDX
20	0b11	787423240	0b1111, 0b10101, 0b10000, 0b101111, 0b1111111	Парк	EDX	EAX
21	0x78FF1F	-6534	0x45,0x5, 0xF4545, 0x6, 0xFFF	Кофе	EAX	EBX
22	0b1111111	0xDF	55,55,55,55	Луна	EBX	ECX
23	0x567FF	76576	11110000, 111000, 11110000, 111000	Птица	ECX	EDX
24	-8798675	543	0xFF00FF, 0xFF00FF, 0xFF00FF, 0xFF00FF	Океан	EDX	EAX
25	-999999	-22	-65,-76575, 0, 5423543	Музыка	EAX	EBX
26	7777777	0xABCD	0b1110001, 0xFFFFFF, 675765, -4	Компьютер	EBX	ECX
27	-54765899	0b11101	555, 6667, 77777, 8888	Шоколад	ECX	EDX
28	-34	-88945765	564,-1, 1, 0, 45, 567890	Кинотеатр	EDX	EAX
29	3	0xFADFF	0,0,1,0,0,1	Машина	EAX	EBX
30	0x5435D	0x111111		Дворец	EBX	ECX

Для создания переменных выделить минимально необходимый объем памяти.

2. Открыть программу в отладчике. Указать адреса расположения переменных в памяти, заполнить табл. 4.2.

Табл.4.2. Адреса переменных в памяти

Имя переменной	Адрес	Расположение байт в памяти

3. Выполнить программу в отладчике в пошаговом режиме. После выполнения каждого шага заносить данные в табл. 4.3.

Табл.4.3. Состояние регистров* и стека

EAX	EBX	ECX	EDX	ESP	стек

*выбрать необходимое

4. По результатам п.2 и п.3 сделайте выводы.

Содержание отчёта:

1. Название работы.
2. Цель работы.
3. Постановка задачи.
4. Текст программ с комментариями.
5. Результаты выполнения программы в отладчике.
6. Выводы.

Лабораторная работа №5

Выполнение целочисленных арифметических операций в процессоре x86

Цель работы:

- знакомство с работой центрального процессора и памяти;
- изучение арифметических команд Ассемблера;
- получение навыков работы с отладчиком.

Немного теории…

Краткое описание арифметических команд

ADD	Сложение
ADC	Сложение с переносом
SUB	Вычитание
SBB	Вычитание с заемом
IMUL	Знаковое умножение
MUL	Беззнаковое умножение
IDIV	Знаковое деление
DIV	Беззнаковое деление

INC	Инкремент
DEC	Декремент

Команда ADD выполняет арифметическое сложение двух операндов *операнд_1* и *операнд_2*. Операнды должны иметь одинаковый размер. Результат записывается на место *операнд_1*.

ADD *операнд_1*, *операнд_2*
операнд_1 = операнд_1 + операнд_2

Команда ADC (Add with Carry) производит целочисленное сложение двух знаковых или беззнаковых операндов и флага переноса. Первый операнд может быть переменной в регистре или в памяти (*r8, r16, r32, r/m8, r/m16, r/m32*). Второй операнд – непосредственным значением (*imm8, imm16, imm32*), переменной в регистре или в памяти. При этом оба операнда одновременно не могут быть переменными в памяти.

ADC *операнд_1*, *операнд_2*
операнд_1 = операнд_1 + операнд_2 + флаг CF

Команда ADC обычно используется в многобайтных или многословных операциях сложения:

```
xor edx, edx                ; EDX = 0
mov  eax, 0FFFF FFFFh       ; EAX = 4 294 967 295(10)
add  eax, 0FFFF FFFFh       ; EAX = EAX + 4 294 967 295(10)
adc  edx, 0                 ; EDX = EDX + CF  (учитываем перенос)
                            ; EDX:EAX = 0000 0001h:FFFF FFFEh
```

Команда SUB (SUBtract) выполняет арифметическое вычитание двух операндов *операнд_1* и *операнд_2*. Операнды должны иметь одинаковый размер. Результат записывается на место *операнд_1*.

SUB *операнд_1*, *операнд_2*
операнд_1 = операнд_1 - операнд_2

В процессорах x86 для умножения чисел без знака предназначена команда MUL. У этой команды только один операнд — второй множитель, который должен находиться в регистре или в памяти. Местоположение первого множителя и результата задаётся неявно и

зависит от размера операнда:

Размер операнда	1-й множитель	Результат
1 байт	AL	AX
2 байта	AX	DX:AX
4 байта	EAX	EDX:EAX

Запись «DX:AX» означает, что старшее слово результата будет находиться в DX, а младшее — в AX.

