КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 11 базовые инструкции IA32

+375 17 293 8039 (505a-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

Кафедра ЭВМ, 2022

2022.04.13

Оглавление

Базовые инструкции IA32

- инструкции перемещения данных;
- арифметические инструкции (двоичная и двоично-десятичная арифметика);
- логические инструкции;
- инструкции сдвигов;
- инструкции для работы с битами и байтами;
- инструкции для работы со строками;
- инструкции передачи управления;
- инструкции ввода/вывода;
- инструкции управления флагами;
- инструкции для работы с сегментными регистрами;
- другие инструкции.

Ввод/вывод

Помимо передачи данных во внешнюю память и из нее, процессоры IA-32 также могут пересылать данные в порты ввода-вывода (порты ввода-вывода) и получать данные из них.

Порты ввода/вывода создаются в аппаратном обеспечении системы с помощью электронных схем, которые декодируют управляющие сигналы, данные и адреса на выводах процессора.

Для связи с периферийными устройствами порты ввода/вывода конфигурируются.

Порт ввода/вывода может быть входным портом, выходным портом или двунаправленным.

Некоторые порты ввода/вывода используются для передачи данных, например, в регистры передачи и из регистров приема устройства последовательного интерфейса.

Другие порты ввода/вывода используются для управления периферийными устройствами, например, такими как управляющие регистры контроллера диска.

Доступ к портам ввода/вывода

Процессор позволяет приложениям получать доступ к портам ввода-вывода одним из двух способов:

- через отдельное адресное пространство ввода-вывода;
- через отображенный в памяти ввод-вывод.

Доступ к портам ввода-вывода через адресное пространство ввода-вывода осуществляется с помощью набора инструкций ввода/вывода и специального механизма защиты ввода/вывода.

Доступ к портам ввода-вывода через отображенный в памяти ввод-вывод осуществляется с помощью универсальных инструкций перемещения данных и строковых инструкций, а защита обеспечивается за счет сегментации или разбиения на страницы.

Порты ввода-вывода могут быть отображены так, чтобы они появлялись в адресном пространстве ввода-вывода, в адресном пространстве физической памяти (ввод-вывод с отображением в памяти — **memory mapped I/O**) или в обоих.

Одним из преимуществ использования адресного пространства ввода-вывода является тот факт, что запись в порты ввода-вывода гарантированно будет завершена до выполнения следующей инструкции в потоке команд.

Таким образом, запись в порты аппаратного обеспечения системы управления приведет к тому, что оно установится в новое состояние прежде, чем будут выполнены какие-либо другие инструкции.

Аппаратное обеспечение портов В/В

С аппаратной точки зрения адресация ввода-вывода осуществляется через адресные линии шины процессора.

Для процессоров семейства P6, Pentium 4 и Intel Xeon линии команд запроса типа доступа указывают, относится ли состояние адресных линий процессора к адресам памяти или к адресам ввода/вывода. Для процессоров Pentium и более ранних процессоров IA-32 вывод M/IO# указывает адрес памяти (1) или адрес ввода/вывода (0).

Когда выбрано отдельное адресное пространство ввода/вывода, аппаратные средства платформы соответствующим образом декодируют транзакцию шины, чтобы выбрать порты ввода/вывода, а не память.

Данные передаются между процессором и устройством ввода/вывода по линиям данных.

Адресное пространство ввода-вывода

Адресное пространство ввода-вывода отделено от адресного пространства физической памяти. Оно состоит из 2^{16} (64 КБ) индивидуально адресуемых 8-битных портов ввода-вывода, пронумерованных от 0 до FFFFH.

Адреса портов от 0F8H до 0FFH зарезервированы и им не следует назначать порты.

Результат попытки адресации за пределами адресного пространства ввода-вывода FFFFH зависит от реализации конкретных процессоров.

Любые два последовательных 8-битных порта могут рассматриваться как один 16-битный порт, а любые четыре последовательных порта могут быть 32-битным портом.

Это позволяет процессору в адресном пространстве ввода-вывода передавать на устройство или из него 8, 16 или 32 бита.

Как и слова в памяти, 16-битные порты должны быть выровнены по четным адресам (0, 2, 4, ...), чтобы все 16 битов могли передаваться в одном цикле шины.

Аналогично, 32-битные порты должны быть выровнены по адресам, кратным четырем (0, 4, 8, ...).

Процессор поддерживает обмен данными с невыровненными портами, но при этом снижается производительность из-за использования дополнительных циклов шины.

Точный порядок циклов шины, используемых для доступа к невыровненным портам, не определен.

Если аппаратное или программное обеспечение требует, чтобы порты ввода-вывода были записаны в определенном порядке, этот порядок должен быть указан явно.

Например, чтобы записать в порт ввода-вывода по адресу 2H одно слово, а затем в порт по адресу 4H другое слово, необходимо использовать две операции записи размером в слово, а не одну операцию размером в двойное слово по адресу 2H.

Процессор не обрабатывает ошибки четности для циклов шины в адресном пространстве ввода-вывода. Таким образом, доступ к портам ввода-вывода через адресное пространство ввода-вывода является возможным источником ошибок четности.

Инструкции ввода/вывода

Инструкции процессора ввода-вывода обеспечивают доступ к портам ввода-вывода через адресное пространство ввода-вывода. Эти инструкции нельзя использовать для доступа к отображенным в память портам ввода-вывода.

