КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 03.2

Адресация операндов и формат инструкций

Преподаватель: Поденок Леонид Петрович, 505a-5

+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

2022.02.23

Оглавление

Адресация операндов

Машинные инструкции IA-32 работают как с операндами, так и без. Некоторые операнды указываются явно, а другие — неявно.

В инструкции может быть указано до трех операндов.

Двухоперандные инструкции содержат информацию об адресе операнда источника (**src**) и адресе операнда-назначения (**dst**).

Данные для операнда-источника могут быть расположены:

- в самой инструкции (непосредственный операнд — **immediate** operand);
- в регистре;
- в ячейке памяти (memory location);
- в порту ввода/вывода (I/O port).

Когда инструкция возвращает данные в операнд-назначение (dst), они могут быть возвращены в:

- регистр;
- ячейку памяти;
- порт ввода/вывода.

Непосредственные операнды (Immediate Operands)

Некоторые инструкции используют в качестве операнда-источника данные, закодированные в самой инструкции. Эти операнды называются **непосредственными операндами** (или просто непосредственными). Например, следующая инструкция ADD добавляет непосредственное значение 14 к содержимому регистра EAX:

```
ADD EAX, 14
```

Все арифметические инструкции (за исключением инструкций DIV и IDIV) допускают в качестве операнда-источника непосредственное значение.

Максимальное значение, которое можно использовать в качестве непосредственного операнда, варьируется в зависимости от инструкций, но никогда не может быть больше максимального значения целого числа без знака ($2^{32} - 1$).

Регистровые операнды (Register Operands)

Операнды-источники и операнды-назначения в зависимости от выполняемой инструкции могут быть любыми из регистров:

- восемь 32-разрядных РОН[1] (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP);
- восемь 16-разрядных РОН (AX, BX, CX, DX, SI, DI, SP, BP);
- восемь 8-разрядных РОН (AH, BH, CH, DH, AL, BL, CL, DL);
- сегментные регистры (CS, DS, SS, ES, FS, GS);
- регистр флагов EFLAGS;
- регистры x87 FPU (ST0-ST7), слово состояния, управляющее слово, слово тегов, указатель на операнд, указатель на инструкцию;
- восемь MMX регистров (MM0-MM7);
- восемь XMM регистров (XMM0-XMM7) и регистр управления/состояния MXCSR;
- регистры управления (CR0, CR2, CR3, CR4);
- регистры-указатели на системные таблицы (GDTR, LDTR, IDTR, регистр задачи);
- регистры отладки (DR0, DR1, DR2, DR3, DR6, DR7);
- регистры MSR.

1 РОН — Регистр Общего Назначения

Некоторые инструкции (например, инструкции **DIV** и **MUL**) используют операнды типа **quadword** (квадрослово), располагающиеся в паре 32-разрядных регистров.

Пары регистров в описаниях инструкций представлены разделяющим их двоеточием. Например, в паре регистров **EDX:EAX**, содержащих операнд-квадрослово, **EDX** содержит биты высокого порядка, а **EAX** — биты низкого порядка.

Для загрузки и сохранения содержимого регистра **EFLAGS**, или для установки или очистки отдельных флагов в этом регистре можно использовать несколько инструкций (например, инструкции **PUSHFD** и **POPFD**).

Другие инструкции (например, инструкции **Jcc**) используют состояние флагов состояния (state of the status flags) в регистре **EFLAGS** в качестве условий для ветвления или других операций принятия решений.

Процессор содержит набор системных регистров, используемых для:

- управления памятью
- обработки прерываний и исключений
- управления задачами
- управления процессором и отладкой.

Некоторые из этих системных регистров доступны прикладной программе, операционной системе или исполняющей системе с помощью набора системных инструкций.

При обращении к системному регистру с помощью системной инструкции регистр обычно является подразумеваемым операндом команды.

