МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

ЭВМ и периферийные устройства

Электронный учебно-методический комплекс
по дисциплине в LMS Moodle

САМАРА
2012

УДК 681.3.07

Автор-составитель: **Лёзин Илья Александрович**

ЭВМ и периферийные устройства [Электронный ресурс] : электрон. учеб.-метод. комплекс по дисциплине в LMS Moodle / Минобрнауки России, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т); авт.-сост. И. А. Лёзин. - Электрон. текстовые и граф. дан. - Самара, 2012. – 1 эл. опт. диск (CDROM).

В состав учебно-методического комплекса входят:
1. Курс лекций.
2. Методические указания к лабораторным работам 1-5.
3. Методические указания к лабораторной работе 6.
4. Задания для лабораторных работ.
5. Вопросы к экзамену.
6. Тесты для контроля знаний.
7. Рабочая программа курса.

УМКД «ЭВМ и периферийные устройства» предназначен для студентов факультета информатики, обучающихся по направлению подготовки бакалавров 230100.62 «Информатика и вычислительная техника» в 6 семестре.

УМКД разработан на кафедре информационных систем и технологий.

Конспект лекций для студентов дневного отделения по дисциплине «ЭВМ и периферийные устройства»

Тема 1. Становление и эволюция цифровой вычислительной техники

Вычислительная машина и вычислительная система

Изучение любого вопроса принято начинать с договоренностей о терминологии. В нашем случае определению подлежат понятия вычислительная машина (ВМ) и вычислительная система (ВС). Сразу же оговорим, что предметом рассмотрения будут исключительно цифровые машины и системы, то есть устройства, оперирующие дискретными величинами. В литературе можно найти множество самых различных определений терминов «вычислительная машина» и «вычислительная система». Итак, вычислительная машина – это:

1. Устройство, которое принимает данные, обрабатывает их в соответствии с хранимой программой, генерирует результаты и обычно состоит из блоков ввода, вывода, памяти, арифметики, логики и управления.

2. Функциональный блок, способный выполнять реальные вычисления, включающие множественные арифметические и логические операции, без участия человека в процессе этих вычислений.

3. Устройство, способное:

- хранить программу или программы обработки и по меньшей мере информацию, необходимую для выполнения программы;
- быть свободно перепрограммируемым в соответствии с требованиями пользователя;
- выполнять арифметические вычисления, определяемые пользователем;
- выполнять без вмешательства человека программу обработки, требующую изменения действий путем принятия логических решений в процессе обработки.

Мы воспользуемся наиболее общим их определением.

Термином *вычислительная машина* будем обозначать комплекс технических и программных средств, предназначенный для автоматизации подготовки и решения задач пользователей.

В свою очередь, *вычислительную систему* определим как совокупность взаимосвязанных и взаимодействующих процессоров или вычислительных машин, периферийного оборудования и программного обеспечения, предназначенную для подготовки и решения задач пользователей.

Таким образом, формально отличие ВС от ВМ выражается в количестве вычислителей.

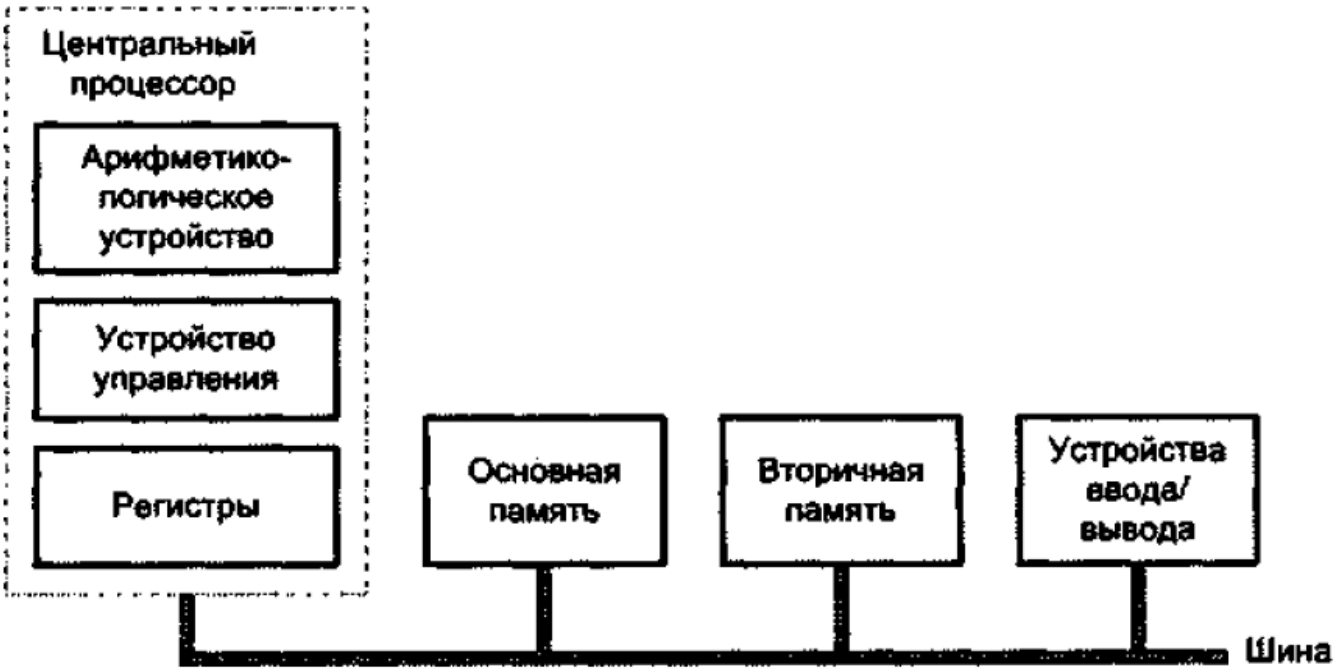
Структуры вычислительных машин

До недавнего времени примерно одинаковое распространение получили два способа построения вычислительных машин: *с непосредственными связями* и *на основе шины*. Типичным представителем первого способа может служить классическая фон-неймановская ВМ. В ней между взаимодействующими устройствами (процессор, память, устройство ввода/вывода) имеются непосредственные связи.

Особенности связей (число линий в шинах, пропускная способность и т. п.) определяются видом информации, характером и интенсивностью обмена. Достоинством архитектуры с непосредственными связями можно считать возможность развязки «узких мест» путем улучшения структуры и характеристик только определенных связей, что экономически может быть наиболее выгодным решением. У фон-неймановских ВМ таким

«узким местом» является канал пересылки данных между ЦП и памятью, и «развязать» его достаточно непросто. Кроме того, ВМ с непосредственными связями плохо поддаются реконфигурации.

В варианте с общей шиной все устройства вычислительной, машины подключены к магистральной шине, служащей единственным трактом для потоков команд, данных и управления. Наличие общей шины существенно упрощает реализацию ВМ, позволяет легко менять состав и конфигурацию машины. Благодаря этим свойствам шинная архитектура получила широкое распространение в мини- и микроЭВМ. Вместе с тем, именно с шиной связан и основной недостаток архитектуры: в каждый момент передавать информацию по шине может только одно устройство. Основную нагрузку на шину создают обмены между процессором и памятью, связанные с извлечением из памяти команд и данных и записью в память результатов вычислений. На операции ввода/вывода остается лишь часть пропускной способности шины. Практика показывает, что даже при достаточно быстрой шине для 90% приложений этих остаточных ресурсов обычно не хватает, особенно в случае ввода или вывода больших массивов данных.



В целом следует признать, что при сохранении фон-неймановской концепции последовательного выполнения команд программы шинная архитектура в чистом ее виде оказывается недостаточно эффективной. Более распространена *архитектура с иерархией шин,* где помимо магистральной шины имеется еще несколько дополнительных шин. Они могут обеспечивать непосредственную связь между устройствами с наиболее интенсивным обменом, например процессором и кэш-памятью. Другой вариант использования дополнительных шин – объединение однотипных устройств ввода/вывода с последующим выходом с дополнительной шины на магистральную. Все эти меры позволяют снизить нагрузку на общую шину и более эффективно расходовать ее пропускную способность.

Структуры вычислительных систем

Понятие «вычислительная система» предполагает наличие множества процессоров или законченных вычислительных машин, при объединении которых используется один из двух подходов. В вычислительных *системах с общей памятью* имеется общая основная память, совместно используемая всеми процессорами системы. Связь процессоров с памятью обеспечивается с помощью коммуникационной сети, чаще всего

вырождающейся в общую шину. Таким образом, структура ВС с общей памятью аналогична рассмотренной выше архитектуре с общей шиной, в силу чего ей свойственны те же недостатки. Применительно к вычислительным системам данная схема имеет дополнительное достоинство: обмен информацией между процессорами не связан с дополнительными операциями и обеспечивается за счет доступа к общим областям памяти.

Альтернативный вариант организации – *распределенная* система, где общая память вообще отсутствует, а каждый процессор обладает собственной локальной памятью. Часто такие системы объединяют отдельные ВМ. Обмен информацией между составляющими системы обеспечивается с помощью коммуникационной сети посредством обмена сообщениями.

Подобное построение ВС снимает ограничения, свойственные для общей шины, но приводит к дополнительным издержкам на пересылку сообщений между процессорами или машинами.

Основные показатели вычислительных машин

Использование конкретной вычислительной машины имеет смысл, если ее показатели соответствуют показателям, определяемым требованиями к реализации заданных алгоритмов. В качестве основных показателей ВМ обычно рассматривают: емкость памяти, быстродействие и производительность, стоимость и надежность.

Целесообразно рассматривать два вида быстродействия: номинальное и среднее. *Номинальное быстродействие* характеризует возможности ВМ при выполнении

стандартной операции. В качестве стандартной обычно выбирают короткую операцию сложения. *Среднее быстродействие* характеризует скорость вычислений при выполнении эталонного алгоритма или некоторого класса алгоритмов. Величина среднего быстродействия зависит как от параметров ВМ, так и от параметров алгоритма.

Производительность ВМ оценивается количеством эталонных алгоритмов, выполняемых в единицу времени. Производительность является более универсальным показателем, чем среднее быстродействие, поскольку в явном виде зависит от порядка прохождения задач через ВМ.

Вычислительную машину можно определить множеством показателей, характеризующих отдельные ее свойства. Возникает задача введения меры для оценки степени приспособленности ВМ к выполнению возложенных на нее функций – меры эффективности.

Эффективность определяет степень соответствия ВМ своему назначению. Она измеряется либо количеством затрат, необходимых для получения определенного результата, либо результатом, полученным при определенных затратах. Произвести сравнительный анализ эффективности нескольких ВМ, принять решение на использование конкретной машины позволяет критерий эффективности.

Критерий эффективности – это правило, служащее для сравнительной оценки качества вариантов ВМ. Критерий эффективности можно назвать правилом предпочтения сравниваемых вариантов.

Строятся критерии эффективности на основе частных показателей эффективности (показателей качества). Способ связи между частными показателями определяет вид критерия эффективности.

Тема 2. Принципы построения и функционирования

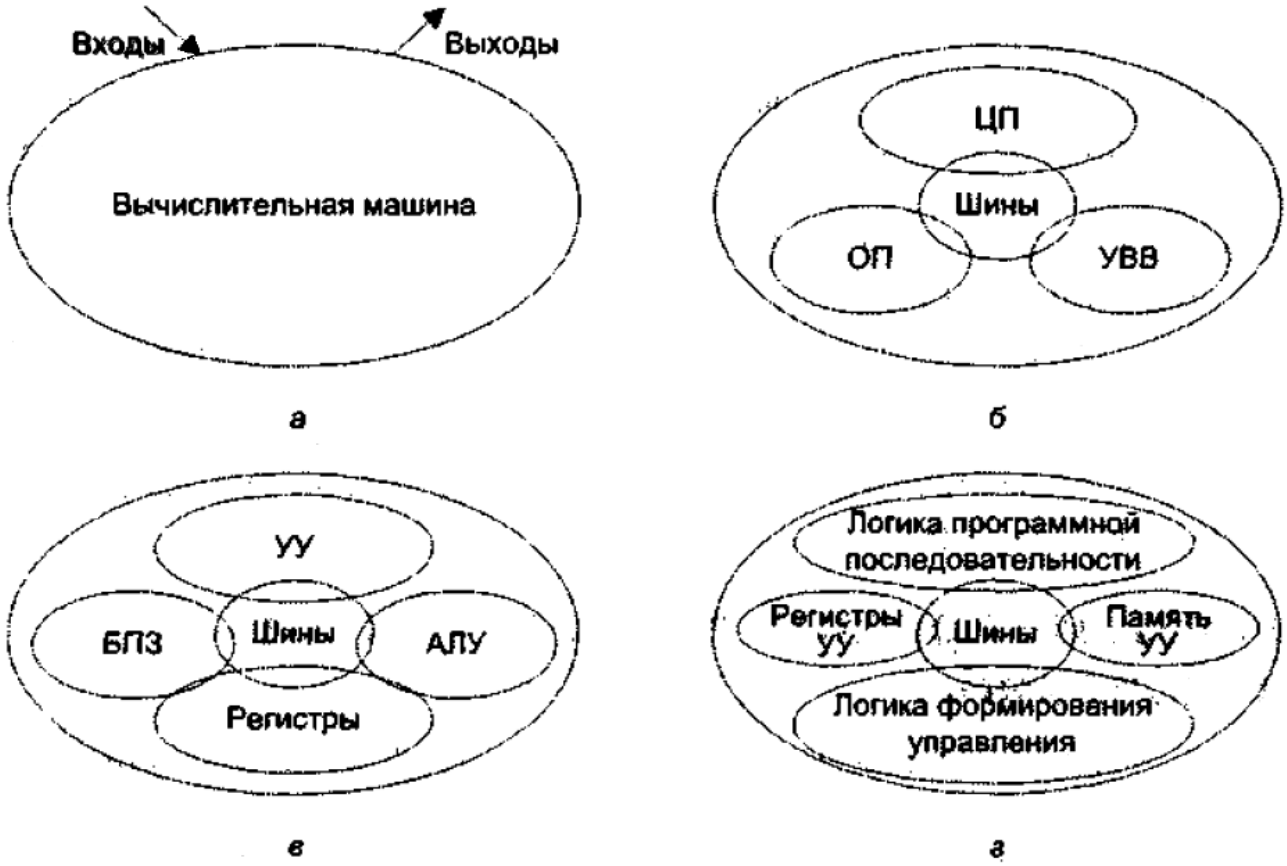
Архитектура и уровни детализации

Рассмотрение принципов построения и функционирования вычислительных машин и систем предварим определением термина *архитектура*.

Под архитектурой вычислительной машины обычно понимается логическое построение ВМ, то есть то, какой машина представляется программисту. Впервые термин «архитектура вычислительной машины» (computer architecture) был употреблен фирмой IBM при разработке машин семейства IBM 360 для описания тех средств, которыми может пользоваться программист, составляя программу на уровне машинных команд. Подобную трактовку называют «узкой», и охватывает она перечень и формат команд, формы представления данных, механизмы ввода/вывода, способы адресации памяти и т. п. Из рассмотрения выпадают вопросы физического построения вычислительных средств: состав устройств, число регистров процессора, емкость памяти, наличие специального блока для обработки вещественных чисел, тактовая частота центрального процессора и т. д. Этот круг вопросов принято определять понятием *организация* или *структурная организация.*

Архитектура (в узком смысле) и организация – это две стороны описания ВМ и ВС. Поскольку для наших целей, помимо теоретической строгости, такое деление не дает каких-либо преимуществ, то в дальнейшем будем пользоваться термином «архитектура», правда, в «широком» его толковании, объединяющем как архитектуру в узком смысле, так и организацию ВМ.

Круг вопросов, рассматриваемых в данном курсе, по большей части относится к компетенции системного архитектора и охватывает различные степени детализации ВМ и ВС.



Уровни детализации вычислительной машины: а – уровень «черного ящика»; б – уровень общей архитектуры; в – уровень архитектуры центрального процессора; г – уровень архитектуры устройства управления.

Концепция машины с хранимой в памяти программой

Введем новое определение термина «вычислительная машина» как совокупности технических средств, служащих для автоматизированной обработки дискретных данных по заданному алгоритму.

Алгоритм – одно из фундаментальных понятий математики и вычислительной техники. Международная организация стандартов (ISO) формулирует понятие *алгоритм* как «конечный набор предписаний, определяющий решение задачи посредством конечного количества операций» (ISO 2382/1-84). Помимо этой стандартизированной формулировки существуют и другие определения. Приведем наиболее распространенные из них. Итак, алгоритм – это:

- способ преобразования информации, задаваемый с помощью конечной системы правил;
- совокупность правил, определяющих эффективную процедуру решения любой задачи из некоторого заданного класса задач;
- точно определенное правило действий, для которого задано указание, как и в какой последовательности это правило необходимо применять к исходным данным задачи, чтобы получить ее решение.

Основными свойствами алгоритма являются: дискретность, определенность, массовость и результативность.

Дискретность выражается в том, что алгоритм описывает действия над дискретной информацией (например, числовой или символьной), причем сами эти действия также дискретны.

Свойство *определенности* означает, что в алгоритме указано все, что должно быть сделано, причем ни одно из действий не должно трактоваться двояко.

Массовость алгоритма подразумевает его применимость к множеству значений исходных данных, а не только к каким-то уникальным значениям.

Наконец, *результативность* алгоритма состоит в возможности получения результата за конечное число шагов.

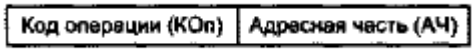
Рассмотренные свойства алгоритмов предопределяют возможность их реализации на ВМ, при этом процесс, порождаемый алгоритмом, называют *вычислительным процессом.*

Сущность фон-неймановской концепции вычислительной машины можно свести к четырем принципам:

- двоичного кодирования;
- программного управления;
- однородности памяти;
- адресности.

Принцип двоичного кодирования

Согласно этому принципу, вся информация, как данные, так и команды, кодируются двоичными цифрами 0 и 1. Каждый тип информации представляется двоичной последовательностью и имеет свой *формат.* Последовательность битов в формате, имеющая определенный смысл, называется *полем.* В числовой информации обычно выделяют *поле знака* и *поле значащих разрядов.* В формате команды можно выделить два поля: *поле кода операции* (КОп) и *поле адресов* (адресную часть — АЧ).



Код операции представляет собой указание, какая операция должна быть выполнена. Вид адресной части и число составляющих ее адресов зависят от типа команды.

Принцип программного управления

Все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде *программы,* состоящей из последовательности управляющих слов – *команд.* Каждая команда предписывает некоторую операцию из набора операций, реализуемых вычислительной машиной. Команды программы хранятся в последовательных ячейках памяти вычислительной машины и выполняются *в естественной последовательности,* то есть в порядке их положения в программе.

При необходимости, с помощью специальных команд, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд программы принимается либо на основании анализа результатов предшествующих вычислений, либо безусловно.

Принцип однородности памяти

Команды и данные хранятся в одной и той же памяти и внешне в памяти неразличимы. Распознать их можно только по способу использования. Это позволяет производить над командами те же операции, что и над числами, и, соответственно, открывает ряд возможностей. Так, циклически изменяя адресную часть команды, можно обеспечить обращение к последовательным элементам массива данных.

Такой прием носит название *модификации команд* и с позиций современного программирования не приветствуется. Более полезным является другое следствие принципа однородности, когда команды одной программы могут быть получены как результат исполнения другой программы. Эта возможность лежит в основе *трансляции* – перевода текста программы с языка высокого уровня на язык конкретной ВМ.

Принцип адресности

Структурно основная память состоит из пронумерованных ячеек, причем процессору в произвольный момент доступна любая ячейка. Двоичные коды команд и данных разделяются на единицы информации, называемые *словами,* и хранятся в ячейках памяти, а для доступа к ним используются номера соответствующих ячеек – *адреса.*

Типы и форматы операндов

Машинные команды оперируют данными, которые в этом случае принято называть операндами. К наиболее общим (базовым) типам операндов можно отнести: адреса, числа, символы и логические данные. Помимо них ВМ обеспечивает обработку и более сложных информационных единиц: графических изображений, аудио-, видео- и анимационной информации. Такая информация является производной от базовых типов данных и хранится в виде файлов на внешних запоминающих устройствах. Для каждого типа данных в ВМ предусмотрены определенные форматы.

Числовая информация

Среди цифровых данных можно выделить две группы:

• целые типы, используемые для представления целых чисел;

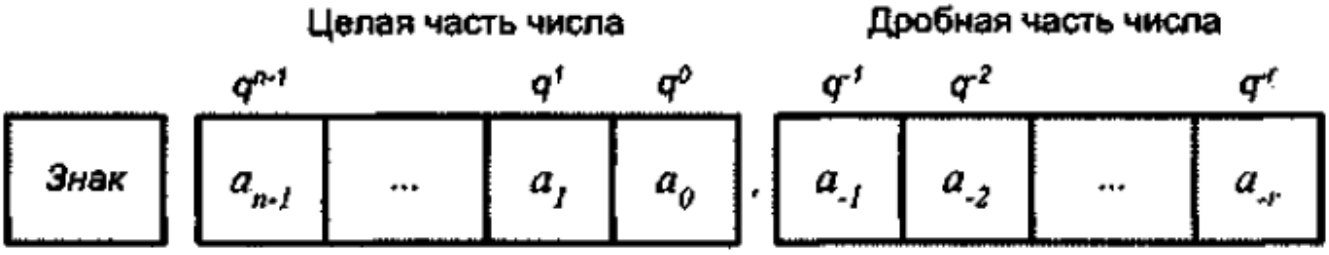
• вещественные типы для представления рациональных чисел.

В рамках первой группы имеется несколько форматов представления численной информации, зависящих от ее характера. Для представления вещественных чисел используется форма с плавающей запятой.

Числа в форме с фиксированной запятой

Представление числа X в форме с фиксированной запятой (ФЗ), которую иногда называют также естественной формой, включает в себя знак числа и его модуль в q-ичном коде. Здесь q – основание системы счисления или база. Для современных ВМ характерна двоичная система (q = 2), но иногда используются также восьмеричная (q = 8) или шестнадцатеричная (q = 16) системы счисления. Запятую в записи числа называют соответственно двоичной, восьмеричной или шестнадцатеричной. Знак положительного числа кодируется двоичной цифрой 0, а знак отрицательного числа – цифрой 1.

Числам с ФЗ соответствует запись вида $X = \pm a_{n-1}...a_1a_0a_{-1}a_{-2}...a_{-r}$. Отрицательные числа обычно представляются в дополнительном коде. Разряд кода числа, в котором размещается знак, называется знаковым разрядом кода. Разряды, где располагаются значащие цифры числа, называются цифровыми разрядами кода. Знаковый разряд размещается левее старшего цифрового разряда. Положение запятой одинаково для всех чисел и в процессе решения задач не меняется. Хотя запятая и фиксируется, в коде числа она никак не выделяется, а только подразумевается. В общем случае разрядная сетка ВМ для размещения чисел в форме с ФЗ имеет вид, где n разрядов используются для записи целой части числа и r разрядов – для дробной части.



Упакованные целые числа

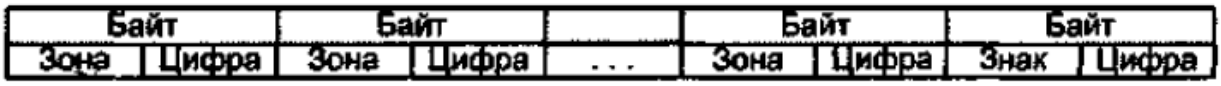
В АСК современных микропроцессоров имеются команды, оперирующие целыми числами, представленными в упакованном виде. Связано это с обработкой мультимедийной информации. Формат предполагает упаковку в пределах достаточно длинного слова (обычно 64-разрядного) нескольких небольших целых чисел, а соответствующие команды обрабатывают все эти числа параллельно. Если каждое из чисел состоит из четырех двоичных разрядов, то в 64-разрядное слово можно поместить до 16 таких чисел. Неиспользованные разряды заполняются нулями. В микропроцессорах фирмы Intel, начиная с Pentium MMX, присутствуют специальные команды для обработки мультимедийной информации (MMX-команды), оперирующие целыми числами, упакованными в квадрослова (64-разрядные слова). Предусмотрены три формата: упакованные байты (восемь 8-разрядных чисел); упакованные слова (четыре 16-разрядных числа) и упакованные двойные слова (два 32-разрядных числа).

Байты в формате упакованных байтов нумеруются от 0 до 7, причем байт 0 располагается в младших разрядах квадрослова. Аналогичная система нумерации и размещения упакованных чисел применяется для упакованных слов (номера 0-3) и упакованных двойных слов (номера 0-1).

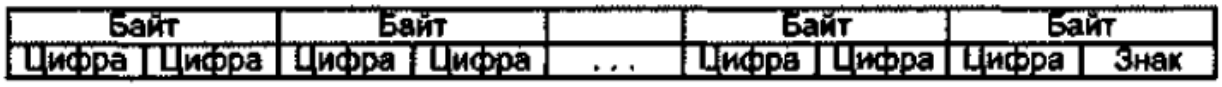
Идентичные форматы упакованных данных применяются также в другой технологии обработки мультимедийной информации, предложенной фирмой AMD. Эта технология носит название 3DNow!, а реализована в микропроцессорах данной фирмы.

Десятичные числа

В ряде задач, главным образом, учетно-статистического характера, приходится иметь дело с хранением, обработкой и пересылкой десятичной информации. Особенность таких задач состоит в том, что обрабатываемые числа могут состоять из различного и весьма большого количества десятичных цифр. Традиционные методы обработки с переводом исходных данных в двоичную систему счисления и обратным преобразованием результата зачастую сопряжены с существенными накладными расходами. По этой причине в ВМ применяются иные специальные формы представления десятичных данных. В их основу положен принцип кодирования каждой десятичной цифры эквивалентным двоичным числом из четырех битов (тетрадой), то есть так называемым двоично-десятичным кодом (BCD – Binary Coded Decimal).



а



б

Используются два формата представления десятичных чисел (все числа рассматриваются как целые): *зонный (распакованный)* и *уплотненный (упакованный).* В обоих форматах каждая десятичная цифра представляется двоичной тетрадой, то есть заменяется двоично-десятичным кодом. Из оставшихся задействованных шести четырехразрядных двоичных комбинаций ($2^4 = 16$) две служат для кодирования знаков «+» и «-». Например, в ВМ семейства IBM 360/370/390 для знака «плюс» выбран код $1100_2 = C_{16}$, а для знака «минус» - код $1101_2 = D_{16}$.

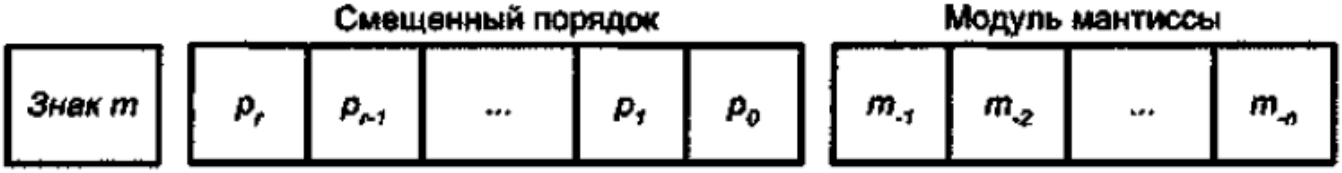
Числа в форме с плавающей запятой

От недостатков ФЗ в значительной степени свободна форма представления чисел с *плавающей запятой* (ПЗ), известная также под названиями *нормальной* или

полулогарифмической формы. В данном варианте каждое число разбивается на две группы цифр. Первая группа цифр называется *мантиссой,* вторая – *порядком.* Число представляется в виде произведения $X = \pm mq^{\pm p}$, где m – мантисса числа X, p – порядок числа, q – основание системы счисления.

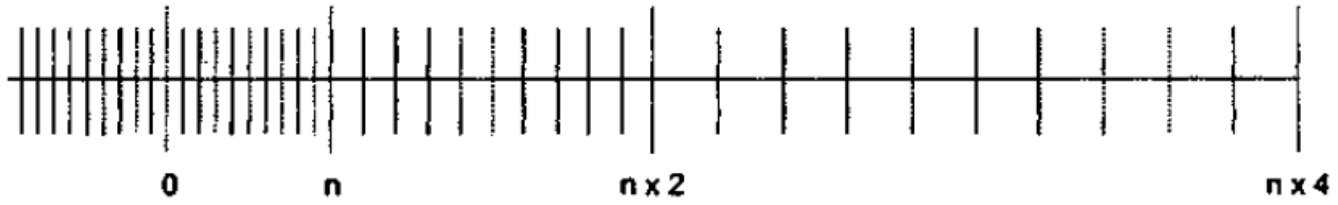
Для представления числа в форме с ПЗ требуется задать знаки мантиссы и порядка, их модули в q-ичном коде, а также основание системы счисления. Нормальная форма неоднозначна, так как взаимное изменение m и p приводит к «плаванию» запятой, чем и обусловлено название этой формы. Диапазон и точность представления чисел с ПЗ зависят от числа разрядов, отводимых под порядок и мантиссу.

