Факультет компьютерных наук

А. С. Коваль,

А.В. Сычев

Архитектура ЭВМ и систем

Учебно-методическое
пособие для вузов

Воронеж 2007

Утверждено научно-методическим советом факультета компьютерных наук, 27 декабря 2006г., протокол N5.

Рецензент: доц. кафедры цифровых технологий ФКН, Кургалин С.Д.

Учебно-методическое пособие подготовлено на кафедре информационных систем факультета компьютерных наук Воронежского государственного университета.

Рекомендовано для проведения лабораторных занятий по предметам «Архитектура ЭВМ и систем», «Архитектура ЭВМ и системное ПО» со студентами 1-го курса дневного отделения факультета компьютерных наук.

Для специальностей: 230201 (071900) - информационные системы и технологии и 230200 (654700) – информационные системы (бакалавр)

Введение

Целью проведения практических занятий по курсу «Архитектура ЭВМ» является изучение основ организации и архитектуры ЭВМ на примере двух процессорных семейств: *ставшего для многих учебных изданий классическим* семейства *PDP-11* и *наиболее распространенного сейчас* семейства процессоров *Intel x86*. Такой подход, на наш взгляд, позволяет, с одной стороны, дать прочные базовые знания фон-неймановской архитектуры, с другой стороны, не обойти вниманием популярную для построения персональных компьютеров серию x86, на которых студенту придется работать в течение учебных семестров, выполнять курсовые и дипломные работы. Кроме того, изучение двух различных подходов дает возможность сравнения, несомненно, полезную в методическом плане.

Во время практических занятий студент использует для выполнения большинства заданий программную модель PDP11 в среде операционной системы Windows 2000 и программу Turbo Debugger фирмы Borland при изучении процессоров Intel x86.

1 Модель PDP11

Модель PDP11, написанная на языке Си (MS Visual C++), выполняется в среде ОС Windows 9x/NT/2000 и имеет следующие технические данные:

- Система счисления для чисел и команд - двоичная.
- Разрядность для чисел и команд - 16 двоичных разрядов.
- Объем адресуемой оперативной памяти - 32К 16-разрядных слов.
- Число регистров общего назначения - 8.
- Система команд: безадресная, одноадресная, двухадресная.
- Виды адресации: регистровая, косвенно-регистровая, автоинкрементная, косвенно-автоинкрементная, автодекрементная, косвенно-автодекрементная, индексная и косвенно-индексная.
- Обработка внешних и внутренних прерываний выполняется с помощью памяти магазинного типа (стека).

Структурная схема учебной ЭВМ представлена на Рис. 1.1.

1.1 Регистры общего назначения

Модуль центрального процессора учебной ЭВМ содержит 16-разрядные регистры общего назначения (РОН), используемые для выборки операндов и записи результатов при выполнении арифметико-логических операций аналогично ячейкам памяти и регистрам внешних устройств.

Два из восьми имеющихся регистров общего назначения R0 - R7 имеют, кроме того, специальное назначение. Регистр R6 - *Указатель Стека* (УС) (Stack Pointer - SP) содержит адрес последней заполненной ячейки стека. Регистр R7 служит *Счетчиком Команд* (СК) (Program Counter - PC) и со-

держит адрес ячейки памяти, из которой процессор выбирает очередную команду для выполнения. В связи с этим обычно этот регистр (R7) используется только для адресации и не используется как накопительный регистр для хранения операндов. Процессор работает таким образом, что после выборки из памяти ЭВМ команды по указанному в R7 (PC) адресу содержимое этого регистра, т.е. адрес очередной команды, автоматически увеличивается на два. Благодаря этому после выполнения очередной команды в регистре R7 будет находиться адрес следующей по порядку команды.

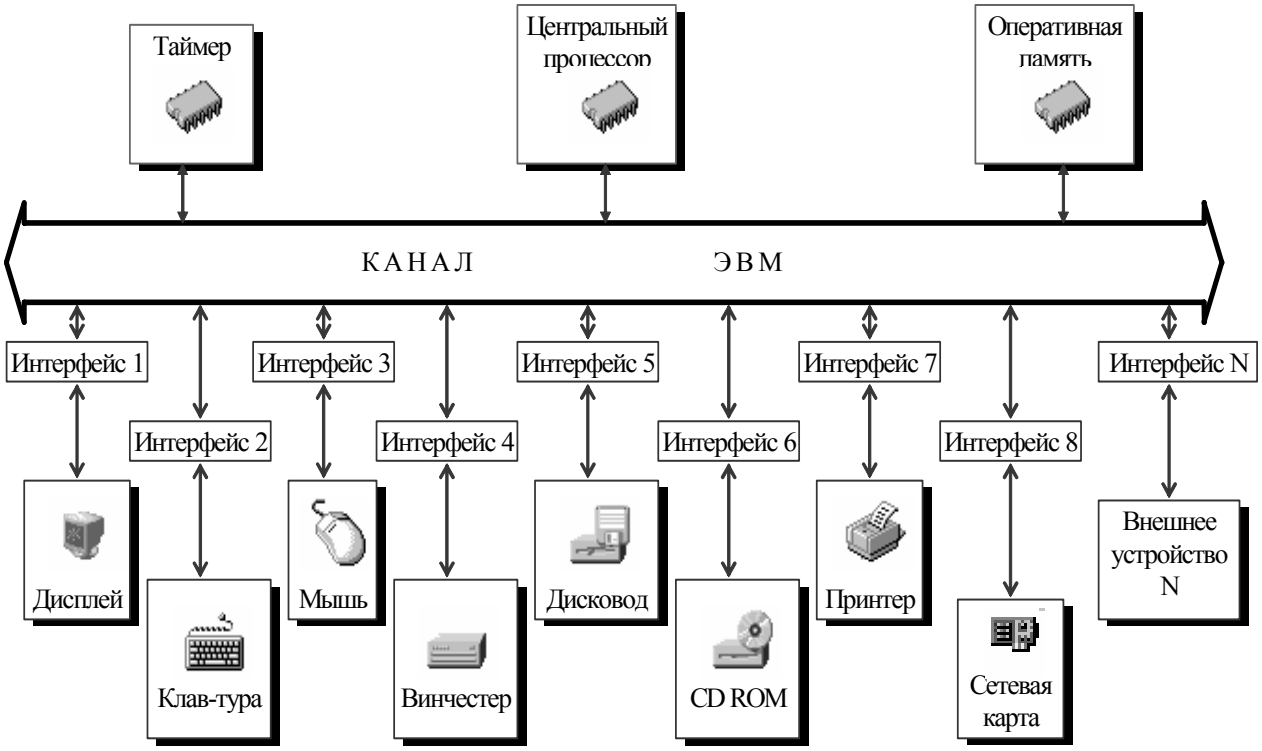


Рис. 1.1 Структурная схема учебной ЭВМ

1.2 Регистр состояния процессора (РСП)

Регистр состояния процессора содержит информацию о текущем состоянии процессора. Это информация (слово состояния процессора - ССП) о значении разрядов (кодов) условий ветвления программы, зависящих от результата выполнения команды, текущем приоритете процессора и др.

Рис. 1.2 Формат слова состояния процессора

На Рис. 1.2 показан формат регистра состояния процессора. Разряд при-

оритета процессора (7-ой разряд РСП) может находиться в состоянии "0" или "1". В последнем случае внешние устройства не могут вызывать прерывания текущей программы. Для удовлетворения требований прерывания программы седьмой разряд регистра состояния процессора должен быть равен "0".

Коды условий ветвления (разряды 0, 1, 2, 3) содержат информацию о результате последней выполненной процессором команды:

Z=1 результат операции равен нулю

N=1 результат отрицателен

C=1 признак переноса из самого старшего разряда или в ситуации, когда при сдвиге вправо из самого младшего разряда была выдвинута единица

V=1 признак арифметического переполнения

1.3 Обращение к памяти и распределение адресов канала

Весь обмен информацией и управляющими сигналами между различными устройствами и ЭВМ осуществляется через единый канал передачи информации. 16-разрядный код адреса позволяет обращаться к 32К 16-разрядных ячеек. Старшие 4К адресов (28К-32К) отведены под регистры внешних устройств. Распределение адресов канала показано на Рис. 1.3. Все адреса даны в восьмеричном коде. Буква "К" используется для обозначения числа, равного $2^{10} = 1024$.

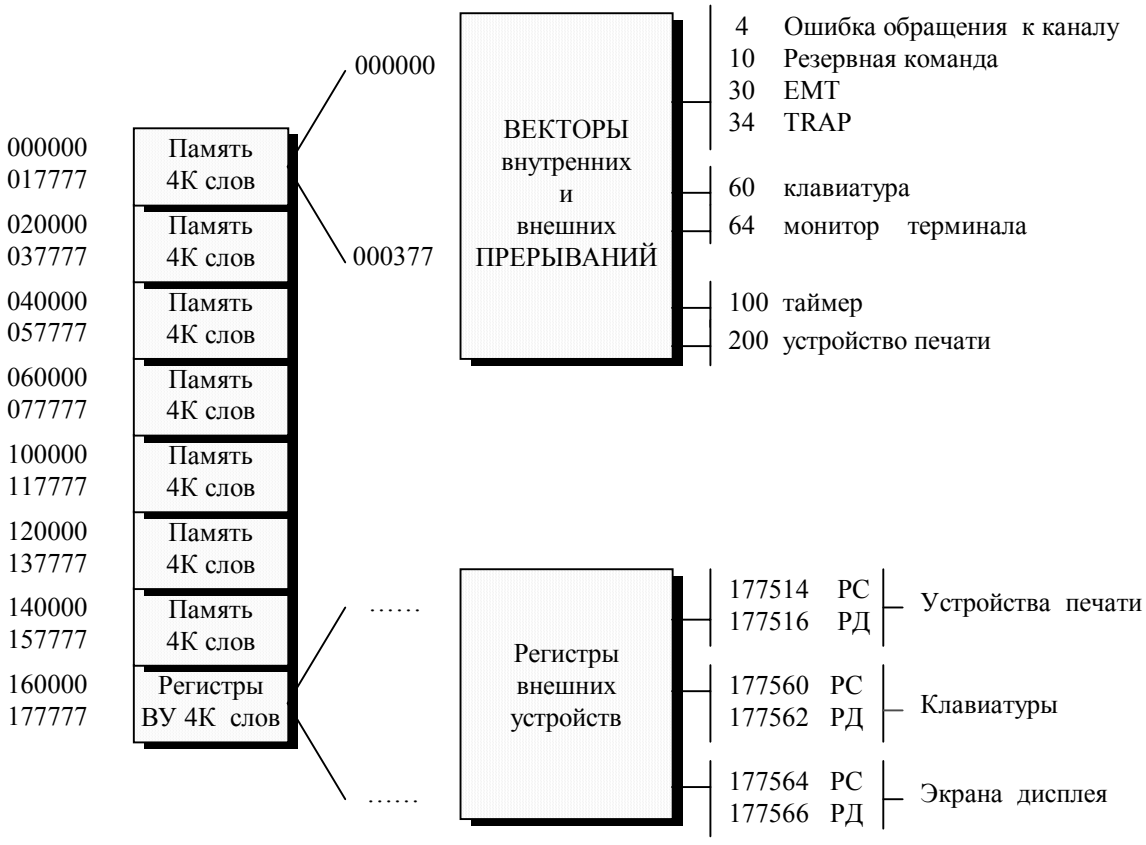


Рис. 1.3 Распределение адресов канала

Канал ЭВМ позволяет адресоваться к 32К 16-разрядных слов или к 64К байт. Ячейки памяти с 000000 по 000376 зарезервированы под векторы прерывания и использовать их для других целей не рекомендуется. Для каждого вектора необходимы две 16-разрядные ячейки, поэтому адреса векторов прерываний являются четными и заканчиваются на 0 или на 4. Последние 4К 16-разрядных адресов обычно отводятся для регистров внешних устройств, поэтому максимальный объем реальной памяти равен 28К 16-разрядных слов. Однако пользователь не обязан использовать все адреса этого пространства для этой цели и может руководствоваться соображениями необходимости.

Как показано на Рис. 1.4, машинное 16-разрядное слово делится на старший и младший байты. Ячейки, содержащие полные слова, всегда имеют четные адреса. Младшие байты слов хранятся в ячейках с четными адресами, а старшие с нечетными (Рис. 1.4).

Рис. 1.4 Формат слова учебной ЭВМ

СЛОВО		**адреса**	**БАЙТ**	**адреса**
байт	**байт**			
Старший	Младший	000000	Младший	000000
Старший	Младший	000002	Старший	000001
Старший	Младший	000004	Младший	000002
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
Старший	Младший	017770	Младший	017774
Старший	Младший	017772	Старший	017775
Старший	Младший	017774	Младший	017776
Старший	Младший	017776	Старший	017777

Рис. 1.5 Организация памяти ЭВМ по словам и по байтам для первых 4К адресов

1.4 Обмен данными между внешними устройствами и ЭВМ

Канал ЭВМ обеспечивает три типа обмена данными - это программный обмен, обмен в режиме прямого доступа к памяти и обмен в режиме прерывания программы. Обмен информацией между центральным процессором и внешними устройствами выполняется при помощи стандартных циклов обращения к каналу. Для организации обмена каждое внешнее устройство должно иметь один или несколько регистров (регистры данных, регистры состояния и др.), адреса которых определяет пользователь.

Как правило, регистры внешних устройств имеют четные адреса, однако при помощи байтовых команд можно обращаться к любому байту 16-разрядного регистра. Каждое внешнее устройство может иметь несколько различных регистров.

Регистр состояния (РС) содержит информацию об операции, выполняемой внешним устройством, характеризует состояние внешнего устройства и участвует в операциях по предоставлению прерывания.

Регистр данных (РД) используется при обмене данными между центральным процессором и внешним устройством.

Различные разряды регистров внешних устройств могут выполнять различные функции. Некоторые из них могут использоваться как для записи, так и для считывания информации, другие - только для записи или только для считывания. Типичным примером разряда, используемого для считывания и для записи, является разряд разрешения прерывания в регистре состояния внешнего устройства. Примером разряда, только принимающего информацию, является разряд пуска, а разрядом, используемым только для считывания, разряд ошибки регистра состояния внешнего устройства. Регистры данных внешних устройств, как правило, являются обычными накопительными регистрами, и их формат определяется только требованиями пользователя. Формат регистров состояния внешних устройств показан на Рис. 1.6. Данный формат не является обязательным, но желательным для обеспечения унификации операций, выполняемых при обращении к внешним устройствам. Заметим, что регистры состояния большинства внешних устройств имеют меньше 16 разрядов.

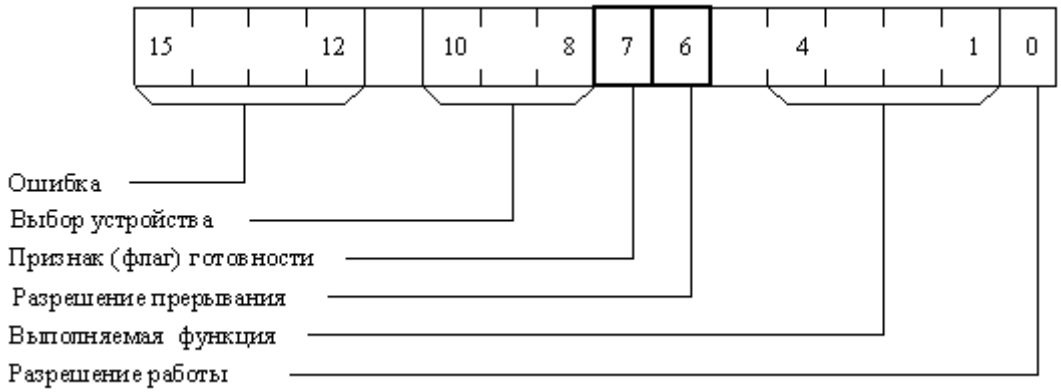


Рис. 1.6 Формат регистра состояния внешнего устройства

1.5 Система команд учебной ЭВМ и методы адресации

1.5.1 Общие понятия

Команды ЭВМ, предназначенные для обработки данных, помимо кода выполняемой операции должны тем или иным образом указывать местонахождение (адрес) этих данных (операндов) в памяти ЭВМ. В связи с этим большое значение имеют реализованные в конкретной ЭВМ методы

адресации операндов, т.е. способы указания в машинной команде местонахождения операндов в памяти ЭВМ.

Способы адресации можно классифицировать на *прямые* и *косвенные*. При прямом способе адресации исполнительный адрес берется непосредственно из команды или вычисляется с использованием значения указанного в команде и содержимого какого-либо регистра.

Косвенный способ адресации предполагает, что в команде содержится значение косвенного адреса , т.е. адреса ячейки памяти, в которой находится *окончательный исполнительный* адрес.

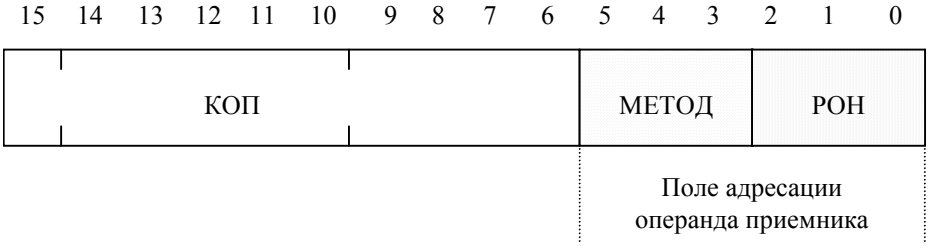
При реализации методов адресации ЭВМ существенным образом используются *регистры центрального процессора* (РОН). Далее мы будем использовать термин адресный регистр для обозначения любого регистра центрального процессора, содержащего адрес.

1.5.2 Формат команд обработки данных в учебной ЭВМ

В командах обработки данных может быть указано местонахождение от одного до нескольких операндов, используемых при выполнении конкретной операции. В учебной ЭВМ используются одноадресные и двухадресные команды. При этом обычно различают операнд - *источник* и операнд - *приемник*. Операнд - источник это содержимое ячейки памяти или регистра, которое используется при выполнении указанной в команде операции и которое в процессе выполнения команды не изменяется. Операнд - приемник это ячейка памяти или РОН, содержимое которых также может быть использовано при выполнении команды и в которые помещается результат выполненной операции (приемник результата). Ниже в приведенных примерах адресат - источник обозначается буквами src или S (source - источник), а операнд - приемник dst или D (destination - приемник). Поле команды, содержащее код операции, будет обозначаться аббревиатурой КОП.

1.5.3 Формат одноадресных команд

Формат одноадресных команд (HALT, CLR …) имеет следующий вид:



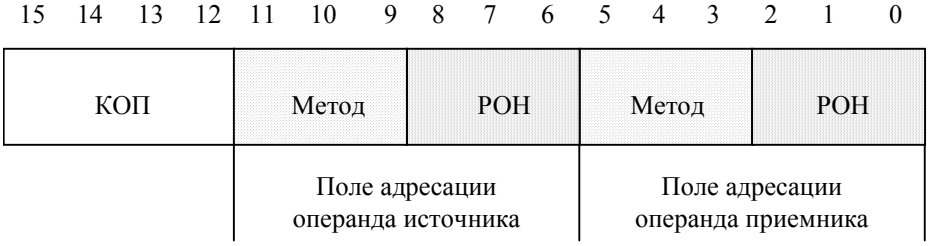
Разряды 15 - 06 содержат код операции, который определяет выполняемую команду. Разряды 05 - 00 образуют шестиразрядное поле, именуемое полем адресации операнда приемника, которое в свою очередь состоит из двух подполей:

1) Разряды 02 - 00 определяют один из восьми РОН, который использует данная команда;

2) Разряды 05 - 03 определяют способ использования выбранного регистра (метод адресации). Причем, разряд 03 определяет прямую или косвенную адресации.

1.5.4 Формат двухадресных команд

Операции над двумя операндами (такие, как сложение, пересылка, сравнение) выполняются с помощью команд, в которых задаются два адреса. Задание разрядов в полях адресации операндов источника и приемника определяют используемые методы адресации и регистры общего назначения. Формат двухадресной команды имеет следующий вид:

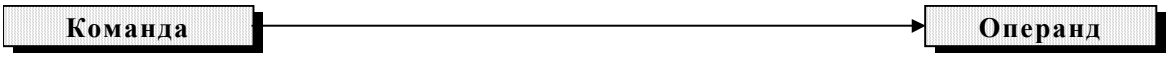


Поле адресации операнда источника используется для выборки операнда источника. Поле адресации операнда приемника используется для выборки операнда приемника и занесения результата. Например, по команде **ADD A,B** содержимое ячейки "A" (операнда источника) складывается с содержимым ячейки "B" (операнд приемника). После выполнения операции сложения в ячейке "B" будет находиться результат операции, а содержимое ячейки "A" не изменится.

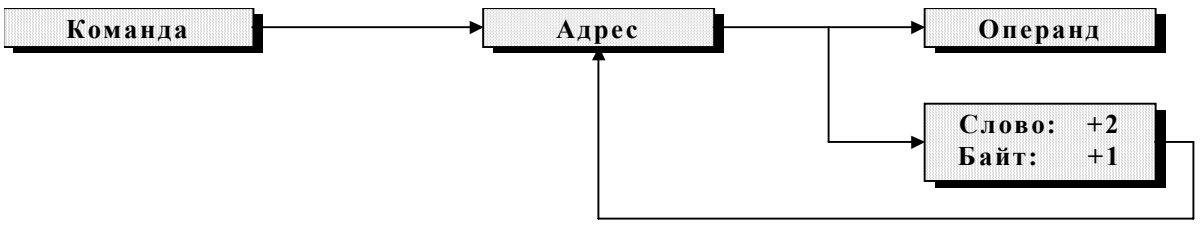
1.5.5 Методы прямой адресации

На Рис. 1.7 показаны последовательности операций при выполнении команд с каждым из четырех методов прямой адресации. При регистровом методе адресации операнд находится в выбранном регистре, который может быть использован как накопитель. Так как РОН аппаратно реализованы в ИС центрального процессора, они обладают более высоким быстродействием, чем любая другая память, работающая под управлением процессора. Это их преимущество особенно проявляется при операциях с переменными, к которым необходимо часто обращаться.