Существует две группы инструкций ввода-вывода:

1) **IN**, **OUT** передают один элемент (байт, слово или двойное слово) между портом ввода-вывода и регистром общего назначения;

2) **INS**, **INSB**, **INSW**, **INSD**, **OUTS**, **OUTSB**, **OUTSW**, **OUTSD** передают строки элементов (строки байтов, слов или двойных слов) между портом ввода-вывода и памятью.

Инструкции регистрового ввода-вывода

IN (ввод из порта ввода-вывода) и **OUT** (вывод в порт ввода-вывода) перемещают данные между портами и регистром **EAX** (32-битный ввод-вывод), регистром **AX** (16-битный ввод-вывод) или регистром **AL** (8-битный ввод-вывод).

Адрес порта ввода-вывода указывается непосредственным значением в инструкции или значением в регистре **DX**.

Строковые инструкции ввода-вывода

INS (ввод строки из порта ввода-вывода) и **OUTS** (вывод строки в порт ввода-вывода) перемещают данные между портом и местом в памяти.

Адрес порта ввода-вывода, к которому осуществляется доступ, указывается в регистре **DX**.

Адрес памяти источника или получателя указывается в регистре **DS:ESI** или **ES:EDI**, соответственно.

Блочные (строковые) операции

Инструкции **INS** и **OUTS** могут использоваться с одним из префиксов повторения, например, REP, при этом они выполняют строковые (или блочные) операции ввода или вывода.

Префикс повторения **REP** модифицирует инструкции **INS** и **OUTS** таким образом, чтобы они осуществляли передачу блоков данных между портом ввода-вывода и памятью.

В этом случае регистр **ESI** или **EDI** увеличивается или уменьшается (в соответствии с настройкой флага **DF** в регистре **EFLAGS**) после передачи каждого байта, слова или двойного слова между выбранным портом ввода-вывода и памятью.

IN — ввод из порта

```
IN    Acc, imm8    ; Acc — AL, AX, EAX
IN    Acc, DX      ; Acc — AL, AX, EAX
```

Описание

Копирует значение из порта ввода-вывода, указанного во втором операнде (исходный операнд), в целевой операнд (первый операнд).

Операндом-источником может быть порт, адрес которого указан вторым операндом непосредственно (imm8) или в регистре **DX**.

Операндом-приемником может быть регистр **AL**, **AX** или **EAX**, в зависимости от размера порта, к которому осуществляется доступ (8, 16 или 32 бита, соответственно).

Использование регистра **DX** в качестве исходного операнда позволяет получить доступ к адресам портов ввода-вывода от 0 до 65535.

Использование непосредственного байтового значения позволяет получить доступ к адресам портов от 0 до 255.

При доступе к 8-битному порту ввода-вывода размер порта определяется кодом операции. При доступе к 16- и 32-битному порту ввода-вывода размер порта определяется атрибутом размера операнда.

На уровне машинного кода инструкции ввода-вывода будет короче при доступе к 8-битным портам ввода-вывода. В данном случае верхние восемь бит адреса порта будут приниматься равными 0.

Флаги не изменяются

OUT — вывод в порт

```
OUT    imm8, Acc    ; Acc – AL, AX, EAX
OUT    DX, Acc      ; Acc – AL, AX, EAX
```

Описание

Копирует значение из второго операнда (операнда источника) в порт ввода-вывода, указанный в операнде назначения (первом операнде).

Операндом-источником может быть регистр **AL**, **AX** или **EAX**, в зависимости от размера порта, к которому осуществляется доступ (8, 16 или 32 бита соответственно).

Целевой операнд может быть непосредственным байтом или регистром **DX**.

Использование регистра **DX** в качестве исходного операнда позволяет получить доступ к адресам портов ввода-вывода от 0 до 65535. Использование непосредственного байтового значения позволяет получить доступ к адресам портов от 0 до 255.

При доступе к 8-битному порту ввода-вывода размер порта определяется кодом операции. При доступе к 16- и 32-битному порту ввода-вывода размер порта определяется атрибутом размера операнда.

На уровне машинного кода инструкции ввода-вывода будет короче при доступе к 8-битным портам ввода-вывода. В данном случае верхние восемь бит адреса порта будут приниматься равными 0.

Флаги не изменяются

INS/INSB/INSW/INSD — ввод из порта в строку памяти

```
INS     m, DX    ; m8, m16, m32 [ES:(E)DI]
INSB             ; m8
INSW             ; m16
INSD             ; m32
```

Описание

Копирует данные из порта ввода-вывода, указанного операндом-источникам (второй операнд) в операнд-получатель (первый операнд).

Операндом-источником является адрес порта ввода-вывода (от 0 до 65 535), который считывается из регистра **DX**.

Операндом-получателем является ячейка памяти, адрес которой считывается из регистров **ES:DI** или **ES:EDI**, в зависимости от атрибута *address-size* инструкции.

Сегмент ES не может быть переопределен с помощью префикса переопределения сегмента.

Размер порта ввода-вывода, к которому осуществляется доступ (размер операнда источника и назначения), определяется кодом операции для 8-битного порта ввода-вывода или атрибутом размера операнда инструкции для 16- или 32-битного порта.

