Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего профессионального образования
«Севастопольский государственный университет»

Исследование способов анализа и отладки приложений Win 32 в среде отладчике OllyDbg

Методические указания

к выполнению лабораторной работы

для студентов, обучающихся по направлению

09.03.02 “Информационные системы и технологии”

дневной и заочной формы обучения

Севастополь 2019

Исследование способов анализа и отладки приложений Win32 в среде отладчика OllyDbg. Методические указания к лабораторным занятиям по дисциплине "Технические средства информационных систем" / Сост. А.В. Волкова, В.С. Чернега – Севастополь: Изд-во СевГУ, 2019 – 18 с.

Методические указания предназначены для проведения лабораторных работ по дисциплине “Технические средства информационных систем“. Целью методических указаний является помощь студентом в изучении возможностей отладчика OllyDbg при разработке и отладке приложений Win32. Излагаются теоретические и практические сведения необходимые для выполнения лабораторной работы, программа исследований, требования к содержанию отчета.

Методические указания рассмотрены и утверждены на методическом семинаре и заседании кафедры информационных систем (протокол № 1 от 30 августа 2019 г.)

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

Лабораторная работа

Исследование способов анализа и отладки приложений Win32 в среде отладчика OllyDbg

1 Цель работы

Исследование архитектуры 32-разрядных процессоров и возможностей отладчика OllyDbg при разработке и отладке приложений Win32 для, приобретение практических навыков по анализу и отладке программ на языке ассемблера.

2 Основные теоретические положения

С точки зрения программиста 32-разрядные процессоры корпорации Intel представляют собой 16 программно доступных регистров для пользовательских программ и ещё 11 регистров для работы с мультимедийными приложениями (MMX) и числами с плавающей точкой (FPU/NPX). Все команды так или иначе изменяют содержимое регистров.

Следует отметить, что регистры могут быть неравнозначны и при использовании определенных инструкций могут иметь специальное значение:

- EAX - аккумулятор, операнд-источник или приемник результата;
- EBX - указатель на данные в сегменте DS;
- ECX - счетчик для цепочечных (например, MOVS) и циклических (с префиксом REP и т.п.) инструкций;
- EDX - адрес порта ввода-вывода для инструкций IN/INS, OUT/OUTS;
- ESI - указатель на операнд-источник в сегменте DS для цепочечных инструкций;
- EDI - указатель на операнд-приемник в сегменте ES для цепочечных инструкций;
- EBP - указатель на данные в сегменте SS.

32-разрядные микропроцессоры содержат также шесть непосредственно доступных 16-битных регистров селекторов сегментов. С каждым сегментным регистром ассоциирован программно-недоступный кэш дескриптора соответствующего сегмента, содержащий базовый адрес сегмента в линейном адресном пространстве, предел сегмента и атрибуты сегмента. Этот кэш заполняется при загрузке значения в сегментный регистр.

Не все сегментные регистры равнозначны. Регистр CS хранит селектор сегмента кода. Процессор извлекает очередную инструкцию для исполнения, формируя логический адрес из селектора в CS и смещения в регистре EIP. Значение этого регистра нельзя изменить непосредственно, оно меняется в командах межсегментного перехода (FAR JMP), межсегментного вызова (FAR CALL), при вызове обработчика прерывания (INT) и при возврате из дальней процедуры (RETF) или обработчика прерывания (IRET).

Регистр SS хранит селектор сегмента стека. Стек используется для передачи параметров подпрограммам и для сохранения адреса возврата при вызове подпрограммы или обработчика прерывания. Вершиной стека считается байт, логический адрес которого образуется из селектора в регистре SS и смещения в регистре ESP. Программа может непосредственно изменить значение SS, что дает ей возможность переключать-

ся между несколькими стеками. Причем на время выполнения команды MOV SS,xxxx и одной команды следующей за ней (обычно это MOV ESP,xxxx) запрещаются маскируемые и блокируются немаскируемые прерывания.

Регистры DS, ES, FS и GS хранят селекторы сегментов данных. Если инструкция обращается к памяти, но содержит только смещение, то считается, что она обращается к данным в сегменте DS. Сегмент ES может использоваться без явного указания в цепочечных командах. Сегменты FS и GS используются при обращении к памяти только при явном использовании в инструкции префиксов этих сегментов.

Указатель команд (EIP) является 32-разрядным регистром. Он содержит смещение следующей команды, подлежащей выполнению. Относительный адрес отсчитывается от начала сегмента исполняемой задачи. Указатель команд непосредственно недоступен программисту, но он управляется явно командами управления потоком, прерываниями и исключениями (JMP, CALL, RET, IRET, команды условного перехода). Получить текущее значение EIP можно, если выполнить команду CALL, а затем прочитать слово на вершине стека. Младшие 16 бит регистра EIP обозначаются IP и могут быть использованы процессором независимо при исполнении 16-битного кода.

Регистр системных флагов

Регистр EFLAGS содержит группу флагов состояния, управления и системных флагов. Младшие 16 бит регистра представляют собой 16-разрядный регистр флагов и состояния МП 8086, называемый FLAGS, который наиболее полезен при исполнении программ для МП 8086 и 80286. Структура регистра флагов показана на рисунке. Неопределенные биты зарезервированы, то есть на данный момент они не имеют значения, однако могут быть использованы для специальных целей в последующих версиях микропроцессора. Далее термин "установлен" означает значение 1, а термин "сброшен" - значение 0.

Некоторые из флагов могут быть изменены специально предназначенными для этой цели инструкциями. Для изменения или проверки группы флагов можно воспользоваться командами:

- LAHF/SAHF - загрузка/сохранение младших 8 битов регистра флагов в регистре AH;
- PUSHF/POPF - помещение/извлечение из стека младших 16 битов регистра флагов;
- PUSHFD/POPFD - помещение/извлечение из стека 32-битного регистра EFLAGS.

Флаги статуса CF, PF, AF, ZF, SF и OF отражают статус выполнения арифметических инструкций (таких как ADD, SUB, MUL, DIV).

- *CF - флаг переноса (Carry Flag).* Установлен, если операция привела к переносу из старшего бита при сложении или к займу для старшего бита при вычитании, иначе сброшен. Для 8-, 16-, 32-разрядных операций этот бит устанавливается при переносе из битов 7, 15 и 31 соответственно. Для беззнаковых операций флаг сигнализирует о переполнении. Значение этого флага может быть изменено непосредственно при помощи инструкций: CLC - сбросить CF в 0, STC - установить CF в 1, CMC - инвертировать CF. Также используется в операциях сдвига.
- *PF - флаг четности (Parity Flag).* Установлен, если младшие восемь бит операнда содержат четное число единиц (проверка на четность) иначе сброшен. На этот флаг влияют только младшие восемь бит независимо от длины операнда.

- *AF - флаг вспомогательного переноса (Adjust Flag).* Используется для упрощения сложения и вычитания упакованных двоично-десятичных чисел. Независимо от длины операнда (8, 16 или 32 бит) флаг AF установлен, если операция привела к займу из бита 3 при вычитании или переносу из бита 3 при сложении, иначе он сброшен.
- *ZF - флаг нуля (Zero Flag).* Установлен, если все биты результата равны нулю, иначе сброшен.
- *SF - флаг знака (Sign Flag).* Установлен, если установлен старший бит результата, иначе он сброшен. Для 8-, 16- и 32-разрядных операций этот флаг отражает состояние 7, 15 и 31 бита соответственно. Для знаковых чисел старший бит отражает знак числа: 0 - неотрицательное, 1 - отрицательное.
- *OF - флаг переполнения (Overflow Flag).* Флаг установлен, если операция привела к переносу (займу) в знаковый (самый старший) бит результата, но не привела к переносу (займу) из самого старшего бита, или наоборот. Для операций над числами со знаком сигнализирует о переполнении.

