ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

П.С.Епифанов
Н.А.Краев
Ф.В.Частиков

МИКРОПРОЦЕССОРЫ СЕМЕЙСТВА i80x86

Учебное пособие

Киров 2010

УДК 621.396.6

Составители: профессор, д. т. н. А. В. Частиков
доцент, к. т. н. Н. А. Краев
доцент, к.т.н. П.С. Епифанов

Рецензент: доцент, к. т. н. В. С. Грудинин, каф. ЭП

Редактор А.Н.Корсаков

Подписано в печать Усл. печ. л.
Бумага книжно-журнальная. Печать матричная.
Заказ № Тираж Бесплатно.
Текст напечатан с оригинал-макета, представленного авторами

610000, г. Киров, ул. Московская, 36.
Оформление обложки, изготовление – ПРИП

1. АРХИТЕКТУРА, ОРГАНИЗАЦИЯ ПАМЯТИ, СПОСОБЫ АДРЕСАЦИИ, СИСТЕМА КОМАНД ПРОЦЕССОРОВ i80x86

1.1. Архитектура ЭВМ.

Архитектура ЭВМ – это абстрактное представление ЭВМ, которое отражает ее структурную, схемотехническую и логическую организацию. Понятие архитектуры ЭВМ является комплексным и включает в себя:

- структурную схему ЭВМ;
- средства и способы доступа к элементам структурной схемы ЭВМ;
- организацию и разрядность интерфейсов ЭВМ;
- набор и доступность регистров;
- организацию и способы адресации памяти;
- способы представления и форматы данных ЭВМ;
- набор машинных команд ЭВМ;
- форматы машинных команд;
- обработку нештатных ситуаций (прерываний).

Все современные ЭВМ обладают некоторыми общими и индивидуальными свойствами архитектуры. Индивидуальные свойства присущи только конкретной модели компьютера и отличают его от больших и малых собратьев. Наличие общих архитектурных свойств обусловлено тем, что большинство типов существующих машин принадлежат 4 и 5–му поколениям ЭВМ так называемой фон–неймановской архитектуры. К числу *общих архитектурных свойств и принципов* можно отнести:

- *Принцип хранимой программы.* Согласно ему, код программы и ее данные находятся в одном адресном пространстве в оперативной памяти.
- *Принцип микропрограммирования.* Суть этого принципа заключается в том, что машинный язык еще не является той конечной субстанцией, которая физически приводит в действие процессы в машине. В состав процессора входит *блок микропрограммного управления*. Этот блок для каждой машинной команды имеет набор действий–сигналов, которые нужно сгенерировать для физического выполнения требуемой машинной команды. Здесь уместно вспомнить характеристику ЭВМ 1–го поколения. В них для генерации нужных сигналов необходимо было осуществить ручное программирование всех логических схем – поистине адская и неблагодарная работа!
- *Линейное пространство памяти* – совокупность ячеек памяти, которым последовательно присваиваются номера (адреса) 0, 1, 2, ….
- *Последовательное выполнение программ.* Процессор выбирает из памяти команды строго последовательно. Для изменения прямолинейного хода выполнения программы или осуществления ветвления необходимо использовать специальные команды. Они называются командами условного и безусловного перехода.

С точки зрения процессора нет принципиальной разницы между данными и командами. Данные и машинные команды находятся в одном пространстве памяти в виде последовательности нулей и единиц. Это свойство связано с предыдущим. Процессор, исполняя содержимое некоторых последовательных ячеек памяти, всегда пытается трактовать его как коды машинной команды, а если это не так, то происходит аварийное завершение программы, содержащей некорректный фрагмент. Поэтому важно в программе всегда четко разделять пространство данных и команд.

- *Безразличие к целевому назначению данных.* Машине все равно, какую логическую нагрузку несут обрабатываемые ею данные.

Суперскалярная архитектура. Для того чтобы пояснить этот термин, разберемся вначале со значением другого термина – *конвейеризация вычислений.* Важным элементом архитектуры, появившимся в i486, стал *конвейер* – специальное устройство, реализующее такой метод обработки команд внутри микропроцессора, при котором исполнение команды разбивается на несколько этапов, i486 имеет пятиступенчатый конвейер. Соответствующие пять этапов включают:

- выборку команды из кэш–памяти или оперативной памяти;
- декодирование команды;
- генерацию адреса, при которой определяются адреса операндов в памяти;
- выполнение операции с помощью АЛУ;
- запись результата (куда будет записан результат, зависит от алгоритма работы конкретной машинной команды).

На стадии выполнения каждая машинная команда как бы разбивается на более элементарные операции. Очередная команда после ее выборки попадает в блок декодирования. Блок выборки свободен и может выбрать следующую команду. В результате на конвейере могут находиться в различной стадии выполнения пять команд. Скорость вычисления в результате существенно возрастает. Микропроцессоры, имеющие один конвейер, называются *скалярными.* Pentium имеет два конвейера, а Pentium Pro – три, поэтому эти микропроцессоры называются *суперскалярными.*

Раздельное кэширование кода и данных. Кэширование – это способ увеличения быстродействия системы за счет хранения часто используемых данных и кодов в так называемой «кэш–памяти первого уровня» (быстрой памяти), находящейся внутри микропроцессора i486, к примеру, содержит один блок встроенной кэш–памяти размером 8 Кбайт, который используется для кэширования и кодов, и данных. Pentium содержит два блока кэш–памяти: один для кода и один для данных, каждый по 8 Кбайт. При этом становится возможным одновременный доступ к коду и данным, что увеличивает скорость работы компьютера.

Предсказание правильного адреса перехода. Под *переходом* понимается запланированное алгоритмом изменение последовательного характера выполнения программы. Как показывает статистика, типичная программа на

каждые 6–8 команд содержит 1 команду перехода. Последствия этого предсказать несложно: при наличии конвейера через каждые 6–8 команд его нужно очищать и заполнять заново в соответствии с адресом перехода. Все преимущества конвейеризации теряются. Поэтому в архитектуру Pentium был введен *блок предсказания переходов.* Суть этого метода заключается в следующем. Pentium имеет буфер адресов перехода, который хранит информацию о последних 256 переходах. Если некоторая команда управляет ветвлением, то в буфере запоминаются эта команда, адрес перехода и предположение о том, какая ветвь программы будет выполнена следующей. Почти в любой программе имеются циклы, в ходе выполнения которых периодически необходимо принимать решение либо о выходе из цикла, либо о переходе на его начало. Специальный блок предсказания адреса перехода прогнозирует, какое решение будет принято программой. При этом он основывается на предположении, что ветвь, которая была пройдена, будет использоваться снова, и загружает соответствующую команду перехода на конвейер. В случае, если это предсказание верно, переход осуществляется без задержки. Для того чтобы судить об эффективности этого нововведения, достаточно отметить, что вероятность правильного предсказания составляет около 80%.

Усовершенствованный блок вычислений с плавающей точкой. Он позволяет выполнять одну команду с плавающей точкой за один такт микропроцессора.

1.2. Набор регистров.

Программная модель микропроцессора содержит 32 регистра, в той или иной мере доступных для использования программистом. Их можно разделить на две большие группы:

- 16 пользовательских регистров;
- 16 системных регистров.

1.2.1. Пользовательские регистры.

Как следует из названия, *пользовательскими* регистры называются потому, что программист может использовать их при написании своих программ. К этим регистрам относятся (рис. 1):

- восемь 32–битных регистров, которые могут использоваться программистами для хранения данных и адресов (их еще называют регистрами общего назначения (РОН)): eax/ax/ah/al, ebx/bx/bh/bl, edx/dx/dh/dl, ecx/cx/ch/cl, ebp/bp, esi/si, edi/di, esp/sp;
- шесть регистров сегментов: cs, ds, ss, es, fs, gs;
- регистры состояния и управления: регистр флагов eflags/flags и регистр указателя команды eip/ip.

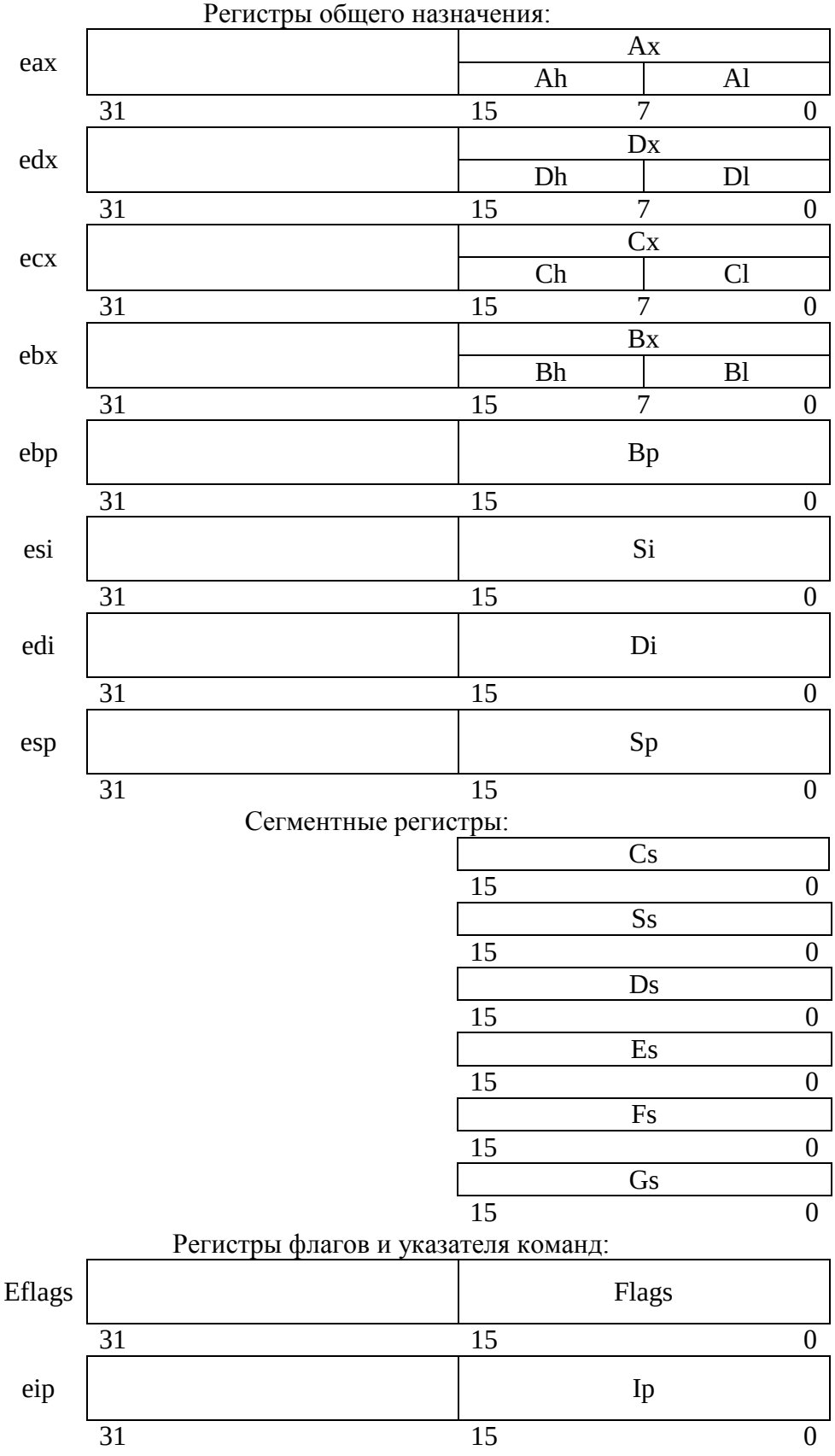


Рис. 1. Пользовательские регистры, микропроцессоров i486 и Pentium

Многие из этих регистров приведены с наклонной разделительной чертой. Это не разные регистры – это части одного большого 32–разрядного регистра. Их можно использовать в программе как отдельные объекты. Это сделано для обеспечения работоспособности программ, написанных для младших 16–разрядных моделей микропроцессоров фирмы Intel, начиная с i8086. Микропроцессоры i486 и Pentium имеют, в основном, 32–разрядные регистры. Их количество, за исключением сегментных регистров, такое же, как и у i8086, но размерность больше, что и отражено в их обозначениях – они имеют приставку e (Extended).

1.2.2. Регистры общего назначения.

Все регистры этой группы позволяют обращаться к своим «младшим» частям (см. рис. 1). Использовать для самостоятельной адресации можно только младшие 16–и 8–битные части этих регистров. Старшие 16 бит этих регистров как самостоятельные объекты недоступны. Это сделано для совместимости с младшими 16–разрядными моделями микропроцессоров фирмы Intel. Перечислим регистры, относящиеся к группе регистров общего назначения. Так как эти регистры физически находятся в микропроцессоре внутри арифметико-логического устройства (АЛУ), то их еще называют *регистрами АЛУ:*

- eax/ax/ah/al (Accumulator register) – *аккумулятор*. Применяется для хранения промежуточных данных. В некоторых командах использование этого регистра обязательно;
- ebx/bx/bh/bl (Base register) – *базовый регистр*. Применяется для хранения базового адреса некоторого объекта в памяти;
- ecx/cx/ch/cl (Count register) – *регистр–счетчик*. Применяется в командах, производящих некоторые повторяющиеся действия. Его использование зачастую неявно и скрыто в алгоритме работы соответствующей команды. К примеру, команда организации цикла loop кроме передачи управления команде, находящейся по некоторому адресу, анализирует и уменьшает на единицу значение регистра ecx/cx;
- edx/dx/dh/dl (Data register) – *регистр данных*. Так же, как и регистр eax/ax/ah/al, он хранит промежуточные данные. В некоторых командах его использование обязательно; для некоторых команд это происходит неявно.
- Следующие два регистра используются для поддержки так называемых цепочечных операций, то есть операций, производящих последовательную обработку цепочек элементов, каждый из которых может иметь длину 32, 16 или 8 бит:
- esi/si (Source Index register) – *индекс источника*. Этот регистр в цепочечных операциях содержит текущий адрес элемента в цепочке–источнике;

- edi/di (Destination Index register) – *индекс приемника (получателя).* Этот регистр в цепочечных операциях содержит текущий адрес в цепочке–приемнике.
- В архитектуре микропроцессора на программно–аппаратном уровне поддерживается такая структура данных, как *стек.* Для работы со стеком в системе команд микропроцессора есть специальные команды, а в программной модели микропроцессора для этого существуют специальные регистры:
- esp/sp (Stack Pointer register) – *регистр указателя стека.* Содержит указатель вершины стека в текущем сегменте стека;
- ebp/bp (Base Pointer register) – *регистр указателя базы кадра стека.* Предназначен для организации произвольного доступа к данным внутри стека.

Не всегда нужно придерживаться столь жесткого функционального назначения регистров АЛУ. На самом деле, большинство из них могут использоваться при программировании для хранения операндов практически в любых сочетаниях. Использование жесткого закрепления регистров для некоторых команд позволяет более компактно кодировать их машинное представление. Знание этих особенностей позволит вам при необходимости хотя бы на несколько байт сэкономить память, занимаемую кодом программы.

1.2.3. Сегментные регистры.

В программной модели микропроцессора имеется шесть *сегментных регистров*: cs, ss, ds, es, gs, fs. Их существование обусловлено спецификой организации и использования оперативной памяти микропроцессорами Intel. Она заключается в том, что микропроцессор аппаратно поддерживает структурную организацию программы в виде трех частей, называемых *сегментами.* Соответственно, такая организация памяти называется *сегментной.* Для того чтобы указать на сегменты, к которым программа имеет доступ в конкретный момент времени, и предназначены сегментные регистры. Фактически, с небольшой поправкой, как мы увидим далее, в этих регистрах содержатся адреса памяти, с которых начинаются соответствующие сегменты. Логика обработки машинной команды построена так, что при выборке команды, доступе к данным программы или к стеку неявно используются адреса во вполне определенных сегментных регистрах. Микропроцессор поддерживает следующие типы сегментов:

1. *Сегмент кода.* Содержит команды программы. Для доступа к этому сегменту служит регистр cs (code segment register) – *сегментный регистр кода.* Он содержит адрес сегмента с машинными командами, к которому имеет доступ микропроцессор (то есть эти команды загружаются в конвейер микропроцессора);

2. *Сегмент данных.* Содержит обрабатываемые программой данные. Для доступа к этому сегменту служит регистр ds (data segment register) – *сегментный регистр данных,* который хранит адрес сегмента данных текущей программы;
3. *Сегмент стека.* Этот сегмент представляет собой область памяти, называемую стеком. Работу со стеком микропроцессор организует по следующему принципу: последний записанный в эту область элемент выбирается первым. Для доступа к этому сегменту служит регистр ss (stack segment register) – *сегментный регистр стека,* содержащий адрес сегмента стека;
4. *Дополнительный сегмент данных.* Неявно алгоритмы выполнения большинства машинных команд предполагают, что обрабатываемые ими данные расположены в сегменте данных, адрес которого находится в сегментном регистре ds. Если программе недостаточно одного сегмента данных, то она имеет возможность использовать еще три дополнительных сегмента данных. Но в отличие от основного сегмента данных, адрес которого содержится в сегментном регистре ds, при использовании дополнительных сегментов данных их адреса требуется указывать явно с помощью специальных префиксов переопределения сегментов в команде. Адреса дополнительных сегментов данных должны содержаться в регистрах es, gs, fs (extension data segment registers).

1.2.4. Регистры состояния и управления.

В микропроцессор включены несколько регистров (см. рис. 1), которые постоянно содержат информацию о состоянии как самого микропроцессора, так и программы, команды которой в данный момент загружены на конвейер.

К этим регистрам относятся:

- регистр флагов eflags/flags;
- регистр указателя команды eip/ip.

Используя эти регистры, можно получать информацию о результатах выполнения команд и влиять на состояние самого микропроцессора. Рассмотрим подробнее назначение и содержимое этих регистров:

eflags/flags (flag register) – *регистр флагов.* Разрядность eflags/flags – 32/16 бит. Отдельные биты данного регистра имеют определенное функциональное назначение и называются *флагами.* Младшая часть этого регистра полностью аналогична регистру flags для i8086. На рис. 2 показано содержимое регистра eflags.