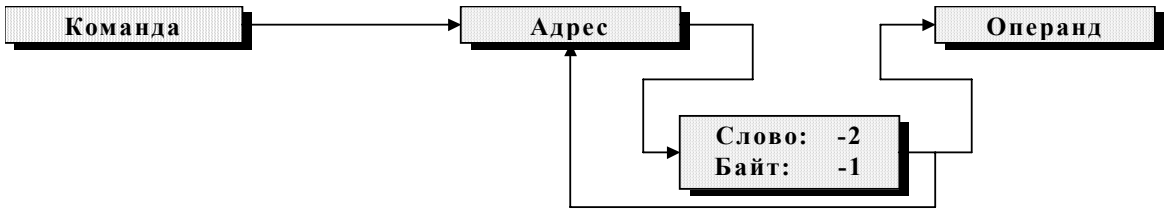
1. ***Регистровый метод адресации*** (обозначение **R**, код $\mathbf{0}_8$ - $\mathbf{000}_{2)}$.



2. ***Автоинкрементный метод адресации*** (обозначение **(R)+** , код $\mathbf{2}_8$ - $\mathbf{010}_2$).



3. ***Автодекрементный метод адресации*** (обозначение **-(R)** , код $\mathbf{4}_8$ $\mathbf{-100}_2$).



4. ***Индексный метод адресации*** (обозначение **X(R)** , код $\mathbf{6}_8$ $\mathbf{- 110}_2$).

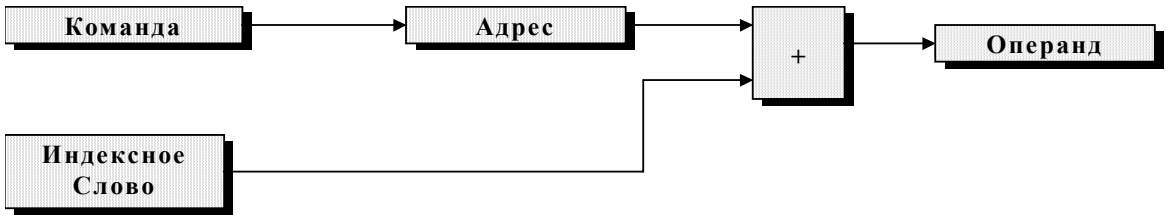


Рис. 1.7 Методы прямой адресации

Регистровый метод адресации

При регистровом методе адресации операнд находится непосредственно в указанном в команде регистре.

Пример 1.

Мнемоника	Восьмеричный код	Название
INC R3	***005203***	***Прибавление единицы***

Действие: к содержимому R3 прибавляется единица.

Автоинкрементный метод адресации

При автоинкрементном методе адресации содержимое выбранного регистра является адресом операнда. После выборки операнда содержимое этого регистра автоматически наращивается для обеспечения возможности обращения в дальнейшем к последующей ячейке. При *байтовых* операциях наращивание происходит на ***1***, при операциях с полными *словами* - на ***2***. Содержимое ***R6***, ***R7*** всегда наращивается на ***2***.

Автоинкрементный метод адресации особенно удобен при операциях с массивами и стеками. С помощью этого метода можно выбрать элемент таблицы, а затем нарастить указатель для обращения к следующему элементу в таблице. Хотя этот метод наиболее удобен при работе с таблицами, он может быть использован как общий метод для различных целей.

Пример 2.

Мнемоника	Восьмеричный код	Название
CLR (R5)+	***005025***	***Очистка***

Действие: ячейка, адрес которой содержится в R5, очищается, после чего адрес (содержимое R5) увеличивается на 2.

До выполнения операции		*После выполнения операции*	
20000/	005025	20000/	005025
30000/	111116	30000/	000000
R5/	030000	R5/	030002

Автодекрементный метод

Автодекрементный метод адресации также используется для обработки табулированных данных. Однако в отличие от автоинкрементного метода, адресация к ячейкам массива идет в противоположном направлении. При этом методе адресации содержимое выбранного РОН вначале уменьшается (для байтовых команд - на единицу, для команд с полными словами - на два), а затем используется как исполнительный адрес.

Сочетание автоинкрементного и автодекрементного методов адресации может быть эффективно использовано при работе со стеком.

Пример 3.

Мнемоника	*Восьмеричный код*	*Название*
INC -(R0)	***005240***	***Прибавление единицы***

Действие: содержимое *R0* уменьшается на *2* и используется как исполнительный адрес. К операнду, выбранному из ячейки по этому адресу, прибавляется единица.

До выполнения операции		После выполнения операции	
100/	005240	100/	005240
17774/	000000	17774/	000001
R0/	017776	R0/	017774

Индексный метод адресации

При индексном методе адресации исполнительный адрес определяется как сумма содержимого выбранного РОН с индексным словом. Этот метод позволяет осуществлять произвольный доступ к элементам структуры данных. Индексное слово содержится в следующей за командным словом ячейке памяти. При индексном методе адресации содержимое выбранного регистра может быть использовано в качестве базы для вычисления серии адресов.

Пример 4.

Мнемоника	Восьмеричный код	Название
CLR 200(R4)	***005064***	***Очистка***

Действие: адрес операнда определяется прибавлением к содержимому R4 кода 200, после чего ячейка с вычисленным адресом очищается.

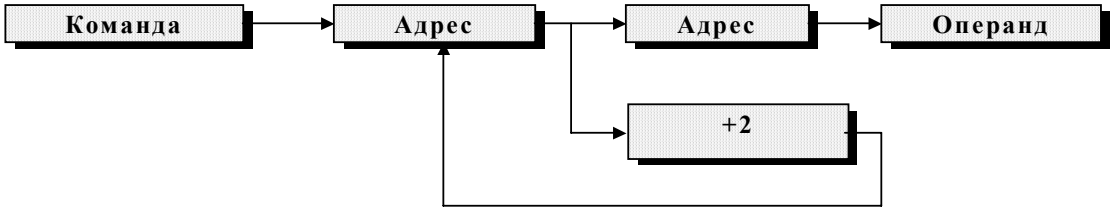
До выполнения операции	*После выполнения операции*
1020/ 005064	1020/ 005064
1022/ 000200	1022/ 000200
1200/ 177777	1200/ 000000
R4/ 001000	R4/ 001000

1.5.6 Методы косвенной адресации

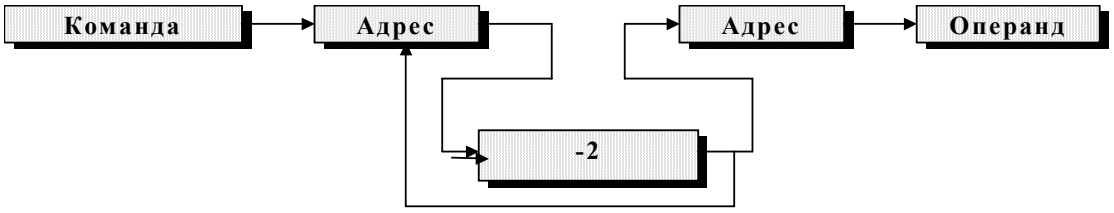
Четыре основных метода могут быть использованы в комбинации с косвенной адресацией. Если при регистровом методе содержимое выбранного регистра является операндом, то при косвенно - регистровом методе это содержимое является адресом операнда. При трех других косвенных методах вычисленный адрес позволяет выбрать только адрес операнда, а не сам операнд. Эти методы используются при обращении к таблицам, состоящим из адресов, а не из операндов (Рис. 1.8).

1.***Косвенно-регистровый метод адресации*** (обозначение **@R**, код $\mathbf{1}_8$ - 001_2)

2.***Косвенно-автоинкрементный метод адресации*** (обозн. **@(R)+**, код $\mathbf{3}_8$ - 010_2)



3.***Косвенно-автодекрементный метод адресации*** (обозн. **@-(R)** , код $\mathbf{5}_8$ - 101_2)



4.***Косвенно - индексный метод адресации*** (обозн. **@X(R)** , код $\mathbf{7}_8$ - 111_2)

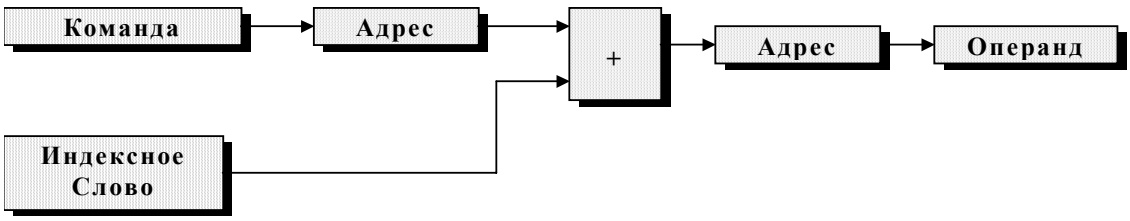


Рис. 1.8 Методы косвенной адресации

Пример 5.

Мнемоника	Восьмеричный код	Название
INC @(R2)+	***005232***	***Прибавление единицы***

Действие: содержимое ячейки, адрес которой находится в R2, используется как адрес операнда, операнд увеличивается на единицу, а содержимое R2 - на 2.

До выполнения операции	*После выполнения операции*
1000/ 005232	1000/ 005232
1010/ 000000	1010/ 000001
10300/ 001010	10300/ 001010
R2/ 010300	R2/ 010302

1.5.7 Использование счетчика команд (PC) в качестве РОН

Счетчик команд R7 может быть использован со всеми методами адресации, применяемыми в микро - ЭВМ. Однако наиболее эффективно он используется только с четырьмя. Эти методы адресации получили специальные наименования: непосредственный, абсолютный, относительный и косвенно - относительный. Использование этих методов дает возможность построения программ, работоспособность которых не теряется при перемещении их в любую область памяти. В таблице ниже приведены методы адресации с использованием R7. Необходимо понимать, что эти четыре метода аналогичны описанным выше, но в качестве РОН используется R7. Методы адресации с использованием счетчика команд в значительной мере упрощают обработку данных, не сформированных в массивы.

Восьм. код	**Двоичный код**	**Название**	**Функция**
2	010	*Непосредственный*	Операнд выбирается из ячейки, следующей за командным словом.
3	011	*Абсолютный*	Из ячейки, следующей за командным словом, выбирается адрес операнда.
6	110	*Относительный*	Операнд выбирается из ячейки, адрес которой определяется как сумма содержимого R7 и ячейки, следующей за командным словом.

7	111	*Косвенно-относительный*	Из ячейки, адрес которой определяется как сумма содержимого R7 и ячейки, следующей за командным словом, выбирается адрес операнда.

Непосредственный метод адресации

Непосредственный метод адресации имеет символическое обозначение #N. Он эквивалентен автоинкрементному методу адресации через счетчик команд R7. Этот метод обеспечивает экономию времени программиста при составлении программы за счет возможности помещения константы в ячейку памяти вслед за командным словом.

Пример 6.

Мнемоника	Восьмеричный код	Название
ADD #10, R0	***062700***	***Сложение***

Действие: содержимое R0 складывается с числом 10. Результат записывается в R0.

До выполнения операции	*После выполнения операции*
1020/ 062700	1020/ 062700
1022/ 000010	1022/ 000010
R0/ 000020	R0/ 000030

Примечание. После выборки команды содержимое R7 (адрес этой команды) увеличивается на 2. Так в поле адреса операнда источника записан код 27, R7 используется как указатель адреса при выборке операнда, после чего содержимое его вновь увеличивается на 2 для указания на следующую команду.

Абсолютный метод адресации

Абсолютный метод адресации имеет символическое обозначение @#A. Он эквивалентен косвенно-автоинкрементной адресации через R7. Этот метод удобен тем, что адрес операнда является его абсолютным адресом (т.е. он остается постоянным независимо от места расположения программы в памяти).

Пример 7.

Мнемоника	Восьмеричный код	Название
CLR @#1100	***005037***	***Очистка***

Действие: содержимое ячейки, следующей за командой, используется в качестве адреса операнда (в данном случае исполнительным адресом является код 1100). Содержимое ячейки с адресом 1100 очищается.

До выполнения операции	*После выполнения операции*
20/ 005037	20/ 005037
22/ 001100	22/ 001100
1100/ 177777	1100/ 000000

Относительный метод адресации

Относительный метод адресации имеет символическое обозначение X(PC) или A, где X - исполнительный адрес по отношению к счетчику команд. Этот метод эквивалентен индексной адресации через R7. Индексное слово хранится в следующей за командным словом ячейке и, будучи сложенным с содержимым R7, дает адрес операнда. Этот метод полезен при написании программы, которая может располагаться в различных местах памяти, так как адрес операнда фиксируется по отношению к содержимому R7. При необходимости перемещения программы в памяти операнд перемещается на то же число ячеек, что и сама команда.

Пример 8.

Мнемоника	Восьмеричный код	Название
INC A	***005267***	***Прибавление единицы***

Действие: к операнду, адрес которого определяется сложением содержимого R7 и индексного слова (000054), прибавляется "1".

До выполнения операции	*После выполнения операции*
1020/ 005267	1020/ 005267
1022/ 000054	1022/ 000054
1024/ ……	1024/ ……
1100/ 000000	1100/ 000001

Косвенно - относительный метод адресации

Косвенно - относительный метод адресации имеет символическое обозначение @X(PC) или @A, где X - адрес ячейки, содержащей исполнительный адрес, по отношению к счетчику команд. Этот метод эквивалентен косвенно - индексной адресации через СК.

1.6 Выполнение команд

Описание каждой команды включает: мнемонику, восьмеричный код,

формат команды, двоичный код, описание выполнения команды и выработки признаков, специальные пояснения и примеры.

1.6.1 Обозначения, используемые при описании команд

R	регистр общего назначения	***B***	байтовая команда
УС (SP)	указатель стека (R6)	***СК***	счетчик команд
РСП (RS)	регистр состояния процессора	***(PC)***	(R7)
SRC	источник	***ССП***	слово состояния процессора
(SRC)	операнд источника	***SS***	поле адресации операнда источника
DST	приемник	***DD***	поле адресации операнда приемника
(DST)	операнд приемника	***NN***	смещение (6 двоичных разрядов)
XXX	смещение (8 двоичных разрядов)	*******	"исключающее ИЛИ"
( )	содержимое ячейки	***&***	логическое умножение ("И")
V	логическое сложение ("ИЛИ")	***/=/***	не равно
=	равно	**<>**	не равно
$\bar{A}$	отрицание А ("НЕ")	**>=**	больше или равно
←	становится равным	**<=**	меньше или равно
PUSH	запись в стек	*****	умножение
POP	выборка из стека	******	возведение в степень

1.6.2 Выполнение байтовых команд

Большинство команд ЭВМ оперируют как с полными словами, так и с байтами. Байтовые команды с автоинкрементным или автодекрементными методами адресации для обращения к следующему байту изменяют содержимое указанного регистра на "1". Байтовые команды при регистровом методе адресации производят обработку младшего байта выбранного регистра. Если старший разряд командного слова (разряд 15) установлен в "1", он указывает, что команда байтовая. Если же в разряде 15 командного слова записан "0", команда оперирует с полным словом.

1.6.3 Одноадресные команды

Очистка	**CLR**	**0050*DD***
	CLR*B*	**1050*DD***

Действие:	***(DST)*** ← **0**
Описание:	в указанную ячейку записывается нуль. Для байтовой команды нуль записывается в указанный байт.
Признаки:	**N V Z C** **0 1 0 0**

Инвертирование	**COM**	**0051*DD***
	COM*B*	**1051*DD***

Действие:	***(DST)*** ← ***(DST)***
Описание:	содержимое указанной ячейки заменяется его двоичным обратным кодом (каждый разряд, содержащий ***0***, устанавливается, а каждый разряд, содержащий ***1***, очищается). Для байтовой команды операция производится по отношению к указанному байту.
Признаки:	**N V Z C** *** 0 * 1** • ***N*** = **1**, если *результат* < 0 • ***Z*** = **1**, если *результат* = 0

Прибавление единицы	**INC**	**0052*DD***
	INC*B*	**1052*DD***

Действие:	***(DST)*** ← ***(DST)*** + ***1***
Описание:	к содержимому указанной ячейки (или байту, если команда байтовая) прибавляется единица.
Признаки:	**N V Z C** *** * *** • **N** = **1**, если *результат* < **0** • **Z** = 1, если *результат* = 0 • **V** = 1, если *операнд* = **077777** • **C** - не изменяется.

Вычитание единицы

DEC	**0053*DD***
DEC*B*	**1053*DD***

Действие:	***(DST)*** ← ***(DST)*** *- 1*
Описание:	из содержимого указанной ячейки (или указанного байта для байтовых команд) вычитается единица.
Признаки:	**N** **V** **Z** **C** * * * • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0** • **V** = **1**, если *операнд* = **100000** • **C** - не изменяется.

Изменение знака

NEC	**0054*DD***
NEC*B*	**1054*DD***

Действие:	***(DST)*** ← ***(DST)***
Описание:	содержимое указанной ячейки (или байта для байтовых команд) заменяется его двоичным дополнением. Следует заметить, что число 100000 заменяется самим собой, так как не существует соответствующего ему положительного числа.
Признаки:	**N** **V** **Z** **C** * * * * • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0** • **V** = **1**, если *операнд* = **100000** • **C** = **0**, если *результат* = **0**

Проверка

TST	**0057*DD***
TST*B*	**1057*DD***

Действие:	***(DST)*** ← ***(DST)***

Описание:	в зависимости от содержимого указанной ячейки (или байта для байтовых команд) устанавливаются или очищаются признаки **N** и **Z**.
Признаки:	**N V Z C** *** 0 * 0** • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0**

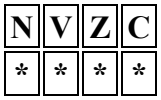
Увеличение или уменьшение чисел в кратное степени ***2*** число раз выполняется с помощью команд арифметического сдвига: ***ASR*** - арифметического сдвига вправо и ***ASL*** - арифметического сдвига влево. Знаковый разряд операнда (разряд ***15***) при арифметическом сдвиге вправо восстанавливается. В младший разряд при арифметическом сдвиге влево заносится нуль. Информация, сдвинутая за пределы ***C*** - разряда, теряется.

Арифметический сдвиг вправо	**ASR**	**0062*DD***
	ASR*B*	**1062*DD***

Действие:	***(DST)*** ← сдвинутое на *одну* позицию вправо ***(DST)***
Описание:	все разряды операнда сдвигаются вправо на одну позицию. Содержимое знакового разряда восстанавливается. **C** - разряд загружается содержимым младшего разряда операнда. Таким образом, ***ASR*** или ***ASRB*** выполняет деление числа со знаком на **2**.
Признаки:	**N V Z C** *** * * *** • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0** • **V** = **N*C** (после сдвига) • **C** = *содержимое младшего разряда* указанной ячейки

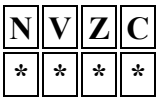
Арифметический сдвиг влево	**ASL**	**0063*DD***
	ASL*B*	**1063*DD***

Действие:	***(DST)*** ← сдвинутое на *одну* позицию влево ***(DST)***
Описание:	все разряды операнда сдвигаются влево на одну позицию. В младший разряд результата записывается ноль. **C** - разряд загружается содержимым старшего разряда операнда. Таким образом, ***ASL*** или ***ASLB*** выполняет умножение числа со знаком на **2**.

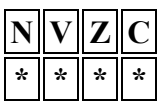
Признаки:	 • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0** • **V** = **N*C** (после сдвига) • **C** = *содержимое старшего разряда* указанной ячейки

Для облегчения последовательной проверки и поразрядной обработки операнда используются команды циклического сдвига. Они оперируют со словом операнда и **C**-разрядом как с содержимым ***17*** - разрядного регистра с циклическим переносом.

Циклический сдвиг вправо	**ROR**	**0060*DD***
	ROR*B*	**1060*DD***

Действие:	***(DST)*** ← циклически сдвинутое на одну позицию вправо ***(DST)***
Описание:	все разряды операнда циклически сдвигаются на одну позицию влево. Содержимое младшего разряда загружается в **C**- разряд, а прежнее содержимое **C** - разряда загружается в старший разряд результата.
Признаки:	 • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0** • **V** = **N*C** (после сдвига) • **C** = *содержимое младшего разряда* операнда

Циклический сдвиг влево	**ROL**	**0061*DD***
	ROL*B*	**1061*DD***

Действие:	***(DST)*** ← циклически сдвинутое на одну позицию влево ***(DST)***
Описание:	все разряды операнда циклически сдвигаются на одну позицию влево. Содержимое старшего разряда загружается в **C**- разряд, а прежнее содержимое **C** - разряда загружается в младший разряд результата.
Признаки:	 • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0**

	• **V** = **N*C** (после сдвига) • **C** = *содержимое старшего разряда* операнда

1.6.4 Двухадресные команды

Использование двухадресных команд обеспечивает возможность экономии машинного времени и сокращения количества команд в программе. Список двухадресных команд содержит четыре арифметические и четыре логические команды.

Арифметические команды

Пересылка	**MOV**	**01*SSDD***
	MOV*B*	**11*SSDD***

Действие:	***(DST)*** ← ***(SRC)***
Описание:	операнд источника ***(SRC)*** пересылается по адресу операнда приемника. Прежнее содержимое ячейки ***DST*** теряется, содержимое ячейки ***SRC*** не изменяется. При операциях с байтами команда ***MOVB*** с использованием регистрового метода адресации (единственная среди байтовых команд) расширяет старший разряд младшего байта (расширение знака). Все разряды старшего байта устанавливаются или сбрасываются в зависимости от того, установлен или сброшен старший (знаковый) разряд младшего байта. В других случаях ***MOVB*** оперирует с байтами так, как ***MOV*** со словами.
Признаки:	N V Z C * * • **N** = **1**, если ***(SRC)*** < **0** • **Z** = **1**, если ***(SRC)*** = **0** • **C** - не изменяется

Сравнение	**CMP**	**02*SSDD***
	CMP*B*	**12*SSDD***

Действие:	***(SRC) - (DST)***
Описание:	сравниваются операнды источника и приемника и, как результат сравнения, изменяются признаки, которые затем могут быть использованы для команд условных переходов. Оба операнда не изменяются. За ко-

	мандой сравнения обычно следует команда условного ветвления. Заметим, что в отличие от команды вычитания при выполнении команды ***CMP*** операнды меняются местами, т.е. имеет место ***(SRC) - (DST)***, а не ***(DST) - (SRC)***.
Признаки:	**N V Z C** *** * * *** • **N = 1**, если *результат* < **0** • **Z = 1**, если *результат* = **0** • **V = 1**, если арифметическое переполнение • **C = 1** , если перенос из старшего разряда

Сложение	**ADD**	**06*SSDD***

Действие:	***(DST)*** ← ***(SRC)*** + ***(DST)***
Описание:	операнд источника ***(SRC)*** складывается с операндом приемника ***(DST)*** и результат записывается по адресу операнда приемника. Первоначальное содержимое ***(DST)*** теряется. Содержимое ***(SRC)*** не изменяется. Сложение выполняется в двоичном дополнительном коде.
Признаки:	**N V Z C** *** * * *** • **N = 1**, если *результат* < **0** • **Z = 1**, если *результат* = **0** • **V = 1**, если арифметическое переполнение

Вычитание	**SUB**	**16*SSDD***

Действие:	***(DST)*** ← ***(DST)*** - ***(SRC)***
Описание:	из операнда приемника (DST) вычитается операнд источника (SRC) и результат записывается по адресу DST. Первоначальное содержание DST теряется, а содержимое SRC остается без изменения.
Признаки:	**N V Z C** *** * * *** • **N = 1**, если *результат* < **0** • **Z = 1**, если *результат* = **0** • **V = 1**, если арифметическое переполнение • **C = 1** , если перенос из старшего разряда

Логические команды

Из четырех логических команд три имеют такой же формат, как и двухадресные арифметические команды. Четвертая команда имеет специфический формат. Логические команды позволяют осуществлять поразрядную обработку данных.

