Лекция А2
БГТУ, ФИТ, ПОИТ, 3 семестр,
Конструирование программного обеспечения

Модели памяти Ассемблера. Основные понятия языка ассемблера

План лекции:
- основные модели памяти Ассемблера;
- регистры 32-битной архитектуры центрального процессора;
- основные понятия языка ассемблера;
- препроцесссор;
- примеры.

1. Плоская модель памяти (flat) (архитектура Win32):
приложению для кода и данных предоставляется один непрерывный сегмент;
базы сегментов кода (CS) и сегмента данных (DS) равны 0;
верхняя граница – 0xffffffff (4ГБ);
виртуальные смещения совпадают с линейными адресами.

Механизм управления памятью полностью аппаратный.

Модели использования оперативной памяти:
- сегментированная модель;
- страничная модель;
- плоская модель.

В сегментированной модели память для программы делится на непрерывные области памяти, называемые сегментами.
Сегмент представляет собой независимый, поддерживаемый на аппаратном уровне блок памяти. Операционная система размещает сегменты программы в оперативной памяти по определенным физическим адресам.
Адреса помещаются в сегментные регистры (для сегмента кода – регистр **CS**, для сегмента данных – регистр **DS**, для сегмента стека – регистр **SS**).
Для доступа к данным внутри сегмента обращение производится относительно начала сегмента линейно, начиная с 0 и заканчивая адресом, равным размеру сегмента.

Страничная модель памяти – это надстройка над сегментной моделью.
Оперативная память делится на блоки фиксированного размера – страницы (число их должно быть кратно степени двойки).
Программа также разбивается на фрагменты – страницы (все фрагменты программы имеют одинаковую длину, за исключением, возможно, последней страницы).
Память разбивается на физические страницы, а программа – на виртуальные страницы.

Плоская модель управления памятью.
Программа состоит из одного сегмента, который разбит на страницы. Получаем страничный механизм работы с виртуальной памятью. Базы всех сегментов установлены в 0, а лимиты в 4 гигабайта.

Основные модели памяти Ассемблера:

Модель памяти	**Адресация кода**	**Адресация данных**	**Операционная система**	**Чередование кода и данных**
TINY	NEAR	NEAR	MS-DOS	Допустимо
SMALL	NEAR	NEAR	MS-DOS, Windows	Нет
MEDIUM	FAR	NEAR	MS-DOS, Windows	Нет
COMPACT	NEAR	FAR	MS-DOS, Windows	Нет
LARGE	FAR	FAR	MS-DOS, Windows	Нет
HUGE	FAR	FAR	MS-DOS, Windows	Нет
FLAT	NEAR	NEAR	Windows NT, Windows 2000, Windows XP,	Допустимо

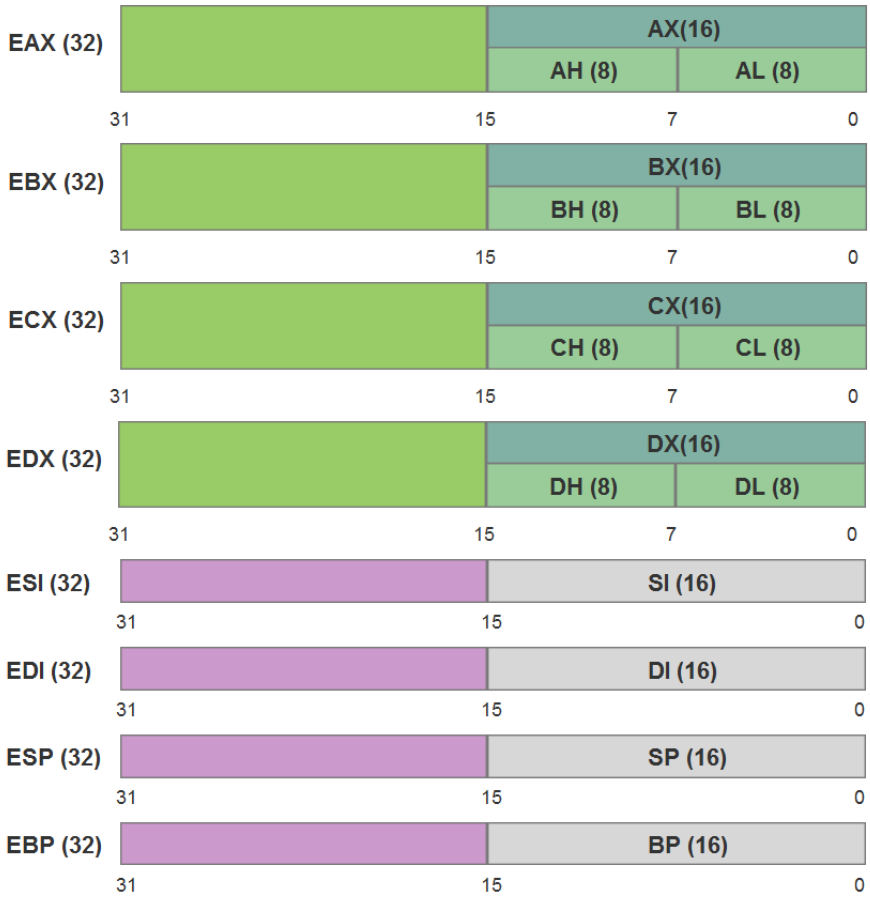
1. Модель **TINY** – код, данные и стек размещаются в одном и том же физическом сегменте размером до 64 Кб, работает в 16-разрядных приложениях.
2. Модель **SMALL** – код размещается в одном сегменте, а данные и стек – в другом.
3. Модель **COMPACT** – код размещается в одном сегменте, а для хранения данных могут использоваться несколько сегментов.
4. Модель **MEDIUM** – код размещается в нескольких сегментах, а все данные – в одном.
5. Модель **LARGE** и **HUGE** – и код, и данные могут занимать несколько сегментов.
6. Модель **FLAT** – используется в 32-разрядных операционных системах – это несегментированная конфигурация программы. Максимальный размер сегмента, содержащего данные, код и стек, – 4 Мб.

