Лекция 3: набор команд процессора.

Вступление.

Набор команд процессора является связующим звеном между аппаратным и программным обеспечением. Сейчас часто этот набор называют архитектурой процессора, или языком ассемблера (что\в общем то неверно).
Исторически, это первый - и изначально единственный - уровень программного обеспечения. Сейчас конечно можно было бы создать аппаратное обеспечение, работающее напрямую с каким-либо языком программирования: C++ или Java; но делать этого не стоит, так как теряется возможность работать с программами, написанными на других языках, да и архитектура процессора усложняется многократно. Сейчас программы, написанные на любом языке программирования, для исполнения на процессоре проходят специальную стадию преобразования в команды процессора. Вариантов два:

- компиляция - из исходного кода однократно формируется исполняемая программа на машинном языке и в дальнейшем запускается именно она
- интерпретация - в ходе исполнения программы она транслируется “на лету” в машинный язык

Достоинства и недостатки есть у обоих методов и их рассмотрение не входит в рамки этой лекции.
При создании нового процессора перед инженерами, разрабатывающими новый набор команд. возникает один, но самый главный вопрос - программная совместимость с предыдущими процессорами. Если не будет обеспечена возможность запуска всего уже разработанного ПО (от операционной системы, до программ пользователя),то будущее у нового процессора будет только если он предоставит существенно большие возможности программистам и пользователям,чем предыдущие поколения. Набор команд разумеется оказывает большое влияние на производительность процессора, производительность эквивалентных процессоров может отличатся до 25% из-за разных наборов команд, однако совместимость это то требование, которое существенно

ограничивает развитие процессоров.
Какой набор команд можно будет считать “хорошим”?
Во-первых, набор команд должен обеспечивать не только эффективную реализацию современных требований, но иметь потенциал расширения в будущем. Предугадать требуемые в будущем команды тяжело, потому набор должен предусматривать возможность добавления новых команд.
Во-вторых,набор должен быть предельно однозначным; не должно быть неточностей в правилах для компилирования приложений с других языков программирования.

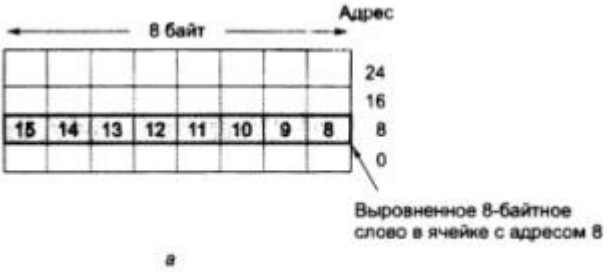
Общее для набора команд процессора.

В принципе, набор команд процессора это уровень на котором и должны создаваться программы. Именно в командах набора и предоставляются процессору для выполнения приложения. Программисты очень редко сейчас прибегают к программированию на уровне набора команд, поэтому слегка изменим определение: программа уровня архитектуры набора команд это результат работы компилятора с языка высокого уровня. Чтобы компилятор работал эффективно, создатель компилятора должен знать о аппаратной архитектуре несколько основных моментов

- модель памяти
- список регистров процессора
- типы данных
- способ адресации в памяти
- формат команды набора
- типы команд набора
- схему механизмов прерываний

Модель памяти.
В компьютерах память разделена на равные по размеру ячейки. **Размер ячейки** определяется архитектурой. В современных архитектурах размер ячейки принят равным 8 битам, то есть 1 байту. Выбор именно 8 бит обусловлен размером таблицы символов ASCII - 128 символов + бит четности. В более поздних таблицах символов. типа Unicode символы представляются группами бит,кратными 8.

Ранее встречались архитектуры с размером ячейки от 1 до 60 бит. Байты обычно группируются в **слова** по 4 или 8 байт. Архитектуры часто требуют выравнивания границ слов по очевидным границам. Например, в модели памяти с 32 битными словами, адреса конкретных слов начинаются с 0,4,8 и так далее байтов, но не с 1,3,6, 7 и аналогичных.
Выравнивание требуется для эффективной работы с памятью.