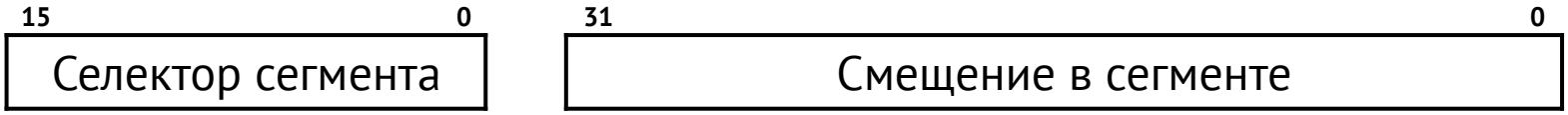
Операнды в памяти (Memory Operands)

На операнды-источники и операнды-назначения, расположенные в памяти, ссылаются с помощью селектора сегмента и смещения (см. Рисунок).

Селекторы сегмента указывают сегмент, в котором находится операнд.

Смещения указывают линейный или эффективный адрес операнда.

Смещения могут быть размером в 32 бита (размер обозначается как **`m16:32`**) или 16 бит (размер обозначается как **`m16:16`**).



Указание селектора сегмента

Селектор сегмента может быть указан неявно или явно.

Наиболее распространенный способ задания селектора сегмента — это загрузить его в регистр сегмента и затем позволить процессору неявно выбирать регистр в зависимости от типа выполняемой операции. Процессор автоматически выбирает сегмент в соответствии с правилами, приведенными в таблице.

Тип ссылки	Используемые		Правила выбора по умолчанию
	Регистр	Сегмент	
Инструкции	**CS**	Сегмент кода	При извлечении всех инструкций
Стек	**SS**	Сегмент стека	Все push и pop для стека Все ссылки на память, которые используют регистры **ESP** или **EBP** в качестве базового.
Локальные данные	**DS**	Сегмент данных	Все ссылки на память за исключением случаев адресации относительно стека или строки-назначения.
Строка-назначение	**ES**	Сегмент данных, на который указывает **ES**	Цель строковой инструкции.
	FS		
	GS		

При сохранении данных в памяти или загрузке данных из памяти сегмент **DS**, использующийся по умолчанию, может быть переопределен — это позволяет обеспечить доступ к другим сегментам.

В ассемблере переопределение сегмента обычно указывается с помощью оператора «двоеточие» ':'

Например, следующая инструкция **MOV** перемещает значение из регистра **EAX** в сегмент, на который указывает регистр **ES**. Смещение в сегменте содержится в регистре **EBX**:

```
MOV ES:[EBX], EAX
```

На машинном уровне переопределение сегмента указывается с префиксом переопределения сегмента, который представляет собой байт, помещенный в начале кода инструкции.

Следующие сегменты, выбираемые по умолчанию, не могут быть переопределены:

- выборки инструкций должны выполняться из сегмента кода **CS**;
- строки назначения в строковых инструкциях должны сохраняться в сегменте данных, на который указывает регистр **ES**;
- операции **PUSH** и **POP** должны всегда ссылаться на **SS** сегмент.

Некоторые инструкции требуют, чтобы селектор сегмента был указан явно.

В этих случаях 16-битный селектор сегмента может быть расположен в ячейке памяти или в 16-битном регистре.

Например, следующая инструкция **MOV** перемещает селектор сегмента, расположенный в регистре **BX**, в регистр сегмента **DS**:

```
MOV DS, BX
```

Селекторы сегментов также могут быть указаны явно как часть 48-битного дальнего указателя (far pointer), расположенного в памяти. В этом случае первое двойное слово в памяти содержит смещение, а следующее слово содержит селектор сегмента.

Указание смещения

Часть адреса памяти, представляющая смещение (offset), может быть указана непосредственно в виде *статического значения*, называемого смещением (displaysment) или путем вычисления адреса, состоящего из одного или нескольких компонент из следующего набора:

смещение (displaysment) — 8-, 16- или 32-битное значение.