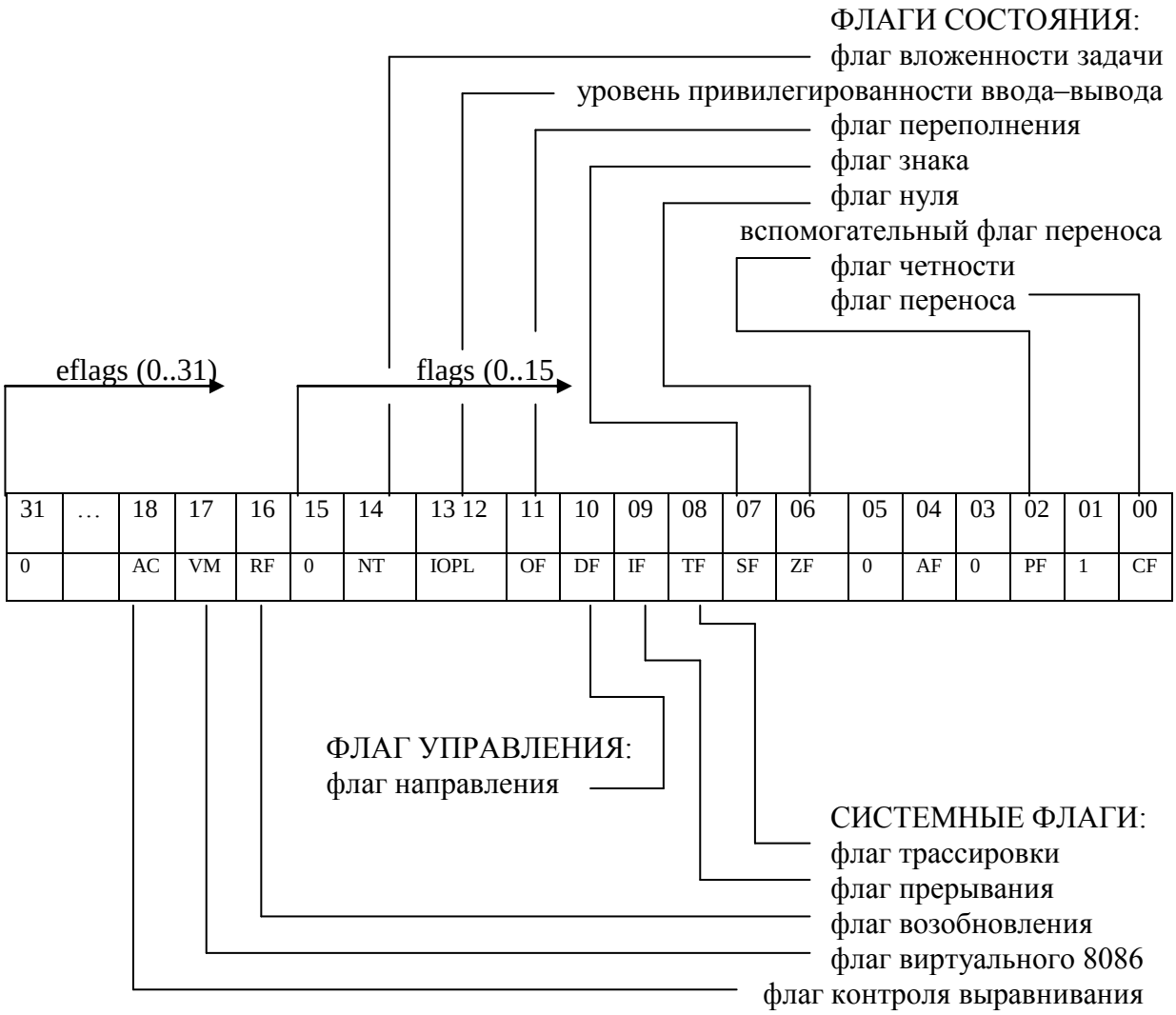


Рис. 2. Содержимое регистра eflags

Исходя из особенностей использования, флаги регистра eflags/flags можно разделить на три группы:

- 8 *флагов состояния.* Эти флаги могут изменяться после выполнения машинных команд. Флаги состояния регистра eflags отражают особенности результата исполнения арифметических или логических операций. Это дает возможность анализировать состояние вычислительного процесса и реагировать на него с помощью команд условных переходов и вызовов подпрограмм. В табл. 1 приведены флаги состояния и указано их назначение;
- 1 *флаг управления.* Обозначается как df (Directory Flag). Он находится в 10–м бите регистра eflags и используется цепочечными командами. Значение флага df определяет направление поэлементной обработки в этих операциях: от начала строки к концу (df = 0), либо наоборот, от конца строки к ее началу (df = 1). Для работы с флагом df существуют специальные команды eld (снять флаг df) и std (установить флаг df). Применение этих команд

позволяет привести флаг df в соответствие с алгоритмом и обеспечить автоматическое увеличение или уменьшение счетчиков при выполнении операций со строками;

- 5 *системных флагов,* управляющих вводом/выводом, маскируемыми прерываниями, отладкой, переключением между задачами и виртуальным режимом 8086. Прикладным программам не рекомендуется модифицировать без необходимости эти флаги, так как в большинстве случаев это приведет к прерыванию работы программы. В табл. 2 перечислены системные флаги, их назначение.

Таблица 1. Флаги состояния

Мнемоника флага	**Флаг**	**Номер бита в *eflags***	**Содержание и назначение**
Cf	Флаг переноса (Carry Flag)	0	1 – арифметическая операция произвела перенос из старшего бита результата. Старшим является 7, 15 или 31-й бит в зависимости от размерности операнда; 0 – переноса не было
Pf	Флаг паритета (Parity Flag)	2	1 – 8 младших разрядов (этот флаг – только для 8 младших разрядов операнда любого размера) результата содержат четное число единиц; 0 – 8 младших разрядов результата содержат нечетное число единиц
Af	Вспомогательный флаг переноса (Auxiliary carry Flag)	4	Только для команд, работающих с BCD-числами. Фиксирует факт заема из младшей тетрады результата: 1 – в результате операции сложения был произведен перенос из разряда 3 в старший разряд или при вычитании был заем в разряд 3 младшей тетрады из значения в старшей тетраде; 0 – переносов и заемов в(из) 3 разряд(а) младшей тетрады результата не было
Zf	Флаг нуля (Zero Flag)	6	1 – результат нулевой; 0 – результат ненулевой
Sf	Флаг знака (Sign Flag)	7	Отражает состояние старшего бита результата (биты 7, 15 или 31 для 8, 16 или 32-разрядных операндов соответственно): 1 – старший бит результата равен 1; 0 – старший бит результата равен 0
Of	Флаг переполнения (Overflow Flag)	11	Флаг of используется для фиксирования факта потери значащего бита при арифметических операциях: 1 – в результате операции происходит перенос (заем) в(из) старшего, знакового бита результата (биты 7, 15 или 31 для 8, 16 или 32-разрядных операндов соответственно); 0 – в результате операции не происходит переноса (заема) в(из) старшего, знакового бита результата

Iopl	Уровень Привилегий ввода-вывода (Input/Output Privilege Level)	12, 13	Используется в защищенном режиме работы микропроцессора для контроля доступа к командам ввода-вывода в зависимости от привилегированности задачи
nt	Флажок вложенности задачи (Nested Task)	14	Используется в защищенном режиме работы микропроцессора для фиксации того факта, что одна задача вложена в другую

Таблица 2. Системные флаги

Мнемоника флага	**Флаг**	**Номер бита в *eflags***	**Содержание и назначение**
tf	Флаг трассировки (Trace Flag)	8	Предназначен для организации пошаговой работы микропроцессора. 1 – микропроцессор генерирует прерывание с номером 1 после выполнения каждой машинной команды. Может использоваться при отладке программ, в частности отладчиками; 0 – обычная работа
if	Флаг прерывания (Interrupt enable Flag)	9	Предназначен для разрешения или запрещения (маскирования) аппаратных прерываний (прерываний по входу INTR). 1 – аппаратные прерывания разрешены; 0 – аппаратные прерывания запрещены
rf	Флаг возобновления (Resume Flag)	16	Используется при обработке прерываний от регистров отладки.
vm	Флаг виртуального (Virtual 8086 Mode)	17	Признак работы микропроцессора в режиме виртуального 8086. 1 – процессор работает в режиме виртуального 8086; 0 – процессор работает в реальном или защищенном режиме
ac	Флаг контроля выравнивания (Alignment Check)	18	Предназначен для разрешения контроля выравнивания при обращениях к памяти. Используется совместно с битом am в системном регистре cr0. К примеру, Pentium разрешает размещать команды и данные с любого адреса. Если требуется контролировать выравнивание данных и команд по адресам, кратным 2 или 4, то установка данных битов приведет к тому, что все обращения по некратным адресам будут возбуждать исключительную ситуацию.

eip/ip (Instruction Pointer register) – *указатель команд.* Регистр eip/ip имеет разрядность 32/16 бит и содержит смещение следующей подлежащей выполнению команды относительно содержимого сегментного регистра cs в текущем сегменте команд. Этот регистр непосредственно недоступен программисту, но загрузка и изменение его значения производятся различными командами управления, к которым относятся команды условных и безусловных переходов, вызова процедур и возврата из процедур. Возникновение прерываний также приводит к модификации регистра eip/ip.

1.3. Организация памяти.

Физическая память, к которой микропроцессор имеет доступ по шине адреса, называется *оперативной памятью* (или оперативным запоминающим устройством – ОЗУ). На самом нижнем уровне память компьютера можно рассматривать как массив *битов.* Один бит может хранить значение 0 или 1. Для физической реализации битов и работы с ними идеально подходят логические схемы. Но микропроцессору неудобно работать с памятью на уровне битов, поэтому реально ОЗУ организовано как последовательность ячеек – *байтов.* Один байт состоит из 8 бит. Каждому байту соответствует свой уникальный адрес (его номер), называемый *физическим.* Диапазон значений физических адресов зависит от разрядности шины адреса микропроцессора. Для i486 и Pentium он находится в пределах от 0 до $2^{32} – 1$ (4 Гбайт).

Механизм управления памятью полностью аппаратный. Это означает, что программа не может сама сформировать физический адрес памяти на адресной шине. Ей приходится «играть» по правилам микропроцессора. В конечном итоге этот механизм позволяет обеспечить:

- компактность хранения адреса в машинной команде;
- гибкость механизма адресации;
- защиту адресных пространств задач в многозадачной системе;
- поддержку виртуальной памяти.

Микропроцессор аппаратно поддерживает несколько моделей использования оперативной памяти:

- *сегментированную модель.* В этой модели память для программы делится на непрерывные области памяти (сегменты), а сама программа может обращаться только к данным, которые находятся в этих сегментах;
- *страничную модель.* Ее можно рассматривать как надстройку над сегментированной моделью. В случае использования этой модели оперативная память рассматривается как совокупность блоков фиксированного размера 4 Кбайт. Основное применение этой модели связано с организацией виртуальной памяти, что позволяет операционной системе использовать для работы программ пространство памяти большее,

чем объем физической памяти. Для микропроцессоров i486 и Pentium размер возможной виртуальной памяти может достигать 4 Тбайт (терабайт).
- Особенности использования и реализации этих моделей зависят от режима работы микропроцессора:
- *Режим реальных адресов* или просто *реальный режим.* Это режим, в котором работал i8086. Наличие его в i486 и Pentium обусловлено тем, что фирма Intel старается обеспечить в новых моделях микропроцессоров функционирование программ, разработанных для ранних моделей микропроцессоров.
- *Защищенный режим.* Этот режим позволяет максимально реализовать все архитектурные идеи, заложенные в модели микропроцессоров Intel, начиная с i80286. Программы, разработанные для i8086 (реального режима), не могут функционировать в защищенном режиме. Одна из причин этого связана именно с особенностями формирования физического адреса в защищенном режиме.
- *Режим виртуального 8086.* Переход в этот режим возможен, если микропроцессор уже находится в защищенном режиме. Отличительной особенностью этого режима является возможность одновременной работы нескольких программ, разработанных для i8086. Несмотря на то, что микропроцессор находился в защищенном режиме, в режиме виртуального i8086 возможна работа программ реального режима. Это объясняется тем, что процесс формирования физического адреса для этих программ производится по правилам реального режима.

1.3.1. Сегментированная модель памяти.

Сегментация – механизм адресации, обеспечивающий существование нескольких независимых адресных пространств как в пределах одной задачи, так и в системе в целом для защиты задач от взаимного влияния. В основе механизма сегментации лежит понятие *сегмента,* который представляет собой независимый, поддерживаемый на аппаратном уровне блок памяти.

Когда мы рассматривали сегментные регистры, то отмечали, что для микропроцессоров Intel, начиная с i8086, принят особый подход к управлению памятью. Каждая программа в общем случае может состоять из любого количества сегментов, но непосредственный доступ она имеет только к трем основным сегментам: кода, данных и стека, – и от одного до трех дополнительных сегментов данных. Программа никогда не знает, по каким физическим адресам будут размещены ее сегменты. Этим занимается операционная система. Операционная система размещает сегменты программы в оперативной памяти по определенным физическим адресам, после чего помещает значения этих адресов в определенные места. Куда именно, зависит от режима работы микропроцессора. Так, в реальном режиме эти адреса помещаются непосредственно в соответствующие

сегментные регистры, а в защищенном режиме они размещаются в элементы специальной системной дескрипторной таблицы. Внутри сегмента программа обращается к адресам относительно начала сегмента линейно, то есть начиная с 0 и заканчивая адресом, равным размеру сегмента. Этот относительный адрес, или *смещение,* который микропроцессор использует для доступа к данным внутри сегмента, называется *эффективным.*

Рассмотрим порядок формирования физического адреса в реальном режиме. Под физическим адресом понимается адрес памяти, выдаваемый на шину адреса микропроцессора (см. рис. 3). Другое название этого адреса – *линейный адрес.* Эта двойственность в названии обусловлена наличием страничной модели организации оперативной памяти. Эти названия являются синонимами только при отключении страничного преобразования адреса (в реальном режиме страничная адресация всегда отключена). Страничная модель, как отмечалось выше, является надстройкой над сегментированной моделью. В страничной модели линейный адрес и физический адрес имеют разные значения. Чуть ниже будет обсуждаться рис. 3, на котором показан порядок формирования адреса в реальном режиме работы микропроцессора. Обратите внимание на наличие в этой схеме устройства страничного преобразования адреса. Это устройство предназначено для того, чтобы совместить две принципиально разные модели организации оперативной памяти и выдать на шину адреса истинное значение физического адреса памяти.

1.3.2. Формирование физического адреса в реальном режиме.

В реальном режиме механизм адресации физической памяти имеет следующие характеристики:

- диапазон изменения физического адреса от 0 до 1 Мбайт. Эта величина определяется тем, что шина адреса i8086 имела 20 линий;
- максимальный размер сегмента 64 Кбайт. Это объясняется 16–разрядной архитектурой i8086. Нетрудно подсчитать, что максимальное значение, которое могут содержать 16–разрядные регистры, составляет $2^{16} – 1$, что применительно к памяти и определяет величину 64 Кбайт;
- для обращения к конкретному физическому адресу оперативной памяти необходимо определить адрес начала сегмента (сегментную составляющую) и смещение внутри сегмента. Но следует помнить, что сегментная составляющая адреса (или *база сегмента)* представляет собой всего лишь 16–битное значение, помещенное в один из сегментных регистров. Максимальное значение, которое при этом получается, соответствует $2^{16} – 1$. Если так рассуждать, то получается, что адрес начала сегмента может быть только в диапазоне 0–64 Кбайт от начала оперативной памяти. Возникает вопрос о том, как адресовать остальную часть оперативной памяти вплоть до 1 Мбайт с учетом того, что размер самого сегмента не превышает 64 Кбайт.

Дело в том, что в сегментном регистре содержатся только старшие 16 бит физического адреса начала сегмента. Недостающие младшие четыре бита 20–битного адреса получаются сдвигом значения в сегментном регистре влево на 4 разряда. Эта операция сдвига выполняется аппаратно и для программного обеспечения абсолютно прозрачна. Получившееся 20–битное значение и является настоящим физическим адресом, соответствующим началу сегмента. Что касается второго компонента, участвующего в образовании физического адреса некоторого объекта в памяти, – смещения – то оно представляет собой 16–битное значение. Это значение может содержаться явно в команде либо косвенно в одном из регистров общего назначения. В микропроцессоре эти две составляющие складываются на аппаратном уровне, в результате чего получается физический адрес памяти размерностью 20 бит. Данный механизм образования физического адреса позволяет сделать программное обеспечение перемещаемым, то есть не зависящим от конкретных адресов загрузки его в оперативной памяти. Он показан на рис. 2.3.

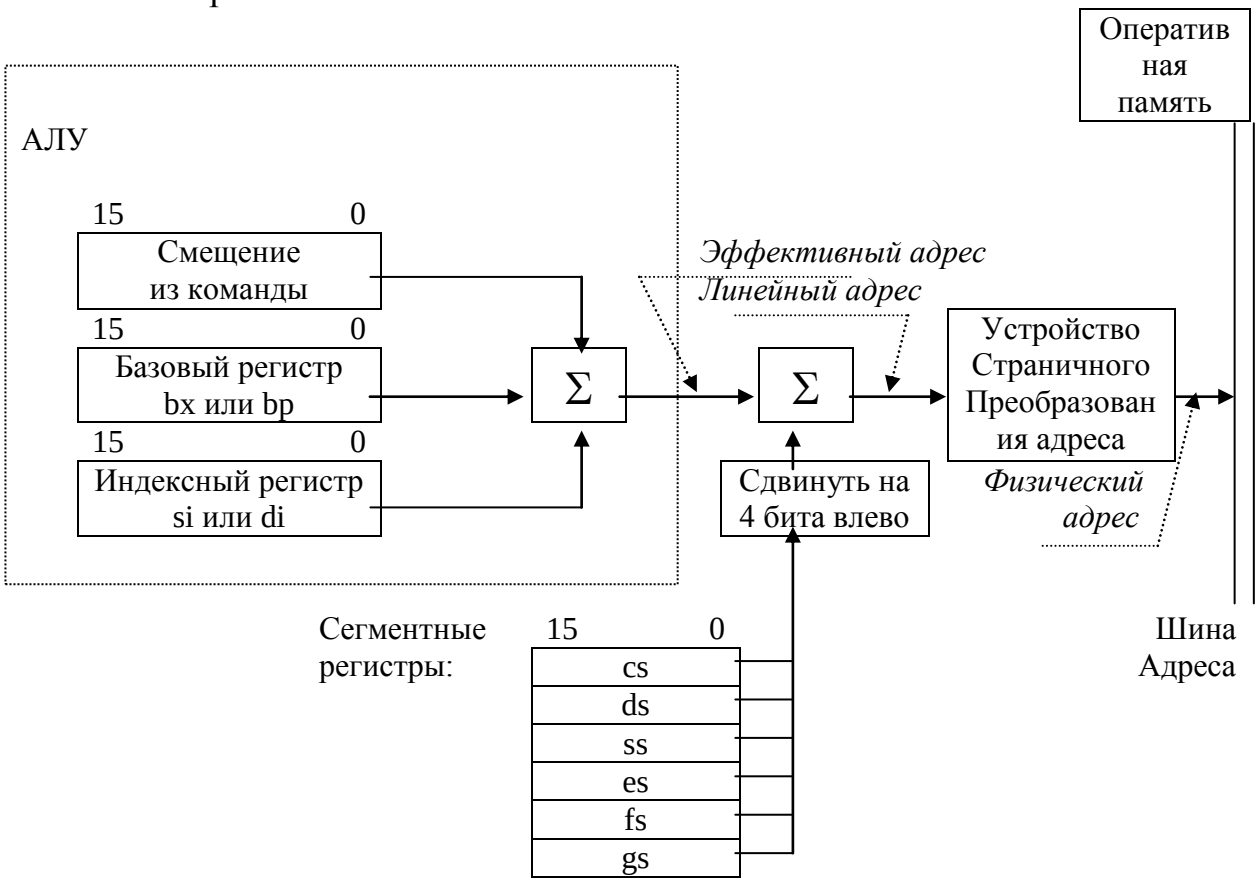


Рис. 3. Механизм формирования физического адреса в реальном режиме

На рис. 3 хорошо видно, как формируется некоторый целевой физический адрес: сегментная часть извлекается из одного из сегментных регистров, сдвигается на четыре разряда влево и суммируется со смещением. В свою очередь видно, что значение смещения можно получить минимум из

одного и максимум из трех источников: из значения смещения в самой машинной команде и (или) из содержимого одного базового и (или) одного индексного регистра. Количество источников, участвующих в формировании смещения, определяется кодированием конкретной машинной команды, и если таких источников несколько, то значения в них складываются. Не стоит волноваться насчет того, что существует несоответствие размеров шины адреса микропроцессора i486 или Pentium (32 бита) и 20–битного значения физического адреса реального режима. Пока микропроцессор находится в реальном режиме, старшие 12 линий шины адреса попросту недоступны, хотя при определенных условиях и существует возможность работы с первыми 64 Кбайт оперативной памяти, лежащими сразу после первого мегабайта.