а

б

Расположение слова из 8 байт в памяти: выровненное слово (*а*); невыровненное слово (*б*). Некоторые машины требуют, чтобы слова в памяти были выровнены

Часто, исходя из аппаратных особенностей памяти, чтение “не выровненных” слов вообще невозможно. Например, интерфейс памяти DDR3 поддерживает только обращения к выровненным блокам по 8 байт, значит и процессор, использующий этот интерфейс не может обращаться к не выровненным словам.
Это требование может вызывать затруднения для программ. Например, процессоры core i7 используют интерфейс памяти ddr3, но программы на наборе команд x86, используемом в core i7, могут

обратиться к словам. начинающимся с любого байта. Процессору приходится извлекать два выровненных слова из памяти и потом выделять из них требуемое не выровненное слово. Это сильно замедляет работу с памятью. Выходов из этой ситуации видится как минимум два:

1. внесение изменения в набор x86, запрещающее обращение к не выровненным словам памяти, но это противоречит требованиям совместимости ПО;
2. внесение изменений в компилятор таким образом, что при генерировании кода он автоматически оказывается выровненным по словам, однако это приведет к перерасходу оперативной памяти и потребует поддержки со стороны ОС.

Вторым моментом, связанным с моделью памяти, является **адресное пространство памяти**. В большинстве машин память представляется в виде единого массива байтов, пронумерованных от 0 до 2^{32}, или до 2^{64}. Но в некоторых архитектурах имеются отдельные адресные пространства для программы и данных. Обращение к 8 слову в коде, и к 8 слову в данных приведет к получению разных значений. Эта система намного сложнее в реализации, но и значительно безопаснее, так как исчезает возможность случайной перезаписи кода приложения его данными.
Третий момент это **семантика памяти**. Например команда POP, вызванная сразу после команды PUSH должна вернуть только что положенное в стек значение. Однако из лекции о процессорах мы знаем о переупорядочивании команд в ходе исполнения. Выходов здесь три:

- на уровне архитектуры процессора запретить параллельное выполнение операций с памятью. Это обеспечивает корректность работы, но сильно снижает производительность
- не управлять порядком выполнения операций с памятью, однако ввести дополнительную команду (например sync), которая блокирует все операции с памятью до завершения предыдущих. Это усложняет работу компилятора
- промежуточное решение это ввести блокировку определенной последовательности операций с памятью, а

остальными запросами к памяти не управлять. Например, блокировать обращения к памяти на время операции записи в память (политика RAW -ReadAfterWrite).

Именно третий способ в основном сейчас и используется.

Список регистров процессора.
Во всех процессорах есть определенный набор ячеек внутренней памяти, именуемых регистрами. Они используются в программах. а так же служат для управления аппаратным обеспечением и прочих служебных функций. Существует два основных класса регистров

- регистры общего назначения
- специализированные регистры

Регистры общего назначения используются для хранения переменных и результатов команд. Их назначение, это доступ к используемым в текущей команде данным. Регистры общего назначения могут быть использованы в свободном порядке командами набора, то есть одна и та же команда может использовать разные регистры для хранения данных и результатов. В части наборов команд у регистров общего назначения есть рекомендованные функции, которые они исполняют в командах. Даже если регистры используются свободно, компилятор часто соблюдает определенные неформальные соглашения использования регистров, нарушать которые не рекомендуется. Например, если компилятор использует регистр R1 для хранения важной переменной. а потом вызовет функцию, которая перезапишет в ходе работы этот регистр, то значение переменной потеряется.
Специализированные регистры это регистры, обладающие какими либо выделенными функциями, например :

- счетчик команд
- указатель стека
- слово состояния процессора и так далее

Кроме этого списка регистров, доступных для программ в обычном режиме, существуют специальные регистры, доступные только в привилегированном режиме работы процессора. Эти регистры используются для управления кэш памятью, участвуют в обработке прерываний и прочее. Эти регистры доступны только специализированным модулям ядра операционной системы, за исключением одного регистра - PSW (Processor Status Word). Он содержит флаги текущего состояния процессора и условий обработки текущих команд программы. К примеру

- бит N - результат вычисления отрицателен
- бит Z - результат равен нулю
- бит V - результат вызвал переполнение
- и прочее

Так же,к примеру ,есть бит трассировки - происходит обычное исполнение приложения, или отладка.