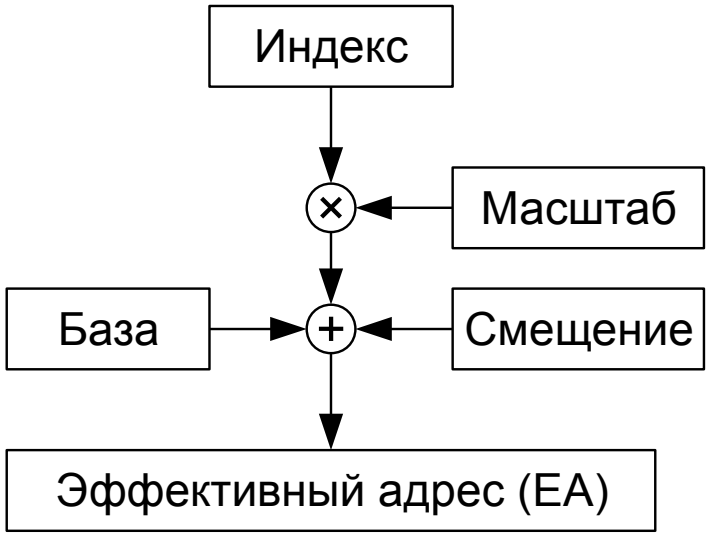
база (base) — значение в регистре общего назначения.

индекс (index) — значение в регистре общего назначения.

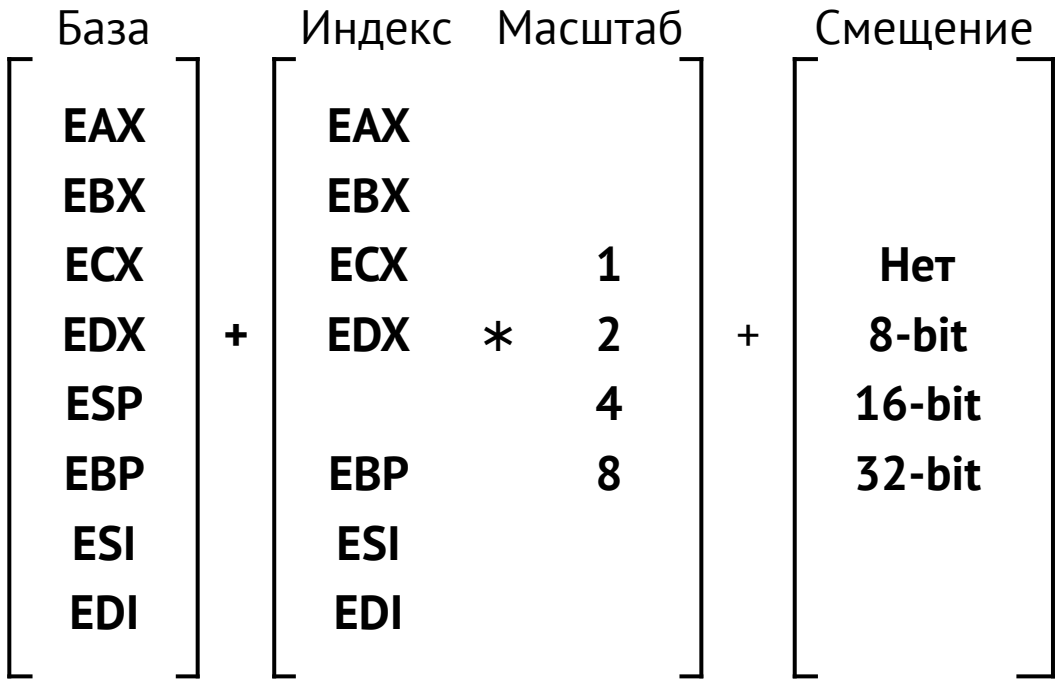
коэффициент масштабирования (scale factor) — значение 2, 4 или 8, умноженное на значение индекса.

Смещение (offset), возникающее в результате суммирования этих компонентов, называется **эффективным адресом**.

Каждый из этих компонентов может иметь как положительное, так и отрицательное значение (в виде дополнения до 2-х), за исключением коэффициента масштабирования.



На рисунке показаны все возможные способы объединения этих компонентов для создания эффективного адреса в выбранном сегменте.



