МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ОДЕССКИЙ НАЦИОНАЛЬНЫЙ МОРСКОЙ УНИВЕРСИТЕТ
Кафедра «Информационные технологии»

Бойко Виктор Дмитриевич

ОПЕРАЦИОННЫЕ СИСТЕМЫ

Учебное пособие для изучения курса и выполнения контрольных заданий
Компьютерные науки

Лабораторная работа 5. Debug, часть 1

Одесса — 2016

Содержание

1 Аннотация

Краткая аннотация: данная работа начинает цикл работ, посвященнный знакомству с отладчиком debug, который позволяет:

- набирать небольшие фрагменты ассемблерного кода;
- выполнять трассировку фрагментов кода;
- отслеживать состояние регистров процессора и ячеек оперативной памяти в процессе трассировки;
- тестировать, корректировать, отлаживать набранные в отладчике или загруженные в отладчик фрагменты кода;
- сохранять на диске (в виде исполняемых файлов com-типа) и загружать в отладчик фрагменты кода.

Студент должен освоить макрокоманды отладчика debug, научиться набирать, тестировать, трассировать небольшие фрагменты кода (в пределах 10-20 ассемблерных команд).

Приобретенный в ходе выполнения работы опыт должен позволить в последующем ориентироваться в листингах ассемблерных модулей.

2 Цель работы

Первоначальное знакомство с макрокомандами отладчика, работа с регистрами и ОЗУ эмулируемой среды.

3 Лабораторное задание

Проработать ключевые положения, дублируя приводимые в тексте действия и занося в протокол результаты их выполнения. Самостоятельно выполнить задания, приведенные в конце работы. Ответить на вопросы для самоконтроля, выполнить задания для самостоятельного исследования.

4 Содержание протокола лабораторной работы

1. Титульная страница.
2. Результаты пошагового выполнения команд и комментарии.
3. Результаты выполнения самостоятельного задания.
4. Выводы.

Отчеты выполненные не по форме к рассмотрению не допускаются!
Результаты работы сдаются преподавателю в электронной форме.

5 Ключевые положения

5.1 Краткие сведения об отладчике debug

> Основанная на базовых принципах, которым обучают всех падаванов (чтобы могли защититься от выстрелов), сорсу была на редкость проста и настолько ограничена и ориентирована на оборону, что считалась пассивной
>
> *(c) Мэтью Стовер*

Отладчик debug.exe, входит в любую версию DOS/Windows. debug.exe прост, доступен и подходит для начинающих программистов на языке Ассемблер.

16-разрядная утилита операционной системы является консольным приложением и предназначена для создания или изменения кода файлов. С ее помощью можно создавать простые приложения под MS-DOS и отслеживать их работу. Данный отладчик находится на самом низком уровне компиляторов assembler, но при этом обладает широкими возможностями такими как просмотр, изменение памяти и получение состояния регистров.

Название программы происходит от Bugs – насекомые. На программистском слэнге Bugs означает «ошибки в программе».

5.2 Запуск debug

Отладчик debug находится в системной папке system32 папки WINDOWS (файл debug.exe).

Запуск отладчика удобнее всего выполнять из меню `Пуск/Выполнить`. Для чего следует запустить работу в сеансе MS DOS (`Пуск -> командная строка -> cmd`).

Другим вариантом является работа из виртуального окружения (что актуально в связи с прогрессом современных операционных систем), например DOSBox, dosemu, qemu и так далее.

Обратите внимание: в данной работе debug запускается из виртуального окружения! Работа в debug в Windows может привести к неконтролируемым последствиям. Работая в «живой системе» не забывайте делать бэкапы!

Для запуска из вертуального окружения следует перейти в папку программы (например `C:\DEBUG125`) и набрать название программы:

```
C:\>cd debug125

C:\DEBUG125>debug
```

После этого включится основной режим работы отладчика debug (характерная особенность – присутствие чёрточки и мигающего справа от неё курсора). Основной режим работы устанавливается автоматически после загрузки отладчика.

В этом режиме можно набирать любые макрокоманды отладчика от A до W. Выход из этого режима означает завершение работы отладчика.

Все значения команд задаются в *шестнадцатиричной системе счисления*, при этом отметки типа hex или h не ставятся. Однобайтовые значения задаются двузначными 16-ми числами, двухбайтовые значения задаются четырёхзначными 16-ми числами.

Например:

```
80   (1 байт)  - byte
1A80 (2 байта) - word
```

При необходимости обращения к *ячейкам памяти ОЗУ* адреса так же задаются в шестнадцатиричной системе счисления, при этом адрес состоит из двух частей - адреса сегмента данный (берется в регистре сегмента данных) и адреса собственно ячейки памяти:

```
072A:0100
```

Все указанные команды вводятся в ответ на подсказку отладчика - минус (-). Список команд отладчика можно получить по команде `?`. Все команды (кроме Q) воспринимают параметры, разделяемые запятыми (шестнадцатиричные значения) или пробелами, например:

```
d cs:110 120
```

Адреса задаются сегментным регистром (по умолчанию CS для A, G, L, T, U и W и DS для других команд) или его адресом из 4 цифр и смещением. Точка с запятой между сегментом и смещением обязательна.

debug не различает регистр букв.

Пробелы в командах используется только для разделения параметров.

Сегмент и смещение записываются с использованием двоеточия, в формате сегмент:смещение, например, CS:3C1 (смещение 3C1h в сегменте кода) или 40:17 (смещение 17h в сегменте, адрес начала которого - 40[0]h).