Недостатки такой организации памяти:

- сегменты бесконтрольно размещаются с любого адреса, кратного 16 (так как содержимое сегментного регистра аппаратно смещается на 4 разряда). Как следствие, программа может обращаться по любым адресам, в том числе и реально не существующим;
- сегменты имеют максимальный размер 64 Кбайт;
- сегменты могут перекрываться с другими сегментами.

Желанием ввести в архитектуру средства, позволяющие избавиться от указанных недостатков, в частности, и обусловлено появление защищенного режима.

1.4. Типы данных.

Понятие *типа данных* носит двойственный характер. С точки зрения размерности микропроцессор аппаратно поддерживает следующие основные типы данных (рис. 4):

- *Байт* – восемь последовательно расположенных битов, пронумерованных от 0 до 7, при этом бит 0 является самым младшим значащим битом.
- *Слово* – последовательность из двух байт, имеющих последовательные адреса. Размер слова – 16 бит; биты в слове нумеруются от 0 до 15. Байт, содержащий нулевой бит, называется *младшим байтом,* а байт, содержащий 15–й бит – *старшим байтом.* Микропроцессоры Intel имеют важную особенность – младший байт всегда хранится по меньшему адресу. *Адресом слова* считается адрес его младшего байта. Адрес старшего байта может быть использован для доступа к старшей половине слова.
- *Двойное слово* – последовательность из четырех байт (32 бита), расположенных по последовательным адресам. Нумерация этих бит производится от 0 до 31. Слово, содержащее нулевой бит, называется *младшим словом,* а слово, содержащее 31–й бит, – *старшим словом.* Младшее слово хранится по меньшему адресу. *Адресом двойного слова*

считается адрес его младшего слова. Адрес старшего слова может быть использован для доступа к старшей половине двойного слова.

- *Учетверенное слово* – последовательность из восьми байт (64 бита), расположенных по последовательным адресам. Нумерация бит производится от 0 до 63. Двойное слово, содержащее нулевой бит, называется *младшим двойным словом,* а двойное слово, содержащее 63–й бит, – *старшим двойным словом.* Младшее двойное слово хранится по меньшему адресу. *Адресом учетверенного слова* считается адрес его младшего двойного слова. Адрес старшего двойного слова может быть использован для доступа к старшей половине учетверенного слова.

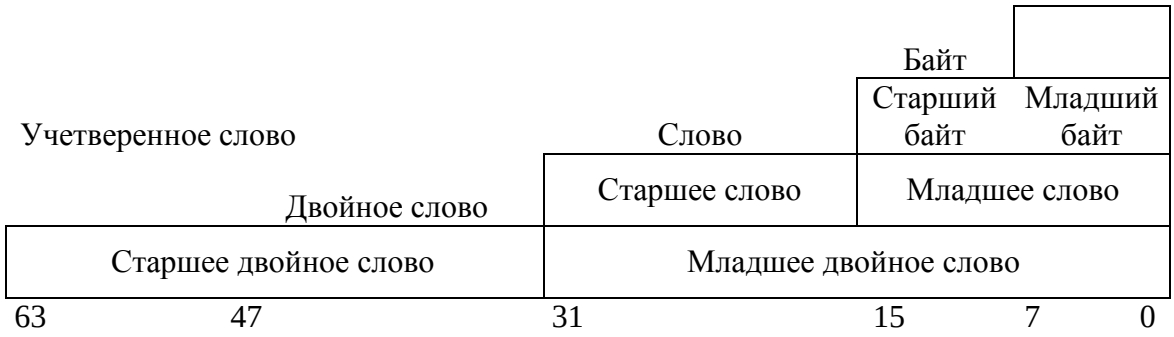


Рис. 4. Основные типы данных микропроцессора

Кроме трактовки типов данных с точки зрения их разрядности, микропроцессор на уровне команд поддерживает *логическую* интерпретацию этих типов (рис. 5).

- *Целый тип со знаком* – двоичное значение со знаком размером 8, 16 или 32 бита. Знак в этом двоичном числе содержится в 7, 15 или 31–м бите соответственно. Ноль в этих битах в операндах соответствует положительному числу, а единица – отрицательному. Отрицательные числа представляются в дополнительном коде. Числовые диапазоны для этого типа данных следующие:

  8-разрядное целое – от – 128 до +127;
  16-разрядное целое – от – 32 768 до +32 767;
  32-разрядное целое – от -2^{31} до $+2^{31} - 1$.

- *Целый тип без знака* – двоичное значение *без знака* размером 8, 16 или 32 бита. Числовой диапазон для этого типа следующий:

  байт – от 0 до 255;
  слово – от 0 до 65 535;
  двойное слово – от 0 до $2^{32} - 1$.

- *Указатель на память* бывает двух типов.

  - *Ближний тип* – 32-разрядный логический адрес, представляющий собой относительное смещение в байтах от начала сегмента. Эти указатели могут также использоваться в сплошной (плоской) модели памяти, где сегментные составляющие одинаковы.

- *Дальний тип* – 48-разрядный логический адрес, состоящий из двух частей: 16-разрядной сегментной части – *селектора* и 32-разрядного смещения.

- *Цепочка* представляет собой некоторый непрерывный набор байтов, слов или двойных слов максимальной длиной до 4 Гбайт.
- *Битовое поле* представляет собой непрерывную последовательность бит, в которой каждый бит является независимым и может рассматриваться как отдельная переменная. Битовое поле может начинаться с любого бита любого байта и содержать до 32 бит.
- *Неупакованный двоично-десятичный тип* – байтовое представление десятичной цифры от 0 до 9. Неупакованные десятичные числа хранятся как байтовые значения без знака по одной цифре в каждом байте. Значение цифры определяется младшим полубайтом.
- *Упакованный двоично-десятичный тип* представляет собой упакованное представление двух десятичных цифр от 0 до 9 в одном байте. Каждая цифра хранится в своем полубайте. Цифра в старшем полубайте (биты 4–7) является старшей.

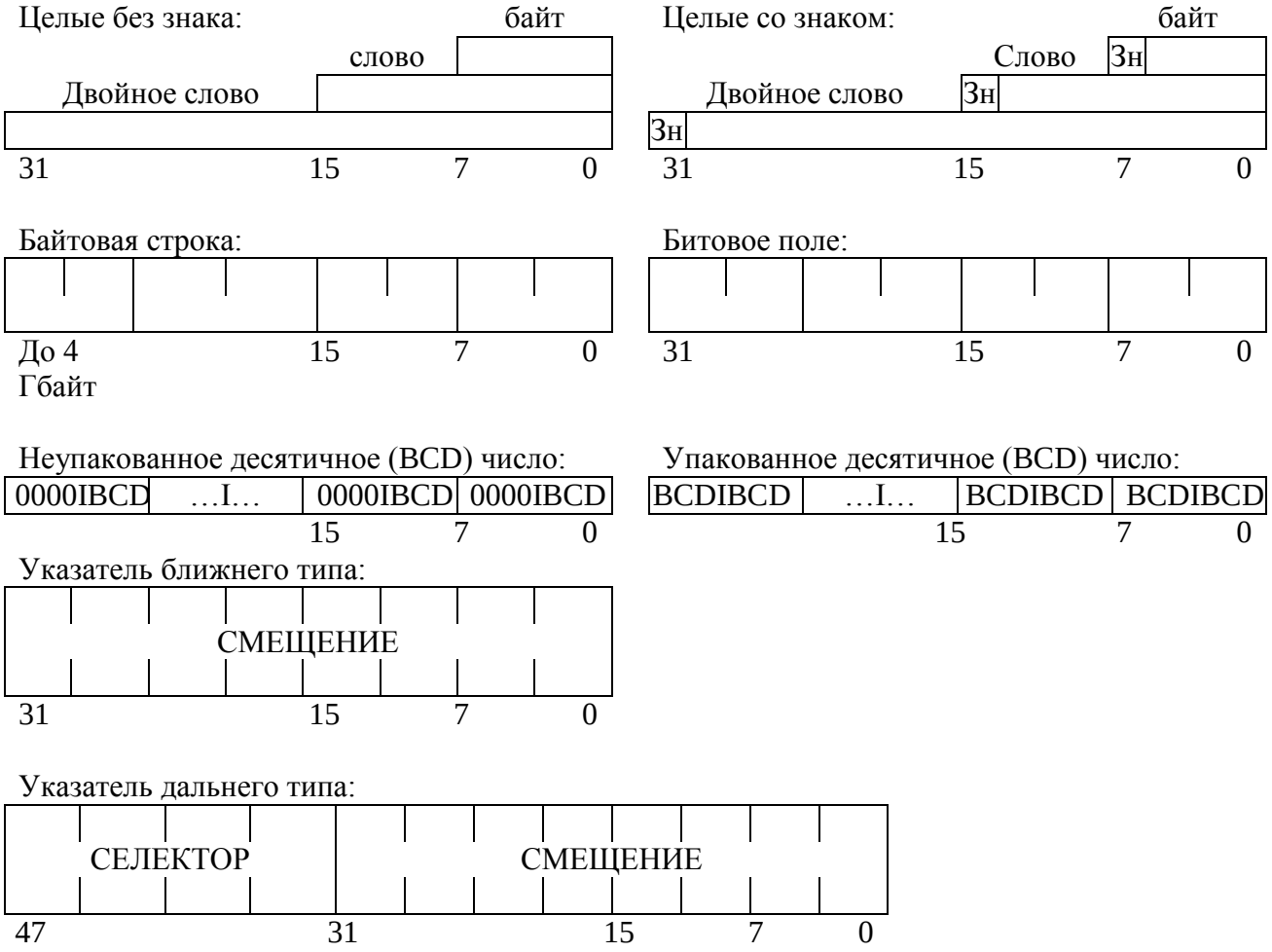


Рис. 5. Основные логические типы данных микропроцессора

1.5. Формат команд.

Программирование на уровне машинных команд – это тот минимальный уровень, на котором еще возможно программирование компьютера. Система машинных команд должна быть достаточной для того, чтобы реализовать требуемые действия, выдавая указания аппаратуре машины. Каждая машинная команда состоит из двух частей: *операционной части,* определяющей «что делать?» и *операндной части,* определяющей объекты обработки, то есть то «над чем делать?».

Вся эта информация, разумеется, должна быть определенным образом закодирована и формализована.

В машинную команду микропроцессора явно или неявно входят следующие элементы:

- *поле префиксов* – элемент команды, который уточняет либо модифицирует действие этой команды в следующих аспектах:
  - замена сегмента, если нас по какой–либо причине не удовлетворяет сегмент по умолчанию;
  - изменение размерности адреса;
  - изменение размерности операнда;
  - указание на необходимость повторения данной команды;
- *поле кода операции,* определяющее действие данной команды. Одной и той же команде могут соответствовать несколько кодов операций в зависимости от ее операндов;
- *поле операндов;* содержит от 0 до 2 элементов.

Важной особенностью машинных команд является то, что они не могут манипулировать одновременно двумя операндами, находящимися в оперативной памяти. Это означает, что в команде могут быть использованы один регистр и/или регистр и некоторый операнд, который может либо непосредственно находиться в команде, либо располагаться в памяти.

По этой причине возможны только следующие сочетания операндов в команде:

- регистр – регистр;
- регистр – память;
- память – регистр;
- непосредственный операнд – регистр;
- непосредственный операнд – память.

1.6. Способы адресации (для МП I8086, К1810ВМ86).

МП ВМ86 предоставляет множество способов доступа к операндам, с которыми должна работать программа. Операнды могут содержаться в регистрах, в самих командах, в памяти или в портах ввода-вывода. Режимы

адресации можно разделить на семь групп: 1) регистровая адресация; 2) непосредственная адресация; 3) прямая адресация; 4) косвенная регистровая адресация; 5) адресация по базе; 6) прямая адресация с индексированием; 7) адресация по базе с индексированием.

МП выбирает один из семи режимов адресации по значению поля режима команды. Ассемблер присваивает то или иное значение полю режима в зависимости от вида операндов в исходной программе.

В табл. 2 приведены форматы операндов языка ассемблера для всех семи режимов адресации, и для каждого формата указано, какой из регистров сегмента используется для вычисления физического адреса.

Таблица 3. Режимы адресации МП К1810ВМ86

Режим адресации	**Формат операнда**	**Регистр сегмента**
Регистровый	Регистр	не используется
Непосредственный	Данное	не используется
Прямой	Сдвиг Метка	DS DS
Косвенный регистровый	[BX] [BP] [DI] [SI]	DS SS DS DS
По базе	[BX]+сдвиг [BP]+сдвиг	DS SS
Прямой с индексированием	[DI]+сдвиг [SI]+сдвиг	DS DS
По базе с индексированием	[BX][SI]+сдвиг [BX][DI]+сдвиг [BP][SI]+сдвиг [BP][DI]+сдвиг	DS DS SS SS

Важное замечание: при исполнении команд, манипулирующих строками предполагается, что регистр DI указывает на ячейку дополнительного сегмента, а не сегмента данных. В качестве регистра сегмента используется регистр ES.

Примечания:

1) компонент "сдвиг" при адресации по базе с индексированием необязателен;

2) операнд "регистр" может быть любым 8- или 16-битовым регистром, кроме регистра PI;

3) операнд "данное" может быть 8- или 16-битовым значением константы;

4) компонент "сдвиг" может быть 8- или 16-битовым значением смещения со знаком.

1.6.1. Регистровая адресация.

При регистровой адресации МП извлекает операнд из регистра (или загружает его в регистр). Например, команда MOV AX,CX копирует 16-битовое содержимое регистр CX в регистр AX.

1.6.2. Непосредственная адресация.

Непосредственная адресация позволяет указывать 8- или 16-битовое значение в качестве операнда-источника. Эта константа содержится в команде, а не в регистре или в ячейке памяти. Например, команды:

```
MOV CX,500;  Загружает в регистр CX десятичное
MOV CX,500D;  число,
MOV CL,11001011B;  двоичное число (в регистр CL),
MOV CX,0AC69H;  шестнадцатеричное число.
```

1.6.3. Прямая адресация.

При прямой адресации исполнительный адрес является составной частью команды. МП добавляет этот адрес к сдвинутому на 4 двоичных разряда содержимому регистра сегмента (например DS) и получает 20-битовый физический адрес. Обычно прямая адресация применяется, если операндом служит метка. Например, команда

```
MOV AX,TABLE
```

загружает содержимое ячейки памяти TABLE в регистр AX.

1.6.4. Косвенная регистровая адресация.

При косвенной регистровой адресации исполнительный адрес операнда содержится в базовом регистре BX, регистре указателя базы BP или индексном регистре (SI или DI). Косвенные регистровые операнды надо заключать в квадратные скобки, чтобы отличить их от регистровых операндов. Например, команда

```
MOV AX,[BX]
```

загружает в регистр AX содержимое ячейки памяти, адресуемой значением регистра BX.

1.6.5. Адресация по базе.

При адресации по базе ассемблер вычисляет исполнительный адрес с помощью сложения значения сдвига с содержимым регистров BX и BP.

Регистр BX удобно использовать при доступе к структурированным записям данных, расположенным в разных областях памяти. В этом случае базовый адрес записи помещается в базовый регистр BX и доступ к ее отдельным элементам осуществляется по их сдвигу относительно базы. Для доступа к разным записям одной и той же структуры достаточно соответствующим образом изменить содержимое базового регистра.

Пример:
MOV AX,[BP]+4; Стандартная форма записи,
MOV AX, 4[BP]; но сдвиг можно указывать как на первом
MOV AX,[BP+4]; месте, так и внутри скобок

1.6.6. Прямая адресация с индексированием.

При прямой адресации с индексированием исполнительный адрес вычисляется как сумма значений сдвига и индексного регистра (DI или SI). Удобна для доступа к элементам таблицы, когда сдвиг указывает на начало таблицы, а индексный регистр - на ее элемент.

Например, если B_TABLE - таблица байтов, то команды
MOV DI,2
MOV AL,B_TABLE[DI]
загрузят третий элемент таблицы B_TABLE в регистр AL.

1.6.7. Адресация по базе с индексированием.

Исполнительный адрес вычисляется как сумма значений базового регистра, индексного регистра и, возможно, сдвига. Так как в этом режиме адресации складываются два отдельных смещения, то он удобен для адресации двумерных массивов, когда базовый регистр содержит начальный адрес массива, а значения сдвига и индексного регистра суть смещения по строке и столбцу.

MOV AX,[BX][DI]+2

Суммарные сведения о режимах адресации МП К1810ВМ86 приведены в табл. 3.

Таблица 4. Кодирование режимов адресации в МП К1810ВМ86

Поле r/m	**Поле md**				
	00	**01**	**10**	**11**	
				w=0	**w=1**
000	BX+SI	BX+SI+D8	BX+SI+D16	AL	AX
001	BX+DI	BX+DI+D8	BX+DI+D16	CL	CX
010	BP+SI	BP+SI+D8	BP+SI+D16	DL	DX
011	BP+DI	PB+DI+D8	BP+DI+D16	BL	BX
100	SI	SI+D8	SI+D16	AH	SP
101	DI	DI+D8	DI+D16	CH	BP
110	D16	BP+D8	BP+D16	DH	SI
111	BX	BX+D8	BX+D16	BH	DI

Примечание: D8 = disp L; D16 = disp H disp L.

1.7. Некоторые итоги:

1. Понимание архитектуры ЭВМ является ключевым для изучения ассемблера. Это касается любого типа компьютера. Структура ассемблера, формат его команд, адресация операндов и т. д. полностью отражают особенности архитектуры компьютера. Есть общие архитектурные свойства, присущие всем современным машинам фон–неймановской архитектуры, и есть частные свойства, присущие конкретному типу компьютеров.
2. Целью изучения архитектуры является:
   - выявление набора доступных для программирования регистров, их функционального назначения и структуры;
   - понимание организации оперативной памяти и порядка ее использования;
   - знакомство с типами данных;
   - изучение формата машинных команд;
   - выяснение организации обработки прерываний. Микропроцессор содержит 32 доступных тем или иным образом регистра. Они делятся на пользовательские и системные регистры.
3. Пользовательские регистры имеют определенное функциональное назначение. Среди них особо нужно выделить регистр флагов eflags и регистр указателя команды eip. Назначение регистра eflags – отражать состояние микропроцессора после выполнения последней машинной команды. Регистр eip содержит адрес следующей выполняемой машинной команды. Доступ к этим регистрам, в силу их специфики, со стороны программ пользователя ограничен.
4. Микропроцессор имеет три режима работы:
   - реальный режим, который использовался для i8086 и поддерживается до сих пор для обеспечения совместимости программного обеспечения;
   - защищенный режим, который впервые появился в i80286;
   - режим виртуального i8086. Обеспечивает полную эмуляцию микропроцессора i8086, позволяя при этом организовать многозадачную работу нескольких таких программ.
5. Микропроцессор имеет сложную систему управления памятью, работа которой зависит от режима микропроцессора.
6. Микропроцессор благодаря гибкой системе команд поддерживает довольно большую номенклатуру типов данных.

2. СИСТЕМА КОМАНД.