Типы данных.

Память является хранилищем информации, но интерпретировать эту информацию можно по разному. Интерпретация этих данных начинается с определения типа данных.

Ключевым вопросом о типах данных является наличие поддержки этого типа данных в процессоре. то есть существуют ли в наборе команд инструкции. способные обрабатывать этот тип данных, или нет. Бывает, что от наличия или отсутствия поддержки какого либо типа данных зависит применимость этого процессора в той, или иной сфере. Например, процессор неподдерживающий вычисления с плавающей точкой, значительно проще и дешевле и он отлично подходит в бухгалтерию. где вычисления по большей части целочисленные. Но для научных расчётов такой набор инструкций непригоден.

Все типы данных относятся к одному из двух типов

- числовые
- нечисловые

Числовые типы данных. Среди них основными являются целые числа различной длины (8,16,32,64 бита), со знаком или без. В большинстве наборов реализована поддержка чисел с плавающей

точкой. Они могут быть 32,64 или 128 бит. Зачастую, для обработки чисел с плавающей точкой используются отдельные команды, или даже отдельный математический сопроцессор.
Нечисловые типы данных. к ним относятся

- булевые
- символьные
- строковые
- ссылки

Булевый тип требует для своего хранения 1 бит, но по архитектуре памяти выделение такого объема невозможно. поэтому чаще всего булевые значения хранятся в виде беззнакового целого числа. Исключением является битовая карта - объединение нескольких битовых переменных в одну бинарную. Почти во всех системах, 0 это ложь, а 1 истина.
Строковый тип представляет собой набор символов, заканчивающихся особым символом - терминальным. Аппаратные команды чаще всего поддерживают копирование и поиск в строке. Последний не числовой тип данных это указатели. Формально, это целое число, обозначающее адрес в памяти. Интерпретирует его как адрес именно команда из набора. которая ожидает в качестве параметра именно адрес в памяти, а не число. В этом и проявляется особенность типов данных в машинном языке - технически это всё целые беззнаковые числа, интерпретацию как другой тип данных предоставляет им именно контекст конкретной команды.

Формат команды набора.
Команда состоит из кода операции и дополнительной информации, которая может содержать данные и адрес для результата выполнения команды. Процесс определения адресов данных называется адресацией. Дополнительной информации в команде может и не быть.
В одних машинах все команды одинаковы по длине. в других могут быть разными. Если все команды одинаковы по длине и кратны длине аппаратного слова, да еще и выровнены в памяти. то это кардинальным образом упрощает декодирование и адресацию. При проектировании набора команд следует предусмотреть

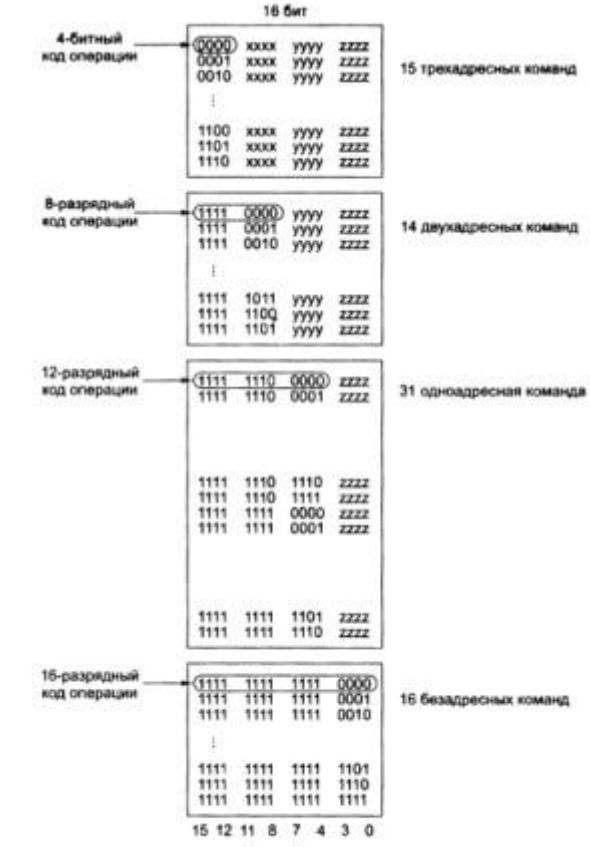
возможность расширения набора команд. иначе через пару поколений процессора набор команд придется менять и основное требование заказчиков - совместимость будет потеряно.