Проверка разрядов	**BIT**	**03*SSDD***
	BIT*B*	**13*SSDD***

Действие:	***(DST)*** & ***(SRC)***
Описание:	выполняется логическая операция "И" над (SRC) и (DST) с соответствующим изменением признаков. Оба операнда не изменяют своего значения.
Признаки:	**N V Z C** *** 0 * ** • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0** • **C** - не изменяется

Команда ***BIT*** используется для проверки состояния разрядов операнда источника ***(SRC)***, для которых установлены соответствующие разряды в операнде приемника ***(DST)***.

Очистка разрядов	**BIC**	**04*SSDD***
	BIC*B*	**14*SSDD***

Действие:	***(DST)*** ← ***(SRC)*** & ***(DST)***
Описание:	каждый разряд операнда ***(DST)***, соответствующий установленному разряду операнда ***(SRC)***, очищается. Первоначальное содержимое ***DST*** теряется, содержимое ***SRC*** не изменяется.
Признаки:	**N V Z C** *** 0 * ** • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0** • **C** - не изменяется

Логическое сложение	**BIS**	**05*SSDD***
	BIS*B*	**15*SSDD***

Действие:	***(DST)*** ← ***(SRC)*** ∨ ***(DST)***
Описание:	над содержимым SRC и DST выполняется логическая операция "ИЛИ" и записывается результат по адресу DST. Разряды DST устанавливаются в "1", если соответствующие им разряды (SRC) находятся в "1". Прежнее содержимое DST теряется, а содержимое SRC остается неизменным.
Признаки:	**N V Z C** *** 0 * ** • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0** • **C** - не изменяется

Исключающее ИЛИ	**XOR**	**74*RDD***

Действие:	***(DST)*** ← ***R * (DST)***
Описание:	над содержимым указанного регистра и содержимым ***DST*** выполняется операция *исключающее* ***ИЛИ***. Результат записывается в ***DST***. Содержимое регистра ***R*** не изменяется.
Признаки:	**N V Z C** *** 0 * ** • **N** = **1**, если *результат* < **0** • **Z** = **1**, если *результат* = **0** • **C** - не изменяется

1.6.5 Команды управления программой

К командам управления программой относятся команды ветвления, обращения к подпрограмме, возврата из подпрограммы, безусловного перехода и другие.

Команды ветвления.

Эти команды вызывают ветвление по адресу, являющемуся суммой смещения (умноженного на ***2***) и текущего содержимого счетчика команд ***R7***, если условие ветвления выполняется.

Смещение показывает, на сколько ячеек нужно перейти относительно текущего содержимого счетчика команд в ту или другую сторону. Так как слова имеют четные адреса, то для получения истинного исполнительного адреса смещение необходимо умножить на два перед прибавлением к счетчику команд ***R7***, который всегда указывает на слово. Старший разряд смещения (разряд *7*) является знаковым разрядом. Если он установлен в ***1***, смещение отрицательное, ветвление происходит в сторону уменьшения адреса (в обратном направлении). Если в разряде *7* содержится ***0***, смещение - положительное, и ветвление происходит в сторону увеличения адресов (в прямом направлении). Восьмиразрядное смещение позволяет производить ветвление в обратном направлении максимально на ***2008*** слов от слова, на которое указывает текущее содержимое ***СК***, и на ***1778*** слов в прямом направлении.

Ветвление безусловное	**BR**	**000400 + *XXX***
Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***	
Описание:	с помощью одной команды управление программой передается ячейке, адрес которой находится в ограниченной области.	
Признаки:	**N V Z C** □ □ □ □ — *Не изменяются*	

Новое содержимое ***СК*** = *текущее содержимое* ***СК*** + ***2 * XXX*** (смещение), где текущее содержимое ***СК*** = *адрес* команды ветвления + ***2***.

Простые условные ветвления

Ветвление, если не равно (нулю)	**BNE**	**001000 + *XXX***
Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***Z*** = **0**	
Описание:	проверяется состояние разряда ***Z*** и вызывается ветвление, если он очищен. Команда ***BNE*** обратна по действию команде ***BEQ***. Вместе с командой ***BIT*** она используется для проверки того, что установленные разряды операнда источника соответствуют установленным разрядам операнда приемника. В общем случае она используется для проверки неравенства нулю результата предыдущей операции.	
Признаки:	**N V Z C** □ □ □ □ — *Не изменяются*	

Ветвление, если равно (нулю)	**BEQ**	**001400 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***Z*** = **1**
Описание:	проверяется состояние разряда ***Z*** и вызывается ветвление, если он установлен. Команда ***BEQ*** обратна по действию команде ***BNE***. Вместе с командой ***CMP*** команда ***BEQ*** используется для проверки равенства двух величин. Вместе с командой ***BIT*** она используется для проверки того, что очищенные разряды операнда источника соответствуют установленным разрядам приемника. В общем случае эта команда используется для проверки равенства нулю результата предыдущей операции.
Признаки:	**N V Z C** (□ □ □ □) *Не изменяются*

Ветвление, если плюс	**BPL**	**100000 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***N*** = **0**
Описание:	проверяется разряд ***N*** и вызывается ветвление, если он очищен. Вместе с командой ***TSTB*** она используется для проверки установки разряда *7* (флага готовности) *регистра состояния* периферийного устройства. В общем случае эта команда используется для проверки положительности результата предыдущей операции. Команда ***BPL*** обратна по действию команде ***BMI***.
Признаки:	**N V Z C** (□ □ □ □) *Не изменяются*

Ветвление, если минус	**BMI**	**100400 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***N*** = **1**
Описание:	проверяется состояние разряда ***N*** и вызывается ветвление, если он установлен. Используется для проверки знака (старший разряд) результата предыдущей операции.
Признаки:	**N V Z C** (□ □ □ □) *Не изменяются*

Ветвление, если нет арифметического переполнения	**BVC**	**102000 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***V*** = **0**
Описание:	поверяется состояние разряда ***V*** и вызывается ветвление, если он очищен.
Признаки:	**N V Z C** □ □ □ □ *Не изменяются*

Ветвление, если арифметическое переполнение	**BVS**	**102400 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***V*** = **1**
Описание:	проверяется состояние разряда ***V*** и вызывается ветвление, если он установлен.
Признаки:	**N V Z C** □ □ □ □ *Не изменяются*

BVS используется для обнаружения арифметического переполнения в результате исполнения предыдущей операции. ***BVS*** обратна по действию команде ***BVC.***

Ветвление, если нет переноса	**BCC**	**103000 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***C*** = **1**
Описание:	проверяется разряд ***C*** и вызывается ветвление, если он очищен.
Признаки:	**N V Z C** □ □ □ □ *Не изменяются*

Ветвление, если перенос	**BCS**	**103400 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***C*** = **1**
Описание:	проверяется разряд ***C*** и вызывается ветвление, если он установлен. ***BCS*** используется для проверки наличия переноса в результате предыдущей операции.
Признаки:	**N V Z C** (□ □ □ □) *Не изменяются*

Команда **BCS** обратна по действию команде **BCC**.

Условные ветвления по результату операций над числами

Особые комбинации разрядов признаков поверяются с помощью команду условного ветвления по результату операций над числами. Эти команды используются для проверки результатов выполнения команд, в которых операнды рассматриваются как двоичные числа, имеющие знак. Заметим, что отличие в сравнении чисел, имеющих знак, и чисел без знака обусловлено их различным представлением в арифметике, использующей дополнительные коды. Для 16-разрядных чисел, не имеющих знака, последовательность следующая:

наибольшее	***177777***
	077776
	.....
	000002
	000001
наименьшее	***000000***

Командами условного ветвления по результату операции над числами являются следующие: ***BGE***, ***BLT***, ***BGT***, ***BLE***.

Ветвление, если больше или равно (нулю)	**BGE**	**002000 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***N*V = 0***
Описание:	вызывается ветвление, если оба разряда признаков N и V установлены или очищены (т.е. если результат операции *исключающее ИЛИ* над содержимым разрядов N и V равен ***0***). Таким образом, команда ***BGE*** всегда будет вызывать ветвление, если она следует за операцией сложения двух положительных чисел. Команда ***BGE*** будет также вызывать ветвление по нулевому результату.
Признаки:	N V Z C _ _ _ _ *Не изменяются*

Ветвление, если меньше (нуля)	**BLT**	**002400 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***N*V = 1***
Описание:	вызывается ветвление, если результат операции *исключающее ИЛИ* над содержимым разрядов N и V равен ***1***. Команда ***BLT*** обратна по действию команде ***BGE***. Таким образом, команда ***BLT*** всегда будет вызывать ветвление, если она следует за операцией сложения двух отрицательных чисел, даже если происходит переполнение. В частности, команда ***BLT*** будет всегда вызывать ветвление, если она следует за командой сравнения отрицательного операнда и положительного операнда назначения, даже если произошло переполнение. Команда ***BLT*** никогда не будет вызывать ветвления, если она следует за командой сравнения (***CMP***) положительного операнда источника и отрицательного операнда назначения. Она также не будет вызывать ветвления, если результат предыдущей операции равен нулю без переполнения.
Признаки:	N V Z C _ _ _ _ *Не изменяются*

Ветвление, если больше (нуля)	**BGT**	**003000 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***Z*** ∨ (***N*C) = 0***
Описание:	команда ***BGT*** подобна команде ***BGE***, за исключением того, что команда ***BGT*** не вызывает ветвления по нулевому результату.
Признаки:	N V Z C □ □ □ □ *Не изменяются*

Ветвление, если меньше или равно (нулю)	**BLE**	**003400 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***Z*** ∨ (***N*C) = 1***
Описание:	команда ***BGT*** подобна команде ***BGE***, за исключением того, что команда ***BGT*** не вызывает ветвления по нулевому результату.
Признаки:	N V Z C □ □ □ □ *Не изменяются*

Условные ветвления по результату операции над кодами.

Условные ветвления по результату операции над кодами обеспечивают методы проверки результата операций сравнения операндов, рассматриваемых как величины без знака.

Ветвление, если больше	**BHI**	**101000 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***C = 1*** или ***Z = 0***
Описание:	вызывается ветвление, если предыдущая операция не вызывала переноса или появления нулевого результата. Это происходит при операциях сравнения ***CMP***, когда операнд источника больше операнда приемника.
Признаки:	N V Z C □ □ □ □ *Не изменяются*

Ветвление, если больше или равно	**BHIS**	**103000 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***C = 1*** или ***Z = 0***
Описание:	По своему действию команда ***BHIS*** идентична команде ***BCC***. Другая мнемоника вводится только в связи с другим использованием команды.
Признаки:	N V Z C □ □ □ □ *Не изменяются*

Ветвление, если меньше	**BLO**	**103400 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***, если ***C = 1*** или ***Z = 0***
Описание:	По своему действию команда ***BLO*** идентична команде ***BCS***. Другая мнемоника вводится только в связи с другим использованием команды.
Признаки:	N V Z C □ □ □ □ *Не изменяются*

Ветвление, если меньше или равно	**BLOS**	**101100 + *XXX***

Действие:	***(СК)*** ← ***(СК)*** + ***2 * XXX***
Описание:	По своему действию команда ***BLOS*** является обратной команде ***BHI***. Ветвление будет происходить, если операнд источника меньше или равен операнду приемника.
Признаки:	N V Z C □ □ □ □ *Не изменяются*

Безусловный переход	**JMP**	**0001*DD***

Действие:	***(СК)*** ← ***(DST)***
Описание:	команда ***JMP*** обеспечивает возможность перехода на любую команду программы (не ограничиваясь пределами в *+177* и *-200* слов как команда ***BR***) с использованием всех методов адресации, за исключением регистрового. Использование регистровой адресации вызывает прерывание программы по условию *запрещенная команда* через адрес вектора ***4***. Метод косвенной адресации может применяться и вызывает передачу управления программой по адресу, содержащемуся в указанном регистре. Заметим, что команды - это полные слова и поэтому должны выбираться из ячеек с четными адресами. Команда ***JMP*** с косвенно - индексным методом адресации позволяет передать управление по адресу, являющемуся элементом таблицы адресов.
Признаки:	**N V Z C** *Не изменяются*

Команды обращения к подпрограмме и выхода из подпрограммы

Эти команды обеспечивают возможность автоматического вложения подпрограмм, выхода из подпрограммы и многократного входа в подпрограмму. В подпрограммах могут быть обращения к другим подпрограммам (или к самим себе) без специального программного запоминания адресов возврата. Процедура обращения к подпрограмме и выхода из нее не изменяет подпрограмму. Это позволяет использовать одну и ту же подпрограмму несколькими процессами, осуществляющими прерывание программы.

Обращение к подпрограмме	**JSR**	**004*RDD***

Действие:	• ***(TMP)*** ← ***(DST)***	запись содержимого приемника во внутренний регистр процессора:
	• *PUSH* (***SP***) ← ***(R)***	запись содержимого указанного регистра в стек:
	• ***(R)*** ← ***(PC)***	счетчик команд ***(PC)*** содержит адрес ячейки, следующей за командой ***JSR***; этот адрес заносится в регистр ***R***;
	• ***(PC)*** ← ***(TMP)***	занесение в счетчик команд нового содержимого, определяющего начальный адрес подпрограммы.

Описание:	При выполнении команды ***JSR*** старое содержимое указанного регистра (*указатель связи*) автоматически засылается в стек, и в регистр поступает новая связующая информация. Таким образом, обращение к подпрограмме, вложенное в подпрограмму на любую глубину, осуществляется с помощью регистра *указатель связи*. Нет необходимости в том, чтобы задавать максимальную глубину обращения к данной подпрограмме или включать команды запоминания и восстановления *указателя связи* в каждую подпрограмму. Обращение к подпрограмме по команде ***JSR*** может осуществляться с помощью автоинкрементной адресации (если каждый последующий вход в подпрограмму осуществляется через ячейку, адрес которой на ***2*** больше предыдущего) или индексом адресации (если вход в подпрограмму осуществляется по адресам, расположенным в произвольном порядке). Оба эти метода могут быть также косвенными.
Признаки:	N V Z C □ □ □ □ *Не изменяются*

Возврат из подпрограммы	**RTS**	**00020*R***

Действие:	• ***(PC)*** ← ***(R)*** • ***(R)*** ← ***(SP)*** *POP*
Описание:	Содержимое регистра ***(R)*** загружается в счетчик команд, после чего извлекается верхний элемент стека и засылается в указанный регистр. Возврат из подпрограммы обычно выполняется через тот же самый регистр, который используется при обращении к ней. Таким образом, выход и подпрограммы, обращение к которой осуществляется командой ***JSR PC, DST***, выполняется командой ***RTS PC***, а выход из подпрограммы, обращение к которой осуществлялось командой ***JSR R5, DST*** с использованием любого из методов адресации, выполняется командой ***RTS R5***.
Признаки:	N V Z C □ □ □ □ *Не изменяются*

Вычитание единицы и ветвление	**SOB**	**077*RNN***

Действие:	• ***(R)*** ← ***(R) - 1*** • ***(PC)*** ← ***(PC) - 2 * NN***, если *результат* <> ***0*** • ***(PC)*** ← ***(PC)***, если *результат* = ***0***

Описание:	Содержимое регистра уменьшается на единицу. Если результат не равен , в счетчик команд загружается новое содержимое, определяемое вычитанием удвоенного смещения из текущего содержимого счетчика команд. В команде ***SOB*** *смещением* является шестиразрядное положительное число. Эта команда может быть эффективно использована для организации различного рода счетчиков, циклов. Следует отметить, что команда ***SOB*** не может быть использована для передачи управления в прямом направлении.
Признаки:	N V Z C ☐ ☐ ☐ ☐ *Не изменяются*

Команды прерывания

Команды прерывания обеспечивают возможность обращения к программам управления вводом - выводом, программам отладки и программам, разработанным пользователем. Когда происходит прерывание, текущее содержимое счетчика команд и содержимое регистра состояния процессора записывается в стек. Новое содержимое счетчика команд и регистра состояния процессора загружается из вектора прерывания, состоящего из двух слов. При выходе из прерывания используются команды ***RTI*** и ***RTT***, которые восстанавливают ***СК*** и ***РСП***, извлекая их прежнее содержимое из стека. Векторы прерывания расположены по фиксированным, приписанным каждому виду прерывания адресам.

Командное прерывание для системных программ	**EMT**	**104000 - 104377**
Действие:	• *PUSH* ***(SP) ← (RS)*** • *PUSH* ***(SP) ← (PC)*** • ***(PC) ← (30)*** • ***(RS) ← (32)***	
Описание:	команды ***EMT*** имеют коды операций от *104000* до *104377*, которые могут быть использованы для передачи информации в моделирующую программу (т.е. информации о функции, которая должна быть выполнена). Вектор прерывания команды ***EMT*** находится по адресу ***30***. Новое содержимое ***СК*** берется из ячейки с адресом ***30***, а новое содержимое ***РСП*** - из ячейки с адресом ***32***.	
Признаки:	N V Z C ☐ ☐ ☐ ☐ *Не изменяются*	

Возврат из прерывания	**RTI**	**000002**

Действие:	• ***(PC)*** ← ***(SP)*** *POP* • ***(RS)*** ← ***(SP)*** *POP*
Описание:	команда ***RTI*** используется для выхода из подпрограмм обслуживания внешних и внутренних прерываний. Содержимое счетчика команд и регистра состояния процессора восстанавливается с помощью стека.
Признаки:	**N V Z C** ☐ ☐ ☐ ☐ *Не изменяются*

1.6.6 Команды управления процессором

Останов	**HALT**	**000000**

Действие:	• ***(PC)*** ← ***(SP)*** *POP* • ***(RS)*** ← ***(SP)*** *POP*
Описание:	Центральный процессор переходит в режим опроса пультового терминала. В *счетчике команд* сохраняется адрес команды, которая должна быть исполнена *следующей*.
Признаки:	**N V Z C** ☐ ☐ ☐ ☐ *Не изменяются*

2 Семейство процессоров Intel x86

Первым представителем семейства Intel x86, или, согласно официальной классификации фирмы Intel (**Int**egrated **El**ectronics, США), семейства процессоров IA (Intel Architecture), является микропроцессор 8086, разработанный к 1978 году. Программы, написанные для него, выполняются на всех последующих процессорах семейства, включая все современные модели. Предшествующие процессоры - 8080, 8085 и 4004 (разработка 1967 г.) из-за несовместимости по объектному коду, остаются вне семейства, являясь, тем не менее, важными этапами на пути развития Intel x86 [3]. Сегодня процессоры этого семейства стали стандартом де-факто для большинства персональных компьютеров (ПК) во всем мире. Ниже приводятся основные данные наиболее известных представителей этой серии:

Таблица 1

Название	Начало производства	Макс. тактовая частота первых серийных образцов	Число транзисторов, млн.	Размер основных регистров ЦПУ	Ширина внешней шины данных	Размер внешнего адресного пространства памяти	Кэш-память
8086	1978	8 МГц	0.029	16	16	1 Мб	нет
80286	1982	12.5МГц	0.134	16	16	16 Мб	6В на СР [1]
80386	1985	20 МГц	0.275	32	32	4 Гб	8В на СР [1]
80486	1989	25 МГц	1.2	32	32	4 Гб	8КВ L1
Pentium	1993	60 МГц	3.1	32	64	4 Гб	16КВ L1
Pentium Pro (P6)	1995	200 МГц	5.5	32	64	64 Гб	16 КВ L1, 256КВ L2
Pentium III (P6)	1999	1 ГГц	9.5	32	64	64 Гб	2x16К L1, 512КВ L2
Pentium 4	2000	1.4 ГГц	42	32	2x64	64 Гб	8К+12К L1, 256КВ L2
семейство Itanium 1	2001	733МГц	25	64	64	$2^{63}+2^{63}$ bytes	16Ki+16KB L1, 96KB-2.5MB L2, 2-32MB L3
семейство Itanium 2	2002	900МГц	220-1720	64	128		

Примечание 1: СР - сегментный регистр.

2.1 Микроархитектура процессоров 8086 и Pentium Pro

Микропроцессор 8086 ориентирован на выполнение команд параллельно с их выборкой и может быть условно разделен на две части, работающие асинхронно (

Рис. 2.1): устройство сопряжения с внешними шинами (УС) и устройство обработки (УО). Устройство сопряжения обеспечивает формирование 20-разрядного физического адреса памяти, выборку команд и операндов из памяти, организацию очередности команд и запоминание результатов выполнения команд в памяти. В состав УС входит шесть 8-разрядных регистров очереди команд, четыре 16-разрядных сегментных регистра, 16-разрядный регистр обмена и 16-разрядный сумматор адреса, интерфейс с внешними шинами. Регистры очереди команд организованы по принципу FIFO - «первым пришел - первым вышел». УС готово выполнить цикл выборки 16-разрядного слова из памяти всякий раз, когда в очереди освобождаются, по меньшей мере, два байта, а УО извлекает из очереди команды по мере их выполнения. При выполнении команд передачи управления, например условных и безусловных переходов, очередь очищается УС и начинает заполняться заново.