5.3 Базовые команды debug

Двумя базовыми структурами данных в архитектуре фон Неймана являются микропроцессор (и его реистры) и оперативная память.

Рассмотрим основы работы с этими структурами в системе DEBUG.

Обратите внимание: конкретные числовые данные могут отличатся от приведенных ниже - поскольку DEBUG на различных компьютерах и в различных средах будет запущен с разными исходными данными и в разном окружении.

5.3.1 Работа с регистрами микропроцессора

Макрокоманда `r` позволяет просматривать и изменять содержимое регистров микропроцессора.

Если она задана без параметров - она возвращает содержимое всех регистров микропроцессора.

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0826 ES=0826 SS=0826 CS=0826 IP=0100 NV UP EI NG NZ NA PO NC
0826:0100 C3               RET
```

В данном случае:

```
AX=0000 BX=0000 CX=0000 DX=0000  - регистры общего назначения

SP=FFFE BP=0000 SI=0000 DI=0000  - индексные регистры

DS=0826 ES=0826 SS=0826 CS=0826  - регистры сегментов памяти процессора

IP=0100                          - счетчик выполнения программ

NV UP EI NG NZ NA PO NC          - содержимое регистра флагов
```

Назначение каждого из регистров будет подробно рассмотрено на следующих занятиях.

Пока обратите внимание на то, как сочетание регистров `CS` (code segment) и `IP` (указатель команды) задает содержимое третьей строки:

```
 0826:0100 C3                RET

+----+----+
| CS | IP |
+----+----+
```

В этой строке приводится команда ассемблера, которая будет выполнена на следующем шаге при пошаговом выполнении - при этом указывается адрес команды, соответствующий ей машинный код и мнемоника.

```
 0826:0100            C3                RET

 адрес команды     машинный код        мнемоника
```

С помощью команды `r` можно не только просматривать, но менять содержимое регистров процессора. Для этого `r` задается в формате `r <название регистра>` - и далее вписывается необходимое значение. Изменим содержимое регистров `AX` и `DS`.

Рассмотрим исходное состояние регистров:

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0826 ES=0826 SS=0826 CS=0100 IP=0100 NV UP EI NG NZ NA PO NC
0100:0100 0000              ADD     [BX+SI],AL                      DS:0000=CD
```

Изменим содержимое `AX` на 1234:

```
-r AX
AX 0000  :1234
```

Убедимся, что изменения были внесены в регистр:

```
-r
AX=1234 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0826 ES=0826 SS=0826 CS=0100 IP=0100 NV UP EI NG NZ NA PO NC
0100:0100 0000              ADD     [BX+SI],AL                      DS:0000=CD
```

Аналогично:

```
-r DS
DS 0826  :0100

-r
AX=1234 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0100 ES=0826 SS=0826 CS=0100 IP=0100 NV UP EI NG NZ NA PO NC
0100:0100 0000              ADD     [BX+SI],AL                      DS:0000=00
```

Изменения можно вносить и непосредственно «с ходу», сразу вводя нужное значение регистра данных:

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0826 ES=0826 SS=0826 CS=0826 IP=0100 NV UP EI NG NZ NA PO NC
0826:0100 C3                RET

-r CS 0200

-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0826 ES=0826 SS=0826 CS=0200 IP=0100 NV UP EI NG NZ NA PO NC
0200:0100 2F                DAS
```

Директива `r f` позволяет работать непосредственно с регистром флагов.

```
-r f
NV UP EI NG NZ NA PO NC   :
```

Регистр флагов и его назначение будут рассмотренны на следующих занятиях, здесь мы просто приведем их значения:

Флаг	Установлен	Снят
Переполнение	ov	nv
Направление	dn (уменьшение)	up (увеличение)
Прерывание	ei (включено)	di (выключено)
Знак	ng (отрицательный)	pl (положительный)
Ноль	zr	nz

```
+--------------------+------------------+----------------+
| Добавочный перенос |      ac          |      na        |
+--------------------+------------------+----------------+
|     Четность       |      pe          |      po        |
|                    |     (чет)        |    (нечет)     |
+--------------------+------------------+----------------+
|     Перенос        |      cy          |      nc        |
+--------------------+------------------+----------------+
```

5.3.2 Просмотр содержимого ячеек оперативной памяти

Макрокоманда `d` отладчика `debug` выводит на экран содержимое ячеек заданного диапазона.

Результат Выполнения команды `d` называется «дампом» (от английского dump - «свалка, груда хлама; мусорная куча» - вывод команды на первый взгляд действительно похож на свалку) и выглядит следующим образом:

```
-d
0826:0200  24 03 00 75 0C 30 C0 E9-DF 02 A3 24 03 F7 1E 24 $..u.0.....$...$
0826:0210  03 B0 FF E9 D3 02 53 0F-A8 E8 91 01 89 C3 8E EA ......S.........
0826:0220  E8 28 FE 65 89 47 21 65-89 57 23 0F A9 5B C3 53 .(.e.G!e.W#..[.S
0826:0230  51 E8 79 01 89 C3 8E C2-26 8B 47 21 26 8B 57 23 Q.y.....&.G!&.W#
0826:0240  B9 07 00 D1 EA D1 D8 E2-FA 26 89 47 0C 26 8A 47 .........&.G.&.G
0826:0250  21 24 7F 26 88 47 20 59-5B C3 53 56 57 0F A8 C8 !$.&.G Y[.SVW...
0826:0260  06 00 00 89 C6 89 D3 89-4E FE E8 C2 FF 89 F0 89 ........N.......
0826:0270  DA E8 39 01 89 46 FC 8E-EA 89 C7 65 8B 45 21 89 ..9..F.....e.E!.
```

Повторное выполнение команды `d` - обратите внимание на смену адресации ячеек!