Четыре формата команд: безадресная команда (*а*); одноадресная команда (*б*); двухадресная команда (*в*); трехадресная команда (*г*)

Некоторые возможные отношения между размерами команды и слова

Также стоит попробовать предположить то, какова будет частота тактового генератора и время доступа к памяти через некоторое время. Часть команд критична к этим параметрам. Например, если в текущий момент и в обозримом будущем время доступа к памяти меньше. чем время одного такта генератора, то смысла закладывать защиту операций доступа к памяти немного и это сильно упростит логику процессора а значит и снизит его себестоимость.
при прочих равных, короткие команды предпочтительнее длинных. Чем короче команды, тем меньше места в оперативной памяти занимает приложение. Соответственно, снижаются затраты на доступ к памяти и улучшается работа кеша. Однако сокращение длины команды может усложнить её декодирование и параллельное выполнение.
Также возможны проблемы с операндами команды. Например, машина поддерживающая 2^n операция не может иметь длину команды меньше n. Так же и адреса операндов. Например,

принимаем, что длина любой команды не более 16 бит. Значит даже при наличии у нас в наборе всего 16 команд, под адреса операндов у нас остаётся 12 бит. Получаем, что даже при всего одном операнде, максимально возможный адресуемый объем памяти не превышает 4 kb.
Далеко не в каждой команде необходима дополнительная информация. Да и количество полей с дополнительной информацией может отличатся.



Расширение кода операции обеспечивает 15 трехадресных команд, 14 двухадресных команд, 31 одноадресную команду и 16 безадресных команд. Поля xxxx, yyyy и zzzz — это 4-разрядные адресные поля

На этом принципе основано **расширение кода операции** - можно объявлять некоторые операции признаком для отдельного интерпретирования остальных бит команды. Например, в разрабатываемом наборе команд зафиксирована длина команды в 16 бит. Известно, что нам в наборе требуется не более 15 команд с тремя операндами. Объявляем, что первые 4 бита это код команды.а далее идут три операнда по 4 бита. Коды с 0000 до 1110 резервируем под команды.а код 1111 объявляем признаком расширения. Далее нам требуется не более 14 команд с двумя параметрами. Получаем, что первые 8 бит с 1111000 до 11111101 обозначают коды команд, а оставшиеся 8 бит это два операнда. Код 11111110 это тоже расширение набора команд. С 11111110000 до 111111111110 это коды 31 команды с всег одним оператором, располагающимся в оставшихся 4 битах. Остальные коды, с 1111111111110000 до 1111111111111111 обозначают 16 команд без операндов. Таким образом, мы уместили до 66 команд с разным количеством операндов достаточно малую длину команды.

Способ адресации в памяти.
В командах набора чаще всего присутствуют операнды, адрес которых в памяти необходимо указать. Для этого и предназначена адресация в памяти. Режимы адресации

- непосредственная
- прямая
- регистровая
- косвенная регистровая
- индексная
- относительная индексная
- стековая

Непосредственная адресация представляет из себя хранение значения непосредственно в операнде команды. Таким образом чаще всего указываются константы при вычислениях. Недостатком такого типа адресации можно назвать возможность работать только с константами и не большими, чем размер поля операнда. Достоинством является отсутствие необходимости в дополнительном обращении к памяти.
Прямая адресация заключается в указании полного адреса данных в памяти в операнде команды. Недостаток в том, адрес в команде не

меняется и соответственно доступ таким образом может корректно осуществляться только к глобальным переменным.

Регистровая адресация. Аналогична прямой, но вместо фиксированного адреса памяти в качестве операнда указывается регистр процессора. Это основной способ адресации, так как чаще всего работа происходит именно с регистрами.