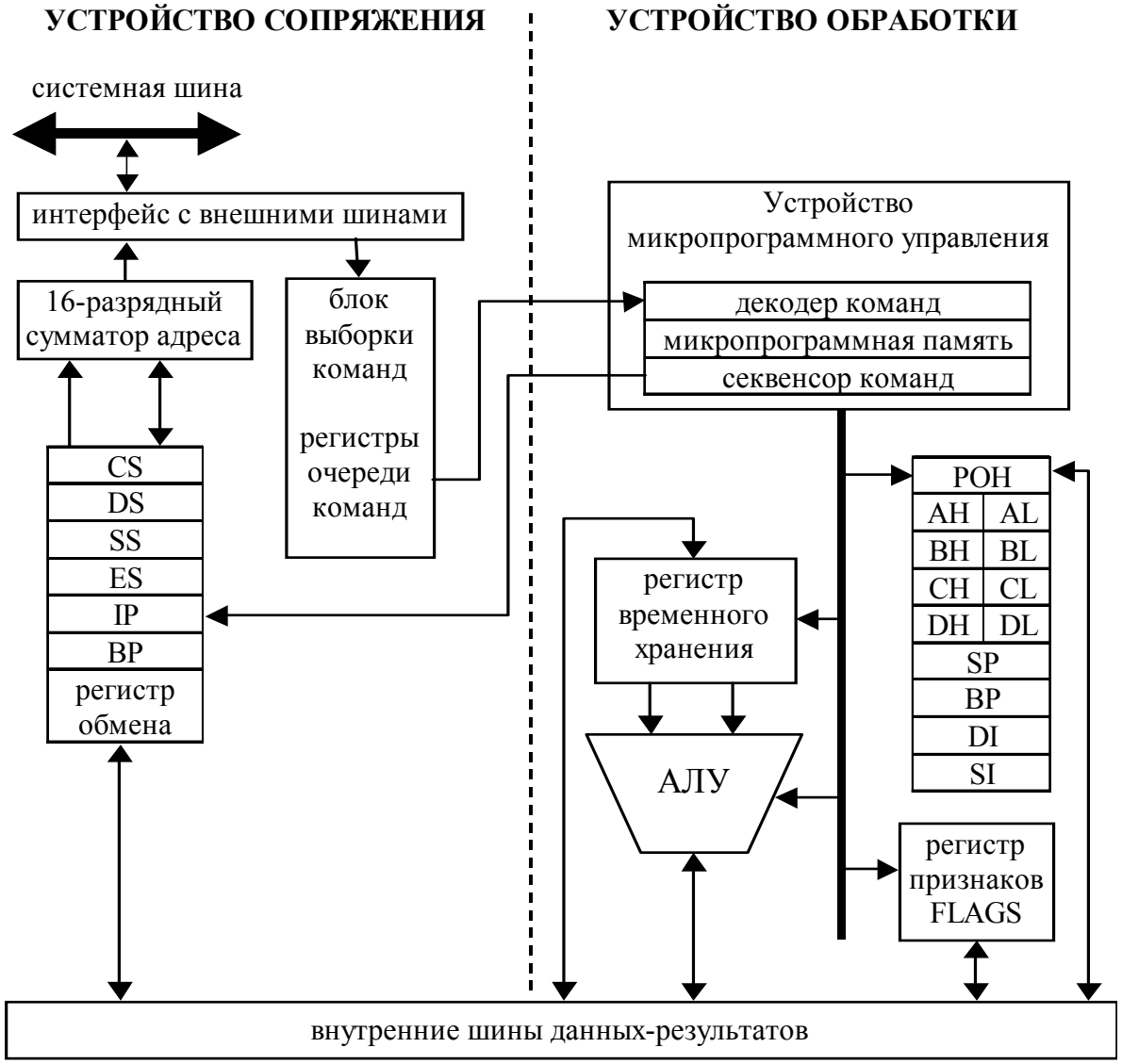


Рис. 2.1. Микроархитектура процессора 8086

Устройство обработки предназначено для выполнения операций по обработке данных и состоит из устройства микропрограммного управления (УМУ), 16-разрядного АЛУ, восьми 16-разрядных регистров общего назначения и регистра признаков. Команды из очереди, сформированной УС, поступают в УМУ, где декодируются и выполняются в 16-разрядном АЛУ согласно процедурам, записанным в памяти микропрограмм. Последовательное выполнение команд обеспечивается секвенсором команд, часть которого (регистр счетчика команд IP) изображена в составе УС, т.к. именно УС записывает в IP смещение следующей команды, т.е. положение новой команды относительно начала сегмента команд. УО обменивается данными с УС через внутреннюю 16-разрядную шину и регистр обмена (

Рис. 2.1).

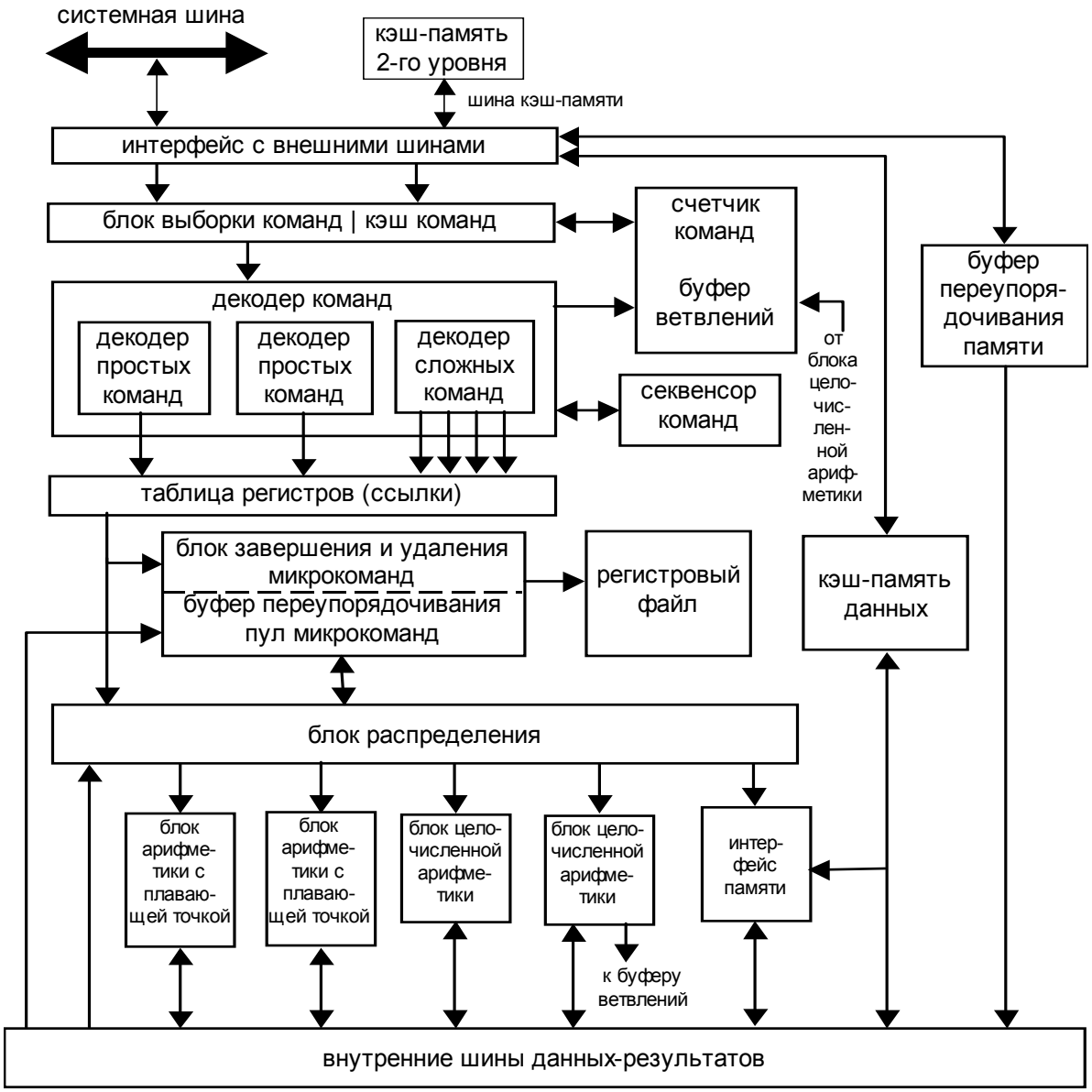


Рис. 2.2. Микроархитектура процессоров семейства Pentium Pro

Для сравнения приводится (Рис. 2.2) микроархитектура процессоров

шестого поколения Pentium Pro. Как видно, архитектура со времени 8086 претерпела не только количественные (разрядность внутренних и внешних шин данных, регистров), но и существенные качественные изменения. Pentium Pro имеет *суперскалярную* архитектуру, т.е. может одновременно выполнять несколько команд за один такт. Эту возможность обеспечивают несколько АЛУ - два блока целочисленной арифметики и два блока с плавающей точкой. Другая важная особенность – т.н. динамическое исполнение – команды разбиваются на простейшие операции, порядок независимого исполнения которых определяется блоком «завершения и удаления микрокоманд» с буферами переупорядочивания и пулом микрокоманд (Рис. 2.2).

На Рис. 2.3 представлены обозначения микропроцессоров для принципиальных электрических схем, на которых видны внешние шины и сигналы 16-разрядного 8086 и 32-разрядного 80486.

2.2 Система команд и методы адресации процессоров 8086/8088

Процессоры 8086/8088 – первые в семействе x86, отличаются друг от друга шириной шины данных: в 8088 – 8-разрядная шина, в 8086 – 16-разрядная. Поэтому чтение и запись 16-разрядных данных выполняются примерно в два раза медленнее для 8088, т.к. возможна передача только одного байта за один цикл, а не 16-разрядного слова как у 8086.

2.2.1 Основные характеристики микропроцессора 8086

Система команд	135 команд
Адресация	безадресная, одно-, двухадресная
Типы обрабатываемых данных	биты, байты, 16-разрядные слова, строки до 64К байт
Число программно доступных регистров	14 шестнадцатиразрядных
Число адресуемых устройств ввода/вывода	64К/64К
Число способов вычисления адреса операндов памяти	24
Разрядность шин адреса/данных	20/16
Объем адресуемой памяти	1Мбайт
Тактовая частота (на момент выпуска)	8 МГц
Максимальное быстродействие (операции/с типа "регистр-регистр")	4 млн.
Потребляемая мощность	не более 1,75 Вт

2.2.2 Регистры процессора

Микропроцессор 8086 имеет 12 программно-доступных шестнадцатираз-

рядных регистров (

Рис. 2.1), регистр счетчика команд IP (Instruction Pointer) и регистр флагов (или регистр состояния процессора) FLAGS.

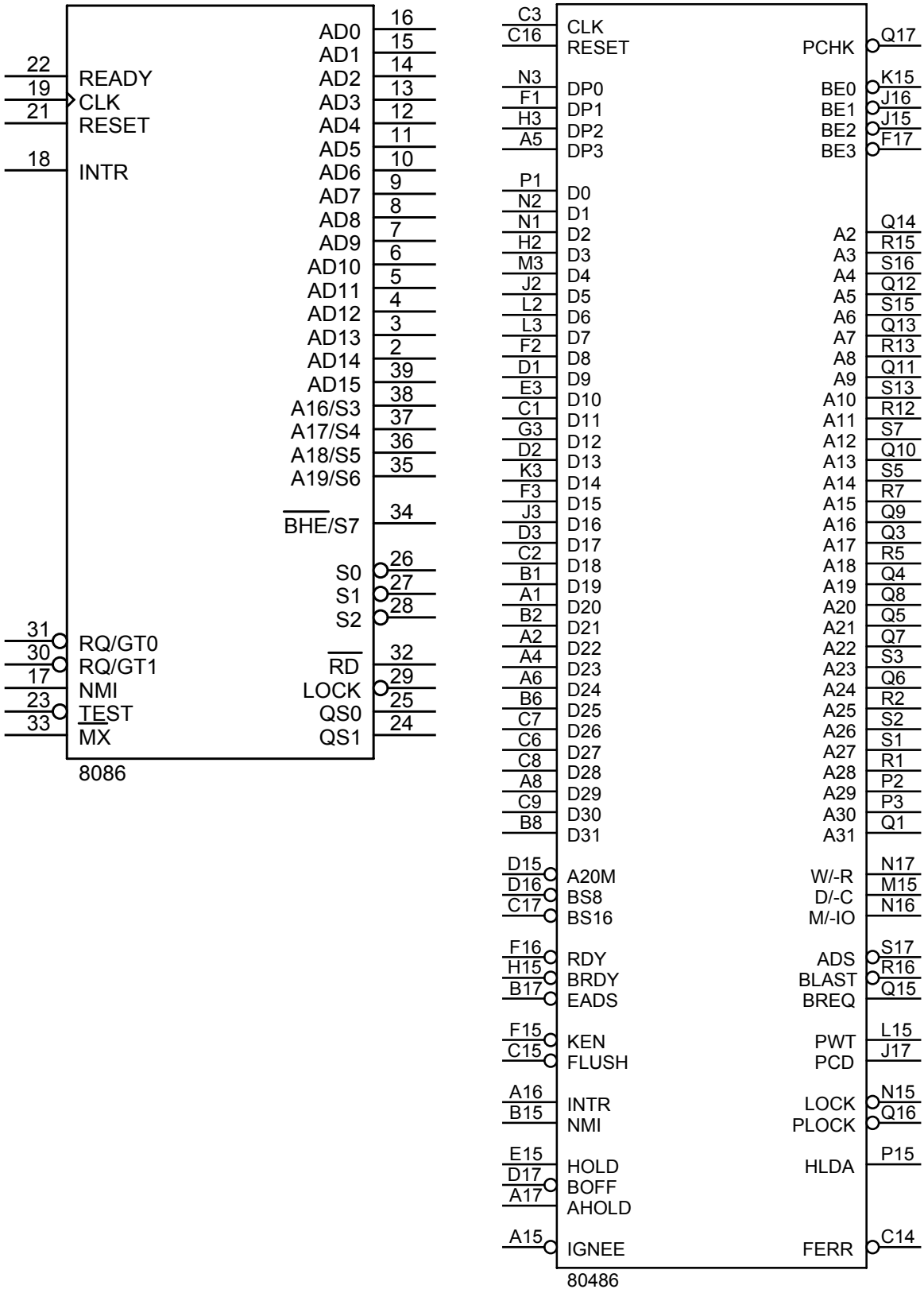


Рис. 2.3 ИС процессоров 8086 и 80486 в DIP и PGA корпусах

Среди программно-доступных регистров выделяют следующие группы (см. Рис. 2.4):

- Регистры данных: AX – аккумулятор (Accumulator); BX - базовый регистр (Base); CX - регистр счетчика (Counter); DX - регистр данных (Data).
- Регистры-указатели (индексные регистры): SI - индекс источника (Source Index); DI - индекс приемника (Destination Index); BP - указатель базы (Base Pointer); SP - указатель стека (Stack Pointer).
- Сегментные регистры: SS - сегмент стека (Stack Segment); DS - сегмент данных (Data Segment); ES - дополнительный сегмент (Extended data Segment); CS – сегмент кода (Code Segment).

16-битные регистры AX, BX, CX, DX состоят из двух 8-битных половин, к которым можно независимо обращаться по именам AH, BH, CH, DH - старшие байты и AL, BL, CL, DL - младшие байты.

Рис. 2.4 Регистры процессора 8086

Биты (или флаги) регистра признаков FLAGS разделяются на условные, отражающие результат предыдущей операции ALU, и управляющие, от которых зависит выполнение специальных функций.

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Рис. 2.5 Флаги процессора 8086

Флаги TF, IF, DF относятся к управляющим флагам, остальные - к условным (флагам состояния). Флаги отражают следующие условия:

- флаг переноса *CF* устанавливается в 1 при переносе/заеме, возникающем при сложении/вычитании байтов или слов, а также принимает значение выдвигаемого бита при сдвигах операнда;
- флаг четности *PF* устанавливается в 1, если младшие 8 бит резуль-

тата имеют четное число единиц;

- флаг вспомогательного переноса *AF* устанавливается в 1, если есть перенос/заем из младшей тетрады в старшую 8- или 16-битного результата в операциях десятичного сложения/вычитания;
- флаг нуля *ZF* устанавливается в 1 при нулевом результате операции;
- флаг знака *SF* устанавливается в 1 при отрицательном результате операции при использовании дополнительного кода;
- флаг переполнения *OF* устанавливается в 1 при потере старшего бита результата сложения или вычитания;
- если флаг направления *DF* установлен в 1, то используется автодекрементная адресация при выполнении операции обработки строк, если *DF* установлен в 0, то используется автоинкрементная адресация;
- если флаг прерывания *IF* установлен в 1, то внешние маскируемые прерывания разрешены, иначе запрещены;
- если флаг трассировки *TF* установлен в 1, то процессор переходит в состояние прерывания после выполнения каждой операции, что позволяет проводить пошаговую отладку программ.

2.2.3 Организация памяти

Хотя процессор имеет 20-разрядную адресную шину, которая соединяет его с физической памятью, он оперирует с 16-битными логическими адресами, состоящими из 16-разрядного базового адреса сегмента и 16-разрядного смещения в сегменте. Физические, 20-разрядные адреса данных и команд формируются путем сложения содержимого регистров-указателей и смещенного на 4 бита влево содержимого сегментных регистров. Т.н. *эффективный адрес* данных получается как сумма содержимого регистров BX или BP, содержимого регистров SI или DI и смещения. Затем из эффективного адреса и содержимого сегментного регистра формируется физический адрес (Рис. 2.6). В формировании физического адреса команды участвуют IP и CS. Таким образом, адресное пространство разбивается на 4 сегмента емкостью 64К адресов по числу сегментных регистров. Регистр CS указывает на текущий сегмент кода (программы), откуда выбираются команды. Регистр DS указывает на текущий сегмент данных, в котором содержатся переменные. Регистр SS адресует текущий сегмент стека, в котором реализуются все стековые операции. Наконец, регистр ES определяет текущий дополнительный сегмент данных. Смещенное содержимое сегментного регистра определяет положение сегмента в 20-разрядном адресном пространстве, а регистры-указатели определяют положение команды или данных внутри сегментов.

Поскольку при формировании эффективного адреса содержимое сегментного регистра сдвигается на 4 бита, сегмент всегда начинается с адре-

са, кратного 16, т.е. на границе 16-байтового блока памяти (параграфа). Сегменты в памяти могут располагаться как последовательно, так и с наложением друг на друга. Если программа превышает 64 Кбайт, то необходимо перезагружать сегментный регистр CS новым значением базового адреса. Точно также, если данные превышают 64 Кбайт, то необходимо перезагрузить регистр DS.

Рис. 2.6 Формирование физического адреса данных и команды

Память организована в виде одномерного массива байтов с физическими адресами от 00000_{16} до $FFFFF_{16}$. Две области адресного пространства памяти зарезервированы для выполнения специальных функций, связанных с обработкой прерываний и системным сбросом. Этими областями являются первые 128 байт (физические адреса 00000 - 0007F) и последние 16 байт (физические адреса FFFF0 - FFFFF). Данные области использовать для других целей нельзя.

Байты в памяти организуются в слова таким образом, что байту, имеющему меньший адрес, соответствуют менее значимые позиции разрядов в слове. Каждый байт или слово памяти адресуется с помощью 20-битного адреса, причем в случае адресации слова адрес указывает на его младшую часть. Например, адрес 00000_{16} может обозначать и байт с этим адресом, что условно записывается в виде [00000] = 34h, и слово с таким же адресом, что записывается в виде [00000] = 1234h. Тогда старший байт слова, [00001] = 12h. Квадратные скобки обозначают ячейку памяти, адрес которой находится в этих скобках, h – шестнадцатеричную систему счисления. Команды, байты и слова можно размещать по любому адресу байта, однако рекомендуется размещать слова в памяти по четным адресам, так как процессор может передавать такие слова за один цикл обращения к памяти. Слово с четным адресом называется выравненным на границу слова. Слова с нечетными адресами (невыравненные) также допустимы, однако они считываются в два раза медленнее (требуют два цикла обращения к памяти).

2.2.4 Форматы команд

Команды i8086 имеют переменную длину от 1 до 6 байт. По числу обрабатываемых операндов команды подразделяются на безадресные, одноадресные и двухадресные. В двухадресных командах результат всегда записывается по первому адресу и только один из операндов может находиться в памяти. При этом в мнемонике, *операнд-приемник записывается слева от запятой-разделителя операндов, а операнд-источник – справа*.

Всего существует четыре источника операндов: тело команды, регистр, память и порт ввода/вывода. В первом случае операнд называется непосредственным.

Первый байт команды содержит код операции (*КОП*), в состав которого могут входить специальные разряды d, s и w. При w=1 операции выполняются с 16-разрядными словами, при w=0 – с байтами. Разряд d определяет направление передачи данных в двухоперандных командах: из регистра в регистр/память (d=0) или из регистра/памяти в регистр (d=1). S – определяет расширение 8-битных непосредственных данных до полного размера (s=1) или нет (s=0). При некоторых сочетаниях команд и методов адресации (регистровый метод адресации) положение операнда может задаваться непосредственно в байте кода операции (см. следующий раздел), но чаще для этого используется т.н. «постбайт».

В командах, имеющих длину 2 и более байта, второй байт называется *постбайтом*. Он выполняет функции кодирования адресов операндов. Байты 3 - 6 присутствуют в команде в зависимости от типа адреса операнда, описанного постбайтом и наличия непосредственного операнда.

Постбайт состоит из трех полей: *режима - MOD*, *регистра - REG* и *регистра/памяти - R/M*. Поле MOD занимает 2 бита (6 и 7) постбайта. Поле REG занимает 3 бита (3-5) постбайта. Поле R/M занимает 3 бита (0-2) постбайта. Полями MOD и R/M совместно кодируется тип адреса операнда, находящегося в памяти или регистре. 32 значения этих полей определяют нахождение операнда 24 возможными методами адресации либо в одном из 8 регистров.

Ниже приведена структура байта кода операции и постбайта, а в таблицах показано формирование адресов регистровых операндов и адресов операндов памяти. DISP8 и DISP16 – смещения длиной 8 и 16 бит, расположенные в команде непосредственно за байтом адресации.