В большинстве вычислительных машин для упрощения операций над порядками последние приводят к целым положительным числам, применяя так называемый смещенный порядок. Для этого к истинному порядку добавляется целое положительное число – смещение. Например, в системе со смещением 128 порядок -3 представляется как 125 (-3 + 128). Обычно смещение выбирается равным половине представимого диапазона порядков. Отметим, что смещенный порядок занимает все биты поля порядка, в том числе и тот, который ранее использовался для записи знака порядка.



Мантисса в числах с ПЗ обычно представляется в *нормализованной форме.* Это означает, что на мантиссу налагаются такие условия, чтобы она по модулю была меньше единицы ($|q| < 1$), а первая цифра после точки отличалась от нуля. Полученная таким образом мантисса называется *нормализованной.*

Следует также отметить, что числа в форме с ПЗ, в отличие от чисел в форме с ФЗ, размещены на числовой оси неравномерно. Возможные значения в начале числовой оси расположены плотнее, а по мере движения вправо – все реже. Это означает, что многие вычисления приводят к результату, который не является точным, то есть представляет собой округление до ближайшего значения, представимого в данной форме записи.



Упакованные числа с плавающей запятой

В последних версиях АСК, предусматривающих особые команды для обработки мультимедийной информации, помимо упакованных целых чисел используются и упакованные числа с плавающей запятой. Так, в уже упоминавшейся технологии 3DNow! фирмы AMD имеются команды, служащие для увеличения производительности систем при обработке трехмерных приложений, описываемых числами с ПЗ. Каждая такая команда работает с двумя операндами с плавающей запятой одинарной точности. Операнды упаковываются в 64-разрядные группы.

Символьная информация

В общем объеме вычислительных действий все большая доля приходится на обработку символьной информации, содержащей буквы, цифры, знаки препинания, математические и другие символы. Каждому символу ставится в соответствие определенная двоичная комбинация. Совокупность возможных символов и назначенных

им двоичных кодов образует таблицу кодировки. В настоящее время применяется множество различных таблиц кодировки. До недавнего времени наиболее распространенными были кодовые таблицы, в которых символы кодируются с помощью восьмиразрядных двоичных комбинаций (байтов), позволяющих представить 256 различных символов:

• расширенный двоично-кодированный код EBCDIC (Extended Binary Coded Decimal Interchange Code);

• американский стандартный код для обмена информацией ASCII (American ' Standard Code for Information Interchange).

Код EBCDIC используется в качестве внутреннего кода в универсальных ВМ фирмы IBM. Он же известен под названием ДКОИ (двоичный код для обработки информации).

Стандартный код ASCII – 7-разрядный, восьмая позиция отводится для записи бита четности. Это обеспечивает представление 128 символов, включая все латинские буквы, цифры, знаки основных математических операций и знаки пунктуации. Позже появилась европейская модификация ASCII, называемая Latin 1 (стандарт ISO 8859-1). В ней «полезно» используются все 8 разрядов. Дополнительные комбинации (коды 128-255) в новом варианте отводятся для представления специфических букв алфавитов западно-европейских языков, символов псевдографики, некоторых букв греческого алфавита, а также ряда математических и финансовых символов. Именно эта кодовая таблица считается мировым стандартом де-факто, который применяется с различными модификациями во всех странах. В зависимости от использования кодов 128-255 различают несколько вариантов стандарта ISO 8859.

Хотя код ASCII достаточно удобен, он все же слишком тесен и не вмещает множества необходимых символов. По этой причине в 1993 году консорциумом компаний Apple Computer, Microsoft, Hewlett-Packard, DEC и IBM был разработан 16-битовый, стандарт ISO 10646, определяющий универсальный набор символов (UCS, Universal Character Set). Новый код, известный под названием Unicode, позволяет задать до 65 536 символов, то есть дает возможность одновременно представить символы всех основных «живых» и «мертвых» языков. Для букв русского языка выделены коды 1040-1093.

Логические данные

Элементом логических данных является логическая (булева) переменная, которая может принимать лишь два значения: «истина» или «ложь». Кодирование логического значения принято осуществлять битом информации: единицей кодируют истинное значение, нулем – ложное. Как правило, в ВМ оперируют наборами логических переменных длиной в машинное слово. Обрабатываются такие слова с помощью команд логических операций (И, ИЛИ, НЕ и т. д.), при этом все биты обрабатываются одинаково, но независимо друг от друга, то есть никаких переносов между разрядами не возникает.

Строки

Строки – это непрерывная последовательность битов, байтов, слов или двойных слов. *Битовая строка* может начинаться в любой позиции байта и содержать до 2^{32} бит. *Байтовая строка* может состоять из байтов, слов или двойных слов. Длина такой строки варьируется от нуля до 2^{32}-1 байт (4 Гбайт). Приведенные цифры характерны для 32-разрядных ВМ. Если байты байтовой строки представляют собой коды символов, то говорят о *текстовой строке.* Поскольку длина текстовой строки может меняться в очень широких пределах, то для указания конца строки в последний байт заносится код-ограничитель – обычно это нули во всех разрядах байта. Иногда вместо ограничителя длину строки указывают числом, расположенным в первом байте (двух) строки.

Типы команд

Несмотря на различие в системах команд разных ВМ, некоторые основные типы операций могут быть найдены в любой из них. Для описания этих типов примем следующую классификацию:

- команды пересылки данных;
- команды арифметической и логической обработки;
- команды работы со строками;
- команды SIMD;
- команды преобразования;
- команды ввода/вывода;
- команды управления потоком команд.

Команды пересылки данных

Это наиболее распространенный тип машинных команд. В таких командах должна содержаться следующая информация:

- адреса источника и получателя операндов – адреса ячеек памяти, номера регистров процессора или информация о том, что операнды расположены в стеке;
- длина подлежащих пересылке данных (обычно в байтах или словах), заданная явно или косвенно;
- способ адресации каждого из операндов, с помощью которого содержимое адресной части команды может быть пересчитано в физический адрес операнда.

Рассматриваемая группа команд обеспечивает передачу информации между процессором и ОП, внутри процессора и между ячейками памяти. Пересылочные операции внутри процессора имеют тип «регистр-регистр». Передачи между процессором и памятью относятся к типу «регистр-память», а пересылки в памяти – к типу «память-память».

Команды арифметической и логической обработки

В данную группу входят команды, обеспечивающие арифметическую и логическую обработку информации в различных формах ее представления. Для каждой формы представления чисел в АСК обычно предусматривается некий стандартный набор операций.

Помимо вычисления результата выполнение арифметических и логических операций сопровождается формированием в АЛУ признаков (флагов), характеризующих этот результат. Наиболее часто фиксируются такие признаки, как: Z (Zero) – нулевой результат; N (Negative) – отрицательный результат; V (oVerflow) – переполнение разрядной сетки; C (Carry) – наличие переноса.

SIMD-команды

Название данного типа команд представляет собой аббревиатуру от Single Instruction Multiple Data – буквально «одна инструкция – много данных». В отличие от обычных команд, оперирующих двумя числами, SIMD-команды обрабатывают сразу две группы чисел (в принципе их можно называть групповыми командами). Операнды таких команд обычно представлены в одном из упакованных форматов.

Идея SIMD-обработки была выдвинута в Институте точной механики и вычислительной техники им. С. А. Лебедева в 1978 году в рамках проекта «Эльбрус-1». С 1992 года команды типа SIMD становятся неотъемлемым элементом АСК микропроцессоров фирм Intel и AMD. Поводом послужило широкое распространение мультимедийных приложений.

Команды для работы со строками

Для работы со строками в АСК обычно предусматриваются команды, обеспечивающие перемещение, сравнение и поиск строк. В большинстве машин перечисленные операции просто имитируются за счет других команд.

Команды преобразования

Команды преобразования осуществляют изменение формата представления данных. Примером может служить преобразование из десятичной системы счисления в двоичную или перевод 8-разрядного кода символа из кодировки ASCII в кодировку EBCDIC, и наоборот.

Команды ввода/вывода

Команды этой группы могут быть подразделены на команды управления периферийным устройством (ПУ), проверки его состояния, ввода и вывода. Команды *управления периферийным устройством* служат для запуска ПУ и указания ему требуемого действия. Например, накопителю на магнитной ленте может быть предписана необходимость перемотки ленты или ее продвижения вперед на одну запись. Трактовка подобных инструкций зависит от типа ПУ.

Команды *проверки состояния ввода/вывода* применяются для тестирования различных признаков, характеризующих состояние модуля В/ВЫВ и подключенных к нему ПУ. Благодаря этим командам центральный процессор может выяснить, включено ли питание ПУ, завершена ли предыдущая операция ввода/вывода, возникли ли в процессе ввода/вывода какие-либо ошибки и т.п.

Собственно обмен информацией с ПУ обеспечивают команды ввода и вывода. Команды *ввода* предписывают модулю В/ВЫВ получить элемент данных (байт или слово) от ПУ и поместить его на шину данных, а команды *вывода* – заставляют модуль В/ВЫВ принять элемент данных с шины данных и переслать его на ПУ.

Команды управления системой

Команды, входящие в эту группу, являются привилегированными и могут выполняться, только когда центральный процессор ВМ находится в привилегированном состоянии или выполняет программу, находящуюся в привилегированной области памяти (обычно привилегированный режим используется лишь операционной системой). Так, лишь эти команды способны считывать и изменять состояние ряда регистров устройства управления.

Команды управления потоком команд

Концепция фон-неймановской вычислительной машины предполагает, что команды программы, как правило, выполняются в порядке их расположения в памяти. Для получения адреса очередной команды достаточно увеличить содержимое счетчика команд на длину текущей команды. В то же время основные преимущества ВМ заключаются именно в возможности изменения хода вычислений в зависимости от возникающих в процессе счета результатов. С этой целью в АСК вычислительной машины включаются команды, позволяющие нарушить естественный порядок следования и передать управление в иную точку программы. В адресной части таких команд содержится адрес точки перехода (адрес той команды, которая должна быть выполнена следующей). Переход реализуется путем загрузки адреса точки перехода в счетчик команд (вместо увеличения содержимого этого счетчика на длину команды).

В системе команд ВМ можно выделить три типа команд, способных изменить последовательность вычислений:

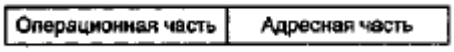
- безусловные переходы;
- условные переходы (ветвления);
- вызовы процедур и возвраты из процедур.

Форматы команд

Типовая команда, в общем случае, должна указывать:

- подлежащую выполнению операцию;
- адреса исходных данных (операндов), над которыми выполняется операция;
- адрес, по которому должен быть помещен результат операции.

В соответствии с этим команда состоит из двух частей: операционной и адресной.



Формат команды определяет ее структуру, то есть количество двоичных разрядов, отводимых под всю команду, а также количество и расположение отдельных полей команды. *Полем* называется совокупность двоичных разрядов, кодирующих составную часть команды. При создании ВМ выбор формата команды влияет на многие характеристики будущей машины. Оценивая возможные форматы, нужно учитывать следующие факторы:

- общее число различных команд;
- общую длину команды;
- тип полей команды (фиксированной или переменной длины) и их длина;
- простоту декодирования;
- адресность и способы адресации;
- стоимость оборудования для декодирования и исполнения команд.

Длина команды

Это важнейшее обстоятельство, влияющее на организацию и емкость памяти, структуру шин, сложность и быстродействие ЦП. С одной стороны, удобно иметь в распоряжении мощный набор команд, то есть как можно больше кодов операций, операндов, способов адресации, и максимальное адресное пространство. Однако все это требует выделения большего количества разрядов под каждое поле команды, что приводит к увеличению ее длины. Вместе с тем, для ускорения выборки из памяти желательно, чтобы команда была как можно короче, а ее длина была равна или кратна ширине шины данных. Для упрощения аппаратуры и повышения быстродействия ВМ длину команды обычно выбирают кратной байту, поскольку в большинстве ВМ основная память организована в виде 8-битовых ячеек. В рамках системы команд одной ВМ могут использоваться разные форматы команд.

Обычно это связано с применением различных способов адресации. В таком случае в состав кода команды вводится поле для задания способа адресации (СА), и обобщенный формат команды приобретает вид:

В большинстве ВМ одновременно уживаются несколько различных форматов команд.

Количество двоичных разрядов, отводимых под код операции, выбирается так, чтобы можно было представить любую из операций.

В адресной части команды содержится информация о местонахождении исходных данных и месте сохранения результата операции. Обычно местонахождение каждого из операндов и результата задается в команде путем указания адреса соответствующей ячейки основной памяти или номера регистра процессора. Принципы использования информации из адресной части команды определяет *система адресации.* Система адресации задает *число адресов в команде* команды и принятые *способы адресации.*

Для определения количества адресов, включаемых в адресную часть, будем использовать термин *адресность*. В «максимальном» варианте необходимо указать три компонента: адрес первого операнда, адрес второго операнда и адрес ячейки, куда заносится результат операции. В принципе может быть добавлен еще один адрес, указывающий место хранения следующей инструкции. В итоге имеет место *четырехадресный формат команды*.

Время выполнения одной команды складывается из времени выполнения операции и времени обращения к памяти.

Для трехадресной команды последнее суммируется из четырех составляющих времени:

- выборки команды;
- выборки первого операнда;
- выборки второго операнда;
- записи в память результата.

Одноадресная команда требует двух обращений к памяти:

- выборки команды;
- выборки операнда.

Как видно, на выполнение одноадресной команды затрачивается меньше времени, чем на обработку трехадресной команды, однако для реализации одной трехадресной команды, как правило, нужно три одноадресных. Этих соображений тем не менее не достаточно, чтобы однозначно отдать предпочтение тому или иному варианту адресности.

Система операций

Системой операций называется список операций, непосредственно выполняемых техническими средствами вычислительной машины. Система операций ВМ определяется областью ее применения, требованиями к стоимости, производительности и точности вычислений.

Связь системы операций с алгоритмами решаемых задач проявляется в степени ее приспособленности для записи программ реализации этих алгоритмов. Степень приспособленности характеризуется близостью списка операций системы команд и операций, используемых на каждом шаге выполнения алгоритмов. Простоту программирования алгоритма часто определяют термином «программируемость вычислительной машины». Чем меньше команд требуется для составления программы реализации какого-либо алгоритма, тем программируемость выше.

В архитектурах типа CISC улучшения программируемости добиваются введением в систему операций большого количества операций, в том числе и достаточно сложных. Это может приводить и к повышению производительности ВМ, хотя в любом случае увеличивает аппаратные затраты.

Качество системы операций можно характеризовать двумя свойствами: функциональной полнотой и эффективностью.

Функциональная полнота – это достаточность системы операций для описания любых алгоритмов. Системы операций ВМ включают в себя большое количество машинных операций и практически всегда являются функционально полными.

Эффективность системы операций показывает степень соответствия СО заданному классу алгоритмов и требованиям к производительности ВМ. Количественно эффективность характеризуется затратами оборудования, затратами времени на реализацию алгоритмов и вероятностью правильного выполнения программ.

Цикл команды

Программа в фон-неймановской ЭВМ реализуется центральным процессором (ЦП) посредством последовательного исполнения образующих эту программу команд.

Действия, требуемые для выборки (извлечения из основной памяти) и выполнения команды, называют циклом команды. В общем случае цикл команды включает в себя несколько составляющих (этапов):

- выборку команды;
- формирование адреса следующей команды;
- декодирование команды;
- вычисление адресов операндов;
- выборку операндов;
- исполнение операции;
- запись результата.

Перечисленные этапы выполнения команды в дальнейшем будем называть стандартным циклом команды. Отметим, что не все из этапов присутствуют при выполнении любой команды (зависит от типа команды), тем не менее этапы выборки, декодирования, формирования адреса следующей команды и исполнения имеют место всегда.

В определенных ситуациях возможны еще два этапа:

- косвенная адресация;
- реакция на прерывание.

Этап выборки команды

Цикл любой команды начинается с того, что центральный процессор извлекает команду из памяти, используя адрес, хранящийся в счетчике команд (СК). Двоичный код команды помещается в регистр команды (РК) и с этого момента становится «видимым» для процессора.

Система команд многих ВМ предполагает несколько форматов команд, причем в разных форматах команда может занимать 1, 2 или более ячеек, а этап выборки команды можно считать завершенным лишь после того, как в РК будет помещен полный код команды. Информация о фактической длине команды содержится в полях кода операции и способа адресации. Обычно эти поля располагают в первом слове кода команды, и для выяснения необходимости продолжения процесса выборки необходимо предварительное декодирование их содержимого. Такое декодирование может быть произведено после того, как первое слово кода команды окажется в РК. В случае многословного формата команды процесс выборки продолжается вплоть до занесения в РК всех слов команды.

Этап формирования адреса следующей команды

Для фон-неймановских машин характерно размещение соседних команд программы в смежных ячейках памяти. Если извлеченная команда не нарушает естественного порядка выполнения программы, для вычисления адреса следующей выполняемой команды достаточно увеличить содержимое счетчика команд на длину текущей команды, представленную количеством занимаемых кодом команды ячеек памяти.

Длина команды, а также то, способна ли она изменить естественный порядок выполнения команд программы, выясняются в ходе ранее упоминавшегося предварительного декодирования. Если извлеченная команда способна изменить последовательность выполнения программы (команда условного или безусловного перехода, вызова процедуры и т. п.), процесс формирования адреса следующей команды переносится на этап исполнения операции. В силу сказанного, в ряде ВМ рассматриваемый этап цикла команды следует не за выборкой команды, а находится в конце цикла.

Этап декодирования команды

После выборки команды она должна быть декодирована, для чего ЦП расшифровывает находящийся в РК код команды. В результате декодирования выясняются следующие моменты:

• находится ли в РК полный код команды или требуется дозагрузка остальных слов команды;

• какие последующие действия нужны для выполнения данной команды;

• если команда использует операнды, то откуда они должны быть взяты (номер регистра или адрес ячейки основной памяти);

• если команда формирует результат, то куда этот результат должен быть направлен.

Ответы на два первых вопроса дает расшифровка кода операции, результатом которой может быть унитарный код, где каждый разряд соответствует одной из команд. На практике вместо унитарного кода могут встретиться самые разнообразные формы представления результатов декодирования, например адрес ячейки специальной управляющей памяти, где хранится первая микрокоманда микропрограммы для реализации указанной в команде операции.

Полное выяснение всех аспектов команды, помимо расшифровки кода операции, требует также анализа адресной части команды, включая поле способа адресации.

По результатам декодирования производится подготовка электронных схем ВМ к выполнению предписанных командой действий.

Этап вычисления адресов операндов

Этап имеет место, если в процессе декодирования команды выясняется, что команда использует операнды. Если операнды размещаются в основной памяти, осуществляется вычисление их исполнительных адресов, с учетом указанного в команде способа адресации. Так, в случае индексной адресации для получения исполнительного адреса производится суммирование содержимого адресной части команды и содержимого индексного регистра.

Этап выборки операндов

Вычисленные на предыдущем этапе исполнительные адреса используются для считывания операндов из памяти и занесения в определенные регистры процессора. Например, в случае арифметической команды операнд после извлечения из памяти может быть загружен во входной регистр АЛУ. Однако чаще операнды предварительно заносятся в специальные вспомогательные регистры процессора, а их пересылка на вход АЛУ происходит на этапе исполнения операции.

Этап исполнения операции

На этом этапе реализуется указанная в команде операция. В силу различия сущности каждой из команд ВМ, содержание этого этапа также сугубо индивидуально.

Этап записи результата

Этап записи результата присутствует в цикле тех команд, которые предполагают занесение результата в регистр или ячейку основной памяти. Фактически его можно считать частью этапа исполнения, особенно для тех команд, которые помещают результат сразу в несколько мест.

Машинный цикл с прерыванием

Практически во всех ВМ предусмотрены средства, благодаря которым модули ввода/вывода (и не только они) могут прервать выполнение текущей программы для внеочередного выполнения другой программы, с последующим возвратом к прерванной.

Первоначально прерывания были введены для повышения эффективности вычислений при работе с медленными ПУ. Положим, что процессор пересылает данные на принтер, используя стандартный цикл команды. После каждой операции записи ЦП будет вынужден сделать паузу, в ожидании подтверждения от принтера об обработке символа. Длительность этой паузы может составлять сотни и тысячи циклов команды. Ясно, что такое использование ЦП очень неэффективно. В случае прерываний, пока протекает операция ввода/вывода, ЦП способен выполнять другие команды.

В упрощенном виде процедуру прерывания можно описать следующим образом. Объект, требующий внеочередного обслуживания, выставляет на соответствующем входе ЦП сигнал *запроса прерывания.* Перед переходом к очередному циклу команды процессор проверяет этот вход на наличие запроса. Обнаружив запрос, ЦП запоминает информацию, необходимую для продолжения нормальной работы после возврата из прерывания, и переходит к выполнению *программы обработки прерывания* (обработчика прерывания). По завершении обработки прерывания ЦП восстанавливает состояние прерванного процесса, используя запомненную информацию, и продолжает вычисления.

В терминах цикла команды сказанное выглядит так. Для учета прерываний к циклу команды добавляется этап прерывания, в ходе которого процессор проверяет, не поступил ли запрос прерывания. Если запроса нет, ЦП переходит к этапу выборки следующей команды программы. При наличии запроса процессор:

1. Приостанавливает выполнение текущей программы и запоминает содержимое всех регистров, которые будут использоваться программой обработки прерывания. Это называется *сохранением контекста программы.* В первую очередь необходимо сохранить содержимое счетчика команд, аккумулятора и регистра признаков. Контекст программы обычно сохраняется в стеке.

2. Заносит в счетчик команд начальный адрес программы обработки прерывания. Теперь процессор продолжает с этапа выборки первой команды обработчика прерывания. Обработчик (обычно он входит в состав операционной системы) определяет природу прерывания и выполняет необходимые действия. Когда программа обработки прерывания завершается, процессор может возобновить выполнение программы пользователя с точки, где она была прервана. Для этого он восстанавливает контекст программы (содержимое СК и других регистров) и начинает с цикла выборки очередной команды прерванной программы.

Тема 3. Функциональная организация фон-неймановской ВМ

Фон-неймановская архитектура

В статье фон Неймана определены основные устройства ВМ, с помощью которых должны быть реализованы вышеперечисленные принципы. Большинство современных ВМ по своей структуре отвечают принципу программного управления.

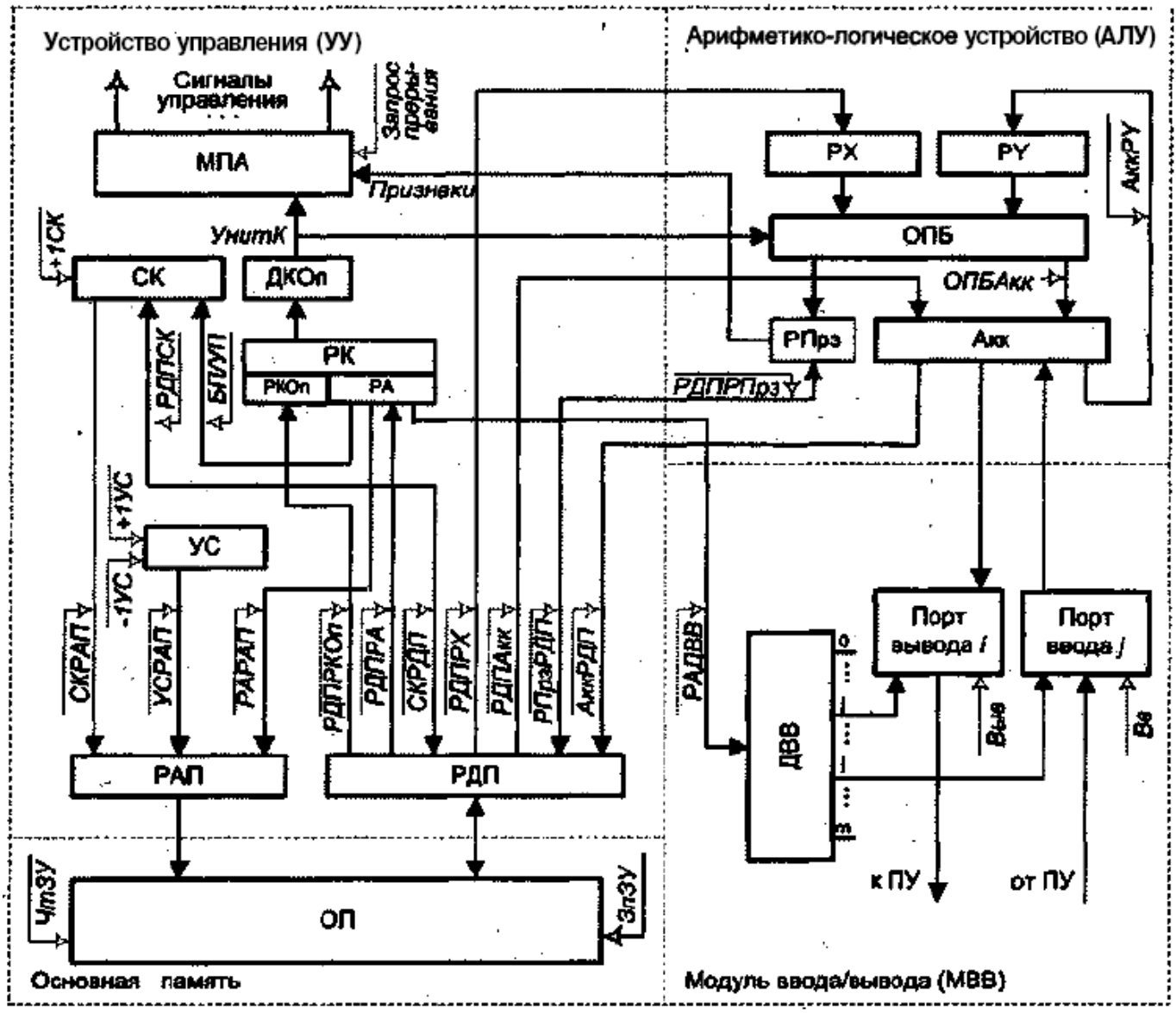
Типичная фон-неймановская ВМ содержит: память, устройство управления, арифметико-логическое устройство и устройство ввода/вывода.

В любой ВМ имеются средства для ввода программ и данных к ним. Информация поступает из подсоединенных к ЭВМ *периферийных устройств* (ПУ) ввода. Результаты вычислений выводятся на периферийные устройства вывода. Связь и взаимодействие ВМ и ПУ обеспечивают *порты ввода* и *порты вывода.*

Введенная информация сначала запоминается в основной памяти, а затем переносится во вторичную память, для длительного хранения. Чтобы программа могла выполняться, команды и данные должны располагаться в *основной памяти* (ОП), организованной таким образом, что каждое двоичное слово хранится в отдельной ячейке, идентифицируемой адресом, причем соседние ячейки памяти имеют следующие по порядку адреса. Доступ к любым ячейкам запоминающего устройства (ЗУ) основной памяти может производиться в произвольной последовательности. Такой вид памяти известен как *память с произвольным доступом.*

Функциональная схема фон-неймановской ВМ

Чтобы получить более детальное представление о структуре и функциях устройств ВМ, обратимся к схеме гипотетической машины с аккумуляторной архитектурой.



Для упрощения изложения приняты следующие характеристики машины:

- **Одноадресные команды.** Адресная часть команды содержит только один адрес. При выполнении операций с двумя операндами предполагается, что другой операнд находится в специальном регистре АЛУ – аккумуляторе, а результат также остается в аккумуляторе.
- **Единство форматов.** Длина команд и данных совпадает с разрядностью ячеек памяти, то есть любая команда или операнд занимают только одну ячейку памяти. Таким образом, адрес очередной команды в памяти может быть получен путем прибавления единицы к адресу текущей команды, а для извлечения из памяти любой команды или любого операнда достаточно одного обращения к памяти.

На функциональной схеме показаны типовые узлы каждого из основных устройств ВМ, а также сигналы, инициирующие выполнение отдельных операций по пересылке информации и ее обработке, необходимых для функционирования машины. УУ организует автоматическое выполнение программ и функционирование ВМ как единой системы. Теперь остановимся на описании узлов, реализующих целевую функцию УУ.

Счетчик команд

Счетчик команд (СК) – неотъемлемый элемент устройства управления любой ВМ, построенной в соответствии с фон-неймановским принципом программного управления. Согласно этому принципу соседние команды программы располагаются в ячейках памяти

со следующими по порядку адресами и выполняются преимущественно в той же очередности, в какой они размещены в памяти ВМ. Таким образом, адрес очередной команды может быть получен путем увеличения адреса ячейки, из которой была считана текущая команда, на длину выполняемой команды, представленную числом занимаемых ею ячеек. Реализацию такого режима и при зван обеспечивать счетчик команд – двоичный счетчик, в котором хранится и модифицируется адрес очередной команды программы. Перед началом вычислений в СК заносится адрес ячейки основной памяти, где хранится команда, которая должна быть выполнена первой. В процессе выполнения каждой команды путем увеличения содержимого СК на длину выполняемой команды в счетчике формируется адрес следующей подлежащей выполнению команды. В рассматриваемой ВМ любая команда занимает одну ячейку, поэтому содержимое СК увеличивается на единицу, что обеспечивается подачей сигнала управления +1СК. По завершении текущей команды адрес следующей команды программы всегда берется из счетчика команд. Для изменения естественного порядка вычислений (перехода в иную точку программы) достаточно занести в СК адрес точки перехода.