Базовый набор команд МП ВМ86 включает основные команды без учета их модификации за счет использования различных способов адресации. В наборе команд можно выделить следующие группы команд: 1) команды пересылки данных; 2) арифметические команды; 3) команды манипулирования битами; 4) команды передачи управления; 5) команды обработки строк; 6) команды прерывания;

Большинство команд могут быть выполнены в байтовом режиме. В этом случае в качестве операнда используется не слово, а байт данных. Описание и коды команд приведены в табл. 4.

2.1. Команды пересылки данных.

Команды пересылки данных осуществляют обмен данными и адресами между регистрами и ячейками памяти или портами ввода - вывода.

Команды пересылки данных не влияют на значение регистра флагов.

а) команды общего назначения

Основная команда общего назначения MOV (MOVE - переслать) осуществляет пересылку байта или слова между регистром и ячейкой памяти или между двумя регистрами, а также пересылает непосредственно адресуемое значение в регистр или ячейку памяти.

В команде MOV исключаются следующие сочетания операндов:

1. Нельзя осуществить непосредственную пересылку данных из одной ячейки памяти в другую.
2. Нельзя загрузить непосредственную информацию в регистр сегмента.
3. Нельзя непосредственно переслать значение одного регистра сегмента в другой.
4. Нельзя использовать регистр CS в качестве приемника в команде пересылки.

Команды PUSH и POP предназначены для работы со стеком. Команда PUSH помещает содержимое регистра или ячейки памяти размером 16-бит в вершину стека. Команда POP, наоборот, снимает слово с вершины стека и помещает его в регистр или ячейку памяти. Под вершиной стека понимается ячейка в сегменте стека, адрес которой содержится в указателе стека SP. Т.к. стек "растет" по направлению к младшим адресам памяти, то первое помещаемое в стек слово запоминается в ячейке стека с наибольшим адресом. На рис. 6 показаны состояния стека и его указателя до и после использования команд PUSH и POP.

Необходимо помнить, что с помощью команды POP слова будут извлекаться в порядке, обратном их помещению.

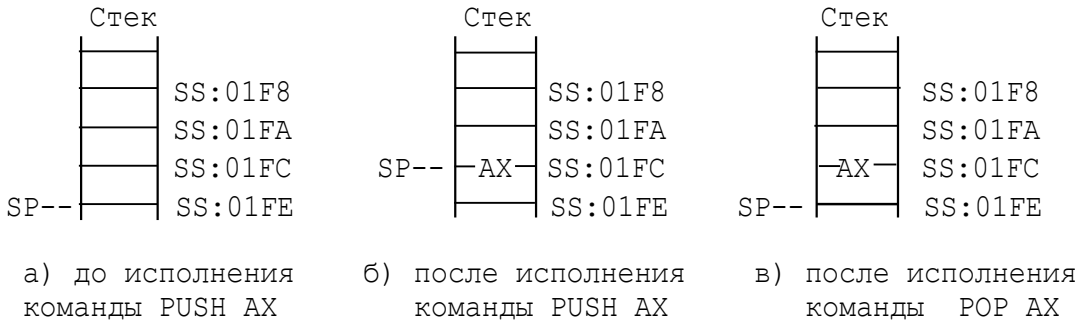


Рис. 4. Воздействие команд PUSH и POP на стек

Команда XCHG (exchange - обменять) меняет между собой значения двух регистров или регистра и ячейки памяти. Однако она не может выполнить обмен значений регистров сегмента.

б) команды ввода-вывода IN и OUT

Команды ввода-вывода используются для взаимодействия с периферийными устройствами системы. Операндом "порт" может быть десятичное значение от 0 до 255, что позволяет адресоваться к 256 устройствам. В качестве операнда "порт" можно использовать регистр DX, что позволяет изменять номер порта и адресоваться косвенно к 65536 портам:

```
IN  AL,200;  Ввести байт из порта 200
IN  AL,PORT_VAL;  или из порта, указанного константой.
OUT 30H,AX;  Вывести слово в порт 30H
OUT DX,AX;  или в порт, указанный в DX.
```

в) команды пересылки адреса

Команды пересылки адреса передают не содержимое переменных, а их адреса.

Команда LEA (load effective address - загрузить исполнительный адрес) пересылает смещение ячейки памяти в любой 16-битовый регистр общего назначения, регистр указателя или индексный регистр. В отличие от команды MOV с операцией OFFSET, операнд "память 16" может индексироваться, что дает возможность осуществить гибкую адресацию.

Команда LDS (load pointer using DS - загрузить указатель, используя DS) считывает из памяти двойное слово и загружает первые 16 битов в заданный регистр, а следующие 16 битов - в регистр DS.

Команда LES (load pointer using ES - загрузить указатель, используя ES) идентична команде LDS, но загружает номер блока в регистр ES, а не в DS.

2.2. Арифметические команды.

МП может выполнять арифметические команды над двоичными числами со знаком или без знака, а также над десятичными числами без знака (упакованными и неупакованными).

Таблица 5. Влияние арифметических команд на флаги

Команды	**Флаги**					
	OF	SF	ZF	AF	PF	CF
ADD ADC SUB SBB CMP NEG	+	+	+	+	+	+
INC DEC	+	+	+	+	+	-
MUL IMUL	+	+	?	?	?	?
DIV IDIV	?	?	?	?	?	?

Примечания: "+" - результат операции влияет на флаг;
"-" - результат операции не влияет на флаг;
"?" - флаг не определен.

а) команды сложения

Команды ADD (add - сложить) и ADC (add with carry - сложить с переносом) могут складывать как 8-, так и 16-битовые операнды. Команда ADD складывает содержимое операнда-источника и операнда-приемника и помещает результат в операнд-приемник. Команда ADC делает то же, что и команда ADD, но при сложении используется также флаг переноса CF.

Обычно пара команд ADD и ADC используется для сложения чисел с повышенной точностью. Одна из них (ADD) складывает младшие части чисел с повышенной точностью, другая (ADC) используется для сложения старших частей значений повышенной точности. Пример:

```
ADD AX,CX;  Сначала сложить младшие 16 бит,  а затем
ADC BX,DX;  старшие 16 бит; Результат в регистрах BX и AX.
```

Команда INC (increment - прирастить) добавляет 1 к содержимому регистра или ячейки памяти, но в отличие от команды ADD не воздействует на флаг переноса CF. Команда INC удобна для приращения счетчиков в циклах команд. Ее можно использовать для приращения значения индексного регистра или указателя при доступе к последовательно расположенным ячейкам памяти.

б) команды вычитания

Команды SUB (substract - вычесть) и SBB (substract with borrow - вычесть с заемом) аналогичны соответственно командам сложения ADD и ADC, только при вычитании флаг переноса CF действует как признак заёма. Как и в командах сложения первая команда SUB вычитает младшие биты чисел с повышенной точностью, а команда SBB - старшие биты чисел с

повышенной точностью. Пример:

```
SUB AX,BX;  Вычесть младшие 16 битов,
SBB BX,DX;  а затем - старшие 16 битов.
```

Команда DEC (decrement - уменьшить) вычитает 1 из содержимого регистра или ячейки памяти, не воздействуя при этом на флаг переноса CF. Часто используется в циклах для уменьшения значения счетчика, а также при доступе к последовательно расположенным ячейкам памяти для уменьшения значения индексного регистра или указателя.

Команда NEG вычитает значение операнда-приемника из нулевого значения и тем самым формирует его дополнение до двух.

Команда CMP (compare - сравнить) используется для сравнения операндов. Подобно команде SUB команда CMP вычитает операнд-источник из операнда-приемника и в зависимости от результата изменяет состояния флагов. В отличие от команды SUB команда CMP не сохраняет результат вычитания, т.е. операнды не изменяются.

в) команды умножения

Команда MUL (multiply - умножить) умножает числа без знака, а IMUL (integer multiply - умножить целые числа) - числа со знаком. Обе команды умножают как байты, так и слова. Команды имеют формат:

```
MUL  источник,
IMUL  источник,
```

где источник - регистр общего назначения или ячейка памяти размером в байт или слово. В качестве второго операнда команды MUL и IMUL используют содержимое регистра AL (при операциях над байтами) или регистра AX (при операциях над словами). Произведение имеет двойной размер и размещается следующим образом:

- при умножении байтов 16-битовое произведение находится в регистрах AH (старший байт) и AL (младший байт).
- при умножении слов 32-битовое произведение находится в регистрах DX (старшее слово) и AX (младшее слово).

При завершении исполнения этих команд флаги переноса CF и переполнения OF показывают, какая часть произведения существенна для последующих операций. После исполнения команды MUL флаги CF и OF равны 0, если старшая половина произведения равна 0; в противном случае оба эти флага равны 1. После исполнения команды IMUL флаги CF и OF равны 0, если старшая половина произведения представляет собой лишь расширение знака младшей половины. В противном случае они равны 1.

Команды MUL и IMUL не позволяют в качестве операнда использовать непосредственное значение. Такое значение перед умножением надо загрузить в регистр или в ячейку памяти.

г) команды деления

Команда DIV (divide - разделить) выполняет деление чисел без знака, а команда IDIV (integer divide - разделить целые числа) выполняет деление чисел со знаком. Эти команды имеют формат:

DIV источник,

IDIV источник,

где источник - делитель размером в байт или слово, находящийся в регистре или ячейке памяти. Делимое должно иметь двойной размер и храниться в регистрах AH и AL (при делении на 8-битовое число) или в регистрах AX и DX (при делении на 16-битовое число). Результаты деления размещаются следующим образом:

- если операнд-источник представляет собой байт, то частное помещается в регистр AL, а остаток в регистр AH;
- если операнд-источник представляет собой слово, то частное помещается в регистр AX, а остаток - в регистр DX.

Обе команды оставляют состояние регистра флагов неопределенным, но если частное не помещается в регистр-приемник (AL или AX), то МП генерирует прерывание типа 0 (деление на 0).

Команды DIV и IDIV не позволяют прямо разделить на непосредственное значение; его надо предварительно загрузить в регистр или ячейку памяти.

2.3. Команды манипулирования битами.

Эти команды манипулируют группами битов в регистрах или ячейках памяти. В табл. 4 они разделены на 3 группы: логические команды, команды сдвига и циклического сдвига.

Таблица 6. Влияние команд манипулирования битами на флаги

Команды	**Флаги**					
	OF	SF	ZF	AF	PF	CF
AND OR XOR TEST	0	0	?	+	+	+
*1 SHL SHR	+	+	?	+	+	+
*n SHL SHR	?	+	?	+	+	+
SAR	0	+	?	+	+	+
*1 ROL ROR RCL RCR	+	+	-	-	-	-
*n ROL ROR RCL RCR	?	+	-	-	-	-

Примечания: *1 - одиночный сдвиг;
*n - могоразрядный сдвиг.

а) логические команды

Эти команды действуют по правилам формальной логики.

Команда AND маскирует (обнуляет) некоторые биты. При исполнении команды AND биты операнда-приемника становятся равными 0 всюду, где операнд-источник содержит 0, и сохраняются там, где операнд-источник содержит 1.

Команда OR полагает равными 1 те биты операнда-источника, в которых хотя бы один из операндов содержит 1. Эта команда обычно используется для принудительного присваивания 1 заданным битам.

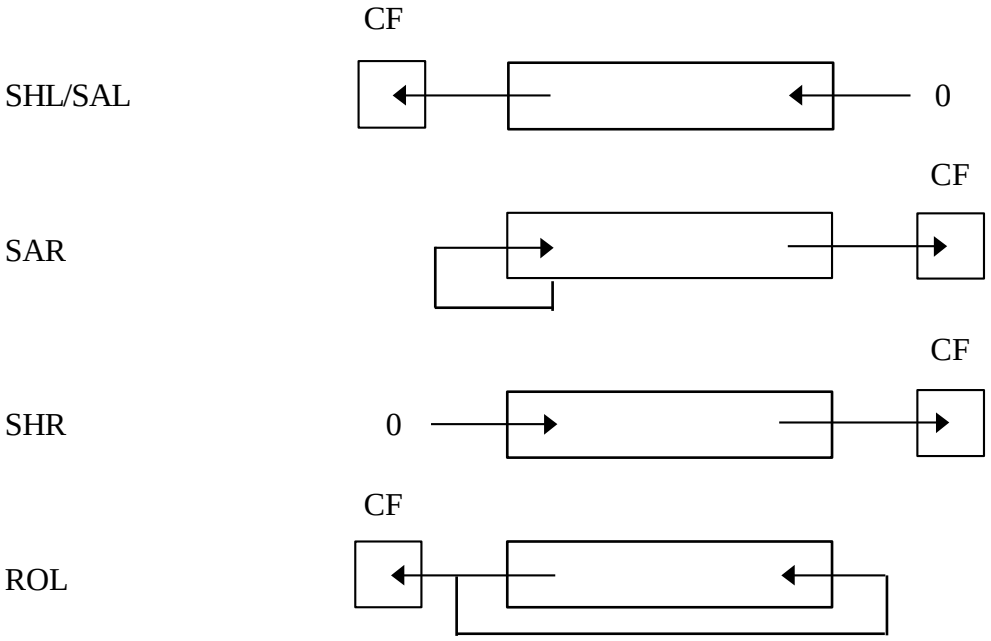
Команда XOR используется, если надо выяснить, в каких битах значения операндов различаются, или надо инвертировать состояния заданных битов. Команда XOR полагает равными 1 все те биты приемника, в позициях которых операнды имеют различные значения. Если оба операнда содержат в данной позиции равные значения, то команда XO обнуляет данные биты приемника.

Команда NOT (НЕ) инвертирует (дополнение до 1) состояние каждого бита регистра (ячейки памяти) и не воздействует на флаги.

Команда TEST (test - проверить) выполняет операцию AND над операндами, но воздействует только на флаги и не изменяет значения операндов.

б) команды сдвига и циклического сдвига

Команды сдвига и циклического сдвига делятся на две группы. Логические команды сдвигают операнд, не считаясь со знаком; они используются для действий над числами без знака или над нечисловыми значениями. Арифметические команды сохраняют старший, знаковый бит операнда; они используются для действий над числами со знаком. Действие этих команд показано на рис. 5.



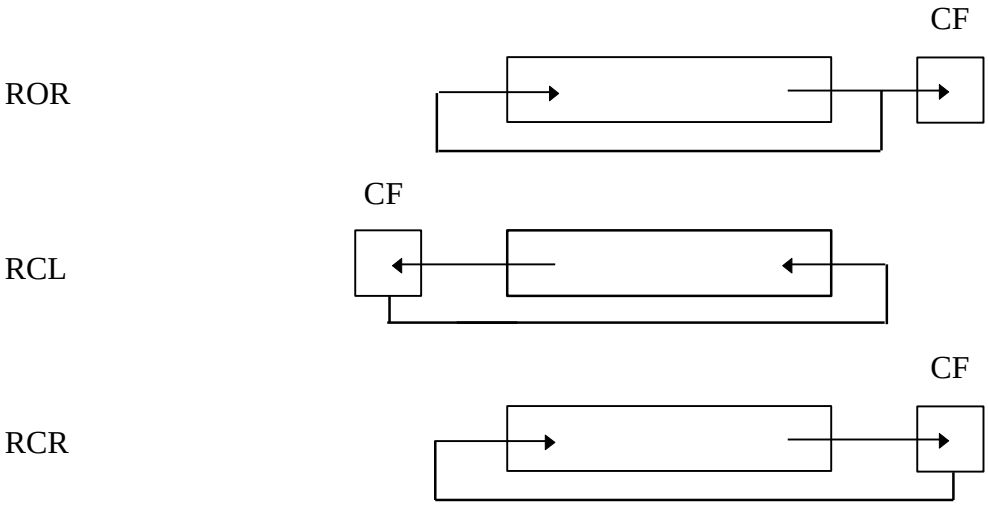


Рис. 5. Команды сдвига и циклического сдвига

Команды SAL (shift aritmetic left - сдвинуть влево арифметически) и SAR (shift aritmetic right - сдвинуть вправо арифметически) сдвигают числа со знаком. Команда SAR сохраняет знак операнда, репродуцируя его при выполнении сдвига. Команда SAL не сохраняет знак, но заносит 1 во флаг переполнения OF в случае изменения знака операнда. При каждом сдвиге операнда команда SAL заносит 0 в освобождающийся бит этого операнда.

Команды SHL (shift logical left - сдвинуть влево логически) и SAR (shift iogical right - сдвинуть вправо логически) сдвигают числа без знака. Команда SHL идентична команде SAL. Команда SHR сдвигает операнд вправо. При каждом сдвиге операнда команда SHR заносит 0 в старший бит этого операнда.

Команды циклического сдвига воздействуют только на флаги CF и OF. Команды циклического сдвига похожи на команды сдвига, но в отличие от последних сохраняют сдвинутые за пределы операнда биты, помещая их обратно в операнд. Как и при исполнении команд сдвига, сдвинутый за пределы операнда бит запоминается во флаге переноса CF.

При исполнении команды ROL (rotate left - сдвинуть влево циклически) и ROR (rotate right - сдвинуть вправо циклически) вышедший за пределы операнда бит входит в него с противоположного конца. При исполнении команды RCL (rotate left through carry - сдвинуть влево циклически вместе с флагом переноса) и RCR (rotate right through carry - сдвинуть вправо циклически вместе с флагом переноса) в противоположный конец операнда помещается значение флага переноса CF.

2.4. Команды передачи управления.

Команды передачи управления обеспечивают переход из одной части программы в другую. Эти команды можно подразделить на три группы:

команды безусловной передачи управления, команды условной передачи управления и команды управления циклами.

Команды передачи управления не воздействуют на регистр флагов.

а) команды вызова процедуры CALL и возврата из процедуры RET

Команды, обеспечивающие выполнение процедур, должны выполнять три функции:

1. Обеспечить сохранение содержимого указателя команд IP. Когда процедура исполнена, находившийся в этом указателе адрес (адрес возврата) используется МП для возврата к месту вызова.

2. Заставить МП начать исполнение процедуры.

3. Использовать сохраненное содержимое указателя команд IP для возврата в программу и обеспечить продолжение ее исполнения с этого места.

Эти функции выполняются двумя командами: CALL (call a procedure

- вызвать процедуру) RET (return from procedure - возвратиться из процедуры).

Команда CALL помещает в стек адрес возврата, занимающий 16 битов, если процедура определена с атрибутом NEAR, и 32 бита, если она определена с атрибутом FAR. Команда CALL имеет формат

CALL имя,

где "имя" - имя вызываемой процедуры. Если "имя" имеет атрибут NEAR, то команда CALL помещает смещение адреса следующей команды в стек. Если процедура "имя" имеет атрибут FAR, то команда CALL помещает в стек содержимое регистра CS, а затем смещение адреса.

После сохранения адреса возврата команда CALL загружает смещение адреса метки "имя" в указатель команд IP. Если процедура имеет атрибут FAR, то команда CALL загружает также номер блока метки "имя" в регистр CS.

Команда RET заставляет МП возвратиться из процедуры в программу, вызвавшую эту процедуру, делая это "откатом" всего, что сделала команда CALL. Команда RET обязательно должна быть последней командой процедуры, исполняемой МП.

Команда RET извлекает из стека адрес возврата. Если процедура имеет атрибут NEAR, то команда RET извлекает из стека одно слово и помещает его в указатель команд IP. Если процедура имеет атрибут FAR, то команда RET извлекает из стека два слова: сначала смещение адреса для загрузки в указатель команд IP, а затем номер блока для загрузки в регистр CS.