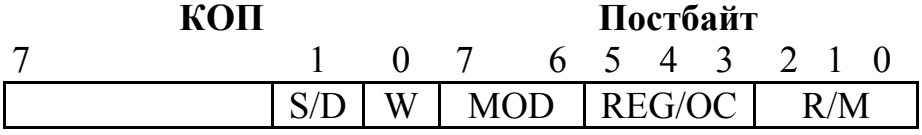


Рис. 2.7 Формат байта КОП и постбайта

Таблица 2 Формат поля REG

Поле REG	Регистры		Сегментные регистры
	W=1	W=0	
000	AX	AL	ES
001	CX	CL	CS
010	DX	DL	SS
011	BX	BL	DS
100	SP	AH	-

101	BP	CH	-
110	SI	DH	-
111	DI	BH	-

Таблица 3 Кодирование типа адреса операнда

Поле MOD	**Поле R/M**	**Базовый регистр**	**Эффективный адрес**	**Длина команды**
00	000	DS	[BX+SI]	2
00	001	DS	[BX+DI]	2
00	010	SS	[BP+SI]	2
00	011	SS	[BP+DI]	2
00	100	DS	[SI]	2
00	101	DS	[DI]	2
00	110	DS	DISP16	4
00	111	DS	[BX]	2
01	000	DS	DISP8+[BX+SI]	3
01	001	DS	DISP8+[BX+DI]	3
01	010	SS	DISP8+[BP+SI]	3
01	011	SS	DISP8+[DP+DI]	3
01	100	DS	DISP8+[SI]	3
01	101	DS	DISP8+[DI]	3
01	110	SS	DISP8+[BP]	3
01	111	DS	DISP8+[BX]	3
10	000	DS	DISP16+[BX+SI]	4
10	001	DS	DISP16+[BX+DI]	4
10	010	SS	DISP16+[BP+SI]	4
10	011	SS	DISP16+[BP+DI]	4
10	100	DS	DISP16+[SI]	4
10	101	DS	DISP16+[DI]	4
10	110	SS	DISP16+[BP]	4
10	111	DS	DISP16+[BX]	4
11	000	-	AX or AL	2
11	001	-	CX or CL	2
11	010	-	DX or DL	2
11	011	-	BX or BL	2
11	100	-	SP or AH	2
11	101	-	BP or CH	2
11	110	-	SI or DH	2
11	111	-	DI or BH	2

Команде может предшествовать префикс – байт со специальным кодированием, которое изменяет операцию следующей за ним команды. В системе команд процессоров 8086/8088 есть два таких префикса – REP (RE-

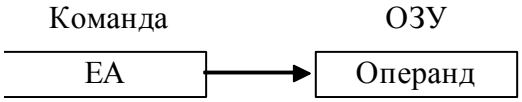
Peat) для повторения команд обработки строк и SEG (SEGment) для прямого указания команде сегментного регистра вместо регистра, используемого по умолчанию.

2.2.5 Методы адресации

Методы адресации можно разделить на два класса: адресация данных и адресация переходов. Все методы можно отнести к одной из следующих групп:

Прямая адресация.

16-битный эффективный адрес (*EA*) операнда является частью команды:



Пример: mov al, [0000h]

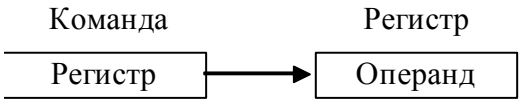
Код	Действие
a00000	Запись байта с адресом 0000h в регистр al

До выполнения	После выполнения
al=0	al=7
[0000]=7	[0000]=7 (Предварительно запишите 7 в DS:[0000])
ip=100	ip=103

Примечание: запись в квадратных скобках обозначает ячейку, адрес которой (смещение по отношению к DS) записан в квадратных скобках.

Регистровая адресация.

Операнд содержится в определяемом командой регистре. 16-битный операнд может находиться в регистрах AX, BX, CX, DX, SI, DI, SP или BP, а 8-битный - в регистрах AH, AL, BH, BL, CH, CL, DH, DL:



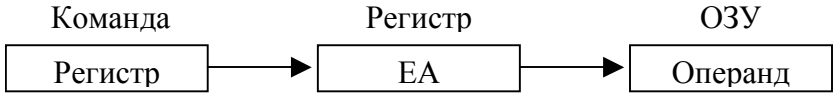
Пример: mov ax, bx

Код	Действие
8bc3	Запись содержимого регистра bx, в регистр ax

До выполнения	После выполнения
ax=7	ax=3
bx=3	bx=3
ip=100	ip=102

Косвенно-регистровая адресация.

Эффективный адрес операнда находится в базовом регистре BX или индексном регистре (SI или DI):



Пример: mov ax, [bx]

Код	Действие
8b07	Запись содержимого ячейки памяти с адресом из регистра bx (смещение внутри сегмента DS), в регистр ax

До выполнения	После выполнения
ax=3	ax=7
bx=0	bx=0
[0000]=7	[0000]=7 (Предварительно запишите 7 в DS:[0000])
ip=100	ip=102

Физический адрес определяется парой сегмент-смещение (например CS:IP – адрес следующей команды), и для каждого регистра, содержащего смещение, есть сегментный регистр, заданный по умолчанию. Некоторые сегменты разрешается принудительно переназначать, мнемоническое обозначение переназначения – «сегментный_регистр:смещение», в коде команды появляется дополнительный (первый) байт – байт замены сегмента.

Регистры, хранящие смещение	Сегмент «по умолчанию»	Возможная замена
IP	CS	-
SP	SS	-
BP	SS	CS, DS, ES
BP+SI, BP+DI	SS	CS, DS, ES
BX	DS	CS, DS, ES
SI, DI (кроме адресации строк)	DS	CS, DS, ES
SI (адресация строк)	DS	CS, DS, ES
DI (адресация строк)	ES	-

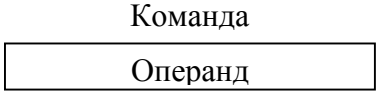
Пример: mov ax, cs:[bx]

Код	Действие
2e8b07	Запись содержимого ячейки памяти с адресом из регистра bx (смещение внутри сегмента CS), в регистр ax

До выполнения	После выполнения
ax=3	ax=5
bx=103	bx=103
cs=1554	cs=1554 (посмотрите на текущее значение CS у Вас)
[1554:103]=5	[1554:103]=5 (предварительно запишите [103]=5)
ip=100	ip=103

Непосредственная адресация.

Операнд длиной байт или слово является частью команды. Операнд помещается в последние байты команды, причем младший байт следует первым (находится по меньшему адресу).



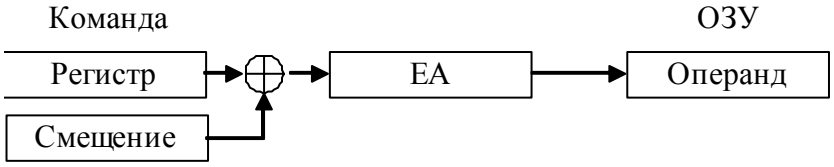
Пример: mov ax, 1234h

Код	Действие
b83412	Запись операнда, указанного в команде, в регистр ax

До выполнения	После выполнения
ax=5	ax=1234
ip=100	ip=103

Относительная косвенно-регистровая адресация.

Эффективный адрес операнда – сумма 8- или 16-разрядного смещения и значения одного из базовых или индексных регистров. Этот метод также называют *базовым*, если используются регистры BX, BP или *индексным*, при использовании SI, DI.



Базовая адресация обеспечивает возможность работы со структурами

данных, размещенными в памяти, например, с соседними ячейками памяти относительно эффективного адреса в базовом регистре.

Пример: mov ax, [bx+10], другая форма записи - mov ax, 10[bx]

Код	Действие
8b4710	Запись содержимого ячейки памяти с EA (сегмент DS), определяемым как сумма содержимого bx и смещения в команде (100+10=110) в регистр ax

До выполнения	После выполнения
ax=1234	ax=4
bx=100	bx=100
[110]=4	[110]=4 (Предварительно запишите 4 в DS:[0110])
ip=100	ip=103

Индексный метод адресации удобен при обработке массивов, когда смещение указывает стартовый адрес массива, а содержимое индексного регистра соответствует индексу массива.

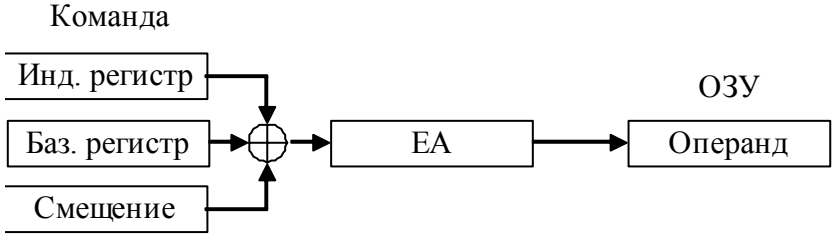
Пример: mov ax, 0100[si], другая форма записи - mov ax, [si+0100]

Код	Действие
8b840001	Запись содержимого ячейки памяти с EA (сегмент DS), определяемым как сумма содержимого si и смещения в команде (2+100=102) в регистр ax. Можно рассматривать значение si как индекс массива со стартовым адресом 100.

До выполнения	После выполнения
ax=4321	ax=7
si=2	si=2
[0102]=7	[0102]=7 (Предварительно запишите 7 в DS:[0102])
ip=100	ip=104

Базовая индексная адресация.

Эффективный адрес равен сумме содержимого базового (BX или BP) и индексного (SI или DI) регистров.



Пример: mov ax, 100[bx][si], другая форма записи - mov ax, [bx+si+0100]

Код	Действие
8b800001	Запись содержимого ячейки памяти с EA (сегмент DS), определяемым как сумма содержимого регистров si, bx и смещения в команде (10+2+100=112) в регистр ax. Можно рассматривать значение si как индекс выбранного регистром bx одномерного массива в двумерном массиве со стартовым адресом 100.

До выполнения	После выполнения
ax=7	ax=5
bx=10	bx=10
si=2	si=2
[0112]=5	[0112]=5 (Предварительно запишите 5 в DS:[0112])
ip=100	ip=104

Неявная адресация.

Неявная адресация задается операцией. Например, в командах обработки строк неявно используются регистры SI, DI. В командах управления циклами неявно используется регистр CX и т.д.

Стековая адресация.

Стековая адресация применяется в командах работы со стеком PUSH и POP. Для этих команд адрес операнда находится в указателе стека SP и автоматически уменьшается или увеличивается на два при записи в стек или при чтении из стека. Заполнение стека происходит в направлении уменьшения адресов ячеек памяти. Стек может обмениваться данными с регистрами общего назначения и сегментными регистрами. Команды обмена данными между стеком и памятью содержат байт адресации, в котором 3-разрядное поле REG, совместно с полем КОП, идентифицирует команду.

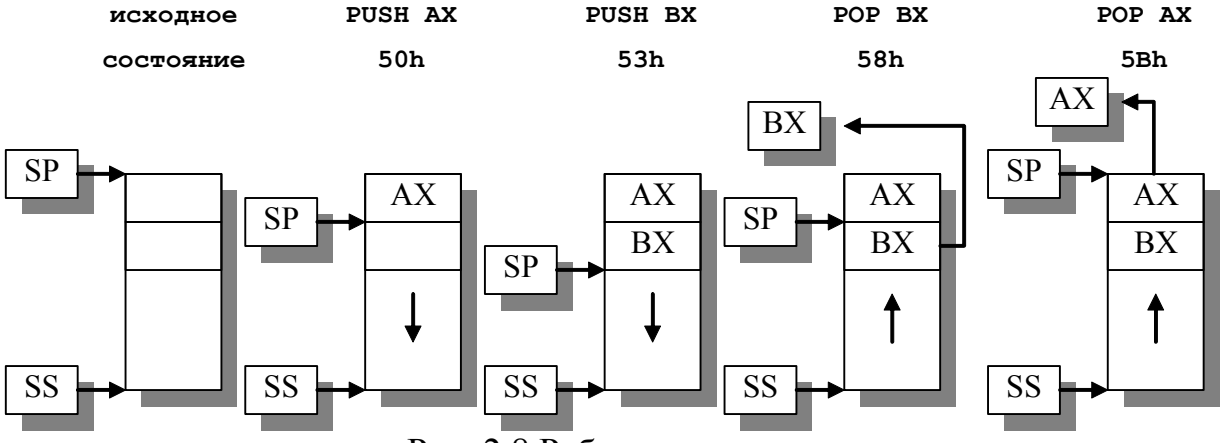
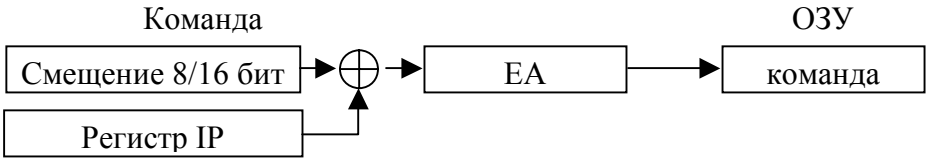


Рис. 2.8 Работа со стеком

Адресация переходов. Внутрисегментный прямой переход.

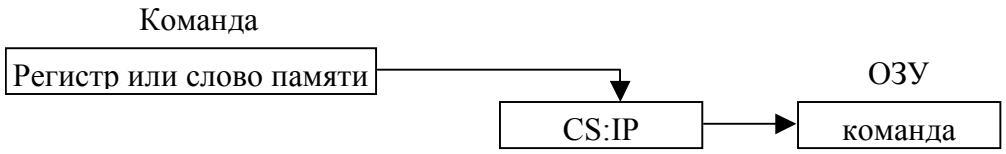
Адрес следующей команды определяется суммированием смещения со знаком, представленным в двоично-дополнительном коде, и значения IP:



Пример: jmp 100

Адресация переходов. Внутрисегментный косвенный переход.

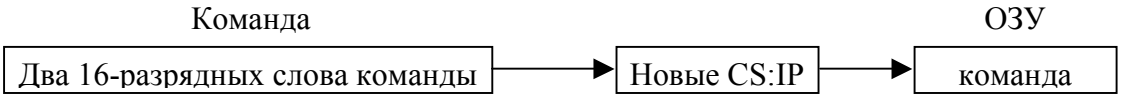
Содержимое регистра IP заменяется содержимым регистра или слова памяти, заданным любым методом адресации, кроме непосредственного:



Пример: jmp [bx]

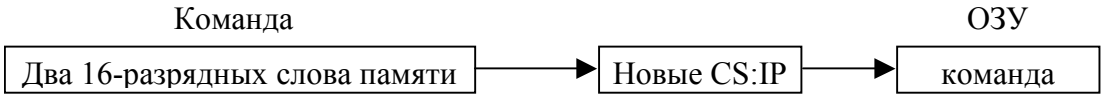
Адресация переходов. Межсегментный прямой переход.

Содержимое регистров CS и IP заменяются словами из команды:



Адресация переходов. Межсегментный косвенный переход.

Содержимое регистров CS и IP заменяется словами, последовательно расположенными в памяти. Слова могут быть указаны любым методом адресации, кроме непосредственного:



2.2.6 Система команд

Команды передачи данных

предназначены для пересылок данных между регистрами и памятью. Эту группу команд можно разделить на четыре подгруппы: команды пересылки данных общего назначения; команды, использующие аккумулятор; команды пересылки адресных объектов; команды пересылки флагов. Все команды этой группы не влияют на установку флагов, за исключением команд SAHF и POPF.

Пример: mov ax, bx

Код	Действие
8bc3	Пересылка 16-разрядного содержимого регистра bx в ax.

До выполнения	После выполнения
ax=0	ax=1234
bx=1234	bx=1234
ip=100	ip=102

Мне-моника	Действие	Байт КОП	Постбайт
MOV	Передача операнда из памяти/операнда регистра в регистр или из регистра в память/операнд регистра.	100010dw	mod reg r/m
	Передача непосредственного операнда в память/операнд регистра.	1100011w	mod 000 r/m
	Передача непосредственного операнда в регистр.	1011w reg	нет
	Передача операнда из памяти в аккумулятор.	1010000w	нет
	Передача операнда из аккумулятора в память.	1010001w	нет
	Передача операнда из памяти/регистра в сегментный регистр.	10001110	mod 0 reg r/m
	Передача операнда из сегментного регистра в память/регистр.	10001100	mod 0 reg r/m
XCHG	Обмен операндом из памяти/регистра с операндом из регистра.	1000011w	mod reg r/m
	Обмен операндом из регистра с операндом из аккумулятора.	10010reg	нет
PUSH	Передача операнда из памяти/операнда регистра по адресу в указателе стека SP.	11111111	mod 110 r/m
	Передача операнда из регистра по адресу в указателе стека SP.	01010reg	нет
	Передача операнда из сегментного регистра по адресу в указателе стека SP.	000reg110	нет
POP	Передача операнда из адреса в указателе стека SP в память/регистр.	10001111	mod 000 r/m

	Передача операнда из адреса в указателе стека SP в регистр.	01011reg	нет
	Передача операнда из адреса в указателе стека SP в сегментный регистр.	000reg11	нет
PUSHF	Передача содержимого регистра флагов по адресу в указателе стека SP.	10011100	нет
POPF	Передача содержимого из адреса в указателе стека SP в регистр флагов.	10011101	нет
LEA	Загрузка исполнительного адреса памяти в регистр общего назначения.	10001101	mod reg r/m
LDS	Загрузка из памяти относительного адреса (смещения) и адреса сегмента в один из регистров общего назначения и сегментный регистр DS.	11000101	mod reg r/m
LES	Загрузка из памяти относительного адреса (смещения) и адреса сегмента в один из регистров общего назначения и сегментный регистр ES.	11000100	mod reg r/m
LAHF	Передача младшего байта регистра флагов в регистр AH.	10011111	нет
SAHF	Передача содержимого регистра AH на место младшего байта регистра флагов.	10011110	нет
XLAT	Передача байта в регистр AL из 256-байтовой таблицы кодов.	11010111	нет
IN	Передача байта/слова из фиксированного порта ввода в аккумулятор AL/AX.	1110010w	нет
	Передача байта/слова из переменного порта ввода в аккумулятор AL/AX.	1110110w	нет

Арифметические команды

предназначены для выполнения четырех основных видов арифметических действий над 8- и 16- разрядными операндами в знаковом и беззнаковом представлении. Кроме основных арифметических команд имеются операции коррекции арифметических результатов для их перевода в упакованную или неупакованную форму. Признаки полученного результата отображаются в 6 битах регистра Flags (CF, AF, SF, ZF, PF и OF).

Пример: add ax, bx

Код	Действие
03c3	Сложение 16-разрядного содержимого регистров bx, ax и запись результата в ax.

До выполнения После выполнения

ax=2	ax=5
bx=3	bx=3
ip=100	ip=102

Мнемо-ника	Действие	Байт КОП	Постбайт
ADD	Сложение операнда из памяти/регистра с операндом из регистра.	000000dw	mod reg r/m
	Сложение непосредственного операнда с операндом из памяти/регистра.	100000sw	mod 000 r/m
	Сложение непосредственного операнда с операндом в аккумуляторе.	0000010w	нет
ADC	Сложение операнда из памяти/регистра с операндом из регистра (с учетом переноса).	0001010w	mod reg r/m
	Сложение непосредственного операнда с операндом из памяти/регистра (с учетом переноса).	100000sw	mod 010 r/m
	Сложение непосредственного операнда с операндом в аккумуляторе (с учетом переноса).	0001010w	нет
INC	Увеличение на единицу содержимого памяти/регистра.	1111111w	mod 000 r/m
	Увеличение на единицу содержимого регистра.	01000reg	нет
AAA	Коррекция содержимого аккумулятора AL при сложении двоично-десятичных кодов в неупакованном формате.	00110111	нет
DAA	Тоже, но в упакованном формате.	00100111	нет
SUB	Вычисление разности между операндом из памяти/регистра и операндом из регистра.	001010dw	mod reg r/m
	Вычитание непосредственного операнда из операнда в памяти/регистре.	100000sw	mod 101 r/m
	Вычитание непосредственного операнда из операнда в аккумуляторе.	0010110w	нет
SBB	Вычисление разности между операндом из памяти/регистра и операндом из регистра (с заемом).	000110dw	mod reg r/m

	Вычитание непосредственного операнда из операнда в памяти/регистре (с заемом).	100000sw	mod 011 r/m
	Вычитание непосредственного операнда из операнда в аккумуляторе (с заемом).	0001110w	нет
DEC	Вычитание единицы из операнда в памяти/регистре.	1111111w	mod 001 r/m
	Вычитание единицы из операнда в регистре.	01001reg	нет
NEG	Вычитание исходного операнда из нуля (изменение знака).	1111011w	mod 011 r/m
CMP	Сравнение операнда из памяти/регистра с операндом из регистра.	001110dw	mod 011 r/m
	Сравнение непосредственного операнда с операндом в памяти/регистре.	100000sw	mod reg r/m
	Сравнение непосредственного операнда с операндом в аккумуляторе.	0011110w	нет
AAS	Коррекция содержимого аккумулятора AL при вычитании двоично-десятичных кодов в неупакованном формате.	00111111	нет
DAS	То же, но в упакованном формате.	00101111	нет
MUL	Умножение без учета знака операнда в аккумуляторе на операнд из памяти/регистра.	1111011w	mod 100 r/m
IMUL	Умножение со знаком операнда из памяти/регистра.	1111011w	mod 101 r/m
DIV	Деление без учета знака операнда длиной в одно/два слова на операнд из памяти/регистра.	1111011w	mod 110 r/m
IDIV	Деление со знаком операнда длиной в одно/два слова на операнд из памяти/регистра.	1111011w	mod 111 r/m
AAM	Коррекция содержимого аккумулятора AX при умножении двоично-десятичных кодов в неупакованном формате.	11010100	00001010
AAD	Коррекция содержимого аккумулятора AL при делении двоично-десятичных кодов в неупакованном формате.	11010101	00001010

Логические команды

предназначены для выполнения четырех логических действий над 8- и 16-битовыми логическими структурами: получение инверсного кода, логическое произведение, логическая сумма, сумма по модулю два. Команды AND, TEST, OR и XOR воздействуют на арифметические флаги следующим образом: флаги OF и CF всегда сбрасываются в нулевое состояние; состояния флагов SF, ZF, PF зависят от полученного результата и определяются по тем же правилам, что и в командах арифметических операций; состояние флага AF не определено. Команда NOT не влияет на состояние флагов.

Пример: not ax

Код	Действие
F7D0	Значения 16 бит регистра ax меняются на противоположные.