На уровне языка ассемблера допускаются две формы этой инструкции — с явным и неявным указанием операндов.

Форма с явными операндами **INS** позволяет явно указывать операнд-источник и целевой операнд. Операндом-источником должен быть любой символ, который определенно указывает размер порта ввода-вывода и адреса источника, а операндом-адресатом должен быть регистр **DX**.

Данная форма предоставляется для само-документирования.

Важно: способ документирования, предоставляемый этой формой, может вводить в заблуждение. Символ операнда назначения должен всегда указывать ассемблеру правильный тип (размер) операнда (байт, слово или двойное слово), хотя и не обязан указывать правильное местоположение.

Местоположение всегда указывается регистрами `ES:DI` или `ES:EDI`, которые должны быть правильно загружены перед выполнением инструкции `INS`.

Форма без операндов предоставляет «короткие формы» байтовых, двухбайтовых (слово) и четырех байтовых (двойное слово) версий инструкций **INS**.

В этом случае также **`ES:DI или ES:EDI`** является операндом-получателем, а регистр **DX** — операндом-источником.

Размер порта ввода-вывода указывается с помощью мнемоники: **INSB** (байт), **INSW** (слово) или **INSD** (двойное слово).

После передачи байта, слова или двойного слова из ячейки памяти в порт ввода-вывода регистр **DI**/**EDI** автоматически увеличивается или уменьшается в соответствии с настройкой флага **DF** в регистре **EFLAGS**. Если флаг **DF=0**, регистр **EDI** инкрементируется, если флаг **DF=1**, регистр **EDI** декрементируется.

Регистр **DI**/**EDI** увеличивается или уменьшается на 1 для операций с байтами, на 2 для операций со словами и на 4 для операций с двойными словами.

Инструкциям **INS**, **INSB**, **INSW** и **INSD** может предшествовать префикс **REP** для блочного ввода нескольких байтов, слов или двойных слов, количество которых указывается в регистре ECX перед началом операции.

Данная инструкция полезна только для доступа к портам ввода-вывода, расположенным в адресном пространстве ввода-вывода процессора.

Флаги не изменяются

OUTS/OUTSB/OUTSW/OUTSD — вывод строки памяти в порт

```
OUTS    DX, m    ; m8, m16, m32 [DS:(E)DI]
OUTSB            ; m8
OUTSW            ; m16
OUTSD            ; m32
```

Описание

Копирует данные из исходного операнда (второго операнда) в порт ввода-вывода, указанный в операнде назначения (первом операнде).

Операндом-источником является ячейка памяти, адрес которой считывается из регистров **DS:SI**, **DS:ESI** в зависимости от атрибута *address-size* команды.

Сегмент DS может быть переопределен с помощью префикса переопределения сегмента.

Операндом-адресатом является адрес порта ввода-вывода (от 0 до 65 535), который считывается из регистра **DX**.

Размер порта ввода-вывода, к которому осуществляется доступ (размер операнда источника и назначения), определяется кодом операции для 8-битного порта ввода-вывода или атрибутом размера операнда инструкции для 16- или 32-битного порта.

На уровне кода ассемблера допускаются две формы этой инструкции — с явным и неявным указанием операндов.

Nasm не поддерживает явное указание операндов, аналогично тому, как это имеет место для MOVS, и по тем же самым причинам.

Форма с явными операндами **OUTS** позволяет явно указывать операнд-источник и целевой операнд. Операндом-источником должен быть любой символ, который определенно указывает размер порта ввода-вывода и адреса источника, а операндом-адресатом должен быть регистр **DX**.

Данная форма предоставляется для самодокументирования.

Важно: способ документирования, предоставляемый этой формой, может вводить в заблуждение — символ операнда-источника должен указывать ассемблеру правильный тип (размер) операнда (байт, слово или двойное слово), но не обязан указывать правильное местоположение. Местоположение всегда указывается регистрами **DS:SI** или **DS:ESI**, которые должны быть правильно загружены перед выполнением инструкции **OUTS**.

Форма без операндов предоставляет «короткие формы» байтовых, двухбайтовых (слово) и четырех байтовых (двойное слово) версий инструкций **OUTS**.

Здесь также **DS:SI** или **DS:ESI** является операндом-источником, а регистр **DX --** операндом назначения. Размер порта ввода-вывода указывается с помощью мнемоники: **OUTSB** (байт), **OUTSW** (слово) или **OUTSD** (двойное слово).

После передачи байта, слова или двойного слова из ячейки памяти в порт ввода-вывода регистр **SI**/**ESI** автоматически увеличивается или уменьшается в соответствии с настройкой флага **DF** в регистре **EFLAGS**. Если флаг **DF=0**, регистр **ESI** инкрементируется, если флаг **DF=1**, регистр **ESI** декрементируется.

Регистр **SI**/**ESI** увеличивается или уменьшается на 1 для операций с байтами, на 2 для операций со словами и на 4 для операций с двойными словами.

Инструкциям **OUTS**, **OUTSB**, **OUTSW** и **OUTSD** может предшествовать префикс **REP** для блочного ввода нескольких байтов, слов или двойных слов, количество которых указывается в регистре ECX перед началом операции.

Данная инструкция полезна только для доступа к портам ввода-вывода, расположенным в адресном пространстве ввода-вывода процессора.