8 * (1 + 7 * 4) * 4 = 928 режимов адресации

Ограничения

Использование регистров общего назначения в качестве базовых или индексных компонентов ограничено следующим образом:

- регистр **ESP** нельзя использовать в качестве индексного регистра;
- в том случае, когда регистры **ESP** или **EBP** используются в качестве базы, сегментом по умолчанию является сегмент **SS**. Во всех остальных случаях сегментом по умолчанию является сегмент **DS**;

Компоненты базы, индекса и смещения могут использоваться в любой комбинации, и любой из этих компонентов может иметь значение **NULL**;

Масштабный коэффициент может использоваться только тогда, когда используется индекс;

Любая возможная комбинация полезна для работы со структурами данных, обычно используемыми программистами на языках высокого уровня и ассемблере.

Комбинации компонент адреса

Часто встречающиеся комбинации компонент адреса можно использовать в следующих режимах адресации.

Смещение (Displacement) — смещение, присутствующее без указания остальных компонент эффективного адреса, представляет собой прямое (не вычисленное) смещение операнда. Поскольку смещение закодировано в инструкции, эту форму адреса иногда называют **абсолютным или статическим адресом**.

Обычно используется для доступа к статически размещенным скалярным операндам.

```
static int val;

void foo(int p) {
  val = p;        // 'Disp <- &val
}
```

База (Base) — база без указания остальных компонент представляет косвенное (indirect) смещение операнда. Поскольку значение в базовом регистре может изменяться, его можно использовать для динамического хранения переменных и структур данных.

База + Смещение (Base + Displacement)

Базовый регистр и смещение вместе могут использоваться в двух разных целях:

- в качестве индекса в массиве, когда размер элемента равен 1 байт (не равен 2, 4 или 8 байтов). Компонент смещения в этом случае кодирует статическое смещение начала массива. Базовый регистр содержит результаты расчетов, использующихся для определения смещения конкретного элемента в массиве.

```
char array[1024];
array[17] = 255;   // 'Disp <- &array,  'Base <- 17
```

- Для доступа к полю записи (record) — базовый регистр содержит адрес начала записи, а смещение является статическим смещением поля.

```
#include <stddef.h> // offsetof()

struct s {
  int     a;
  int    *p;
  double  x;
} record;

record.a = 123; // 'Disp <- offsetof(struct s, a), 'Base <- &record
```

База + Смещение при вызове процедуры

Важным частным случаем комбинации «База + смещение» является доступ к параметрам записи активации процедуры (procedure activation record).

Запись активации процедуры — это кадр стека, создаваемый при входе в процедуру. Поскольку регистр **EBP** автоматически выбирает сегмент стека, он является наилучшим выбором для базового регистра и обеспечивает наиболее компактный код для реализации этой часто используемой функции.

(Индекс * Масштаб) + Смещение ((Index * Scale) + Displacement)

Этот режим адресации предлагает эффективный способ индексации в статическом массиве, если размер элемента составляет 2, 4 или 8 байтов. Смещение (displacement) определяет начало массива, индексный регистр содержит индекс нужного элемента массива, и процессор автоматически преобразует индекс элемента в массиве в нужное байтовое смещение относительно начала массива, используя коэффициент масштабирования.

База + Индекс + Смещение (Base + Index + Displacement)

Совместное использование двух регистров поддерживает доступ либо к двумерному массиву (смещение содержит адрес начала массива), либо к одному из нескольких экземпляров записей в массиве (смещение является смещением поля в записи).

База + (Индекс * Масштаб) + Смещение (Base + (Index * Scale) + Displacement) — совместное использование всех компонентов адресации позволяет эффективно индексировать двумерный массив, когда элементы массива имеют размер 2, 4 или 8 байтов.

Режимы адресации ассемблера и компилятора

На уровне машинного кода выбранная комбинация смещения, базового регистра, индексного регистра и масштабного коэффициента кодируется в инструкции.

Все ассемблеры позволяют программисту использовать любую допустимую комбинацию этих компонентов адресации для адресации операндов.

Компиляторы языка высокого уровня выберут подходящую комбинацию этих компонентов на основе языковой конструкции, которую определяет программист.