До выполнения	После выполнения
ax=5555	ax=AAAA
ip=100	ip=102

Мнемоника	Действие	Байт КОП	Постбайт
NOT	Инвертирование разрядов операнда в памяти/регистре.	1111011w	mod 010 r/m
AND	Логическое умножение операнда из памяти/регистра и операнда из регистра.	001000dw	mod reg r/m
	Логическое умножение непосредственного операнда и операнда из памяти/регистра.	1000000w	mod 100 r/m
	Логическое умножение непосредственного операнда и операнда в аккумуляторе.	0010010w	нет
TEST	Установка регистра FLAGS в соответствии с результатом логического умножения операнда из памяти/регистра и операнда из регистра.	1000010w	mod reg r/m
	Установка регистра FLAGS в соответствии с результатом логического умножения непосредственного операнда и операнда из памяти/регистра.	1111011w	mod 000 r/m

	Установка регистра FLAGS в соответствии с результатом логического умножения непосредственного операнда и операнда в аккумуляторе.	1010100w	нет
OR	Логическое сложение операнда из памяти/регистра и операнда из регистра.	000010dw	mod reg r/m
	Логическое сложение непосредственного операнда и операнда из памяти/регистра.	1000000w	mod 001 r/m
	Логическое сложение непосредственного операнда и операнда в аккумуляторе.	0000110w	нет
XOR	Операция "исключающее ИЛИ" над операндами из памяти/регистра и из регистра.	001100dw	mod reg r/m
	Операция "исключающее ИЛИ" над непосредственным операндом и операндом из памяти/регистра.	1000000w	mod 110 r/m
	Операция "исключающее ИЛИ" над непосредственным операндом и операндом в аккумуляторе.	0011010w	нет

Команды сдвига

предназначены для выполнения логических, арифметических и циклических сдвигов. Поле операнда имеет формат mem/reg, count. Здесь mem/reg адресует регистр или ячейку памяти, а count (счет или счетчик) определяет число сдвигов. Число сдвигов может быть указано как константа 1 (статический сдвиг) или как регистр CL. В первом случае осуществляется сдвиг на один байт, а во втором - число сдвигов определяется содержимым регистра CL, которое должно быть беззнаковым целым двоичным числом. Таким образом, число сдвигов можно задать переменной, вычисляемой во время выполнения программы (так называемый динамический сдвиг). При выполнении команд сдвигов флаги изменяются следующим образом:

- состояние флага AF всегда не определено;
- флаг CF всегда содержит значение последнего выдвинутого бита;
- в однобитных сдвигах флаг OF=0, если операция изменила значение старшего (знакового) бита операнда; при сдвиге на несколько бит состояние флага OF не определено;
- циклические сдвиги влияют только на флаги OF и CF;
- в арифметических и логических сдвигах флаги SF, ZF и PF изменяются в соответствии с полученным результатом.

Пример: shl ax, 1

Код	Действие
D1E0	Поразрядный сдвиг 16-разрядного содержимого регистра ax на одну двоичную позицию влево, в сторону старших разрядов.

До выполнения	После выполнения
ax=1111	ax=2222
ip=100	ip=102

Мнемоника	Действие	Байт КОП	Постбайт
SHL, SAL	Логический (арифметический) сдвиг влево операнда из памяти/регистра на число разрядов, определяемое константой сдвига.	110100vw	mod 100 r/m
SHR	Логический сдвиг вправо операнда из памяти/регистра на число разрядов, определяемое константой сдвига.	110100vw	mod 101 r/m
SAR	Арифметический сдвиг вправо операнда из памяти/регистра на число разрядов, определяемое константой сдвига.	110100vw	mod 111 r/m
ROL	Циклический сдвиг влево операнда из памяти/регистра на число разрядов, определяемое константой сдвига.	110100vw	mod 000 r/m
ROR	Циклический сдвиг вправо операнда из памяти/регистра на число разрядов, определяемое константой сдвига.	110100vw	mod 001 r/m
RCL	Циклический сдвиг влево, с использованием CF, операнда из памяти/регистра на число разрядов, определяемое константой сдвига.	110100vw	mod 010 r/m
RCR	Циклический сдвиг вправо, с использованием CF, операнда из памяти/регистра на число разрядов, определяемое константой сдвига.	110100vw	mod 011 r/m

Команды передачи управления

включают в себя безусловные переходы, команды обращения к подпрограммам и возврата из них, а также команды управления циклами. Выполнение этих команд приводит к изменению содержимого указателя команд IP и регистра CS. Команды безусловных переходов и обращений к подпро-

граммам обеспечивают как внутрисегментные, так и межсегментные переходы с прямой и косвенной адресацией. Команды управления циклами обеспечивают переход только в области текущего сегмента с коротким смещением.

Пример: jmp 110

Код	Действие
EB0E	Загрузка регистра IP значением 110. Следующая команда имеет внутрисегментный адрес - 110.

До выполнения	После выполнения
ip=100	ip=110

Пример: call 110

Код	Действие
E80D000	Выполняется декремент SP на 2, в стек включается содержимое IP, а затем, к IP прибавляется смещение, которое интерпретируется, как знаковое целое.

До выполнения	После выполнения
ip=100	ip=110
sp=80	sp=7E
[80]=0000	[80]=0000
[7E]=0000	[7E]=0103

Мнемоника	Действие	Байт КОП	Постбайт
JMP	Безусловный внутрисегментный переход с прямой адресацией.	11101001	нет
	Безусловный короткий внутрисегментный переход с прямой адресацией.	11101011	нет
	Безусловный внутрисегментный переход с косвенной адресацией.	11111111	mod 100 r/m
	Безусловный межсегментный переход с прямой адресацией.	11101010	нет
	Безусловный межсегментный переход с косвенной адресацией.	11111111	mod 101 r/m
CALL	Внутрисегментный вызов подпрограммы с прямой адресацией.	11101000	нет

	Внутрисегментный вызов подпрограммы с косвенной адресацией.	11111111	mod 010 r/m
	Межсегментный вызов подпрограммы с прямой адресацией.	10011010	нет
	Межсегментный вызов подпрограммы с косвенной адресацией.	11111111	mod 011 r/m
RET	Внутрисегментный возврат из подпрограммы.	11000011	нет
	Внутрисегментный возврат из подпрограммы с увеличением содержимого указателя стека SP.	11000010	нет
	Межсегментный возврат из подпрограммы.	11001011	нет
	Межсегментный возврат из подпрограммы с увеличением содержимого указателя стека SP.	11001010	нет
LOOP	Передача управления, если содержимое регистра CX не равно нулю.	11100010	нет
LOOPE, LOOPZ	Передача управления, если содержимое регистра CX не равно нулю и флаг ZF установлен в 1.	11100001	нет
LOOPNE, LOOPNZ	Передача управления, если содержимое регистра CX не равно нулю и флаг ZF установлен в 0.	11100000	нет
JCXZ	Передача управления, если содержимое регистра CX равно нулю.	11100011	нет

Команды условных переходов

обеспечивают только внутрисегментные переходы.

Мнемоника	Действие	Байт КОП
JE, JZ	Передача управления по равенству/нулю.	01110100
JNE, JNZ	Передача управления, если не равно/не нуль.	01110101
JL, JNGE	Передача управления, если меньше/не больше и не равно.	01111100
JLE, JNG	Передача управления, если меньше или равно/не больше.	01111110
JB, JNAE	Передача управления, если меньше/не больше и не равно.	01110010
JBE, JNA	Передача управления, если меньше или равно/не больше.	01110110
JP, JPE	Передача управления при четности.	01111010

JNP, JPO	Передача управления при нечетности.	01111011
JO	Передача управления при переполнении.	01110000
JNO	Передача управления при отсутствии переполнения.	01110001
JS	Передача управления по отрицательному результату.	01111000
JNS	Передача управления по положительному результату.	01111001
JG, JNLE	Передача управления, если больше/не меньше и не равно.	01111111
JGE, JNL	Передача управления, если больше или равно/не меньше.	01111101
JA, JNBE	Передача управления, если больше/не меньше и не равно.	01110111
JAE, JNB	Передача управления, если больше или равно/не меньше.	01110011

Команды управления процессором

за исключением ESC являются однобайтовыми и могут быть разделены на две группы: команды, изменяющие содержимое регистра флагов, и команды, предназначенные для работы с внешними устройствами и не воздействующие на флаги.

Мнемоника	Действие	Байт КОП
CLC	Сброс признака переноса.	11111000
CMC	Инвертирование признака переноса.	11110101
STC	Установка признака переноса.	11111001
CDL	Сброс признака направления.	11111100
STD	Установка признака направления.	11111101
CLI	Сброс признака разрешения прерывания.	11111010
HLT	Останов.	11110100
WAIT	Перевод процессора в состояние ожидания.	10011011
ESC	Выдача кода операции или операнда для сопроцессора.	11011x
LOCK	Однобайтный префикс блокировки шины.	11110000

Команды обработки строк

Все команды обработки строк символов имеют длину один байт. Бит 0 показывает операцию с байтом (бит 0=0) или словом (бит 0=1).

Пример: movsw

Код	Действие
A5	Выполняется пересылка слова строки источника (src), адресуемой регистром SI, в строку приемника (dst), адресуемую регистром DI. Команда не устанавливает флагов.

До выполнения	После выполнения
ip=100	ip=101
si=20	si=22
di=50	di=52
[20]=0102	[20]=0102
[50]=0000	[50]=0102

Пример: rep movsw

Код	Действие
F3A5	Выполняется пересылка строки источника (src), адресуемой регистром SI, в строку приемника (dst), адресуемую регистром DI. На каждом шаге производится коррекция индексных регистров на требуемую величину и направление. Выполнение команды прекращается по исчерпании счетчика CX.

До выполнения	После выполнения
ip=100	ip=102
si=20	si=26
di=50	di=56
cx=3	cx=0
[20]=0102	[20]=0102
[22]=0304	[22]=0304
[24]=0506	[24]=0506
[50]=0000	[50]=0102
[52]=0000	[52]=0304
[54]=0000	[54]=0506

Мнемоника	Действие	Байт КОП
REP	Циклическое повторение команды обработки строки, количество повторов – в CX. (*Префикс*)	1111001z
MOVSB, MOVSW	Передача элемента строки - байта/слова.	1010010w

CMPSB, CMPSW	Сравнение элементов строк - байтов/слов.	1010011w
SCASB, SCASW	Сканирование строк символов - байтов/слов.	1010111w
LODSB, LODSW	Загрузка элементов строк символов - байтов/слов в регистр AL/AX.	1010110w
STOSB, STOSW	Запись байтов/слов из регистра AL/AX в строку символов.	1010101w

2.2.7 Математический сопроцессор

Математический сопроцессор (MCP - math coprocessor) – это расширение основной архитектуры и множества команд основного процессора. Сопроцессор дополняет возможности процессора новыми командами для работы с вещественными числами и новыми регистрами и реализован в виде отдельной ИС для процессоров до 80386 включительно (8087, 80287, 80387), либо непосредственно в микросхеме центрального процессора, как внутренний модуль (Floating Point Unit, FPU). Сопроцессор поддерживает семь типов данных: 16-, 32-, 64-битные целые числа; 32-, 64-, 80-битные вещественные числа и 18-разрядные числа в двоично-десятичном формате. Форматы чисел с плавающей точкой соответствуют стандартам IEEE 754, 854 и представлены на Рис. 2.9, где D_i – разряды десятичного числа в двоично-десятичном представлении; M – мантисса, E – порядок вещественного числа, а S – знаковый разряд.

Декодирование инструкций для сопроцессора и доставка данных осуществляется основным процессором, сопроцессор только исполняет свои команды. Для хранения операндов и промежуточных данных имеется восемь 80-разрядных регистров данных R0-R7, в которых данные представлены в *расширенном вещественном* формате (см. Рис. 2.9). При загрузке регистра из памяти, данные автоматически преобразуются в этот формат. Регистры данных образуют стек, обращение к которому возможно через относительные имена ST(N). Пять регистров специального назначения служат для управления вычислениями и определения состояния сопроцессора (Рис. 2.10).

Слово состояния (SW) состоит из флагов (начиная с 0-го разряда): IE – недействительная команда (пустой операнд, неопределенный результат), DE – денормализован хотя бы один из операндов, ZE – деление на нуль, OE - переполнение, UE - антипереполнение, PE – выполнено округление, ES (другое обозначение - IR) – возникновение немаскируемого исключения, C[0:3] – код условия, интерпретируемый в зависимости от выполненной команды, ST[0:2] (Stack Top) – номер регистра из стека, выполняющего роль вершины стека.

Управляющее слово (CW) - определяет условия выполнения команд. Разряды 0..5 маскируют исключения: IM – неверная команда, DM – операнд денормализован, ZM – деление на нуль, OM - переполнение, U M -

антипереполнение, PM – округление. Поле PC (биты 8,9) – управление точностью: 00 – 24 бита, 10 – 53 бита, 11 – 64 бита. Поле RC (биты 10,11) – управление округлением: 00 – к ближайшему значению, 01 – к -∞,10 – к +∞, 11 – к нулю.

Слово тэгов (TW) - состоит из восьми двухбитных признаков (тэгов) для каждого из регистров данных сопроцессора. Четыре возможных значения тэга: 00, 01, 10, 11, определяют содержимое регистра как, соответственно, ненулевое число, нуль, специальное значение (бесконечность, неизвестный формат, и т.п.)

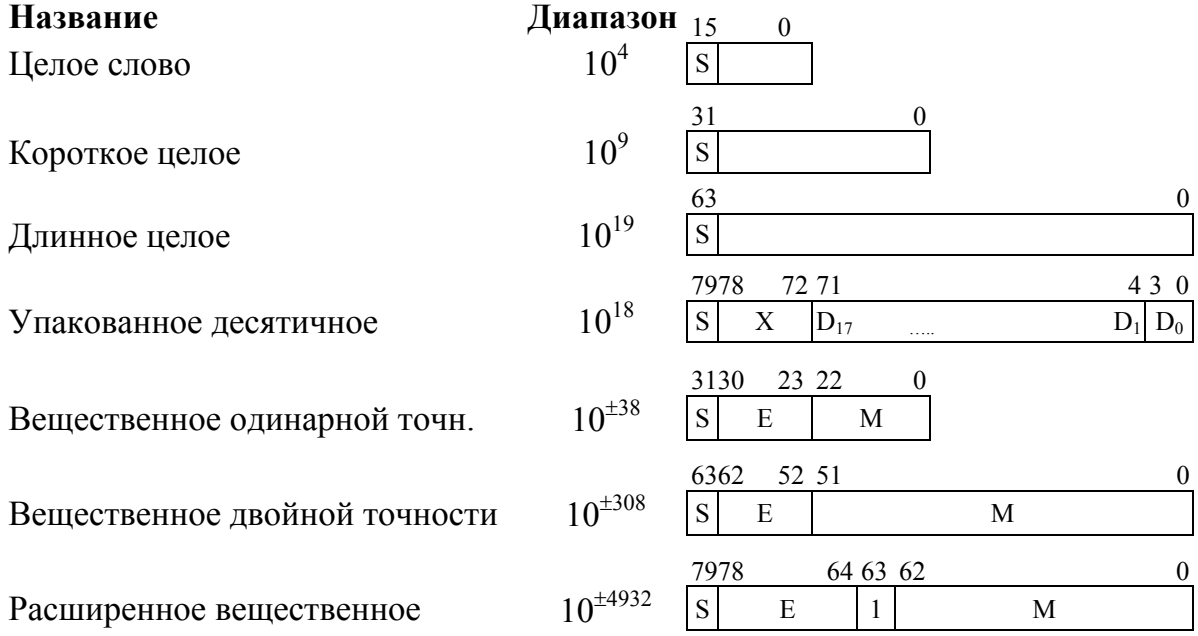


Рис. 2.9 Форматы данных сопроцессора

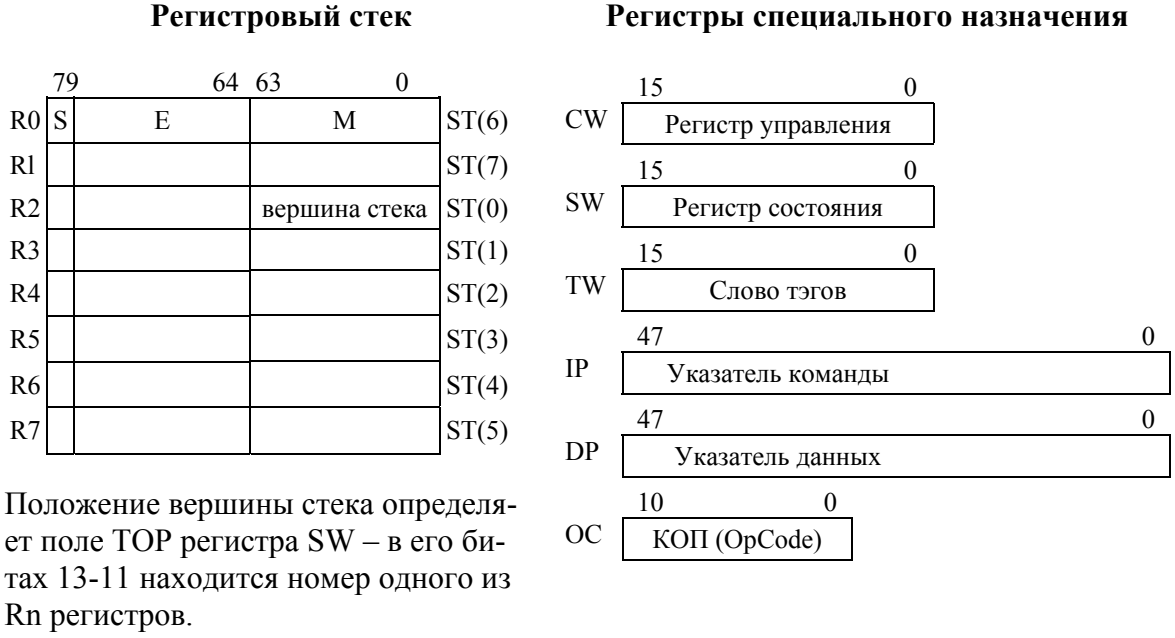


Рис. 2.10 Регистры сопроцессора

Система команд сопроцессора состоит из команд передачи данных и за-

грузки констант, основных арифметических команд, сравнения, управления сопроцессором. Все команды сопроцессоры относятся к классу т.н. ESC- команд и начинаются с битовой последовательности 11011 ($D1_{16}$). Ниже приведены ассемблерная мнемоника и краткое описание действий команд. Операнды обозначены следующим образом:

- m32real, m64real, m80real – вещественные числа в памяти;
- m16int, m32int, m64int – целочисленные значения в памяти;
- m80bcd – упакованное двоично-десятичное число;
- st(i) - регистр стека сопроцессора, смещенный от верхушки на i;
- reg/mem – операнд в регистре/памяти.

Команды передачи данных и загрузки констант

FLD src	st(0) ← src (m32real/m64real/m80real)
FILD src	st(0) ← src (m16int/m32int/m64int)
FBLD src	st(0) ← src (m80bcd)
FLDZ	st(0) ← 0.0
FLD1	st(0) ← 1.0
FLDPI	st(0) ← π
FLDL2T	st(0) ← $\log_2(10)$
FLDL2E	st(0) ← $\log_2(e)$
FLDLG2	st(0) ← $\log_{10}(2)$
FLDLN2	st(0) ← ln(2)
FST dest	dest ← st(0) (m32real/m64real)
FSTP dest	dest ← st(0) (m32real/64/80); “вытолкнуть” стек
FIST dest	dest ← st(0) (m16int/32)
FISTP dest	dest ← st(0) (m16int/32/64); “вытолкнуть” стек
FBST dest	dest ← st(0) (m80bcd)
FBSTP dest	dest ← st(0) (m80bcd); “вытолкнуть” стек

Команды сравнения

FCOM	Установить флаги как для st(0) - st(1)
FCOM op	Установить флаги как для st(0) - op (m32real/64)
FCOMP op	Сравнить st(0) с op (reg/m32real/64); “вытолкнуть” стек
FCOMPP	Сравнить st(0) с st(1); “вытолкнуть” стек 2 раза
FICOM op	Установить флаги как для st(0) - op (m16int/m32int)
FICOMP op	Сравнить st(0) с op (m16int/m32int); “вытолкнуть” стек
FTST	Сравнить st(0) с 0.0
FXAM	Установить флаги по содержимому st(0)
FUCOMP st(i)	Сравнить st(0) с st(i); “вытолкнуть” стек
FUCOMPP st(i)	Сравнить st(0) с st(i); “вытолкнуть” стек 2 раза

Арифметические команды и команды трансцендентных функций

FADD	st(0) ← st(0) + st(1)

FADD src	st(0) ← st(0) + src (m32real/m64real)
FADD st(i),st	st(i) ← st(i) + st(0)
FADDP st(i),st	st(i) ← st(i) + st(0); “вытолкнуть” стек
FIADD src	st(0) ← st(0) + src (m16real/m32real)
FSUB	st(0) ← st(0) - st(1)
FSUB src	st(0) ← st(0) - src (reg/mem)
FSUB st(i),st	st(i) ← st(i) - st(0)
FSUBP st(i),st	st(i) ← st(i) - st(0); “вытолкнуть” стек
FSUBR st(i),st	st(0) ← st(i) - st(0)
FSUBRP st(i),st	st(0) ← st(i) - st(0); “вытолкнуть” стек
FISUB src	st(0) ← st(0) - src (m16int/m32int)
FISUBR src	st(0) ← src - st(0) (m16int/m32int)
FMUL	st(0) ← st(0) * st(1)
FMUL st(i)	st(0) ← st(0) * st(i)
FMUL st(i),st	st(i) ← st(0) * st(i)
FMULP st(i),st	st(i) ← st(0) * st(i); “вытолкнуть” стек
FIMUL src	st(0) ← st(0) * src (m16int/m32int)
FDIV	st(0) ← st(0) ÷ st(1)
FDIV st(i)	st(0) ← st(0) ÷ st(i)
FDIV st(i),st	st(i) ← st(0) ÷st(i)
FDIVP st(i),st	st(i) ← st(0) ÷ st(i); “вытолкнуть” стек
FIDIV src	st(0) ← st(0) ÷src (m16int/m32int)
FDIVR st(i),st	st(0) ← st(i) ÷ st(0)
FDIVRP st(i),st	st(0) ← st(i) ÷ st(0); “вытолкнуть” стек
FIDIVR src	st(0) ← src ÷ st(0) (m16int/m32int)
FSQRT	st(0) ← квадратный корень от st(0)
FSCALE	st(0) ← масштабировать на степень 2-ки st(0)
FXTRACT	st(0) ← порядок st(0); втолкнуть в стек; st(0) ← мантисса st(0)
FPREM	st(0) ← st(0) MOD st(1)
FRNDINT	st(0) ← INT(st(0)); зависит от флага RC
FABS	st(0) ← ABS(st(0)); убрать знак
FCHS	st(0) ← -st(0)
FCOS	st(0) ← COS(st(0))
FPTAN	st(0) ← TAN(st(0))
FPATAN	st(0) ← ATAN(st(0))
FSIN	st(0) ← SIN(st(0))

Команды управления сопроцессором

FINIT	Инициализировать сопроцессор
FSTSW AX	AX ← MSW
FSTSW dest	dest ← MSW (m16int)
FLDCW src	FPU CW ← src (m16int)
FSTCW dest	dest ← FPU CW

FCLEX	Очистить исключения
FSTENV dest	записать слова состояния, управления, тэгов и указатели исключений в память – dest
FLDENV src	загрузить окружение с src
FSAVE dest	записать состояние сопроцессора в 94-байтное dest
FRSTOR src	восстановить состояние сопроцессора, сохраненное FSAVE
FINCSTP	st(6)←st(5); st(5)←st(4),...,st(0)←?
FDECSTP	st(0)←st(1); st(1)←st(2),...,st(7)←?
FFREE st(i)	Отметить регистр st(i) как неиспользуемый
FNOP	st(0) ← st(0), холостая команда
WAIT/FWAIT	Остановить ЦПУ, пока сопроцессор не завершил выполнение текущей команды

Покажем, как используется стек сопроцессора и его команды для выполнения последовательности вычислений на примере вычисления скалярного произведения (Рис. 2.11). Вначале разместим исходные данные в сегменте данных DS последовательно (1.2; 3.4; 5.6; 7.8) в формате float. Для этого изменим представление данных сегмента DS как данных с плавающей точкой (Ctrl^D → Float).