Можно осуществить и косвенный вызов процедуры через регистр или ячейку памяти. При косвенных вызовах через ячейку памяти МП извлекает значение указателя команд для процедуры из сегмента данных, если не используется регистр BP или не указана замена сегмента. Если для адресации ячейки памяти используется регистр BP, то МП извлечет значение указателя команд из сегмента стека. Пример:

CALL BX

В данном случае регистр BX содержит смещение адреса процедуры относительно регистра сегмента CS.

Процедуру с атрибутом FAR можно вызвать косвенно, используя переменную размером двойное слово:

CALL DWORD PTR [BX]

CALL DWORD PTR SS:MEM_DWORD[SI]

б) команда безусловного перехода JMP

Команда JMP (jamp unconditionaly - перейти безусловно) заставляет МП извлечь новую команду не из следующей ячейки памяти, а из какой-то другой. Команда JMP имеет формат:

JMP имя,

где операнд "имя" подчиняется тем же правилам, что и операнд команды CALL: т.е. он может иметь атрибут NEAR или FAR, быть прямым или косвенным. Команда JMP занимает три байта с атрибутом NEAR и пять байтов с атрибутом FAR.

Если адрес метки находится не далее -128 или +128 байтов от команды JMP, то можно сделать команду JMP двухбайтовой, указав, что ее операнд имеет тип SHORT (short - короткий). Например:

JMP NEAR_LABEL

JMP SHORT NEAR_LABEL

в) команды условной передачи управления

Команды условной передачи управления позволяют МП "принять решение" о ходе исполнения программы в зависимости от определенных условий. Если такое условие выполнено, то МП выполнит переход; в противном случае он продолжит исполнение со следующей команды программы. Общий формат команд условной передачи управления:

Jx близкая_метка,

где x - модификатор, состоящий из одной, двух или трех букв. Запись операнда "близкая_метка" подчеркивает, что метка перехода должна находиться не далее -128 или +127 байтов от команды (в отличие от JMP, которая может передать управление в любое место).

г) команды управления циклами

Команды управления циклами обеспечивают условные передачи управления при организации циклов. У МП регистр счетчика CX служит счетчиком числа повторений циклов. Каждая команда управления циклами уменьшает содержимое регистра CX на 1, а затем использует его новое значение для "принятия решения" о выполнении или не выполнении перехода.

Основная команда этой группы LOOP (loop until Count complete - повторять цикл до конца счетчика) уменьшает содержимое регистра CX на 1 и передает управление операнду близкая_метка, если содержимое регистра CX не равно 0. В приведенном ниже примере цикл выполнится 100 раз.

```
MOV CX,100;  Загрузить число повторений в CX
START: . . .          (повторяемая группа команд)
. . .
LOOP START;  Если CX не равен 0, перейти к метке START,
. . .                ;  в противном случае выйти из цикла.
```

Команда LOOPE (loop if equal - повторять цикл, если равно), имеющая синоним LOOPZ (loop if zero - повторять цикл, если ноль), уменьшает на 1 содержимое регистра CX, а затем осуществляет переход, если содержимое регистра CX не равно нулю и флаг ZF равен 1. Повторение цикла завершается, если либо содержимое CX равно 0, либо флаг ZF равен 0, либо оба они равны 0.

Команда LOOPNE (loop if not equal - повторять цикл, если не равно), имеющая синоним LOOPNZ (loop if not zero - повторять цикл, если не ноль), уменьшает содержимое регистра CX на 1 и осуществляет переход, если содержимое регистра CX не равно 0 и флаг нуля ZF равен 0. Повторение цикла завершается, если содержимое регистра CX равно 0, либо флаг ZF равен 1, либо будет выполнено и то и другое.

2.5. Обработка прерываний.

По определению *прерывание* означает временное прекращение основного процесса вычислений для выполнения некоторых запланированных или незапланированных действий, вызываемых работой аппаратуры или программы. Эти действия могут носить сервисный характер, быть запросами со стороны программы пользователя на выполнение обслуживания со стороны операционной системы либо быть реакцией на нештатные ситуации.

Механизм прерываний поддерживается на аппаратном уровне и позволяет реализовать эффективное взаимодействие программ с операционной системой и эффективное управление программой аппаратной частью компьютера.

В зависимости от источника, прерывания классифицируются так:

- *аппаратные,* возникающие как реакция микропроцессора на физический сигнал от некоторого устройства компьютера (клавиатура, системный таймер, жесткий диск и т. д.). По времени возникновения эти прерывания *асинхронны,* то есть происходят в случайные моменты времени;
- *программные,* которые вызываются искусственно с помощью соответствующей команды из программы (команда int). Они предназначены для выполнения некоторых действий операционной системы. Эти прерывания являются *синхронными;*
- *исключения* – разновидность программных прерываний, являющихся реакцией микропроцессора на нестандартную ситуацию, возникшую *внутри* микропроцессора во время выполнения некоторой команды программы.

2.6. Команды управления микропроцессором.

Эти команды позволяют управлять работой МП из программы. Они делятся на три группы: команды управления флагами, команды внешней синхронизации и команда холостого хода NOP.

а) команды управления флагами

У МП есть семь команд, которые позволяют изменять флаг переноса CF, флаг направления DF и флаг прерывания IF.

Команды STC (set Carry flag - установить флаг переноса) и CLC (clear Carry flag - обнулить флаг переноса) переводят флаг CF в состояние 1 и 0 соответственно. Команда CMC (complement Carry flag - обратить флаг переноса) инвертирует флаг CF.

Команды STD (set direction flag - установить флаг направления) и CLD (clear direction flag - обнулить флаг направления) переводят флаг DF в состояние 1 или 0 соответственно.

Команда CLI (clear interrupt flag - установить флаг прерывания) обнуляет флаг IF, что заставляет МП игнорировать маскируемые прерывания. Команда STI (set interrupt flag - установить флаг прерывания) переводит флаг IF в состояние 1, что разрешает МП реагировать на внешние прерывания.

б) команды внешней синхронизации

Эти команды используются для синхронизации действий МП с внешними событиями.

Команда HLT (halt - остановиться) переводит МП в состояние останова, при котором он находится на холостом ходу и не выполняет никакие команды. МП выходит из состояния останова только в том случае, если его заново инициировали или если он получил внешнее прерывание.

Команда WAIT (wait - ожидать) также переводит МП на холостой ход, но при этом через каждые 5 тактов он проверяет активность входной линии TEST. В этом состоянии МП способен обрабатывать прерывания, но по завершении программы обработки прерывания он вновь возвратится в это состояние. Если линия TEST становится активной, то МП переходит к следующей за WAIT команде.

Команда ESC (escape - убежать) заставляет МП извлечь содержимое указанного в ней операнда и передать его на шину данных. Тем самым команда ESC обеспечивает другим микропроцессорам системы возможность получения своих команд из потока команд МП. Команда ESC имеет формат

ESC внешний_код,источник,

где "внешний_код" - 6-битовый непосредственный операнд, а "источник" - регистр или переменная. Команда ESC часто используется для передачи команд математическому сопроцессору. В этом случае "внешний _код" представляет собой код операции сопроцессора, а содержимое "источник" - его операнд.

Команда холостого хода NOP (no operation - нет операции) не выполняет никакой операции. Команда NOP не действует ни на флаги, ни на регистры, ни на ячейки памяти; она только увеличивает значение указателя команд IP.

3. РАЗРАБОТКА И ОТЛАДКА ПРОГРАММ

Для выполнения лабораторных работ по курсу «Цифровые устройства и микропроцессоры» необходимо самостоятельно разработать и отладить небольшие программы на ассемблере в соответствии с заданными преподавателем вариантами.

Возможны два варианта написания и отладки программ:

1) написание и отладка в интегрированной среде Turbo Pascal с помощью встроенного отладчика;

2) написание отладка с использованием пакета программ TASN, TLINK, TD.

Каждый из вариантов имеет свои преимущества и недостатки. Использование первого варианта предпочтительней в тех случаях (лабораторная работа №1), где отлаживаемые фрагменты независимы друг от друга и небольшие по объему, а величина исполнительного модуля не минимизируется. Использование второго варианта дает преимущества при условии минимизации величины исполнительного модуля, позволяет более простыми средствами управлять ресурсами программы (лабораторные работы №№2,4).

3.1. Использование Turbo Pascal с языком ассемблера.

Встроенный Ассемблер реализует большое подмножество синтаксиса, поддерживаемого Turbo Assembler (TASM) и макроассемблером Microsoft (MASM). Встроенный Ассемблер поддерживает все коды операций 8086/ 8087 и почти все операторы выражений Turbo Assembler.

За исключением DB, DW, DD (определить байт, слово и двойное слово) ни одна из директив Turbo Assembler, таких как EQU, PROC, STRUC, SEGMENT и MACRO не поддерживается встроенным Ассемблером. Однако операции, поддерживаемые директивами Turbo Assembler, во многом соответствуют конструкциям Turbo Pascal. Например, большинство директив EQU соответствует объявлениям const, var и type в Turbo Pascal, директива PROC соответствует объявлениям procedure и function, а директива STRUC соответствует типам record в Turbo Pascal. Следовательно, встроенный Ассемблер Turbo Pascal можно рассматривать как компилятор с языка Ассемблер, который использует синтаксис Паскаля для всех объявлений.

К встроенному Ассемблеру обращаются через оператор asm.

```
asm
mov ax,Left;   xchg ax,Right;   mov Left,ax;
end;
asm
mov ah,0
int 16H
mov CharCode,al
mov ScanCode,ah
end;
```

На одной строке можно поместить несколько операторов Ассемблера, разделенных ";". Разделения ";" не требуется между двумя ассемблерными операторами, если они на разных строках. Точка с запятой не указывает, что оставшаяся часть строки - комментарий, т.к. комментарий должен быть написан в стиле Паскаля, используя или (* *).

Внутри операндов следующие зарезервированные слова имеют во встроенном Ассемблере предопределенный смысл:

AH	BP	CX	DX	NEAR	SEG	ST
AL	BX	DH	ES	NOT	SHL	TBYTE
AND	BYTE	DI	FAR	OFFSET	SHR	TYPE
AX	CH	DL	HIGH	OR	SI	WORD
BH	CL	DS	LOW	PTR	SP	XOR
BL	CS	DWORD	MOD	QWORD		SS

Зарезервированные слова всегда имеют приоритет над идентификаторами пользователя. Для доступа к символу, определенному пользователем с тем же именем, необходимо использовать "&" для перекрытия оператора (&byte).

3.1.1. Использование регистров.

Правила использования регистров в операторе asm такие же, как и в external процедурах и функциях. Оператор asm должен сохранять регистры BP, SP, SS и DS и может свободно изменять регистры AX, BX, CX, DX, SI, DI, ES и Flags. На входе оператора asm BP указывает на текущий стек, SP указывает на вершину стека, SS содержит сегментный адрес сегмента стека, а DS содержит сегментный адрес сегмента данных. За исключением BP, SP, SS и DS, оператор asm не должен делать предположений о содержимом остальных регистров при входе в оператор.

3.1.2. Синтаксис ассемблерных операторов.

Синтаксис ассемблерного оператора:
[Label ":"] < Prefix > [Opcode [Operand < "," Operand >]],
где Label - идентификатор метки, Prefix - код префикса, Opcode - директива или инструкция Ассемблера и Operand - ассемблерное выражение.

3.1.3. Метки.

Метки определяются в Ассемблере так же, как в Паскале, записывая идентификатор метки с двоеточием до оператора. Как и в Паскале, метки, определенные в Ассемблере, должны объявляться в декларативной части label в блоке, содержащем оператор asm. Однако существует одно исключение из этого правила: локальные метки.

Локальные метки - это метки, которые начинаются с @. Поскольку @ не может быть частью идентификатора Паскаля, использование таких локальных меток допускается только внутри оператора asm, который определяет их (т.е. сфера действия локальной метки расширяется от ключевого слова asm до ключевого слова end для этого оператора asm). В отличие от обычных меток, локальные метки не объявляются в разделе объявления label до их использования.

Идентификатор локальной метки состоит из символа @ с последующей одной или более буквой A..Z, цифр 0..9, "_" или @. Как и для всех меток, после идентификатора идет ":".

Нормальная метка может быть определена внутри оператора asm и использована вне оператора asm, и наоборот. Одно и то же имя локальной метки может использоваться в различных операторах asm.

3.1.4. Автоматический размер перехода.

Если не было указано противное, встроенный Ассемблер оптимизирует инструкции перехода, автоматически выбирая самую короткую, и, следовательно, самую эффективную инструкцию перехода. Автоматический выбор инструкции перехода применяется к инструкции безусловного перехода JMP и ко всем инструкциям условного перехода, когда назначение - метка (а не процедура или функция).

3.1.5. Операторы выражений.

Встроенный Ассемблер предоставляет операторы, разделенные на 12 классов по приоритетам. Таблица 7 приводит операторы встроенного Ассемблера в порядке уменьшения их приоритета.

Таблица 7. Операторы выражений встроенного Ассемблера

Оператор	Комментарий
&	Оператор перекрытия идентификатора
()	Селектор элемента структуры
[]	
.	
HIGH LOW	
+ -	Унарные операторы
:	Оператор перекрытия сегмента
OFFSET SEG TYPE PTR *	
/ MOD AND OR XOR	
+ -	Бинарные операторы
NOT AND OR XOR	Побитовые операторы

Примечание: приоритет операторов встроенного Ассемблера отличается от Паскаля. Например, в ассемблерном выражении оператор AND имеет меньший приоритет, чем операторы + и -, а в Паскале наоборот.

3.1.6. Ассемблерные процедуры и функции.

Ассемблерная директива Turbo Pascal позволяет писать процедуры или функции полностью на встроенном Ассемблере, не требуя операторной части begin...end. Пример ассемблерной функции:

```
function LongMul(X, Y: Integer): Longint; assembler;
asm
mov  ax,X
imul Y
end;
```

Функции порядкового типа (Integer, Char, Boolean и перечислимые типы возвращают результаты в AL (8-битное значение), AX (16-битное значение), или DX:AX (32-битное значение). Функции типа Real возвращают результат в DX:BX:AX

3.1.7. Подготовка к отладке в интегрированной среде Turbo Pascal.

Перед выполнением компиляции отлаживаемой программы необходимо убедиться, что требуемые переключатели установлены в положение On.

1. Активное состояние опции Options/Compiler/Debugging/Debug Information (отладочная информация) позволяет установить однозначную связь между операторами исходного текста программы и кодов компилятора (номеров строк программы с адресами соответствующих объектных кодов). Только после компиляции с активной опцией Debug Information становится возможной автоматическая локализация ошибки периода исполнения, а также пошаговая отладка программы.

2. Опция Options/Compiler/Local Symbols (локальные символы) управляет включением в программу информации о локальных символах, содержащих имена и типы всех локальных переменных и констант. При наличии этой информации можно вызывать локальные переменные по их именам и изменять их значения. Также можно проверять вызовы процедур и функций (командой Debug/Call Stack).

3. В связи с тем, что карта распределения памяти при выполнении лабораторной работы в среде Turbo Pascal не требуется, в активное состояние устанавливается опция Options/Linker/Map File/ Off.

4.Опция Options/Debugger/Debugging определяет используемый отладчик. Активное состояние опции Options/Debugger/Debugging/Integrated (отладка в интегрированной среде) разрешает установку точек останова и пошаговую отладку (при этом должна быть включена опция Debug Information).

3.1.8. Отладка программ в интегрированной среде Turbo Pascal.

1. Трассировка, или пошаговое выполнение программ. Помогает понять, в какой последовательности выполняются операторы. Выполнение программы по шагам обеспечивают команды Run/Trace Into (F7) и Run/ Step Over (F8). При необходимости выполнить часть пограммы за один шаг можно воспользоваться командой Run/Goto Cursor (F4).

2. Индикация изменения значений переменных при трассировке программы. Для введения в окно Watch отображаемую переменную необходимо подвести курсор под идентификатор переменной и нажать CTRL+F7.

3. Вычисления и модификация значений переменных. Эти операции выполняются по команде Debug/Evaluate (CTRL+F4).

3.2. Создание и отладка программы с использованием TASM, TLINK, TD.

Для выполнения лабораторных работ по курсу «Цифровые устройства и микропроцессоры» необходимо самостоятельно разработать и отладить небольшие программы на ассемблере в соответствии с заданными преподавателем вариантами. На рис. 1 приведена общая схема процесса разработки программы на ассемблере.

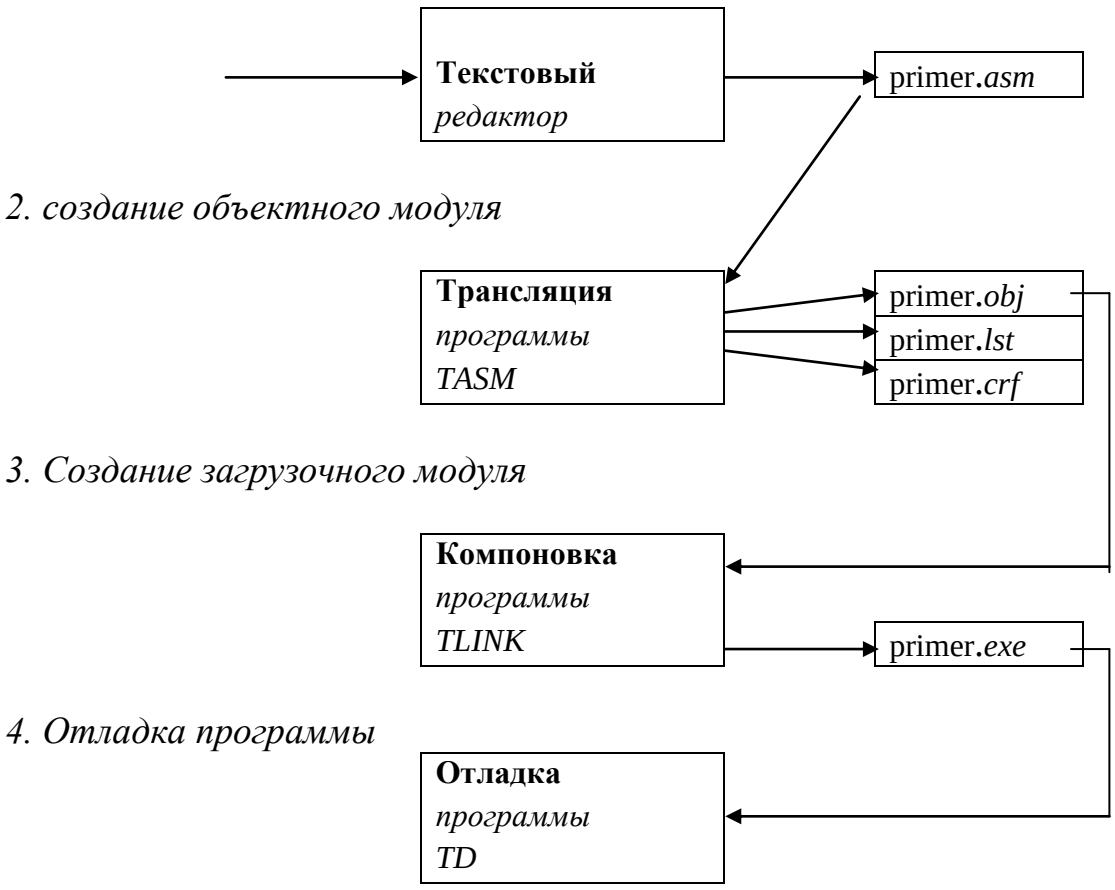


Рис. 1. Процесс разработки программы на ассемблере.