Косвенная регистровая адресация. При этом типе адресации обращение происходит к памяти, но адрес не фиксируется жестко, как в случае с прямой адресацией, а указывается регистр, в котором хранится адрес в памяти. Содержимое регистра при этом называется **указателем**. Такой тип адресации используется в циклах, при работе с большими типами данных, например строками и так далее.

Есть способ обойтись и без косвенной регистровой адресации - это самомодифицирующаяся программа: программа использует только прямую и регистровую адресацию, а для смены адреса в команде сначала модифицирует свой код (заменяет операнд в команде), а потом его исполняет. Является потенциально опасной технологией и не пригодна к использованию в компьютерах в кеш памятью, или раздельными адресными пространствами.

Индексная адресация - это обращение к памяти по известному смещению, относительно значения регистра. Аналогично можно использовать константу для адреса в памяти. а регистр как смещение относительно этого адреса. Используется например в циклах и при обработке строк.

Относительная индексная адресация использует соединение значений двух регистров и - опционально - константы смещения. Один из регистров называется в этом случае базой, а второй смешением. Используется при обращении к памяти, если диапазон номеров ячеек памяти больше, чем размерность регистров процессора. Таким образом, например, можно обращаться к 4 Гб памяти, имея только 16-битные регистры.

Стековая адресация. Мы отмечали, что желательно сделать машинные команды как можно короче, идеально вообще без адресов в памяти. Есть способ так сделать - хранить данные в стеке, загружая их в необходимом порядке. А далее вызвать нужную команду, которая заберет параметры из стека и туда же вернёт результат. Такой способ адресации и есть стековая адресация.

Одним из способов применения стековой адресации является вычисления при помощи обратной польской записи.

Типы команд набора.
Команды в списке могут быть условно сгруппированы по типам, которые в принципе одинаковы для всех компьютеров. Дополнительно в каждом наборе команд могут быть команды, добавленные для совместимости, или по требованию заказчика, и не подходящие под указанные ниже типы:

- команды перемещения данных
- унарные операции
- бинарные операции
- сравнения и условные переходы
- команды вызова процедур
- команды управления циклами
- команды ввода-вывода

Команды перемещения данных это одни из самых распространённых команд. Под перемещением чаще всего понимается копирование, то есть создание ещё одной копии данных по другому адресу памяти. Используются для заполнения переменных или регистров процессора.
Обычно переносится 1 машинное слово, перенос большего числа данных требует указания не только адресов источника и приемника, но и количества передаваемых данных. или признака окончания копирования. В подавляющем большинстве случаев такое копирование выполняется не аппаратной командой, а программным способом.
Унарные операции оперируют только одним операндом. Как правило это отрицание, логические сдвиги (применяются вместо умножения или деления на степени двойки. так как работают существенно быстрее,чем умножение или тем более деление), инкремент и декремент и некоторые другие.
Бинарные операции вычисляют результат на основе значений двух операндов. Могут выполнять арифметические, булевы, логические действия.
Сравнения и условные переходы. Все программы должны иметь возможность проверить некоторое условие и на основе этой проверки изменить последовательность выполнения команд. Самым

простым является проверка операнда на 0. На основе этой проверки осуществляется переход в программе по указанному адресу. Используются для создания циклов и операторов ветвления.
Команды вызова процедур. Процедурой называют группу команд, которая выполняет определенную задачу и которую требуется вызывать из разных мест программы неоднократно. При вызове процедуры (или функции) требуется запомнить номер команды.вызвавшей процедуру. чтобы потом вернутся к следующей команде. Следовательно,либо этот адрес нужно передавать самой процедуре. либо где то хранить.
Адрес возврата может помещаться либо в регистре, либо в памяти,либо в стеке. Наихудшее решение это поместить адрес в фиксированную ячейку памяти. В этом случае вызов другой процедуры внутри первой затрет адрес возврата и программа завершится досрочно.
Можно разместить адрес возврата в первой инструкции процедуры, а исполнение начинать со второй.После завершения процедуры считываем первую инструкцию и переходим по нужному адресу возврата. Вызов другой процедуры внутри первой не нарушит логику, так как у каждой процедуры своя память под адрес возврата. Логика нарушится при рекурсивном вызове процедуры.
Удачные решения это поместить адрес возврата в регистр (правда процедуре самой придётся сначала сохранить этот адрес в памяти. а в последней своей команде восстановить этот регистр) и в стек (здесь не требуется действий от процедуры, а адрес возврата всегда получается одним и тем же определенным способом - выталкиванием значения из стека).
Команды управления циклами являются необязательными и могут легко быть заменены сравнениями и условными переходами.
Команды ввода-вывода. Эта группа команд является самой сильно отличающейся группой команд в каждом конкретном наборе. В современных наборах различают три схемы ввода-вывода