Хотя термин «счетчик команд» считается общепринятым, его нельзя признать вполне удачным из-за того, что он создает неверное впечатление о задачах данного узла. По этой причине разработчики ВМ используют иные названия, в частности *программный счетчик* (PC, Program Counter) или *указатель команды* (IP, Instruction Pointer). Последнее определение представляется наиболее удачным, поскольку точнее отражает назначение рассматриваемого узла УУ.

В заключение добавим, что в ряде ВМ счетчик команд реализуется в виде обычного регистра, а увеличение его содержимого производится внешней схемой (схемой инкремента/декремента).

Регистр команды

Счетчик команд определяет лишь местоположение команды в памяти, но не содержит информации о том, что это за команда. Чтобы приступить к выполнению команды, ее необходимо извлечь из памяти и разместить в *регистре команды* (РК). Этот этап носит название *выборки команды*. Только с момента загрузки команды в РК она становится «видимой» для процессора. В РК команда хранится в течение всего времени ее выполнения. Как уже отмечалось ранее, любая команда содержит два поля: поле кода операции и поле адресной части. Учитывая это обстоятельство, регистр команды иногда рассматривают как совокупность двух регистров – *регистра кода операции* (РКОп) *и регистра адреса* (РА), в которых хранятся соответствующие составляющие команды.

Если команда занимает несколько последовательных ячеек, то код операции всегда находится в том слове команды, которое извлекается из памяти первым. Это позволяет по коду операции определить, требуются ли считывание из памяти и загрузка в РК остальных слов команды. Собственно выполнение команды начинается только после занесения в РК ее полного кода.

Указатель стека

Указатель стека (УС) – это регистр, где хранится адрес вершины стека. В реальных вычислительных машинах стек реализуется в виде участка основной памяти, обычно расположенного в области наибольших адресов. Заполнение стека происходит в сторону уменьшения, адресов, при этом вершина стека – это ячейка, куда была произведена последняя по времени запись. Для хранения адреса такой ячейки и предназначен УС. При выполнении операции *push* (занесение в стек) содержимое УС с помощью сигнала -1УС сначала уменьшается на единицу, после чего используется в качестве адреса, по которому производится запись. Соответствующая ячейка становится новой вершиной стека. Считывание из стека (операция *pop*) происходит из ячейки, на которую указывает

текущий адрес в УС, после чего содержимое указателя стека сигналом +1УС увеличивается на единицу. Таким образом, вершина стека опускается, а считанное слово считается удаленным из стека. Хотя физически считанное слово и осталось в ячейке памяти, при следующей записи в стек оно будет заменено новой информацией.

Регистр адреса памяти

Регистр адреса памяти (РАП) предназначен для хранения адреса ячейки основной памяти вплоть до завершения операции (считывание или запись) с этой ячейкой. Наличие РАП позволяет компенсировать различия в быстродействии ОП и прочих устройств машины.

Регистр данных памяти

Регистр данных памяти (РДП) призван компенсировать разницу в быстродействии запоминающих устройств и устройств, выступающих в роли источников и потребителей хранимой информации. В РДП при чтении заносится содержимое ячейки ОП, а при записи – помещается информация, подлежащая сохранению в ячейке ОП. Собственно момент считывания и записи в ячейку определяется сигналами ЧтЗУ и ЗпЗУ соответственно.

Дешифратор кода операции

Дешифратор кода операции (ДКОп) преобразует код операции в форму, требуемую для работы микропрограммного автомата (МПА). Информация после декодирования определяет последующие действия МПА, а ее вид зависит от организации МПА. В рассматриваемой ВМ – это унитарный код УнитК. Часто код операции преобразуется в адрес первой команды микропрограммы, реализующей указанную в команде операцию. С этих позиций ДКОп правильнее было бы назвать не дешифратором, а преобразователем кодов.

Микропрограммный автомат

Микропрограммный автомат (МПА) правомочно считать центральным узлом устройства управления. Именно МПА формирует последовательность сигналов управления, в соответствии с которыми производятся все действия, необходимые для выборки из памяти и выполнения команд. Исходной информацией для МПА служат: декодированный код операции, состояние признаков (флагов), характеризующих результат предшествующих вычислений, а также внешние запросы на прерывание текущей программы и переход на программу обслуживания прерывания.

Арифметико-логическое устройство

Это устройство, как следует из его названия, предназначено для арифметической и логической обработки данных. Оно содержит следующие узлы:

- операционный блок,
- регистры операндов,
- регистр признаков,
- аккумулятор.

Операционный блок

Операционный блок (ОПБ) представляет собой ту часть АЛУ, которая, собственно, и выполняет арифметические и логические операции над поданными на вход операндами. Выбор конкретной операции из возможного списка операций для данного ОПБ определяется кодом операции команды. В нашей ВМ код операции поступает непосредственно из регистра команды. В реальных машинах КОп зачастую преобразуется

в МПА в иную форму и уже из микропрограммного автомата поступает в АЛУ. Операционные блоки современных АЛУ строятся как комбинационные схемы, то есть они не обладают внутренней памятью и до момента сохранения результата операнды должны присутствовать на входе блока.

Регистры операндов

Регистры РХ и РУ обеспечивают сохранение операндов на входе операционного блока вплоть до получения результата операции и его записи (в нашем случае в аккумулятор).

Регистр признаков

Регистр признаков (РПрз) предназначен для фиксации и хранения признаков (флагов), характеризующих результат последней выполненной арифметической или логической операции. Такие признаки могут информировать о равенстве результата нулю, о знаке результата, о возникновении переноса из старшего разряда, переполнении разрядной сетки и т. д. Содержимое РПрз обычно используется устройством управления для реализации условных переходов по результатам операций АЛУ. Под каждый из возможных признаков отводится один разряд РПрз.

Формирование признаков осуществляется блоком формирования состояний регистра признаков, который может входить в состав ОПБ либо реализуется в виде внешней схемы, располагаемой между операционным блоком и РПрз.

Аккумулятор

Аккумулятор (Акк) – это регистр, на который возлагаются самые разнообразные функции. Так, в него предварительно загружается один из операндов, участвующих в арифметической или логической операции. В Акк может храниться результат предыдущей команды и в него же заносится результат очередной операции. Через Акк зачастую производятся операции ввода и вывода.

Строго говоря, аккумулятор в равной мере можно отнести как к АЛУ, так и к УУ, а в ВМ с регистровой архитектурой его можно рассматривать как один из регистров общего назначения.

Основная память

Вне зависимости от типа используемых микросхем основная память (ОП) представляет собой массив запоминающих элементов (ЗЭ), организованных в виде ячеек, способных хранить некую единицу информации, обычно один байт. Каждая ячейка имеет уникальный адрес. Ячейки ОП организованы в виде матрицы, а выбор ячейки осуществляется путем подачи разрешающих сигналов на соответствующие строку и столбец этой матрицы. Это обеспечивается дешифратором адреса памяти, преобразующим поступивший из РАП адрес ячейки в разрешающие сигналы, подаваемые в горизонтальную и вертикальную линии, на пересечении которых расположена адресуемая ячейка. При современной емкости ОП для реализации данных сигналов приходится использовать несколько микросхем запоминающих устройств (ЗУ). В этих условиях процесс обращения к ячейке состоит из выбора нужной микросхемы (на основании старших разрядов адреса) и выбора ячейки внутри микросхемы (определяется младшими разрядами адреса). Первая часть процедуры производится внешними схемами, а вторая – внутри микросхем ЗУ.

Модуль ввода/вывода

Структура приведенного *модуля ввода/вывода* (МВБ) обеспечивает только пояснение логики работы ВМ. В реальных ВМ реализация этого устройства машины может существенно отличаться от рассматриваемой. Задачей МВВ является обеспечение подключения к ВМ различных периферийных устройств (ПУ) и обмена информацией с ними. В рассматриваемом варианте МВВ состоит из дешифратора номера порта ввода/вывода, множества портов ввода и множества портов вывода.

Порты ввода и порты вывода

Портом называют схему, ответственную за передачу информации из периферийного устройства ввода в аккумулятор АЛУ (порт ввода) или из аккумулятора на периферийное устройство вывода (порт вывода). Схема обеспечивает электрическое и логическое сопряжение ВМ с подключенным к нему периферийным устройством.

Тема 4. Архитектура системы команд

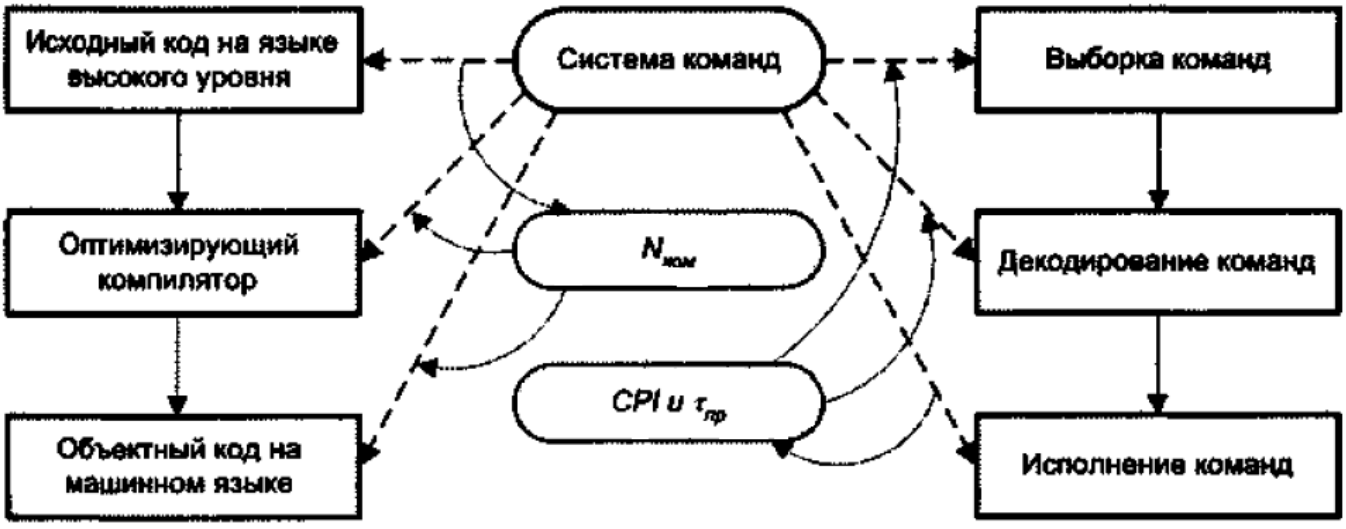
Архитектура системы команд

Системой команд вычислительной машины называют полный перечень команд, которые способна выполнять данная ВМ. В свою очередь, под *архитектурой системы команд* (АСК) принято определять те средства вычислительной машины, которые видны и доступны программисту. АСК можно рассматривать как линию согласования нужд разработчиков программного обеспечения с возможностями создателей аппаратуры вычислительной машины.

В конечном итоге, цель тех и других – реализация вычислений наиболее эффективным образом, то есть за минимальное время, и здесь важнейшую роль играет правильный выбор архитектуры системы команд. В упрощенной трактовке время выполнения программы ($T_{выч}$) можно определить через число команд в программе ($N_{ком}$), среднее количество тактов процессора, приходящихся на одну команду (CPI), и длительность тактового периода:

$$T_{выч} = N_{ком} * CPI * \tau_{пр}.$$

Каждая из составляющих выражения зависит от одних аспектов архитектуры системы команд и, в свою очередь, влияет на другие, что свидетельствует о необходимости чрезвычайно ответственного подхода к выбору АСК.



Общая характеристика архитектуры системы команд вычислительной машины складывается из ответов на следующие вопросы:

1. Какого вида данные будут представлены в вычислительной машине и в какой форме?

2. Где эти данные могут храниться помимо основной памяти?

3. Каким образом будет осуществляться доступ к данным?

4. Какие операции могут быть выполнены над данными?

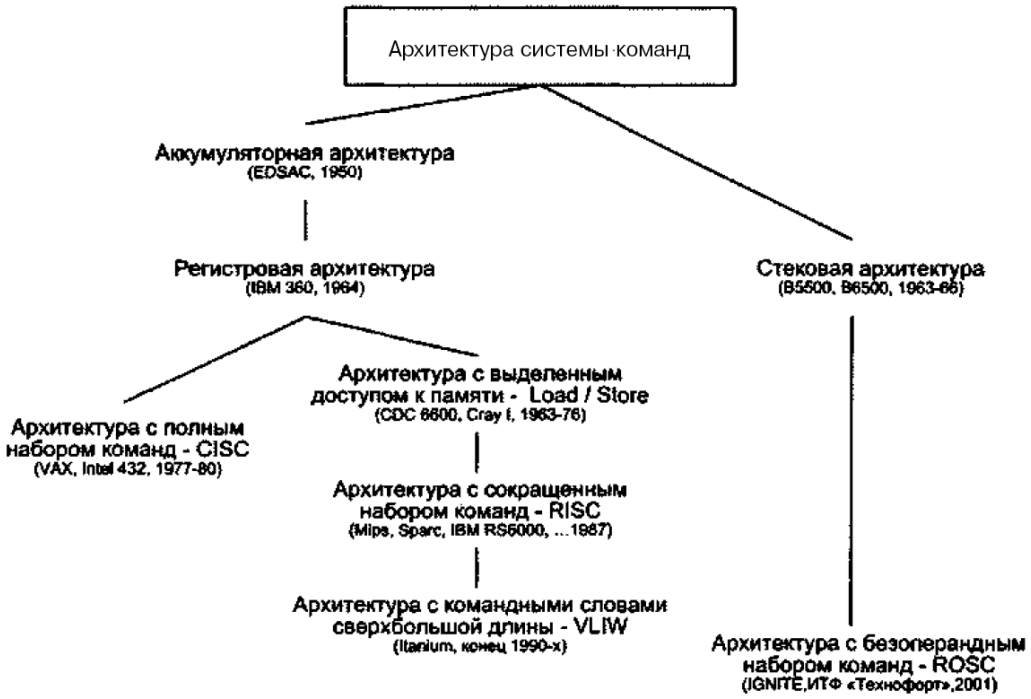
5. Сколько операндов может присутствовать в команде?

6. Как будет определяться адрес очередной команды?

7. Каким образом будут закодированы команды?

Классификация архитектур системы команд

В истории развития вычислительной техники как в зеркале отражаются изменения, происходившие во взглядах разработчиков на перспективность той или иной архитектуры системы команд.



Среди мотивов, чаще всего предопределяющих переход к новому типу АСК, остановимся на двух наиболее существенных. Первый – это состав операций, выполняемых вычислительной машиной, и их сложность. Второй – место хранения операндов, что влияет на количество и длину адресов, указываемых в адресной части команд обработки данных. Именно эти моменты взяты в качестве критериев излагаемых ниже вариантов классификации архитектур системы команд.

Классификация по составу и сложности команд

Современная технология программирования ориентирована на языки высокого уровня (ЯВУ), главная цель которых – облегчить процесс программирования. Переход к ЯВУ, однако, породил серьезную проблему: сложные операторы, характерные для ЯВУ, существенно отличаются от простых машинных операций, реализуемых в большинстве вычислительных машин. Проблема получила название *семантического разрыва,* а ее следствием становится недостаточно эффективное выполнение программ на ВМ. Пытаясь преодолеть семантический разрыв, разработчики вычислительных машин в настоящее время выбирают один из трех подходов и, соответственно, один из трех типов АСК:

• архитектуру с полным набором команд: CISC (Complex Instruction Set Computer);

• архитектуру с сокращенным набором команд: RISC (Reduced Instruction Set Computer);

• архитектуру с командными словами сверхбольшой длины: VLIW (Very Long Instruction Word).

Классификация по месту хранения операндов

Количество команд и их сложность, безусловно, являются важнейшими факторами, однако не меньшую роль при выборе АСК играет ответ на вопрос о том, где могут храниться операнды и каким образом к ним осуществляется доступ. С этих позиций различают следующие виды архитектур системы команд:

• стековую;
• аккумуляторную;
• регистровую;
• с выделенным доступом к памяти.

Выбор той или иной архитектуры влияет на принципиальные моменты: сколько адресов будет содержать адресная часть команд, какова будет длина этих адресов, насколько просто будет происходить доступ к операндам и какой, в конечном итоге, будет общая длина команд.

Стеком называется память, по своей структурной организации отличная от основной памяти ВМ. Выделим только те аспекты, которые требуются для пояснения особенностей АСК на базе стека. Стек образует множество логически взаимосвязанных ячеек, взаимодействующих по принципу «последним вошел, первым вышел» (LIFO, Last In First Out).

Верхнюю ячейку называют *вершиной стека.* Для работы со стеком предусмотрены две операции: *push* (проталкивание данных в стек) и *pop* (выталкивание данных из стека). Запись возможна только в верхнюю ячейку стека, при этом вся хранящаяся в стеке информация предварительно проталкивается на одну позицию вниз. Чтение допустимо также только из вершины стека. Извлеченная информация удаляется из стека, а оставшееся его содержимое продвигается вверх. В вычислительных машинах, где реализована АСК на базе стека (их обычно называют стековыми), операнды перед обработкой помещаются в две верхних ячейки стековой памяти.

Архитектура на базе аккумулятора исторически возникла одной из первых. В ней для хранения одного из операндов арифметической или логической операции в процессоре имеется выделенный регистр – аккумулятор. В этот же регистр заносится и результат операции. Поскольку адрес одного из операндов предопределен, в командах обработки достаточно явно указать местоположение только второго операнда. Изначально оба операнда хранятся в основной памяти, и до выполнения операции один из них нужно загрузить в аккумулятор. После выполнения команды обработки результат находится в аккумуляторе и, если он не является операндом для последующей команды, его требуется сохранить в ячейке памяти.

В машинах с регистровой архитектурой процессор включает в себя массив регистров (регистровый файл), известных как регистры общего назначения (РОН). Эти регистры, в каком-то смысле, можно рассматривать как явно управляемый кэш для хранения недавно использовавшихся данных. Размер регистров обычно фиксирован и совпадает с размером машинного слова. К любому регистру можно обратиться, указав его номер. Количество РОН в архитектурах типа CISC обычно невелико (от 8 до 32), и для представления номера конкретного регистра необходимо не более пяти разрядов, благодаря чему в адресной части команд обработки допустимо одновременно указать номера двух, а зачастую и трех регистров (двух регистров операндов и регистра результата). RISC-архитектура предполагает использование существенно большего числа РОН (до нескольких сотен), однако типичная для таких ВМ длина команды (обычно 32 разряда) позволяет определить в команде до трех регистров. Регистровая архитектура допускает расположение операндов в одной из двух запоминающих сред: основной памяти или регистрах. С учетом

возможного размещения операндов в рамках регистровых АСК выделяют три подвида команд обработки: регистр-регистр, регистр-память, память-память.

В архитектуре с выделенным доступом к памяти обращение к основной памяти возможно только с помощью двух специальных команд: load и store. В английской транскрипции данную архитектуру называют Load/Store architecture. Команда load (загрузка) обеспечивает считывание значения из основной памяти и занесение его в регистр процессора (в команде обычно указывается адрес ячейки памяти и номер регистра). Пересылка информации в противоположном направлении производится командой store (сохранение). Операнды во всех командах обработки информации могут находиться только в регистрах процессора (чаще всего в регистрах общего назначения). Результат операции также заносится в регистр. В архитектуре отсутствуют команды обработки, допускающие прямое обращение к основной памяти. Допускается наличие в АСК ограниченного числа команд, где операнд является частью кода команды.

Тема 5. Архитектура и программирование сопроцессора

Сопроцессор

Устройства для обработки чисел с плавающей точкой появились в компьютерах давно. В архитектуре семейства процессоров Intel 80x86 устройство для обработки чисел с плавающей точкой появилось в составе компьютера на базе процессора I8086/88 и получило название математический сопроцессор (далее просто сопроцессор).

Для чего нужен сопроцессор, какие возможности добавляет он к тому, что делает основной процессор, кроме обработки еще одного формата данных? Перечислим некоторые из них.

Полная поддержка стандартов IEEE-754 и 854 на арифметику с плавающей точкой. Эти стандарты описывают как форматы данных, с которыми должен работать сопроцессор, так и набор реализуемых им функций.

Поддержка численных алгоритмов для вычисления значений тригонометрических функций, логарифмов и т. п. Эта работа сопроцессора выполняется абсолютно прозрачно для программиста, что само по себе очень ценно, так как не требует от него разработки соответствующих подпрограмм.

Обработка десятичных чисел с точностью до 18 разрядов, что позволяет сопроцессору без округления выполнять арифметические операции над целыми десятичными числами со значениями до 10^{18}.

Обработка вещественных чисел из диапазона $3{,}37*10^{-4932}...1{,}18*10^{+4932}$.

Архитектура сопроцессора

Как и в случае с основным процессором, интерес для нас представляет программная модель сопроцессора. С точки зрения программиста, сопроцессор представляет собой совокупность регистров, каждый из которых имеет свое функциональное назначение.

В программной модели сопроцессора можно выделить три группы регистров.

Восемь регистров RO.. .R7 составляют основу программной модели сопроцессора – стек сопроцессора. Размерность каждого регистра – 80 битов. Реализация численных алгоритмов на основе стека сопроцессора позволяет получить существенный выигрыш в скорости вычислений.

Три служебных регистра:

- регистр состояния сопроцессора SWR (Status Word Register) отражает информацию о текущем состоянии сопроцессора и какие исключения возникли после выполнения последней команды и т. д.;
- управляющий регистр сопроцессора CWR (Control Word Register) управляет режимами работы сопроцессора;
- регистр слова тегов TWR (Tags Word Register) используется для контроля за состоянием каждого из регистров RO.. .R7.

Два регистра указателей – данных DPR (Data Point Register) и команд IPR (Instruction Point Register) – предназначены для запоминания информации об адресе команды, вызвавшей исключительную ситуацию, и адресе ее операнда.

Все эти регистры являются программно доступными. Однако к одним из них доступ получить довольно легко, для этого в системе команд сопроцессора существуют специальные команды, а к другим его получить сложнее, так как специальных команд для этого нет, поэтому необходимо выполнять дополнительные действия.

Тема 6. Организация шин

Организация шин

Совокупность трактов, объединяющих между собой основные устройства ВМ (центральный процессор, память и модули ввода/вывода), образует *структуру взаимосвязей* вычислительной машины. Структура взаимосвязей должна обеспечивать обмен информацией между:

• центральным процессором и памятью;
• центральным процессором и модулями ввода/вывода;
• памятью и модулями ввода/вывода.

Информационные потоки, характерные для основных устройств ВМ, показаны на рисунке.

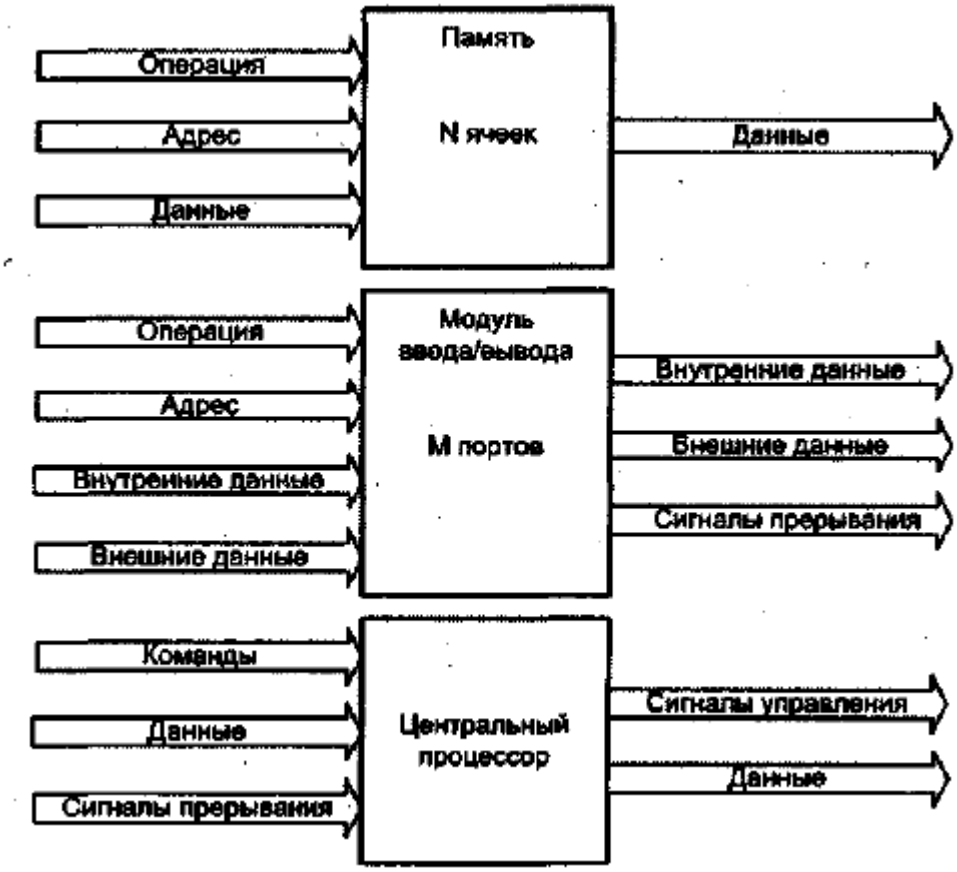
Взаимосвязь частей ВМ и ее «общение» с внешним миром обеспечиваются системой шин. Большинство машин содержат несколько различных шин, каждая из которых оптимизирована под определенный вид коммуникаций. Часть шин скрыта внутри интегральных микросхем или доступна только в пределах печатной платы. Некоторые шины имеют доступные извне точки, с тем чтобы к ним легко можно было подключить дополнительные устройства, причем большинство таких шин не просто доступны, но и отвечают определенным стандартам, что позволяет подсоединять к шине устройства различных производителей.

Чтобы охарактеризовать конкретную шину, нужно описать:

• совокупность сигнальных линий;

• физические, механические и электрические характеристики шины;

• используемые сигналы арбитража, состояния, управления и синхронизации;



• правила взаимодействия подключенных к шине устройств (протокол шины).

Шину образует набор коммуникационных линий, каждая из которых способна передавать сигналы, представляющие двоичные цифры 1 и 0. По линии может пересылаться развернутая во времени последовательность таких сигналов. При

совместном использовании несколько линий могут обеспечить одновременную (параллельную) передачу двоичных чисел. Физически линии шины реализуются в виде отдельных проводников, как полоски проводящего материала на монтажной плате либо как алюминиевые или медные проводящие дорожки на кристалле микросхемы.

Операции на шине называют *транзакциями.* Основные виды транзакций – *транзакции чтения* и *транзакции записи.* Если в обмене участвует устройство ввода/вывода, можно говорить о *транзакциях ввода* и *вывода,* по сути эквивалентных транзакциям чтения и записи соответственно. Шинная транзакция включает в себя две части: посылку адреса и прием (или посылку) данных.

Важным критерием, определяющим характеристики шины, может служить ее целевое назначение. По этому критерию можно выделить: шины «процессор-память», шины ввода/вывода, системные шины.

Тема 7. Память

Память

В любой ВМ, вне зависимости от ее архитектуры, программы и данные хранятся в памяти. Функции памяти обеспечиваются запоминающими устройствами (ЗУ), предназначенными для фиксации, хранения и выдачи информации в процессе работы ВМ. Процесс фиксации информации в ЗУ называется *записью,* процесс выдачи информации – *чтением* или *считыванием,* а совместно их определяют как *процессы обращения* к ЗУ.

Характеристики систем памяти

Перечень основных характеристик, которые необходимо учитывать, рассматривая конкретный вид ЗУ, включает в себя:

- месторасположения;
- емкость;
- единицу пересылки;
- метод доступа;
- быстродействие;
- физический тип;
- физические особенности;
- *с*тоимость.

По месту расположения ЗУ разделяют на процессорные, внутренние и внешние. Наиболее скоростные виды памяти (регистры, кэш-память первого уровня) обычно размещают на общем кристалле с центральным процессором, а регистры общего назначения вообще считаются частью ЦП. Вторую группу (внутреннюю память) образуют ЗУ, расположенные на системной плате. К внутренней памяти относят основную память, а также кэш-память второго и последующих уровней (кэш-память второго уровня может также размещаться на кристалле процессора). Медленные ЗУ большой емкости (магнитные и оптические диски, магнитные ленты) называют внешней памятью, поскольку к ядру ВМ они подключаются аналогично устройствам ввода/вывода.

При оценке быстродействия необходимо учитывать применяемый в данном типе *ЗУ метод доступа* к данным. Различают четыре основных метода доступа:

• **Последовательный доступ.** ЗУ с последовательным доступом ориентировано на хранение информации в виде последовательности блоков данных, называемых записями. Для доступа к нужному элементу (слову или байту) необходимо прочитать все предшествующие ему данные. Время доступа зависит от положения требуемой записи в последовательности записей на носителе информации и позиции элемента внутри данной записи. Примером может служить ЗУ на магнитной ленте.

• **Прямой доступ.** Каждая запись имеет уникальный адрес, отражающий ее физическое размещение на носителе информации. Обращение осуществляется *как* адресный доступ к началу записи, с последующим последовательным доступом к определенной единице информации внутри записи. В результате время доступа к определенной позиции является величиной переменной. Такой режим характерен для магнитных дисков.