Флаги не изменяются

Механизм защиты ввода/вывода

Когда процессор работает в защищенном режиме, доступ к портам ввода-вывода регулируют следующие механизмы защиты:

1) При доступе к портам ввода-вывода через адресное пространство ввода-вывода доступ контролируют два механизма защиты:

- поле уровня привилегий ввода-вывода (**IOPL**) в регистре **EFLAGS**;
- битовая карта разрешений ввода-вывода сегмента состояния задачи (**TSS**).

2) При доступе к отображенным в память портам ввода-вывода на доступ к этим портам влияет обычная страничная защита и защита сегментации.

Уровень привилегий ввода-вывода

Поле **IOPL** в регистре **EFLAGS** контролирует доступ к адресному пространству ввода-вывода, ограничивая использование выбранных инструкций.

Этот механизм защиты позволяет операционной системе или исполняющей системе устанавливать уровень привилегий, необходимый для выполнения операций ввода-вывода. В типичной модели колец защиты доступ к адресному пространству ввода-вывода ограничен уровнями привилегий 0 и 1 — ядру и привилегированным драйверам устройств разрешено выполнять инструкции ввода-вывода, тогда как менее привилегированным драйверам устройств и прикладным программам доступ в адресное пространство ввода-вывода запрещен.

Для выполнения операций ввода-вывода прикладные программы должны выполнять соответствующие вызовы операционной системы.

Ряд инструкций могут быть выполнены, только если текущий уровень привилегий (**CPL**) программы или задачи, выполняемой в настоящее время, меньше или равен **IOPL**. Это **IN**, **INS**, **OUT**, **OUTS**, **CLI** (флаг сброса разрешения прерывания) и **STI** (установить флаг разрешения прерываний).

Эти инструкции называются командами, чувствительными по отношению к вводу-выводу, потому что они зависят от поля **IOPL**.

Любая попытка менее привилегированной программы или задачи использовать чувствительную инструкцию ввода-вывода приводит к сигналу исключения общей защиты (**#GP**). Поскольку каждая задача имеет свою собственную копию регистра **EFLAGS**, каждая задача может иметь свой собственный **IOPL**.

Для изменения влияния **IOPL** на чувствительные инструкции ввода-вывода с целью предоставить доступ к некоторым портам ввода-вывода менее привилегированным программам или задачам, может использоваться битовая карта разрешений ввода-вывода в **TSS**.

Программа или задача может изменить свой IOPL только с помощью инструкций POPF и IRET, однако такие изменения являются привилегированными. Никакая процедура не может изменить текущий IOPL, если она не выполняется на уровне привилегий 0.

Попытка менее привилегированной процедуры изменить **IOPL** не приводит к исключению — просто **IOPL** остается неизменным.

Для изменения состояния флага **IF** (как и инструкции **CLI** и **STI**) могла бы использоваться инструкция **POPF**, однако она также чувствительна к вводу-выводу.

Процедура может использовать инструкцию **POPF**, чтобы изменить настройку флага **IF**, только если **CPL** меньше или равен текущему **IOPL**. Попытка менее привилегированной процедуры изменить флаг **IF** не приводит к исключению — флаг **IF** просто остается неизменным.

Битовая карта разрешения ввода-вывода

Битовая карта разрешений ввода-вывода — это механизм, предоставляющий менее привилегированным программам или задачам, а также задачам, работающим в режиме virtual-8086, ограниченный доступ к портам ввода-вывода. Эта карта находится в **TSS**.

Поскольку каждая задача имеет свой собственный **TSS**, у каждой задачи есть своя битовая карта разрешений ввода-вывода — это позволяет отдельным задачам предоставить доступ к отдельным портам ввода-вывода.

Если в защищенном режиме **CPL** меньше или равен текущему **IOPL**, процессор разрешает выполнение всех операций ввода-вывода. Если **CPL** больше **IOPL** или если процессор работает в режиме virtual-8086, процессор проверяет битовую карту разрешений ввода-вывода, чтобы определить, разрешен ли доступ к конкретному порту.

Каждый бит на карте соответствует байтовому адресу порта ввода-вывода. Например, бит для адреса 29H порта ввода-вывода в адресном пространстве ввода-вывода находится в позиции 1 бита шестого байта в битовой карте.

Перед предоставлением доступа к вводу-выводу процессор проверяет все биты, соответствующие адресуемому порту. Например, для доступа по двойному слову процессоры проверяют четыре бита, соответствующие четырем соседним 8-битным адресам портов. Если установлен какой-либо бит из проверяемых, инициируется исключение общей защиты (**#GP**). Если все проверяемые биты сброшены, операции ввода-вывода разрешается.

Доступ к портам ввода/вывода из программы на C (linux)

```
#include <sys/io.h>

unsigned char inb(unsigned short int port);
unsigned char inb_p(unsigned short int port);
unsigned short int inw(unsigned short int port);
unsigned short int inw_p(unsigned short int port);
unsigned int inl(unsigned short int port);
unsigned int inl_p(unsigned short int port);

void outb(unsigned char value, unsigned short int port);
void outb_p(unsigned char value, unsigned short int port);
void outw(unsigned short int value, unsigned short int port);
void outw_p(unsigned short int value, unsigned short int port);
void outl(unsigned int value, unsigned short int port);
void outl_p(unsigned int value, unsigned short int port);
```

Вызовы **out{bwl}** выполняют запись в порт, а вызовы **in{bwl}** — чтение из порта.