Комбинации компонент адреса	masm	nasm
смещение	`table`	`[table]`
база	`[ebx]`	`[ebx]`
база + смещение	`table[ebx]`	`[table+ebx]`
(индекс * масштаб) + смещение	`dword ptr table[edi]`	`[table+4*edi]`
База + (индекс * масштаб) + смещение	`dword ptr table[edi][ebx]`	`[ebx+table+2*edi]`
Загрузка адреса	`mov eax, offset table`	`mov eax, table`
Переопределение сегментного регистра	`ES:[edi]`	`[ES:edi]`

(¤) Порты ввода/вывода

К шине может быть подключено большое число различных контроллеров внешних устройств, поэтому их необходимо различать, т.е. определять, кому какие данные, выставленные на шину, предназначены. Для этой цели вводится понятие **порта ввода-вывода (I/O Port)**.

Порт ввода-вывода представляет собой абстракцию, имеющую адрес, представимый на данной шине.

С точки зрения процессора для каждого порта возможны операции записи и чтения (Read/Write Operations), т.е. передача некоторого значения по данному адресу и запроса значения с заданного адреса.

Диапазон значений и адресов портов зависит от архитектуры шины.

За каждым из контроллеров числится один или несколько портов, а то и целая область ввода-вывода, причем некоторые из них могут быть только прочитаны, другие — только записаны.

Порт в определенной степени похож на ячейку памяти.

В некоторых системах порты ввода-вывода располагаются в том же адресном пространстве, что и ОП, а также читаются и записываются одними и теми же командами ЦПУ.

Тем не менее, порт не является ячейкой памяти, поскольку в большинстве случаев он он ничего не хранит, и зачастую значения, которые туда записываются, отличаются от значений, оттуда прочитанных, если порт допускает и чтение и запись.

Значение, записываемое в порт, может представлять собой код команды, например, включение мотора жесткого диска, а читаться из него будет текущее состояние устройства.

Каждый контроллер имеет свой протокол для взаимодействия.

Некоторые контроллеры имеют буфера ввода-вывода (I/O Buffers).

Такие буфера представляют собой оперативную память, конструктивно являющуюся частью контроллера, предназначенную для обмена массивами данных между контроллером и ЦПУ. Например, данные, которые требуется записать на диск, следует занести в буфер диска, после чего выдать команду на запись в соответствующий порт. Наоборот, при чтении данных следует выдать контроллеру команду чтения в соответствующий порт, после чего можно будет скопировать данные из буфера в ОП.

Еще один пример буфера — видеопамять.

(¤) Адресация портов ввода/вывода

Некоторые системы имеют отдельное адресное пространство для портов ввода-вывода, часто меньшей разрядности по сравнению с разрядностью адресного пространства ОП. Это вызвано тем, что для работы с адресным пространством ВВ ЦПУ используют отдельные команды (**IN, OUT**), ОП-код которых это пространство ограничивает.

В системах, где порты располагаются в адресном пространстве ОП, для их записи/чтения используются обычные команды пересылки данных, типа **MOV**.

Схема с общим адресным пространством для оперативной памяти и портов ввода/вывода позволяет предоставить возможность отобразить некоторые области портов на адресное пространство определенных процессов, позволив тем самым им самостоятельно взаимодействовать с определенными устройствами напрямую, минуя ОС. В результате появляется возможность вынести некоторые драйверы устройств из ядра на уровень пользователя.

Поскольку ОП и порты контроллеров — сущности различные, то помещение их на общую шину часто оказывается затруднительным и может потребовать отдельной шины. Часто подход является комбинированным (I/O+MEM+DATA).

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

- через отдельное адресное пространство ввода/вывода (I/O address space);
- через отображенный в памяти ввод/вывод (memory-mapped I/O).

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 обрабатывается общими командами перемещения и строковых инструкций процессоров, а защита обеспечивается посредством сегментации или разбиения на страницы.

Порты ввода-вывода могут быть отображены в адресное пространство ввода-вывода или в адресное пространство физической памяти (**ввод-вывод с отображением в памяти**) или в обоих.

Одно из преимуществ использования адресного пространства ввода/вывода состоит в том, что запись в порты ввода/вывода гарантированно будет завершена до выполнения следующей инструкции в потоке команд.

Таким образом, запись в аппаратное обеспечение системы управления приводит к тому, что аппаратное обеспечение устанавливается в новое состояние, прежде чем будут выполнены какие-либо другие инструкции.