```
-d
0826:0280  46 FA FF 76 0C 89 CF 8B-0D 89 F0 89 DA E8 E3 FD F..v............
0826:0290  88 C1 8B 7E FC 65 8B 45-21 2B 46 FA 8B 7E FE 89 ...~.e.E!+F..~..
0826:02A0  05 89 F0 89 DA E8 87 FF-88 C8 E9 DA FE 53 56 57 .............SVW
0826:02B0  0F A8 C8 04 00 00 89 C7-89 D6 E8 72 FF 89 F8 89 ...........r....
0826:02C0  F2 E8 E9 00 89 C3 8E EA-65 8B 47 0C 89 46 FC 65 ........e.G..F.e
0826:02D0  8A 47 20 88 46 FE 51 B9-01 00 89 F8 89 F2 E8 92 .G .F.Q.........
0826:02E0  FD 8B 4E FC 65 89 4F 0C-8A 66 FE 65 88 67 20 E9 ..N.e.O..f.e.g .
0826:02F0  B4 00 53 56 57 0F A8 C8-06 00 00 89 C6 89 56 FC ..SVW.........V.
```

Видно, что начальный адрес автоматически увеличился на 80 (80=10x8 - система 16-ричная, то есть выведено 8 рядов по 16 значений в каждом ряду).

В приведенном выше примере:

```
содержимое начальной ячейки                    содержимое конечной ячейки

          ||                                              ||
          ||                                              ||
```

```
          \/                                        \/

0826:0280  46 FA FF 76 0C 89 CF 8B-0D 89 F0 89 DA E8 E3 FD F..v............

     ^^^^
     ||||

     адрес
   начальной
    ячейки
```

Таким образом адрес начальной ячейки `0826:0280`, ее содержимое в шестнадцатиричной форме `46`, а символ соответствующий этому коду - `F`

Нумерация остальных ячеек выглядит следующим образом (из-за недостатка места двойка в адресах 281, 282, 283 и т.д. приписана сверху):

```
             2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
          280 81 82 83 84 85 86 87-88 89 8A 8B 8C 8D 8E 8F

0826:0280  46 FA FF 76 0C 89 CF 8B-0D 89 F0 89 DA E8 E3 FD F..v............
```

В колонке адресов ячеек оперативной памяти используется следующий формат адресов: `XXXX:YYYY`

`XXXX` – сегментная часть логического адреса (эта часть адреса в наших примерах не меняется, поэтому в последующем мы на неё внимания не обращаем).

`YYYY` – смещение логического адреса (адрес смещения), именно на эту часть логического адреса мы в последующем постоянно будем обращать внимание.

В приведённом выше примере `XXXX = 0826` (сегментная часть логического адреса), `YYYY = 0280, 0281, 0283, 0284` и так далее

В команде `d` можно задать начальный адрес диапазона - тогда будет показан дамп памяти начиная с указанного адреса:

```
-d 110
0100:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 E9 8C ................
0100:0150  55 FC 8C C8 8E D8 8E D0-8B 26 74 54 C7 06 76 54 U........&tT..vT
0100:0160  0A 01 1E 07 E8 92 0F 75-03 E8 00 00 80 00 43 C2 .......u......C.
0100:0170  CC 06 53 C2 F1 06 53 C2-FE FF 8E 02 00 01 8E 02 ..S...S.........
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..

-d 2000
0100:2000  51 55 A8 10 74 14 A8 04-75 10 A8 08 74 E9 8B 36 QU..t...u...t..6
0100:2010  3F 55 80 3C FF 74 03 E9-16 FB A0 3D 55 3C 00 74 ?U.<.t.....=U<.t
0100:2020  21 3C F0 75 09 F6 06 51-55 01 74 EB EB 14 A1 41 !<.u...QU.t....A
0100:2030  55 24 FE 3D FF 00 77 DF-BF 61 37 B9 07 00 F2 AE U$.=..w..a7.....
```

```
0100:2040  75 D5 F6 06 3C 55 10 74-12 A1 41 55 3D FF 00 77 u...<U.t..AU=..w
0100:2050  C6 BF 58 37 B9 09 00 F2-AE 75 BC 8B 1E 49 55 0B ..X7.....u...IU.
0100:2060  DB 74 0C A0 50 55 F6 D8-D0 E0 84 47 07 75 A8 8B .t..PU.....G.u..
0100:2070  16 B2 54 8B 1E B6 54 F6-06 3C 55 02 74 05 B0 9B ..T...T..<U.t...
```

При необходимости задать диапазон адресов (просмотреть участок памяти определенной длины), в команде `d` указываются два параметра подряд.

Диапазоны адресов можно задать указывая начальный адрес и число ячеек (показать 20 ячеек, начиная с адреса 150) или указывая начальный и конечный адреса (показать ячейки начиная с адреса 150 и заканчивая адресом 17F).