DF - флаг направления (Direction Flag) управляет поведением цепочечных инструкций (MOVS, CMPS, SCAS, LODS, STOS). Когда флаг сброшен, при выполнении цепочечной команды происходит автоинкремент адресов источника и приемника. Когда флаг установлен - автодекремент. Флаг можно непосредственно установить при помощи инструкции STD и сбросить при помощи CLD.

Системные флаги и поле IOPL влияют на процесс исполнения задачи, и поэтому не должны изменяться прикладной программой. Назначение основных флагов следующее:

- *TF - флаг ловушки (Trap Flag).* Установка флага TF переводит МП в пошаговый режим для отладки. Процессор автоматически генерирует исключение #1 после каждой команды, что позволяет проверить программу на исполнение каждой команды. Когда флаг TF сброшен, то ловушка по исключению #1 возникает в точках адресов останова, загружаемых в регистры отладки DR0-DR3.
- *IF - флаг разрешения прерываний (Interrupt enable Flag).* Установка флага IF позволяет МП воспринимать запросы внешних маскируемых прерываний. Очистка этого бита запрещает такие прерывания. Флаг не влияет на обработку, как немаскируемых внешних прерываний, так и исключений.
- *IOPL - уровень привилегий ввода-вывода (I/O Privilege Level field).* Это двухбитное поле используется в защищенном режиме. Биты IOPL показывают наивысшее значение текущего уровня привилегий (CPL), позволяющее выполнять команды ввода-вывода, не приводя к исключению #13 или обращению к битовой карте разрешения ввода-вывода. Это поле показывает также наивысшее значение CPL, которое позволяет изменять бит IF с помощью команд STI или CLI, а также при выборке нового значения из стека в регистр EFLAGS. Это поле может быть изменено инструкциями POPF или IRET только, если текущий уровень привилегий задачи равен 0.
- *NT - флаг вложенной задачи (Nested Task flag).* Если при переключении задач происходит вложение задач, то этот флаг устанавливается в 1. Совместно с полем "Связь TSS" в сегменте состояния задачи обеспечивает корректное вложение задач.
- *RF - флаг возобновления (Resume Flag).* Временно приостанавливает обработку исключений отладки (т.е. возвращает к нормальному исполнению программы)

так, что исполнение команды может быть повторено после обработки исключения для отладки, не вызывая немедленно обработку другого исключения для отладки.

- *VM - режим виртуального МП 8086 (Virtual-8086 Mode flag).* Бит обеспечивает для задачи функционирование в режиме виртуального МП 8086. Бит VM может быть установлен только двумя способами: при восстановлении флагов из стека по инструкции IRET на нулевом уровне привилегий и переключением на задачу, в TSS которой в образе EFLAGS бит VM установлен.
- *AC - флаг контроля выравнивания (Alignment Check flag).* Разрешает контроль выравнивания для текущей задачи. Контроль выравнивания производится, если CR0.AM=1 и EFLAGS.AC=1 и CR0.PE=1 и CPL=3. Контроль выравнивания требует, чтобы при обращениях к памяти двойное слово обязательно должно начинаться с адреса, кратного 4, а 16-битное слово - с адреса, кратного 2, иначе генерируется нарушение контроля выравнивания (исключение #17).
- *VIF - виртуальный флаг прерывания (Virtual Interrupt Flag).* Виртуальный образ флага IF, используется совместно с флагом VIP. Процессор распознает VIF, если CR4.VME=1 или CR4.PVI=1 (разрешено расширение виртуального режима) и IOPL<3.
- *VIP - виртуальный флаг задержки прерывания (Virtual Interrupt Pending Flag).* Системное ПО устанавливает этот флаг, если требуется отложить обработку прерывания. Используется совместно с VIF. Процессор читает этот флаг, но никогда не изменяет его. Флаг распознается, если CR4.VME=1 или CR4.PVI=1 (разрешено расширение виртуального режима) и IOPL<3.

Рассмотрим основные, наиболее часто используемые команды ассемблера.

Команды пересылки данных:

MOV dst, src – пересылка данных из src в dst.

PUSH src – помещает в вершину стека содержимое src.

POP dst – снимает слово с вершины стека и помещает его в dst.

Арифметические команды:

ADD dst, src – содержимое src складывается с содержимым dst, и результат помещается в src.

SUB dst, src – содержимое src вычитается из содержимого dst, и результат помещается в src.

INC dst – прибавляет 1 к содержимому dst.

DEC dst – вычитает 1 из содержимого dst.

MUL src – одноадресная команда, указывается только множитель, множимое берется из AL, а результат помещается в EAX.

DIV src – одноадресная команда, указывается только делитель, делимое берется из EAX, результат помещается также в EAX.

Команда безусловной передачи управления:

JMP opr – команда безусловной передачи управления.

NOP (No OPeration) - это инструкция, которая при выполнении не производит никаких изменений в регистрах, стеке или памяти. Поэтому ее можно использовать, например, если нужно заменить одну инструкцию на другую, более короткую. Чтобы процессор не столкнулся с ошибками, лишнее место, возникшее при замене, заполняется NOP'ами.

Эта команда Также служит для полного уничтожения другой инструкции. Для этого нужно заменить ее соответствующим количество NOP'ов, или попросту заNOPать.

3. Описание лабораторной установки

Лабораторная установка состоит из персонального компьютера, на котором установлена программа OllyDbg. OllyDbg – это бесплатный 32-битный низкоуровневый отладчик с интуитивно понятным интерфейсом, рассчитанный под операционные системы семейства Windows и, выполняющий анализ и модификации откомпилированных exe- и dll-файлов. Отладчик удобен тем, что он дизассемблирует программу и позволяет вести анализ программы, когда ее исходники недоступны. Отладчик отображает значения регистров, может показывать содержимое памяти, распознает процедуры, API-функции, переходы, строковые и цифровые константы, имеется возможность переименовывать переменные и делать комментарии в дизассемблированном коде.

Скачанный архив OllyDbg нужно распаковать в легко доступную папку на жестком диске. После того, как файл был распакован, необходимо зайти в созданную папку и найти исполняемый файл OLLYDBG.exe, который и нужно запустить. Для удобства на рабочем столе создан ярлык отладчика.

При запуске отладчика появляется рабочее окно, в котором в дальнейшем можно во всех подробностях изучать принцип работы исследуемой программы. Исследуемый процесс может быть выбран как в виде исполняемого файла (File → open, либо F3), так и в виде уже запущенного процесса (File → attach). В последнем случае следует при завершении работы с отлаживаемым процессом, после закрытия OllyDbg, остановится и сам процесс, так что необходимо быть аккуратными при отладке файлов типа Csrss.EXE, Winlogon.EXE или Explorer.EXE.

Внешний вид отладчика OllyDbg 2.01 с запущенным под ним процессом представлен на рисунке 3.1, где:

1 – окно (фрейм, по терминологии разработчика) с дизассемблированным листингом отлаживаемой программы;

2 – окно (фрейм) дампа памяти (расположение переменных в памяти);

3 – окно (фрейм) регистров процессора (CPU);

4 – окно (фрейм) стека.

В окне отлаживаемого кода (1) с кодом программы рекомендуется включить режим подсказки для условных и безусловных переходов: (Options → Debugging Options → CPU – активировать опции Show jump path, Show grayed path if jump is not taken, Show jumps to selected command). Эти опции необходимы для удобства и не являются обязательными. Просто при попадании, к примеру, на команду типа JNE, JS, JZ и т.п., в зависимости от установленных флагов отладчик подсветит направление выполняющегося перехода красным либо не выполняющегося серым цветом.