Пример:

```
mul bl                  ; AX = AL · BL
mul ax                  ; DX:AX = AX · AX
```

Если старшая часть результата равна нулю, то флаги CF и OF будут иметь нулевое значение. В этом случае старшую часть результата можно отбросить. Это свойство можно использовать в программе, если результат должен быть такого же размера, как множители.

Команда IMUL умножает целые числа со знаком и может использовать один, два или три операнда:

- один операнд, аналогично MUL;
- два операнда

IMUL операнд_1, операнд_2
*операнд_1 = операнд_1 * операнд_2*

варианты: *IMUL reg, reg*
IMUL reg, mem
IMUL reg, imm

Пример:

```
imul edx, ecx           ; EDX = EDX * ECX
imul ebx, A             ; EBX = EBX * A
imul ecx, 6             ; ECX = ECX * 6
```

- три операнда

IMUL операнд_1, операнд_2, операнд_3
*операнд_1 = операнд_2 * операнд_3*

варианты: *IMUL reg, reg, imm*
IMUL reg, mem, imm
IMUL reg, imm, imm.

Пример:

```
imul ebx, A, 9                ; EBX = A * 9
imul ecx, edx, 11             ; ECX = EDX * 11
```

В процессорах x86 для деления чисел без знака предназначена команда DIV. У этой команды только один операнд — делитель, который должен находиться в регистре или в памяти. Местоположение делимого и результата задаётся неявно и зависит от размера операнда:

Размер операнда	Делимое	Частное	Остаток
1 байт	AX	AL	AH
2 байта	DX:AX	AX	DX
4 байта	EDX:EAX	EAX	EDX

Запись «DX:AX» означает, что старшее слово результата будет находиться в DX, а младшее — в AX.

Пример:

```
div bl                        ; AL = AX / BL
div bx                        ; AX = DX:AX / BX
```

Команда IDIV используется для деления чисел со знаком, синтаксис ее такой же, как у команды DIV.

Пример линейной программы

Написать программу, вычисляющую значение выражения

$$x = (a + b) \cdot (b - 1) / (d + 8).$$

```
.data
a sword 25
b sword -6
d sword 11

.data?
x sword ?
```

```
.code
start:
        mov cx, d                   ; CX = d
        add cx, 8                   ; CX = d + 8
        mov bx, b                   ; BX = b
        dec bx                      ; BX = b – 1
        mov ax, a                   ; AX = a
        add ax, d                   ; AX = a + d
        imul bx                     ; DX:AX = (a + d) · (b – 1)
        idiv cx                     ; AX = (a + d) · (b – 1) / cx
        mov x, ax                   ; x = AX

        invoke ExitProcess, NULL
end start
```

Задание.

1. Написать программу, вычисляющую заданное в соответствии с номером студента в журнале выражение:

№	Функция
1	$y = (b^2 – (c + 1) \cdot d) / b$
2	$y = a \cdot x / 2 – (a + b) / 2$
3	$y = (k – 5)^2 / 4 + 2 \cdot k$
4	$y = a^3 / 3 – c \cdot (x + 3)$
5	$y = a \cdot x – 3 \cdot (b + 3 / k)$
6	$y = 3 \cdot a \cdot x / [5 \cdot (b – 5)]$
7	$y = a \cdot (a + b / 4) / (k – 1)$
8	$y = c \cdot (c + b / 4) / (k – 1)$
9	$y = a \cdot j – j^2 / (k + 2)$
10	$y = a \cdot j – j^2 / (k + 2)$
11	$y = a / c – k + (d + 1) \cdot 5$
12	$y = a \cdot b / 2 – k + a / 2 – b$
13	$y = (b^2 – (c + 1) \cdot d) / b$
14	$y = b \cdot (c – d) – c / (d – 1)$
15	$y = (a + b) / d – d^2 \cdot a – b$
16	$y = (m – 5) \cdot (m + 2) + m + a / 2$
17	$y = a^2 / 2 – b^3 / (4 – a) + b$
18	$y = (a – b^2) / (x – a) + a^2 – c$
19	$y = (1 - a)^2 / c + k – 1 + c / 2$
20	$y = a \cdot c^2 – b \cdot a / c + a / b$
21	$y = a \cdot x \cdot (b – a) / 4 + a^2 – 2$
22	$y = q^3 – 2 \cdot a \cdot q + a^2 / q$