```
-d 150
0100:0150  55 FC 8C C8 8E D8 8E D0-8B 26 74 54 C7 06 76 54 U........&tT..vT
0100:0160  0A 01 1E 07 E8 92 0F 75-03 E8 00 00 80 00 43 C2 .......u......C.
0100:0170  CC 06 53 C2 F1 06 53 C2-FE FF 8E 02 00 01 8E 02 ..S...S.........
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..
0100:0190  20 2F 65 3A 31 30 32 34-20 2F 70 0D 0A 00 00 00  /e:1024 /p.....
0100:01A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:01B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:01C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 82 44 ...............D
-d 150 L20
0100:0150  55 FC 8C C8 8E D8 8E D0-8B 26 74 54 C7 06 76 54 U........&tT..vT
0100:0160  0A 01 1E 07 E8 92 0F 75-03 E8 00 00 80 00 43 C2 .......u......C.
-d 150 17F
0100:0150  55 FC 8C C8 8E D8 8E D0-8B 26 74 54 C7 06 76 54 U........&tT..vT
0100:0160  0A 01 1E 07 E8 92 0F 75-03 E8 00 00 80 00 43 C2 .......u......C.
0100:0170  CC 06 53 C2 F1 06 53 C2-FE FF 8E 02 00 01 8E 02 ..S...S.........
-d 150,17F
0100:0150  55 FC 8C C8 8E D8 8E D0-8B 26 74 54 C7 06 76 54 U........&tT..vT
0100:0160  0A 01 1E 07 E8 92 0F 75-03 E8 00 00 80 00 43 C2 .......u......C.
0100:0170  CC 06 53 C2 F1 06 53 C2-FE FF 8E 02 00 01 8E 02 ..S...S.........
```

В приведенном примере:

`-d 150 L20` - показать 20 ячеек, начиная с адреса 150

`-d 150 17F` - показать ячейки начиная с адреса 150 и заканчивая адресом 17F

`-d 150,17F` - так же показать ячейки начиная с адреса 150 и заканчивая адресом 17F, однако диапазон адресов указан не через пробел, а через запятую.

В качестве адреса может использоваться один из сегментных регистров - например `CS`

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0100 ES=0100 SS=0100 CS=0100 IP=0100 NV UP EI NG NZ NA PO NC
0100:0100 1111              ADC     [BX+DI],DX                      DS:0000=0000

-r CS 1234
```

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0100 ES=0100 SS=0100 CS=1234 IP=0100 NV UP EI NG NZ NA PO NC
1234:0100 89C8              MOV     AX,CX

-d CS:100 L30
1234:0100  89 C8 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
1234:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
1234:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

Обратите внимание на следующие моменты: регистр `CS` использован как адрес смещения в команде `d` (CS:0100). Кроме того - в последней команде `r` видна мнемоника команды `MOV AX,CX` и ее машинный код (`89 C8`) - и этот же код виден в соответствующих ячейках.

5.3.3 Редактирование содержимого ячеек оперативной памяти

Для ввода значений в ячейки оперативной памяти используется макрокоманда `e` (`edit`). С ее помощью данные вводятся в ячейки оперативной памяти:

Возвращение из режима ввода в основной режим работы отладчика происходит по нажатию клавиши Enter. Продолжение работы в режиме ввода данных без ввода данных – нажатие клавиши пробел. Нажимая клавишу пробел, можно попасть на любую ячейку памяти и ввести в неё нужное число. Ввод значений завершается нажатием клавиши Enter.

Пример. Сначала рассмотрим содержимое памяти с помощью уже знакомой команды `d`:

```
-d 170
0100:0170  CC 06 53 C2 F1 06 53 C2-FE FF 8E 02 00 01 8E 02 ..S...S.........
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..
0100:0190  20 2F 65 3A 31 30 32 34-20 2F 70 0D 0A 00 00 00  /e:1024 /p.....
0100:01A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:01B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:01C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 82 44 ...............D
0100:01D0  3F 3F 3F 3F 3F 3F 3F 3F-3F 3F 31 11 00 00 00 00 ??????????1.....
0100:01E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

Теперь заполним несколько ячеек памяти своими данными (выход происходит по Enter, переход к следующей ячейке осуществляется по пробелу):

```
-e 170
0100:0170  CC.11   06.22   53.33   C2.44   F1.55   06.
-
```

Проверим, что изменилось:

```
-d 170
0100:0170  11 22 33 44 55 06 53 C2-FE FF 8E 02 00 01 8E 02 .”3DU.S.........
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..
0100:0190  20 2F 65 3A 31 30 32 34-20 2F 70 0D 0A 00 00 00  /e:1024 /p.....
0100:01A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:01B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:01C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 82 44 ...............D
0100:01D0  3F 3F 3F 3F 3F 3F 3F 3F-3F 3F 31 11 00 00 00 00 ??????????1.....
0100:01E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

Обратите внимание - в первой строке:

```
0100:0170  11 22 33 44 55 06 53 C2-FE FF 8E 02 00 01 8E 02 .”3DU.S.........
```

Значения, которые мы ввели с помощью команды `e`.