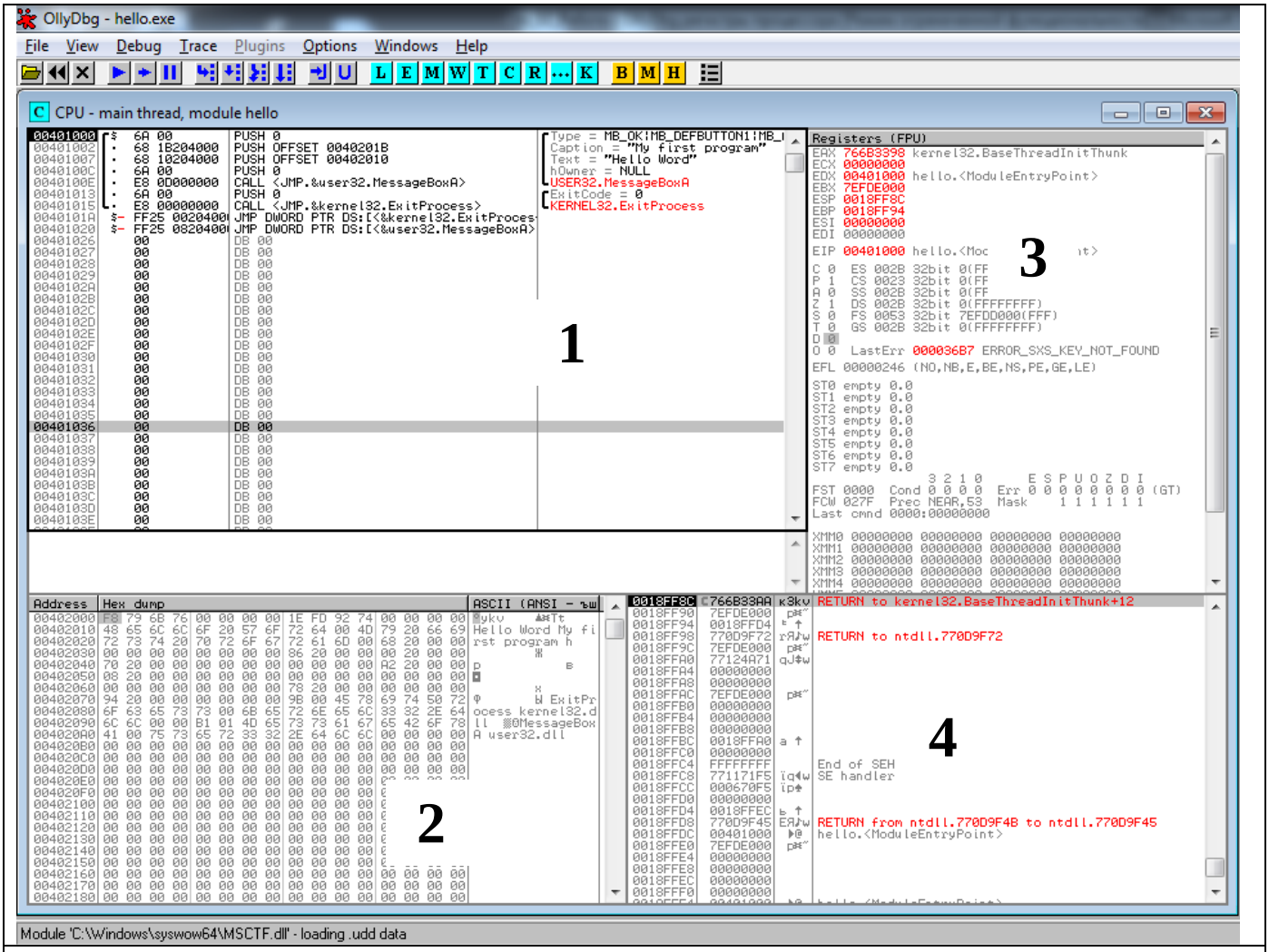


Рисунок 3.1 – Внешний вид отладчика OllyDbg 2.01 с запущенным процессом

На рисунке 3.2 выделено 4 области фрейма отслеживания кода, представляющих из себя инструменты информирования и интерактивности, которые предоставляет OllyDbg, где:

1 – виртуальные адреса;

2 – машинный код;

3 – дизассемблированный листинг (команды ассемблера);

4 – комментарии отладчика.

Трассировка программы выполняется следующими клавишами:

F7 – с заходом в процедуры CALL address;

F8 – без захода в процедуры.

Также возможно выполнение программы (клавиша F9) полностью, либо до установленной точки останова, либо автоматическое выполнение кода между двумя точками останова.

Точки останова (breakpoint), можно установить следующими способами: ручной режим (выбрать строчку программы, где нужна точка останова, и нажать F2)

либо через CommandBar (но только при наличии установленного плагина). Установленная точка останова подсвечивается в колонке адресации.

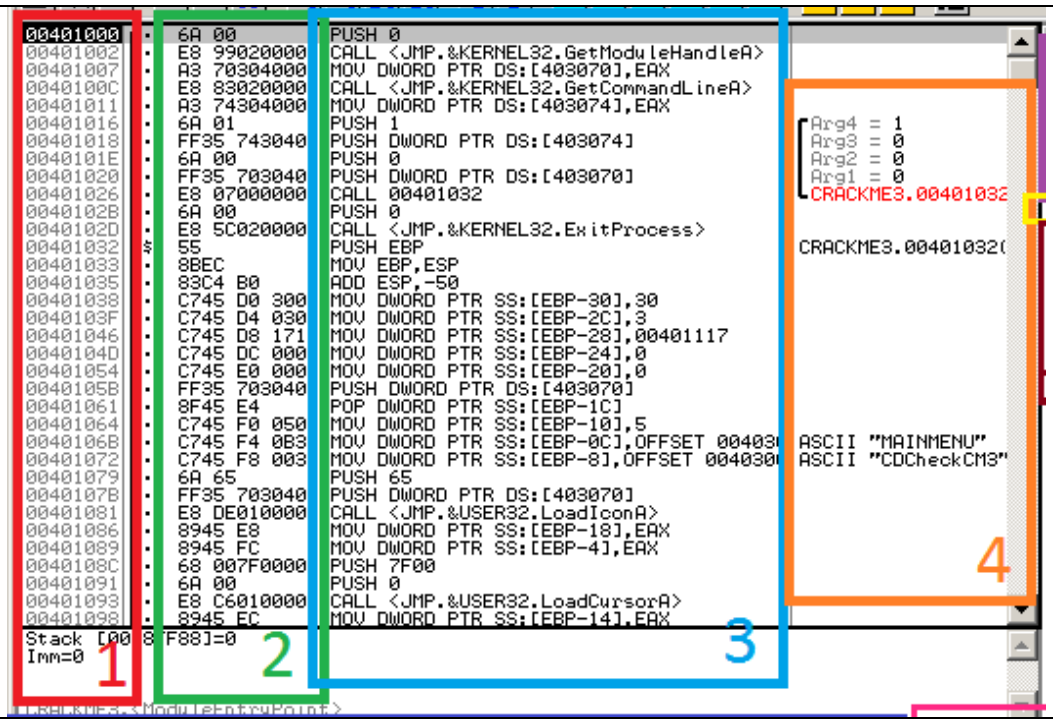


Рисунок 3.2 – Области информирования и интерактивности Окна (фрейма) отслеживания кода

Двойной клик в окне по команде (либо же правый клик на команде – Assemble, либо пробел) вызывает окно редактирования кода, что иногда помогает отключить некоторые вызовы функций, изменить направление ветвления и т.п. При замене, допустим команды CALL длиной 5 байт командой NOP (Not OPeration - нет операции), длина которой 1 байт, OllyDbg автоматически заполнит лишние четыре байта таким же NOP-ом. Следует только учитывать, что после перезагрузки программы в отладчике (Debug → Restart) все изменения сбрасываются.