2. Набор регистров 32-битной архитектуры центрального процессора

Регистры общего назначения. Названия регистров происходят от их назначения:

- **EAX/AX/AH/AL** (аккумулятор) – применяется для хранения промежуточных данных, ***автоматически*** применяется при операциях умножения, деления для хранения ***первого*** операнда;
- **EBX/BX/BH/BL** (база) – регистр базы, применяется для хранения базового адреса некоторого объекта в памяти (например, массива);
- **ECX/CX/CH/CL** (регистр-счетчик) – ***автоматически*** применяется в качестве счетчика цикла, его использование может быть неявно и скрыто в алгоритме работы соответствующей команды;
- **EDX/DX/DH/DL** – регистр данных, применяется в операциях умножения и деления, используется как расширение регистра-аккумулятора EAX при работе с 32- разрядными числами;
- **ESI/SI** – индекс источника;
- **EDI/DI** – индекс приёмника (получателя);
- **ESP/SP** – регистр указателя стека;
- **EBP/BP** – регистр указателя базы.

Подрегистры AX, BX, CX, DX позволяют независимо обращаться к их старшей (H) и младшей (L) половине.
Подрегистры старшей (H) и младшей (L) половины имеют размерность 8 бит и названия AH, AL, BH, BL, CH, CL, DH, DL сответственно.



Указатель команд:

Регистр **EIP** (указатель команд) содержит смещение в сегменте кода следующей выполняемой команды. Как только команда начинает выполннение, значение IP увеличивается на ее длину и будет адресовать следующую команду.

Регистр флагов.

Сегментные регистры:

Сегментные регистры: **CS** (регистр сегмента кода), **DS** (регистр сегмента данных), **ES**, **FS**, **GS** (регистры сегментов дополнительных данных), **SS** (регистр сегмента стека).

3. Регистры общего назначения. Примеры

```
; Регистры общего назначения

.586P                       ; система команд(процессор Pentium)
.MODEL FLAT, STDCALL        ; модель памяти, соглашение о вызовах
includelib kernel32.lib     ; компановщику: компоновать с kernel32

ExitProcess PROTO : DWORD   ; прототип функции для завершения процесса Windows

.STACK 4096                 ; выделение стека объемом 4 мегабайта

.CONST; сегмент констант
dreg dword 12345678h        ; двойное слово длиной 4 байта = ox12345678

.DATA                       ; сегмент данных

.CODE                       ; сегмент кода

main PROC                   ; точка входа main

mov eax, dreg               ; копировать 4 байта в регистр EAX
mov ebx, dreg               ; копировать 4 байта в регистр EBX
mov ecx, dreg               ; копировать 4 байта в регистр EDX
mov edx, dreg               ; копировать 4 байта в регистр ECX


        push 0              ; код возврата процесса Windows(параметр ExitProcess)
        call ExitProcess    ; так завершается любой процесс Windows
        main ENDP           ; конец процедуры

        end main            ; конец модуля main
```

Регистры

```
EAX = 12345678 EBX = 12345678 ECX = 12345678 EDX = 12345678 ESI = 00000000
  EDI = 00000000 EIP = 00821027 ESP = 002FFDD8 EBP = 002FFDE0 EFL = 00000244
```

Контрольные значения 1

Имя	Значение
eax	0x12345678
ax	0x5678
ah	0x56 'V'
al	0x78 'x'
ebx	0x12345678
bx	0x5678
bh	0x56 'V'
bl	0x78 'x'
ecx	0x12345678
cx	0x5678
ch	0x56 'V'
cl	0x78 'x'
edx	0x12345678
dx	0x5678
dh	0x56 'V'
dl	0x78 'x'

```
; Регистры общего назначения

.586                          ; система команд(процессор Pentium)
.MODEL FLAT, STDCALL          ; модель памяти, соглашение о вызовах
includelib kernel32.lib       ; компановщику: компоновать с kernel32
ExitProcess PROTO : DWORD     ; прототип фун                  я процесса Windows
.STACK 4096                   ; выделение стека объемом 4 мегабайта

.CONST                        ; сегмент констант
dreg dword 12345678h          ; двойное слово длиной 4 байта = ox12345678

.DATA                         ; сегмент данных
dmem  dword ?                 ; 4 байта
wmem  word  ?                 ; 2 байта
bmemh byte  ?                 ; 1 байт
bmeml byte  ?                 ; 1 байт

.CODE                         ; сегмент кода

main PROC                     ; точка входа main

   mov eax, dreg              ; копировать 4 байта(dreg) в регистр EAX

   mov dmem, eax              ; копировать 4 байта из регистра EAX
   mov wmem, ax               ; копировать 2 байта из регистра AX
   mov bmemh, ah              ; копировать 1 байта из регистра AH
   mov bmeml, al              ; копировать 1 байта из регистра AL

   push 0                     ; код возврата процесса Windows(параметр ExitProcess)
   call ExitProcess           ; так завершается любой процесс Windows
main ENDP                     ; конец процедуры

end main                      ; конец модуля main
```

Контрольные значения 1

Имя	Значение	Ти
eax	0x12345678	un
ax	0x5678	un
ah	0x56 'V'	un
al	0x78 'x'	un
dmem	0x12345678	un
wmem	0x5678	un
bmemh	0x56 'V'	un
bmeml	0x78 'x'	un

Регистры

```
EAX = 12345678 EBX = 7EE4A000 ECX = 00000000 EDX = 000A1005 ESI = 00000000
EDI = 00000000 EIP = 000A102B ESP = 004DFF74 EBP = 004DFF7C EFL = 00000244
```

```
; Регистры общего назначения

.586                            ; система команд(процессор Pentium)
.MODEL FLAT, STDCALL            ; модель памяти, соглашение о вызовах
includelib kernel32.lib         ; компановщику: компоновать с kernel32
ExitProcess PROTO : DWORD       ; прототип                  шения процесса Windows
.STACK 4096          ; выделение стека объемом 4 мегабайта

.CONST               ; сегмент констант
.DATA                ; сегмент данных
.CODE                ; сегмент кода

main PROC            ; точка входа main
; некоторые инструкции используют регистры неявно
    mov eax, 4h      ; 0x04 -> EAX
    mov ebx, 3h      ; 0x03 -> EBX
    imul ebx         ; ebx*eax -> eax

    push 0               ; код возврата процесса Windows(параметр ExitProcess)
    call ExitProcess     ; так завершается любой процесс Windows
main ENDP                ; конец процедуры

end main                 ; конец модуля main
```