Вызовы с суффиксом **b** работают с данными шириной один байт, с суффиксом **w** — в одно слово, с суффиксом **l** — в двойное слово.

Вызовы с суффиксом **_p** ждут завершения операции ввода-вывода. Предназначены, в основном, для работы внутри ядра, но могут быть вызваны и из пользовательского пространства.

Программа должна компилироваться с флагами **-O**, **-O2** и им подобными. Данные вызовы определены в виде встроенных макросов и не будут подставляться без включённой оптимизации, что приводит к появлению неразрешаемых ссылок в процессе компоновки программы.

Строковые варианты

```
void insb(unsigned short int port, void *addr, unsigned long int count);
void insw(unsigned short int port, void *addr, unsigned long int count);
void insl(unsigned short int port, void *addr, unsigned long int count);

void outsb(unsigned short int port, const void *addr, unsigned long int count);
void outsw(unsigned short int port, const void *addr, unsigned long int count);
void outsl(unsigned short int port, const void *addr, unsigned long int count);
```

Установка прав на работу с портами ввода/вывода в Linux

```
int ioperm(unsigned long from, // Начиная с порта # from
           unsigned long num,  // Колическтво портов
           int turn_on);       // 0 -- сбросить, 1 -- установить
```

Привилегии наследуются **`fork()`** и **`execve()`**

```
int iopl(int level);           //
```

Изменяет уровень привилегий ввода/вывода вызывающего процесса, задаваемый двумя младшими битами в значении **`level`**.

Дополнительно разрешается **`cli/sti`**.

Привилегии не наследуются ни **`fork()`**, ни **`execve()`**.

Инструкции управления флагами (EFLAG)

Команды управления флагами позволяют читать или изменять состояние выбранных флагов в регистре **EFLAGS**.

Обычно эти инструкции делят на подгруппы:

- **STC/CLC/CMC** — управление флагами переносов
- **CLD/STD** — управление флагами направления;
- **LAHF/SAHF/PUSHF/PUSHFD/POPF/POPFD** — управление флагами в **EFLAGS**;
- **STI/CLI — управление флагами прерывания.**

STC/CLC/CMC — управление флагами переносов

```
STC — Set CF
CLC — Clear CF
CMC — Invert CF
```

Операция

```
EFLAGS.CF ← 1
EFLAGS.CF ← 0
EFLAGS.CF ← NOT EFLAGS.CF
```

Описание

Инструкции **STC**, **CLC** и **CMC** устанавливают, очищают и инвертируют флаг переноса **CF** в **EFLAGS**.

Флаги

Флаги **OF**, **ZF**, **SF**, **AF**, **PF** остаются нетронутыми.

`CLD/STD` — управление флагами направления

```
CLD – Clear Direction Flag (esi++, edi++)
STD – Set Direction Flag   (esi--, edi--)
```

Операция

```
EFLAGS.DF ← 0
EFLAGS.DF ← 1
```

Описание

Инструкции **CLD** и **STD** очищают и устанавливают флаг направления **DF** в **EFLAGS**.

Когда флаг **DF** установлен в 1 строковые операции декрементируют индексные регистры **ESI** и/или **EDI**. В противном случае инкрементируют.

Флаги

Флаги **CF**, **OF**, **ZF**, **SF**, **AF**, **PF** остаются нетронутыми.

STI/CLI — управление флагами прерывания

```
STI — Разрешить прерывания
CLI — Запретить прерывания
```

Описание

Очистка флага **IF** заставляет процессор игнорировать маскируемые ***внешние*** прерывания.

Состояние флага **IF**, а также инструкции **CLI** и **STI** не влияют на генерацию исключений и немаскируемых прерываний **NMI**.

Если виртуальные прерывания в защищенном режиме не включены, **CLI** очищает флаг **IF** в регистре **EFLAGS**, а **STI** его устанавливает. Другие флаги при этом не затрагиваются.

После того, как установлен флаг **IF**, процессор начинает реагировать на внешние маскируемые прерывания только после выполнения следующей инструкции.

Задержка обеспечивает разрешение прерываний непосредственно перед возвратом из процедуры или подпрограммы. Например, если за инструкцией **STI** следует инструкция **RET**, **RET** выполнится до того, как будут распознаны внешние прерывания. Если за инструкцией **STI** следует инструкция **CLI** (которая очищает флаг IF), эффект от инструкции **STI** сводится на нет.

Флаги

Флаги кроме **IF** и **VIF** остаются без изменений.