На первом этапе для ввода кода программы можно использовать любой текстовый редактор, который не вставляет в текст спецсимволов редактирования. Для экономии ресурсов ПК и времени разработчика рекомендуется запустить ПК в режиме MS-DOS и использовать встроенный редактор NC. Полученный текстовый файл должен иметь расширение .asm .

Программы, реализующие остальные этапы разработки, входят в состав программных пакетов ассемблера. Наиболее широкое распространение среди пакетов ассемблеров для процессоров Intel получили

два пакета: «макроассемблер» MASM фирмы Microsoft, и Turbo Assembler фирмы Borland. Пакет MASM получил свое название потому, что позволял программисту задавать *макроопределения* (макросы), представляющие собой именованные группы команд.Эти группы можно вставлять в любом месте программного кода, указав лишь имя группы. Пакет Turbo Assembler фирмы Borland имеет два режима работы. Один из этих режимов, называемый MASM, поддерживает все основные возможности макроассемблера MASM. Другой режим, называемый Ideal предоставляет более удобный синтаксис написания программ, более эффективное использование памяти при трансляции программ и некоторые другие преимущества.

Так как программы пакета Turbo Assembler фирмы Borland входят в состав таких популярных продуктов как Turbo C++ и Turbo Pascal, уже используемых в учебном процессе, дальнейшее рассмотрение этапов разработки программ проиллюстрировано использованием именно этого пакета

Для получения *объектного модуля* исходный файл необходимо подвергнуть трансляции при помощи программы TASM.EXE. Формат командной строки следующий:

tasm [опции] имя_исходного_файла [,имя_объектного_файла] [,имя_файла_листинга] [,имя_файла_перекрестных_ссылок]

Необязательный аргумент [опции] позволяет задать режим работы транслятора TASM. Для получения представления об этих опциях следует воспользоваться литературой [1-6].

Для устранения ошибок, обнаруженных при трансляции, необходимо определить место их возникновения и проанализировать ситуацию. Место ошибки легко определить по значению в скобках в сообщении об ошибке. Это значение является номером ошибочной строки. Однако не всегда номера строк, указанных как ошибочные при трансляции, совпадают с номерами строк в исходном тексте (например, при использовании описаний макрокоманд). В таких случаях рекомендуется запускать трансляцию исходного файла с опциями, создаваемыми *файл листинга.* Файл листинга содержит много полезной для отладки информации, в частности, код ассемблера исходной программы, номера строк кодов программы, размещение кодов в памяти.

После устранения ошибок при создании объектного модуля можно приступать к следующему этапу- *созданию исполняемого* (загрузочного) модуля с помощью компоновщика TLINK.Полный формат командной строки для запуска компановщика достаточно сложен, и для выполнения лабораторных работ вполне достаточно упрощенного формата:

TLINK [опции] список_объектных_файлов [,имя_загрузочного_модуля] [,имя_файла_карты] [,имя_файла_библиотеки]

Также, как и в предыдущем примере командной строки для TASM, параметры, указанные в квадратных скобках, необязательны.

После получения исполняемого модуля можно приступить к тестированию и отладке программы. При *использовании отладчика* Turbo Debugger (TD) программисту предоставляются следующие возможности:

- выполнение трассировки программы в прямом направлении;
- выполнение трассировки программы в обратном направлении;
- просмотр и изменение состояния аппаратных ресурсов микропроцессора во время покомандного выполнения программы.

Необходимо отметить, что TD не позволяет вносить изменения в текст исходной программы, хотя на этапе отладки можно изменять машинный код и сразу наблюдать соответствующие изменения. После завершения работы отладчика изменения кода не будут сохранены.

Обобщая вышеизложенные сведения, для выполнения лабораторных работ рекомендуется следующая последовательность действий и выбор инструментальных средств:

1. С помощью встроенного редактора NC получить исходный текст кодов программы с расширением .asm. В исходном тексте программы обязательно должна быть определена метка для первой команды, с которой начинается выполнение программы. Имя этой метки обязательно нужно указывать в конце программы в качестве операнда директивы END:

```
. . . . . .
main
. . .

END main
```

2. Для сохранения возможности отладки на уровне исходного текста исходный модуль должен быть оттранслирован с опцией /zi:

```
TASM  /zi  primer_1.asm
```

Применение опции /zi разрешает транслятору сохранять связь символических имен в программе и их смещений в сегменте кода, что позволяет при отладке использовать оригинальные имена.

3. Компановка модуля должна осуществляться с опцией /v:

```
TLINK /v  primer_1.obj
```

Опция /v указывает на необходимость сохранения отладочной информации в исполняемом файле.

4. Запуск отладчика производится из командной строки с указанием исполняемого модуля программы:

```
TD  primer_1.exe
```

Управление работой отладчика ведется с помощью двух типов меню:

- глобального меню – находится в верхней части экрана и доступно постоянно. Вызов пунктов меню производится нажатием клавиши F10 или указателем «мыши»;
- локального меню – для каждого окна отладчика можно вызвать его собственное меню, которое отражает особенности данного окна. Вызвать данное меню можно активизировав окно и щелкнув правой кнопкой «мыши» или нажав Alt-F10.

Для детального изучения работы модуля и использования ресурсов процессора следует выбрать пошаговый режим выполнения программы.
Для активизации этого режима нужно нажать клавиши F7 (Run/Trace into) или F8 (Run/Step over). Отличие в режимах работы проявляется при использовании переходов в процедуру или прерываниях.

Кроме окна Module, в некоторых случаях открывающегося по умолчанию, полезно использовать окно CPU. Последнее состоит из пяти подокон:

- окна с исходной программой в дизассемблированном виде;
- окна регистров микропроцессора Registers, отражающего текущее содержимое регистров процессора. По умолчанию здесь установлена программная модель i8086. Для переключения на регистровую модель i486 или Pentium выберите из локального меню команду Registers 32-bit -Yes;
- окна флагов, которые отражают текущее состояние флагов микропроцессора в соответствии с их мнемоническими названиями;
- окна стека Stack, отражающего содержимое памяти, выделенной для стека.Адрес области стека определяется автоматически по содержимому регистров SS и SP;
- окна с дампом памяти Dump, отражающего содержимое области памяти по адресу, который формируется из компонентов, указанных в левой части окна. В этом окне можно увидеть содержимое произвольной области памяти, для определения адресов этой области нужно использовать пункты локального меню.

Прервать выполнение программы можно нажатием клавиш Ctrl+F2.

4. ПРИНЦИП РАБОТЫ ПЕРИФЕРИЙНЫХ УСТРОЙСТВ ПК

4.1. Архитектура программируемого таймера КР580ВИ53

БИС программируемого таймера КР580ВИ53 предназначена для организации работы микропроцессорных систем в режиме реального времени и позволяет формировать сигналы с различными временными и частотными характеристиками.

Программируемый таймер (ПТ) имеет три независимых канала, каждый из которых содержит 16-разрядный вычитающий счетчик. Счетчики могут работать в двоичном или двоично-десятичном коде, с однобайтными или двухбайтными числами. Скорость счета программно изменяется от 0 До 2 МГц.

Упрощенная структурная схема ПТ приведена на рис. 3.1. В состав БИС входят: буфер данных (*BD*), предназначенный для обмена данными и управляющими словами между МП и ПТ; схема управления чтением-записью (*RWCU*), обеспечивающая выполнение операций ввода-вывода информации в ПТ; регистр управляющего слова (*RGR*), предназначенный для записи управляющих слов, задающих режимы работы счетчиков; счетчик каналов (*CTO – CT2*).

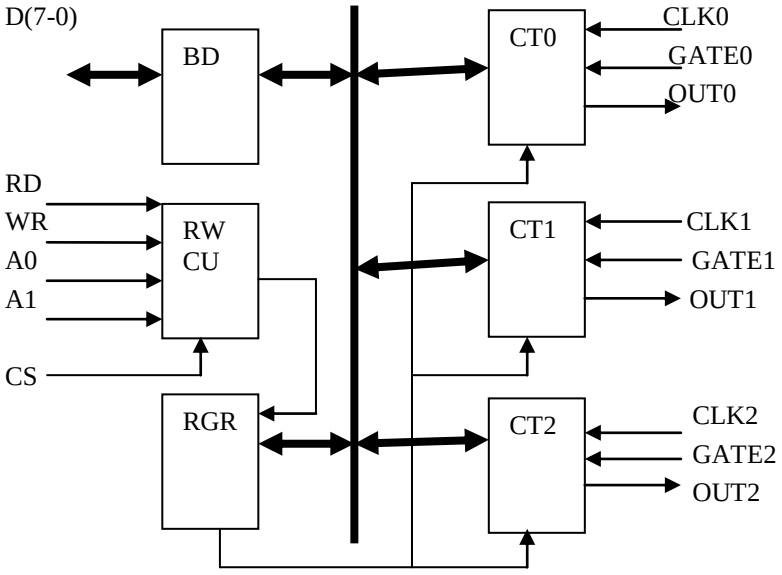


Рис 3.1. Структурная схема ПТ КР580ВИ80

Назначение входных, выходных и управляющих сигналов ПТ указано при описании выводов микросхемы в табл. 3.1.

Подключение ПТ к шинам микропроцессора показано на рис. 3.2. Операции обмена информацией между ПТ и МП, задаваемые сигналами управления и адресными входами, приведены в табл. 3.2. Установка

режима работы каждого канала ПТ производится программно путем записи управляющего слова и начального значения содержимого

Таблица 3.1. Описание выводов ПТ.

Обозначение вывода	**Номер контакта**	**Назначение вывода**
D(7-0)	1; 2; 3; 4,5; 6; 7; 8	Канал данных
RD	22	Сигнал «чтение»
WR	23	Сигнал «запись»
Обозначение вывода	**Номер контакта**	**Назначение вывода**
AO , AI	19; 20	Адресные входы, выбирающие один из каналов ПТ или управляющий регистр
CS	21	Выбор микросхемы
CLK0 - CLK2	9; 15; 18	Входы синхронизации счетчиков
CATE0 - CATE 2	11; 14; 16	Входы управления счетчиков
OUT0 - OUT2	10; 13; 17	Выходные сигналы счетчиков
U cc	24	Напряжение питания (+5В)
GND	12	Напряжение питания (0 В)

счетчика (*N*) с помощью команд вывода *(OUT*). Формат управляющего слова и назначения отдельных разрядов представлены на рис. 3.3. Управляющее слово задает номер счетчика (разряды *D7*, *D6*), последовательность записи и считывания содержимого счетчика (разряды *D5*, *D4*), режим работы (разряды *D3 – D1*) и вид используемого кода (разряд *D0*).

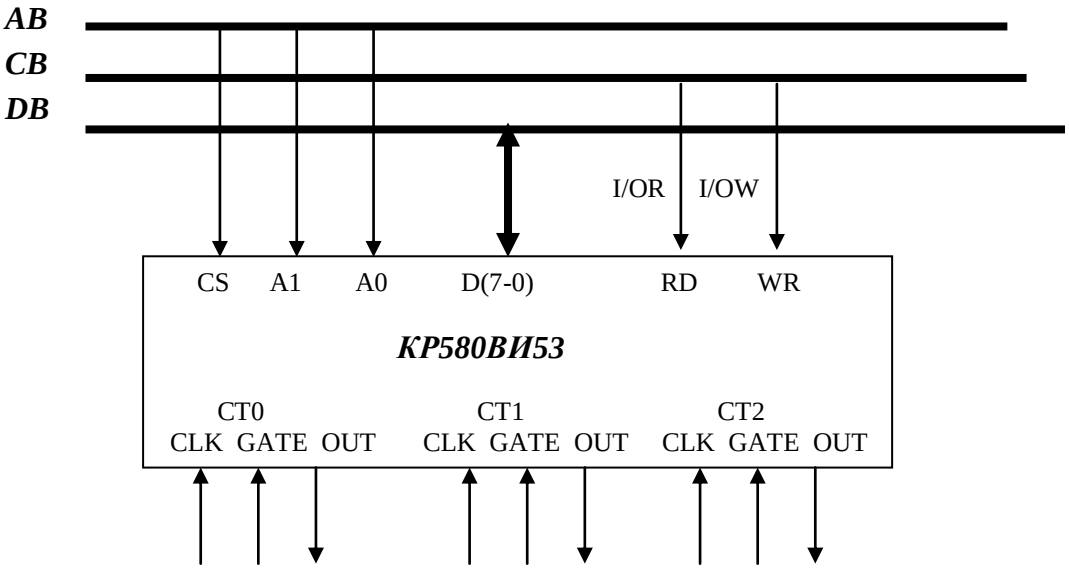


Рис 3.2. Подключение ПТ к шинам микропроцессора

В процессе работы ПТ содержимое любого из счетчиков можно прочитать двумя способами:

1) приостановив работу счетчика подачей соответствующего сигнала *GATE L*-уровня или блокировкой тактовых импульсов; прочитав

содержимое счетчика, начиная с младшего байта, с помощью двух команд ввода (*1N*), если запрограммировано чтение двух байтов;

2) записав в ПТ управляющее слово, содержащее нули в разрядах *D4, D5* (рис. 3.3); нули в этих разрядах указывают на выполнение операции «защелкивания» счета в момент чтения; прочитав содержимое счетчика с помощью команд ввода.

Таблица 3.2.Операции обмена информацией между ПТ и микропроцессором.

Операция	**Сигналы управления**				
Запись управляющего слова в регистр управляющего слова из МП	0	1	0	1	1
Загрузка *CT0* с *D (7-0)*	0	1	0	0	0
Загрузка *CT1* с *D (7-0)*	0	1	0	0	1
Загрузка *CT2* с *D (7-0)*	0	1	0	1	0
Чтение *CT0* на *D (7-0)*	1	0	0	0	0
Чтение *CT1* на *D (7-0)*	1	0	0	0	1
Чтение *CT2* на *D (7-0)*	1	0	0	1	0
Отключение ПТ от *D (7-0)*	1	1	0	X	X
Отключение ПТ от *D (7-0)*	1	0	0	1	1
Отключение ПТ от *D (7-0)*	X	X	1	X	X
Примечание: X – безразличное состояние сигнала.					

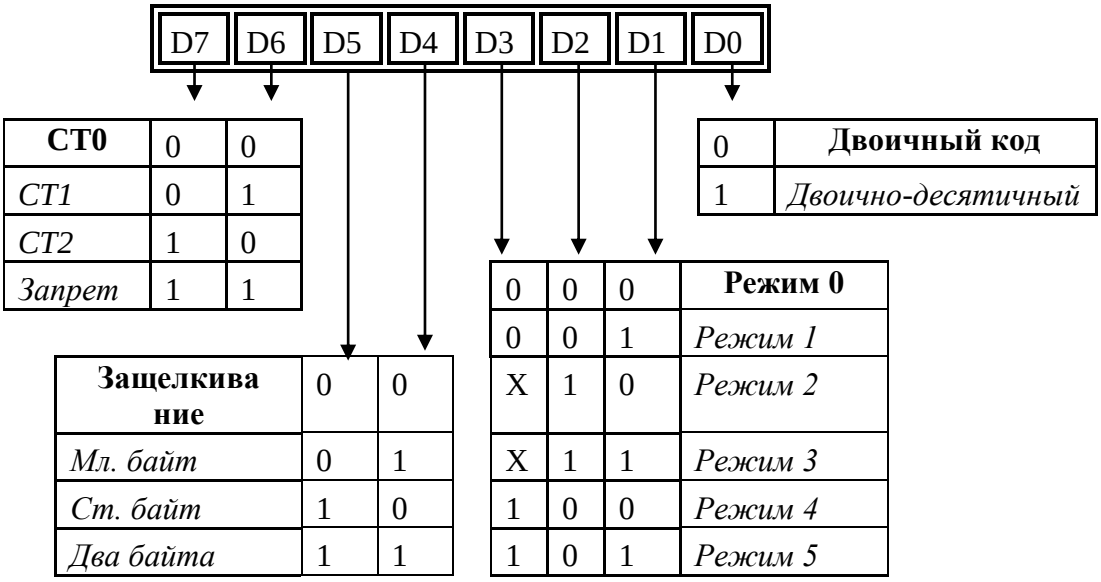


Рис.3.3. Формат управляющего слова.

Каждый из счетчиков ПТ может работать в одном из шести режимов: в режиме *0* – программируемая задержка; в режиме *1* – программируемый ждущий мультивибратор; в режиме *2* – программируемый генератор тактовых сигналов; в режиме *3* – генератор прямоугольных сигналов; в режиме *4* – программно управляемый строб; в режиме *5* – аппаратно-управляемый строб. Воздействие сигнала *GATE* на соответствующий

счетчик зависит от режима работы. Функции, выполняемые сигналом *GATE* для различных режимов, приведены в табл. 3.3.

Таблица 3.3. Функции сигнала GATE.

Режим	**Низкий уровень или отрицательный фронт**	**Положительный фронт**	**Высокий уровень сигнала**
0	Запрещает счет	-	Разрешает счет
1	-	Начинает счет; устанавливает *L*-уровень сигнала *OUT* со следующего такта *CLK*	-
2	Запрещает счет; устанавливает Н-уровень сигнала	Начинает счет	Разрешает счет
3	То же	Начинает счет	Разрешает счет
4	Запрещает счет	-	Разрешает счет
5	-	Начинает счет	-

В режиме 0 (рис. 3.4,а) после занесения управляющего слова на выходе *OUT* устанавливается *L*- уровень. Уменьшение содержимого счетчика начинается при *H*-уровне сигнала *GATE*. После окончания счета на выходе *OUT* устанавливается напряжение *H*-уровня. Загрузка в счетчик нового значения младшего байта в процессе счета останавливает счет, а загрузка нового значения старшего байта начинает новый цикл счета.

В режиме 1 (рис. 3.4,б) при *H*-уровне сигнала *GATE* на выходе *OUT* формируется отрицательный импульс длительностью *N* периодов тактовых импульсов *CLK*. Загрузка в процессе счета нового значения *N* не изменяет текущего режима счета. Импульс новой длительности формируется при следующем нарастании фронта сигнала *GATE*.

В режиме 2 (рис. 3.4,в) ПТ генерирует периодический сигнал с частотой, в *N* раз меньшей частоты тактовых импульсов *CLK*. Выходной сигнал *L*-уровня устанавливается на последнем такте периода. Загрузка счетчика новым значением *L* в процессе счета приводит к изменению величины следующего периода. Сигнал *GATE* можно использовать для внешней синхронизации ПТ, так как *L*-уровень сигнала *GATE* запрещает счет, устанавливая *H*-уровень сигнала *OUT*, а *H*-уровень сигнала *GATE* начинает счет сначала.

Режим 3 (рис. 3.5,а) отличается от режима 2 тем, что при четном значении *N* на выходе счетчика генерируется сигнал *H*-уровня в течение первой половины периода и сигнал *L*-уровня в течение другой половины. При нечетном *N* длительность сигнала *H*-уровня на один такт больше, чем для сигнала *L*-уровня. (В режиме 3 число $N = 3$ нельзя загружать в счетчик).

В режиме 4 (рис. 3.5,б) генерируется выходной сигнал *H*-уровня длительностью *N* периодов тактового сигнала *CLK*. После завершения счета устанавливается выходной сигнал L-уровня, на один период сигнала *CLK*.

Перезагрузка младшего байта в процессе счета не влияет на текущий счет, а загрузка старшего байта начинает новый цикл счета.

Режим 5 (рис. 3.5,в) аналогичен режиму *4*. Запуск счетчика производится положительным фронтом сигнала *GATE*. Загрузка счетчика новым значением числа *N* в процессе счета не влияет на длительность текущего цикла, но следующий цикл счета будет соответствовать новому значению *N*.