Процессор поддерживает адресное пространство ввода/вывода, которое содержит до 65536 8-битных портов ввода/вывода. 16-битные и 32-битные порты также могут быть определены в адресном пространстве ввода-вывода.

К порту ввода/вывода можно обратиться либо с непосредственным операндом, либо со значением в регистре DX.

Форматы инструкций

Коды инструкций архитектуры IA-32 (и AMD64) являются подмножествами формата, приведенного на рисунке ниже. Инструкции состоят из:

- необязательных **префиксов** команд размером 1 байт каждый (в любом порядке);
- первичных байтов **кода операции** (opcode — 1..3 байта);
- **спецификатора формы адресации** (если требуется), состоящего из байта **ModR/M** и иногда байта **SIB** (Scale-Index-Base);
- **смещение** (размером 1, 2 или 4 байта, либо отсутствует);
- поле **непосредственных данных** (размером 1, 2 или 4 байта, либо отсутствует).

Префикс	Код операции	ModR/M	SIB	Смещение	Immediate

ModR/M —

SIB —

7 6	5 3	2 0
Mod	Reg/Opcode	R/M
Scale	Index	Base

Некоторые редкие инструкции могут принимать смещение или непосредственные данные размером 8 байт.

Префиксы команд (Instruction Prefixes)

Префикс	Код операции	ModR/M	SIB	Смещение	Immediate

Префиксы команд разделены на четыре группы, каждая с набором допустимых кодов префиксов. Для каждой инструкции полезно включать всего лишь по одному префиксному коду из каждой из четырех групп. Группы могут быть расположены в любом порядке относительно друг друга.

Группа 1 — префиксы блокирования и повторения

Префикс блокирования `LOCK` кодируется как **`F0H`**.

Префикс **LOCK (F0H)** инициирует операцию, которая обеспечивает исключительное использование совместно используемой памяти (shared memory) в многопроцессорной среде.

Префикс повторения `REPNE/REPNZ` (Repeat-Not-Zero) кодируется как **`F2H`**.

Префиксы **`REP`** или **`REPE/REPZ`** кодируются как **`F3H`**.

Префикс повторения применяется только к строкам и инструкциям ввода/вывода, а также используется в качестве обязательного префикса для инструкций **`POPCNT`**, **`LZCNT`** и **`ADOX`**.

Префиксы повторения (F2H, F3H) приводят к тому, что инструкция повторяется для каждого элемента строки. Эти префиксы используются только с строковыми инструкциями и инструкциями ввода/вывода (**`MOVS`**, **`CMPS`**, **`SCAS`**, **`LODS`**, **`STOS`**, и **`INS`**, **`OUTS`**).

Использование повторяющихся префиксов и/или неопределенных кодов операций с другими инструкциями Intel 64 или IA-32 недопустимо — это может вызвать непредсказуемое поведение.

Некоторые инструкции могут использовать префиксы повторения **`F2H`**, **`F3H`** в качестве обязательного префикса для выполнения особых функций.

Группа 2 — переопределение сегментов и подсказки для оптимизации ветвления

Префиксы переопределения сегментов:

2EH - префикс переопределения переопределение сегмента **CS**;
36H - префикс переопределения сегмента **SS**;
3EH - префикс переопределения сегмента **DS**;
26H - префикс переопределения сегмента **ES**;
64H - префикс переопределения сегмента **FS**;
65H - префикс переопределения сегмента **GS**;
Использование этих префиксов с любой инструкцией ветвления невозможно.

Префиксы подсказок ветвления:

2EH—ветвь наименее вероятна.
3EH—ветвь наиболее вероятна.
Используются только с инструкциями условных переходов **Jcc**.

Префиксы подсказок ветвления (2EH, 3EH) позволяют программе дать подсказку процессору о наиболее вероятном пути кода для ветви и используются только с инструкциями условного перехода (**Jcc**).

Другое использование префиксов подсказок ветвления и/или других неопределенных кодов операций с инструкциями AMD64 или IA-32 не допускается — такое использование может вызвать непредсказуемое поведение.