23	$y = a / c – k + (d + 1) \cdot 5$
24	$y = q^2 / 3 – a \cdot d + 5$
25	$y = a \cdot x^2 – b \cdot x / a + x / (x + a)$
26	$y = b^2 \cdot (x + d) + (d – 1) / c$
27	$y = (t^3 – 1) / (j – 4) – 5$
28	$y = (a^2 – b^2) / 2 + a \cdot (k + 1)$
29	$y = (a – c)^2 + 2 \cdot a \cdot c / k$
30	$y = (b^2 – 2 \cdot b) / (3 \cdot a + b)$

ПРИМЕЧАНИЕ:

Перед вычислением программа должна выполнить запрос значений параметров, обозначенных в выражениях буквенными символами, например так:

```
Введите значение a:
56
Введите значение b:
-67
```

Затем программа должна произвести вычисление, и вывести его результат, например так:

```
Ответ: y = 347
```

2. Открыть программу в отладчике. Выполнить программу в пошаговом режиме. После выполнения каждого шага заносить данные в таблицу:

Регистры				Флаги				Память
EAX	EBX	…	…	FC	FZ	…	…	

* в таблицу должны быть включены те элементы (регистры, флаги и пр.), которые имеют место в конкретном варианте программы

4. По результатам п.2 и п.3 сделайте выводы.

Содержание отчёта:

1. Название работы.
2. Цель работы.
3. Постановка задачи.

4. Текст программ с комментариями.

5. Результаты выполнения программы в пошаговом режиме в отладчике.

6. Выводы.

Лабораторная работа №6

Исследование работы арифметического сопроцессора

Цель работы:

- знакомство с работой арифметического сопроцессора;
- исследование инструкций сопроцессора;
- получение навыков работы с отладчиком.

Немного теории…

Ознакомиться с микроархитектурой и системой команд сопроцессора. Описание системы команд можно найти, например, здесь https://prog-cpp.ru/asm-coprocessor-command/ .

Для вывода результатов вычислений в консоль или окно необходимо двоичный код результата преобразовать в строку. Для этого можно использовать функцию *FpuFLtoA*.

Функция *FpuFLtoA* предназначена для вывода вещественного числа на экран. Данная функция не является API-функцией, а является внутренней MASM32-функцией и входит в специальную библиотеку *FPU.LIB*. Подробное описание функции можно найти в справке MASM32, которая обычно располагается в каталоге *\masm32\help\fpuhelp.chm*.

Функция *FpuFLtoA* может работать только с 80-битовыми числами и имеет четыре параметра:

FpuFLtoA (*lpSrc1*, *lpSrc2*, *lpszDest*, *uID*)

– *lpSrc1* – адрес 80-битового числа, которое будет выводиться на экран (этот параметр игнорируется, если *uID* имеет значение *SRC1_FPU*);

– *lpSrc2* – беззнаковое целое, указывающее количество десятичных знаков после запятой или адрес беззнакового целого (в зависимости от параметра *uID*);

– *lpszDest* – адрес буфера, куда будут записаны символы, в которые преобразуется число. Буфер должен быть достаточно большим, чтобы вместить возвращаемые символы. Самое большое число может иметь 17 символов перед знаком десятичной точки, знак десятичной точки, 15 десятичных цифр после точки, и завершающий нуль (таким образом, максимум 34 символа);

– *uID* – флаги, управляющие работой функции. Один из флагов *SRC1_?* может объединяться с помощью операции *OR* только с одним из *SRC2_?* и одним из *STR_?* флагов (Флаг *STR_REG* не нужно объединять оператором *OR*, если строка должна быть возвращена в десятичном формате).

Значение *uID* флагов:

– *SRC1_FPU* – первый параметр функции *Src* не используется (уже в FPU);

– *SRC1_REAL* – первый параметр *Src* должен быть адресом 80-битного числа хранящегося в обычной памяти;

– *SRC2_DMEM* – второй параметр *Src2* должен быть адресом 32-битного беззнакового числа;

– *SRC2_DIMM* – второй параметр *Src2* должен быть простым 32-битным беззнаковым числом;

– *STR_REG* – строка должна быть возвращена в десятичном формате;

– *STR_SCI* – строка должна быть возвращена в научном формате.

Для оконного вывода результатов необходимо изменить параметры линковщика в файле *Run.bat*, а именно:

```
C:\masm32\bin\link.exe /SUBSYSTEM:WINDOWS /LIBPATH:C:\masm32\lib\
Test.obj
```

Постановка задачи.

1. Согласно номеру студента в журнале группы выбрать вариант задания и написать программу вычисления заданного выражения на языке Ассемблера.
2. Выполнить программу в пошаговом режиме в отладчике и заполнить таблицу, отображающую состояние регистров и флагов сопроцессора.
3. Прокомментировать полученный результат.