• **Произвольный доступ.** Каждая ячейка памяти имеет уникальный физический адрес. Обращение к любой ячейке занимает одно и то же время и может производиться в произвольной очередности. Примером могут служить запоминающие устройства основной памяти.

• **Ассоциативный доступ.** Этот вид доступа позволяет выполнять поиск ячеек, содержащих такую информацию, в которой значение отдельных битов совпадает с состоянием одноименных битов в заданном образце. Сравнение осуществляется параллельно для всех ячеек памяти, независимо от ее емкости. По ассоциативному принципу построены некоторые блоки кэш-памяти.

Иерархия запоминающих устройств

Память часто называют «узким местом» фон-неймановских ВМ из-за ее серьезного отставания по быстродействию от процессоров, причем разрыв этот неуклонно увеличивается. Так, если производительность процессоров возрастает вдвое примерно каждые 1,5 года, то для микросхем памяти прирост быстродействия не превышает 9% в год (удвоение за 10 лет), что выражается в увеличении разрыва в быстродействии между процессором и памятью приблизительно на 50% в год.

Если проанализировать используемые в настоящее время типы ЗУ, выявляется следующая закономерность:

• чем меньше время доступа, тем выше стоимость хранения бита;

• чем больше емкость, тем ниже стоимость хранения бита, но больше время доступа.

При создании системы памяти постоянно приходится решать задачу обеспечения требуемой емкости и высокого быстродействия за приемлемую цену. Наиболее распространенным подходом здесь является построение системы памяти ВМ по иерархическому принципу. *Иерархическая память* состоит из ЗУ различных типов, которые, в зависимости от характеристик, относят к определенному уровню иерархии. Более высокий уровень меньше по емкости, быстрее и имеет большую стоимость в пересчете на бит, чем более низкий уровень. Уровни иерархии взаимосвязаны: все данные на одном уровне могут быть также найдены на более низком уровне, и все данные на этом более низком уровне могут быть найдены на следующем нижележащем уровне и т.д.

Основная память

Основная память (ОП) представляет собой единственный вид памяти, к которой ЦП может обращаться непосредственно (исключение составляют лишь регистры центрального процессора). Информация, хранящаяся на внешних ЗУ, становится доступной процессору только после того, как будет переписана в основную память.

Основную память образуют запоминающие устройства с произвольным доступом. Такие ЗУ образованы как массив ячеек, а «произвольный доступ» означает, что обращение к любой ячейке занимает одно и то же время и может производиться в произвольной последовательности. Каждая ячейка содержит фиксированное число запоминающих элементов и имеет уникальный адрес, позволяющий различать ячейки при обращении к ним для выполнения операций записи и считывания.

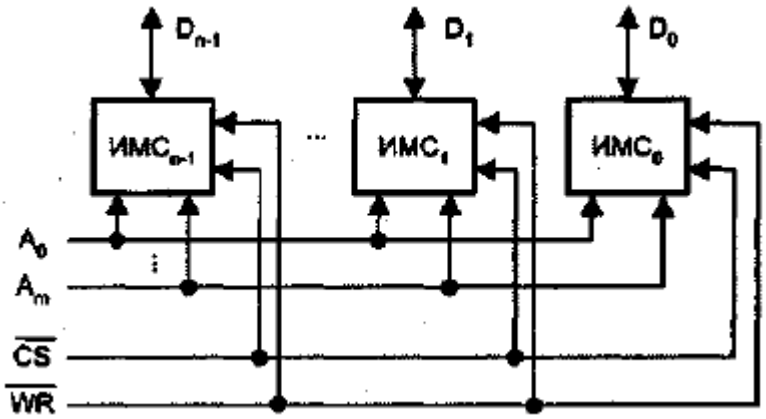
Следствием огромных успехов в области полупроводниковых технологий стало изменение элементной базы основной памяти. На смену ЗУ на базе ферромагнитных

колец пришли полупроводниковые микросхемы, использование которых в наши дни стало повсеместным.

Блочная организация основной памяти

Емкость основной памяти современных ВМ слишком велика, чтобы ее можно было реализовать на базе единственной интегральной микросхемы (ИМС). Необходимость объединения нескольких ИМС ЗУ возникает также, когда разрядность ячеек в микросхеме ЗУ меньше разрядности слов ВМ.

Увеличение разрядности ЗУ реализуется за счет объединения адресных входов объединяемых ИМС ЗУ. Информационные входы и выходы микросхем являются входами и выходами модуля ЗУ увеличенной разрядности. Полученную совокупность микросхем называют *модулем памяти* Модулем можно считать и единственную микросхему, если она уже имеет нужную разрядность. Один или несколько модулей образуют *банк памяти.*



Для получения требуемой емкости ЗУ нужно определенным образом объединить несколько банков памяти меньшей емкости. В общем случае основная память ВМ практически всегда имеет блочную структуру, то есть содержит несколько банков.

Расслоение памяти

Помимо податливости к наращиванию емкости, блочное построение памяти обладает еще одним достоинством – позволяет сократить время доступа к информации. Это возможно благодаря потенциальному параллелизму, присущему блочной организации. Большей скорости доступа можно достичь за счет одновременного доступа ко многим банкам памяти. Одна из используемых для этого методик называется *расслоением памяти.* В ее основе лежит так называемое *чередование адресов* (address interleaving), заключающееся в изменении системы распределения адресов между банками памяти.

Прием чередования адресов базируется на свойстве локальности по обращению, согласно которому последовательный доступ в память обычно производится к ячейкам, имеющим смежные адреса. Иными словами, если в данный момент выполняется обращение к ячейке с адресом 5, то следующее обращение, вероятнее всего, будет к ячейке с адресом 6, затем 7 и т. д. Чередование адресов обеспечивается за счет циклического разбиения адреса.

Поскольку в каждом такте на шине адреса может присутствовать адрес только одной ячейки, параллельное обращение к нескольким банкам невозможно, однако оно может быть организовано со сдвигом на один такт. Адрес ячейки запоминается в индивидуальном регистре адреса, и дальнейшие операции по доступу к ячейке в каждом банке протекают независимо. При большом количестве банков среднее время доступа к ОП сокращается почти в В раз (В – количество банков), но при условии, что ячейки, к которым производится последовательное обращение, относятся к разным банкам. Если же

запросы к одному и тому же банку следуют друг за другом, каждый следующий запрос должен ожидать завершения обслуживания предыдущего. Такая ситуация называется конфликтом по доступу. При частом возникновении конфликтов по доступу метод становится неэффективным.

Организация микросхем памяти

Интегральные микросхемы (ИМС) памяти организованы в виде матрицы ячеек, каждая из которых, в зависимости от разрядности ИМС, состоит из одного или более запоминающих элементов (ЗЭ) и имеет свой адрес. Каждый ЗЭ способен хранить один бит информации. Для ЗЭ любой полупроводниковой памяти характерны следующие свойства:

• два стабильных состояния, представляющие двоичные 0 и 1;

• в ЗЭ (хотя бы однажды) может быть произведена запись информации, посредством перевода его в одно из двух возможных состояний;

• для определения текущего состояния ЗЭ его содержимое может быть считано.

При матричной организации ИМС памяти (рис. 5.6) реализуется координатный принцип адресации ячеек. Адрес ячейки, поступающий по шине адреса ВМ, пропускается через логику выбора, где он разделяется на две составляющие: адрес строки и адрес столбца. Адреса строки и столбца запоминаются соответственно в регистре адреса строки и регистре адреса столбца микросхемы. Регистры соединены каждый со своим дешифратором. Выходы дешифраторов образуют систему горизонтальных и вертикальных линий, к которым подсоединены запоминающие элементы матрицы, при этом каждый ЗЭ расположен на пересечении одной горизонтальной и одной вертикальной линии.

Возможности «ускорения ядра» микросхемы ЗУ весьма ограничены и связаны в основном с миниатюризацией запоминающих элементов. Наибольшие успехи достигнуты в интерфейсной части ИМС, касаются они, главным образом, операции чтения, то есть способов доставки содержимого ячейки на шину данных. Наибольшее распространение получили следующие шесть фундаментальных подходов:

• последовательный;
• конвейерный;
• регистровый;
• страничный;
• пакетный;
• удвоенной скорости.

Последовательный режим

При использовании последовательного режима адрес и управляющие сигналы подаются на микросхему до поступления синхроимпульса.

В момент прихода синхроимпульса вся входная информация запоминается во внутренних регистрах – по его переднему фронту, и начинается цикл чтения. Через некоторое время, но в пределах того же цикла данные появляются на внешней шине, причем момент этот определяется только моментом прихода синхронизирующего импульса и скоростью внутренних цепей микросхемы.

Конвейерный режим

Конвейерный режим (pipelined mode) – это такой метод доступа к данным, при котором можно продолжать операцию чтения по предыдущему адресу в процессе запроса по следующему.

При чтении из памяти время, требуемое для извлечения данных из ячейки, можно условно разбить на два интервала. Первый из них – непосредственно доступ к массиву запоминающих элементов и извлечение данных из ячейки. Второй – передача данных на

выход (при этом происходит детектирование состояния ячейки, усиление сигнала и другие операции, необходимые для считывания информации).

В отличие от последовательного режима, где следующий цикл чтения начинается только по окончании предыдущего, в конвейерном режиме процесс разбивается на два этапа. Пока данные из предыдущего цикла чтения передаются на внешнюю шину, происходит запрос на следующую операцию чтения. Таким образом, два цикла чтения перекрываются во времени. Из-за усложнения схемы передачи данных на внешнюю шину время считывания увеличивается на один такт, и данные поступают на выход только в следующем такте, но такое запаздывание наблюдается лишь при первом чтении в последовательности операций считывания из памяти.

Все последующие данные поступают на выход друг за другом, хотя и с запаздыванием на один такт относительно запроса на чтение. Так как циклы чтения перекрываются, микросхемы с конвейерным режимом могут использоваться при частотах шины, вдвое превышающих допустимую для ИМС с последовательным режимом чтения.

Регистровый режим

Регистровый режим (Register to Latch) используется относительно редко и отличается наличием регистра на выходе микросхемы. Адрес и управляющие сигналы выдаются на шину до поступления синхронизирующего импульса. С приходом положительного фронта синхроимпульса адрес записывается во внутренний регистр микросхемы, и начинается цикл чтения. Считанные данные заносятся в промежуточный выходной регистр и хранятся там до появления отрицательного фронта (спада) синхроимпульса, а с его поступлением передаются на шину. Метод однозначно определяет момент появления данных на выходе ИМС, причем изменяя ширину импульса синхронизации можно менять время появления данных на шине. Данное свойство часто оказывается весьма полезным при проектировании специализированных ВМ. По быстродействию микросхемы с регистровым режимом идентичны ИМС с последовательным режимом.

Страничный режим

В основе идеи лежит тот факт, что при доступе к ячейкам со смежными адресами (согласно принципу локальности такая ситуация наиболее вероятна), причем к таким, где все ЗЭ расположены в одной строке матрицы, доступ ко второй и последующим ячейкам можно производить существенно быстрее. Действительно, если адрес строки при очередном обращении остался прежним, то все временные затраты, связанные с повторным занесением адреса строки в регистр ИМС, дешифровкой, зарядом паразитной емкости горизонтальной линии и т. п., можно исключить. Для доступа к очередной ячейке достаточно подавать на ИМС лишь адрес нового столбца, сопровождая его сигналом CAS. Отметим, что обращение к первой ячейке в последовательности производится стандартным образом – поочередным заданием адреса строки и адреса столбца, то есть здесь время доступа уменьшить практически невозможно. Рассмотренный режим называется режимом страничного доступа или просто страничным режимом (Page Mode). Под страницей понимается строка матрицы ЗЭ. Микросхемы, где реализуется страничный режим и его модификации, принято характеризовать формулой x-y-y-y. Первое число x представляет количество тактов системной шины, необходимое для доступа к первой ячейке последовательности, а y – к каждой из последующих ячеек.

Так, выражение 7-3-3-3 означает, что для обработки первого слова необходимо 7 тактовых периодов системной шины (в течение шести из которых шина простаивает в ожидании), а для обработки последующих слов – по три периода, из которых два системная шина также простаивает.

Режим быстрого страничного доступа

Режим быстрого страничного доступа (FPM – Fast Page Mode) представляет собой модификацию стандартного страничного режима. Основное отличие заключается в способе занесения новой информации в регистр адреса столбца. Полный адрес (строки и столбца) передается только при первом обращении к строке. Активизация буферного регистра адреса столбца производится не по сигналу CAS, а по заднему фронту сигнала RAS. Сигнал RAS остается активным на протяжении всего страничного цикла и позволяет заносить в регистр адреса столбца новую информацию не по спадающему фронту CAS, а как только адрес на входе ИМС стабилизируется, то есть практически по переднему фронту сигнала CAS. В целом же потери времени сокращаются на два такта, которые ранее требовались для передачи адреса каждой строки и сигнала RAS. Реальный выигрыш, однако, наблюдается лишь при передаче блоков данных, хранящихся в одной и той же строке микросхемы. Если же программа часто обращается к разным областям памяти, переходя с одной строки ИМС на другую, преимущества метода теряются.

Режим нашел широкое применение в микросхемах ОЗУ, особенно динамического типа.

Пакетный режим

Пакетный режим (Burst Mode) – режим, при котором на запрос по конкретному адресу память возвращает пакет данных, хранящихся не только по этому адресу, но и по нескольким последующим адресам.

Разрядность ячейки памяти современных ВМ обычно равна одному байту, в то время как ширина шины данных, как правило, составляет четыре байта. Следовательно, одно обращение к памяти требует последовательного доступа к четырем смежным ячейкам – пакету. С учетом этого обстоятельства в ИМС памяти часто используется модификация страничного режима, носящая название группового или пакетного режима. При его реализации адрес столбца заносится в ИМС только для первой ячейки пакета, а переход к очередному столбцу производится уже внутри микросхемы.

Это позволяет для каждого пакета исключить три из четырех операций занесения в ИМС адреса столбца и тем самым еще более сократить среднее время доступа.

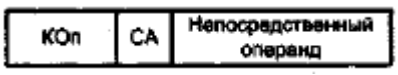
Режим удвоенной скорости

Важным этапом в дальнейшем развитии технологии микросхем памяти стал режим DDR (Double Data Rate) – удвоенная скорость передачи данных. Сущность метода заключается в передаче данных по обоим фронтам импульса синхронизации, то есть дважды за период. Таким образом, пропускная способность увеличивается в те же два раза.

Помимо упомянутых используются и другие приемы повышения быстродействия ИМС памяти, такие как включение в состав микросхемы вспомогательной кэш-памяти и независимые тракты данных, позволяющие одновременно производить обмен с шиной данных и обращение к матрице ЗЭ и т. д.

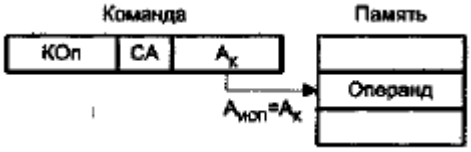
Непосредственная адресация

При *непосредственной адресации* (НА) в адресном поле команды вместо адреса содержится непосредственно сам операнд. Этот способ может применяться при выполнении арифметических операций, операций сравнения, а также для загрузки констант в регистры.



Прямая адресация

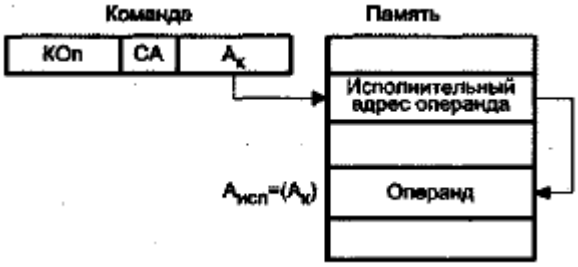
При прямой или абсолютной адресации (ПА) адресный код прямо указывает номер ячейки памяти, к которой производится обращение, то есть адресный код совпадает с исполнительным адресом.



Косвенная адресация

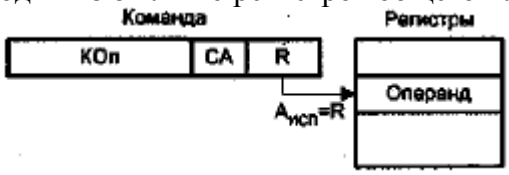
Одним из путей преодоления проблем, свойственных прямой адресации, может служить прием, когда с помощью ограниченного адресного поля команды указывается адрес ячейки, в свою очередь, содержащей полноразрядный адрес операнда. Этот способ известен как *косвенная адресация* (КА). Запись *(Ак)* означает содержимое ячейки, адрес которой указан в скобках.

При косвенной адресации содержимое адресного поля команды остается неизменным, в то время как косвенный адрес в процессе выполнения программы можно изменять. Это позволяет проводить вычисления, когда адреса операндов заранее неизвестны и появляются лишь в процессе решения задачи. Дополнительно такой прием упрощает обработку массивов и списков, а также передачу параметров подпрограммам.



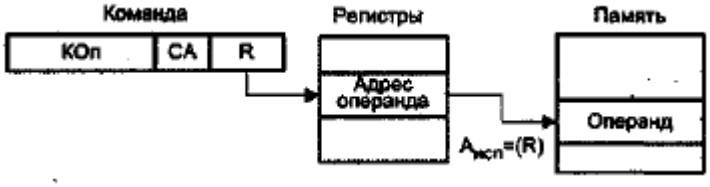
Регистровая адресация

Регистровая адресация (РА) напоминает прямую адресацию. Различие состоит в том, что адресное поле инструкции указывает не на ячейку памяти, а на регистр процессора. Идентификатор регистра в дальнейшем будем обозначать буквой *R*. Обычно размер адресного поля в данном случае составляет три или четыре бита, что позволяет указать соответственно на один из 8 или 16 регистров общего назначения (РОН).



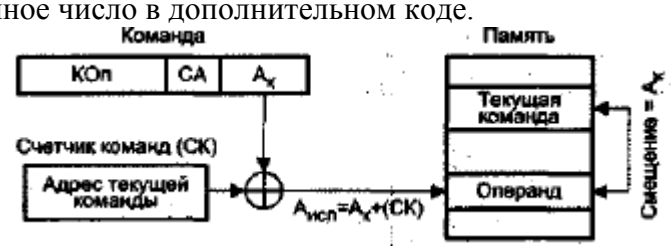
Косвенная регистровая адресация

Косвенная регистровая адресация (КРА) представляет собой косвенную адресацию, где исполнительный адрес операнда хранится не в ячейке основной памяти, а в регистре процессора. Соответственно, адресное поле команды указывает не на ячейку памяти, а на регистр.



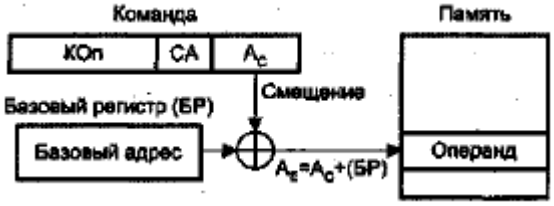
Относительная адресация

При *относительной адресации* (ОА) для получения исполнительного адреса операнда содержимое подполя *Ак* команды складывается с содержимым счетчика команд. Таким образом, адресный код в команде представляет собой смещение относительно адреса текущей команды. Следует отметить, что в момент вычисления исполнительного адреса операнда в счетчике команд может уже быть сформирован адрес следующей команды, что нужно учитывать при выборе величины смещения. Обычно подполе *Ак* трактуется как двоичное число в дополнительном коде.



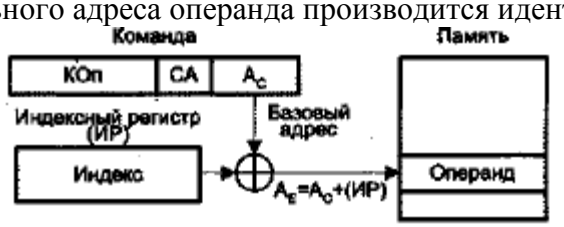
Базовая регистровая адресация

В случае *базовой регистровой адресации* (БРА) регистр, называемый базовым, содержит полноразрядный адрес, а подполе *Ас* – смещение относительно этого адреса. Ссылка на базовый регистр может быть явной или неявной. В некоторых ВМ имеется специальный базовый регистр и его использование является неявным, то есть подполе *R* в команде отсутствует.



Индексная адресация

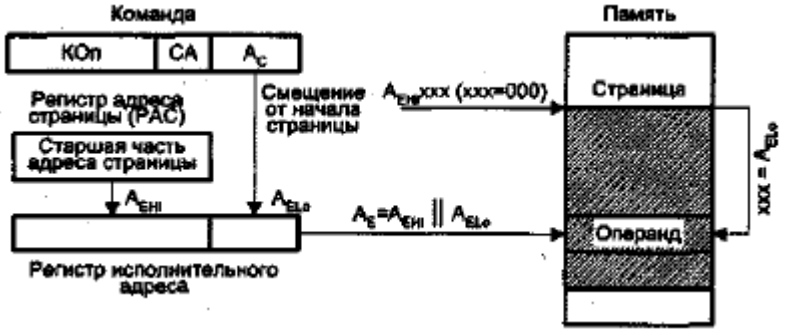
При *индексной адресации* (ИА) подполе *Ас* содержит адрес ячейки памяти, а регистр (указанный явно или неявно) – смещение относительно этого адреса. Как видно, этот способ адресации похож на базовую регистровую адресацию. Поскольку при индексной адресации в поле *Ас* находится полноразрядный адрес ячейки памяти, играющий роль базы, длина этого поля больше, чем при базовой регистровой адресации. Тем не менее вычисление исполнительного адреса операнда производится идентично.



Страничная адресация

Страничная адресация (СТА) предполагает разбиение адресного пространства на страницы. Страница определяется своим начальным адресом, выступающим в качестве базы. Старшая часть этого адреса хранится в специальном регистре – *регистре адреса страницы* (РАС). В адресном коде команды указывается смещение внутри страницы, рассматриваемое как младшая часть исполнительного адреса.

Исполнительный адрес образуется конкатенацией (присоединением) *Ас к* содержимому РАС. На рисунке символ || обозначает операцию конкатенации.



Блочная адресация

Блочная адресация используется в командах, для которых единицей обработки служит блок данных, расположенных в последовательных ячейках памяти. Этот способ очень удобен при работе с внешними запоминающими устройствами и в операциях с векторами. Для описания блока обычно берется адрес ячейки, где хранится первый или последний элемент блока, и общее количество элементов блока, заданное числом байтов или ячеек. Вместо длины блока может использоваться специальный признак «конец блока», помещаемый за последним элементом блока.

Тема 8. Устройства управления

Функции центрального устройства управления

Устройство управления (УУ) вычислительной машины реализует функции управления ходом вычислительного процесса, обеспечивая автоматическое выполнение команд программы. Процесс выполнения программы в ВМ представляет собой последовательность машинных циклов. Детализируем основные целевые функции, реализуемые устройством управления в ходе типового машинного цикла. Для простоты примем, что ВМ обеспечивает одноадресную систему команд. При этом, в частности, полагается, что до начала выполнения двухоперандной арифметической команды второй операнд уже находится в процессоре.

Первым этапом в машинном цикле является выборка команды из памяти (этап ВК). Целевую функцию этого этапа будем обозначать как ЦФ-ВК.

За выборкой команды следует этап декодирования ее операционной части (кода операции). Для простоты пока будем рассматривать этот этап в качестве составной части этапа ВК.

Вторая целевая функция – формирование адреса следующей команды. На это выделяется специальный такт работы – этап ФАСК, которому соответствует целевая функция ЦФ-ФАСК.

Далее следует этап формирования исполнительного адреса операнда или адреса перехода (этап ФИА), на котором УУ реализует функцию ЦФ-ФИА. Функция имеет столько модификаций, сколько способов адресации (СА) предусмотрено в системе команд ВМ.

На четвертом этапе реализуется целевая функция выборки операнда (ЦФ-ВО) из памяти по исполнительному адресу, сформированному на предыдущем этапе.

Наконец, на последнем этапе машинного цикла действия задаются целевой функцией исполнения операции – ЦФ-ИО. Очевидно, что количество модификаций ЦФ-ИО равно количеству операций, имеющихся в системе команд ВМ.

Порядок следования целевых функций полностью определяет динамику работы устройства управления и всей ВМ в целом. Этот порядок удобно задавать и отображать в виде граф-схемы этапов исполнения команды (ГСЭ). Как и граф-схема микропрограммы, ГСЭ содержит начальную, конечную, операторные и условные вершины. В начальной и

конечной вершинах проставляется условное обозначение конкретной команды, а в условной вершине записывается логическое условие, влияющее на порядок следования этапов. В операторные вершины вписываются операторы этапов.

По форме записи оператор этапа – это оператор присваивания, в котором:

- слева от знака присваивания указывается наименование результата действий, выполненных на этапе;
- справа от знака присваивания записывается идентификатор целевой функции, определяющей текущие действия, а за ним (в скобках) приводится список исходных данных этапа.

Исходной информацией для первого этапа служит хранящийся в счетчике команд адрес A_{Ki} текущей команды Ki. Процесс выборки команды отображается оператором первого этапа: Ki := ВК(A_{Ki}).

Адрес A_{Ki} обеспечивает также второй этап, результатом которого является адрес следующей команды A_{Ki+1}, поэтому оператор второго этапа имеет вид: A_{Ki+1} := ФАСК(A_{Ki}).

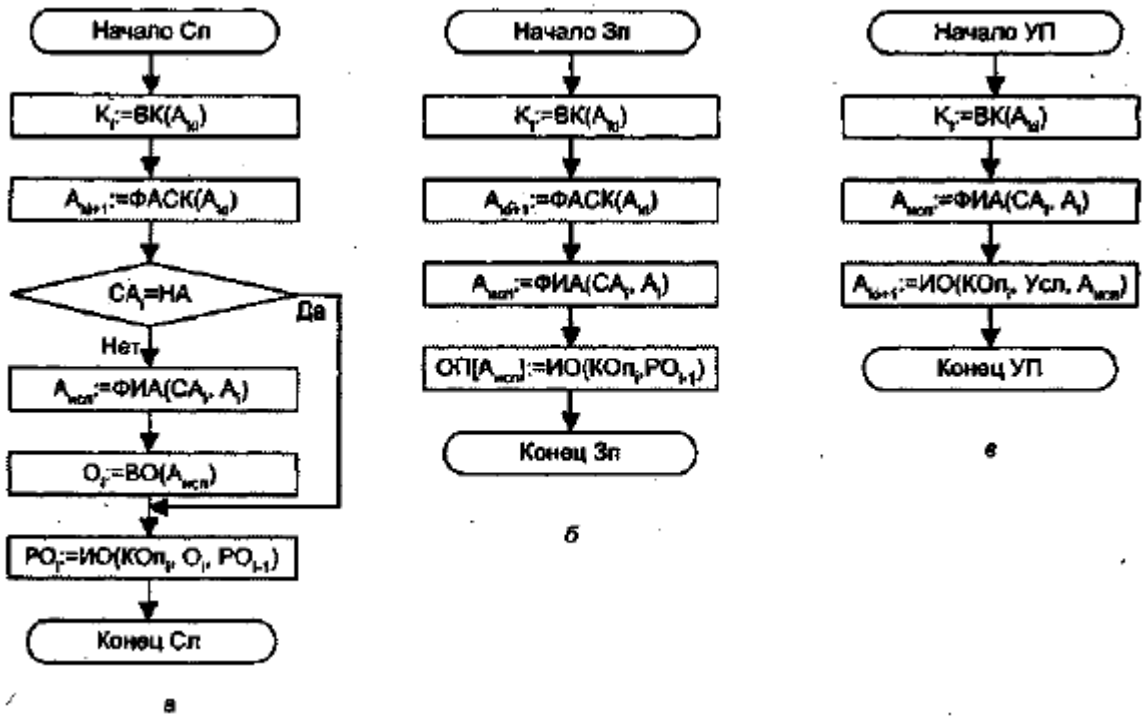
В качестве исходных данных для третьего этапа машинного цикла выступают содержащиеся в коде текущей команды способ адресации CA_i, (он определяет конкретную модификацию ЦФ-ФИАО) и код адресной части А. Результатом становится исполнительный адрес операнда $A_{исп}$:= ФИА(C_{Ai}, A_i).

Полученный адрес используется на четвертом этапе для выборки операнда O_i := ВО($A_{исп}$).

Результат исполнения операции $P0_i$, получаемый на пятом этапе машинного цикла, зависит от кода операции i-й команды $КОп_i$ (определяет модификацию ЦФ-ИО), кода первого операнда 0_i и кода второго операнда – результата предыдущей (i-1)-й операции $P0_{i-1}$: $P0_i$:= ИО($КОп_i$, 0_i, $P0_{i-1}$).

В соответствии со структурой граф-схемы этапов все команды ВМ можно разделить на три типа:

- команды типа «Сложение» (Сл);
- команды типа «Запись» (Зп);
- команды типа «Условный переход» (УП).



Видно, что количество этапов в командах типа «Сл» колеблется от трех (для непосредственной адресации НА) до пяти. При непосредственной адресации второй операнд записан в адресной части команды, поэтому нет необходимости в реализации

устройством управления целевых функций ЦФ-ФИА, ЦФ-ВО. Количество этапов для команд типа «Зп» постоянно и равно четырем – здесь отсутствует необходимость в ЦФ-ВО. Машинный цикл команд типа «УП» состоит из трех этапов, поскольку здесь, помимо выборки операнда, можно исключить и этап ФАСК – действия, обычно выполняемые на этом этапе, фактически реализуются на этапе ИО.