Группа 3 — префикс переопределения размера операнда

Префикс переопределения размера операнда — кодируется как 66H (66H также используется в качестве обязательного префикса для некоторых инструкций).

Префикс переопределения размера операнда (66H) позволяет программе переключаться между 16- и 32-битными размерами операнда. Любой размер может быть установлен размером операнда по умолчанию — использование префикса выбирает размер, отличный от установленного по умолчанию.

Некоторые инструкции и инструкции SSE2/SSE3/SSSE3/SSE4, использующие трехбайтовую последовательность для первичного кода операции, могут использовать 66H в качестве обязательного префикса для выполнения особых функций.

Другое использование префикса 66H недопустимо — это может вызвать непредсказуемое поведение.

Группа 4 — префикс переопределения размера адреса

Префикс переопределения размера адреса кодируется как 67H.

Префикс переопределения размера адреса (67H) позволяет программам переключаться между 16- и 32-битной адресацией. Любой размер может быть установлен размером адреса по умолчанию — использование префикса выбирает размер, отличный от установленного по умолчанию.

Использование этого префикса и/или других неопределенных кодов операций, когда операнды для инструкции не находятся в памяти, недопустимо — это может вызвать непредсказуемое поведение.

Код операции

Префикс	Код операции	ModR/M	SIB	Смещение	Immediate

Основной код операции может иметь длину 1, 2 или 3 байта.

Дополнительное 3-битное поле кода операции иногда кодируется в байте ModR/M.

В основном коде операции также могут быть определены небольшие дополнительные поля, отвечающие за направление работы инструкции, размер смещений, кодирование регистров, коды условий, расширение знака.

Поля кодирования, используемые кодом операции, различаются в зависимости от класса операции.

Двухбайтовые форматы кодов операций для команд общего назначения и SIMD состоят из одного из следующих полей:

- экранирующий байт (escape-байт) кода операции **0FH** в качестве основного кода операции и второй байт кода операции;

- обязательный префикс (66H, F2H или F3H), байт escape-кода операции и второй байт кода операции (аналогично предыдущему пункту).

Например, инструкция **CVTDQ2PD**[2] состоит из следующей последовательности:

```
F3 0F E6
```

Первый байт является обязательным префиксом (он не считается повторным префиксом).

2 Convert packed doubleword integers to packed double-precision floating-point values (SSE2 — Streaming SIMD Extensions 2).

Трехбайтовые форматы кодов операций для команд общего назначения и SIMD состоят из одного из следующих полей:

- экранирующий байт (escape-байт) кода операции **0FH** в качестве основного кода операции плюс два дополнительных байта кода операции;

- обязательный префикс (66H, F2H или F3H), байт escape-кода операции, плюс два дополнительных байта кода операции (аналогично предыдущему пункту).

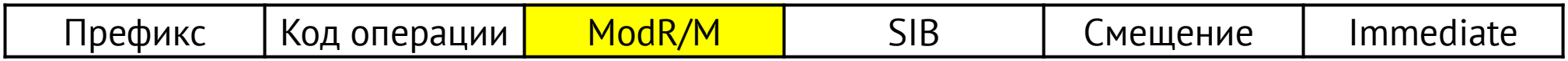
Например, **PHADDW**[3] для регистров XMM состоит из следующей последовательности:

```
66 0F 38 01
```

Первый байт является обязательным префиксом.

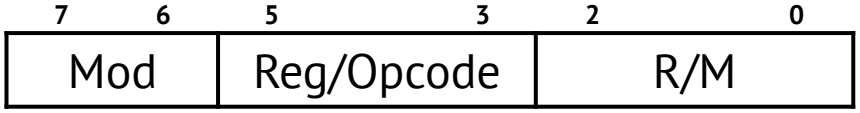
3 Adds two adjacent, signed 16-bit integers horizontally from the source and destination operands and packs the signed 16-bit results to the destination operand (SSSE3 — Supplemental Streaming SIMD Extensions 3).