Перемещение по коду программы можно осуществляется через правый клик мышки (на самом деле, практически всю работу можно свести к правому клику). Для этого просто надо выполнить правый клик мышки → Go to → expression, и в появившемся окне ввести адрес, на который необходимо посмотреть (следует учесть, что это не меняет содержимое регистра EIP, т.е. не является трассировкой). Вернуться на прежнюю позицию кода можно либо нажав правый клик мышки → Go to → Previous, либо нажав «минус» на клавишах NumPad. Значение EIP также не меняется. Эти команды являются инструментом для обзора кода.

Бывает, что произошла остановка на некотором вызове типа CALL, но сложно определить, необходимо ли его трассировать по F7. В данном случае можно применить такую функцию, как «подсмотреть» код внутри вызываемой процедуры. Сделать это можно либо через правый клик мышки → Folow, либо через клавишу Enter. Значение EIP как и в предыдущем случае, не изменяется. Вернуться из просмотра можно той же клавишей «минус» на клавиатуре NumPad.

В случае, когда нет необходимости выполнять весь код отлаживаемой программы, а только какую-то его отдельную функцию, можно использовать такое свойство, как установка адреса выполняемой программы (правый клик мышки → New origin here) на нужной команде. Это позволяет оттрассировать отдельную процедуру.

Если модифицируется код программы (assemble) и нужно сохранить его в exe-файле необходимо воспользоваться такой опцией, как сохранение изменения (правый клик мышки → Edit → Copy to Executable → Selection или правый клик мышки → Copy to Executable → All Modifications).

Внешний вид отладчика можно настроить под вкус пользователя опцией правый клик мышки → Appearance.

2.1 Фрейм дампа памяти (2)

В данном фрейме также возможны установки точек останова на чтение-запись-выполнение (через правый клик мышки). Через контекстное меню можно задавать любой требуемый для выполнения текущей задачи вид дампа памяти. Дамп может быть представлен в виде HEX-кодировки, в виде ASCII и UNICODE строк, в виде Short, Long, Float чисел или же может быть показан PE-заголовок (для этого достаточно перейти на адрес базы EXE-файла и выбрать правый клик мышки → Special → PE headers).

Формат исполняемых файлов PE представляет собой структуру данных, содержащую всю информацию, необходимую PE-загрузчику для проецирования (отображения) файла в память. Исполняемый код включает в себя ссылки для связывания динамически загружаемых библиотек, таблицы экспорта и импорта API функций, данные для управления ресурсами и данные локальной памяти потока (TLS). В операционных системах семейства Windows NT формат PE используется для EXE, DLL, SYS (драйверов устройств), и других типов исполняемых файлов.

Существует опция редактирования дампа (правый клик мышки → Edit → Binary Edit). Для этого надо выделить мышкой необходимый фрагмент данных и редактировать его.

2.2 Фрейм регистров процессора (3)

В данном фрейме представлены все регистры процессора. В стандартном виде тут находятся основные регистры, флаговый регистр, регистры сопроцессора (FPU). Через контекстное меню можно переключаться на представление вместо FPU-регистров MMX, либо 3DNow!-регистров (правый клик мыши → view MMX registers/view 3DNow! Registers/view debug registers соответственно). Значения основных регистров (кроме EIP) могут быть изменены на любом этапе отладки. Для этого достаточно просто щелкнуть на нужном регистре правый клик мыши, выбрать опцию Modify и в появившемся окне ввести необходимое значение либо же просто двойной клик на выбранном регистре. Данная опция может быть полезна при оперативной подмене результатов, возвращаемых, к примеру, API-функциями. Так же через контекстное меню есть просто такие простые команды, как, к примеру, Increment (увеличить значение на 1), Decrement (уменьшить значение на 1), Zero (обнулить), Set to 1 (установить в 1). Опции Follow in Dump и Follow in Stack перемещают соот-

ветственно в фрейме дампа либо в фрейме стека на адрес, содержащийся в конкретном регистре.

Для регистра флагов процессора так же имеется полезная функция, иногда применяемая при отладке. Это т.н. установка (Set) и сброс (Reset) значений отдельных флагов, установка – в единицу, сброс – в ноль. Выполняется либо через контекстное меню, либо двойным кликом на выбранном флаге. Крайне полезная опция, когда необходимо изменить направление ветвления, не модифицируя команды с JE на JNE.

Значения регистров и флагов, которые в ходе выполнения последней машинной команды были модифицированы, подсвечиваются красным цветом.

На рисунке 1.3 выделены 3 области фрейма регистров процессора, представляющих из себя инструменты информирования и интерактивности, которые предоставляет OllyDbg, где:

5 – регистры общего назначения;

6 – EIP регистр (показывает виртуальный адрес следующей выполняемой команды);

7 – флаги и регистр флагов.

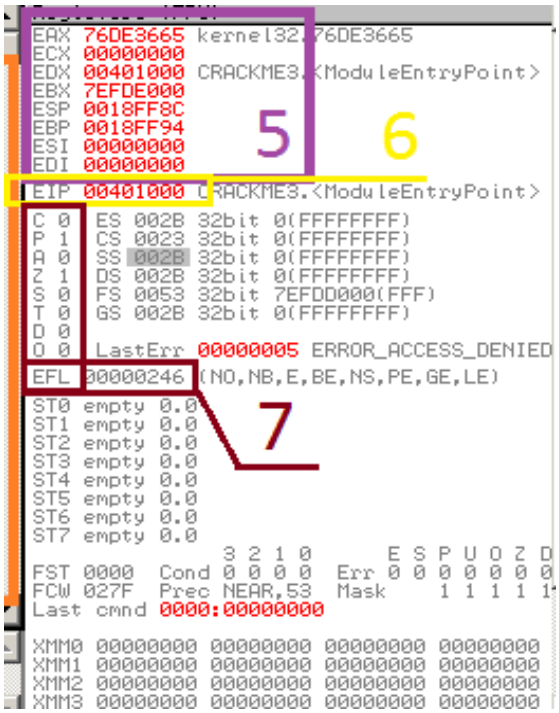


Рисунок 1.3 – Области информирования и интерактивности фрейма регистров процессора

2.3 Фрейм стека (4)

В фрейм стека есть как аналогичные по предыдущим фреймам опции, так и новые. Все рассматриваемые опции находятся в контекстном меню. Например, опция Addressing → Relative to Selections/Relative to EBP/Relative to ESP определяет адресацию в фрейме стека (относительно выбранного адреса/относительно EBP/относительно ESP).

Lock Stack – позволяет зафиксировать выбранный адрес в стеке (сделать его неперемещаемым за границы фрейма, т.е. автоскролл при изменении вершины стека не работает). Unlock Stack – обратная операция.

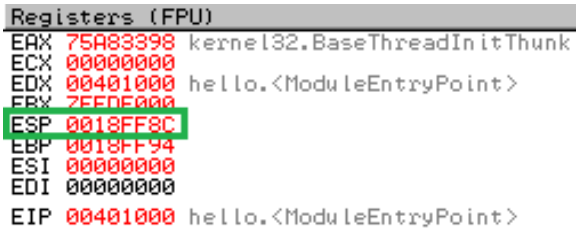
Операции Push DWORD / Pop DWORD позволяют затолкнуть в стек и соответственно вытолкнуть из него двойное слово. При вызове первой инструкции появляется окно, где предлагают ввести заталкиваемое значение. Операции Go to ESP / Go to EBP позволяют переместиться на вершину стека (которую указывает ESP) и на дно кадра стека (указывает EBP).

Во всех фреймах доступна возможность копирования данных в буфер обмена. Делается этого нужно выделить необходимый блок данных (кода) и активировать опцию Copy → To clipboard.