Контрольные значения 1

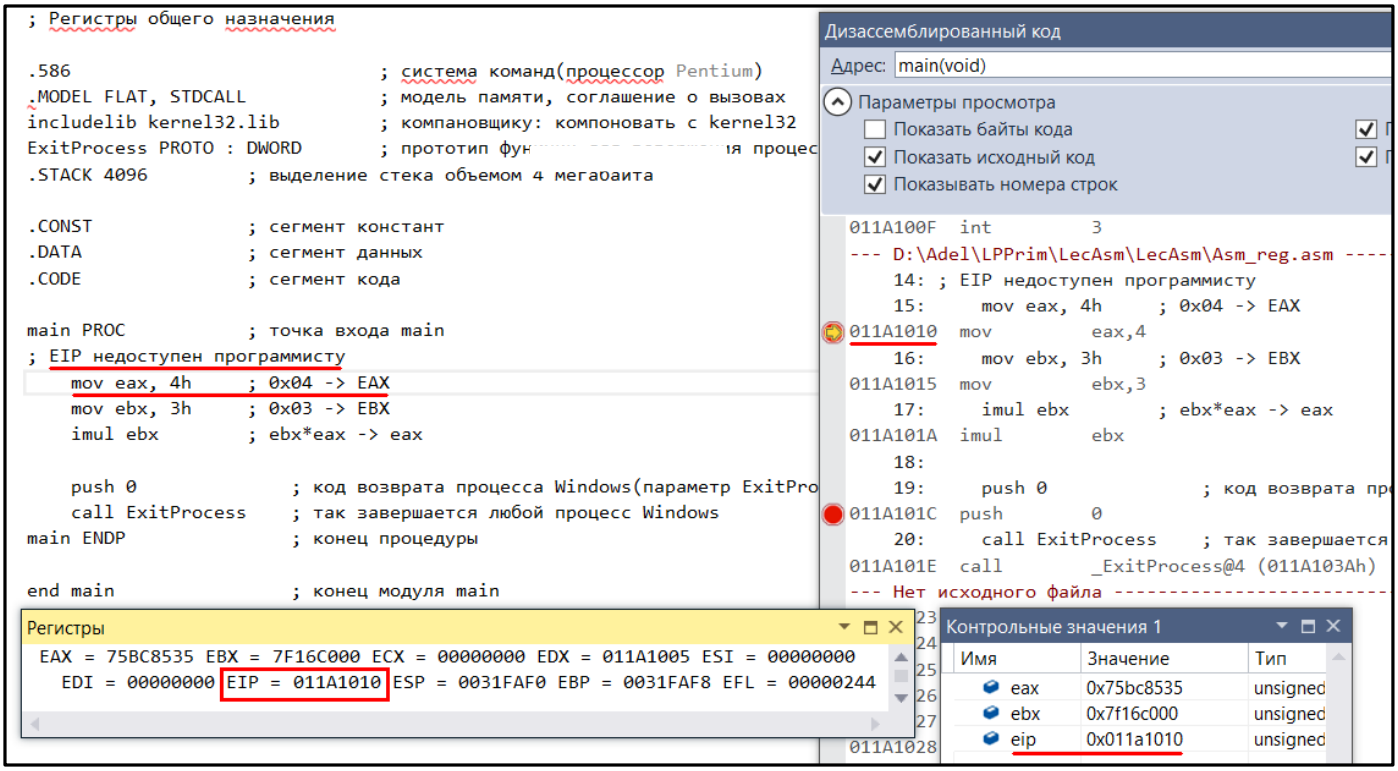
Имя	Значение	Тип
eax	0x0000000c	unsi
ebx	0x00000003	unsi

Регистры

```
EAX = 0000000C EBX = 00000003 ECX = 00000000 EDX = 00000000 ESI = 00000000
   EDI = 00000000 EIP = 012F101C ESP = 005BFE90 EBP = 005BFE98 EFL = 00000244
```

4. Регистры общего назначения: EIP

EIP (указатель команд, содержит адрес следующей команды) – непосредственно не доступен программисту, но его значение можно видеть в режиме отладки.



5. Регистры общего назначения: ESP

ESP (регистр указателя стека) содержит адрес вершины стека.

```
; Регистры общего назначения

.586                            ; система команд(процессор Pentium)
.MODEL FLAT, STDCALL            ; модель памяти, соглашение о вызовах
includelib kernel32.lib         ; компановщику: компоновать с kernel32
ExitProcess PROTO : DWORD       ; прототип функции для завершения процесса Windows
.STACK 4096                     ; выделение стека

.CONST                          ; сегмент констант
.DATA                           ; сегмент данных
staddr dword 0h, 1h, 2h, 3h     ; массив 4 * 4 байтб проинициализирован
.CODE                   ; сегмент кода

main PROC               ; точка входа main
; ESP - адрес вершины стека
    mov staddr, esp     ; esp->staddr
    push 1h             ; записать в стек 4 байта
    mov staddr+4, esp   ; esp->staddr+4
    push 2h             ; записать в стек 4 байта
    mov staddr+8, esp   ; esp->staddr+8
    push 3h             ; записать в стек 4 байта
    mov staddr+12, esp  ; esp->staddr+12
    add esp, 12         ; освободить стек : esp->staddr + 12

    push 77             ; код возврата процесса Windows(параметр ExitProcess)
    call ExitProcess    ; так завершается любой процесс Windows
main ENDP               ; конец процедуры

end main                ; конец модуля main
```

Контрольные значения 1

Имя	Значение	Ти
esp	0x00b3fba4	uns
*(&staddr)	0x00b3fba4	uns
*(&staddr+1)	0x00b3fba0	uns
*(&staddr+2)	0x00b3fb9c	uns
*(&staddr+3)	0x00b3fb98	uns

Регистры

```
EAX = 75BC8535 EBX = 7F155000 ECX = 00000000 EDX = 00D31005 ESI = 00000000
EDI = 00000000 EIP = 00D31031 ESP = 00B3FBA4 EBP = 00B3FBAC EFL = 00000210
```

6. Основные элементы языка ассемблера:

- константы (целочисленные, вещественные, символьные, строковые);
- выражения;
- зарезервированные слова;
- идентификаторы;
- директивы.

Константы:

суффикс	система счисления	символы	пример
h	шестнадцатеричная	цифры и буквы 0-F, если с буквы, то впереди ставим 0	0A3h
d или ничего	десятичная	цифры 0-9	345d
q или o	восьмеричная	цифры 0-7	48o
b	двоичная	цифры 0 и 1	1001b

целочисленные – вид представления: [{+|-}]*цифры*[*суффикс*],
где
знак {+|-}
цифры *цифра* [*цифра*]
цифра {0|1|2|3|4|5|6|7|8|9}

вещественные – вид представления: [*знак*]*цифры*.[*цифры*][*степень*],
где
знак {+|-}
цифры *цифра* [*цифра*]
цифра {0|1|2|3|4|5|6|7|8|9}
степень E[{+|-}]*цифры*

Примеры вещественных констант:
3.
+345.
-48.2E+05
-.2E-05

символьные – один символ, заключенный в одинарные или двойные кавычки. Символьная константа автоматически заменяется на соответствующий ей ASCII-код. Пример: "A", 's'.