ModR/M and SIB Bytes



Инструкции, которые ссылаются на операнд в памяти, имеют байт спецификатора формы адресации (называемый байтом ModR/M), располагающийся за основным кодом операции. Байт ModR/M содержит три поля информации:

ModR/M —



- поле **Mod** и поле **R/M** (2 + 3 = 5 бит) формируют 32 (2^5) возможных варианта расположения операнда — восемь регистров и 24 режима адресации.

Поле **Reg/Opcode** указывает либо номер регистра, либо еще три бита информации о коде операции. Назначение поля **Reg/Opcode** указывается в основном коде операции.

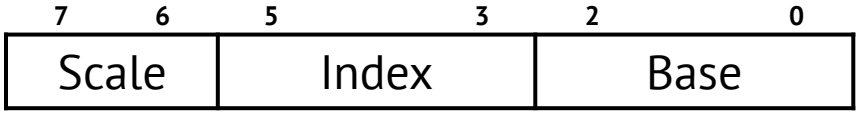
Поле **R/M** может указывать регистр в качестве операнда или комбинируется с полем **Mod** для кодирования режима адресации.

Иногда определенные комбинации поля **Mod** и поля **R/M** для некоторых инструкций используются с целью предоставления дополнительной информации относительно кода операции.

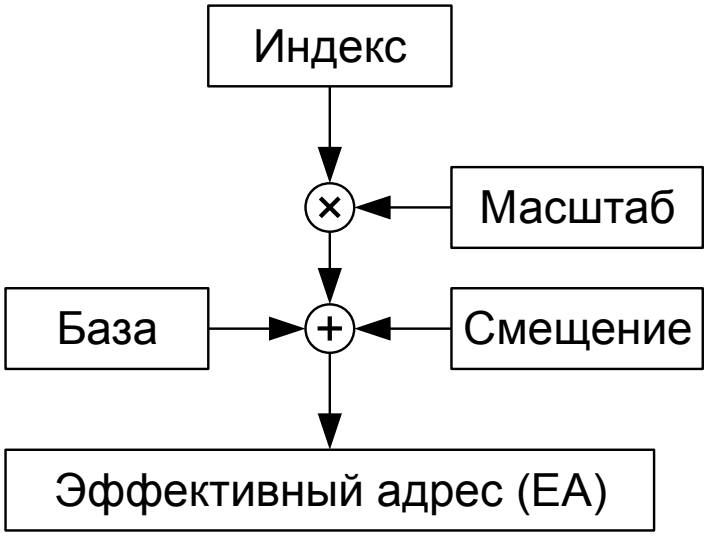
Для определенных кодов байта **ModR/M** требуется дополнительный байт адресации (байт **SIB**).

Для форм 32-битной адресации «База-плюс-Индекс» и «Масштаб-плюс-Индекс» байт **SIB** обязателен (SIB — **S**cale **I**ndex **B**ase). Байт **SIB** включает в себя следующие поля:

SIB —



Scale — поле масштаба указывает коэффициент масштабирования;
Index — индексное поле указывает номер регистра индексного регистра;
Base — базовое поле указывает номер регистра базового регистра.



Смещение и непосредственные байты

Префикс	Код операции	ModR/M	SIB	Смещение	Immediate

Некоторые формы адресации включают смещение (Displaysment), следующее непосредственно за байтом **ModR/M**, или байтом **SIB**, если таковой присутствует.

Если требуется смещение, оно может составлять 1, 2 или 4 байта.

Если инструкция указывает непосредственный операнд, этот операнд всегда следует за любыми байтами смещения. Непосредственный операнд может быть 1, 2 или 4 байта.

(¤) Кодирование режимов адресации байтами ModR/M и SIB

Значения и соответствующие формы адресации байтов **ModR/M** и **SIB** детально описаны в документе «**Intel® 64 and IA-32 Architectures Software Developer's Manual**

Volume 2 (2A, 2B & 2C): Instruction Set Reference, A-Z» в разделе «**CHAPTER 2 INSTRUCTION FORMAT**» в таблицах:

Table 2-1. 16-Bit Addressing Forms with the ModR/M Byte;

Table 2-2. 32-Bit Addressing Forms with the ModR/M Byte;

Table 2-3. 32-Bit Addressing Forms with the SIB Byte.