Управляющие слова могут быть записаны в ПТ в произвольном порядке. В любой последующий момент времени записываются начальные коды счетчиков в соответствии со значениями разрядов *D5*, *D4* управляющих слов;

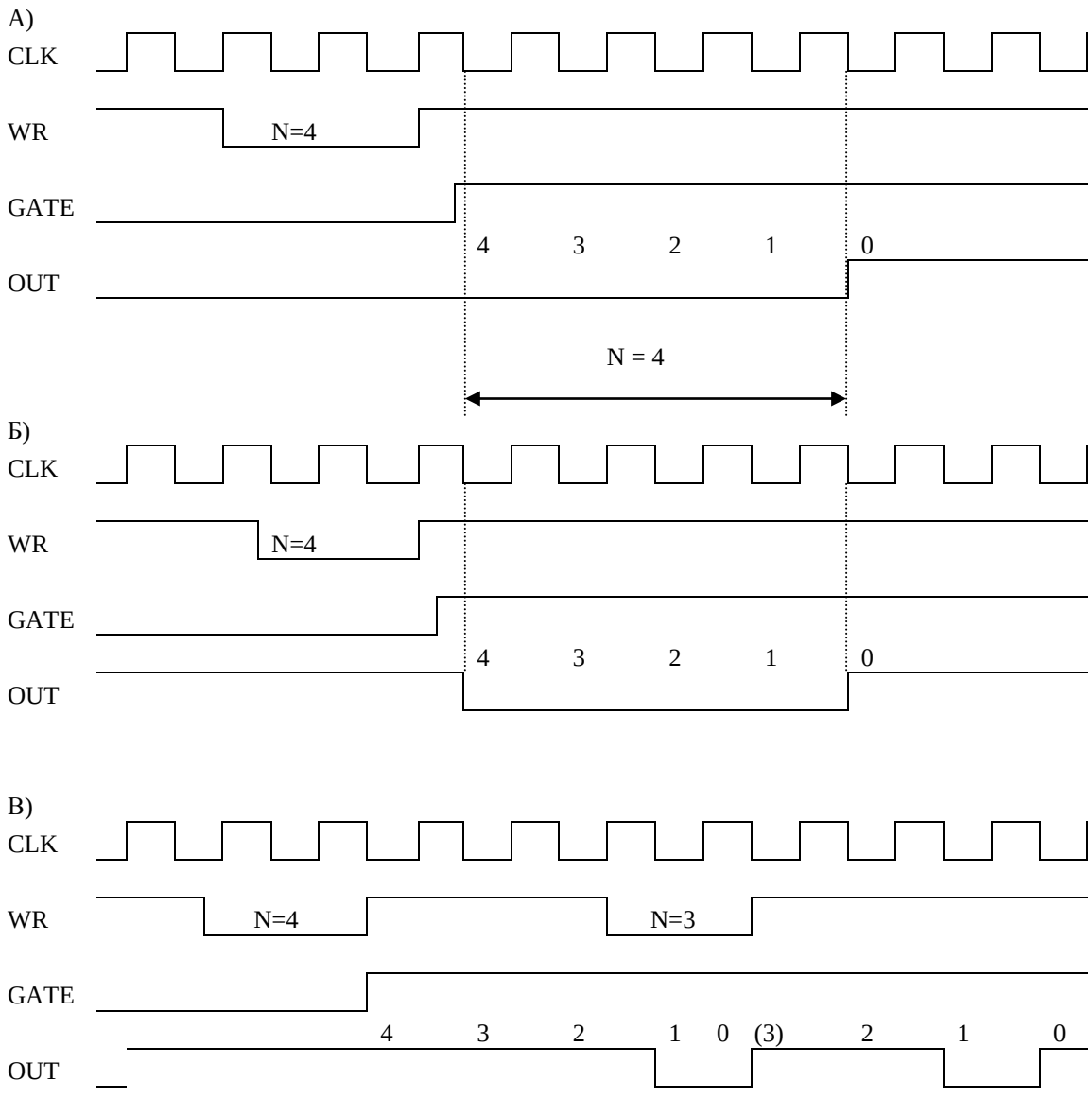


Рис.3.4. Диаграммы режимов 0-2 работы ПТ

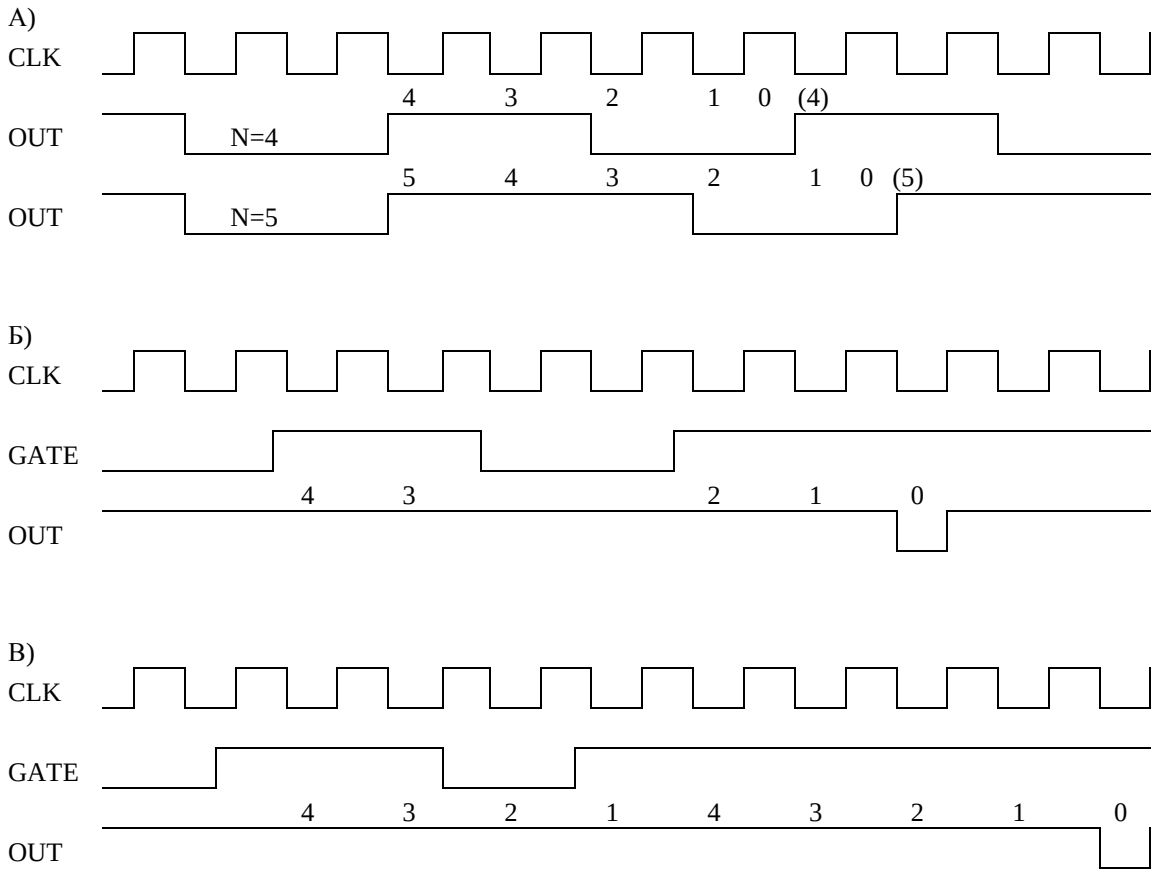


Рис.3.5. Диаграммы режимов 3-5 работы ПТ

Основные электрические параметры микросхемы КР580ВИ53 при температуре окружающей среды (+ 25 ± 10) °С приведены ниже:
Выходное напряжение логического нуля *U OL* , В . . . . . < 0,4
Выходное напряжение логической единицы *U OH* , В. . . . . > 2,4
Ток потребления *Icc*, мА . . . . . < 115
Ток утечки на входах *I IL*, мкА– . . . . 1,...,1
Ток утечки на выходах *I OL*, мкА – . . . . 1,5,...,1,5

4.2. Архитектура БИС параллельного интерфейса КР580ВВ55

БИС программируемого параллельного интерфейса КР580ВВ55 предназначена для организации ввода/вывода параллельной информации различного формата и позволяет реализовать большинство известных протоколов обмена по параллельным каналам. БИС программируемого параллельного интерфейса (ППИ) может использоваться для сопряжения микропроцессора со стандартным периферийным оборудованием (дисплеем, телетайпом, накопителем).

Структурная схема ППИ приведена на рис. 4.1. В состав БИС входят: двунаправленный 8 -разрядный буфер данных (*BD*), связывающий ППИ с

системной шиной данных; блок управления записью/чтением (*RWCU*), обеспечивающий управление внешними и внутренними передачами данных, управляющих слов и информации о состоянии ППИ; три 8-разрядных канала ввода/вывода (*PORT A, B* и *C*) для обмена информацией с внешними устройствами; схема управления группой *A* (*CUA*), вырабатывающая сигналы управления каналом *A*, и старшими разрядами канала *C [PC(7 – 4)]*; схема управления группой *B (CUB*), вырабатывающая сигналы управления каналом *B* и младшими разрядами канала *C[PC(3-0)]*.

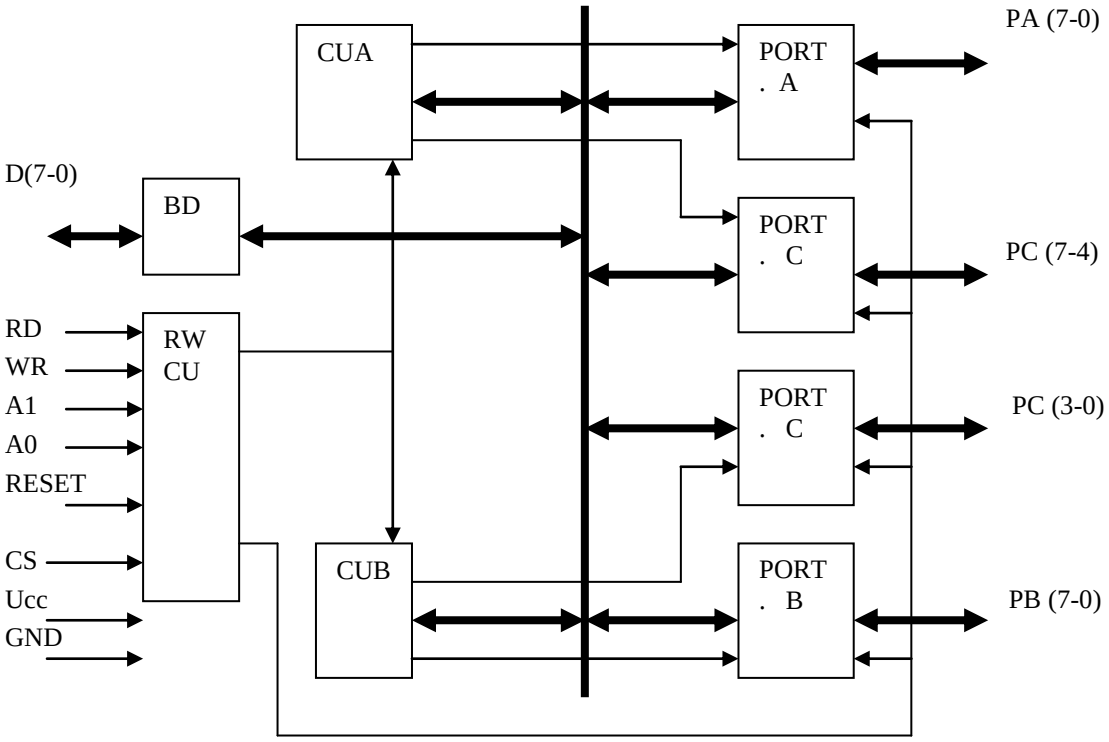


Рис.4.1. Структурная схема ППИ КР580ВВ55

Назначения входных, выходных и управляющих сигналов ППИ приведены при описании выводов микросхемы в табл. 4.1. Сопряжение БИС КР580ВВ55 со стандартной системной шиной показано на рис. 4.2. Сигналы управления работой ППИ подаются на блок *RWCU* (рис. 4.1) и вместе с адресными входами *A0, A1* задают вид операции, выполняемой БИС (табл. 4.2).

Режим работы каждого из каналов ППИ программируется с помощью управляющего слова. Управляющее слово может задать один из трех режимов: основной режим ввода/вывода (режим *0*), стробируемый ввод/вывод (режим *1*), режим двунаправленной передачи информации (режим *2*). Одним управляющим словом можно установить

различные режимы работы для каждого из каналов. Формат управляющего слова представлен на рис. 4.3.

Таблица 4.1. Описание выводов ППИ.

Обозначение вывода	Номер контакта	Назначение вывода
D(7-0)	27; 28; 29; 30; 31; 32; 33; 34	Вход/выход данных
RD	5	Чтение; *L*-уровень сигнала разрешает считывание информации из регистра, адресуемого по входам *A0, A1* на шину *D(7-0)*
WR	36	Запись; *L*-уровень сигнала разрешает запись информации с шины *D(7-0)* в регистр ППИ, адресуемый по входам *A0, A1*
AO , AI	9; 8	Входы, адресации внутренних регистров ППИ.
RESET	35	Сброс; *H*-уровень сигнала обнуляет регистр управляющего слова и устанавливает все порты в режим ввода.
CS	6	Выбор микросхемы; *L*-уровень сигнала подключает ППИ к системной шине.
PA(7-0)	37; 38; 39; 40; 1; 2; 3; 40; 1; 2; 3; 4	Вход/выход канала *A*
PB(7-0)	15; 24; 23; 22; 21; 20; 19; 18	Вход/выход канала *B*
PC(7-0)	10; 11; 12; 13; 17; 16; 15; 14	Вход/выход канала *C*
U cc	26	Напряжение питания (+5В)
GND	7	Напряжение питания (0 В)

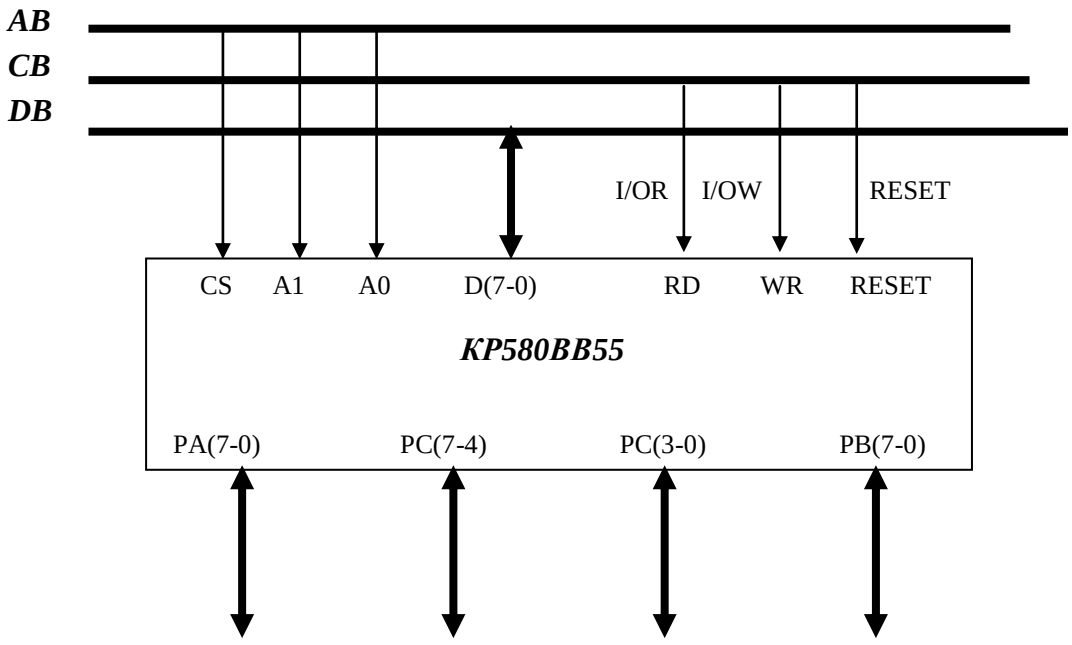


Рис.4.2. . Подключение ППИ к шинам микропроцессора

Таблица 4.2. Операции, задаваемые управляющими сигналами ППИ.

Операция	**Сигналы управления**				
Запись управляющего слова в регистр управляющего слова из МП	0	1	0	1	1
Запись в канал *A*	0	1	0	0	0
Запись в канал *B*	0	1	0	0	1
Запись в канал *C*	0	1	0	1	0
Чтение из канала *A*	0	0	1	0	0
Чтение из канала *B*					
Чтение из канала *C*	0	0	1	0	0
Отключение ППИ от *D (7-0)*	0	0	1	1	0
	1	X	X	X	X
Примечание: X - безразличное состояние сигнала.					

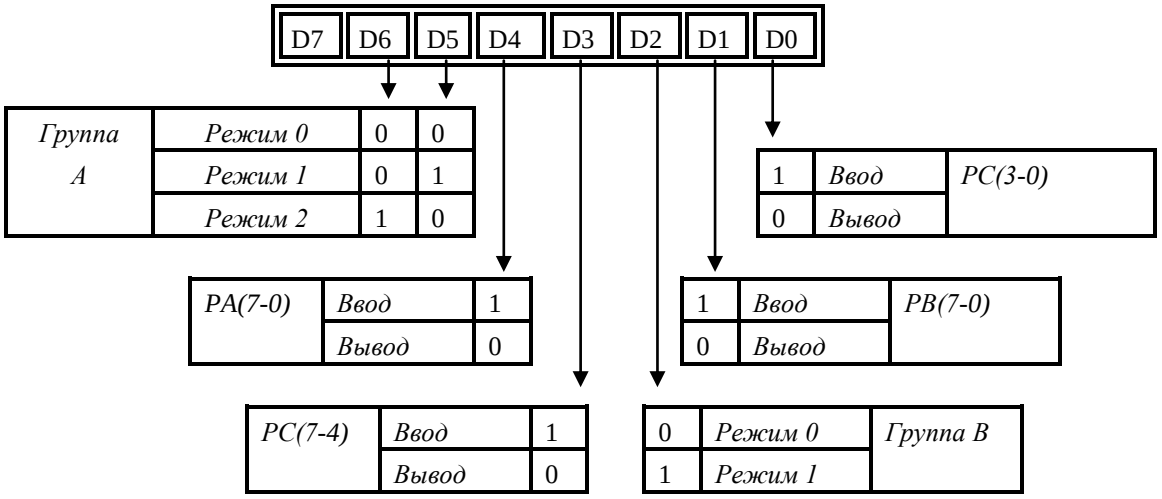


Рис.4.3. Формат управляющего слова ППИ.

Канал *A* может работать в любом из трех режимов, канал *B* – в режимах *0* и *1*. Канал *C* может быть использован для передачи данных только в режиме *0*, а в остальных режимах он служит для передачи управляющих сигналов, сопровождающих процесс обмена по каналам *A* и *B*.

Разряд *D7* управляющего слова (рис. 4.3) определяет либо установку режимов работы каналов (*D7 = 1*), либо работу ППИ в режиме сброса/установки отдельных разрядов канала С(*D7=0*). При поразрядном управлении каналом *C* разряды *D3-D1* определяют номер модифицируемого разряда; разряд *D0* задает сброс(*D0=0*) или установку (*D0=1*) модифицируемого разряда; разряды *D6-D4* не используются.

