|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Как следует из названия, *пользовательскими* регистры называются потому, что программист может использовать их при написании своих программ. К этим регистрам относятся ([рис. 1](http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html#Рис_1)):   * восемь 32-битных регистров, которые могут использоваться программистами для хранения данных и адресов (их еще называют[регистрами общего назначения](http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html#регистры общего назначения) (РОН)):   + **eax/ax/ah/al**;   + **ebx/bx/bh/bl**;   + **edx/dx/dh/dl**;   + **ecx/cx/ch/cl**;   + **ebp/bp**;   + **esi/si**;   + **edi/di**;   + **esp/sp**. * [шесть регистров сегментов:](http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html#Сегментные регистры) **cs, ds, ss, es, fs, gs**; * [регистры состояния и управления](http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html#Регистры состояния и управления):   + регистр флагов **eflags/flags**;   + регистр указателя команды **eip/ip**.   http://www.kolasc.net.ru/cdo/programmes/assembler/pic/ris2_1.gif  ***Рис. 1.****Пользовательские регистры микропроцессоров i486 и Pentium*  Почему многие из этих регистров приведены с наклонной разделительной чертой?  Нет, это не разные регистры — это части одного большого 32-разрядного регистра. Их можно использовать в программе как отдельные объекты.  Так сделано для обеспечения работоспособности программ, написанных для младших 16-разрядных моделей микропроцессоров фирмы Intel, начиная с i8086.  Микропроцессоры i486 и Pentium имеют в основном 32-разрядные регистры. Их количество, за исключением сегментных регистров, такое же, как и у i8086, но размерность больше, что и отражено в их обозначениях — они имеют  приставку **e** (***E****xtended*).  Разберемся подробнее с составом и назначением пользовательских регистров.  **Регистры общего назначения**  Все регистры этой группы позволяют обращаться к своим  “младшим” частям (см. [рис. 1](http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html#Рис_1)).  Рассматривая этот рисунок, заметьте, что использовать для самостоятельной адресации можно только младшие 16 и 8-битные части этих регистров. Старшие 16 бит этих регистров как самостоятельные объекты недоступны. Это сделано, как мы отметили выше, для совместимости с младшими 16-разрядными моделями микропроцессоров фирмы Intel.  Перечислим регистры, относящиеся к группе регистров общего назначения. Так как эти регистры физически находятся в микропроцессоре внутри арифметико-логического устройства (АЛУ), то их еще называют *регистрами АЛУ*:   * **eax/ax/ah/al** (Accumulator register) — *аккумулятор*.  Применяется для хранения промежуточных данных. В некоторых командах использование этого регистра обязательно; * **ebx/bx/bh/bl** (Base register) — *базовый* регистр.  Применяется для хранения базового адреса некоторого объекта в памяти; * **ecx/cx/ch/cl** (Count register) — *регистр-счетчик*.  Применяется в командах, производящих некоторые повторяющиеся действия. Его использование зачастую неявно и скрыто в алгоритме работы соответствующей команды.  К примеру, команда организации цикла **loop** кроме передачи управления команде, находящейся по некоторому адресу, анализирует и уменьшает на единицу значение регистра *ecx/cx*; * **edx/dx/dh/dl** (Data register) — регистр *данных*.  Так же, как и регистр *eax/ax/ah/al*, он хранит промежуточные данные. В некоторых командах его использование обязательно; для некоторых команд это происходит неявно.   Следующие два регистра используются для поддержки так называемых цепочечных операций, то есть операций, производящих последовательную обработку цепочек элементов, каждый из которых может иметь длину 32, 16 или 8 бит:   * **esi/si** (Source Index register) — *индекс источника*.  Этот регистр в цепочечных операциях содержит текущий адрес элемента в цепочке-источнике; * **edi/di** (Destination Index register) — *индекс приемника* (получателя).  Этот регистр в цепочечных операциях содержит текущий адрес в цепочке-приемнике.   В архитектуре микропроцессора на программно-аппаратном уровне поддерживается такая структура данных, как ***стек***. Для работы со стеком в системе команд микропроцессора есть специальные команды, а в программной модели микропроцессора для этого существуют специальные регистры:   * **esp/sp** (Stack Pointer register) — регистр *указателя стека*.  Содержит указатель вершины стека в текущем сегменте стека. * **ebp/bp** (Base Pointer register) — регистр *указателя базы кадра стека*.  Предназначен для организации произвольного доступа к данным внутри стека.   Не спешите пугаться столь жесткого функционального назначения регистров АЛУ. На самом деле, большинство из них могут использоваться при программировании для хранения операндов практически в любых сочетаниях. Но, как мы отметили выше, некоторые команды используют фиксированные регистры для выполнения своих действий. Это нужно обязательно учитывать.  Использование жесткого закрепления регистров для некоторых команд позволяет более компактно кодировать их машинное представление. Знание этих особенностей позволит вам при необходимости хотя бы на несколько байт сэкономить память, занимаемую кодом программы.  **Сегментные регистры**  В программной модели микропроцессора имеется шесть сегментных регистров:*cs, ss, ds, es, gs, fs*.  Их существование обусловлено спецификой организации и использования оперативной памяти микропроцессорами Intel. Она заключается в том, что микропроцессор аппаратно поддерживает структурную организацию программы в виде трех частей, называемых *сегментами*. Соответственно, такая организация памяти называется ***сегментной***.  Для того чтобы указать на сегменты, к которым программа имеет доступ в конкретный момент времени, и предназначены *сегментные регистры*. Фактически, с небольшой поправкой, как мы увидим далее, в этих регистрах содержатся адреса памяти с которых начинаются соответствующие сегменты. Логика обработки машинной команды построена так, что при выборке команды, доступе к данным программы или к стеку неявно используются адреса во вполне определенных сегментных регистрах. Микропроцессор поддерживает следующие типы сегментов:   1. ***Сегмент кода***. Содержит команды программы.  Для доступа к этому сегменту служит регистр **cs** (code segment register) —*сегментный регистр кода*. Он содержит адрес сегмента с машинными командами, к которому имеет доступ микропроцессор (то есть эти команды загружаются в конвейер микропроцессора). 2. ***Сегмент данных***. Содержит обрабатываемые программой данные.  Для доступа к этому сегменту служит регистр **ds** (data segment register) —*сегментный регистр данных*, который хранит адрес сегмента данных текущей программы. 3. ***Сегмент стека***. Этот сегмент представляет собой область памяти, называемую *стеком*.  Работу со стеком микропроцессор организует по следующему принципу:*последний записанный в эту область элемент выбирается первым*. Для доступа к этому сегменту служит регистр **ss** (stack segment register) —*сегментный регистр стека*, содержащий адрес сегмента стека. 4. ***Дополнительный сегмент данных***.  Неявно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в сегментном регистре *ds*.  Если программе недостаточно одного сегмента данных, то она имеет возможность использовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в сегментном регистре *ds*, при использовании дополнительных сегментов данных их адреса требуется указывать явно с помощью специальных *префиксов переопределения сегментов* в команде.  Адреса дополнительных сегментов данных должны содержаться в регистрах **es, gs, fs** (extension data segment registers).   **Регистры состояния и управления**  В микропроцессор включены несколько регистров (см. [рис. 1](http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html" \l "Рис_1)), которые постоянно содержат информацию о состоянии как самого микропроцессора, так и программы, команды которой в данный момент загружены на конвейер. К этим регистрам относятся:   * регистр флагов **eflags/flags**; * регистр указателя команды **[eip/ip](http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html" \l "EipIp)**.   Используя эти регистры, можно получать информацию о результатах выполнения команд и влиять на состояние самого микропроцессора. Рассмотрим подробнее назначение и содержимое этих регистров:  ***eflags/flags*** (flag register) — регистр *флагов*. Разрядность *eflags/flags* — 32/16 бит. Отдельные биты данного регистра имеют определенное функциональное назначение и называются флагами. Младшая часть этого регистра полностью аналогична регистру *flags* для i8086. На [рис. 2](http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html#Рис_2) показано содержимое регистра*eflags*.  http://www.kolasc.net.ru/cdo/programmes/assembler/pic/ris2_2.gif  ***Рис. 2.****Содержимое регистра eflags*  Исходя из особенностей использования, флаги регистра *eflags/flags* можно разделить на три группы:   * *8 флагов состояния*. Эти флаги могут изменяться после выполнения машинных команд.  *Флаги состояния*регистра *eflags* отражают особенности результата исполнения арифметических или логических операций. Это дает возможность анализировать состояние вычислительного процесса и реагировать на него с помощью команд условных переходов и вызовов подпрограмм. В [табл. 1](http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html" \l "Табл. 1) приведены флаги состояния и указано их назначение; * *1 флаг управления*. Обозначается **df** (Directory Flag).  Он находится в 10-м бите регистра *eflags* и используется цепочечными командами. Значение флага *df* определяет направление поэлементной обработки в этих операциях: от начала строки к концу (*df* = 0) либо наоборот, от конца строки к ее началу (*df* = 1).  Для работы с флагом *df* существуют специальные команды: cld (снять флаг *df*) и std (установить флаг *df*).  Применение этих команд позволяет привести флаг *df* в соответствие с алгоритмом и обеспечить автоматическое увеличение или уменьшение счетчиков при выполнении операций со строками; * *5 системных флагов*, управляющих вводом/выводом, маскируемыми прерываниями, отладкой, переключением между задачами и виртуальным режимом 8086.  Прикладным программам не рекомендуется модифицировать без необходимости эти флаги, так как в большинстве случаев это приведет к прерыванию работы программы. В [табл. 2](http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html#Табл. 2) перечислены системные флаги, их назначение.   **Таблица 1. Флаги состояния**   |  |  |  |  | | --- | --- | --- | --- | | Мнемоника флага | Флаг | Номер бита в*eflags* | Содержание и назначение | | cf | Флаг переноса (Carry Flag) | 0 | 1 — арифметическая операция произвела перенос из старшего бита результата. Старшим является 7, 15 или 31-й бит в зависимости от размерности операнда;  0 — переноса не было | | pf | Флаг паритета (Parity Flag) | 2 | 1 — 8 младших разрядов (этот флаг — только для 8 младших разрядов операнда любого размера) результата содержат четное число единиц;  0 — 8 младших разрядов результата содержат нечетное число единиц | | af | Вспомогательный флаг переноса (Auxiliary carry Flag) | 4 | Только для команд работающих с BCD-числами. Фиксирует факт заема из младшей тетрады результата:  1 — в результате операции сложения был произведен перенос из разряда 3 в старший разряд или при вычитании был заем в разряд 3 младшей тетрады из значения в старшей тетраде;  0 — переносов и заемов в(из) 3 разряд(а) младшей тетрады результата не было | | zf | Флаг нуля (Zero Flag) | 6 | 1 — результат нулевой;  0 — результат ненулевой | | sf | Флаг знака (Sign Flag) | 7 | Отражает состояние старшего бита результата (биты 7, 15 или 31 для 8, 16 или 32-разрядных операндов соответственно):  1 — старший бит результата равен 1;  0 — старший бит результата равен 0 | | of | Флаг переполнения (Overflow Flag) | 11 | Флаг of используется для фиксирования факта потери значащего бита при арифметических операциях:  1 — в результате операции происходит перенос (заем) в(из) старшего, знакового бита результата (биты 7, 15 или 31 для 8, 16 или 32-разрядных операндов соответственно);  0 — в результате операции не происходит переноса (заема) в(из) старшего, знакового бита результата | | iopl | Уровень Привилегий ввода-вывода (Input/Output Privilege Level) | 12, 13 | Используется в защищенном режиме работы микропроцессора для контроля доступа к командам ввода-вывода в зависимости от привилегированности задачи | | nt | флажок вложенности задачи (Nested Task) | 14 | Используется в защищенном режиме работы микропроцессора для фиксации того факта, что одна задача вложена в другую |   **Таблица 2. Системные флаги**   |  |  |  |  | | --- | --- | --- | --- | | Мнемоника флага | Флаг | Номер бита в*eflags* | Содержание и назначение | | tf | Флаг трассировки (Trace Flag) | 8 | Предназначен для организации пошаговой работы микропроцессора.  1 — микропроцессор генерирует прерывание с номером 1 после выполнения каждой машинной команды. Может использоваться при отладке программ, в частности отладчиками;  0 — обычная работа | | if | Флаг прерывания (Interrupt enable Flag) | 9 | Предназначен для разрешения или запрещения (маскирования) аппаратных прерываний (прерываний по входу INTR).  1 — аппаратные прерывания разрешены;  0 — аппаратные прерывания запрещены | | rf | Флаг возобновления (Resume Flag) | 16 | Используется при обработке прерываний от регистров отладки. | | vm | Флаг виртуального (Virtual 8086 Mode) | 17 | Признак работы микропроцессора в режиме виртуального 8086.  1 — процессор работает в режиме виртуального 8086;  0 — процессор работает в реальном или защищенном режиме | | ac | Флаг контроля выравнивания (Alignment Check) | 18 | Предназначен для разрешения контроля выравнивания при обращениях к памяти. Используется совместно с битом am в системном регистре cr0. К примеру, Pentium разрешает размещать команды и данные с любого адреса. Если требуется контролировать выравнивание данных и команд по адресам кратным 2 или 4, то установка данных битов приведет к тому, что все обращения по некратным адресам будут возбуждать исключительную ситуацию |   ***eip/ip*** (Instraction Pointer register) — регистр*-указатель команд*.  Регистр *eip/ip* имеет разрядность 32/16 бит и содержит смещение следующей подлежащей выполнению команды относительно содержимого сегментного регистра cs в текущем сегменте команд. Этот регистр непосредственно недоступен программисту, но загрузка и изменение его значения производятся различными командами управления, к которым относятся команды условных и безусловных переходов, вызова процедур и возврата из процедур. Возникновение прерываний также приводит к модификации регистра *eip/ip*. |
|  |

### Регистры

[Регистр](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D0%B8%D1%81%D1%82%D1%80_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%B0) — это небольшой объем очень быстрой памяти, размещённой на процессоре. Он предназначен для хранения результатов промежуточных вычислений, а также некоторой информации для управления работой процессора. Так как регистры размещены непосредственно на процессоре, доступ к данным, хранящимся в них, намного быстрее доступа к данным в [оперативной памяти](http://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C).

Все регистры можно разделить на две группы: пользовательские и системные. Пользовательские регистры используются при написании «обычных» программ.

В микропроцессоре стек работает похожим образом: это область памяти, у которой определена вершина (на неё указывает %esp). Поместить новый элемент можно только на вершину стека, при этом новый элемент становится вершиной. Достать из стека можно только верхний элемент, при этом вершиной становится следующий элемент.