строковые – последовательность символов, заключенных в одинарные или двойные кавычки, автоматически заменяется на последовательность кодов, соответствующих каждому символу строковай костанты.
Пример: "Hello, world".

Идентификаторы (последовательности допустимых символов, использующиеся для обозначения имен переменных, констант или названия меток)**:**

- один или несколько символов латинского алфавита;
- цифры;
- специальные знаки: _, ?, $, @;
- начинается с буквы.

Длина идентификатора до 247 символов.
Транслятор воспринимает лишь первые 32, а остальные игнорирует.
Регистр не учитывается.
Не должен совпадать зарезервированными словами языка ассемблера.

Комментарии:

однострочные – начинаются с символа «точка с запятой» (;).
Все символы после «;» до конца стороки игнорируются компилятором;

многострочные –
```
COMMENT !
первая сторока комментария
еще одна сторока комментария
!
```

Зарезервированные слова.
В языке ассемблера существует список зарезервированных слов.
Примеры:

все мнемоники команд (MOV и другие)
атрибуты переменных (BYTE и другие)
директивы компилятора MASM
опрераторы
встроенные идентификаторы ассемблера (как @data)

Директивы.
Директивы – команды, которые управляют процессом ассемблирования.
В отличие от команд, они ***не генерируют*** машинных кодов.
Например директива .CODE определяет в программе сегмент кода, .data определяет сегмент данных.

7. Команды на языке ассемблера

Команда – оператор программы, который непосредственно выполняется процессором.
Команды языка ассемблера – это символьная форма записи машинных команд. Команды имеют следующий синтаксис:

[метка] (необязательный)	мнемоника	[операнд(ы)]	[;комментарий]

Метка – идентификатор, с помощью которого, можно пометить участок кода или данных. Метка кода должна отделяться двоеточием.
Мнемоника команды – короткое имя, определяющее тип выполняемой процессором операции.
Операнд определяет данные (регистр, ссылка на участок памяти, константное выражение), над которыми выполняется действие по команде, если операндов несколько, то они отделяются друг от друга запятыми.

8. Типы даннных

a. Внутренние типы данных

Команды языка ассемблера оперируют объектами, существующими в оперативной памяти, т.е. байтом и его производными (слово, двойное слово и т.д.). Существует лишь понятие размера занимаемых данных.

В MASM определены несколько ***внутренних типов данных***, значения которых могут быть присвоены переменным, либо они могут являться результатом выполнения выражения.

byte	1 байт без знака
sbyte	1 байт со знаком
word	2 байта без знака (***слово***) (в режиме реальной адресации используется для хранения ближнего указателя)
sword	2 байта со знаком
dword	4 байта без знака (***двойное слово***) (в защищенном режиме используется для хранения ближнего указателя)
sdword	4 байта со знаком
fword	48-битов (в защищенном режиме используется для хранения дальнего указателя)
qword	64-битное целое
tbyte	80-битное (10-байтовое) целое
real4	4-байтовое IEEE-754
real8	8-байтовое IEEE-754
real10	10-байтовое IEEE-754

b. Оператор определения данных

Синтаксис инструкции определения данных:

[имя] *директива* *инициализатор* [*инициализатор*]

Имя переменной (идентификатор) – метка, значение которой соответствует смещению данной переменной относительно начала сегмента, в котором она размещена.

С помощью директив в программе выделяется память под одну или несколько знаковых или беззнаковых переменных соответствующей длинны:

BYTE определяет беззнаковый ***байт***.

SBYTE определяет знаковый ***байт***.

WORD определяет слово без знака (***2 байта***) для хранения 16-разрядных целых значений.

SWORD определяет слово со знаком (***2 байта***) для хранения 16-разрядных целых значений.

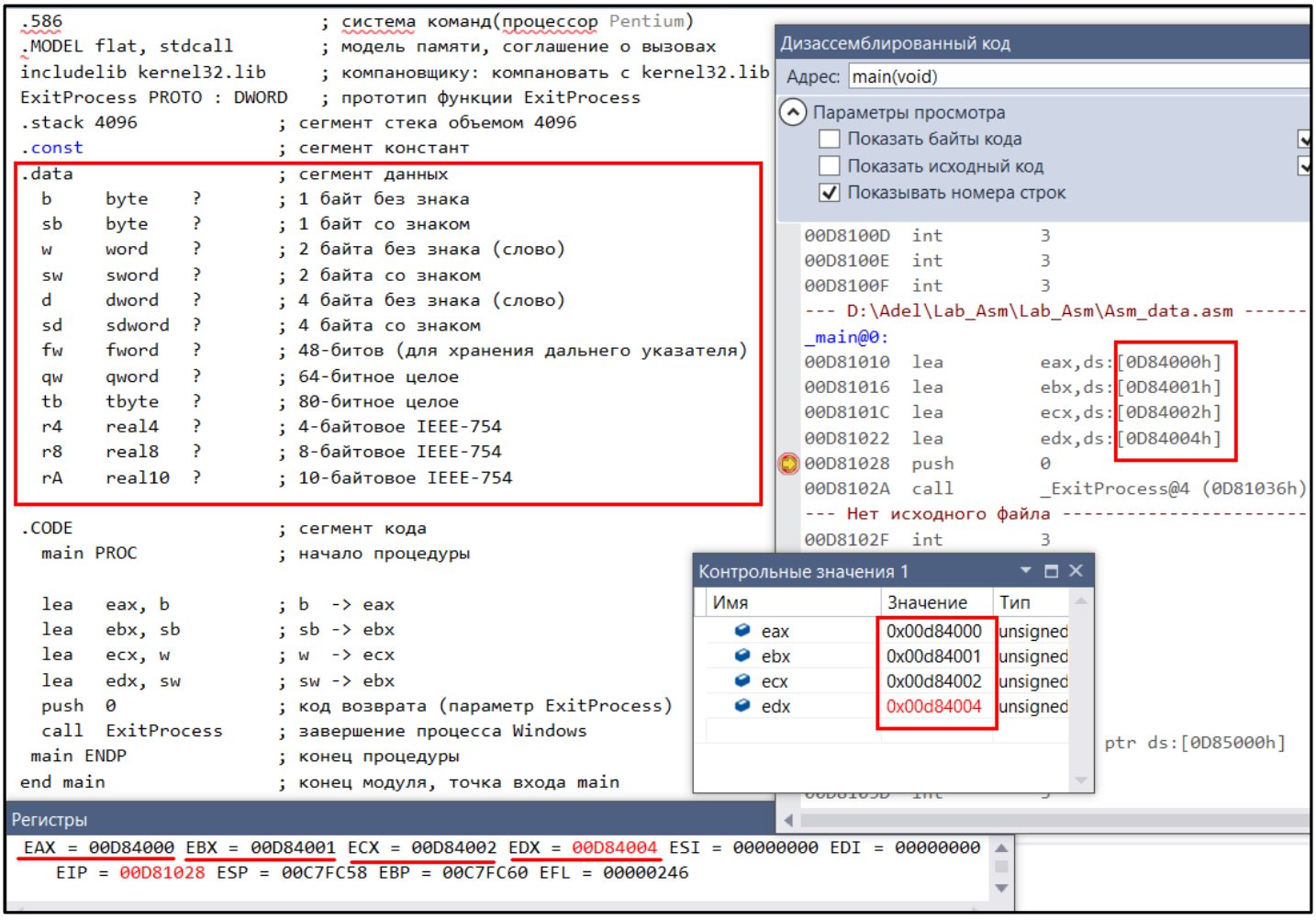
DWORD определяет слово без знака (***4 байта***) для хранения 32-разрядных целых значений.