1. Перед работой со стеком сопроцессора, выполним сброс FINIT.
2. Вторая команда FLD dword ptr [bp] выполняет декремент указателя стека и загружает значение 1.2 из DS:BP в ST(0).
3. Третья команда умножает значение ST(0) на операнд 3.4 из DS:[BP+4] и записывает результат в ST(0).
4. Четвертая команда выполняет декремент указателя стека и загружает значение 5.6 из памяти DS:[BP+8] в ST(0).
5. Пятая команда умножает значение ST(0) на значение 7.8 из DS:[BP+0C] и записывает результат в ST(0).
6. Шестая команда складывает значения ST(1) и ST(0) и записывает результат 47.76 в ST(0).

```
Вычисление скалярного произ-
ведения (1.2x3.4)+(5.6x7.8)
```

```
FINIT
FLD  dword ptr [bp]       ; (a)
FMUL dword ptr [bp+04]; (b)
FLD  dword ptr [bp+08]  ;
FMUL dword ptr [bp+0c]; (c)
FADD ST(1),ST             ; (d)
```

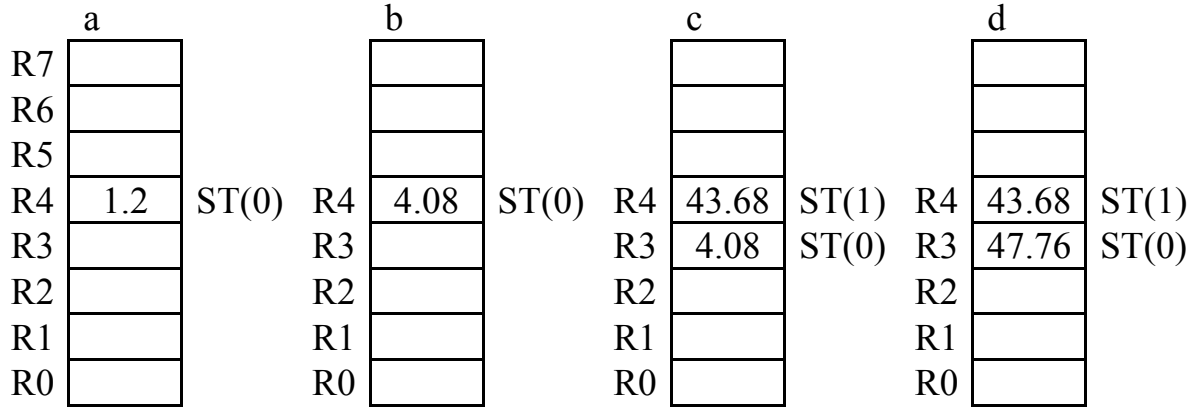


Рис. 2.11 Пример программы для сопроцессора

2.2.8 Организация ввода-вывода

Работа с портами ввода/вывода в процессорах x86 возможна как с использованием специальных команд (IN, OUT), через отдельное адресное пространство ввода/вывода, так и по схеме с отображением регистров устройств на обычное адресное пространство. В последнем случае возможно использование обычных команд из системы команд процессора.

Для адресации портов устройств в пространстве ввода/вывода, используется 16-разрядный адрес, обеспечивая доступ к 64K 8-битным портам с адресами от 0h до FFFFh. Адреса 0F8h – 0FFh– зарезервированы для системных целей. Порты с адресами 0h – 0FFh используются оборудованием системной платы ПК (таймер, контроллер прерываний и т.п.), адреса 0100h–03FFh используют различные контроллеры: дисков, видеомонитора, компьютерной сети.

Команды IN и OUT работают с прямой адресацией (адрес порта находится в команде) для портов с адресами 0h - 0F7h и с косвенной адресацией (адрес порта находится в регистре DX) для любых портов: 0h - FFFFh. Обмен данными происходит только через регистр-аккумулятор (AX, AL), например, чтение из порта: **`in al,dx`** , запись в порт: **`out dx,ax`**

Пример работы с устройством в/в (программируемый таймер 8254)

Для задания временных интервалов и формирования сигналов с различными временными параметрами в ПК на основе процессоров Intel x86 применяется программируемый таймер i8254. В состав таймера входят: буфер шины данных, схема управления вводом-выводом и три независимых канала, каждый из которых содержит *регистр режима*, *схему управления каналом*, *буфер* и 16-разрядный *счетчик*. В современных ПК, таймер реализован не в виде отдельной ИС, а в СБИС чипсета.

Программирование канала осуществляется путем записи управляющих слов в регистр режима каналов и начального значения в его счетчики. Каждый канал имеет управляющий вход GATE и выход OUT и может работать в одном из шести режимов. Частота CLK = 1193181 Гц.

Временные диаграммы одного из режимов таймера (режим 3) приведены на следующем рисунке:

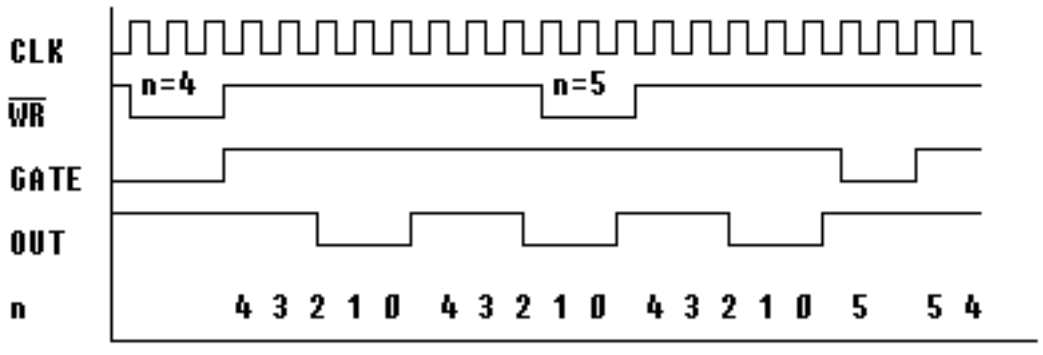


Рис. 2.12 Временная диаграмма 3-го режима таймера

В IBM PC таймер имеет базовый адрес 40h и следующие программируемые регистры:

Адрес	Операция	Назначение
40h	запись чтение	Загрузка счетчика канала 0 Чтение счетчика канала 0
41h	запись чтение	Загрузка счетчика канала 1 Чтение счетчика канала 1
42h	запись чтение	Загрузка счетчика канала 2 Чтение счетчика канала 2
43h	запись	Запись управляющего слова в регистр режима канала

Управляющее слово (порт 43h) имеет следующий формат:

бит 0, представление	биты 1-3, режим работы	биты 4-5, вид загрузки счетчика	биты 6-7, номер канала:
0 - двоичный 1 - двоично-десятичный	000 - режим 0 001 - режим 1 X10 - режим 2 X11 - режим 3 100 - режим 4 101 - режим 5	00 - "защелкивание" 01 - только младший байт 10 - только старший байт 11 - младший байт, затем старший	00 - канал 0 01 - канал 1 10 - канал 2 11 - запрещено

В IBM PC каналы таймера обычно имеют следующее назначение:

Канал	Назначение	Режим
0	системные часы , IRQ0	3, счетчик=0 (65536)
1	запрос для канала 0 ПДП (регенерация памяти)	2, счетчик=18
2	генератор звука	

Тактовая частота каждого канала равна 1,19318 МГц, т.о. каждый такт имеет длительность 0,84 мсек. Вход GATE каналов 0 и 1 всегда имеют высокий уровень, поэтому счет на этих каналах разрешен всегда. Вход GATE канала 2 управляется битом 0 порта 61h.

При начальной загрузке BIOS инициализирует канал 0 для работы в режиме 3 со счетчиком 0 (65536 декрементов на цикл счета). Поэтому частота системных часов равна 1,19 МГц/65536 = 18.2 Гц и прерывание IRQ0, связанное с вектором Int 8, происходит 18,2 раз в секунду, т. е. каждые 55 мсек. BIOS также обновляет свой таймер по адресу 0000:046Ch.

Канал 1 работает в режиме 2 со счетчиком 18, поэтому регенерация памяти происходит каждые 18 мсек. Перепрограммировать его нежелательно. Программирование канала 2 может быть использовано для генерации звука.

На вход звукогенератора поступает логическое «И» двух сигналов: выхода OUT 2-го канала таймера и содержимого бита 1 порта 61h. Поэтому, простейший способ генерации звука состоит в программировании канала 2 таймера так, чтобы он вырабатывал прямоугольный импульс заданной частоты, лежащий в звуковом диапазоне 20 Гц - 20 кГц (учитывая качество динамика ПК 500-5000Гц). Для этого следует использовать режим таймера 3 с подходящим начальным значением счетчика. Если затем установить биты 0 и 1 порта 61h, то импульс начнет поступать на вход звукогенератора (бит 0 - это вход GATE канала 2, разрешающий счет, а бит 1 - разрешение выдачи выхода OUT на вход звукогенератора). Для выключения звука достаточно сбросить биты 0 - 1 в 61h. Значение счетчика 2-го канала вычисляется по формуле n=1193181/f (1193181 - тактовая частота таймера в Гц, f - требуемая частота звука). Обратите внимание, что т.о. будет получено значение для счетчика в *десятичной* системе.

Пример генерации звука (обязательно внимательное прочтение предыдущего описания работы таймера!)

Для работы с регистрами таймера можно воспользоваться программой debug. Эта программа операционной системы DOS, поставляется и во всех последующих ОС MS в рамках эмуляции DOS. Запуск программы можно выполнить так: Start → Run →debug →OK. Полный перечень команд debug можно получить набрав вместо команды знак вопроса. Нам же понадобятся две команды этой программы: i – для чтения из порта ввода/вывода и o – для записи в порт. Чтобы проверить возможности генерации звука на данном ПК, нужно выполнить следующие команды:

```
-o 61 3      Разрешение выхода таймера (OUT) и разрешение счета (GATE)
-o 43 A6     Установка режима 3 для канала 2 (управляющее слово = 86)
-o 42 1      Запись старшего байта счетчика = 1
-o 42 2      Запись старшего байта счетчика = 2 (на октаву вниз)
-o 42 4      Запись старшего байта счетчика = 4 (на октаву вниз)
-o 42 8      Запись старшего байта счетчика = 8 (на октаву вниз)
-o 42 10     Запись старшего байта счетчика = 10 (на октаву вниз)
-o 61 0      Выключение выхода и счета
-quit        Выход из программы debug
```

В данном примере в регистр счетчика загружался только старший байт (управляющее слово, записанное в порт 43h, имело значение A6h). Значения старшего байта счетчика[1; 2; 4; 8; 10], с учетом нулевого младшего байта, в десятичной системе дадут [256; 512; 1024; 2048; 4096]. Соответствующие значения частоты на втором канале таймера будут F=1193180/счетчик = [4660; 2330; 1165; 582; 291] Гц.

2.2.9 Организация прерываний

Прерывания - *приостановление основных вычислений в процессоре, для выполнения вспомогательных действий, технологических сервисных процедур* – можно разделить на два класса: внешние и внутренние. Каждо-

му прерыванию назначен четырехбайтный вектор с номером в диапазоне 0h - FFh. Векторы определяют адреса подпрограмм обслуживания прерывания (ISR, Interrupt Service Routine) и размещаются в младшем килобайте адресного пространства памяти в виде IP:CS (сегмент_ISR:смещение_ISR, смещение записывается по младшему адресу).

Внешние прерывания вызываются сигналами на входах (см. Рис. 2.13) запроса немаскируемого прерывания процессора (NMI) и маскируемого прерывания (INTR). Маскируемые прерывания игнорируются, если очищен флаг I (Рис. 2.5).

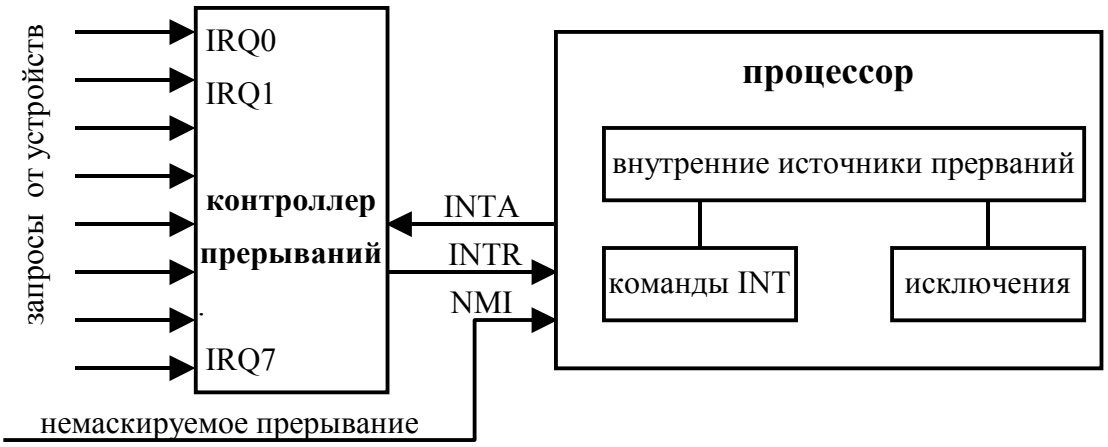


Рис. 2.13 Источники прерываний

После завершения выполнения текущей команды процессор вырабатывает сигналы подтверждения (INTA), получает по шине данных номер прерывания от контроллера прерываний, сохраняет в стеке текущее содержимое Flags, CS, IP и приступает к выполнению ISR, загружая содержимое вектора в CS, IP. В начале выполнения ISR процессор очищает флаг I, запрещая дальнейшие прерывания. Если необходимо, в подпрограмме ISR можно разрешить дальнейшие прерывания, установив флаг I (с помощью команды STI). Подпрограмма ISR должна заканчиваться командой IRET, которая загружает старые значения IP, CS, Flags из стека и т.о. возобновляет работу прерванной программы.

Запросы от различных устройств в ПК обрабатываются контроллером прерываний, который формирует сигнал INTR для процессора (Рис. 2.13). Обычно это две микросхемы Intel8259, каждая из которых имеет 8 входов прерываний. Запросам прерываний IRQ0-IRQ7 ПК соответствуют векторы от 08h до 0Fh, а IRQ8-IRQ15 – векторы от 070h до 077h. Битовые маски, позволяющие запрещать (маскировать) отдельные входы запросов прерываний, находятся по адресам 021h и 0A1h для IRQ0-IRQ7 и IRQ8-IRQ15 соответственно.

К внутренним прерываниям относятся т.н. *исключения* – различного рода ошибки, механизм для отлаживания программ, аварийные состояния и командные прерывания INT:

- вектор 0 – возникает при ошибке деления;

- вектор 1 – отладочное, возникает при установленном TF;
- вектор 3 – вызывается однобайтной командой int 3;
- вектор 4 – вызывается однобайтной командой into, при установленном OF;
- вектор > 4 – вызывается командой int n, где n – номер вектора.

Вход для немаскируемых прерываний с вектором 2 используется для подачи сигналов о серьезных аварийных состояниях, требующих немедленного обслуживания, например, об отсутствии напряжения в сети.

2.2.10 Базовая система ввода-вывода BIOS

BIOS (Basic Input-Output System) - набор подпрограмм, записанный в запоминающее устройство (ПЗУ или флеш), предназначен для начальной инициализации ПК, выполнения программы настройки оборудования ПК Setup, загрузке операционной системы и совместной работы с операционной системой DOS. Прерывания BIOS - это механизм выполнения стандартных операций ввода/вывода, предоставляемый операционной системе и программе Setup. После получения инструкции INT процессор, после обычных процедур со стеком, определяет по номеру прерывания в таблице векторов прерываний адрес операции, который находится в адресном пространстве ПЗУ BIOS, и приступает к ее выполнению. Выполнение продолжается до команды IRET, после чего процессор возвращается к адресу, сохраненному в стеке, и восстанавливает все флаги. Параметры процедурам BIOS обычно передаются через регистры A, B, C и D.

Т.о. *для выполнения той или иной процедуры BIOS необходимо записать ее параметры в регистры и затем выполнить команду Int* с кодом, соответствующим процедуре.

Ниже приводятся коды прерываний и значения регистров (параметров) для некоторых наиболее распространенных процедур работы с видеоконтроллером (Int10), коммуникационными (Int14) и параллельными портами (Int17), манипулятором «мышь» (Int10):

- Int10 - Установка видеорежима. Аргументы: ah=00, al= номер режима. Режимы 0-3 - 16-цветный текст, режимы 4-6 - 4-х цветная графика, режим 7 -монохромный текст, режимы 8-18 зависят от типа видеоадаптера, режим 19 - 256-ти цветная графика.
- Int10 - Установка размера курсора. Аргументы: ah=01, ch=начальная линия, cl=конечная линия курсора.
- Int10 - Установка позиции курсора. Аргументы: ah=02, bh=номер видеостраницы, dh=строка курсора, dl=столбец курсора. Главная видеостраница = 0
- Int10 - Считывание положения и размера курсора. Аргументы: ah=03, bh=номер видеостраницы. Выход: bh=номер видеостраницы,

ch=начальная линия, cl=конечная линия, dh=строка курсора, dl=столбец курсора.

- Int10 - Выбор активной видеостраницы. Аргументы: ah=05, al=номер страницы.
- Только для текстового режима, номера страниц обычно 0-7.
- Int10 - Чтение знака и его атрибутов. Аргументы: ah=08, bh=номер видеостраницы. Выход: ah=байт атрибутов, al=ASCII код буквы. Атрибуты дисплея = однобайтовое число, старшие биты - цвет фона, младшие биты - цвет знака, цвета от 0 до F.
- Int10 - Запись знака и его атрибутов. Аргументы: ah=09, al=ASCII код знака, bh=номер видеостраницы, lb=байт атрибутов (как указано выше), cx=число знаков для вывода на экран.
- Int10 - Запись знака. Аргументы: ah=0A, al=ASCII код знака, bh=номер видеостраницы, bl=цвет, cx=число знаков для вывода на экран.
- Int10 - Запись точки (1 пиксел). Аргументы: ah=0C, al=значение пиксела, cx=столбец, dx=ряд.
- Int10 - Чтение точки (1 пиксел). Аргументы: ah=0D, cx=столбец, dx=ряд. Выход: al=значение пиксела, cx=столбец, dx=ряд.
- Int10 - Информация о видеорежиме. Аргументы: ah=0F. Выход: ah=ширина экрана, al=режим отображения, bh=активная видеостраница.
- Int10 - Запись строки. Аргументы: ah=13, al=видеорежим, bh=номер видеостраницы, bl=атрибуты строки (см. выше), cx=длина строки, dh=строка курсора, dl=столбец курсора, es=сегмент, bp=смещение. Примечание: ES:BP=адрес строки.

- Int14 - Инициализация коммуникационного порта. Вход: ah=00, al=параметр, dx=номер COM порта (начиная с 0, для COM1). Выход: ah=статус линии, al=статус модема.
- Int14 - Передача символа. Вход: ah=01, al=ASCII символ, dx=номер COM порта.
- Int14 - Прием символа. Вход: ah=02, dx=номер COM порта. Выход: ah=возвратный код, al=принятый символ.
- Int14 - Статус COM порта. Вход: ah=03, dx=номер COM порта. Выход: ah=статус линии, al=статус модема.

- Int17 - Печать символа. Вход: ah=00, al=символ, dx=принтер. Выход: ah=статус принтера. Примечание: принтер=0 (LPT1) до 2 (LPT3).
- Int17 - Инициализация принтера. Вход: ah=01, dx=принтер. Выход: ah=статус принтера.
- Int17 - Статус принтера. Вход: ah=02, dx=принтер. Выход: ah=статус принтера.

- Int33 - Сброс манипулятора мышь. Вход: ax=00. Выход: ax=статус.
- Int33 - Показать указатель мыши. Вход: ax=01.

- Int33 - Скрыть указатель мыши. Вход: ax=02.
- Int33 - Определить положение указателя. Вход: ax=03. Выход: bx=статус клавиш: 0 разряд – левая, 1 разряд – правая, 2 разряд – средняя; cx=координаты по оси X (по горизонтали); dx - координаты по оси Y (по вертикали).

2.3 32-разрядные процессоры

IA32 – так по классификации фирмы Intel [3] обозначается архитектура 32-разрядных процессоров Intel от процессора 80386 до 80686. Процессоры IA32 могут работать в двух основных режимах: Real Address Mode (реальный режим) и Protected Virtual Address Mode (защищенный режим). В реальном режиме IA32 полностью совместимы с 8086, имеют аналогичные регистровый файл, адресное пространство, механизм сегментации памяти. В защищенном режиме появляются новые возможности:

- дополненный регистровый файл;
- новые механизмы сегментации и страничной адресации;
- режим виртуального процессора 8086 - Virtual 8086 Mode (V86);
- добавления к системе команд.

В большинстве моделей IA32 введен еще один режим – System Management Mode (режим системного управления) для отладочных целей и для реализации функции энергосбережения.