LAHF/SAHF/PUSHF/PUSHFD/POPF/POPFD — управление флагами в EFLAGS

```
LAHF          -- load flags from EFLAGS into AH
SAHF          -- save flags from AH into EFLAGS

PUSHF/PUSHFD -- save EFLAGS into stack
POPF/POPFD   -- load EFLAGS from stack
```

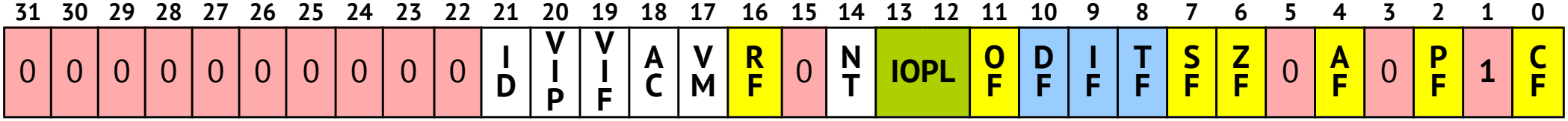
Операция

```
LAHF    AH ← EFLAGS(SF:ZF:0:AF:0:PF:1:CF)
SAHF    EFLAGS(SF:ZF:0:AF:0:PF:1:CF) ← AH;
```

Описание

LAHF загружает флаги состояния в регистр **AH**. Все флаги остаются нетронутыми.

SAHF сохраняет флаги **SF**, **ZF**, **AF**, **PF**, **CF** в регистре **EFLAGS** значениями из соответствующих бит регистра **AH** (7, 6, 4, 2, 0, соответственно). Биты 1, 3, 5 регистра **AH** игнорируются. Соответствующие зарезервированные биты в **EFLAGS** остаются нетронутыми.



PUSHF/PUSHFD — сохраняют регистр EFLAGS в стеке.

Если текущий размер операнда равен 32, указатель стека уменьшается на 4 и содержимое **EFLAGS** помещается в стек.

Если текущий размер операнда равен 16, указатель стека уменьшается на 2, а в стек помещается младшее слово **EFLAGS** (**FLAGS**).

При копировании регистра **EFLAGS** в стек флаги **VM** (16) и **RF** (17) не копируются, а очищаются в образе, сохраненном в стеке.

Мнемоники **PUSHF** и **PUSHFD** производят один и тот же код — действие зависит от размера операнда.

POPF/POPFD — извлечь регистр **EFLAGS** из стека.

Если текущий размер операнда равен 32, из стека извлекаются 4 байта и помещаются в **EFLAGS**, после чего указатель стека увеличивается на 4.

Если текущий размер операнда равен 16, из стека извлекаются 2 байта и помещаются в младшую половину **EFLAGS (FLAGS)**, после чего указатель стека увеличивается на 2.

Мнемоники **PUSHF** и **PUSHFD** производят один и тот же код — действие зависит от размера операнда.

Инструкции для работы с сегментными регистрами

Процессор предоставляет различные инструкции, которые обращаются непосредственно к сегментным регистрам процессора.

Эти инструкции используются только в том случае, если операционная система или исполняющая система использует модель памяти с сегментированным или реальным адресом.

Обычно эти инструкции делят на:

- инструкции загрузки и сохранения сегментных регистров;
- инструкции дальней передачи управления;
- инструкции вызова программных прерываний;
- инструкции обслуживания дальних указателей.

Инструкции загрузки и сохранения сегментных регистров

Инструкция **MOV** и инструкции **PUSH** и **POP** могут перемещать 16-битные селекторы сегментов в и из сегментных регистров (**DS**, **ES**, **FS**, **GS** и **SS**).

Инструкция MOV не может быть использована для загрузки регистра CS.

Перемещения всегда осуществляются между регистрами сегмента и регистрами общего назначения или памятью.

Перемещения между сегментными регистрами не поддерживаются.

Команды **POP** и **MOV** не могут поместить значение в регистр **CS** — на регистр **CS** напрямую влияют только дальние варианты инструкций передачи управления **JMP**, **CALL** и **RET**.

Информация о селекторе сегмента и дескрипторе сегмента проверяется при загрузке этой информации во время исполнения.

```
PUSH  SegR    ; CS, SS, DS, ES, FS, GS (32-битный режим)
POP   SegR    ;     Ssб DS, ES, FS, GS (32-битный режим)
```

LDS/LES/LFS/LGS/LSS — загрузка дальних указателей (Seg Fault)

```
LxS     r16, m16:16 ; Load x SegReg
LxS     r32, m16:32 ; Load x SegReg
```

Инструкции загружают дальний указатель (селектор и смещение сегмента) из второго операнда (операнд-источник) в регистр сегмента и первый операнд (операнд-назначения).

Исходный операнд указывает на 48-битный (m16:32) или 32-битный (m16:16) указатель в памяти в зависимости от текущей настройки атрибута размера операнда (32 или 16 бит соответственно).

Код операции и операнд-адресат определяют пару регистр сегмента/регистр общего назначения. 16-битный селектор сегмента из исходного операнда загружается в регистр сегмента, указанный с помощью кода операции (**DS**, **SS**, **ES**, **FS** или **GS**). 32-разрядное или 16-разрядное смещение загружается в регистр, указанный в операнде-приемнике.

Если инструкция выполняется в защищенном режиме, в скрытую часть выбранного регистра сегмента загружается дополнительная информация из дескриптора сегмента, на который указывает селектор сегмента в операнде источника.

В непривилегированном режиме эти инструкции генерируют исключение «ошибка сегментации (segfault)».

В защищенном режиме селектор **NULL** (значения от **0000** до **0003**) может быть загружен в регистры **DS**, **ES**, **FS** или **GS**, не вызывая исключения защиты. Однако, любая последующая ссылка на сегмент, соответствующий регистр сегмента которого загружен селектором **NULL**, вызывает исключение общей защиты (**#GP**), в результате ссылки на сегмент памяти не возникает.