Оператор этапа исполнения операции для команд «Зп» имеет смысл записи результата предыдущей операции $Р0_{i-1}$ в ячейку с адресом $А_{исп}$: ОП[$А_{исп}$]:=ИО($КОп_i$, $РО_{i-1}$).

Местоположение $Р0_{i-1}$ определяется кодом операции $КОп_i$. Оператор этапа ИО для команд «УП» обеспечивает формирование адреса следующей (i+1)-й команды в зависимости от $А_{исп}$ и значения проверяемого условия перехода Усл: $А_{Ki+1}$:= ИО($КОп_i$, Усл, $А_{исп}$).

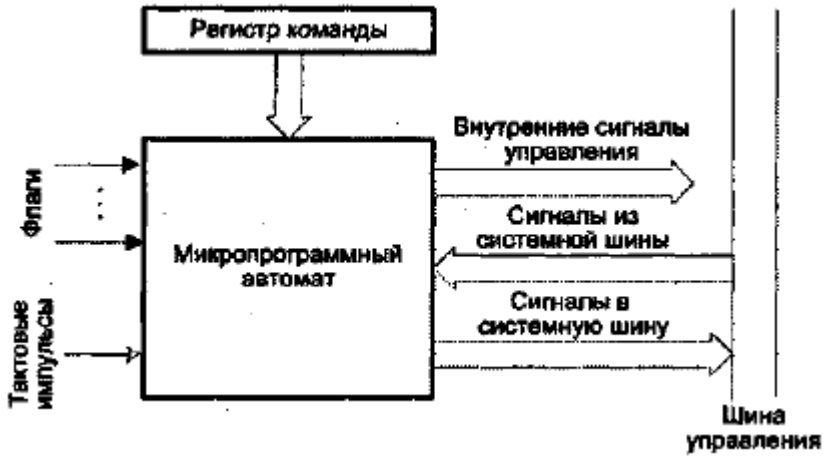
Местоположение проверяемого условия также определяется кодом операции $К0п_i$.

Модель устройства управления

Для выполнения своих функций УУ должно иметь входы, позволяющие определить состояние управляемой системы, и выходы, через которые реализуется управление поведением системы.

Входной информацией для устройства управления служат:

- тактовые импульсы – с каждым тактовым импульсом УУ инициирует выполнение одной или нескольких микроопераций;
- код операции – код операции текущей команды поступает из регистра команды и используется, чтобы определить, какие микрооперации должны выполняться в течение машинного цикла;
- флаги – требуются устройству управления для оценки состояния ЦП и результата предшествующей операции, что необходимо при выполнении команд условного перехода;
- сигналы из системной шины – часть сигналов с системной шины, обеспечивающая передачу в УУ запросов прерывания, подтверждений и т.п.

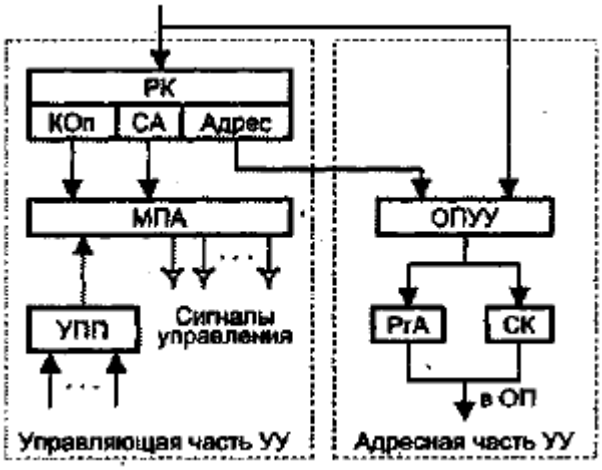


В свою очередь, УУ, а точнее микропрограммный автомат, формирует следующую выходную информацию:

- внутренние сигналы управления – эти сигналы воздействуют на внутренние схемы центрального процессора и относятся к одному из двух типов: тем, которые вызывают перемещение данных из регистра в регистр, и тем, что инициируют определенные функции операционного устройства ВМ;
- сигналы в системную шину – также относятся к одному из двух типов: управляющие сигналы в память и управляющие сигналы в модули ввода/вывода.

Структура устройства управления

Как уже отмечалось ранее, процесс функционирования ВМ состоит из последовательности элементарных действий в ее узлах. Такие элементарные преобразования информации, выполняемые в течение одного такта сигналов синхронизации, называются микрооперациями (МО). Совокупность сигналов управления, вызывающих одновременно выполняемые микрооперации, образует микрокоманду (МК). В свою очередь, последовательность микрокоманд, определяющую содержание и порядок реализации машинного цикла, принято называть микропрограммой. Сигналы управления вырабатываются устройством управления, а точнее одним из его узлов – микропрограммным автоматом (МПА). Название отражает то, что МПА определяет микропрограмму как последовательность выполнения микроопераций.



Микропрограммы реализации перечисленных ранее целевых функций инициируются задающим оборудованием, которое вырабатывает требуемую последовательность сигналов управления и входит в состав управляющей части УУ.

Выполняются микропрограммы исполнительным оборудованием, входящим в состав основной памяти (для ЦФ-ВК и ЦФ-ВО) и операционного устройства (для ЦФ-ИО). Исполнительным оборудованием для целевых функций ЦФ-ФАСК, ЦФ-ФИА служит адресная часть устройства управления. В обобщенной структуре УУ можно выделить две части: управляющую и адресную.

Управляющая часть УУ предназначена для координирования работы операционного блока ВМ, адресной части устройства управления, основной памяти и других узлов ВМ.

Адресная часть УУ обеспечивает формирование адресов команд и исполнительных адресов операндов в основной памяти.

В состав управляющей части УУ входят:

- регистр команды (РК), состоящий из адресной (Адрес) и операционной (КОп, СА) частей;
- микропрограммный автомат (МПА);
- узел прерываний и приоритетов (УПП).

Регистр команды РК предназначен для приема очередной команды из запоминающего устройства. Микропрограммный автомат на основании результатов расшифровки операционной части команды (КОп, СА) вырабатывает определенную последовательность микрокоманд, вызывающих выполнение всех целевых функций УУ.

В зависимости от способа формирования микрокоманд различают микропрограммные автоматы:

- с жесткой или аппаратной логикой;
- с программируемой логикой.

Узел прерываний и приоритетов позволяет реагировать на различные ситуации, связанные как с выполнением рабочих программ, так и с состоянием ВМ.

Адресная часть УУ включает в себя:

- операционный узел устройства управления (ОПУУ);
- регистр адреса (РгА);
- счетчик команд (СК).

Регистр адреса используется для хранения исполнительных адресов операндов, а счетчик команд – для выработки и хранения адресов команд. Содержимое РгА и СК посылается в регистр адреса основной памяти (ОП) для выборки операндов и команд соответственно.

ОПУУ, называемый иначе узлом индексной арифметики или узлом адресной арифметики, обрабатывает адресные части команд, формируя исполнительные адреса операндов, а также подготавливает адрес следующей команды при выполнении команд перехода. Состав ОПУУ может быть аналогичен составу основного операционного устройства ВМ (иногда в простейших ВМ с целью экономии затрат на оборудование ОПУУ совмещается с основным операционным устройством).

В состав УУ могут также входить дополнительные узлы, в частности узел организации прямого доступа к памяти. Этот узел обычно реализуется в виде самостоятельного устройства – контроллера прямого доступа к памяти (КПДП). КПДП обеспечивает совмещение во времени работы операционного устройства с процессом обмена информацией между ОП и другими устройствами ВМ, тем самым повышая общую производительность машины.

Довольно часто регистры различных узлов УУ объединяют в отдельный узел управляющих (специальных) регистров устройства управления.

Все множество технологий, используемых при реализации микропрограммных автоматов устройств управления, можно свести к двум категориям:

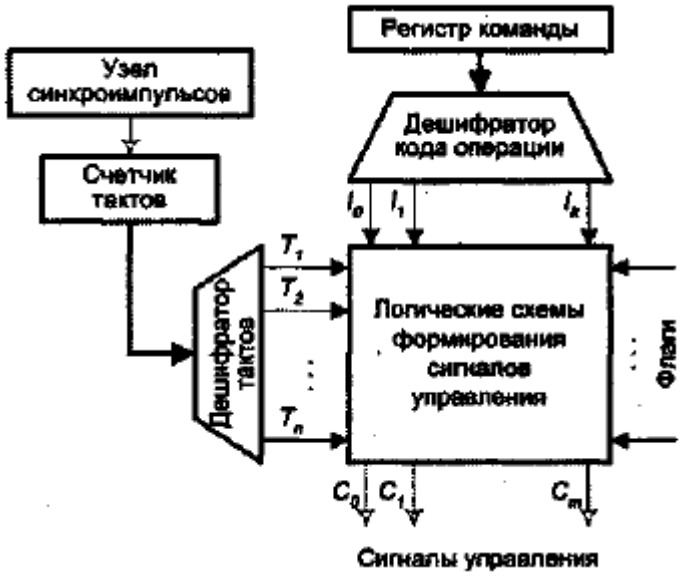
- МПА с «жесткой» логикой или аппаратной реализацией;
- МПА с программируемой логикой.

Микропрограммный автомат с жесткой логикой

Обычно тип микропрограммного автомата (МПА), формирующего сигналы управления, определяет название всего УУ. Так, УУ с жесткой логикой управления имеет в своем составе МПА с жесткой (аппаратной) логикой. При создании такого МПА выходные сигналы управления реализуются за счет однажды соединенных между собой логических схем.

Типичная структура микропрограммного автомата с жесткой логикой управления показана на рисунке.

Исходной информацией для УУ служат: содержимое регистра команды, флаги, тактовые импульсы и сигналы, поступающие с шины управления.



Код операции, хранящийся в РК, используется для определения того, какие СУ и в какой последовательности должны формироваться, при этом, с целью упрощения логики управления, желательно иметь в УУ отдельный логический сигнал для каждого кода операции (I_0, I_v, ..., I_k). Это может быть реализовано с помощью дешифратора. Дешифратор кода операции преобразует код j-й операции, поступающей из регистра команды (РК), в единичный сигнал на j-м выходе.

Машинный цикл выполнения любой команды состоит из нескольких тактов. Сигналы управления, по которым выполняется каждая микрооперация, должны вырабатываться в строго определенные моменты времени, поэтому все СУ «привязаны» к импульсам синхронизации (СИ), формируемым узлом синхроимпульсов. Период СИ должен быть достаточным для того, чтобы сигналы успели распространиться по трактам Данных и другим цепям. Каждый СУ ассоциируется с одним из тактовых периодов в рамках машинного цикла. Формирование сигналов, отмечающих начало очередного тактового периода, возлагается на синхронизатор. Синхронизатор содержит счетчик тактов, осуществляющий подсчет СИ. Узел синхроимпульсов после завершения очередного такта работы добавляет к содержимому счетчика тактов единицу. К выходам счетчика подключен дешифратор тактов, с которого и снимаются сигналы тактовых периодов: T_1, ..., T_n. В i-м состоянии счетчика тактов, то есть во время i-го такта, дешифратор тактов вырабатывает единичный сигнал на своем i-м выходе. При такой организации в УУ должна быть предусмотрена обратная связь, с помощью которой по окончании цикла команды счетчик тактов опять устанавливается в состояние Г.

Дополнительным фактором, влияющим на последовательность формирования СУ, являются состояние осведомительных сигналов (флагов), отражающих ход вычислений, и сигналы с шины управления. Эта информация также поступает на вход УУ, причем каждая линия здесь рассматривается независимо от остальных.

Процесс синтеза схемы МПА с жесткой логикой называется структурным синтезом и разделяется на следующие этапы:

- выбор типа логических и запоминающих элементов;
- кодирование состояний автомата;
- синтез комбинационной схемы, формирующей выходные сигналы.

Микропрограммный автомат с программируемой логикой

Принципиально иной подход, позволяющий преодолеть сложность УУ с жесткой логикой, был предложен британским ученым М. Уилксом в начале 50-х годов. В основе идеи лежит тот факт, что для инициирования любой микрооперации достаточно сформировать соответствующий СУ на соответствующей линии управления, то есть перевести такую линию в активное состояние. Для указания микроопераций, выполняемых в данном такте, можно сформировать управляющее слово, в котором каждый бит соответствует одной управляющей линии. Такое управляющее слово называют микрокомандой (МК). Таким образом, микрокоманда может быть представлена управляющим словом со своей комбинацией нулей и единиц. Последовательность микрокоманд, реализующих определенный этап машинного цикла, образует микропрограмму. В терминологии на английском языке микропрограмму часто называют firmware, подчеркивая тот факт, что это нечто среднее между аппаратурой (hardware) и программным обеспечением (software). Микропрограммы для каждой команды ВМ и для каждого этапа цикла команды размещаются в специальном ЗУ, называемом памятью микропрограмм (ПМК). Процесс формирования СУ можно реализовать, последовательно (с каждым тактовым импульсом) извлекая микрокоманды микропрограммы из памяти и интерпретируя содержащуюся в них информацию о сигналах управления.

Идея заинтересовала многих конструкторов ВМ, но была нереализуема, поскольку требовала использования быстрой памяти относительно большой емкости. Вновь вернулись к ней в 1964 году, в ходе создания системы IBM 360. С тех пор устройства

управления с программируемой логикой стали чрезвычайно популярными и были встроены во многие ВМ. В этой связи следует упомянуть запатентованный академиком В. М. Глушковым принцип ступенчатого микропрограммирования.

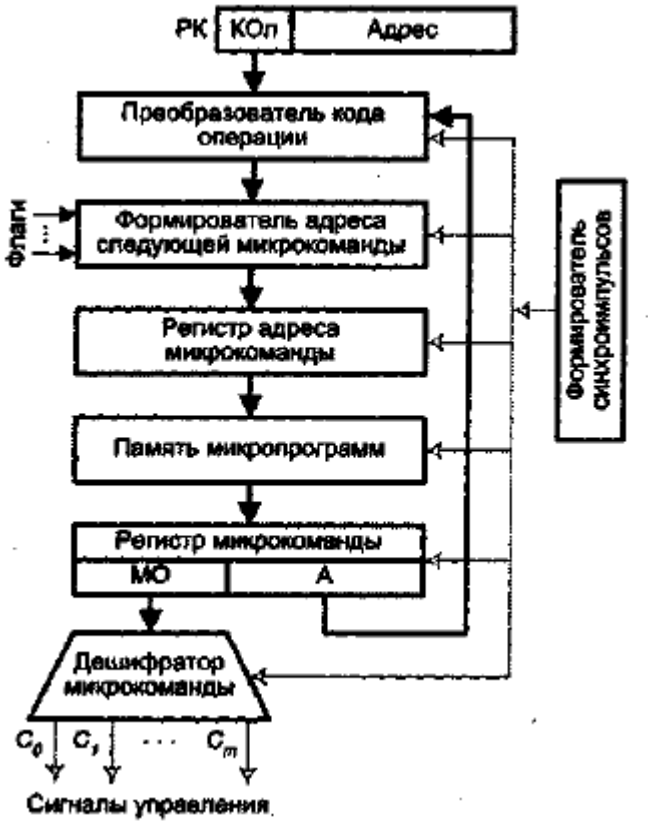
Принцип управления по хранимой в памяти микропрограмме

Отличительной особенностью микропрограммного автомата с программируемой логикой является хранение микрокоманд в виде кодов в специализированном запоминающем устройстве – памяти микропрограмм. Каждой команде ВМ в этом ЗУ в явной форме соответствует микропрограмма, поэтому часто устройства управления, в состав которых входит микропрограммный автомат с программируемой логикой, называют микропрограммными.

Типичная структура микропрограммного автомата представлена на рисунке.

В составе узла присутствуют: память микропрограмм (ПМП), регистр адреса микрокоманды (РАМ), регистр микрокоманды (РМК), дешифратор микрокоманд (ДшМК), преобразователь кода операции, формирователь адреса следующей микрокоманды (ФАСМ), формирователь синхроимпульсов (ФСИ).

Запуск микропрограммы выполнения операции осуществляется путем передачи кода операции из РК на вход преобразователя, в котором код операции преобразуется в начальный (первый) адрес микропрограммы А. Этот адрес поступает через ФАСМ в регистр адреса микрокоманды. Выбранная по адресу Ап из ПМП микрокоманда заносится в РМК. Каждая микрокоманда в общем случае содержит микрооперационную (МО) и адресную (А) части. Микрооперационная часть микрокоманды поступает на дешифратор микрокоманды, на выходе которого образуются управляющие сигналы Ci, инициирующие выполнение микроопераций в исполнительных устройствах и узлах ВМ. Адресная часть микрокоманды подается в ФАСМ, где формируется адрес следующей микрокоманды Амк. Этот адрес может зависеть от адреса на выходе преобразователя кода операции Аи, адресной части текущей микрокоманды А и значений осведомительных сигналов (флагов) X, поступающих от исполнительных устройств. Сформированный адрес микрокоманды снова записывается в РАМ, и процесс повторяется до окончания микропрограммы.



Тема 9. Операционные устройства вычислительных машин

Операционные устройства вычислительных машин

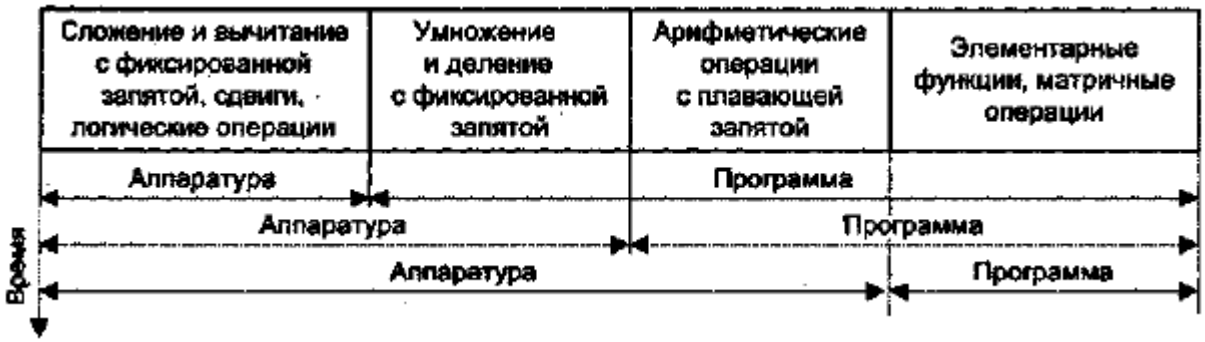
В классической фон-неймановской ВМ функция арифметической и логической обработки данных возлагается на арифметико-логическое устройство (АЛУ). Учитывая разнообразие выполняемых операций и типов обрабатываемых данных, реально можно говорить не о едином устройстве, а о комплексе специализированных операционных устройств (ОУ), каждое из которых реализует определенное подмножество арифметических или логических операций, предусмотренных системой команд ВМ. С этих позиций следует выделить:

- ОУ целочисленной арифметики;
- ОУ для реализации логических операций;
- ОУ десятичной арифметики;
- ОУ для чисел с плавающей запятой.

На практике две первых группы обычно объединяются в рамках одного операционного устройства. Специализированные ОУ десятичной арифметики в современных ВМ встречаются достаточно редко, поскольку обработку чисел, представленных в двоично-десятичной форме, можно достаточно эффективно организовать на базе средств целочисленной двоичной арифметики. Таким образом, будем считать, что АЛУ образуют два вида операционных устройств: целочисленное ОУ и ОУ для обработки чисел в формате с плавающей запятой (ПЗ).

В минимальном варианте АЛУ должно содержать аппаратуру для реализации лишь основных логических операций, сдвигов, а также сложения и вычитания чисел в форме с фиксированной запятой (ФЗ). Опираясь на этот набор, можно программным способом обеспечить выполнение остальных арифметических и логических операций как для чисел с фиксированной запятой, так и для других форм представления информации. Следует, однако, учитывать, что подобный вариант не позволяет добиться высокой скорости вычислений, поэтому по мере расширения технологических возможностей доля аппаратных средств в составе АЛУ постоянно возрастает.

На рисунке показана динамика изменения соотношения между аппаратной и программной реализациями функций АЛУ по мере развития элементной базы вычислительной техники.



Структуры операционных устройств

Набор элементов, на основе которых строятся структуры различных ОУ, называется структурным базисом. Структурный базис ОУ включает в себя:

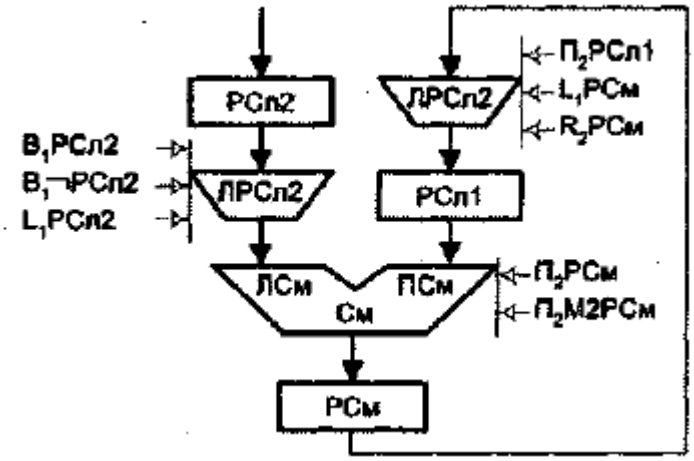
- регистры, обеспечивающие кратковременное хранение слов данных;
- управляемые шины, предназначенные для передачи слов данных;
- комбинационные схемы, реализующие вычисление функций микроопераций и логических условий по управляющим сигналам от устройства управления.

Можно синтезировать ОУ с так называемой канонической структурой, являющейся основополагающей для синтеза других структур. Такая структура образуется путем

замены каждого элемента реализуемой функции соответствующим элементом структурного базиса. Каноническая структура имеет максимальную производительность по сравнению с другими вариантами структур, однако по затратам оборудования является избыточной. С практических позиций больший интерес представляют два иных вида структур ОУ: жесткая и магистральная.

Операционные устройства с жесткой структурой

В ОУ с жесткой структурой комбинационные схемы жестко распределены между всеми регистрами. К каждому регистру относится свой набор комбинационных схем, позволяющих реализовать определенные микрооперации. Пример ОУ с жесткой структурой, обеспечивающего выполнение операций типа «сложение», приведен на рисунке.



В состав ОУ входят три регистра со своими логическими схемами:

- регистр первого слагаемого РСл1 и схема ЛРСл1;
- регистр второго слагаемого РСл2 и схема ЛРСл2;
- регистр суммы РСм и схема комбинационного сумматора См.

Логическая схема ЛРСл2 реализует микрооперации передачи второго слагаемого из РСл2 на левый вход сумматора:

- прямым кодом ЛСм := РСл2 (по сигналу управления B_1РСл2);
- инверсным кодом ЛСм := ¬РСл2 (по сигналу управления B_1¬РСл2);
- со сдвигом на один разряд влево ЛСм := L_1(РСл2·0) (по сигналу управления L_1РСл2).

Логическая схема РСл1 обеспечивает передачу результата из регистра РСм в регистр РСл1:

- прямым кодом РСл1 := РСм (по сигналу управления $П_2$РСл1);
- со сдвигом на один разряд влево РСл1 := L_1(ЛСм·0) (по сигналу управления L_1РСм);
- со сдвигом на два разряда вправо РСл1 := R_2(s·s·ЛСм) (по сигналу управления R_2РСм).

Комбинационный сумматор См предназначен для суммирования (обычного или по модулю 2) операндов, поступивших на его левый (ЛСм) и правый (ПСм) входы. Результат суммирования заносится в регистр РСм := ЛСм + ПСм (по сигналу управления $П_2$РСм) или РСм := ЛСм + ПСм (по сигналу управления $П_2$М2РСм).

Аппаратные затраты на ОУ с жесткой структурой $C_ж$ можно оценить по выражению

$$C_{ж} = nC_T N + 3\sum_{i=1}^{N} n_i \sum_{j=1}^{K} k_{ij} + 3\sum_{i=1}^{N} n_i \sum_{j=1}^{K} C_j k_{ij},$$

где N – количество внутренних слов ОУ; n_1,..., n_N – длины слов; $n = (n_1 + ... + n_N)/N$ – средняя длина слова; k – количество микроопераций типа j =1,2,...,К (сложение, сдвиг,

передача и т. п.), используемых для вычислений слов с номерами i = 1,2,...,N; C_T – цена триггера; C_j – цена одноразрядной схемы для реализации микрооперации j-го типа.

В приведенном выражении первое слагаемое определяет затраты на хранение n-разрядных слов, второе – на связи регистров с комбинационными схемами, а третье – суммарную стоимость комбинационных схем, реализующих микрооперации К типов над N словами.

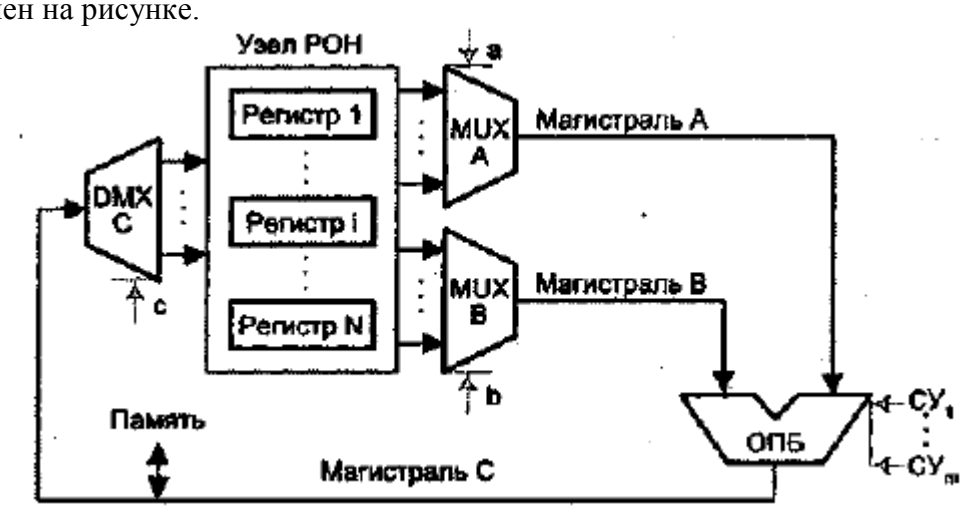
Затраты времени на выполнение операций типа «сложение» в ОУ с жесткой структурой равны $t_Ж = t_B + t_c + t_n$, где t_B – длительность микрооперации выдачи операндов из регистров; t_c – продолжительность микрооперации «сложение»; t_n – длительность микрооперации приема результата в регистр.

Достоинством ОУ с жесткой структурой является высокое быстродействие, недостатком – малая регулярность структуры, что затрудняет реализацию таких ОУ в виде больших интегральных схем.

Операционные устройства с магистральной структурой

В ОУ с магистральной структурой все внутренние регистры объединены в отдельный узел регистров общего назначения (РОН), а все комбинационные схемы – в операционный блок (ОПБ), который зачастую ассоциируют с термином «арифметико-логическое устройство». В операционных устройствах для обработки чисел с плавающей запятой вместо РОН часто используется отдельный узел регистров с плавающей запятой.

Операционный блок и узел регистров сообщаются между собой с помощью магистралей – отсюда и название «магистральное ОУ». Пример магистрального ОУ представлен на рисунке.



В состав узла РОН здесь входят N регистров общего назначения, подключаемых к магистралям A и B через мультиплексоры MUX A и MUX B. Каждый из мультиплексоров является управляемым коммутатором, соединяющим выход одного из РОН с соответствующей магистралью. Номер подключаемого регистра определяется адресом a или b, подаваемым на адресные входы мультиплексора из устройства управления.

По магистралям A и B операнды поступают на входы операционного блока, к которым подключается комбинационная схема, реализующая требуемую микрооперацию (по сигналу управления из УУ). Таким образом, любая микрооперация ОПБ может быть выполнена над содержимым любых регистров ОУ. Результат микрооперации по магистрали C заносится через демультиплексор DMX C в конкретный регистр узла РОН. Демультиплексор представляет собой управляемый коммутатор, имеющий один информационный вход и N информационных выходов. Вход подключается к выходу с заданным адресом c, который поступает на адресные входы DMX C из УУ.

Моделью ОУ с магистральной структурой является М-автомат. М-автоматом называется модель ОУ, построенная на основе принципа объединения комбинационных схем и реализующая в каждом такте только одну микрооперацию.

Выражение для оценки аппаратных затрат на магистральное ОУ можно записать в следующем виде:

$$C_м = nC_T N + 3n(N + K) + n\sum_{j=1}^{K} C_j ,$$

где первое слагаемое определяет затраты на N регистров, второе – затраты на связи узла РОН и ОПБ, а третье – суммарную цену ОПБ.

Из сопоставления выражений для затрат следует, что магистральная структура экономичнее жесткой структуры, если

$$3(N + K - M) < 3\sum_{i=1}^{N}\sum_{j=1}^{K} C_j k_{ij} - \sum_{j=1}^{K} C_j ,$$

где $M = \sum_{i=1}^{N}\sum_{j=1}^{K} k_{ij}$ – количество микроопераций, реализуемых ОУ с жесткой структурой.