SDWORD определяет слово со знаком (***4 байта***) для хранения 32-разрядных целых значений.

TBYTE определяет ***10 байтов*** для хранения 80-разрядных целых значений. Этот тип данных в основном используется для хранения десятичных упакованных целых чисел (двоично-кодированных целых чисел). Для работы с этими числами используется специальный набор команд математического сопроцессора.

Если инициализатор равен **?**, то выделяется память для хранения не инициализированных переменных, заданной длины.

Пример:



9. Инициализация данных

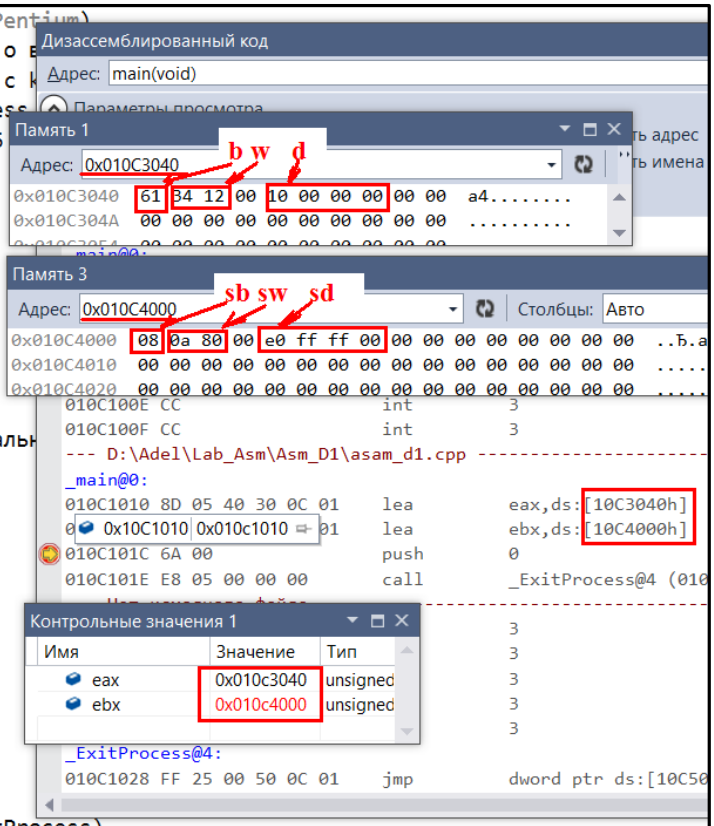
```
.586                    ; система команд(процессор Pentium)
.MODEL flat, stdcall    ; модель памяти, соглашение о в
includelib kernel32.lib ; компановщику: компановать с к
ExitProcess PROTO : DWORD ; прототип функции ExitProcess
.stack 4096             ; сегмент стека объемом 4096

.const                  ; сегмент констант
b       byte 'a'        ; 1 байт без знака
w       word 1234h      ; 2 байта без знака(слово)
d       dword 4096      ; 4 байта без знака(слово)
.data                   ; сегмент данных
sb  byte 1000b          ; 1 байт со знаком
sw  sword 800ah         ; 2 байта со знаком
sd  sdword -8192        ; 4 байта со знаком
fw  fword ?             ; 48 - битов(для хранения дальн
qw  qword ?             ; 64 - битное целое
tb  tbyte ?             ; 80 - битное целое
r4  real4 ?             ; 4 - байтовое IEEE - 754
r8  real8 ?             ; 8 - байтовое IEEE - 754
rA  real10 ?            ; 10 - байтовое IEEE - 754

.CODE                   ; сегмент кода
main PROC               ; начало процедуры

lea eax, b              ; b->eax
lea ebx, sb             ; sb->ebx

push    0               ; код возврата(параметр ExitProcess)
call    ExitProcess     ; завершение процесса Windows
main ENDP               ; конец процедуры
end main                ; конец модуля, точка входа main
```