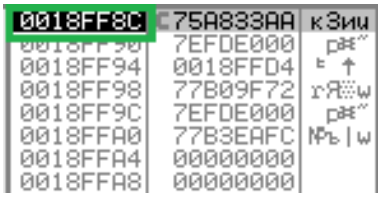
2.4 Особенности работы в OllyDbg с регистрами процессора

Процессору нужны помощники для выполнения программ, которыми являются регистры: когда необходимо выполнить какую-либо инструкцию, например, которая складывает содержимое двух ячеек памяти, процессору нужно разместить содержимое одной из них в регистре, а затем сложить то, что в нем находится с содержимым другой ячейки. Это один из примеров использования регистров.

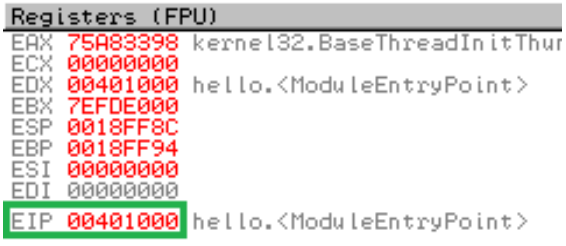
ESP указывает на самое верхнее значение стека:



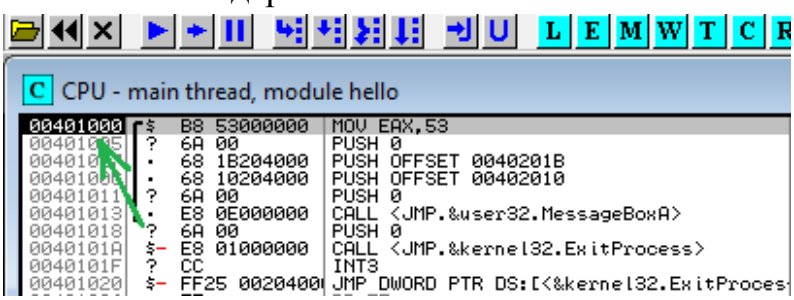
ESP равняется **12FFC4**, и если посмотреть в OllyDbg на стек, то видно, что регистр указывает на верхнее значение стека.



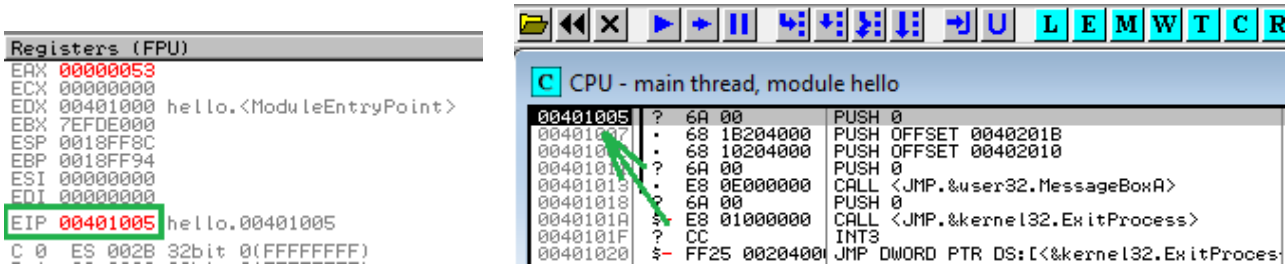
EIP –регистр, который указывает на инструкцию, выполняющуюся в данный момент.



При загрузке exe-файла адресом первой выполняемой инструкции является **401000** и именно это значение содержится в **EIP**.



Если нажать на **F7**, то выполнится первая инструкцию и наступит очередь второй. Теперь **EIP** равен **401005**, и в листинге видно, что первая инструкция выполнилась и находимся сейчас на второй.



Другие регистры могут содержать различные значения и служат для помощи процессору в выполнении инструкций.

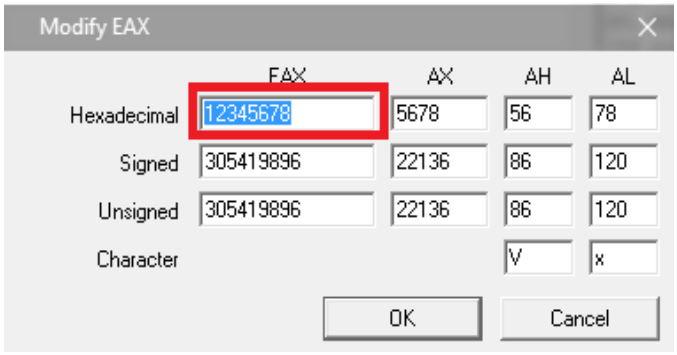
В верхнем правом фрейме (3) на рисунке 1.1 находятся все существующие 32-х битные регистры **EAX**, **ECX**, **EDX**, **EBX**, **ESP**, **EBP**, **ESI**, **EDI** и **EIP**.

OllyDbg отображает их содержимое в шестнадцатеричной форме. Например, минимальное значение **EAX** может быть равно 00000000, а максимальное – FFFFFFFF, что в двоичной системе счисления будет 11111111111111111111111111111111, т.е. 32 бита, каждый из которых может быть равен 0 или 1, поэтому регистры и называются 32-х битными.

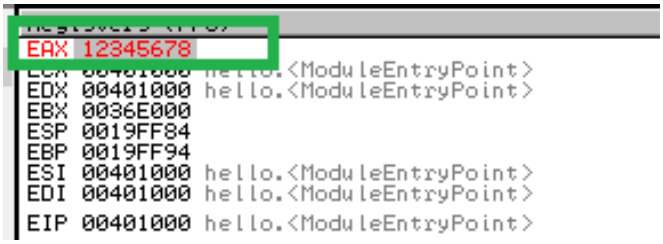
В ассемблере можно обращаться к частям этих 32-х битных регистров.

2.4.1 Пример работы с регистрами.

Необходимо изменить значение **EAX** на нужное, в данном случае на 12345678. Для этого необходимо открыть OllyDbg, где загружен exe-файл. В открывшемся окне в поле «Hexdecimal» написать 12345678 и активировать опцию OK.



Теперь в регистре появится нужное значение. OllyDbg выделяет измененные значения красным цветом.



Можно использовать часть **EAX**, в данном случае **AX** будет 16-ти битным регистром, например, в примере выше в нем будет находиться число 5678.

Таким же образом **EBX** делится на **BX**, **BH** и **BL**, подобное деление существует и почти для всех остальных регистров.

Все, что было проделано с регистром **EAX** применимо и к другим регистрам: необходимо отметить регистр, значение которого надо изменить, затем кликнуть по правой кнопке мыши и активировать опцию Modify. Единственное исключение – это регистр **EIP**, указывающий на инструкцию, которая должна выполнится.

Чтобы изменить его, необходимо сделать следующее: поскольку **EIP** всегда указывает на инструкцию, которая должна выполниться, то надо выбрать новую инструкцию в листинге (в фрейме 1 на рисунке 1.1), а затем нажать правую кнопку мыши → New origin here, и значение **EIP** изменится на соответствующий адрес выбранной строки кода, и таким образом, программа продолжит выполнение именно с этого места.

2.5 Работа с флагами

В OllyDbg под регистрами находятся флаги.

```
EIP 0040102E hello.0040102E
C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7EFDD000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr 000036B7 ERROR_SXS_KEY_NOT_FOUND
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
```

В OllyDbg флаги бывают **C P A Z S T D** и **O**, где:

O – признак переполнения;
D – признак направления;
T – признак трассировки;
S – признак знака: 1 - число < 0, 0 - число > 0;
Z – признак нуля: 1 - число = 0;
A – признак переноса из тетрады;
P – признак четности;
C – признак переноса.

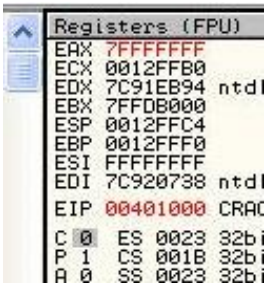
Они могут иметь только два значения: ноль или один. Определенные инструкции при выполнении могут изменять их значение.