С учетом последнего неравенства можно сформулировать следующее сильное условие экономичности магистральных структур: M > N + K.

Затраты времени на сложение в магистральных ОУ больше, чем в ОУ с жесткой структурой:

$t_M = t_B + t_c + t_n + t_{MUX} + t_{DMX} = t_Ж + t_{MUX} + t_{DMX}$,

где t_{MUX} – задержка на подключение операндов из РОН к ОПБ; t_{DMX} – задержка на подключение результата к РОН.

Основным достоинством магистральных ОУ является высокая универсальность и регулярность структуры, что облегчает их реализацию на кристаллах ИС. Вообще говоря, магистральная структура ОУ в современных ВМ является превалирующей.

Классификация операционных устройств с магистральной структурой

Магистральные ОУ классифицируют по виду и количеству магистралей, организации узла РОН, типу ОПБ.

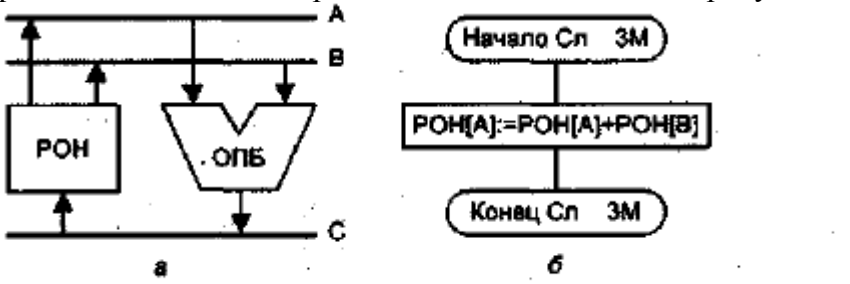
Магистрали ОУ могут быть однонаправленными и двунаправленными, соответственно обеспечивающими передачу данных в одном или двух различных направлениях. Типичным режимом работы магистрали является разделение времени, при котором магистраль используется для передачи функционально разнотипных данных в различные моменты времени.

По функциональному назначению выделяют:

- магистрали внешних связей, соединяющих ОУ с памятью и каналами ввода/вывода ВМ;
- внутренние магистрали ОУ, отвечающие за связь между узлом РОН и операционным блоком.

Количество магистралей внешних связей зависит от архитектуры конкретной ВМ и обычно не превышает двух для внешних связей и трех – для внутренних.

Структура трехмагистрального ОУ представлена на рисунке а, а соответствующая ему микропрограмма выполнения операции типа «сложение» – на рисунке б.



Трехмагистральное ОПУ: а — структура; *б* — микропрограмма сложения

Данный вариант характеризуется наибольшим быстродействием: выборка операндов из РОН, выполнение микрооперации суммирования и запись результата в РОН – все эти действия производятся за один такт. Основной недостаток трехмагистральной организации – большая площадь, занимаемая магистралями на кристалле БИС (от 0,16 до 0,22 от площади кристалла).

Двухмагистральная организация при меньшей площади, покрываемой магистралями (от 0,06 до 0,19 от площади кристалла), требует введения как минимум одного буферного регистра (БР), предназначенного для временного хранения одного из операндов, при этом операция сложения будет выполняться уже за два такта:

- Такт 1: загрузка БР одним из операндов.
- Такт 2: выполнение микрооперации в ОПБ над содержимым БР и одного из РОН; запись результата в РОН.

Наконец, организация ОПУ на основе только одной магистрали минимизирует расходы площади (от 0,03 до 0,09 от площади кристалла).

В одномагистральном ОПУ, вместе с тем, возникает необходимость введения не менее двух буферных регистров БР1, БР2, и длительность операции возрастает до трех Тактов:

- Такт 1: загрузка БР1 одним из операндов.
- Такт 2: загрузка БР2 вторым операндом.
- Такт 3: выполнение микрооперации в ОПБ над содержимым БР1 и БР2; запись результата в один из РОН.

Организация узла РОН магистрального операционного устройства

Количество регистров в узле РОН магистрального ОУ обычно превышает тот минимум, который необходим для реализации универсальной системы операций.

Избыток регистров используется:

- для хранения составных частей адреса (индекса, базы);
- в качестве буферной, сверхоперативной памяти для повышения производительности ВМ за счет уменьшения требуемых пересылок между основной памятью и ОУ.

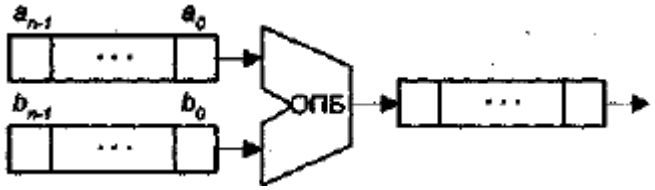
Количество регистров колеблется в среднем от 8 до 16, иногда может достигать 32-64. В процессорах с сокращенным набором команд количество РОН доходит до нескольких сотен.

Организация узла РОН может обеспечивать одноканальный или двухканальный доступ как по входу (записи), так и по выходу (считыванию). В первом случае к входу узла подключается один демультиплексор, а к выходу – один мультиплексор. Во втором случае доступ осуществляется с помощью двух демультиплексоров и (или) двух мультиплексоров. Двухканальный доступ повышает быстродействие ОУ, так как позволяет обратиться параллельно к двум регистрам.

Организация операционного блока магистрального операционного устройства

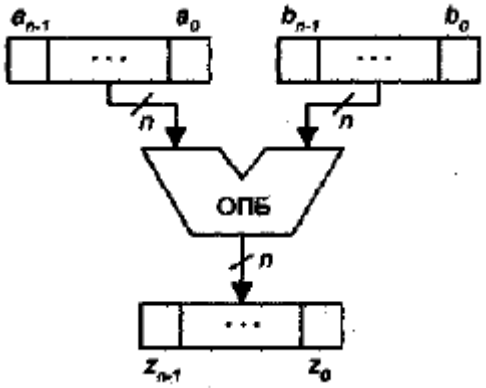
Тип операционного блока (ОПБ) определяется способом обработки данных. Различают ОПБ последовательного и параллельного типа.

В последовательном операционном блоке операции выполняются побитово, разряд за разрядом.

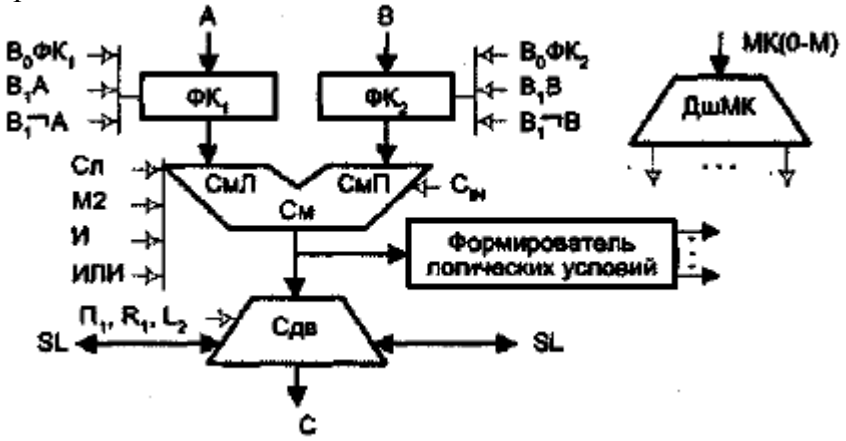


Бит переноса, возникающий при обработке i-го разряда операндов, подается на вход ОПБ и учитывается при обработке (i+l)-го разряда операндов. Результат побитово заносится в выходной регистр, предыдущее содержимое которого перед этим сдвигается на один разряд. Таким образом, после п циклов в выходном регистре формируется слово результата, где каждый разряд занимает предназначенную для него позицию.

При параллельной организации операционного блока все разряды операндов обрабатываются одновременно. Внутренние переносы обеспечиваются схемой ОПБ.



Реализация эффективной системы переносов в рамках «длинного» слова сопряжена с определенными аппаратурными издержками, поэтому на практике часто используют параллельно-последовательную схему ОПБ. В ней слово разбивается на группы по 2, 4 или 8 разрядов, обработка всех разрядов внутри группы осуществляется параллельно, а сами группы обрабатываются последовательно.



Обобщенная схема ОПБ приведена на рисунке. В нее входят: дешифратор микрокоманды ДшМК, формирователи кодов ФК1 и ФК2, многофункциональный сумматор См, сдвигатель Сдв и формирователь логических условий (ЛУ).

Дешифратор микрокоманды вырабатывает внутренние сигналы управления для элементов ОПБ. Он введен в схему с целью минимизации количества связей, требуемых для передачи сигналов управления из УУ.

Формирователи кодов ФК1 и ФК2 служат для формирования прямых и инверсных кодов операндов, поступающих по магистралям А и В. Они реализуют следующий набор микроопераций:

$B_0ФК_1$: СмЛ := 0; $B_0ФК_2$: СмП := 0;
B_1A: СмЛ := A; B_1B: СмП := B;
$B_1\neg$A: СмЛ := $\neg$A; $B_1\neg$B: СмП := $\neg$B.

Многофункциональный сумматор выполняет микрооперации арифметического сложения (с учетом переноса C_{IN}), сложения по модулю два, логического сложения и логического умножения кодов на левом и правом входах:

Сл: См := СмЛ + СмП + $C_{ин}$;
М2: См := СмЛ ⊕ СмП;
И: См := СмЛ ∧ СмП;
ИЛИ: См := СмЛ ∨ СмП.

Формирователь логических условий на основе анализа кода на выходе См вырабатывает значения ознакомительных сигналов, передаваемых в УУ машины. Осведомительными сигналами могут быть: признак знака S, признак переполнения V, признак нулевого значения результата Z и т. п. Сдвигатель служит для выполнения микроопераций сдвига кода на выходе См:

$П_1$: С := См;
R_1: С := R1(SL • См), SR := См(n);
L_1: С := L1(См • SR), SL := См(0).

Микрооперация П1 обеспечивает передачу результата на магистраль С без сдвига. По ходу микрооперации R_1 результат сдвигается на один разряд вправо, при этом в освобождающийся старший разряд заносится значение с внешнего контакта SL, а выдвигаемый (младший) разряд сумматора посылается на внешний контакт SR.

В микрооперации L_1 результат сдвигается на один разряд влево. Здесь в освобождающийся младший разряд заносится значение с внешнего контакта SR, а выдвигаемый (старший) разряд См передается на внешний контакт SL.

Тема 10. Языки описание электронной аппаратуры

Языки описания электронной аппаратуры

Язык описания аппаратуры – тип компьютерных языков для формального описания электрических цепей, особенно цифровой логики. Он описывает структуру и функционирование цепи.

Развитие систем автоматизированного проектирования (САПР) в области электроники привело к созданию унифицированных языковых средств исходного описания проектов сверхбольших интегральных схем (СБИС). Такого рода языки необходимы для решения задач высокоуровневого проектирования цифровых СБИС. Под высокоуровневым понимается алгоритмическое и логическое проектирование СБИС, а основными задачами высокоуровневого проектирования являются моделирование проектов, синтез логических схем и верификация.

Такими языками стали VHDL и Verilog, они фактически являются международными стандартами и входными языками промышленных систем автоматизированного проектирования СБИС различных типов – заказных, полузаказных, и программируемых логических интегральных схем (ПЛИС).

Например, VHDL является базовым языком проектирования (сквозного моделирования и синтеза) в свободно распространяемой САПР WebPack ISE фирмы Xilinx, которая является одним из крупнейших производителей кристаллов ПЛИС. Поддержка VHDL и Verilog является обязательной для всех крупных САПР ведущих мировых производителей, таких как Cadence, Mentor Graphics, Synopsys и другие. Существуют и специализированные САПР, ориентированные на синтез решений на базе ПЛИС различных производителей, например – Active-HDL фирмы ALDEC.

Язык VHDL был разработан в 1983 г. по заказу МО США для формального описания логических схем на всех этапах разработки электронных систем. Аббревиатура VHDL (Very High Speed Integrated Circuit Hardware Description Language) переводится как язык описания высокоскоростных интегрированных схем.

Синтаксис VHDL очень похож на язык Ада. Дело в том, что компания Intermetrics, которой Пентагон поручил специфицировать VHDL, имела большой опыт работы с языком Ада.

Уровни описания электронной аппаратуры

Современные цифровые электронные приборы являются, без сомнения, сложными системами, состоящими из миллионов транзисторов. Проектирование любой сложной системы предполагает применение блочно-иерархического подхода, позволяющего сократить вероятность ошибок, обусловленных большой размерностью задачи. Проектирование цифровой электронной аппаратуры не является исключением, более того, уровни иерархии можно выделить гораздо более чётко, чем во многих других прикладных областях. Для блоков, описываемых на каждом из уровней, характерны свой набор элементов, способ представления информации (язык описания), используемый математический аппарат.

Обычно выделяют 5 уровней описания вычислительных систем:

- системный уровень;
- уровень вычислительных процессов;
- функционально-логический уровень;
- схемотехнический уровень;
- компонентный уровень.

Функционально-логический уровень является самым ёмким и в нём дополнительно выделяют ряд подуровней:

- подуровень функционирования ЭВМ;
- подуровень устройств (узлов) ЭВМ;
- регистровый подуровень;
- вентильный уровень.

Языки описания аппаратуры (Hardware Description Language, HDL) позволяют описывать блоки в первую очередь на функционально-логическом уровне, наибольшее распространение имеют на регистровом и вентильном подуровнях и при описании узлов ЭВМ. Проекты на языках HDL могут быть использованы и для решения задач системного уровня проектирования. Кроме того, специальные расширения, например, VHDL-AMS могут рассматриваться как промежуточный подуровень описания между вентильным подуровнем функционально-логического уровня и схемотехническим уровнем, что даёт возможность моделировать не только цифровую, но и аналоговую аппаратуру.

История развития HDL

С начала 70-х годов стала актуальной проблема создания стандартного средства документации схем и алгоритмов дискретных систем переработки информации и цифровой аппаратуры, одинаково пригодной как для восприятия человеком, так и для обработки на ЭВМ.

Известно большое число предшественников современных HDL, как отечественных, так и зарубежных. Отечественные – «МОДИС», «МОДИС-В87» «Автокод-М», MPL, ООС-2, «Форос», «Алгоритм», «Пульс», «Симпатия» и др. Зарубежные – CDL, DDL, ISPS, CONLAN, HILO и др.

В настоящее время известны 2 стандартных языка описания аппаратуры – VHDL и Verilog-HDL, называемый далее для краткости Verilog.

Язык VHDL (Very high speed integrated circuit Hardware Description Language – язык описания сверхскоростных БИС) был разработан международной группой по заданию Министерства обороны США в начале 80-х годов с целью обеспечения единообразного понимания подсистем различными проектными группами. В 1987 году спецификация языка VHDL была принята в качестве стандарта ANSI/IEEE STD 1076-1987. Удобства и относительная универсальность конструкций этого языка достаточно быстро привели к созданию программ моделирования систем на основании их описания в терминах VHDL.

С начала 90-х годов разрабатываются прямые компиляторы VHDL-описаний в аппаратные реализации различных классов. Это наряду с необходимостью более

адекватного представления в языке современных тенденций в цифровой схемотехнике привело к созданию расширенного стандарта ANSI/IEEE STD 1076-1993. В 1999 году была утверждена последняя версия стандарта ANSI/IEEE STD 1076-1999, известная как VHDL-AMS (AMS – Analog and Mixed-Signal Extentions). Наиболее существенным нововведением этой версии языка, как понятно из названия, является появление конструкций, обеспечивающих эффективное описание аналоговых и смешанных цифро-аналоговых устройств.

Работу над усовершенствованием стандарта ведёт группа VASG (VHDL Analysis and Standardization Group). Ведутся также работы по стандартизации внутренней формы представления VHDL-описаний в ЭВМ (группа VIFASG – VHDL Intermediate Form Analysis and Standardization Group), формы задания тестов для VHDL-моделей (группа WAVES – Wave-form and Vector Exchange to Support Design and Test Verification), задания параметров задержек компонент (группа VITAL – VHDL Initiative Towards Application-Specific Integrated Cir-cuit Libraries), алфавита представления значений сигналов в моделях и операции в этом алфавите (стандарт IEEE std_logic_1164) и т.д.

Язык Verilog был разработан в 1985 году фирмой Gateway Design Automation как язык моделирования, ориентированный на внутреннее применение. Позднее, в 1989 году, эта фирма была куплена корпорацией Cadence, которая открыла Verilog для общественного использования. После этого язык был стандартизован – IEEE 1364-1995. В отличие от VHDL, который строго типизирован и синтаксически напоминает языки ADA и Paskal, Verilog базируется на C, имеет меньше встроенных возможностей саморасширения, но зато более прост в реализации, имеет более развитый интерфейс с языком C и лаконичен. В настоящее время принят стандарт IEEE 1364-2001, который, в частности, включил в себя ряд стилистических средств, сближающих его с VHDL. Несмотря на то, что VHDL был создан раньше и предоставляет более широкие возможности, Verilog стал достаточно популярен и компиляторы с этого языка наряду с VHDL-компиляторами включены в подавляющее большинство САПР БИС. Более того, зачастую поддерживается компиляция смешанных проектов.

Варианты использования HDL

Проектировщик БИС может составить функциональное HDL-описание проектируемого кристалла и, используя систему моделирования САПР, проверить его соответствие спецификации (провести функциональную верификацию). После этого с помощью системы логического синтеза автоматически синтезировать схему (получить её структурное HDL-описание) в заданном элементном базисе. Затем моделирование полученной логической схемы оценить корректность результатов синтеза. После чего с помощью системы автоматизированного конструкторского проектирования провести трассировку соединений, а моделированием проверить правильность работы схемы с учётом задержек и наводок.

Возможен автоматический синтез схем с учётом контролепригодности, синтез контролирующих тестов, а также анализ тестов на полноту и корректность. HDL используется не только для представления проектируемых схем, но и для описания тестирующих программ (testbench) и тестов.

Имея в своём распоряжении выполненные с учётом требований многократного использования HDL-описания ранее спроектированных устройств, с помощью САПР несложно включать эти описания в состав новых проектов, повторно реализовать их на более современной технологии и т.д.

Эксплуатационщик цифровой электронной аппаратуры при наличии документации в виде HDL-описания устройства и тестирующей программы на их базе может осуществить модернизацию схем, использовать HDL-модели при поиске неисправностей в схеме и доработке контрольных тестов.

Стандартизация входных языков и внутренних интерфейсов подсистем САПР, в том числе и на базе HDL, создаёт общую коммуникационную среду проектирования, позволяет упростить интеграцию решений различных производителей программного обеспечения, об-мен библиотеками моделей компонент и проектов, модернизацию отдельных подсистем САПР.

Преимущества HDL

К основным достоинствам HDL следует отнести следующие:

- стандартность;
- многоаспектность и иерархичность;
- пригодность для восприятия человеком и обработки на ЭВМ.

Действительно, языки VHDL и Verilog официально признаны стандартом описания цифровой аппаратуры, который поддерживается военно-промышленным комплексом и радиоэлектронной промышленностью стран-лидеров в области электроники и микроэлектроники. Наличие такого стандарта облегчает обмен данными и документацией между отдельными группами разработчиков и эксплуатационщиков аппаратуры, различными САПР и их подсистемами.

HDL пригодны для описания как схем аппаратуры, так и функциональных тестов и алгоритмов функционирования. Они покрывают широкий диапазон уровней структурной детализации описаний цифровой аппаратуры: от описания архитектуры ЭВМ на уровне устройств типа процессор-память до описания узлов типа триггер на уровне вентилей и МОП-ключей, от описаний алгоритмов ЭВМ на уровне команд до описаний алгоритмов устройств на уровне межрегистровых передач и булевских функций, от описаний функциональных тестов до тестов проверки схем.

На высших уровнях абстракции HDL-описания можно рассматривать как средство спецификации требований к проекту.

Описания на различных HDL легко доступны для понимания человеком и пригодны для обработки такими компонентами САПР БИС, как подсистемы моделирования, подсистемы формальной верификации, подсистемы логического синтеза, подсистемы синтеза с учётом тестопригодности, системы синтеза и анализа контрольных тестов, временные анализаторы, кремниевые компиляторы, подсистемы автоматизации конструкторского проектирования и т.д.

HDL с точки зрения схемотехника

В HDL-описании электронных устройств, как и в любой модели, отражаются только некоторые аспекты реальной системы.

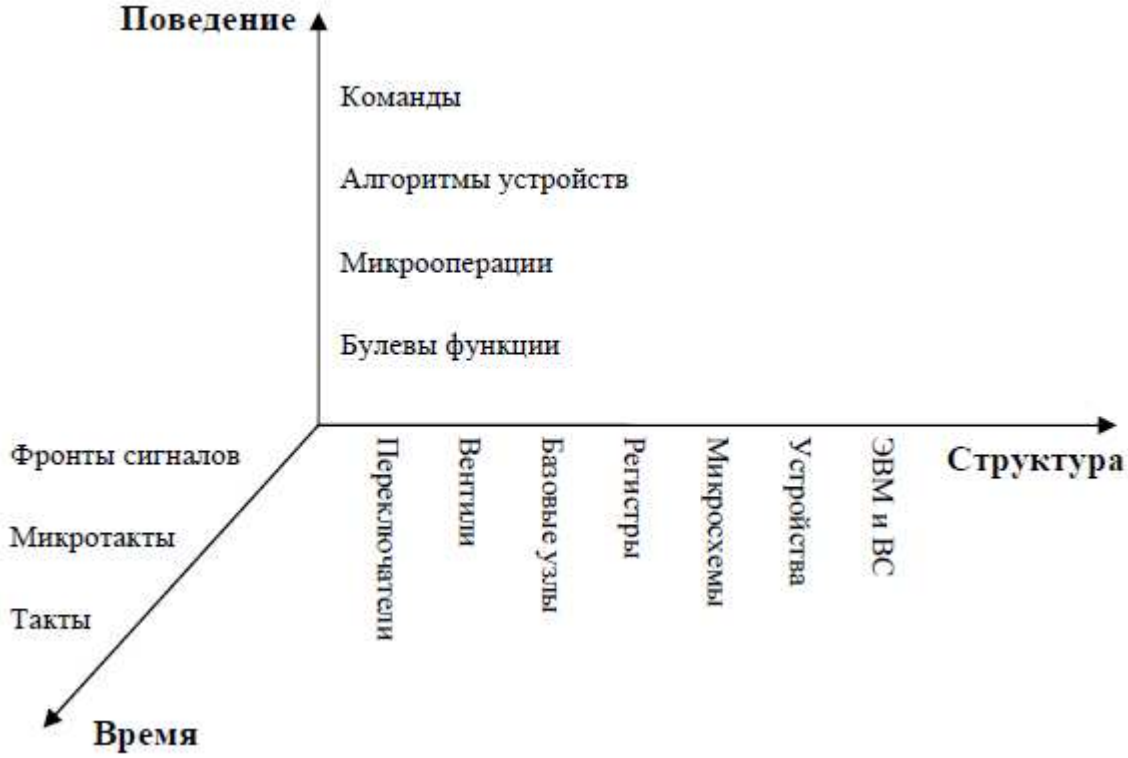
Цифровую аппаратуру характеризуют, например, такие аспекты как:

- функциональный (реализуемая функция, алгоритм);
- временной (задержки, время отклика);
- структурный (типы и связи компонент);
- ресурсный (число вентилей, площадь кристалла);
- надёжностный (время наработки на отказ);
- конструктивный (вес, габариты);
- стоимостной и т.д.

HDL содержат средства, позволяющие отразить в основном функциональный, временной и структурный аспекты.

Как уже отмечалось выше, описание того или иного аспекта может быть детализировано в весьма широких пределах. Например, функциональное описание – от уровня системы команд и алгоритмов устройств до булевских функций; структурное описание – от уровня устройств типа процессор-память до уровня вентилей и переключателей; временные характеристики – от задержек фронтов сигналов до тактов и

задержек электромеханических устройств. Возможности HDL по отражению указанных аспектов приведены на рисунке.



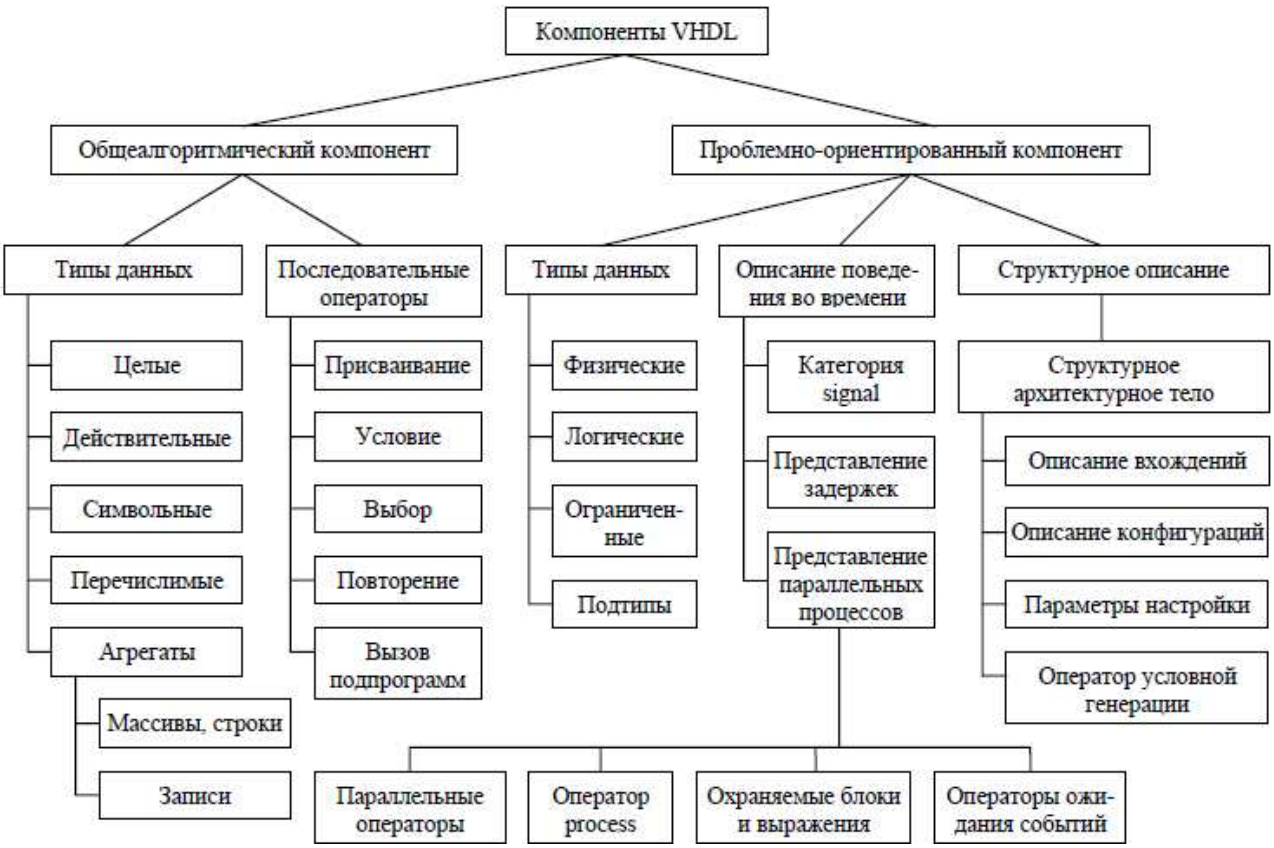
Степень детализации того или иного аспекта определяется конкретными задачами. Например, описание некоторой микропроцессорной вычислительной системы может строиться как описание структуры, состоящей из микросхем БИС и СБИС, а описание самих микросхем строится как поведенческое, если их описание на вентильном уровне либо отсутствует, либо слишком громоздко.

С точки зрения разработчика электронной аппаратуры, HDL-описания должны давать возможность описывать как интерфейсы объектов проекта, так и их структуру (более того – многообразие структур). В зависимости от решаемой задачи при разработке описаний структур проекта могут быть использованы разные стили описания архитектур. Подробно эти понятия будут рассмотрены при изучении базовых понятий языка VHDL.

HDL с точки зрения программиста

Учитывая соображения, высказанные выше, средства HDL можно отнести к одному из двух разделов:

- общеалгоритмический компонент, определяющий набор типов данных и операторов, обеспечивающих общее описание алгоритма функционирования;
- проблемно-ориентированный компонент, включающий такие специфические и важные для описания аппаратных средств разделы, как специальные типы данных, средства для описания процессов с учётом их протекания в реальном времени и средства для структурного представления проекта.



Общеалгоритмический компонент по составу, смыслу и принципам использования её составляющих мало отличается от состава традиционных языков программирования, а форма записи (синтаксис и семантика языковых конструкций) весьма близка к традиционным языкам программирования высокого уровня.

На примере VHDL предварительно остановимся на некоторых составляющих проблемно-ориентированного компонента.

В числе проблемно-ориентированных типов данных следует отметить физический тип, в первую очередь – время. Пользователь также может вводить дополнительные типы данных, отражающих электрические или механические свойства моделируемых объектов.