- программируемый с активным ожиданием
- управляемый прерываниями
- с прямым доступом к памяти

Самый простой это **метод с активным ожиданием**. Используется в самых дешевых и простых решениях. Ввод вывод представляет из себя цикл из команды копирования и проверки флага или условия.

Проверка условия выполняется многократно в цикле, что расходует много системных ресурсов понапрасну.
Метод **ввода вывода с прерываниями** позволяет убрать активное ожидание. Вместо него происходит приостановка выполнения программы до возникновения особо сигнала от устройства ввода вывода - прерывания. Прерывание генерируется дополнительным устройством - контроллером прерываний. При возникновении прерывания исполнение программы продолжается.
Метод с прямым доступом к памяти задействует еще одно дополнительное устройство - контроллер прямого доступа к памяти. Большинство операций ввода вывода представляют из себя копирование некоторого линейного объема данных из одной области памяти в другую. Контроллер прямого доступа к памяти предназначен как раз для этой функции. Программа заносит в контроллер адрес источника,адрес приемника и количество копируемых слов. Далее приложение приостанавливается до возникновения прерывания уже от контроллера.

Схема механизмов прерываний.
Вообще под **прерыванием** понимается изменение в потоке исполнения команд, вызванное не действиями программы, а внешним фактором. При прерывании программа приостанавливается, а управление передается специальной процедуре операционной системы - **обработчику прерываний**. После завершения своих действий обработчик возвращает управление прерванной программе.
Схема действия обработчика прерываний следующая:

- приостановка и сохранение состояния текущей программы
- обращение к специальной таблице в памяти - вектору прерываний
- вызов процедуры, указанной в соответствующем поле таблицы
- по завершению процедуры восстанавливается исходное состояние и управление возвращается в прерванную программу

Понятие прерывание подробно рассматривается в курсе

операционные системы.

Особенности набора инструкций Intel X86.

Адресное пространство памяти и режимы работы процессора.

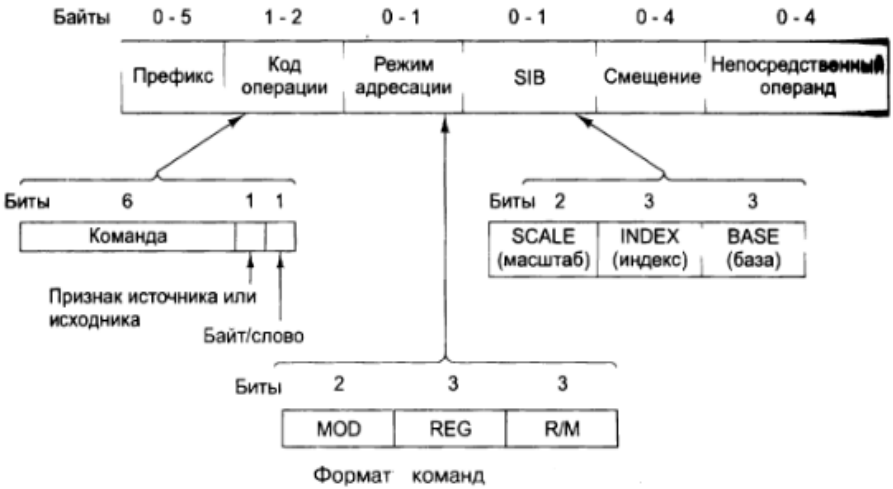
Память в Core i7 разделена на 16384 сегмента по 2^{32} байт. В 32-ух битном режиме задействуется только первый сегмент. Байты пронумерованы от 0 до 2^{32}-1. В словах биты пронумерованы справа налево, то есть самый правый адрес из 6,16,32 или 64-ёх соответствует самому младшему биту.
Процессор может работать в трёх режимах