Как и другие макрокоманды `e` можно использовать с дополнительными параметрами. Самостоятельно разберитесь в следующих примерах (обратите внимание на аргумент `L20` в команде `d` для сокращения размера дампа):

Пример 1

```
-d 170 L20
0100:0170  11 22 33 44 55 06 53 C2-FE FF 8E 02 00 01 8E 02 .”3DU.S.........
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..

-e 170 10 20 30 40 50 60 70 80 90 A0

-d 170 L20
0100:0170  10 20 30 40 50 60 70 80-90 A0 8E 02 00 01 8E 02 . 0@P‘p.........
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..
```

Пример 2

```
-d 170 L20
0100:0170  10 20 30 40 50 60 70 80-90 A0 8E 02 00 01 8E 02 . 0@P‘p.........
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..

-e 17F 24

-d 170 L20
0100:0170  10 20 30 40 50 60 70 80-90 A0 8E 02 00 01 8E 24 . 0@P‘p........$
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..
```

Пример 3

```
-e 177
0100:0177  80.11 90.22   A0.33   8E.34   02.35   00.

-d 170 L20
0100:0170  10 20 30 40 50 60 70 11-22 33 34 35 00 01 8E 24 . 0@P‘p.”345...$
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..
```

Пример 4
Вопрос: в чем разница между примерами 1 и 4?

```
-d 170 L20
0100:0170  10 20 30 40 50 60 70 11-22 33 34 35 00 01 8E 24 . 0@P‘p.”345...$
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..

-e 170 ”01234567890”

-d 170 L20
0100:0170  30 31 32 33 34 35 36 37-38 39 30 35 00 01 8E 24 012345678905...$
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..
```

Пример 5
Обратите внимание на заполнение области памяти **кодами** латинских букв:

```
-d 170 L20
0100:0170  30 31 32 33 34 35 36 37-38 39 30 35 00 01 8E 24 012345678905...$
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..

-e 177 ”ABCDEF”

-d 170 L20
0100:0170  30 31 32 33 34 35 36 41-42 43 44 45 46 01 8E 24 0123456ABCDEF..$
0100:0180  7A 3A 5C 63 6F 6D 6D 61-6E 64 2E 63 6F 6D 00 0B z:\command.com..
```

5.4 Прочие команды работы с памятью

Следующие команды используются не так часто, как директивы `d` и `r`, однако, они так же могут быть полезны в некоторых случаях исследования и работы debug.

5.4.1 Заполнение памяти

Макрокоманда `f` (`fill`) позволяет заполнять ячейки памяти элементами задаваемого списка значений, при этом указывается диапазон ячеек (так же, как он указывался в команде `d`) - и код, которым следует заполнить указанный диапазон. Самостоятельно разберитесь в примерах ниже:
Пример 1

```
-d 100 L40
0100:0100  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

-f 100 L25 11
```

```
-d 100 L40
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0120  11 11 11 11 11 00 00 00-00 00 00 00 00 00 00 00 ................
0100:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

Пример 2

```
-d 100 L40
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0120  11 11 11 11 11 00 00 00-00 00 00 00 00 00 00 00 ................
0100:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

-f 110 120 22

-d 100 L40
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
0100:0120  22 11 11 11 11 00 00 00-00 00 00 00 00 00 00 00 "...............
0100:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

Пример 3

```
-d 100 L40
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
0100:0120  22 11 11 11 11 00 00 00-00 00 00 00 00 00 00 00 "...............
0100:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

-f 120,12F 33

-d 100 L40
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
0100:0120  33 33 33 33 33 33 33 33-33 33 33 33 33 33 33 33 3333333333333333
0100:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

Сравните команды из приведенных примеров с адресацией команды `d` и запишите свои выводы в протокол работы.

5.4.2 Поиск данных

Макрокоманда `s` (Search) выполняет поиск символов из списка; по умолчанию в качестве регистра смещения используется `DS`. Так же как в `d` и `f` можно указывать как длину, так и диапазон.

Формат команды следующий:

```
s <начальный_адрес> L<длина> ’<данные>’
s <начальный_адрес> <конечный_адрес> ’<данные>’
```

Примеры использования.

Пример 1

```
-d 100:100 L50
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
0100:0120  33 33 33 33 33 33 33 33-33 33 33 33 33 33 33 33 3333333333333333
0100:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0100:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 E9 8C ................

-e 100:135
0100:0135  00.12   00.13   00.14   00.

-d 100:100 L50
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
0100:0120  33 33 33 33 33 33 33 33-33 33 33 33 33 33 33 33 3333333333333333
0100:0130  00 00 00 00 00 12 13 14-00 00 00 00 00 00 00 00 ................
0100:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 E9 8C ................

-s 100:100 L50 13
0100:0136

-s 100 L50 12
0100:0135

-s 100 150 14
0100:0137
```

Пример 2

```
-e 100:135
0100:0135  12.ff   13.ff   14.ff  00.ff   00.ff   00.ff   00.

-d 100 L50
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
0100:0120  33 33 33 33 33 33 33 33-33 33 33 33 33 33 33 33 3333333333333333
0100:0130  00 00 00 00 00 FF FF FF-FF FF FF 00 00 00 00 00 ................
0100:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 E9 8C ................

-s 100 150 FF
0100:0135
0100:0136
```

```
0100:0137
0100:0138
0100:0139
0100:013A
-
```

Сравните команды из приведенных примеров с адресацией команды `d` и запишите свои выводы в протокол работы.

5.4.3 Сравнение двух областей памяти

Макрокоманда `c` (compare) сравненивает содержимое двух областей памяти; в качестве адресного регистра по умолчанию используется DS.

В команде указывается либо длина участков, либо диапазон адресов.