Рассмотрим некоторые из них.

2.5.1 Флаг O или флаг переполнения

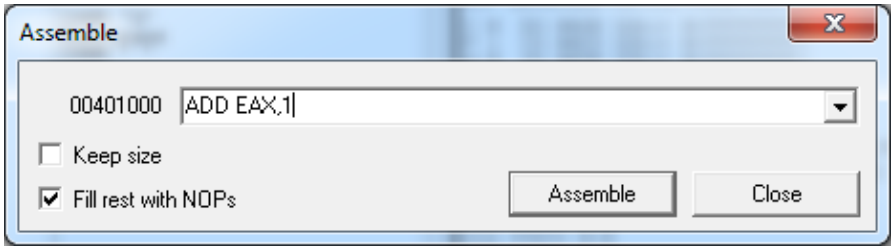
Этот флаг активируется, когда в результате операции изменяется знак и возвращается неправильное значение. Рассмотрим следующий пример в OllyDbg.

Необходимо изменить значение **EAX** на 7FFFFFFF, которое является максимальным положительным.

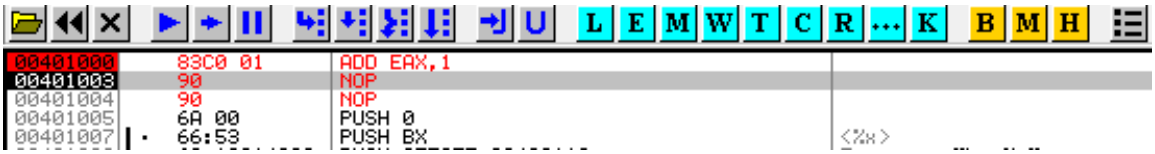


Затем надо прибавить 1, из-за чего сумма превысит максимальное позитивное число, потому что 80000000 соответствует числу отрицательному.

Для этого необходимо активировать окно **Assemble**, где можно ввести данную инструкцию и написать команду **ADD EAX, 1**.

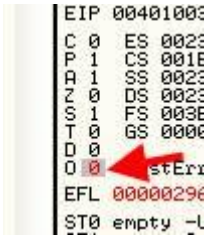


После активации кнопки **Assemble** инструкция по адресу 401000 изменится на ту, которая была введена.



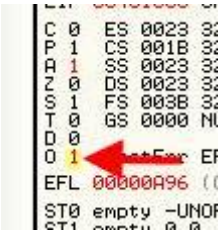
ADD EAX, 1 прибавляет к **EAX** число 1 и сохраняет в этом же регистре полученное значение.

До выполнения этой строки с помощью **F7** флаг **O** равен нулю.



Если выполнить данную строку с помощью **F7**, то в **EAX** окажется значение 80000000 и знак числа будет изменен.

Флаг **O** станет активным и будет показывать 1, то есть, что результат выполненной инструкции превзошел максимально возможный результат. Назначения этого флага – показывать, когда это происходит.

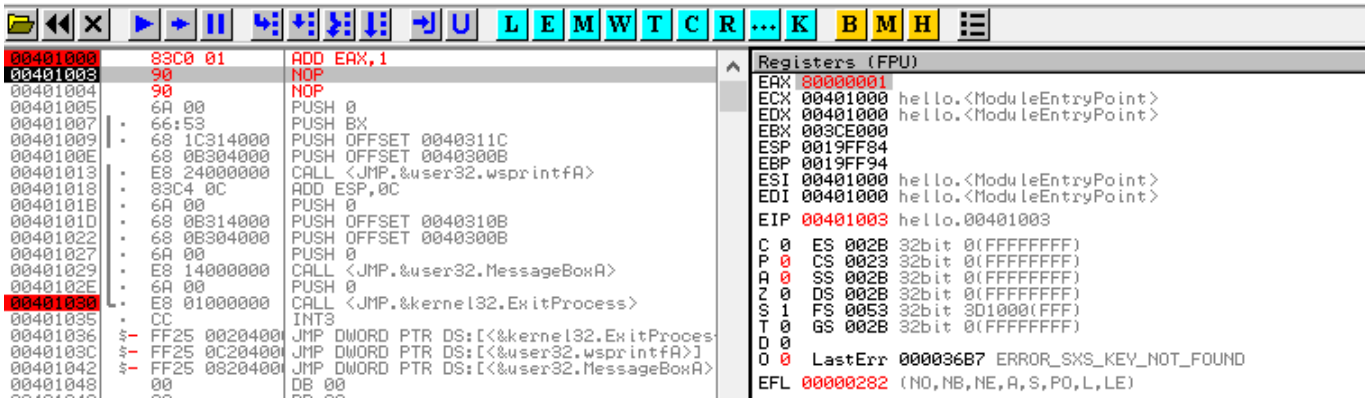


2.5.2 Флаг P или флаг четности

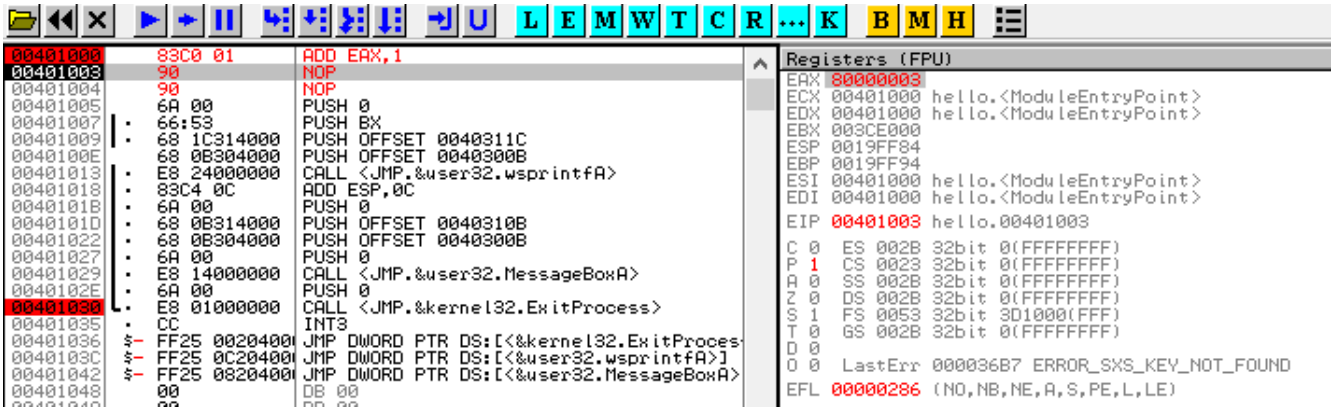
Данный флаг активируется, когда результат выполнения инструкция в двоичной форме содержит четное количество единиц, например, 1010, 1100 или 1111000.

Чтобы изучить принцип его работы, можно воспользоваться уже заданной в OllyDbg инструкцией **ADD EAX, 1**, и для повторного выполнения этой строки, необходимо выделить строку 401000, кликнуть по ней правой кнопкой мыши и выбрать опцию **New orgin here**, чтобы вернуться назад, и теперь, при нажатии **F7**, выполнится инструкция **ADD EAX, 1**.

Сейчас в **EAX** содержится значения 00000000, а флаг **P** равен 1 (как результат выполнения предыдущей инструкции), а если прибавить к **EAX** 1, флаг **P** станет равен 0, так как результат, содержащийся в **EAX**, равен в двоичной форме 1, что в двоичной форме содержит всего лишь одну единицу, то есть нечетное количество.