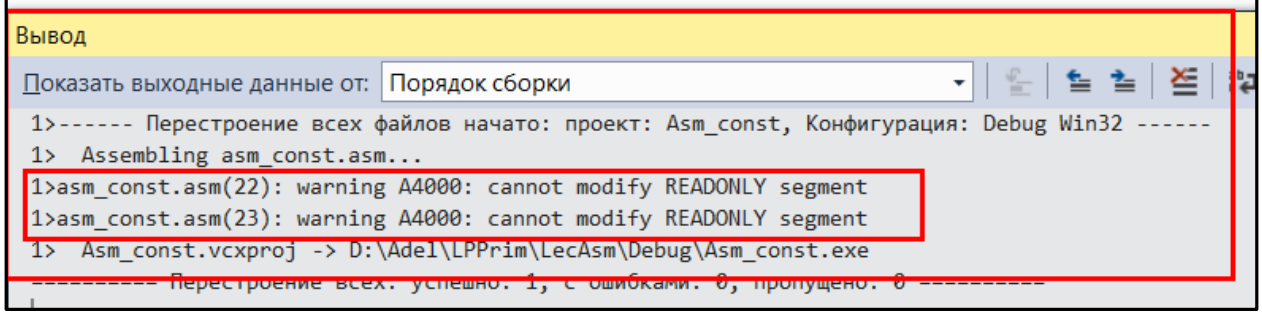
10. В сегменте .const память read only

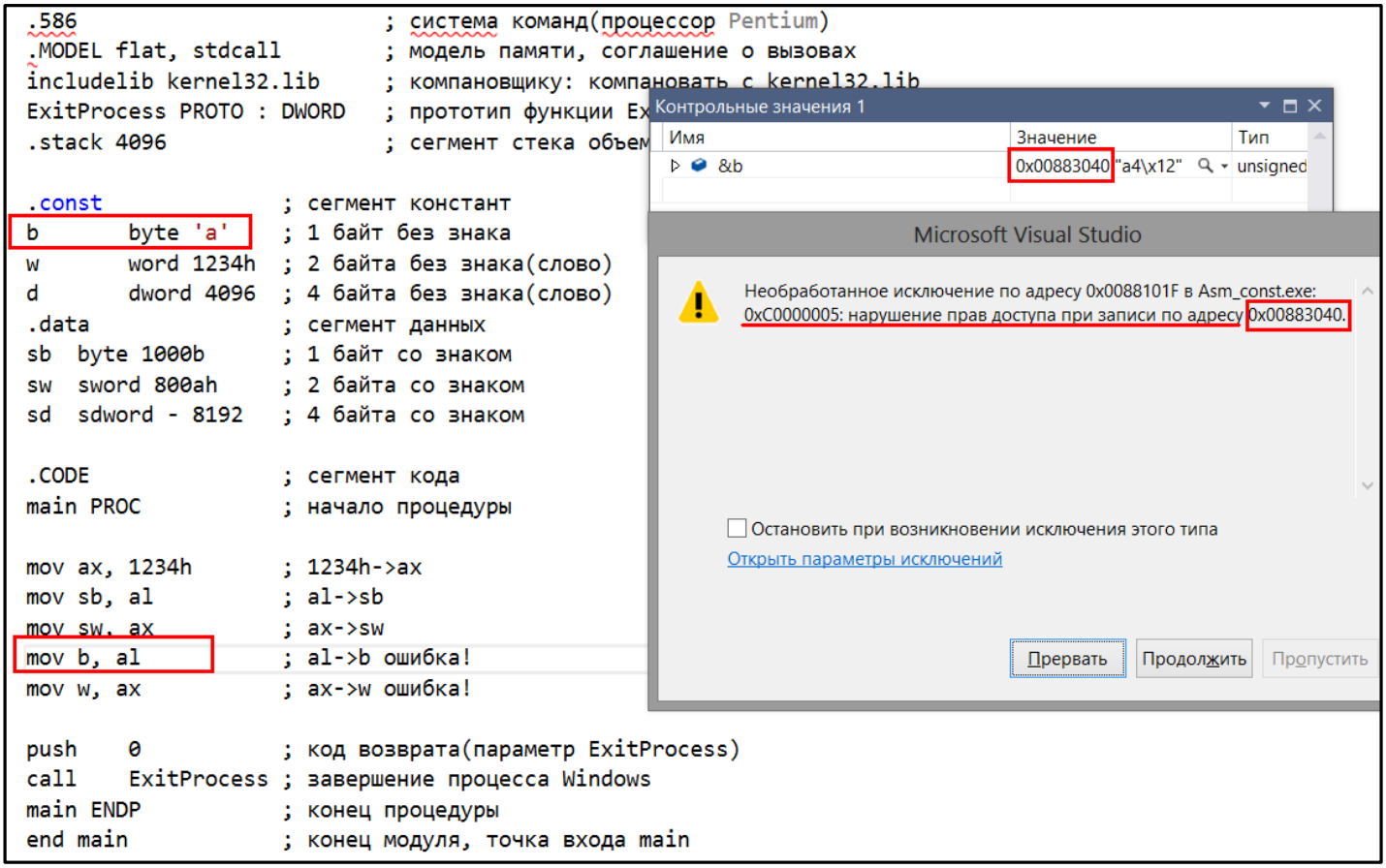
```
.586                            ; система команд(процессор Pentium)
.MODEL flat, stdcall            ; модель памяти, соглашение о вызовах
includelib kernel32.lib         ; компановщику: компановать с kernel32.lib
ExitProcess PROTO : DWORD       ; прототип функции ExitProcess
.stack 4096                     ; сегмент стека объемом 4096

.const                  ; сегмент констант
b       byte 'a'        ; 1 байт без знака
w       word 1234h      ; 2 байта без знака(слово)
d       dword 4096      ; 4 байта без знака(слово)
.data                   ; сегмент данных
sb  byte 1000b          ; 1 байт со знаком
sw  sword 800ah         ; 2 байта со знаком
sd  sdword - 8192       ; 4 байта со знаком

.CODE                   ; сегмент кода
main PROC               ; начало процедуры

mov ax, 1234h           ; 1234h->ax
mov sb, al              ; al->sb
mov sw, ax              ; ax->sw
mov b, al               ; al->b ошибка!
mov w, ax               ; ax->w ошибка!
```





11. Массивы и их инициализация

Для создания массива можно либо явно перечислить значения каждого элемента массива через запятую, либо воспользоваться оператором **DUP**.

a. Множественна инициализация

Если в операторе определения данных используется несколько инициализаторов, то присвоенная оператору метка относится только к первому элементу данных и этот элемент располагается со смещением 0 относительно этой метки.