Многозначная логика формально в языке не определена. Однако входящие в любой программный комплекс моделирования на VHDL стандартные пакеты определяют несколько допустимых алфавитов представления сигналов на основе их декларации как данных перечислимого типа, а также определяют правила их преобразования.

Среди средств представления поведения системы в реальном времени следует отметить средства представления параллелизма в реальной системе:

- понятие сигнала, как единицы передаваемой информации между параллельно работающими компонентами;
- так называемые параллельные операторы, отражающие непосредственное взаимодействие компонентов;
- понятие процесса, как совокупности действий, инициируемой изменениями сигналов.

При моделировании параллельный оператор интерпретируется таким образом, что он исполняется при любом изменении сигналов, являющихся его аргументами, точнее, при обработке реакции на соответствующее событие. Отметим, что моделирование на основе VHDL-описания должно выполняться на базе дискретной событийной модели. Кроме средств описания параллельных процессов определены конструкции, явно указывающие поведение объекта проектирования во времени – выражения задержки after, оператор приостановки wait и ряд других.

Средства структурного представления проекта включают оператор вхождения компонента (Component Instance Statement), задающий тип включаемых в устройство структурных компонентов и способы их соединений, декларации конфигурации (Configuration Declaration) с помощью которых можно выбирать вариант реализации включаемого компонента, и ряд других конструкций языка.

Из представленного обзора можно видеть, что VHDL (и другие HDL) представляет собой развитую алгоритмическую систему, позволяющую описывать разнообразные структуры и явления в информационных системах.

Тема 11. Большие и малые интерфейсы IBM PC

Шины и слоты

Компьютерная шина – подсистема, которая передаёт данные между функциональными блоками компьютера. К шине компьютера можно подключить несколько устройств по одному набору проводников. Каждая шина определяет свой набор коннекторов (соединений) для физического подключения устройств, карт и кабелей.

AGP интерфейс (Accelerated Graphics port) – это модифицированная версия шины PCI, спроектированная для ускоренного обмена с графическими картами.

CardBus – 32-разрядная шина, созданная PCMCIA.

CompactPCI шина – это версия PCI, адаптированная для промышленных и\или встроенных приложений.

FireWire (IEEE1394) шинный интерфейс – был разработан IEEE 1394-1995 стандарт как последовательный протокол передачи данный и система взаимодействия. Также известен как iLink (Sony) или Lynx. Часто реализовывался в устройствах бытовой электроники, цифровых видеокамерах, видеомагнитофонах, некоторых других видах мультимедийного оборудования и компьютерах.

IndustrialPCI – это версия PCI, адаптированная для промышленных и\или встроенных приложений.

ISA шина – Industry Standard Architecture.

Mini PCI шина – это альтернативная реализация PCI, разработанная для маленьких форм-факторов. Ее спецификация – это подвид стандарта PCI, использующий 100 пиновый (Type I/II) или 124 пиновый (Type III) коннектор.

PC Card ATA шина – эта спецификация делает возможным обмен по ATA & PC Card с теми же коннекторами.

PC/104 шина – это компактная версия шины ISA. PC/104 предназначена для специализированных встраиваемых вычислительных сред, где приложения зависят от надежного сбора данных, несмотря на зачастую экстремальные условия.

PCI шина – это высокопроизводительная шина для соединения чипов, плат расширения и подсистем процессора и памяти.

PCI Express 1x, 4x, 8x, 16x шина – это новое последовательное шинное расширение серии спецификаций PCI.

PCI Express Mini Card (также известное как Mini PCI Express, Mini PCIe, and Mini PCI-E) – это замена Mini PCI форм-фактора, базирующаяся на PCI Express. Была разработана в PCI-SIG. Устройство-хостер поддерживает PCI Express и USB 2.0 соединения, и каждая карта использует то, что проектировщик считает наиболее подходящим к задаче. Большинство ноутбуков, построенных после 2005, базируются на PCI Express и могут иметь несколько слотов Mini Card.

PCMCIA (Personal Computer Memory Card International Association) – 16-битная шина

USB (Universal Serial Bus) – спроектирована для соединения ПК с периферийными устройствами, такими как мыши, клавиатуры, сканнеры, цифровые камеры, принтеры, жесткие диски и сетевые компоненты. Она стала стандартным методом соединения для сканнеров, цифровых камер и некоторых принтеров.

USB 3.0 – это развитие USB 2.0.

Разъемы для видеокарт, мониторов

VGA (Video Graphics adapter) – практически все современные графические карты для ПК используют один и тот же 15-пиновый VGA коннектор, идущий от оригинальных карт IBM VGA. Видеоадаптер VGA, в отличие от предыдущих видеоадаптеров IBM (MDA, CGA, EGA), использует аналоговый сигнал для передачи цветовой информации. Переход на аналоговый сигнал был обусловлен необходимостью сокращения числа проводов в кабеле. Также аналоговый сигнал давал возможность использовать VGA-мониторы с последующими видеоадаптерами, которые могут выводить большее количество цветов. В настоящее время VGA считается устаревшим и активно вытесняется цифровыми интерфейсами DVI, HDMI и DisplayPort. Крупнейшие производители электроники Intel и AMD объявили о полном отказе от поддержки VGA к 2015 году. К большинству видеоадаптеров, уже не имеющих разъёма VGA, аналоговый монитор можно подключить к выходу DVI через специальный переходник, поскольку часть его линий на самом деле в целях совместимости является интерфейсом VGA (за исключением формата DVI-D, в котором аналоговые линии отсутствуют).

Digital Visual Interface (DVI) – стандарт на интерфейс и соответствующий разъём, предназначенный для передачи видеоизображения на цифровые устройства отображения, такие как жидкокристаллические мониторы и проекторы. Разработан консорциумом Digital Display Working Group. Single link (одинарный режим) DVI использует четыре витых пары проводов (красный, зелёный, синий, и clock), обеспечивающих возможность передавать 24 бита на пиксель. С ним может быть достигнуто максимальное возможное разрешение при 1920x1200 60 Гц, при 1920x1080 - 75Гц. Dual link (двойной режим) DVI удваивает пропускную способность и позволяет получать разрешения экрана 2560x1600 и 2048x1536. Поэтому для самых крупных LCD мониторов с большим разрешением, таких, как 30" модели, обязательно нужна видеокарта с двухканальным DVI Dual-Link выходом. Если у монитора максимальное разрешение экрана 1280x1024, то подключать его кабелем dual link не имеет смысла, т.к. данный кабель предназначен для мониторов с большим разрешением.

High-Definition Multimedia Interface (HDMI) – интерфейс для мультимедиа высокой чёткости, позволяющий передавать цифровые видеоданные высокого разрешения и многоканальные цифровые аудиосигналы с защитой от копирования (High Bandwidth Digital Copy Protection, HDCP). Разъём HDMI обеспечивает цифровое DVI-соединение нескольких устройств с помощью соответствующих кабелей. Основное различие между HDMI и DVI состоит в том, что разъём HDMI меньше по размеру, а также поддерживает передачу многоканальных цифровых аудиосигналов.

Mini-DVI коннектор используется на компьютерах Apple как цифровая альтернатива Mini-VGA коннектора.

Mini-VGA коннектор используется на ноутбуках и других системах вместо стандартного VGA коннектора. Помимо своей компактной формы, mini-VGA порты имеют дополнительные возможности по выводу композитного сигнала и S-Video (кроме VGA сигналов) через использование EDID.

DisplayPort – стандарт сигнального интерфейса для цифровых дисплеев. Принят VESA (Video Electronics Standard Association) в мае 2006, версия 1.1 принята 2 апреля 2007, а версия 1.2 принята 7 января 2010. DisplayPort предполагается к использованию в качестве наиболее современного интерфейса соединения аудио и видеоаппаратуры, в первую очередь для соединения компьютера с дисплеем, или компьютера и систем домашнего кинотеатра. DisplayPort поддерживает HDCP версии 1.3 и имеет пропускную способность вдвое большую, чем Dual-Link DVI, низкое напряжение питания и низкие посторонние наводки. Размеры разъёма Mini DisplayPort в 10 раз меньше, чем у стандартного разъёма DVI. Технология, реализованная в DisplayPort, позволяет передавать

одновременно как графические, так и аудио сигналы. Основное отличие от HDMI – более широкий канал для передачи данных (10,8 Гбит/с вместо 10,2 Гбит/с). Максимальная длина кабеля DisplayPort составляет 15 метров. Вместо HDCP, защиты от копирования HDMI, будет реализована технология DPCP (DisplayPort Content Protection), основанная на 128-битном AES шифровании. DisplayPort 1.2 имеет максимальную скорость передачи данных 21,6 Гбит/с на расстоянии до 3 метров, что больше, чем HDMI Type B (2x10,2 Гбит/с). Также поддерживает несколько независимых потоков, пропускная способность вспомогательного канала в стандарте увеличена с 1 до 720 Мбит/с. Таким образом, через интерфейс DisplayPort 1.2 можно подключить до двух мониторов, воспроизводящих картинку размером 2560 x 1600 точек с частотой 60 Гц, либо до четырёх мониторов с разрешением 1920 x 1200 точек. При использовании одиночного монитора поддерживаемое разрешение возрастает до 3840 x 2400 точек с частотой 60 Гц, монитор с поддержкой частоты обновления 120 Гц поддерживается при разрешениях до 2560 x 1600 точек. Это позволяет стандарту DisplayPort 1.2 работать с технологиями построения стереоскопического изображения.

Порты для устройств ввода

Keyboard PS/2 – широко используемый коннектор. PS/2 интерфейс клавиатуры электрически идентичен AT интерфейсу, но коннектор кабеля имеет 6-пиновый mini-DIN интерфейс. PS/2 порты используют синхронные последовательные сигналы, чтобы обмениваться между компьютером и клавиатурой или мышкой.

Разъем для локальных сетей

8P8C (8 Positions 8 Contacts), наряду с этим ошибочно называемый RJ45 – унифицированный разъём, используемый в телекоммуникациях, имеет 8 контактов и защёлку. Используется для создания ЛВС по технологиям 10BASE-T, 100BASE-T и 1000BASE-TX с использованием 4-парных кабелей витой пары. Используется во многих других областях и для построения иных сетей. Телефонный унифицированный разъём RJ-11 меньше по размеру и может вставляться в гнёзда 8P8C (для обратной совместимости). Для создания полного соединения с сигнальным кабелем проводники вводятся в разъём по соответствующим стандартам и обжимаются специальным обжимным инструментом (кримпером). При обжиме для сетей Ethernet используются специальные таблицы (TIA/EIA-568-B).

Разъемы для жестких дисков

ATA (AT Attachment) – параллельный интерфейс подключения накопителей (жёстких дисков и оптических приводов) к компьютеру. В 1990-е годы был стандартом на платформе IBM PC; в настоящее время вытесняется своим последователем – SATA и с его появлением получил название PATA (Parallel ATA). Первоначальная версия стандарта была разработана в 1986 году фирмой Western Digital и по маркетинговым соображениям получила название IDE (Integrated Drive Electronics – «электроника, встроенная в привод»). Оно подчеркивало важное нововведение: контроллер привода располагается в нём самом, а не в виде отдельной платы расширения. Поначалу этот интерфейс использовался с жёсткими дисками, но затем стандарт был расширен для работы и с другими устройствами, в основном – использующими сменные носители. Важным этапом в развитии ATA стал переход от PIO Programmed input/output – программный ввод/вывод) к DMA (Direct memory access – прямой доступ к памяти). Долгое время шлейф ATA содержал 40 проводников, но с введением режима Ultra DMA/66 (UDMA4) появилась его 80-проводная версия. Все дополнительные проводники – это проводники заземления, чередующиеся с информационными проводниками. Таким образом, вместо семи проводников заземления их стало 47. Такое чередование проводников уменьшает

ёмкостную связь между ними, тем самым сокращая взаимные наводки. Ёмкостная связь является проблемой при высоких скоростях передачи, поэтому данное нововведение было необходимо для обеспечения нормальной работы установленной спецификацией UDMA4 скорости передачи 66 МБ/с (мегабайт в секунду). Более быстрые режимы UDMA5 и UDMA6 также требуют 80-проводного кабеля.

SATA (Serial ATA) – последовательный интерфейс обмена данными с накопителями информации. SATA является развитием параллельного интерфейса ATA (IDE), который после появления SATA был переименован в PATA (Parallel ATA). Первоначально стандарт SATA предусматривал работу шины на частоте 1,5 ГГц, обеспечивающей пропускную способность приблизительно в 1,2 Гбит/с (150 МБ/с). (20%-я потеря производительности объясняется использованием системы кодирования 8b/10b, при которой на каждые 8 бит полезной информации приходится 2 служебных бита). Пропускная способность SATA/150 незначительно выше пропускной способности шины Ultra ATA (UDMA/133). Главным преимуществом SATA перед PATA является использование последовательной шины вместо параллельной. Несмотря на то, что последовательный способ обмена принципиально медленнее параллельного, в данном случае это компенсируется возможностью работы на более высоких частотах за счёт отсутствия необходимости синхронизации каналов и большей помехоустойчивостью кабеля. Это достигается применением принципиально иного способа передачи данных. Стандарт SATA/300 работает на частоте 3 ГГц, обеспечивает пропускную способность до 3 Гбит/с (300 МБайт/с для данных с учетом 8b/10b кодирования). Впервые был реализован в контроллере чипсета nForce 4 фирмы «NVIDIA». Часто стандарт SATA/300 называют SATA II или SATA 2.0. Теоретически устройства SATA/150 и SATA/300 должны быть совместимы (как контроллер SATA/300 с устройством SATA/150, так и контроллер SATA/150 с устройством SATA/300) за счёт поддержки согласования скоростей (в меньшую сторону), однако для некоторых устройств и контроллеров требуется ручное выставление режима работы (например, на НЖМД фирмы Seagate, поддерживающих SATA/300, для принудительного включения режима SATA/150 предусмотрен специальный джампер). Спецификация SATA Revision 3.0 представлена в июле 2008 и предусматривает пропускную способность до 6 Гбит/с (600 Мбайт/с для данных с учетом 8b/10b кодирования). В числе улучшений SATA Revision 3.0 по сравнению с предыдущей версией спецификации, помимо более высокой скорости, можно отметить улучшенное управление питанием. Также сохранена совместимость, как на уровне разъёмов и кабелей SATA, так и на уровне протоколов обмена. SATA использует 7-контактный разъём вместо 40-контактного разъёма у PATA.

Mini-SATA – это форм-фактор твердотельных жестких дисков, имеющий размер 50,95 x 30 x 3 мм, был объявлен Serial ATA International Organization 21 сентября 2009 года. Поддерживает нетбуки и другие устройства, которые требуют небольшие SSD диски. Разъем mSATA похож на интерфейс PCI Express Mini Card, они электрически совместимы, но требуют переключения некоторых сигналов на соответствующий контроллер.

SCSI (Small Computer System Interface, произносится «скази») – представляет собой набор стандартов для физического подключения и передачи данных между компьютерами и периферийными устройствами . SCSI стандарты определяют команды, протоколы и электрические и оптические интерфейсы. Разработан для объединения на одной шине различных по своему назначению устройств, таких как жёсткие диски, накопители на магнитооптических дисках, приводы CD, DVD, стримеры, сканеры, принтеры и т.д. Теоретически возможен выпуск устройства любого типа на шине SCSI. После стандартизации в 1986 году SCSI начал широко применяться в компьютерах Apple Macintosh, Sun Microsystems. В компьютерах, совместимых с IBM PC, SCSI не пользуется такой популярностью в связи со своей сложностью и сравнительно высокой стоимостью и применяется преимущественно в серверах. SCSI широко применяется на серверах,

высокопроизводительных рабочих станциях; RAID-массивы на серверах часто строятся на жёстких дисках со SCSI-интерфейсом (однако, в серверах нижнего ценового диапазона всё чаще применяются RAID-массивы на основе SATA). В настоящее время устройства на шине SAS постепенно вытесняют устаревшую шину SCSI. Существует реализация системы команд SCSI поверх оборудования (контроллеров и кабелей) IDE/ATA/SATA, называемая ATAPI – ATA Packet Interface. Все используемые в компьютерной технике подключаемые по IDE/ATA/SATA приводы CD/DVD/Blu-Ray используют эту технологию. Также система команд SCSI реализована поверх протокола USB, что является частью спецификации класса Mass Storage device. Это позволяет подключать через интерфейс USB любые хранилища данных (от флеш-накопителей до внешних жёстких дисков), не разрабатывая для них собственного протокола обмена, а вместо этого используя имеющийся в операционной системе драйвер SCSI. Существует множество стандартов SCSI: SCSI-1 (восьмибитная шина, с пропускной способностью в 1,5 МБайт/сек в асинхронном режиме и 5 МБайт/сек в синхронном режиме, максимальная длина кабеля – до 6 метров); SCSI-2 существовал в двух вариантах Fast SCSI (удвоенная пропускная способность (до 10 МБайт/сек) и Wide SCSI (в дополнение имеет удвоенную разрядность шины 16 бит, что позволяет достичь скорости передачи до 20 МБ/сек); SCSI-3 или Ultra SCSI (пропускная способность шины составила 20 МБайт/сек для восьмибитной шины и 40 МБайт/сек – для шестнадцатибитной); Ultra-2 SCSI (максимальная длина кабеля – 12 метров, пропускная способность – до 80 МБайт/сек); Ultra-3 SCSI или Ultra-160 SCSI (имеет удвоенную пропускную способность, которая составила 160 МБайт/сек), а также Ultra-320 SCSI и Ultra-640 SCSI.

Serial Attached SCSI (SAS) – компьютерный интерфейс, разработанный для обмена данными с такими устройствами, как жёсткие диски и ленточные накопители. SAS использует последовательный интерфейс для работы с непосредственно подключаемыми накопителями (Direct Attached Storage (DAS) devices). SAS разработан для замены параллельного интерфейса SCSI и позволяет достичь более высокой пропускной способности, чем SCSI; в то же время SAS обратно совместим с интерфейсом SATA: устройства 3Гбит/с и 6Гбит/с SATA могут быть подключены к контроллеру SAS, но устройства SAS нельзя подключить к контроллеру SATA. Хотя SAS использует последовательный интерфейс в отличие от параллельного интерфейса, используемого традиционным SCSI, для управления SAS-устройствами по-прежнему используются команды SCSI. Протокол SAS разработан и поддерживается комитетом T10. Текущую рабочую версию спецификации SAS можно скачать с его сайта. SAS поддерживает передачу информации со скоростью до 6 Гбит/с; ожидается, что к 2012 году скорость передачи достигнет 12 Гбит/с. Благодаря уменьшенному разъему SAS обеспечивает полное двухпортовое подключение как для 3,5-дюймовых, так и для 2,5-дюймовых дисковых накопителей (раньше эта функция была доступна только для 3,5-дюймовых дисковых накопителей с интерфейсом Fibre Channel).

Тема 12. Внешние запоминающие устройства

Основные характеристики и классификация ВЗУ

Внешняя память ЭВМ образует третий уровень в иерархической структуре памяти ЭВМ. Она строится на основе различного рода внешних ЗУ (ВЗУ): накопителей на магнитных дисках (НМД), накопителей на магнитной ленте (НМЛ), накопителей на оптических дисках (НОД), накопителей на магнитооптических дисках (НМОД) и др.

ВЗУ состоит из трех частей: носитель информации (съемный или стационарный), привод носителя и контроллер накопителя (ВЗУ).

Основные характеристики ВЗУ: емкость, быстродействие, надежность, стоимость. Емкость Е ВЗУ определяется плотностью записи и площадью носителя, измеряется МВ, ГВ.

Быстродействие ВЗУ определяется временем обращения $t_{обр}$:

$V_{ВЗУ} = 1/ t_{обр}$; $t_{обр} = t_{дост} + t_{пер}$.

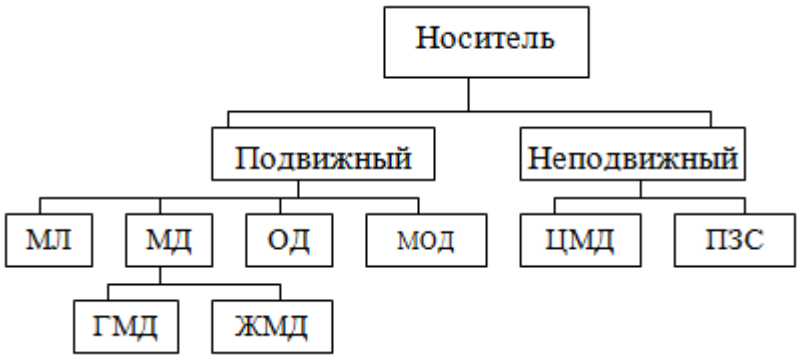
Здесь: $t_{дост}$ – время доступа к блоку информации (время поиска блока на носителе), $t_{пер}$ – время передачи блока информации (чтения или записи на носитель), зависит от многих факторов и определяет другую характеристику – пропускную способность накопителя – скорость передачи $V_{пер} = 1/t_{пер}$ (количество бит или байт в секунду), зависит от скорости движения носителя информации и плотности записи.

Надежность обычно определяется временем наработки на отказ $T = 1/\lambda$, где λ - интенсивность отказов.

Стоимость S – интегральная характеристика: зависит от емкости, быстродействия, надежности. Её можно (принято) характеризовать таким показателем, как удельная стоимость δ: $\delta = S/E$ – стоимость хранения единицы информации, например, стоимость хранения одного МВ информации.

ВЗУ можно классифицировать по различным признакам классификации: по типу носителя, по способу доступа к информации. На рисунке: МЛ – магнитная лента, ОД – оптический диск, МОД – магнитооптический диск, ЦМД – накопитель на основе цилиндрических магнитных доменов, ПЗС – накопитель на основе ПЗС-элементов, т. е. приборов с зарядовой связью.

По способу доступа к информации – ЗУ с произвольным (прямым) доступом к информации, ЗУ с последовательным доступом, со смешанным (комбинированным) доступом к информации.



К ЗУ с произвольным доступом относятся все ЗУ, в которых время доступа к блоку информации является постоянным, не зависит от адреса (номера) блока информации на носителе, поскольку доступ к нему осуществляется напрямую, без обращения к другим блокам на носителе.

К ЗУ с последовательным доступом относятся все ЗУ, в которых время доступа зависит от того, какое положение занимает адресуемый блок относительно средств чтения/записи информации. К этому типу относятся устройства, в которых выбор (поиск) блока с номером (n + N) осуществляется путем последовательного перебора (просмотра) блоков с номерами n+1,…,n+(N-1), где n - номер блока, к которому производилось последнее обращение.

ЗУ с последовательным доступом в свою очередь делятся на устройства с апериодическим и периодическим (циклическим) доступом.

К ЗУ с периодическим доступом относятся ВЗУ, в которых носитель вращается с постоянной скоростью и, следовательно, доступ к информации в них предоставляется (становится возможным) периодически (циклически) с периодом $T = 1/\omega$, где ω - угловая скорость вращения носителя относительно средств чтения/записи. К ним относятся ВЗУ типа накопителей на магнитных барабанах (НМБ), которые в настоящее время не применяются.

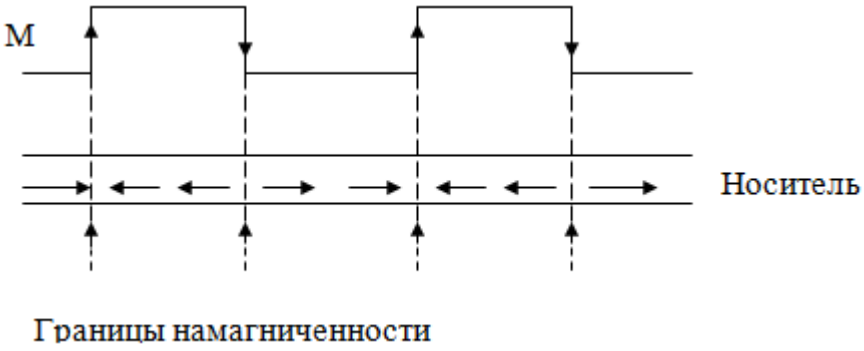
К ЗУ с последовательным апериодическим доступом относятся НМЛ.

Следует отметить, что накопители на магнитных дисках являются ЗУ смешанного типа. В них по номеру поверхности H обеспечивается прямой доступ, по номеру сектора S

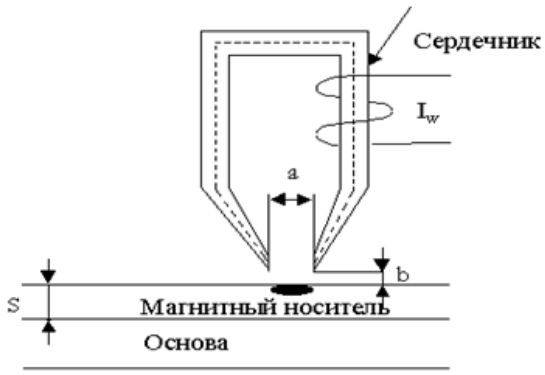
- последовательный периодический, и по номеру цилиндра С - последовательный апериодический доступ, т.к. информация на МД организована в виде трехмерного массива блоков (секторов). С логической точки зрения ЗУ на МД обычно относят к ЗУ с прямым доступом к информации.

Физические основы магнитной записи

Запись на магнитный носитель информации осуществляется при помощи магнитных головок (МГ). Носитель информации представляет собой немагнитную основу (пластик, алюминиевый сплав и т.п.), на поверхность которой нанесен тонкий слой магнитотвердого материала (с большой коэрцитивной силой, с прямоугольной петлей гистерезиса). Если эта магнитная поверхность не подвергалась воздействию внешнего магнитного поля, то магнитные домены в ней не упорядочены, ориентированы хаотично. Если поверхность подвергнуть воздействию магнитного поля, то магнитные домены ориентируются в направлении магнитного поля М.

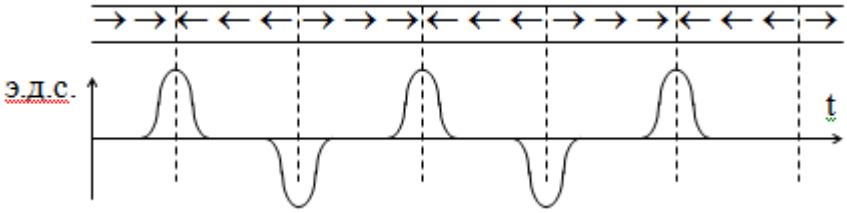


Внешнее магнитное воздействие на носитель создается МГ записи. МГ – это электромагнит, состоящий из электрической обмотки, по которой пропускается электрический ток записи, и магнитного сердечника, выполненного из магнитомягкого материала (с малой коэрцитивной силой H_c).



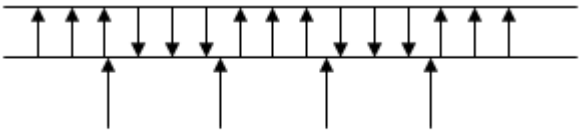
Ток записи I_w, подаваемый в обмотку МГ, создает в сердечнике магнитный поток М, который замыкается через расположенный под зазором головки носитель информации. Под воздействием этого поля домены ориентируются на этом участке поверхности либо в одном направлении, либо в другом, в зависимости от направления тока записи I_w.

Чтение информации осуществляется с помощью МГ чтения. МГ чтения позволяет определить (детектировать) моменты, когда под ней проходят границы участков намагниченности. В этот момент магнитный поток, создаваемый доменами при движении носителя, частично замыкается через сердечник (магнитопровод) МГ чтения и в ее электрической обмотке наводится э.д.с. (по Фарадею): э.д.с. = - kdФ/dt.



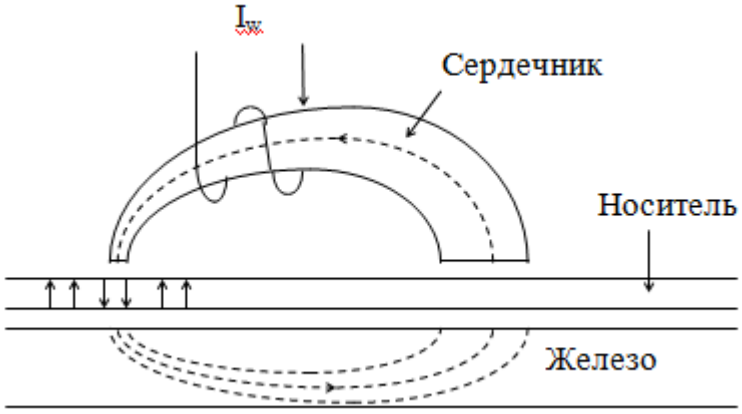
Амплитуда и длительность сигналов э.д.с. зависит от многих факторов: величины зазора a МГ чтения, толщины S покрытия, расстояния (зазора b) между МГ и носителем и магнитных свойств пленки. Амплитуда и длительность сигналов э.д.с. должны быть достаточными для надежного восприятия всех границ намагниченности: от этого зависит плотность записи на магнитный носитель. Рассмотренный способ магнитной записи естественно назвать горизонтальным способом записи, т. к. домены ориентируются вдоль поверхности носителя.

Если домены ориентировать поперек поверхности носителя, то способ естественно называть вертикальным. Он обеспечивает более высокую плотность записи.