Сброс/установку разрядов канала С можно использовать для выработки сигналов запроса прерывания от ППИ. Для каждого пз каналов *A* и *B* в ППИ имеется триггер разрешения прерывания, установка/сброс

которого осуществляется управляющим словом установки/сброса определенного разряда канала *C* . Если триггер разрешения прерывания соответствующего канала установлен (INTE=1), то ППИ может сформировать сигнал запроса прерывания при готовности внешнего устройства к вводу или выводу.

Режим 0 применяется при синхронном обмене или при программной организации асинхронного обмена. Микросхема может рассматриваться в этом режиме как устройство, состоящее из четырех портов (два 8-разрядных и два 4-разрядных), независимо настраиваемых на ввод или вывод. Вывод информации осуществляется по команде *OUT* микропроцессора с фиксацией выводимой информации в регистрах каналов, а ввод - по команде *IN* без запоминания информации.

Режим 1 обеспечивает стробируемый однонаправленный обмен информацией с внешним устройством. Передача данных производится по каналам *A* и *B*, а линии канала С управляют передачей. Работу канала в режиме *1* сопровождают три управляющих сигнала. Если один из каналов запрограммировать на режим *1*, то остальные 13 интерфейсных линий можно использовать в режиме *0*. Если оба канала запрограммированы на режим *1*, то оставшиеся две интерфейсные линии канала *C* могут быть настроены на ввод или вывод.

В режиме 1 для ввода информации используются следующие управляющие сигналы: строб приема (*STB)* - входной сигнал, формируемый внешним устройством, указывает на готовность ВУ к вводу информации; подтверждение приема (*IBF*) - выходной сигнал ППИ, сообщающий ВУ об окончании приема в канал, формируется по спаду *STB*; запрос прерывания *(INTR*) - выходной сигнал ППИ,
информирующий МП о завершении приема информации в канале, *H*-уровень сигнала устанавливается при *STB*=1, *IBF*=1 и INTE=1, сбрасывается спадом сигнала *RD.*

Для операции ввода управление сигналом *INTE* канала *A* осуществляется по линии *PC4* , а канала *B* - по линии *PC2*.

Для вывода информации в режиме *1* используются следующие управляющие сигналы: строб записи (*OBF)* – выходной сигнал, указывающий внешнему устройству о готовности к выводу, формируется по фронту *WR*; подтверждение записи (*АСК*) – входной сигнал от внешнего устройства, подтверждающий прием информации из ППИ; запрос прерывания (*INTR*) – выходной сигнал ППИ, информирующий МП о завершении операции вывода информации, *H*-уровень сигнала устанавливается по фронту сигнала АСК при *OBF*=1 и *INTR*=1, сбрасывается спадом сигнала *WR.*

Для операции вывода управление сигналом *INTE* канала *A* осуществляется по линии *PC6*, а канала *B* – по линии *PC2*.

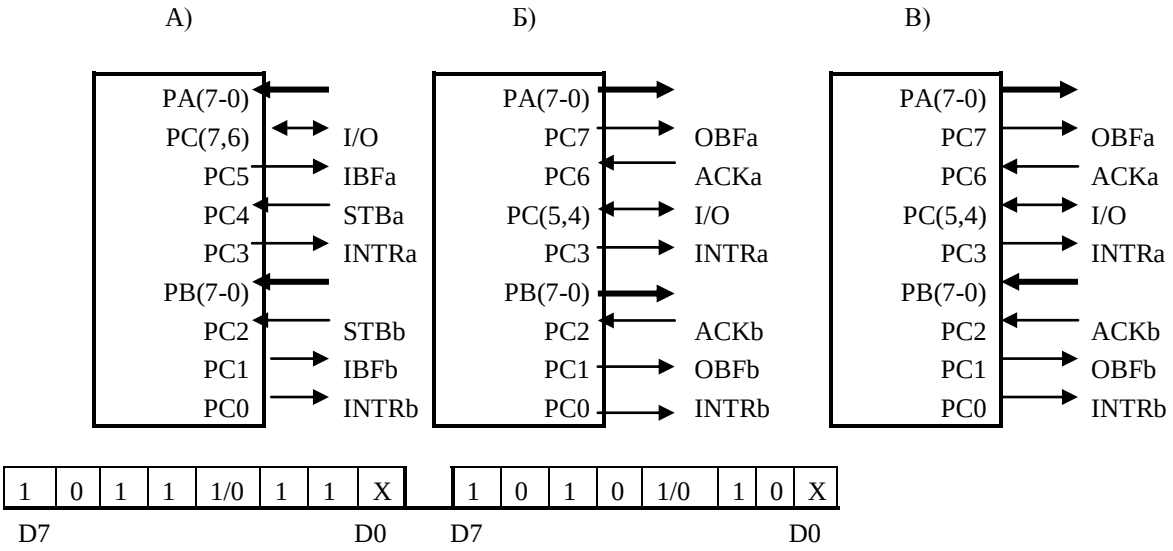


Рис.4.4. Примеры конфигурации ППИ

На рис. 4.4,а приведен пример конфигурации ППИ в режиме 1 и соответствующее ему управляющее слово для ввода по каналам *A, B*, а на рис. 4.4,б – для вывода. Не используемые для передачи управляющих сигналов линии *PC7, PC6* (рис. 4.4,а) и *PC5, PC4* (рис. 4.4,б) могут быть запрограммированы на ввод *(D3*=1) или вывод (*D3*=0).

На рис. 4.4,в приведен вариант конфигурации ППИ в режиме *1* для ввода информации по каналу *A* и ввода по каналу *B*. Управляющее слово этого варианта имеет вид 1010D311X, где *D3* определяет работу линий *PC5, PC4* на ввод или вывод.

Временные диаграммы работы ППИ в режиме *1* при вводе и выводе информации представлены соответственно на рис. 4.5 и 4.6.

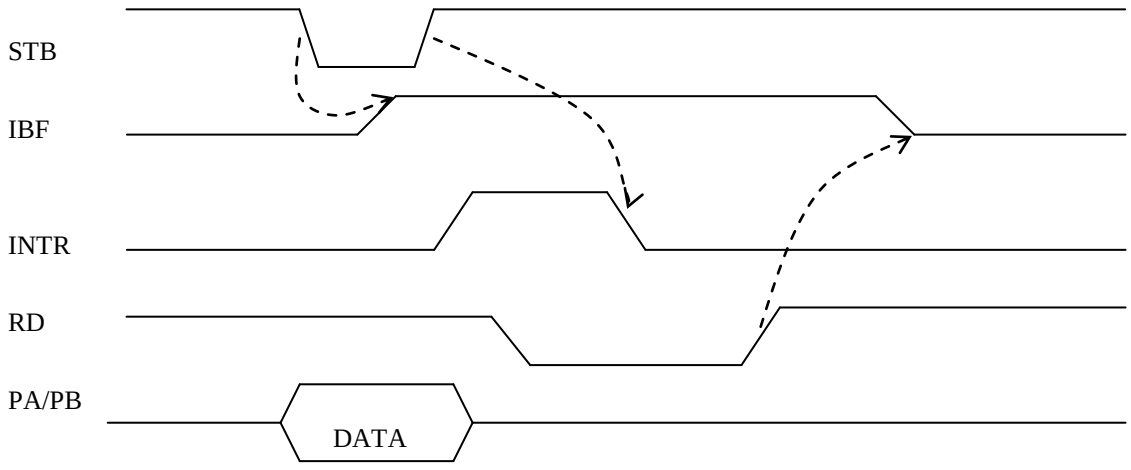


Рис.4.5. Временные диаграммы работы ППИ в режиме 1 при вводе информации.

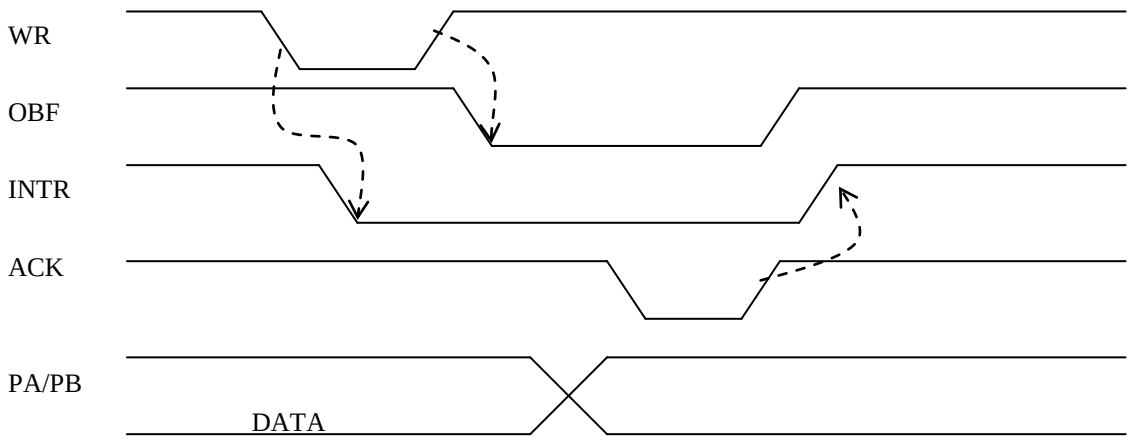


Рис.4.6. Временные диаграммы работы ППИ в режиме 1 при выводе информации.

Режим 2 обеспечивает двунаправленную передачу информации по каналу *A* к внешнему устройству и обратно. Процесс обмена сопровождают пять управляющих сигналов, подаваемых по линиям *PC7 – PC3*. Оставшиеся 11 интерфейсных линий могут настраиваться на режим *0* или режим *1*. Распределение сигналов по интерфейсным линиям и управляющее слово режима *2* приведены на рис. 4.7,а. Разряд *D0* в этой конфигурации ППИ определяет настройку на ввод или вывод интерфейсных линий *PC2, PC1* и *PC0*. Функции управляющих сигналов аналогичны рассмотренным выше сигналам для режима *1*. Управление установкой внутреннего сигнала *INTE* для операции ввода осуществляется по линии *PC4*, а для операции вывода – по линии *PC6*. Временная диаграмма работы ППИ в режиме 2 представлена на рис. 4.8.

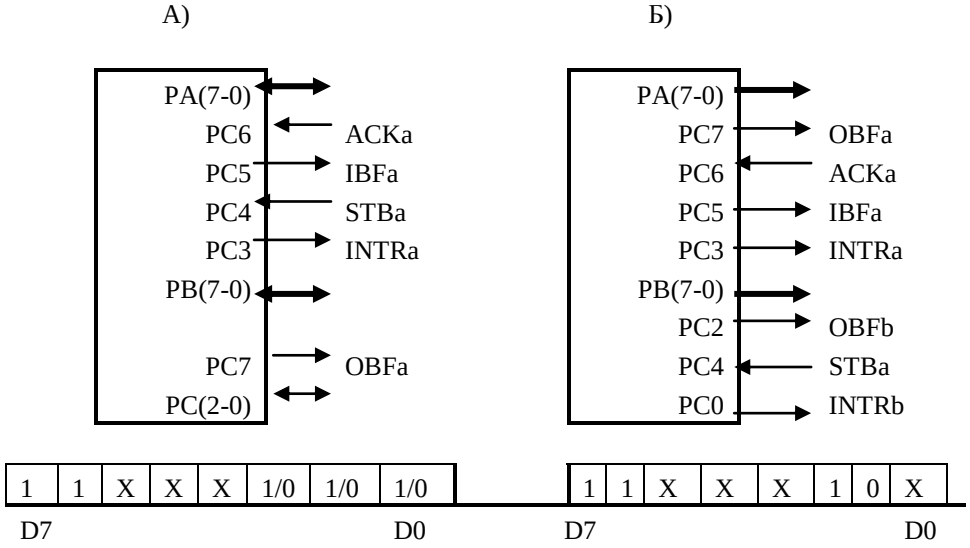


Рис.4.7. Распределение сигналов по интерфейсным линиям при различных режимах ППИ.

На рис. 4.7,б показан один из возможных вариантов комбинированного режима работы ППИ, в котором канал *A* запрограммирован на режим *2* , а канал *B* - на вывод в режиме *1*.

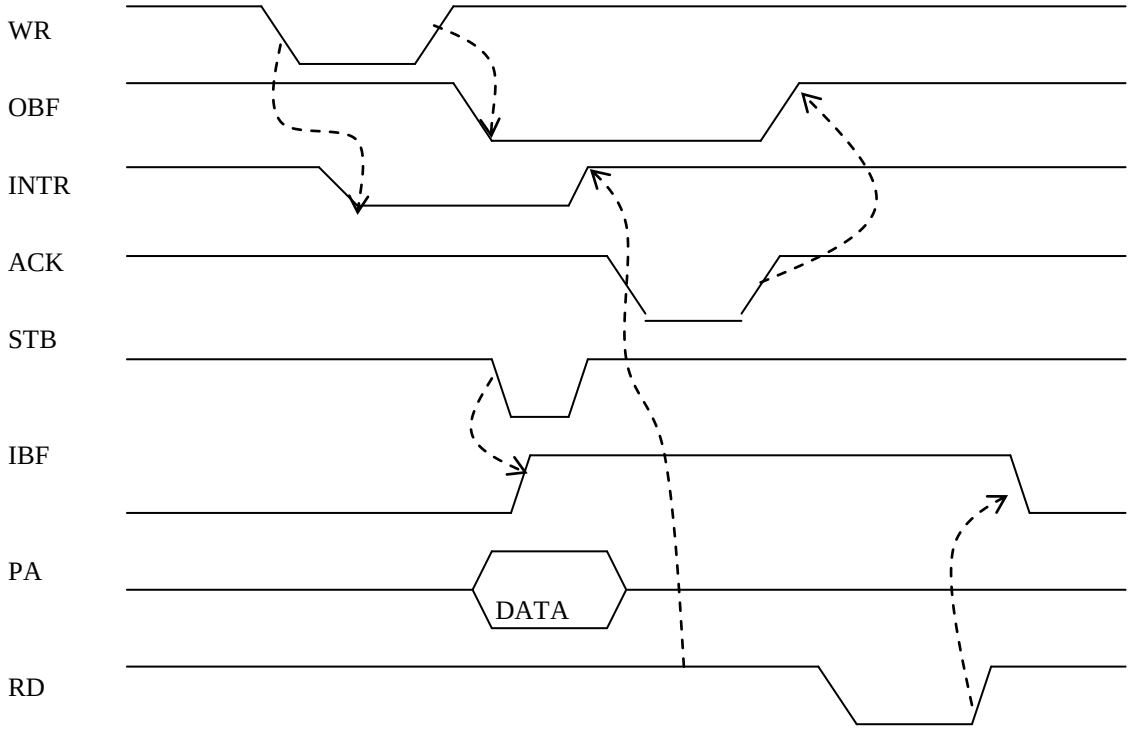
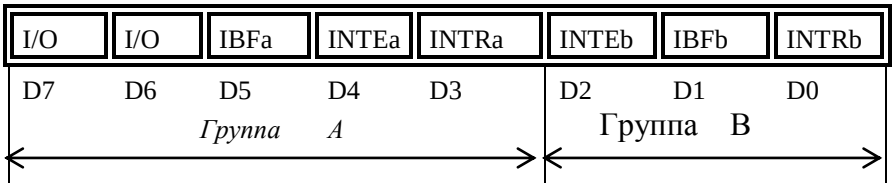
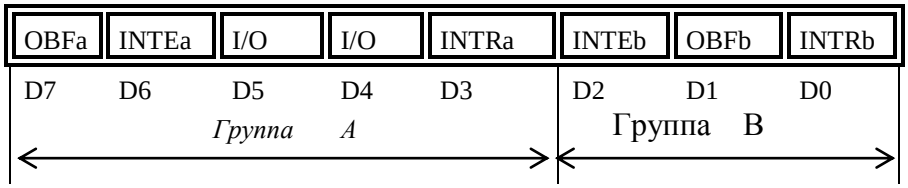


Рис.4.8. Диаграмма работы ППИ в режиме 2.

Режим 1. Ввод



Режим 1. Вывод



Режим 2.

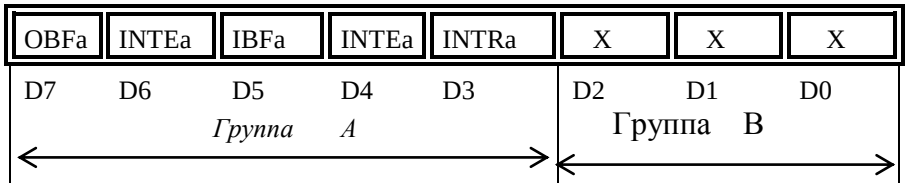


Рис.4.9. Форматы слова состояния ППИ в различных режимах.

В режимах *1* и *2* возможно контролировать состояние работы внешнего устройства и ППИ. Контроль осуществляется чтением слова-состояния канала *C* по команде *IN*. Форматы слова-состояния показаны на рис. 4.9. Для режима *1* сигналы *I/O* в разрядах с определенными номерами указывают на операцию ввода или вывода по интерфейсным линиям канала *C* с такими же номерами. Для режима *2* значения разрядов *D2-D0* определяются только режимом работы группы *B*.

Основные электрические параметры микросхемы КР580ВВ55 следующие:

Выходное напряжение логического нуля U_{ol}, В . . . . . .< 0,4
Выходное напряжение логической единицы U_{oh}, В . . . . . .> 2,4
Ток потребления от источника питания I_{cc}, мА . . . . . .> 60
Ток утечки каналов A,B,C,D при не выбранном
режиме I_{oz}, мкА . . . . . .-100,...,100
Ток утечки на управляющих входах I_{il} мкА . . . . . -100,...,100

6. Литература

1. Юров В., Хорошенко С.
Ассемблер: учебный курс. – СПб: Питер Ком, 1999. – 672 с.

2. Гук М.
Аппаратные средства IBM PC. Энциклопедия. – СПб: Питер Ком, 1999. – 816 с.

3. Айден К., Колесниченко О. и др.
Аппаратные средства PC. 2-е издание, переработанное и дополненное. – СПб: BHV-Санкт-Петербург, 1998. – 608 с.

4. Калабеков Б. А.
Цифровые устройства и микропроцессорные системы. – М: Радио и связь, 1997. – 336 с.

5. Микропроцессорный комплект К1810: Структура, программирование, применение. - Справочник. /Под ред. Ю. М. Казаринова - М.: Высш. шк., 1990 - 269 с.

6. Скэнлон Л. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера. - М.: Радио и связь, 1991. - 464 с.

7. Ю-Чжен Л., Г. Гибсон. Микропроцессоры семейства 8086/8088. Архитектура, программирование и проектирование микрокомпьютерных систем. - М.: Радио и связь, 1987. - 512 с.

8. Брэдли Д. Дж. Программирование на языке ассемблера для персональной ЭВМ фирмы IBM. /Под ред. А. А. Чижова - М.: Радио и связь, 1988. - 447 с.

9. Шнайдер Ол. Язык ассемблера для персонального компьютера фирмы IBM. /А. Шнайдер. Пер. с англ. под ред. Е. К. Масловского и др. - М.: Мир, 1988. - 405 с.

10. Абель П. Язык Ассемблера для IBM PC и программирования./Пер. с англ. Ю. В. Сальникова. - М.: Высш. шк., 1992. - 447 с.

11. Нортон П., Соухе Д. Язык ассемблера для IBM PC - М.: Изд. "Компьютер", финансы и статистика, 1992. - 352 с.

7. СОДЕРЖАНИЕ