- реальном
- виртуальном
- защищенном

В реальном режиме исполняется только набор команд процессора 8086, все остальные отключаются. так же используется только 1-ый мегабайт оперативной памяти (20-ти битная адресация памяти).
В виртуальном режиме работают тоже программы для 8086, нос защитой памяти. Используется в windows для запуска в отдельном окне старых DOS приложений.
В защищенном режиме появляются уровни исполнения приложений: с0 по 3, задаваемых в битах регистра PSW. На 0-ом уровне исполняется ядро операционной системы, а на 3-ем - остальные программы. Уровни 1 и 2 используются редко. В этом режиме уже используется полноценная схема адресации памяти - 64 бита (или 32 - в зависимости от используемого ПО).

Формат команд

Формат команд в наборе x86 очень сложен и нерегулярен. Команды содержат до 6 полей разной длины. Это произошло из исторического развития набора команд, который изначально вообще не предусматривал столь долгого и широкого развития. Из-за требований обратной совместимости уже имеющиеся команды нельзя изменить. Также из-за требований совместимости на наборе есть ряд весьма странных ограничений. Например, есть команды сложения двух регистров, сложения регистра и слова памяти (в любом порядке), но нет команды сложения двух слов памяти. Общая схема команды такова:



Формат команд

В первой версии набора все команды были однобайтовыми и для изменения их значения часто использовался префикс. С течением времени Intel исчерпала коды операций и код **0xFFh** стал **кодом расширения набора**. Отдельные биты редко что-либо говорят о команде в целом, требуется декодировать весь код операции,чтобы установить её класс и тип команды. Это сильно затрудняет и замедляет работу с набором.

После кода операции может следовать байт, определяющий режим операции. Байт разделён на поля

- MOD - 2 бита - тип адресации к параметрам. Именно то, что это поле 2 бита и определяет, что нет режима адресации из памяти в память
- REG - 3 бита
- R/M - 3 бита

Иногда первые 3 бита используются для расширения кода операции. Байт SIB используется для расширения адресации к параметрам.

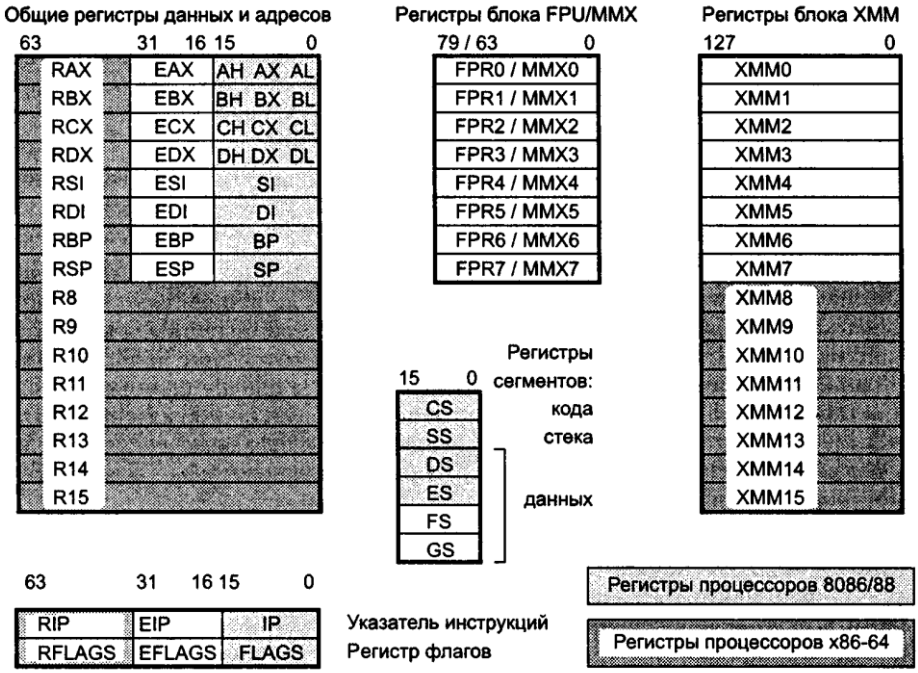
Регистры.