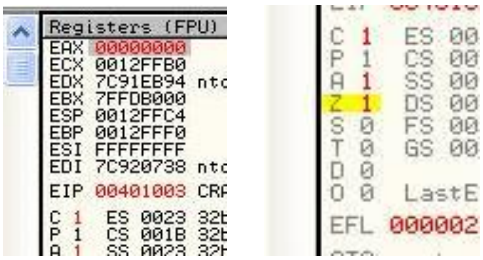
При повторном выполнении команды **ADD EAX, 1**, в **EAX**, содержавшем прежде значение 1, будет находиться значение 2, что в двоичной форме равно 10. Число единиц опять нечетно и флаг **P** неактивен. Если повторить процесс с прибавлением единицы еще раз, в **EAX** уже будет содержаться значение 3, что в двоичной форме равно 11, и сейчас результат содержит четное количество единиц, а флаг четности станет активен.



2.5.3 Флаг Z или флаг нуля

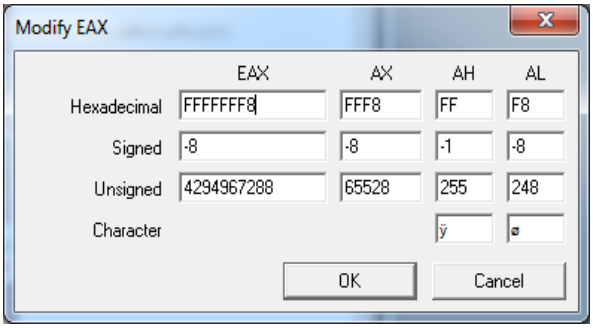
Этот флаг становится активным, когда результатом выполнения инструкции является ноль.

Если вернуться к инструкции **ADD EAX, 1** по адресу 401000, сделать значение **EAX** равным FFFFFFFF, что равно десятиричному -1, и нажать **F7** и запустить инструкцию **ADD EAX, 1**, результатом -1 + 1 будет ноль и флаг **Z** станет активным. После нажатия **F7 EAX** будет содержать ноль, а поскольку результат – ноль, то флаг **Z** активен и равен 1.



2.5.4 Флаг S или флаг знака

Этот флаг равен 1, когда результат операции отрицателен. Чтобы посмотреть его в действии, необходимо изменить значение **EAX** на FFFFFFF8, что равно –8 в десятичной форме.



С помощью «**New origin here**» и кнопки **F7** можно снова выполнить инструкцию **ADD EAX, 1**. Результатом будет FFFFFFF9, то есть -7 в десятичной форме, которое является отрицательным числом, поэтому флаг знака станет активным.

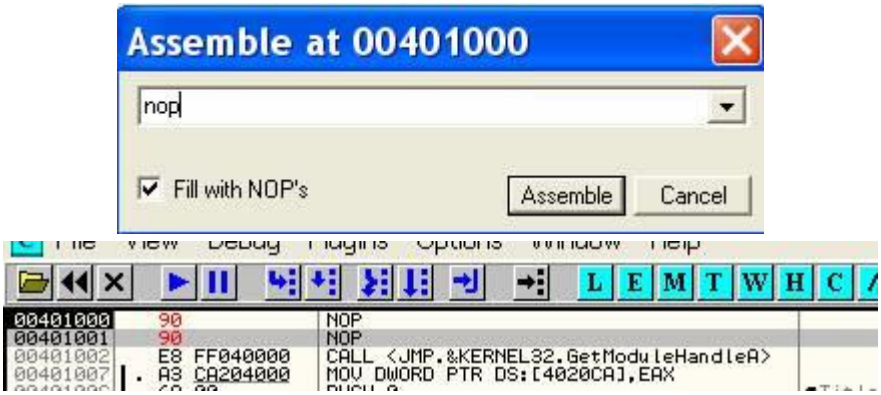
После нажатия **F7** и выполнения инструкции активируется флаг **S**, который теперь равен 1, то есть, отрицательный результат его активирует.

2.5.5 Флаг C или флаг переноса

Активен, когда превышается максимальное значение, которое может содержать регистр, например, если поместить в **EAX** значение FFFFFFFF и прибавить 1, то флаг переноса станет равен 1.

Например, в представленном ниже примере необходимо занопать первую инструкцию **PUSH 0**, которая имеет размер 2 байта, для этого надо отметить соответствующую строку листинга, а затем активировать клавишу «пробел» или же в контекстном меню выбрать команду **Assemble**.

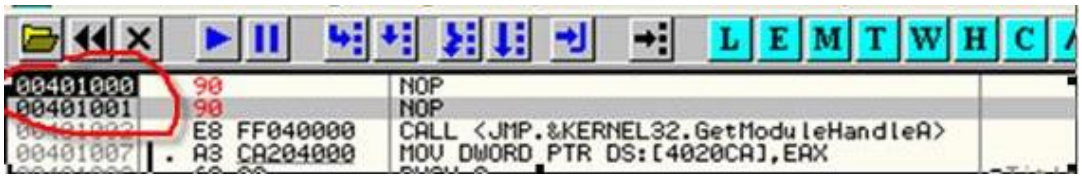
В появившемся окне необходимо написать **NOP** и активировать опцию **Assemble**.



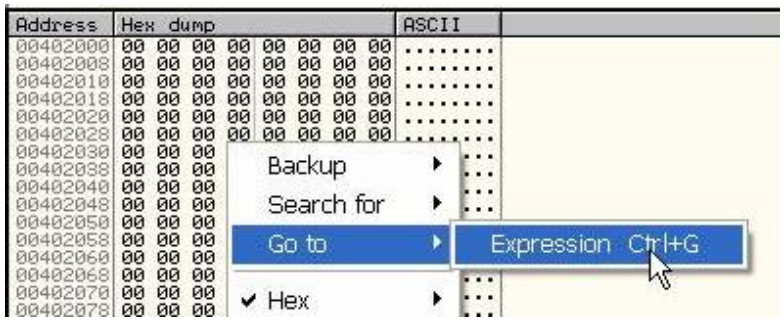
OllyDbg, кроме того, что добавил **NOP**, учел, что **PUSH** занимает два байта, и чтобы не оставлять неопределенным следующий байт, ввел дополнительный **NOP**.

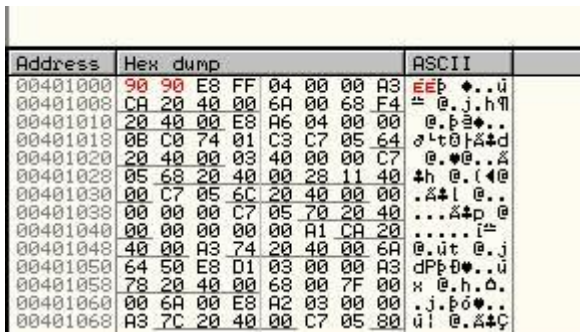
Теперь на месте, где был **PUSH 0**, находятся два NOP'а, которые при исполнении ничего не делают. Изменился только регистр **EIP**, содержащий адрес инструкции, которая должна выполниться следующей, но ничего больше не изменило своего значения: ни другие регистры, ни стек, ни флаги, ни память.

Теперь нужно посмотреть эти два новых байта в **DUMP**, и для того, чтобы найти их там, нужен их адрес в памяти, и это 401000 и 401001.



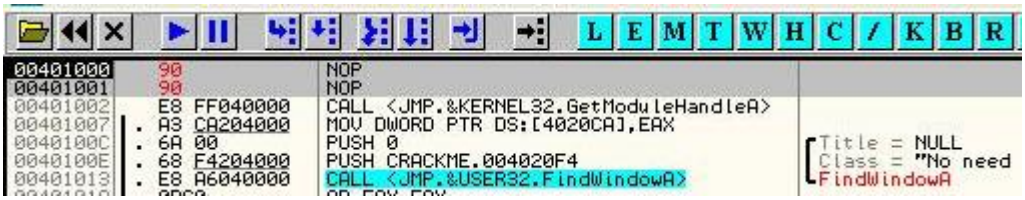
В окне **DUMP**, правой кнопкой мыши необходимо активировать опцию **Go to → Expression**, после чего нужно ввести адрес, где располагаются требуемые байты.