```
c <начальный_адрес_1> L<длина> <начальный_адрес_2>
c <начальный_адрес_1> <конечный_адрес_1> <начальный_адрес_2>
```

Пример использования команды:

```
-d 100
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
0100:0120  33 33 33 33 33 33 33 33-33 33 33 33 33 33 33 33 3333333333333333
0100:0130  00 00 00 00 00 FF FF FF-FF FF FF 00 00 00 00 00 ................
0100:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 E9 8C ................
0100:0150  55 FC 8C C8 8E D8 8E D0-8B 26 74 54 C7 06 76 54 U........&tT..vT
0100:0160  0A 01 1E 07 E8 92 0F 75-03 E8 00 00 80 00 43 C2 .......u......C.
0100:0170  30 31 32 33 34 35 36 41-42 43 44 45 46 01 8E 24 0123456ABCDEF..$

-c 100 L10 110
0100:0100  11  22  0100:0110
0100:0101  11  22  0100:0111
0100:0102  11  22  0100:0112
0100:0103  11  22  0100:0113
0100:0104  11  22  0100:0114
0100:0105  11  22  0100:0115
0100:0106  11  22  0100:0116
0100:0107  11  22  0100:0117
0100:0108  11  22  0100:0118
0100:0109  11  22  0100:0119
0100:010A  11  22  0100:011A
0100:010B  11  22  0100:011B
0100:010C  11  22  0100:011C
0100:010D  11  22  0100:011D
0100:010E  11  22  0100:011E
0100:010F  11  22  0100:011F
```

```
-c 100 L5 105

-c 100 L10 101
0100:010F  11  22  0100:0110

-c 110 115 118

-c 110 115 11B
0100:0115  22  33  0100:0120
```

Разберитесь самостоятельно в работе команды (что выводится и что не выводится при сравнении двух областей памяти) и запишите вывод в отчет.

5.4.4 Копирование содержимого памяти

Макрокоманда m (Move) выполняет копирование содержимого ячеек памяти; по умолчанию в качестве адресного регистра используется DS.

Можно указывать как длину, так и диапазон. Формат использования:

```
m <начальный_адрес> L<длина> <адрес_назначения>
m <начальный_адрес> <конечный_адрес> <адрес_назначения>
```

Самостоятельно разберитесь в следующем примере использования команды:

```
-d 100
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
0100:0120  33 33 33 33 33 33 33 33-33 33 33 33 33 33 33 33 3333333333333333
0100:0130  00 00 00 00 00 FF FF FF-FF FF FF 00 00 00 00 00 ................
0100:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 E9 8C ................
0100:0150  55 FC 8C C8 8E D8 8E D0-8B 26 74 54 C7 06 76 54 U........&tT..vT
0100:0160  0A 01 1E 07 E8 92 0F 75-03 E8 00 00 80 00 43 C2 .......u......C.
0100:0170  30 31 32 33 34 35 36 41-42 43 44 45 46 01 8E 24 0123456ABCDEF..$

-m 100 L7 140

-d 100
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
0100:0120  33 33 33 33 33 33 33 33-33 33 33 33 33 33 33 33 3333333333333333
0100:0130  00 00 00 00 00 FF FF FF-FF FF FF 00 00 00 00 00 ................
0100:0140  11 11 11 11 11 11 11 00-00 00 00 00 00 00 E9 8C ................
0100:0150  55 FC 8C C8 8E D8 8E D0-8B 26 74 54 C7 06 76 54 U........&tT..vT
0100:0160  0A 01 1E 07 E8 92 0F 75-03 E8 00 00 80 00 43 C2 .......u......C.
0100:0170  30 31 32 33 34 35 36 41-42 43 44 45 46 01 8E 24 0123456ABCDEF..$
```

```
-m 110 11F 170

-d 100
0100:0100  11 11 11 11 11 11 11 11-11 11 11 11 11 11 11 11 ................
0100:0110  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
0100:0120  33 33 33 33 33 33 33 33-33 33 33 33 33 33 33 33 3333333333333333
0100:0130  00 00 00 00 00 FF FF FF-FF FF FF 00 00 00 00 00 ................
0100:0140  11 11 11 11 11 11 11 00-00 00 00 00 00 00 E9 8C ................
0100:0150  55 FC 8C C8 8E D8 8E D0-8B 26 74 54 C7 06 76 54 U........&tT..vT
0100:0160  0A 01 1E 07 E8 92 0F 75-03 E8 00 00 80 00 43 C2 .......u......C.
0100:0170  22 22 22 22 22 22 22 22-22 22 22 22 22 22 22 22 """"""""""""""""
```

5.5 Шестнадцатиричный калькулятор h

Макрокоманда `h` (Hexadecimal) выполняет вычисление суммы и разности двух шестнадцатеричных величин и выводит результат. Используется в форме:

```
h <величина_1> <величина_2>
```

Результат выводится в формате:

```
<сумма> <разность>
```

Примеры

```
-h 12 12
0024  0000
-h 12 ff
0111  FF13
-h 1 1
0002  0000
-h 1 2
0003  FFFF
-h aa 11
00BB  0099
-h ffff 1111
00011110  EEEE
-h ffffe 111
0010010F  000FFEED
-h 123456 111111
00234567  00012345
```

6 Задания для самостоятельного выполнения

Выберите число N согласно своему варианту (N = 0 .. 9).

- Средствами debug вычислить число H как младший байт суммы N+F

- Средствами debug вычислить число L как младший байт разности N-F
- Заполнить регистры AX, BX, CX числами N, H, L соответственно.
- Заполнить ячейки памяти начиная с адреса 100:10N числами N, H, L соответственно.
- Заполнить диапазон ячеек 120..137 символом Nh.
- Заполнить диапазон ячеек 14N..14H символом Lh.
- Заполнить диапазон ячеек 138..147 комбинацией символов Nh+2h.
- Заполнить диапазон ячеек 148..157 числами 3 и 9.
- Скопировать диапазон 14N длиной в N+3 ячейки в ячейки начиная со 158.
- Перенести диапазон 14N..157 в диапазон начиная с ячейки 17L.
- Ввести с адреса 118 строку «00001111», а с адреса 128 ввести строку «22223333». Склеить эти строки, расположив их с адреса 140.
- Найти символы «$» в диапазоне 130..9FF.

Примечание: код символа «$» - 24h

- Сравнить любые две области памяти по своему выбору
- Проверка копирайта BIOS и серийного номера

Сведения об авторских правах на BIOS встроены в ROM BIOS по адресу `FE00:0000`. Строку с копирайтом можно легко найти в ASCII-последовательности, а серийный номер - в виде шестнадцатеричного числа. Хотя, строка с указанием авторских прав может быть длинной и не умещаться в выведенную область памяти. В таком случае следует просто ввести еще раз `d`.

Определить и включить в протокол самостоятельно.

- Проверка даты произвоства BIOS

Эта дата также записана в ROM BIOS начиная с адреса FFFF:5. После выполнения соответствующей команды в ASCII-последовательности будет находиться эта дата, записанная в формате мм/дд/гг.

Определить и включить в протокол самостоятельно.

7 Вопросы для самопроверки

1. Что называется «дампом памяти»?
2. Сколько байт занимает одна ячейка памяти в debug?
3. Какими двумя способами можно задать диапазон копируемых ячеек?
4. Как называется макрокоманда отладчика debug, которая работает с регистрами процессора?
5. Как называется макрокоманда отладчика debug, которая позволяет просматривать содержимое ячеек памяти ОЗУ?
6. Как называется макрокоманда отладчика debug, которая позволяет редактировать вручную содержимое ячеек памяти ОЗУ?
7. Как называется макрокоманда отладчика debug, которая позволяет редактировать вручную содержимое ячеек памяти ОЗУ?
8. Как называется макрокоманда отладчика debug, которая позволяет копировать содержимое диапазона ячеек памяти ОЗУ в другой диапазон?
9. Как называется макрокоманда отладчика debug, которая позволяет вести поиск данных в диапазоне ячеек памяти ОЗУ?
10. Как называется макрокоманда отладчика debug, которая позволяет сравнивать между собой два диапазона ячеек памяти ОЗУ?
11. Как называется макрокоманда отладчика debug, которая позволяет суммировать и вычитать шестнадцатиричные числа?
12. Какой командой можно просмотреть содержимое ячеек памяти ОЗУ начиная с адреса 1000:1000 и заканчивая адресом 1000:1100?
13. Какой командой можно просмотреть содержимое 150 ячеек памяти ОЗУ начиная с адреса 1000:1500?
14. Какой командой можно заполнить память ОЗУ тремя байтами `01 02 03` начиная с адреса 1000:1500 (не входя в режим ввода информации вручную)?
15. Какой командой можно сравнить между собой содержимое 1000 ячеек памяти ОЗУ начиная с адреса 1000:1000 и с адреса 1000:2100?
16. Какой командой можно сравнить между собой содержимое диапазона ячеек памяти ОЗУ начиная с адреса 1000:1000 и заканчивая адресом 1000:1399 и с адреса 1000:2100?
17. Какой командой можно заполнить содержимое 1500 ячеек памяти ОЗУ байтом `FF`, начиная с адреса 1040:1000?

18. Какой командой можно провести поиск байта `11` среди 3500 ячеек памяти ОЗУ, начиная с адреса 1100:1000?

19. Какой командой можно провести поиск байта `11` в диапазоне ячеек памяти ОЗУ, начиная с адреса 1000:1800 и заканчивая 1000:3999?

20. Какой командой можно заполнить регистр AX байтами `44 55` (не входя в режим ввода информации вручную)?

21. Какой командой можно заполнить регистр IP байтами `14 15` (не входя в режим ввода информации вручную)?

Для ответов разрешается пользоваться средой `debug`.

8 Приложение

8.1 Приложение 1. Краткий справочник команд среды debug.exe

Примечание Символами [] отмечены необязательные параметры.

- a (assemble)

Транслирование команд ассемблера в машинный код; адрес по умолчанию - CS:0100h.

```
a [<адрес_начала_кода>]
```

- c (compare)

Сравнение содержимого двух областей памяти; по умолчанию используется DS. В команде указывается либо длина участков, либо диапазон адресов.

```
c <начальный_адрес_1> L<длина> <начальный_адрес_2>
c <начальный_адрес_1> <конечный_адрес_1> <начальный_адрес_2>
```

- d (display/dump)

Вывод содержимого области памяти в шестнадцатеричном и ASCII-форматах. По умолчанию используется DS; можно указывать длину или диапазон.

```
d [<начальный_адрес> [L<длина>]]
d [начальный_адрес конечный_адрес]
```

- e (enter)

Ввод в память данные или инструкции машинного кода; по умолчанию используется DS.

```
e [<адрес> [<инструкции/данные>]]
```

- f (fill)

Заполнение области памяти данными из списка; по умолчанию используется DS. Использовать можно как длину, так и диапазон.

```
f <начальный_адрес_1> L<длина> '<данные>'
f <начальный_адрес> <конечный_адрес> '<данные>'
```

- g (go)

Выполнение отлаженной программы на машинном языке до указанной точки останова; по умолчанию используется CS. Убедитесь, что IP содержит корректный адрес.