Первые 4 регистра это регистры общего назначения - RAX, RBX, RCX и RDX. В 8086 процессоре они были 16-ти битными, только части AX, BX,CX и DX. 32-ух битными они стали с 80386 процессора и назывались EAX, EBX,ECX,EDX . Текущая размерность регистров 64 бита.
Хотя эти регистры и называются общими, но в командах чаще всего каждому регистру предполагается определенное назначение:

- RAX (accumulator) - основной арифметический регистр
- RBX (base) - указатель, базовый адрес в памяти
- RCX (count) - счётчик в циклах
- RDX (data) - данные для команды

Эти регистры, в зависимости от типа данных и режима работы процессора, могут быть адресованы как 64-ёх битные, 32-ух битные, 16-ти битные, или как два 8-ми битных. На примере первого регистра это RAX,EAX,AX, AH и AL соответственно. Старшие 48 бит отдельно адресованы быть не могут.

Таблица с регистрами процессора Core i7 :



Следующие регистры формально тоже общие. но их специализация ещё более явная:

- ESI - указатель исходной строки в строковой операции
- EDI - указатель результирующей строки
- EBP - указатель базовой страницы данных
- ESP - указатель вершины стека

Регистры от CS до GS сегментные регистры, остатки несовершенной системы адресации процессора 8088.

- RIP - текущая команда.
- EFLAGS - флаги состояния процессора.

Типы данных.

Набор команд x86 поддерживает

- целые числа (со знаком и без)
- двоично-десятичные числа
- числа с плавающей точкой
- символы и строки

Двоично-десятичные числа являются анахронизмом и поддерживаются только в 8-ми битном режиме работы. Числа с плавающей точкой доступны через сопроцессор и потому доступны только в 32-ух битном и 64-ех битном режимах. Остальные типы данных доступны в любом режиме.

Режимы адресации к памяти.

Режимы адресации нерегулярны и зависят от разрядности команды. Рассмотрим на примере 32-ух битных команд. Возможные режимы адресации

- непосредственная
- прямая
- регистровая
- косвенная регистровая
- индексная
- специальная (для элементов массива)

Беда в том, что не все типы адресации допустимы для всех команд во всех режимах адресации. Таблица помогает определится с возможными способами адресации

32-разрядные режимы адресации процессора Core i7
(M[x] — это слово в памяти с адресом x)

	MOD			
R/M	00	01	10	11
000	M[EAX]	M[EAX + смещение 8]	M[EAX + смещение 32]	EAX или AL
001	M[ECX]	M[ECX + смещение 8]	M[ECX + смещение 32]	ECX или CL
010	M[EDX]	M[EDX + смещение 8]	M[EDX + смещение 32]	EDX или DL
011	M[EBX]	M[EBX + смещение 8]	M[EBX + смещение 32]	EBX или BL
100	SIB	SIB со смещением 8	SIB со смещением 32	ESP или AH
101	Прямая адресация	M[EBP + смещение 8]	M[EBP + смещение 32]	EBP или CH
110	M[ESI]	M[ESI + смещение 8]	M[ESI + смещение 32]	ESI или DH
111	M[EDI]	M[EDI + смещение 8]	M[EDI + смещение 32]	EDI или BH

Сочетание параметров MOD и R/M и дают возможные типы адресации.
В режиме MOD=11 выбирается один из вариантов исходя из типа операции - над словами, или над байтами.
Иногда к команде дополнительно прилагается байт SIB. Он определяет масштабный коэффициент и два регистра. При его наличии индексный регистр умножается на 1,2,4 или 8, в зависимости от значения SCALE, прибавляется к базовому регистру и - в зависимости от поля MOD - добавляется 8-ми или 32-ух битное смещение.
Всё это сложно и не всегда уместно, но продиктовано не совсем корректными решениями 20-ти летней давности и требованием обратной совместимости.