Флаги остаются без изменений.

Другие инструкции

Ряд инструкций предоставляет такие функции, как загрузка эффективного адреса, выполнение «бездействия», извлечение идентификационной информации о процессоре.

Следующие инструкции выполняют операции, которые представляют интерес для прикладных программистов. Это:

- **LEA** — инструкция вычисления эффективного адреса;
- **XLAT**, **XLATB** — инструкции табличного поиска;
- **CPUID** — инструкция идентификации процессора;
- **NOP** и «неопределенная» инструкция.

Инструкция вычисления эффективного адреса

Команда LEA (**L**oad **E**ffective **A**ddress) вычисляет эффективный адрес исходного операнда в памяти (смещение в сегменте) и помещает его в регистр общего назначения.

Синтаксис

```
LEA    r, m
```

Описание

Вычисляет эффективный адрес второго операнда (исходный операнд) и сохраняет его в первом операнде (целевой операнд). Операндом-источником является адрес памяти (часть смещения), заданная одним из режимов адресации процессоров. Целевой операнд — регистр общего назначения.

На действие, выполняемое этой инструкцией, влияют атрибуты *address-size* и operand-size. Атрибут размера операнда инструкции определяется выбранным регистром, в то время как атрибут размера адреса определяется атрибутом сегмента кода.

Эта инструкция может интерпретировать любой из режимов адресации процессора и может выполнять любое индексирование или масштабирование, которые могут потребоваться. Инструкция особенно полезна для инициализации регистров **`ESI`** или **`EDI`** перед выполнением строковых инструкций или для инициализации регистра **`EBX`** перед инструкцией **`XLAT`**.

Флаги остаются без изменений

XLAT, XLATB — инструкции табличного поиска

Инструкции **XLAT** и **XLATB** выполняют поиск по таблице и заменяют содержимое регистра **AL** байтом, считанным из таблицы трансляции, расположенной в памяти.

Начальное значение в регистре **AL** интерпретируется как индекс без знака в таблице.

Значение индекса добавляется к содержимому регистра **EBX**, содержащему базовый адрес таблицы, с целью вычисления адреса соответствующей табличной записи.

Данные инструкции используются для таких приложений, как преобразование кодов символов из одного алфавита в другой (например, код ASCII можно использовать для поиска его эквивалента EBCDIC в таблице).

Синтаксис

```
XLAT    m8
XLATB
```

Операция

```
AL ← [DS:(E)BX + unsigned(AL)]
```

Описание

Находит запись байта в таблице в памяти, используя содержимое регистра AL в качестве индекса таблицы, затем копирует содержимое записи таблицы обратно в регистр AL. Индекс в регистре AL рассматривается как целое число без знака.

Инструкции **XLAT** и **XLATB** получают базовый адрес таблицы в памяти из регистров **DS:EBX** или **DS:BX** в зависимости от атрибута *address-size* команды, равного 32 или 16, соответственно.

Сегмент **DS** может быть переопределен с помощью префикса переопределения сегмента.

На уровне кода ассемблера допускаются две формы этой инструкции — форма с явным операндом и форма без операнда.

Форма с явным операндом **XLAT** позволяет явно указывать базовый адрес таблицы в символьном виде.

Эта форма предоставляется для самодокументирования кода.

Данная форма может вводить в заблуждение, поскольку символьное имя таблицы не обязательно должно указывать на правильный базовый адрес.

Базовый адрес всегда указывается регистрами **DS:(E)BX**, которые должны быть адекватно загружены перед выполнением инструкции **XLAT**.

Форма без операндов (**XLATB**) является «краткой формой» инструкций **XLAT**. В этом случае процессор также предполагает, что регистры **DS:(E)BX** содержат базовый адрес таблицы.

Флаги

Флаги остаются без изменений

CPUID — инструкция идентификации процессора

Данная инструкция возвращает информацию о процессоре, на котором она выполняется.

Синтаксис

```
CPUID
```

Описание

Флаг **ID** (бит 21) в регистре **EFLAGS** указывает, поддерживается ли **CPUID**.

Если программный код может установить и сбросить этот флаг, процессор, на котором выполняется этот код, поддерживает инструкцию **CPUID**.

CPUID возвращает идентификационные данные процессора и информацию о функциях в регистрах **EAX**, **EBX**, **ECX** и **EDX**. Вывод команды зависит от содержимого регистра **EAX** при ее выполнении (в некоторых случаях также и **ECX**).

Например, следующий псевдокод загружает в **EAX** значение **00H** и заставляет **CPUID** возвратить максимальное возвращаемое значение (Maximum Return Value) и строку идентификации поставщика (Vendor Identification String) в соответствующих регистрах:

```
mov    eax, 00h
cpuid
```

Что и когда возвращает CPUID, детально описано убористым текстом на 33-х страницах «Руководства Программиста по архитектуре ia32(e) Order # 325462-074US April.2021»

NOP и «неопределенная» инструкция UD2

Инструкция **NOP** (**N**o **OP**eartion) увеличивает регистр **EIP**, чтобы тот указывал на следующую инструкцию, и больше ничего не делает.