Набор регистров общего назначения включает в себя регистры 16-разрядных процессоров 8086/8088, однако они имеют разрядность 32 бита. К обозначениям регистров добавлена приставка E (Extended - расширенный): EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP. Возможно обращение к младшим 16 разрядам расширенных регистров по именам без приставки, например, AX, BX, а также отдельно к младшим и старшим байтам, например, AL, AH.

Регистр состояния процессора FLAGS расширен до 32 разрядов и обозначается – EFLAGS. Флаги 16-разрядных процессоров с 0 по 11 разряд занимают такое же положение в EFLAGS. Добавлены новые флаги:

- ID Flag (ID, 21 бит) – Если возможно программно устанавливать и сбрасывать этот флаг, процессор поддерживает команду CPUID.
- Virtual Interrupt Pending (VIP, 20 бит) – Указывает на то, что остались прерывания, ожидающие обработку. Устанавливается и сбрасывается программно, процессор только считывает его значение.
- Virtual Interrupt Flag (VIF, 19 бит) – Образ флага IF для режима V86.
- Alignment Check (AC, 18 бит) – Флаг контроля выравнивания. При установке этого флага во время обращении к невыровненному операнду возникает исключение.

- Virtual-8086 Mode (VM, 17 бит) – Включает/выключает режим V86 в защищенном режиме.
- Resume Flag (RF, 16 бит) – Флаг возобновления исполнения при отладке.
- Nested Task (NT, 14 бит) – Флаг вложенной задачи. Устанавливается, когда текущая задача связана с прерванной задачей, очищается, если такой связи нет.
- I/O Privilege Level (IOPL, 12-13 биты) – определяет уровень привилегий ввода/вывода для текущей задачи.

Сегментные регистры CS, SS, DS, ES, как и в процессорах 8086/8088, имеют разрядность 16 бит. К ним добавлены регистры FS и GS – дополнительные сегментные регистры данных.

В реальном режиме, в котором процессор находится сразу после включения питания или сброса, сегментные регистры определяют 64Кб сегменты, как и в 16-разрядных процессорах, а в защищенном режиме содержат указатели (т.н. селекторы) на описатели сегментов (64-разрядные *дескрипторы*), находящиеся в памяти в виде таблиц.

Дескриптор, помимо базового адреса, содержит предельный размер сегмента и атрибуты сегмента (права доступа). Дескрипторы являются основой защиты и мультизадачности. В защищенном режиме сегменты могут начинаться с любого линейного адреса и иметь любой предел вплоть до 4Гбайт. Существуют две обязательные дескрипторные таблицы - глобальная (GDT) и дескрипторная таблица прерывания (IDT), а также множество (до 8192) локальных дескрипторных таблиц (LDT), из которых в один момент времени процессору доступна только одна. Дескрипторы сегментов могут находиться в GDT или LDT. Расположение дескрипторных таблиц определяется регистрами процессора GDTR (Global Descriptor Table Register), IDTR (Interrupt Descriptor Table Register), LDTR (Local Descriptor Table Register). Регистры GDTR и IDTR - 6-байтные, они содержат 32 бита линейного базового адреса дескрипторной таблицы и 16 бит предела таблицы. Программно доступная часть регистра LDTR - 16 бит, которые являются селектором LDT. Дескрипторы LDT находятся в GDT. Однако чтобы не обращаться каждый раз к GDT, в процессоре имеется теневая (программно недоступная) часть регистра LDTR, в которую процессор помещает дескриптор LDT при каждой перегрузке селектора в регистре LDTR.

Значение сегментного регистра (селектор) содержит индекс дескриптора в дескрипторной таблице; бит, определяющий, к какой дескрипторной таблице производится обращение (LDT или GDT); а также запрашиваемые права доступа к сегменту. Таким образом, селектор выбирает дескрипторную таблицу, выбирает дескриптор из таблицы, а по дескриптору определяется положение сегмента в линейном пространстве памяти. Однако обращение к дескрипторным таблицам происходит только при загрузке селектора в сегментный регистр. При этом процессор помещает дескриптор в

теневую (программно недоступную) часть сегментного регистра. При формировании линейного адреса дескриптор сегмента процессору уже известен (см. Рис. 2.14).

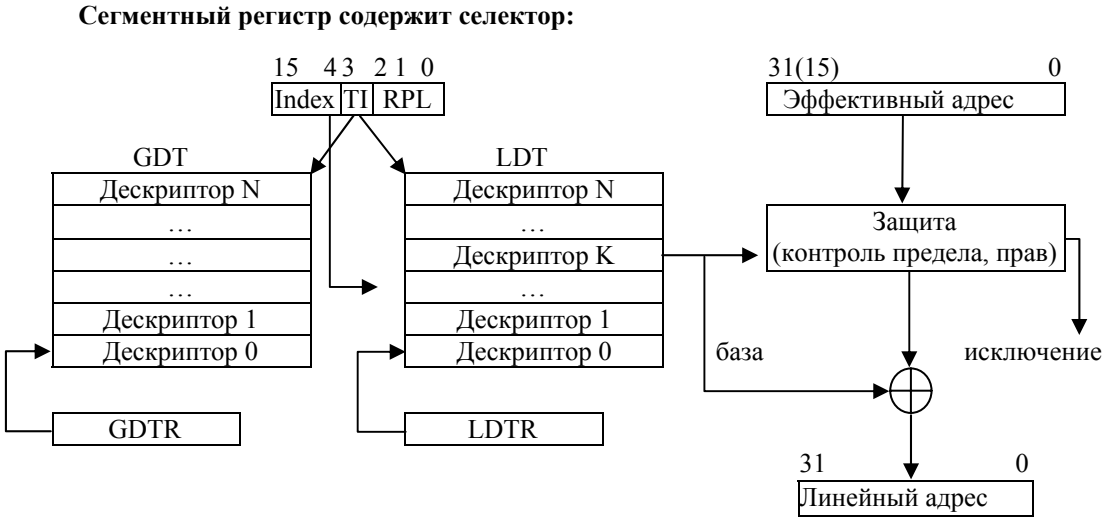


Рис. 2.14 Сегментная адресация в защищенном режиме

В защищенном режиме используются также 32-разрядные регистры управления CR0-CR3, регистры отладки DR0-DR7 и регистры проверки TR0-TR7.

Кроме механизма адресации 32-разрядные процессоры реализуют еще один уровень косвенности при формировании физического адреса – страничное преобразование адресов, где базовым объектом адресации является блок фиксированного размера (4Кбайт и/или 4 Мбайт) – страница. Фиксированный размер страницы позволяет организовать эффективную систему виртуальной памяти с заменой страниц по требованию и защитой на уровне страниц.

В страничном преобразовании участвуют два типа структур: каталоги таблиц и таблицы страниц. Их положение в памяти определяется физическим адресом, записанным в регистр управления CR3. Для включения страничной переадресации устанавливают 31 бит (Paging) в регистре CR0.

В защищенном режиме возможны дополнительные методы адресации. Команды имеют длину от 1 до 15 байт. Им могут предшествовать префиксные байты. Кроме префиксов REP и SEG, введены новые префиксы:

- размера операнда SIZ (operand SIZe), для переключения 16- и 32 – разрядных операндов;
- размера адреса ADDRSIZ (ADDRess SIZe) - для переключения 16- и 32 –разрядных адресов.

Для новых методов адресации в формат команды добавлено поле SIB (Scale Index Base), которое определяет масштаб, индексный и базовый ре-

гистры. Эффективный адрес определяется как сумма значений базового регистра, смещения в команде и индексного регистра, умноженного на масштабный коэффициент (1, 2, 4, 8). Примеры команд с новыми методами адресации приведены в следующей таблице:

Метод адресации	**Пример команд**
Индексный с масштабированием и смещением	mov eax, [2*esi + 100h]
Базовый индексный с масштабированием	mov eax, [ebp + 4*esi]
Базовый индексный с масштабированием и смещением	mov eax, [ebp + 8*esi + 10h]

2.4 Программа Turbo Debugger и лабораторные занятия

Лабораторные занятия по второму разделу (x86) необходимо начинать с изучения работы с отладчиком Turbo Debugger. Затем, рекомендуется выполнить все примеры к разделам 2.2.5 – 2.2.8. В разделе 2.2.10 можно изучить работу функций BIOS для работы с видео (прерывание int 10h) на задачах: вывести точку (группу точек), вывести знак (группу знаков) по заданным координатам. Имеет смысл повторно реализовать задания первого раздела (для учебной модели PDP-11) теперь в системе команд x86.

Отладчики Turbo Debugger фирмы Borland для реального (td.exe) и защищенного (td32.exe) режимов позволяют отлаживать код программ, предназначенных для систем DOS и Windows. Turbo Debugger дает возможность изменять код тестируемых программ и отлаживать его в пошаговом режиме, имеет встроенный ассемблер для ввода команд в мнемонике.

Для начала работы (выполнения примеров и заданий) обычно загружают в отладчик исполняемый файл (меню File→Open→Browse→имя файла) или подключаются к процессу (меню File→Attach→процесс). Например: File → Open → Browse → rtm.exe (или td32.exe для 32-х разрядной версии) → OK. Верхняя строка экрана – меню, содержащее средства перестройки режимов отображения, управления комплексом отлаживаемых модулей и функциональных режимов (выберите View → CPU). В нижней строке указывается назначение функциональных клавиш отладчика.

Обычно используются 5 окон отладчика: команд, стека, данных, регистров и флагов, которые вызываются с помощью команды меню View→CPU. В зависимости от типа активного окна просмотра можно пользоваться следующими сочетаниями клавиш:

окно команд			**окна данных, стека**		
Ctrl G	Goto	Переход по адресу	Ctrl G	Goto	Переход
Ctrl O	Origin	Перейти к IP	Ctrl S	Search	Поиск

окно команд			**окна данных, стека**		
Ctrl F	Follow	Следовать	Ctrl N	Next	Поиск вперед
Ctrl N	Next	Установить IP	Ctrl C	Change	Изменить
Ctrl C	Caller	Вызывающий адрес	Ctrl F	Follow	Следовать
Ctrl P	Previous	К предыдущему IP	Ctrl P	Previ-ous	Поиск назад
Ctrl S	Search	Поиск	Ctrl D	Display	Формат данных
Ctrl V	View	Просмотр	Ctrl B	Block	Работа с блоком
Ctrl M	Mixed	Смешанный			

окно регистров			**окно флагов**		
Ctrl I	Incre-ment	Добавить единицу	Ctrl T	Toggle	Поменять значе-ние
Ctrl D	Decre-ment	Вычесть единицу			
Ctrl Z	Zero	Очистить			
Ctrl C	Change	Изменить			

Для исполнения программы в пошаговом режиме необходимо определить стартовый адрес в паре CS:IP и, затем, нажатиями F7 или F8 (без «захода» в код подпрограмм) выполнять команды. Другие функциональные клавиши приведены ниже в таблице:

Клавиша	**Команда**	**Действие**
F1	Help	Вызов справки
F2	Toggle Breakpoint	Установка/снятие контрольной точки
F3	Module	Загрузка модуля или таблицы символов
F4	Go to Cursor	Выполнить до курсора
F5	Zoom Window	Изменить масштаб окна
F6	Next Window	Следующее окно
F7	Trace Into	Выполнить шаг с заходом в подпрограммы
F8	Step Over	Выполнить шаг без заходов в подпрограммы
F9	Run	Выполнить все
F10	Command menu	Вызов командного меню
Alt F1	Last help	Вызов справки по последней теме
Alt F2	Breakpoint At	Установить контрольную точку по адресу …
Alt F3	Close Window	Закрыть окно
Alt F4	Back trace	Трассировка назад
Alt F5	User Screen	Экран выполняемой программы

Клавиша	Команда	Действие
Alt F6	Undo Close	Отмена закрытия
Alt F7	Instruction Trace	Выполнить одну команду
Alt F8	Until Return	Выполнять до возврата
Alt F9	Execute To	Выполнять до указанного адреса
Alt F10	Speed menu	Вызов контекстного меню
Ctrl F2	Program Reset	Сброс, перезагрузка программы
Ctrl F4	Data evaluate	Оценить значение выражения
Ctrl F5	Window resize	Изменение размера окна
Ctrl F7	Data Watch	Слежение за значением
Ctrl F8	Breakpoint Toggle	Установка/снятие контрольной точки
Ctrl F9	Run	Выполнить все
Ctrl F10	Speed menu	Вызов контекстного меню

Приложения

1. Список команд процессора учебной ЭВМ (подмножество системы команд семейства PDP-11)

Мнемоника	Команда	Код	Стр.
	ОДНОАДРЕСНЫЕ КОМАНДЫ		
CLR ***(B)***	Очистка	***0500** *DD*	17
COM ***(B)***	Инвертирование	***0510** *DD*	17
INC ***(B)***	Прибавление единицы	***0520** *DD*	17
DEC ***(B)***	Вычитание единицы	***0530** *DD*	18
NEG ***(B)***	Изменение знака	***0540** *DD*	18
TST ***(B)***	Проверка	***0570** *DD*	18
ASR ***(B)***	Арифметический сдвиг вправо	***0620** *DD*	19
ASL ***(B)***	Арифметический сдвиг влево	***0630** *DD*	19
ROR ***(B)***	Циклический сдвиг вправо	***0600** *DD*	20
ROL ***(B)***	Циклический сдвиг вправо	***0610** *DD*	20
	ДВУХАДРЕСНЫЕ КОМАНДЫ		
MOV ***(B)***	Пересылка	***1** *SSDD*	21
CMP ***(B)***	Сравнение	***2** *SSDD*	21
ADD	Сложение	**06** *SSDD*	22
SUB	Вычитание	**16** *SSDD*	22
BIT ***(B)***	Проверка разрядов	***3** *SSDD*	23
BIC ***(B)***	Очистка разрядов	***4** *SSDD*	23
BIS ***(B)***	Логическое сложение	***5** *SSDD*	24
XOR	Исключающее ИЛИ	**074** *RDD*	24
	КОМАНДЫ УПРАВЛЕНИЯ ПРОГРАММОЙ		
BR	Ветвление безусловное	**000400**	25
BNE	Ветвление, если не равно (нулю)	**001000**	25
BEQ	Ветвление, если равно (нулю)	**001400**	26
BPL	Ветвление, если плюс	**100000**	26
BMI	Ветвление, если минус	**100400**	26
BVC	Ветвление, если нет арифметического переполнения	**102000**	27

BVS	Ветвление, если есть арифметическое переполнение	**102400**	27
BCS	Ветвление, если перенос	**103400**	28
BCC	Ветвление, если нет переноса	**103000**	27
BGE	Ветвление, если больше или равно (нулю)	**002000**	28
BLT	Ветвление, если меньше (нуля)	**002400**	29
BGT	Ветвление, если больше (нуля)	**003000**	29
BLE	Ветвление, если меньше или равно (нулю)	**003400**	30
BHI	Ветвление, если больше	**101000**	30
BLOS	Ветвление, если меньше или равно	**101400**	31
BHIS	Ветвление, если больше или равно	**103000**	31
BLO	Ветвление, если меньше	**103400**	31
JMP	Безусловный переход	**0001***DD*	32
JSR	Обращение к подпрограмме	**004***RDD*	32
RTS	Возврат из подпрограммы	**00020***R*	33
SOB	Вычитание единицы и ветвление	**077***RNN*	33
	КОМАНДЫ ПРЕРЫВАНИЯ		
EMT	Командное прерывание для системных программ	**104000-104377**	34
TRAP	Командное прерывание	**104400-104777**	
IOT	Командное прерывание для ввода-вывода	**000004**	
BPT	Командное прерывание для отладки	**000003**	
RTI	Возврат из прерывания	**000002**	35
RTT	Возврат из прерывания	**000006**	
	КОМАНДЫ УПРАВЛЕНИЯ ПРОЦЕССОРОМ		
HALT	Останов	**000000**	35

	КОМАНДЫ ИЗМЕНЕНИЯ ПРИЗНАКОВ		
CLN	Очистка N	**000250**	
CLZ	Очистка Z	**000244**	
CLV	Очистка V	**000242**	
CLC	Очистка C	**000241**	
CCC	Очистка всех разрядов (NZVC)	**000257**	
SEN	Установка N	**000270**	
SEZ	Установка Z	**000264**	
SEV	Установка V	**000262**	
SEC	Установка C	**000261**	
SCC	Установка всех разрядов (NZVC)	**000277**	
NOP	Нет операции	**000240**	

2. Список команд процессора i8086 (в алфавитном порядке)

Мнемоника	Название команды
AAA	Коррекция кода ASCII для сложения
AAD	Коррекция кода ASCII для деления
AAM	Коррекция кода ASCII для умножения
AAS	Коррекция кода ASCII для вычитания
ADC A1,A2	Сложение байтов или слов с переносом
ADD A1,A2	Сложение байтов или слов
AND A1,A2	Логическое умножение байтов или слов (И)
CALL address	Вызов подпрограммы
CLC	Сбросить флаг переноса CF в 0
CLD	Сбросить флаг направления DF в 0
CLI	Сброс флага разрешения прерывания IF в 0
CMC	Инвертировать флаг переноса CF
CMP A1,A2	Сравнение байта или слова
CMPS	Сравнение байта или слова строки
CWB	Преобразование байта в слово
CWD	Преобразование слова в двойное слово
DAA	Десятичная коррекция для сложения
DAS	Десятичная коррекция для вычитания
DEC A1	Вычитание единицы из байта или слова

DIV A2	Деление байтов или слов без знака
ESC	Выдача кода для внешнего процессора
HLT	Переход процессора в состояние останова
IDIV A2	Деление байтов или слов со знаком
IMUL A2	Умножение байтов или слов со знаком
IN AC, PORT	Ввод байта или слова из порта
INC A1	Увеличение байта или слова на единицу
INTO	Прерывание по переполнению
IRET	Возврат из прерывания
JA, JNBE	Перейти, если выше/не ниже или равно
JAE, JNB	Перейти, если выше или равно/не ниже
JB, JNAE	Перейти, если ниже/не выше или равно
JBE, JNA	Перейти, если ниже или равно/не выше
JC	Перейти, если есть перенос
JCXZ	Перейти, если содержимое регистра CX=0
JE, JZ	Перейти, если равно/нуль
JG, JNLE	Перейти, если больше/не меньше или равно
JGE, JNL	Перейти, если больше или равно/не меньше
JL, JNGE	Перейти, если меньше/не больше или равно
JLE, JNG	Перейти, если меньше или равно/не больше
JNC	Перейти, если нет переноса
JNE, JNZ	Перейти, если не равно/не нуль
JNO	Перейти, если нет переполнения
JNP, JPO	Перейти, если нет паритета/паритет нечетный
JNS	Перейти, если нет знака
JO	Перейти, если есть переполнения
JP, JPE	Перейти, если есть паритет/паритет четный
JS	Перейти, если есть знак
LAHF	Загрузка флагов в регистр AH
LDS R,A2	Загрузка указателя с загрузкой адреса сегмента в DS
LEA R,A2	Загрузка исполнительного адреса
LES R,A2	Загрузка указателя с загрузкой адреса сегмента в ES
LOCK	Блокировка системной шины
LODS	Загрузка байта или слова строки
LOOP	Цикл пока CX не равно 0
LOOPE	Цикл пока CX не равно 0 и ZF=1
LOOPNE	Цикл пока CX не равно 0 и ZF=0
LOOPNZ	Цикл пока CX не равно 0 и ZF=0

LOOPZ	Цикл пока CX не равно 0 и ZF=1
MOV A1,A2	Пересылка байта или слова
MOVS	Пересылка байта или слова строки
MUL A2	Умножение байтов или слов без знака
NEG A1	Отрицание (дополнительный код) байта или слова
NOP	Команда пустой операции
NOT A1	Инвертирование байта или слова (НЕ)
OR A1,A2	Логическое сложение байтов или слов (ИЛИ)
OUT PORT,AC	Вывод байта или слова в порт
POP A1	Чтение слова из стека
POPF	Установка флагов из стека
PUSH A1	Запись слова в стек
PUSHF	Запись флагов в стек
RCL A1, count	Циклический сдвиг байта или слова влево через CF
RCR A1, count	Циклический сдвиг байта или слова вправо через CF
RET	Возврат из подпрограммы
ROL A1, count	Циклический сдвиг байта или слова влево
ROR A1, count	Циклический сдвиг байта или слова вправо
SAHF	Установка флагов из регистра AH
SAL A1, count	Арифметический сдвиг байта или слова влево
SAR A1, count	Арифметический сдвиг байта или слова вправо
SBB A1,A2	Вычитание байтов или слов с заемом
SCAS	Сравнение байта или слова строки с аккумулятором
SHL A1, count	Логический сдвиг байта или слова влево
SHR A1, count	Логический сдвиг байта или слова вправо
STC	Установить флаг переноса CF в 1
STD	Установить флаг направления DF в 1
STI	Установка флага разрешения прерывания IF в 1
STOS	Запоминание байта или слова строки
SUB A1,A2	Вычитание байтов или слов
TEST A1,A2	Логическое умножение байтов или слов без записи
WAIT	Переход процессора в состояние ожидания
XCHG A1,A2	Обмен байтами или словами
XLAT	Перекодировка байта
XOR A1,A2	Исключающее ИЛИ над байтами или словами

Литература

1. Френк Т.С. PDP-11: Архитектура и программирование: Пер. с англ. – М.: Радио и связь, 1986. – 371 с.
2. PDP11.ORG http://www.pdp11.org/
3. Intel 64 and IA-32 Architectures Software Developer's Manual vol. 1, http://www.intel.com/, 2006. – 466 p.
4. Гук М. Процессоры Pentium II, Pentium Pro и просто Pentium – СПб: ЗАО «Издательство Питер», 1999. – 288 с.
5. Гук, Михаил. Процессоры Pentium 4, Athlon и Duron / Михаил Гук, Виктор Юров.— СПб. и др. : Питер, 2001 .— 511 с.
6. Таненбаум, Эндрю. Архитектура компьютера : пер. с англ. / Э.Таненбаум .— 5-е изд. — СПб. [и др.] : Питер, 2007 .— 698 с.
7. Википедия Intel, http://ru.wikipedia.org/wiki/Intel

Содержание

Учебное издание

Коваль Андрей Сергеевич,
Сычев Александр Васильевич

АРХИТЕКТУРА ЭВМ И СИСТЕМ

Учебное пособие для вузов

Учебно-методическое пособие для лабораторных занятий

Редактор Бунина Т.Д.