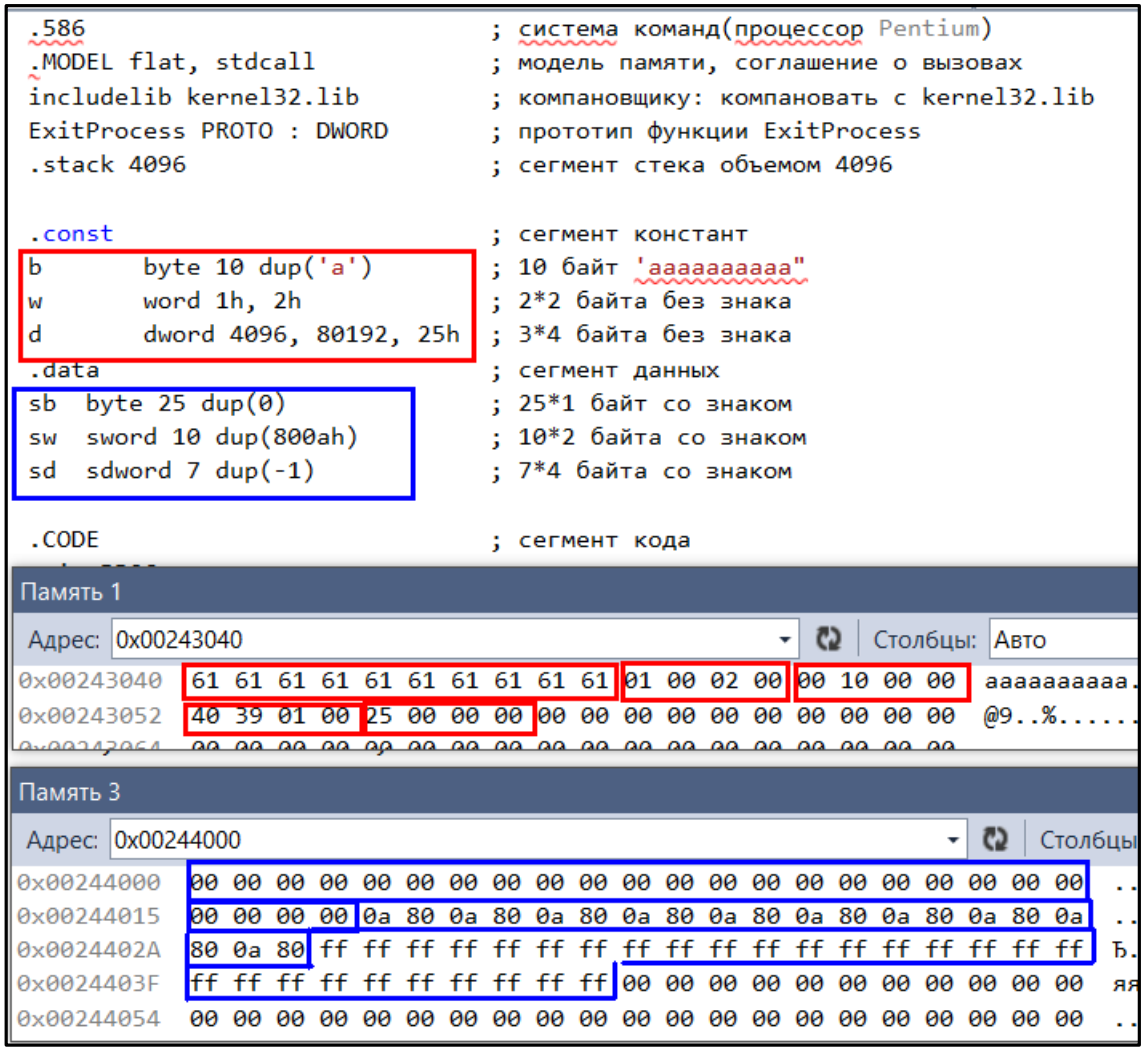
Следующий элемент располагается со смещением равным ***размеру элемента в байтах*** относительно метки оператора.

Метки нужны не для всех операторов определения данных.

В одном операторе определения данных могут использоваться инициализаторы, заданные в разных системах счисления. Кроме того, могут использоваться вперемешку как символы, так и строковые константы.

b. Оператор DUP

Оператор DUP используется для выделения памяти под массив, содержащий повторяющиеся значения байтов, которые могут быть инициализированы или нет. В качестве счетчика байтов используется ***константное выражение***.



12. Строки

Чтобы определить в программе текстовую строку, нужно составляющую ее последовательность символов заключить в кавычки.

Строка должна оканчиваться нулевым байтом, т.е. байтом, значение которого равно двоичному нулю.

13. Сокращения dd, dw, ...

В ***прежних версиях*** ассемблера для определения как знаковых, так и беззнаковых существует ***5 директив*** для определения данных. В новой версии MASM также можно пользоваться этими директивами:

Устаревшие	Новые	Описание
DB (define byte)	BYTE SBYTE	определяет переменную размером в 1 байт
DW (define word)	WORD SWORD	определяет переменную размеров в 2 байта (слово)
DD (define double word)	DWORD SDWORD	определяет переменную размером в 4 байта (двойное слово)
DQ (define quad word)	QWORD	определяет переменную размером в 8 байт (учетверённое слово)
DT (define ten bytes)	TBYTE	определяет переменную размером в 10 байт

```
.586                                ; система команд(процессор Pentium)
.MODEL flat, stdcall                ; модель памяти, соглашение о вызовах
includelib kernel32.lib             ; компановщику: компановать с kernel32.lib
ExitProcess PROTO : DWORD           ; прототип функции ExitProcess
.stack 4096                         ; сегмент стека объемом 4096

.const; сегмент констант
b       byte "Hello world!",0       ; строка
b1      byte 'H', 'e', 'l', 'l', 'o',' ','w','o','r','l','d','!',0
w       dw 1h, 2h                   ; word 2 * 2 байта без знака
d       dd 4096, 80192, 25h         ; dword 3 * 4 байта без знака
.data                               ; сегмент данных
sb  byte 25 dup(0)                  ; 25 * 1 байт со знаком
sw  dw 10 dup(800ah)                ; sword 10 * 2 байта со знаком
sd  dd 7 dup(-1)                    ; sdword 7 * 4 байта со знаком

.CODE                               ; сегмент кода
main PROC                           ; начало процедуры

push    0                           ; код возврата(параметр ExitProcess)
call    ExitProcess                 ; завершение процесса Windows
main ENDP                           ; конец процедуры
end main                            ; конец модуля, точка входа main
```

14. Препроцессор: символы, директива присваивания, счетчик команд

a. Директива присваивания (=) связывает символическое имя с целочисленным выражением.

```
.CODE; сегмент кода
AA = 64                         ; символ (символическое имя) AA
Esc_key = 27                    ; символ Esc_key(ASCII-код клавиши <Esc>)
CC = 25h                        ; символ CC

main PROC                       ; начало процедуры

mov eax, AA                     ; AA->eax
mov ebx, Esc_key                ; Esc_key->ebx
mov ecx, CC                     ; CC->ecx

push    0                       ; код возврата(парам
call    ExitProcess             ; завершение процесс
main ENDP                       ; конец процедуры
end main                        ; конец модуля, точк
```

Дизассемблированный код

Адрес: main(void)

Параметры просмотра

☑ Показать байты кода ☐ Показать исходный код ☑ Показывать номера строк

```
00FB100D CC                 int         3
00FB100E CC                 int         3
00FB100F CC                 int         3
--- D:\Adel\LPPrim\LecAsm\Asm_ECX\asm_ecx.asm ---------
_main@0:
00FB1010 B8 40 00 00 00     mov         eax,40h
00FB1015 BB 1B 00 00 00     mov         ebx,1Bh
00FB101A B9 25 00 00 00     mov         ecx,25h
00FB101F 6A 00              push        0
00FB1021 E8 06 00 00 00     call        _ExitProcess@
--- Нет исходного файла -------------------------------
00FB1026 CC                 int         3
00FB1027 CC                 int         3
```