Синтаксис

```
NOP          ; 90 — однобайтовая
NOP    r/m   ; 0F 1F / xx0 — мультибайтовая
```

Операция

Однобайтовая **NOP** является мнемоническим псевдонимом для инструкции

```
XCHG (E)AX, (E)AX
```

Описание

Эта инструкция не выполняет никаких действий. Это однобайтовая или многобайтовая **NOP**, которая занимает место в потоке команд, но не влияет на машинный контекст, за исключением регистра **EIP**.

Многобайтовая инструкция **NOP** не выполняет никаких операций на поддерживаемых процессорах и генерирует исключение неопределенного кода операции на процессорах, которые не поддерживают многобайтовую **NOP**.

Форма операнда памяти инструкции позволяет программному обеспечению создавать последовательность байтов «без операции» как одну инструкцию.

Инструкция **UD2** генерирует недопустимое исключение кода операции.

Синтаксис

```
UD2
```

Описание

Инструкция предназначена для того, чтобы программное обеспечение могло проверить работу обработчик исключений кода недопустимой операции.

MOVBE — перемещение данных после обмена байтов

```
MOVBE    r, m   ; 16, 32
MOVBE    m, r   ; 16, 32
```

Описание

Выполняет операцию обмена байтов (swap) для данных, скопированных из второго операнда (исходный операнд), и сохраняет результат в первом операнде (целевой операнд).

В качестве операнда-источника и операнда-назначения могут использоваться регистры общего назначения или ячейки памяти. Однако, оба операнда не могут быть одновременно регистрами, и только один операнд может быть ячейкой памяти. Оба операнда должны иметь одинаковый размер, который может быть словом или двойным словом.

Инструкция MOVBE предназначена для обмена байтами при чтении из памяти или при записи в память. Она обеспечивает поддержку для преобразования значений с прямым порядком байтов в формат с обратным порядком байтов и наоборот.

Флаги остаются без изменений

Примечание

```
BSWAP   r32    ; (Byte swap) переставляет порядок байт в 32-разрядном регистре
XCHG    r, r/m ; (Exchange)  обменивает содержимое операндов
XCHG    r/m, r ; (Exchange)  обменивает содержимое операндов
```

Инструкции Enter and Leave

ENTER — Make Stack Frame for Procedure Parameters

```
ENTER imm16, 0    ; создать для процедуры кадр стека
ENTER imm16, 1    ; создать для процедуры кадр стека с указателем вложенности
ENTER imm16, imm8 ; создать для процедуры кадр стека с указателями вложенности
LEAVE             ; вернуть все, как было до ENTER
```

Архитектура IA-32 поддерживает альтернативный метод выполнения вызовов процедур, используя инструкции **ENTER** (вход в процедуру) и **LEAVE** (выход из процедуры). Эти инструкции автоматически создают и освобождают соответственно стековые фреймы (стековые кадры) для вызываемых процедур.

Стековые фреймы содержат пространство фиксированного размера для размещения локальных переменных, а также указатели, необходимые для последовательного возврата из вызванных процедур. Эти инструкции также позволяют реализовывать «правила области видимости», чтобы процедуры могли обращаться к своим собственным локальным переменным и некоторому количеству других переменных, расположенных в других фреймах стека.

ENTER и **LEAVE** предлагают два преимущества:

- они обеспечивают поддержку со стороны машинного языка для вызовов процедур из блочно-структурированных языков, таких как C и Pascal;

- они упрощают процедуру входа и выхода в сгенерированном компилятором коде.

Инструкция **ENTER** создает кадр стека процедуры, состоящий из пространства для динамической памяти и памяти для от одного до 32-х указателей возврата.

Первый операнд (imm16) указывает размер динамической памяти в стековом фрейме (то есть количество байтов, динамически выделяемых в стеке для процедуры).

Второй операнд (imm8) указывает уровень лексической вложенности (от 0 до 31) процедуры.

Размер в байтах пространства хранения для указателей кадров определяется уровнем вложенности (imm8 mod 32) и атрибутом **OperandSize**.

Уровень вложенности определяет количество указателей кадров, которые копируются в «область отображения» нового кадра стека из предыдущего кадра. Размер указателя кадра по умолчанию — это атрибут **StackAddrSize**, но его можно изменить с помощью префикса 66H. Таким образом, атрибут **OperandSize** определяет размер каждого указателя кадра, который будет скопирован в кадр стека, и данных, передаваемых из регистра **(R|E)SP** в регистр **(R|E)BP**.

Инструкции **ENTER** и сопутствующие инструкции **LEAVE** предназначены для поддержки языков с блочной структурой.

Инструкция **ENTER** (если используется) обычно является первой инструкцией в процедуре и используется для установки нового кадра стека для процедуры.

Инструкция **LEAVE** используется в конце процедуры (непосредственно перед инструкцией **RET**) для освобождения кадра стека.

Если уровень вложения равен 0, процессор помещает указатель кадра из регистра **(R|E)BP** в стек, копирует текущий указатель стека из регистра **(R|E)SP** в регистр **(R|E)BPRBP** и загружает регистр **(R|E)SP** текущим значением указателя стека минус значение в операнде размера.

Для уровней вложенности 1 или выше процессор помещает дополнительные указатели кадров в стек перед настройкой указателя стека. Эти дополнительные указатели кадров предоставляют вызываемой процедуре точки доступа к другим вложенным кадрам в стеке. См. «Procedure Calls for Block-Structured Languages» в главе 6.