```
g [=<начальный_адрес>] <адрес_останова> [<адрес_останова> ...]
```

- h (hexadecimal)

Вычисление суммы и разности двух шестнадцатеричных величин.

```
h <величина_1> <величина_2>
```

- i (input)

Считывание и вывод одного байта из порта.

```
i <адрес_порта>
```

- l (load)

Загрузка файла или данных из секторов диска в память; по умолчанию - CS:100h. Файл можно указать с помощью команды n или аргумента при запуске debug.exe.

```
l [<адрес_в_памяти_для_загрузки>]
l [<адрес_в_памяти_для_загрузки> [<номер_диска> <начальный_сектор> <количество_сек
```

- m (move)

Копирование содержимого ячеек памяти; по умолчанию используется DS. Можно указывать как длину, так и диапазон.

```
m <начальный_адрес> L<длина> <адрес_назначения>
m <начальный_адрес> <конечный_адрес> <адрес_назначения>
```

- n (name)

Указание имени файла для команд L и W.

```
n <имя_файла>
```

- o (output)

Отсылка байта в порт.

```
o <адрес_порта> <байт>
```

- p (proceed)

Выполнение инструкций CALL, LOOP, INT или повторяемой строковой инструкции с префиксами REPnn, переходя к следующей инструкции.

```
p [=<адрес_начала>] [<количество_инструкций>]
```

- q (quit)

Завершение работы debug.exe.

```
q
```

- r (register)

Вывод содержимого регистров и следующей инструкции.

```
r <имя_регистра>
```

- s (search)

Поиск в памяти символов из списка; по умолчанию используется DS. Можно указывать как длину, так и диапазон.

```
s <начальный_адрес> L<длина> '<данные>'
s <начальный_адрес> <конечный_адрес> '<данные>'
```

- t (trace)

Пошаговое выполнение программы. Как и в команде P, по умолчанию используется пара CS:IP. *Для выполнения прерываний лучше пользоваться командой P*

```
t [=<адрес_начала>] [<количество_выполняемых_команд>]
```

- u (unassemble)

Дизассемблирование машинного кода; по умолчанию используется пара CS:IP. К сожалению, debug.exe некорректно дизассемблирует специфические команды процессоров 80286+, хотя они все равно выполняются корректно.

```
u [<начальный_адрес>]
u [<начальный_адрес конечный_адрес>]
```

- w (write)

Запись файла из debug.exe; необходимо обязательно задать имя файла командой N, если он не был загружен. Программы записываются только в виде файлов .COM!

```
w [<адрес> [<номер_диска> <начальный_сектор> <количество_секторов>]]
```

8.2 Приложение 2. Краткий справочник по регистру флагов

Регистр флагов — регистр процессора, отражающий текущее состояние процессора. В микропроцессорах Intel 8086 имеет название FLAGS и является 16-разрядным. Расширенные регистры EFLAGS и RFLAGS, введённые в архитектурах IA-32 (процессоры 80386) и x86-64, являются 32-битными и 64-битными соответственно.

Расширенные регистры сохраняют обратную совместимость.

Регистр флагов содержит группу флагов состояния, управляющий флаг и группу системных флагов.

Девять из 16 битов флагового регистра являются активными и определяют текущее состояние машины и результатов выполнения. Многие арифметические команды и команды сравнения изменяют состояние флагов. Назначение флаговых битов:

- O (Переполнение)

Указывает на переполнение старшего бита при арифметических командах.

- D (Направление)

Обозначает левое или правое направ ление пересылки или сравнения строковых данных (данных в памяти превышающих длину одного слова).

- I (Прерывание)

Указывает на возможность внешних прерываний.

- T (Пошаговый режим)

Обеспечивает возможность работы процессора в пошаговом режиме. На пример, программа DOS DEBUG уста навливает данный флаг так, что воз можно пошаговое выполнение каждой команды для проверки изменения содержимого регистров и памяти.

- S (Знак)

Содержит результирующий знак после арифметических операций (0 - плюс, 1 - минус).

- Z (Ноль)

Показывает результат арифметичес ких операций и операций сравнения (0 - ненулевой, 1 - нулевой результат).

- A (Внешний перенос)

Содержит перенос из 3-го бита для 8-битных данных, используется для специальных арифметических операций.

- P (Контроль четности)

Показывает четность младших 8-битовых данных (1 - четное и 0 - нечетное число).

- C (Перенос)

Содержит перенос из старшего бита, после арифметических операций, а также последний бит при сдвигах или циклических сдвигах.

При программировании на ассемблере наиболее часто используются флаги O, S, Z, и C для арифметических операций и операций сравнения, а флаг D для обозначения направления в операциях над строками.

Значение некоторых флагов в регистре флагов можно изменять напрямую, с помощью специальных инструкций (например, `CLD` для сброса флага направления), но нет инструкций, которые позволяют обратиться (проверить или изменить) к регистру флагов как к обычному регистру. Однако, можно сохранять регистр флагов в стек или регистр `(E)AX` и восстанавливать регистр флагов из них с помощью инструкций `LAHF`, `SAHF`, `PUSHF`, `PUSHFD`, `POPF` и `POPFD`.

При приостановке задачи (используя многозадачные возможности процессора), процессор автоматически сохраняет значение флага регистров в `TSS (task state segment)`, при активизации новой задачи процессор загружает регистр флагов из `TSS` новой задачи.

Когда активизируется обработчик прерывания или обработчик исключительной ситуации, процессор автоматически сохраняет значение флага регистров в текущем стеке.