Задание.

Исследовать выполнение арифметических операций сопроцессором. Результат вывести на экран функцией MessageBox.

1. Вычислить 6 значений функции $Y_n = 25 \cdot x^3 - 2{,}1$ (x изменяется с шагом 0,2).
2. Вычислить 5 значений функции $Y_n = 7 \cdot x^3 / (2 \cdot x^2 + 1{,}6)$ (x изменяется от 1 с шагом 4).
3. Вычислить 4 значения функции $Y_n = 37 / (2 \cdot x^2 + 7{,}3)$ (x изменяется от 1 с шагом 2).
4. Вычислить 5 значений функции $Y_n = 5{,}1 \cdot x^2 + 5{,}3$ (x изменяется от 4,7 с шагом 3). Результат округлить к ближайшему целому числу и вывести.
5. Вычислить 6 значений функции $Y_n = 4 \cdot x / (x + 5)$ (x изменяется от 3 с

шагом 1,25). Результат округлить к целому числу.

6. Вычислить 4 значения функции: $Y_n = 125 / (3 \cdot x^2 - 1{,}1)$ (x изменяется от 3 с шагом 1,5). Результат округлить в меньшую сторону.

7. Вычислить 4 значения функции $Y_n = 2{,}5 \cdot x^2 - 3{,}2$ (x изменяется от 4 с шагом 1).

8. Вычислить 3 значения функции $Y_n = 25 \cdot x^2 + 2{,}1$ (x изменяется от 3 с шагом 2,5).

9. Вычислить 4 значения функции $Y_n = 150 / (x^2 - 7)$ (x изменяется от 2 с шагом 3,1).

10. Вычислить 6 значений функции $Y_n = 256 / (3 \cdot x^2 + 31)$ (x изменяется от 2 с шагом 3).

11. Найти первое значение аргумента функции $Y_n = 7 \cdot (x + 0{,}3)$, при котором младшие целые цифры результата выполнения функции будут равны 15 (x изменяется от 2 с шагом 3,5).

12. Найти целое значение аргумента, при котором функция $Y_n = 10 / (x^3 + 1{,}7)$ станет меньше 0,3 (x изменяется от 2 с шагом 2,5).

13. Найти значение x, при котором функция $Y_n = 3 \cdot x / 4 - 6$ будет равна 12,5.

14. Найти значение x, при котором выполняется равенство $8 \cdot \text{arctg } 0{,}1 + \text{arctg } x = \pi/4$.

15. Определить номер элемента функции $Y_n = 2 \cdot x + 5$, при котором сумма элементов превысит 12 000.

16. Найти значение x, при котором функция $\lg(2 \cdot x + 3)$ будет больше 2,5.

17. Найти целое значение аргумента, при котором функция $Y = 5{,}6 \cdot x / 3 \cdot x^2$ будет больше 125.

18. Найти целое значение аргумента, при котором функция $Y = 2 \cdot x + 30$ будет больше 100.

19. Найти первое значение аргумента функции $Y = 9 \cdot (x^2 + 0{,}6)$, при котором младшие целые цифры результата выполнения функции будут равны 12 (x изменяется от 3 с шагом 5,5).

20. Найти целое значение аргумента, при котором функция $Y_n = 2 \cdot x / 5 + 4$ превысит 400.

21. Вычислить 5 значений функции $Y_n = 5{,}2 \cdot \ln(\sin x)$ (x изменяется в градусах с шагом 1,5).

22. Вычислить 4 значения функции $Y_n = 5 \cdot x + \cos x$ (x изменяется от 0,4 с шагом 0,15).

23. Вычислить 6 значений функции $Y_n = 4{,}5 \cdot \lg(\text{tg } x)$ (x изменяется в градусах с шагом 10).

24. Вычислить 6 значений функции $Y_n = 12 \cdot x - \sin x$ (x изменяется с шагом 0,05).

25. Вычислить 8 значений функции $Y_n = 3{,}3 \cdot \log_2(x^2 + 1)$ (x изменяется с шагом 0,3).

26. Вычислить 4 значения функции $Y_n = 5{,}1 \cdot (\sin x)$ (x изменяется в градусах с шагом 8,5).

27. Вычислить 6 значений функции $Y_n = \lg(x^2 + \sqrt{x})$ (x изменяется с шагом 0,3).

28. Вычислить 5 значений функции $Y_n = 4{,}3 \cdot (x^2 + 1)$ (x изменяется с шагом 1,7).

29. Вычислить 6 значений функции $Y_n = 6{,}2 \cdot \lg(\cos x)$ (x изменяется в градусах с шагом 1,5).