Контрольные значения 1

Имя	Значение	Ти
eax	0x00000040	uns
ebx	0x0000001b	uns
ecx	0x00000025	uns

b. Оператор $ возвращает текущее значение счетчика команд.

```
.stack 4096                     ; сегмент стека объемом 4096

.const                          ; сегмент констант
AA = $                          ; счетчик команд
b       byte "Hello world!", 0  ; строка
b1      byte 'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'
w       dw 1h, 2h               ; word 2 * 2 байта без знака
d       dd 4096, 80192, 25h     ; dword 3 * 4 байта без знака
.data                           ; сегмент данных
BB = $                          ; счетчик команд
sb  byte 25 dup(0); 25 * 1 байт со знаком
sw  dw 10 dup(800ah)            ; sword 10 * 2 байта со знаком
sd  dd 7 dup(-1)                ; sdword 7 * 4 байта со знаком

.CODE; сегмент кода
CC = $                          ; счетчик команд

main PROC                       ; начало процедуры

mov eax, AA                     ; AA->eax адрес сегмента констант
mov ebx, BB                     ; BB->ebx адрес сегмента данных
mov ecx, CC                     ; CC->ecx адрес сегмента кода

push    0                       ; код возврата(параметр ExitProcess)
call    ExitProcess             ; завершение процесса Windows
```

Дизассемблированный код

Адрес: main(void)

Параметры просмотра

☐ Показать байты кода ☐ Показать исходный код ☐ Показывать номера строк

```
0090100D  int         3
0090100E  int         3
0090100F  int         3
--- D:\Adel\LPPrim\LecAsm\Asm_ECX\asm
_main@0:
00901010  mov         eax,903040h
00901015  mov         ebx,904000h
0090101A  mov         ecx,901010h
0090101F  push        0
00901021  call        _ExitProcess@4
--- Нет исходного файла -------------
00901026  int         3
```

Контрольные значения 1

Имя	Значение
eax	0x00903040
ebx	0x00904000
ecx	0x00901010

c. Вычисление длины строки.

```
ExitProcess PROTO : DWORD       ; прототип функции ExitProcess
.stack 4096                     ; сегмент стека объемом 4096

.const                          ; сегмент констант
b       byte "Hello world!", 0  ; строка
LEN_b = ($ - b)                 ; длина строки b
b1      byte 'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd'
w       dw 1h, 2h               ; word 2 * 2 байта без знака
d       dd 4096, 80192, 25h     ; dword 3 * 4 байта без знака
LEN_d = ($ - d)                 ; длина строки d
.data                           ; сегмент данных
sb  byte 25 dup(0)              ; 25 * 1 байт со знаком
sw  dw 10 dup(800ah)            ; sword 10 * 2 байта со знаком
LEN_sw = ($ - sw)               ; длина массива sw
sd  dd 7 dup(-1)                ; sdword 7 * 4 байта со знаком

.CODE                           ; сегмент кода
CC = $                          ; счетчик команд

main PROC                       ; начало процедуры

mov eax, LEN_b                  ; LEN_b->eax
mov ebx, LEN_d                  ; LEN_d->ebx
mov ecx, LEN_sw                 ; LEN_sw->ecx

push    0                       ; код возврата(параметр ExitProcess
```

Дизассемблированный код

Адрес: main(void)

Параметры просмотра

Показать байты кода

Показать исходный код

Показывать номера строк

```
0034100F  int         3
--- D:\Adel\LPPrim\LecAsm\Asm_ECX\
_main@0:
00341010  mov         eax,0Dh
00341015  mov         ebx,0Ch
0034101A  mov         ecx,14h
0034101F  push        0
00341021  call        _ExitProcess
--- Нет исходного файла ----------
00341026  int         3
00341027  int         3
00341028  int         3
```

Контрольные значения 1

Имя	Значение
eax	0x0000000d
ebx	0x0000000c
ecx	0x00000014

15. Препроцессор.

Препроцессор – это текстовый процессор, управляющий преобразованием текста исходного кода в ходе первого этапа трансляции, позволяет сократить время написания кода, а сам код автоматически привести к определенному виду. Препроцессор не анализирует исходный текст. Предобработка кода препроцессором выполняется на первой стадии компиляции в соответствии с макросами и выполняет директивы.

a. директива **EQU**

Директива эквивалентности **EQU** позволяет определять константы:

`имя` `EQU` `<операнд>`

Все вхождения ***имени*** заменяются ***операндом***. Операндом может быть константное выражение, строка, другое имя.

> Директива EQU отличается от директивы присваива ния (=) тем, что определенный с ее помощью символ нельзя переопределить в одном и том же исходном файле.

b. директива **TEXTEQU**

Директива **TEXTEQU** создает текстовый макрос.

Существует три формата директивы texteqU:

```
имя     TEXTEQU     <текст>
имя     TEXTEQU     текстовый__макрос
имя     TEXTEQU     %константное_выражение
```

```
.586                            ; система команд(процессор Pentium)
.MODEL flat, stdcall            ; модель памяти, соглашение о вызовах
includelib kernel32.lib         ; компановщику: компановать с kernel32.lib
ExitProcess PROTO : DWORD       ; прототип функции ExitProcess
.stack 4096                     ; сегмент стека обьемом 4096
HW TEXTEQU <"Hello world!">

.const                          ; сегмент констант
b       byte HW, 0              ; строка
LEN_b = ($ - b)                 ; длина строки b
b1      byte 'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '!', 0
w       dw 1h, 2h               ; word 2 * 2 байта без знака
d       dd 4096, 80192, 25h     ; dword 3 * 4 байта без знака
LEN_d = ($ - d)                 ; длина строки d
.data                           ; сегмент данных
sb  byte 25 dup(0)              ; 25 * 1 байт со знаком
sw  dw 10 dup(800ah)            ; sword 10 * 2 байта со знаком
LEN_sw = ($ - sw)               ; длина массива sw
sd  dd 7 dup(-1)                ; sdword 7 * 4 байта со знаком

.CODE                           ; сегмент кода
AA EQU 2
BB EQU LEN_b + 1
CC EQU d + 4 * AA

main PROC                       ; начало процедуры

mov eax, AA                     ; AA->eax
mov ebx, BB                     ; BB->ebx
mov ecx, CC                     ; CC->ecx
```

Контрольные значения 1

Имя	Значение
eax	0x00000002
ebx	0x0000000d
ecx	0x00000025

Контрольные значения 1 | Вывод