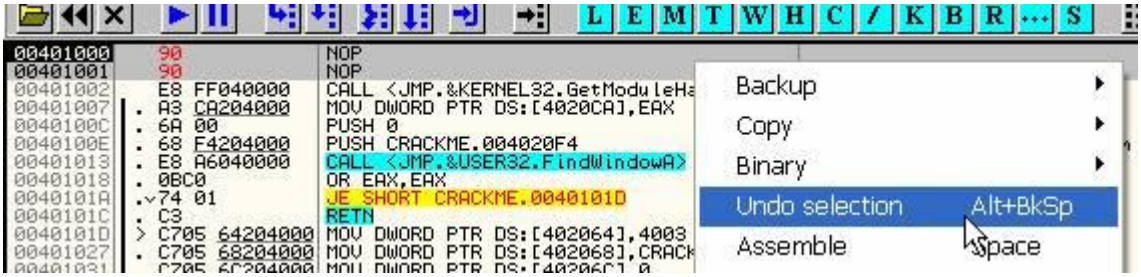




Красным выделены измененные OllyDbg байты. Первым идут два 90, а затем E8, FF и все остальные байты, относящиеся к следующей за NOP'ами инструкцией, которой является **CALL**.

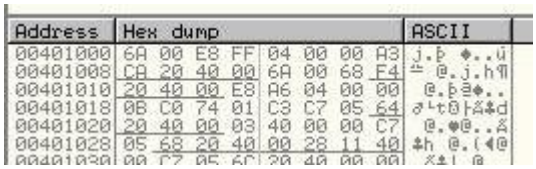
OllyDbg может вернуть обратно измененные байты. Для этого в любом из двух окон – **DUMP** или листинге, необходимо отметить оба байта и кликнув на правую кнопку мыши выбрать опцию **Edit → Undo selection**.





После это снова появляется изначальная команда **PUSH**.

В **DUMP** также находятся изначальные байты.



3 Инструкции стека

3.1 PUSH

Инструкция **PUSH** –инструкция для помещения значения в стек. Первая инструкция в представленном ниже примере – это **PUSH**.

В данном случае это **PUSH 0**, и, будучи выполненной, данная инструкция поместит 0 на самый верх стека, а то, что было до этого наверху, окажется под этим значением.

Так как стек каждый раз может располагаться по разному адресу, при запуске программы начальное содержимое стека может быть различным. По нажатию **F7** на самый верх попадет ноль, а все остальное окажется внизу него. По нажатию **F7**, ноль словно присоединился сверху того, что мы уже видели. Внизу по адресу 12ffc4 по-прежнему находится значение 7c816d4f, и ни одного из других значений в стеке также не изменилось.

Главным отличием является то, что самое верхнее значение стека находится теперь по адресу 12ffc0 – в вершине стека (именно там и находится ноль, который был помещен с помощью инструкции **PUSH**).

А регистр **ESP**, в котором находится адрес самого верхнего значения стека, теперь содержит 12FFc0.

Инструкция **PUSH** имеет несколько вариантов, позволяющих помещать в стек не только числа. **PUSH EAX** помещает значение **EAX** на верх стека. Подобным образом можно поместить в стек значение любого регистра, числа и т.п.

Также можно поместить значение, находящееся в памяти по определенному адресу:

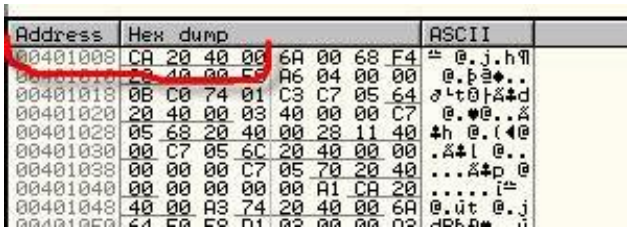
```
PUSH [401008]
```

Эта команда будет интерпретироваться отличным образом от команды:

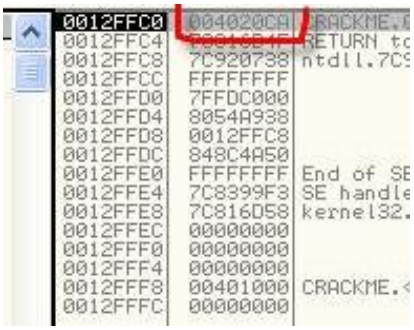
```
PUSH 401008
```

Если выполнить **PUSH 401008**, то в стек будет помещено число 401008.

Если заменить на **PUSH [401008]**. В этих четырех байтах находятся CA 20 40 00.



Запустив **PUSH** с помощью **F7**, в стеке появится значение, которое находится в памяти по этому адресу, но байты перевернуты, то есть они были помещены в обратном порядке. Это одно из свойств процессора: при чтении или записи содержимого из/в память байты всегда переворачиваются.



Таким образом, без квадратных скобок задаваемый параметр является просто числом, а с ними оно является адресом ячейки памяти, где содержится помещаемое в стек значение.

3.2 POP

Инструкция **POP** является обратной по отношению к **PUSH**: она достает первое значение из стека и помещает его в указанное параметром инструкции место назначения. Например, **POP EAX** берет первое значение из стека и помещает его в **EAX**, после чего то значение, которое шло следующим, становится верхним.

4 Программа выполнения работы

4.1. Повторить теоретический материал, касающийся структуры 32-разрядных микропроцессоров, программно доступных регистров и системы команд языка ассемблера (выполняется при домашней подготовке).

4.2. Создать приложение типа Win32 с графическим интерфейсом, вычисляющее сумму и разность двух чисел: первое число – номер в группе, второе – число, противоположное номеру первой буквы фамилии в алфавите (отрицательное число в дополнительном коде). Противоположным к некоторому *x* называется число, равное *x* по абсолютной величине, но обратное по знаку.

4.3.Загрузить разработанное приложение в отладчик.

4.4.Исследовать работу программы в отладчике OllyDbg.

4.5. Проанализировать, какие изменения происходят в различных регистрах процессора:

– изменяйте значения регистров EAX и EBX таким образом, чтобы произошли изменения состояний флагов O, P, Z, S, C;

– поместите значение в стек с помощью инструкции PUSH;

– извлеките значение из стека с помощью инструкции POP;

– изучите принцип работы инструкции MOV с 8-ми, 16-ти и 32-х разрядными регистрами;

– изучите работу математических инструкций: INC, DEC, ADD, SUB, MUL, IMUL, DIV, NEG;

– измените значение регистра ESP;

– измените значение регистра EIP;

– найдите инструкцию PUSH 0 и занопайте ее.

3. Результаты исследований отразите в отчете к данной работе.

5 Содержание отчета

1. Цель работы.
2. Приложение типа Win32 – скриншот.
3. Результаты исследования программы в отладчике OllyDbg.
4. Выводы.

6 Контрольные вопросы

1. Что такое регистры процессора и для чего они предназначаются?
2. Какие регистры микропроцессорной памяти используются для адресации данных, команд программы, стековой памяти?
3. Что такое флаги? Дайте характеристику основным флагам.
4. Приведите примеры ассемблерных команд пересылки данных из регистра в регистр и из памяти/в память в регистр.
5. Приведите примеры команд выполнения арифметических операций.
6. Почему в 32-разрядном процессоре имеются 16- и 80-разрядные регистры?
7. С какой целью в ассемблерных командах используется модификатор PTR? Приведите примеры его использования.
8. Каково назначение отладчика программ? Назовите основные его возможности.
9. С какой целью в компьютерных программах используется стек? Приведите примеры использования команд работы со стеком.
10. Как изменить значение регистров EAX, EBX, ESP и EIP в отладчике OllyDbg.
11. Каково назначение инструкции NOP. Когда она используется?