30. Вычислить 7 значений функции $Y_n = 3{,}1 + 2 / \cos x$ (x изменяется в градусах от 10 с шагом 8).

Пример.

Вычислить 5 значений функции $Y_n = 4{,}3 \cdot (x^2 + 1)$ (x изменяется с шагом 1,7).

```
;-----------------------------------------
; Программа на masm32 для Windows
; Исследование работы арифметического сопроцессора
; Вычислить 5 значений функции Yn = 4,3*(x^2 + 1)
; (x изменяется с шагом 1,7)
;-----------------------------------------

.686                            ; в программе будут использоваться
                                ; команды процессора Pentium Pro
.model flat, stdcall            ; модель памяти и соглашение
                                ; о передаче параметров
option casemap :none            ; включается чувствительность
                                ; к регистру

; Библиотеки и подключаемые файлы проекта
;-----------------------------------------
include C:\masm32\include\windows.inc
include C:\masm32\include\kernel32.inc
include C:\masm32\include\user32.inc
include C:\masm32\include\fpu.inc
                                ; содержит прототип функции FpuFLtoA
includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\user32.lib
includelib C:\masm32\lib\fpu.lib

; Сегмент даных
;-----------------------------------------
.data                           ; инициализированные данные
MsgBoxTitle byte "Операции в сопроцессоре x87", 0
                                ; заголовок окна
MsgBoxText  db  "Вычисление функции Yn = 4,3*(x^2 + 1),", 13,
```

```
                "где x изменяется с шагом 1,7", 13, 13,
                "y1="
res1  db 14 DUP(0), 10, 13        ; зарезервировать 14 байт для первого
                                  ; результата и поместить туда 0
      db "y2="
res2  db 14 DUP(0), 10, 13
      db "y3="
res3  db 14 DUP(0), 10, 13
      db "y4="
res4  db 14 DUP(0), 10, 13
      db "y5="
res5  db 14 DUP(0), 10, 13

CrLf  equ 0A0Dh
y1    TBYTE       0.0             ; тип 80 бит без знака (TBYTE = dt)
y2    dt          0.0
y3    dt          0.0
y4    dt          0.0
y5    dt          0.0
x     DWORD       3.0             ; тип 32 бита без знака (DWORD = dd)
op1   dd          4.3
op2   dd          1.0
zero  dd          0.0
step  dd          1.7

.data?                            ; неинициализированные данные

.const                            ; константы

; Сегмент кода
;-----------------------------------------
.code
start:                            ; метка (точка входа в программу)
      finit                       ; инициализация регистров FPU
                                  ; (CWR = 037Fh, SWR = 0h, TWR = FFFFh,
                                  ; DPR = 0h, IPR = 0h)
      mov ecx, 5                  ; счётчик X
m1:                               ; метка начала цикла
      fld x                       ; x^2
      fmul x
      fadd op2                    ; x^2+1
      fmul op1                    ; 4,3*(x^2+1)
      fld x                       ; увеличение X на величину шага
      fadd step
      fstp x
      loop m1                     ; если ecx = ecx - 1 ≠ 0,
                                  ; переходим на m1
      fstp y5                     ; сохраняем стек в память
      fstp y4
```

```
    fstp y3
    fstp y2
    fstp y1

    ; преобразование результатов вычислений в массив символов
    invoke FpuFLtoA, addr y1, 10, addr res1, SRC1_REAL or SRC2_DIMM
    mov word ptr res1 + 14, CrLf
    invoke FpuFLtoA, addr y2, 10, addr res2, SRC1_REAL or SRC2_DIMM
    mov word ptr res2 + 14, CrLf
    invoke FpuFLtoA, addr y3, 10, addr res3, SRC1_REAL or SRC2_DIMM
    mov word ptr res3 + 14, CrLf
    invoke FpuFLtoA, addr y4, 10, addr res4, SRC1_REAL or SRC2_DIMM
    mov word ptr res4 + 14, CrLf
    invoke FpuFLtoA, addr y5, 10, addr res5, SRC1_REAL or SRC2_DIMM

    ; вывод результатов вычислений
    invoke MessageBox, NULL, addr MsgBoxText, addr MsgBoxTitle,
MB_ICONINFORMATION

    invoke ExitProcess, NULL   ; функция завершения с параметром NULL
end start                      ; окончание программы
```

Содержание отчёта.

1. Название работы.
2. Цель работы.
3. Постановка задачи.
4. Текст программ с комментариями.
5. Результаты выполнения программы в пошаговом режиме в отладчике.
6. Выводы.