Границы намагниченности

Конструкция МГ при вертикальной записи иная. Магнитный поток замыкается через сердечник (железо под носителем информации) и тем самым обеспечивается вертикальная ориентация доменов. Запись осуществляется в зазоре МГ с малым сечением сердечника (слева). Здесь обеспечивается режим $H>H_c$. Справа, где большое сечение сердечника, $H<H_c$ и перемагничивание доменов носителя не происходит.

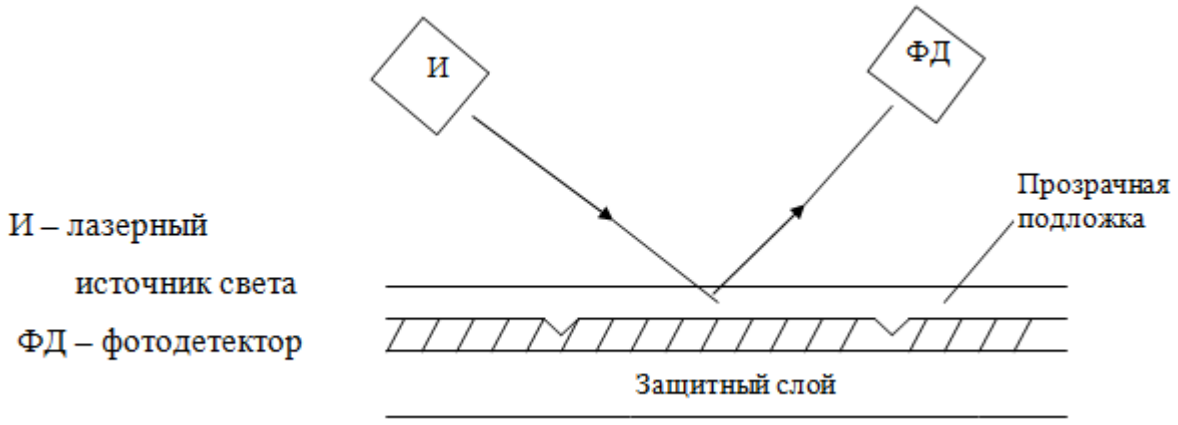


Физические основы оптической записи информации

Оптическая запись обеспечивает более высокую плотность записи – на порядок выше, чем магнитная. При оптической чтении информации используется способность некоторых материалов изменять свойства отражения света на тех участках носителя информации, которые подвергались механическому, тепловому, магнитному или комбинированному воздействию, например, и тепловому и магнитному, как в магнитооптических ЗУ.

Один из первых способов оптической записи основан на способности лазерного луча прожигать отверстие в тонком слое металла. Тонкий слой металла наносят (напыляют) на прозрачный диск – подложку. Прожженные лазером отверстия в металле являются оптическим отпечатком записываемой информации. Считывание производится также лазерным лучом меньшей интенсивности и фотодетектором: отраженный от поверхности

носителя луч попадает (или не попадает – рассеивается) на фотодетектор ФД и преобразуется в электрический сигнал.



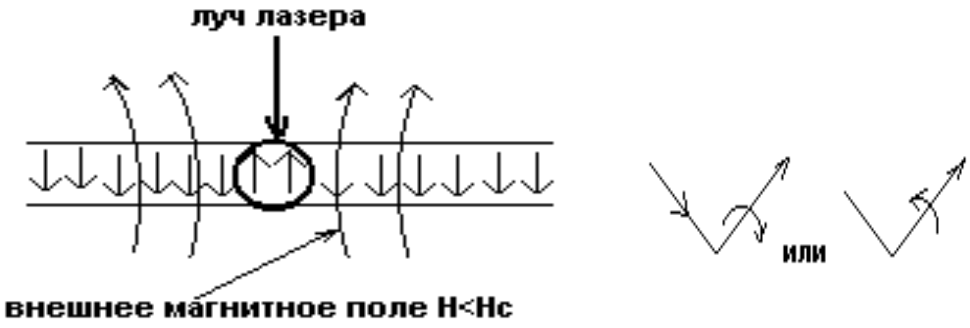
Ясно, что такой способ записи обеспечивает только однократную запись, поэтому используется только в постоянных ВЗУ (ROM ВЗУ).

Другой способ оптической записи основан на свойстве органического вещества увеличиваться в объеме при нагревании. Органическое вещество располагается под тонким слоем металла. Локальный (точечный) нагрев органического вещества обеспечивается лазерным лучом, что приводит к его вспучиванию в точке нагревания и, следовательно, к изменению отражательной способности, что и используется при считывании. Ясно, что этот способ также обеспечивает только однократную запись.

Возможность многократной записи обеспечивается при использовании магнитооптических носителей информации: запоминающий слой магнитооптического диска (МОД) выполняется в виде тонкой магнитной пленки из магнитотвердого материала. Запись информации осуществляется путем одновременного воздействия на эту пленку двух факторов:

- нагревание (локальное) лучом лазера выше температуры Кюри (145°С),
- воздействие на всю поверхность внешним магнитным полем с напряженностью $H<H_c$.

В результате такого совместного воздействия нагретый участок носителя изменяет состояние намагниченности – ориентацию магнитных доменов.



Считывание МОД производится узким лучом поляризованного света: перемагниченные участки изменяют угол поляризации отраженного луча и тем самым обеспечивают детектирование отпечатков.

Повторная запись информации на МОД требует предварительного стирания старой информации. Стирание осуществляется аналогично записи. Лучом лазера стираемый участок нагревается до температуры Кюри. Внешним магнитным полем противоположного направления этот участок приводится в исходное состояние. Итак, повторная запись информации на МОД обеспечивается в два этапа: сначала стирание и затем запись новой информации.

Кроме МОД, обеспечивающих многократную запись, существуют и чисто оптические диски с многократной записью. В них запись информации основана на изменении свойств вещества под воздействием температуры: это т.н. PCR-накопители. Принцип действия основан на изменении фазового состояния вещества в зависимости от его температуры, а именно: при разных значениях температуры вещество может находиться в кристаллическом или аморфном состоянии. Под воздействием температуры в локальных областях диска обеспечивается переход из кристаллического состояния в аморфное и наоборот – из аморфного в кристаллическое. Температурное воздействие обеспечивается достаточно мощным лучом лазера – при записи информации.

Для чтения информации используется менее мощный луч лазера, который по разному отражается от аморфных и кристаллических участков диска. От аморфных плохо отражается (слабая интенсивность отраженного светового потока), от кристаллических участков отражение хорошее, интенсивность высокая, что и обеспечивает уверенное детектирование информации.

Преимущества перед МОД:

1) не требуется предварительное стирание старой информации и, следовательно, не тратится время на стирание;
2) отсутствие магнитных полей и, следовательно, соответствующего оборудования;
3) использование лазера меньшей мощности.

Организация накопителей на оптической основе

Оптические ЗУ делятся на:

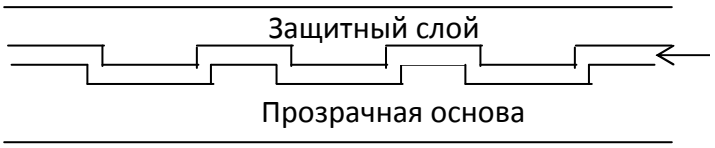
1) постоянные ЗУ типа CD-ROM;
2) ЗУ с однократной записью типа CD-R;
3) ЗУ с многократной записью: МОД с предварительным стиранием и ОД (без стирания) типа CD-RW.

Изготовление ОД типа CD-ROM осуществляется в три этапа:

- изготовление мастер – диска (запись - лучом лазера);
- изготовление матриц для тиражирования;
- тиражирование компакт- дисков путем прессования.

В качестве отражающей основы в CD-ROM обычно используется слой алюминия, нанесенный на поликарбонатную основу путем напыления:

Привод CD ROM должен обеспечивать:



1) постоянную линейную скорость движения относительно считывающей головки и, следовательно, переменную угловую скорость;
2) скорость вращения, увеличенную в k раз, k может принимать различные значения: k=2,4,8,16,24,32,48,... . Например, при k=8 пропускная способность составляет 1,2 МВ/с (при k=1 она равна 150 КВ/с – как при аудиозаписи),
3) позиционирование считывающей головки, осуществляется путем перемещения головки поперек дорожек,
4) точную фокусировку луча на отражающий слой.

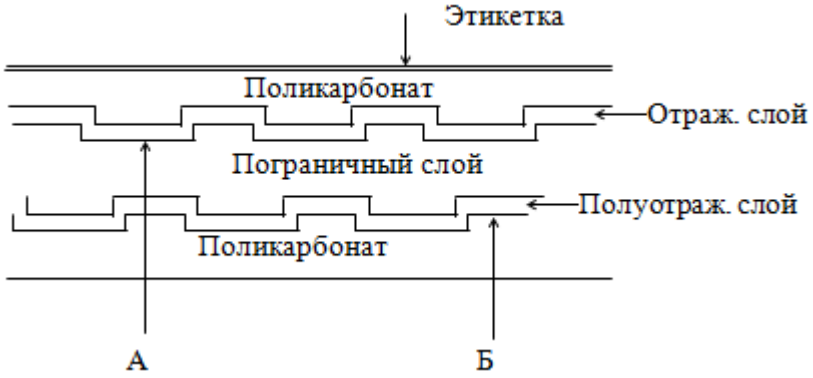
Второй тип ОД – CD-R – однократно записываемые пользователем диски. Чистые диски имеют заранее нанесенную спиральную дорожку, которая используется для позиционирования записывающей головки. Пользовательская информация пишется на отформатированные дорожки.

В качестве металлического слоя в них используется золотое напыление. Этот металлический слой сверху покрыт тонким слоем органического красителя. При записи луч лазера в нужных местах нагревает краситель, при этом краситель темнеет и меняет свою отражательную способность.

Особенность записи: записывать дорожку можно только целиком, а не отдельными секторами, т.е. при записи устройство является устройством последовательного типа.

Диски CD-RW. В них для записи используется изменение состояния вещества под воздействием температуры – переход из аморфного состояния в кристаллическое и наоборот. При этом меняется отражательная способность поверхности, что и используется при чтении.

Диски DVD. В момент создания это название трактовалось как диск для цифровой видеозаписи (Digital Video Disk). Потом трактовка изменилась на Digital Versatile Disk – универсальный цифровой диск. Диски типа DVD – это результат совершенствования дисков CD-RW с целью увеличения плотности записи и емкости диска при тех же размерах – диаметр 120 мм, толщина 1,2 мм. Поперечная плотность записи увеличена вдвое за счет уменьшения вдвое ширины дорожки. Продольная плотность записи также увеличена вдвое (размер битовой ячейки уменьшен вдвое). Кроме того, вместо одной поверхности используется «бутерброд» из двух или четырех рабочих поверхностей. Конструкция двухслойного диска:



А – луч лазера сфокусирован на верхнем отражающем слое, Б – на нижнем полуотражающем слое. DVD диски выпускаются в различных вариантах: ROM, DVD-R (1997г.) DVD-RAM (1999г.), DVD-RW, DVD+RW.

Основные характеристики ОД: емкость – сотни МВ, гигабайты; низкая удельная стоимость; высокая надежность – срок гарантируемого хранения информации от 30 до 100 лет (для CD-RW – 5-10 лет); высокая плотность записи – в десятки раз выше, чем для МД, особенно поперечная – расстояние между дорожками порядка 0,74 мкм.

Общий недостаток – большое время доступа: сотни мс, среднее время доступа порядка 0,5 сек.

Тема 13. Принтеры и способы печати

Методы регистрации текстовой информации

Устройства регистрации текстовой информации (печатающие устройства, принтеры) предназначены для вывода текстовой информации на обычную или специальную бумагу. Современные печатающие устройства способны регистрировать не только символьную, но и графическую информацию. В них используются различные методы (способы) регистрации, т. е. нанесения выводимого изображения на бумагу.

Текстовая информация в памяти ЭВМ обычно хранится в виде ASCII-кодов символов. Поэтому принтер на основе ASCII-кодов выводимых символов должен обеспечивать формирование изображений символов и нанесение их на бумагу – печать символов.

Для формирования изображения символов применяется два способа: непрерывный (знакопечатающий, литерный) и дискретный (знакосинтезирующий, матричный).

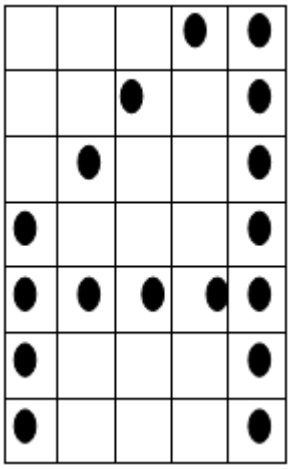
Непрерывный способ обеспечивает формирование изображения символа из непрерывных линий. Существует в двух разновидностях.

1. Траектория движения регистрирующего органа повторяет контуры изображения символа. Так пишет человек и струйный принтер – одна из его разновидностей.

2. Непрерывное изображение символа в виде литеры используется в ударных принтерах для нанесения (выдавливания) краски из ленты на бумагу – за один удар непрерывное изображение символа переносится на бумагу. Множество всех символов (литер) располагается (хранится) в шрифтоносителях. Организация шрифтоносителей, используемых в ударных принтерах, может быть разнообразной. Наибольшее распространение имели: цилиндрические и сферические головки, лепестковые (в виде “ромашки”), барабанные, ленточные.

Знакосинтезирующий, матричный способ обеспечивает формирование изображения символа из точек – дискретное изображение. Для изображения символа используется знакоместо: прямоугольная матрица из nxm элементов. Точки (или другие элементы) в клетках матрицы располагаются таким образом, чтобы получилось изображение символа.

Достоинства: универсальность метода – возможность вывода не только текстовой, но и графической информации, т.е. гибкость при формировании изображения, возможность формирования цветного изображения, электронный шрифтоноситель. Недостатки: невысокое качество печати – дискретные линии символа.



Улучшить качество можно путем увеличения размерности матрицы mxn при тех же размерах знакоместа, т.е. недостаток достаточно просто преодолевается путем увеличения разрешающей способности – количества точек изображения на мм (дюйм). В силу своей универсальности матричный метод используется практически во всех современных принтерах: ударных, струйных, лазерных и др.

Все методы нанесения изображения символа на бумагу разделяются на ударные и не ударные (без контактные и слабо контактные).

В ударных принтерах нанесение изображения на бумагу осуществляется в момент удара по бумаге через красящую ленту. В момент удара часть краски с красящей ленты переносится на бумагу. Достоинства ударного метода: простота, высокое качество печати (для полнопрофильных принтеров). Недостатки: высокий уровень шума, невысокая скорость печати (в силу механической природы метода).

В принтерах неударного типа изображение символов переносится на бумагу различными без контактными или слабо контактными способами.

В струйных полнопрофильных принтерах, например, изображение символа на бумагу наносится тонкой струей капелек жидкого красителя. Траектория движения струи повторяет контуры символа. Струя красителя формируется путем «выстреливания» (с высокой частотой порядка 10 кГц) капелек красителя через узкое отверстие – сопло.

Достоинства способа: высокое качество печати, умеренное (среднее) быстродействие, отсутствие шума, возможность формирования цветного изображения. Недостатки: краска, оставшаяся в узком отверстии сопла, может высохнуть и забить отверстие; высокая стоимость устройства (по сравнению с ударными принтерами).

В термических принтерах изображение символа на бумагу переносится слабо контактным способом – путем локального нагревания специальной термобумаги, которая под действием высокой температуры становится не белой, темнеет. Для нагревания бумаги используются специальные термоголовки. Достоинства: отсутствие шума. Недостатки: низкое качество печати, специальная и, следовательно, более дорогая бумага.

В лазерных принтерах для перенесения изображения символов на бумагу также используется не ударный метод (технология) – т.н. электрофотографический метод. Этот метод основан на физическом явлении локального разрушения электростатического заряда, созданного на поверхности фотопроводника, под воздействием луча света, излучаемого, например, лазерным источником света (отсюда название принтеров). Технология: поверхность из фотопроводника заряжается статическим электричеством (равномерно). Источником зарядов (ионов) является коронный разряд. Затем путем воздействия луча света заряды на некоторых участках поверхности рассеиваются. Тем самым создается “скрытое” изображение символов информации в виде оставленных на поверхности электрических зарядов разных уровней. Затем это ‘скрытое’ изображение проявляется путем осаждения (притягивания) на положительно заряженных участках поверхности отрицательно заряженных частиц краски. “Проявленное” таким образом скрытое изображение переносится на бумагу термосиловым способом: бумага прижимается к поверхности и часть краски притягивается (прилипает) к заряженной бумаге, а затем путем нагревания закрепляется на ней.

Достоинства: высокое качество печати, высокое быстродействие, можно формировать цветное изображение. Недостатки: сложность технологии и устройства, следовательно, его высокая стоимость.

Термосублимация (возгонка) – это быстрый нагрев красителя, когда минуется жидкая фаза. Из твёрдого красителя сразу образуется пар. Чем меньше порция, тем больше фотографическая широта (динамический диапазон) цветопередачи. Пигмент каждого из основных цветов, а их может быть три или четыре, находится на отдельной (или на общей многослойной) тонкой лавсановой ленте (термосублимационные принтеры фирмы Mitsubishi Electric). Печать окончательного цвета происходит в несколько проходов: каждая лента последовательно протягивается под плотно прижатой термоголовкой, состоящей из множества термоэлементов. Эти последние, нагреваясь, возгоняют краситель. Точки, благодаря малому расстоянию между головкой и носителем, стабильно позиционируются и получаются весьма малого размера.

К серьёзным проблемам сублимационной печати можно отнести чувствительность применяемых чернил к ультрафиолету. Если изображение не покрыть специальным слоем, блокирующим ультрафиолет, то краски вскоре выцветут. При применении твёрдых красителей и дополнительного ламинирующего слоя с ультрафиолетовым фильтром для предохранения изображения, получаемые отпечатки не коробятся и хорошо переносят влажность, солнечный свет и даже агрессивные среды, но возрастает цена фотографий. За полноцветность сублимационной технологии приходится платить большим временем печати каждой фотографии.

Тема 14. Сканеры и технология распознавания символов

Устройства автоматического ввода текстовой информации

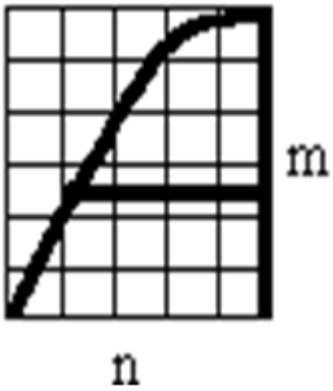
Устройства автоматического ввода текстовой информации – читающие автоматы (ЧА) – предназначены для автоматического ввода текста с первичного документа – оригинала, привычного для человека. Работа ЧА обычно основана на оптическом

восприятии (чтении) информации с листа: интенсивность отраженного от листа светового потока различна на различных участках бумаги – чистых (белых) и не “чистых”.

ЧА автомат должен выполнять следующие функции:

- осмотр и восприятие изображения. Цель – формирование электрического сигнала, соответствующего изображению символа на бумаге, т.е. преобразование информации из графической формы в электрическую. Результат первого этапа - первичное описание символов текста;
- выделение существенных признаков из первичного описания, т.е. формирование вторичного описания символов. Цель - подготовка к распознаванию;
- распознавание символа. Цель этапа – распознавание и кодирование символов информации. При этом каждому опознанному символу ставится в соответствии код (ASCII-код обычно).

Осмотр и восприятие изображения символа. В процессе осмотра символа производится преобразование графического (геометрического) изображения символа в форму электрических сигналов, а также его дискретизация и оцифровка. С целью дискретизации поле символа разбивается на элементарные прямоугольники, т.е. представляется в виде прямоугольной сетки размерностью mxn элементов. При оптическом осмотре этой сетки фиксируется интенсивность света, отраженного от каждой ячейки сетки с помощью фотоприемника (ФП), в котором отраженный световой поток преобразуется в электрический сигнал пропорциональный отраженному световому потоку, т.е. ФП осуществляет собственно восприятие элемента изображения символа. Аналогичные процессы протекают и при попадании изображения на сетчатку глаза человека. Далее производится измерение уровня электрического сигнала при помощи аналого-цифрового преобразователя (АЦП) – “оцифровка”. В результате оцифровки каждому элементу изображения ставится в соответствие n-разрядный двоичный код (число). Опрос всех ячеек сетки производится путем их последовательного осмотра в определенном порядке, путем сканирования. Глаз человека воспринимает изображение также путем сканирования. Совокупность кодов, полученных в результате опроса, представляет собой матрицу, состоящую из n x m кодов, и называется первичным описанием изображения символа. Следует отметить, что в простейшем случае кодирование может осуществляться по принципу: есть отраженный свет – 1, нет – 0. В этом случае АЦП фактически не требуется и первичное описание представляется матрицей, элементы которой заполнены нулями и единицами: каждому элементу изображения ставится в соответствие бит (двоичный код длиной n=1).



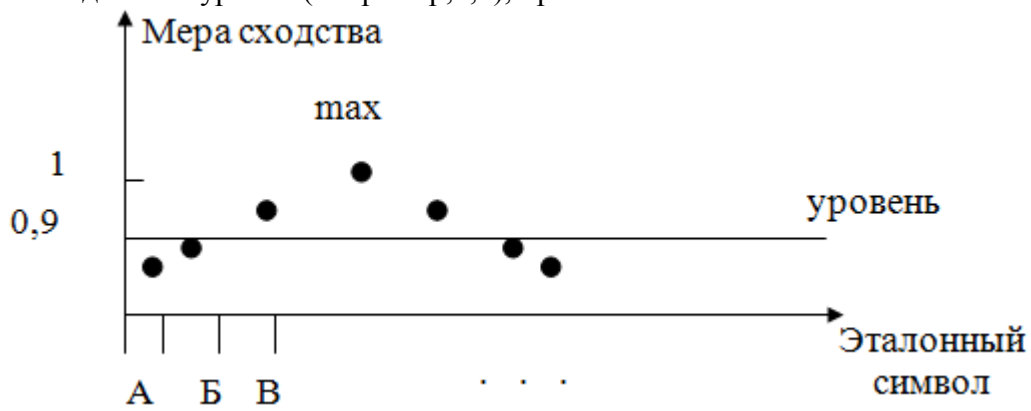
Получив первичное описание одного символа, ЧА переходит к осмотру и восприятию другого символа, и т.д. Этот переход осуществляется одним из двух способов: либо путем перемещения документа (листа) в позицию осмотра следующего символа, либо путем перемещения ‘читающей головки’ автомата в следующую позицию. Второй способ надежнее и проще, поскольку перемещение тонкого и, следовательно, гибкого листа бумаги в двух плоскостях - задача более сложная, чем перемещение

читающей головки с фотоприемником. Второй способ обычно используется в устройствах, которые предназначены для восприятия информации с неподвижного листа бумаги, в сканерах.

Следует отметить, что объем информации в первичном описании символа велик и избыточен (зависит от размеров символа). Для распознавания символа большая часть этой информации не используется. Поэтому перед распознаванием символа производится выделение существенных признаков из первичного описания.

Выделение существенных с точки зрения распознавания признаков осуществляется путем анализа первичного описания и выделения из него геометрических элементов изображения: отрезков прямых (векторов), дуг и т.п. Например, символ А содержит три отрезка. При распознавании символа учитывается также их взаимное расположение элементов символа – топология. Результатом второго этапа является вторичное (сжатое) описание, объем которого существенно меньше первичного. Можно сказать, что на втором этапе происходит ‘сжатие’ избыточного первичного описания путем удаления из него несущественной для распознавания (избыточной) информации.

Распознавание символа осуществляется путем сравнения вторичного описания символа с эталонными описаниями всех символов алфавита, хранимыми в памяти ЧА. В процессе сравнения определяются (вычисляются) меры сходства для всех символов алфавита. Эталон, для которого мера сходства оказалась максимальной и большей некоторого заданного уровня (например,0,9), принимается за истинный.



Дефекты при написании символов, дефекты бумаги, плохой почерк и т.п. факторы приводят к тому, что ЧА не обеспечивают 100% распознавания символов. Поэтому ЧА принято характеризовать:

- вероятностью (частотой) ошибок распознавания, т.е. относительным числом неправильных решений;
- вероятностью (частотой) отказов от распознавания, т.е. относительным числом символов, для которых ЧА не находит эталона.

Следует отметить, что эти характеристики существенно выше у ЧА, предназначенных для чтения машинописных текстов, чем у ЧА, предназначенных для чтения рукописных текстов. В последнем случае эти две характеристики можно существенно улучшить, если вместо рукописных использовать специальные шрифты: кодированные, стилизованные, нормализованные. Пример специального шрифта – почтовые конверты, на которых в специальном поле указывается почтовый индекс.

В качестве примера ниже приведены характеристики ЧА, который выпускался серийно в нашей стране в прошлые годы. ЧА Бланк-3 (ЕС 6031): скорость чтения – до 400 док/мин; частота отказов от распознавания – 0,01% (каждый сотый символ текста); частота ошибок – 0,001%; формат документа А4 (210х297); шрифт стилизованный рукописный.

Современное состояние. ЧА как единое устройство не выпускается. Серийно выпускаются сканеры, которые обеспечивают только первую функцию читающего автомата – осмотр и восприятие изображения. Сканеры каждой точке (элементу)

изображения ставят в соответствие код длиной n разрядов, т.е. первичное описание информации и ввод его в память ЭВМ.

Выделение существенных признаков и распознавание символов текста осуществляется на программном уровне, т.е. путем выполнения специальных программ, реализующих вторую и третью функции ЧА. Современный ЧА – это сканер (аппаратная часть ЧА) и ПО, которое распознает символы текста ставит им в соответствие ASCII-коды.

Организация сканеров

Сканер как устройство ввода обеспечивает дискретизацию графического изображения (разделение на элементы - растрирование), оцифровку элементов изображения и ввод их в память ЭВМ. Источником информации служит носитель информации (лист бумаги или слайд), привычный для человека, содержащий текст, рисунок, фотографию и т.п.

Сканер, работающий с непрозрачными документами, воспринимает изображение в виде отраженного от листа светового истока. Сканер, работающий со слайдами, воспринимает световой поток, прошедший сквозь прозрачный носитель информации.

Восприятие всего документа осуществляется путем его сканирования. Сканирование осуществляется путем перемещения считывающей головки сканера относительно документа. В зависимости от способа движения (перемещения) считывающей головки различают сканеры ручные и автоматические. В ручном сканере движение осуществляется человеком вручную, путем “прокатывания” устройства по листу бумаги.

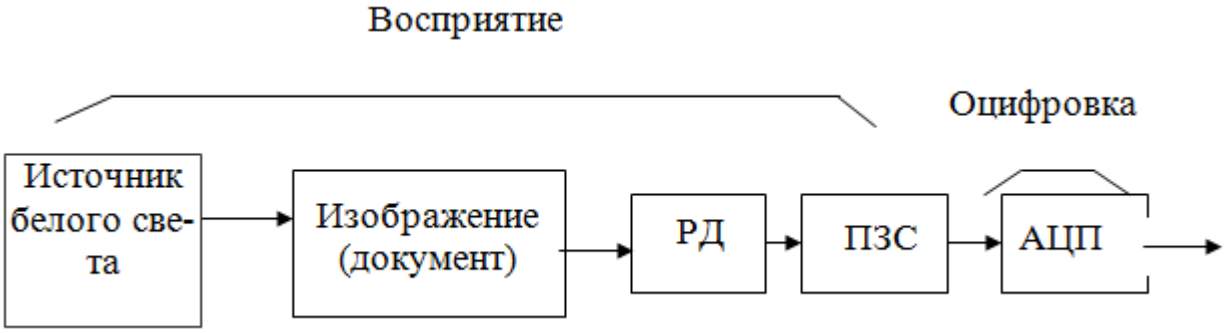
В автоматических сканерах движение обеспечивается встроенным в сканер механизмом перемещения. В зависимости от организации этого механизма автоматические сканеры (настольного типа) разделяются на планшетные, рулонные (барабанные) и проекционные (для чтения слайдов).

В планшетных сканерах носитель (лист бумаги) неподвижен. Сканирование осуществляется путем перемещения считывающей головки приводом, выполненным на основе шагового двигателя.

В рулонных (барабанных) сканерах листы документов с помощью барабана протягиваются сквозь устройство мимо неподвижного считывающего органа, т.е. сканирование осуществляется построчно.

Сканеры различают черно-белые, полутоновые и цветные.

Принцип действия полутоновых сканеров. Вводимое изображение (документ) освещается белым светом. Отраженный от документа свет через уменьшающую (редуцирующую) линзу попадает на фоточувствительный полупроводниковый элемент – прибор с зарядовой связью (т.н. ПЗС-элемент или CCD – Couple-Charged Device). ПЗС–элемент (фототранзистор) накапливает заряд (заряжается) при попадании на него светового потока, т.е. он преобразует фотоны в электрические заряды. Причем ПЗС–элементы обладают высокой квантовой эффективностью – до 95%. Для сравнения, квантовая эффективность глаза человека около одного процента. Квантовая эффективность – это отношение числа накопленных зарядов к числу фотонов. Уровень заряда (напряжения) зависит от интенсивности светового потока и продолжительности освещения, т. е. при постоянном времени освещения он зависит только от светового потока. Если через время $\Delta t = const$ измерить уровень напряжения, то по нему можно судить об интенсивности светового потока. Таким образом, ПЗС–элемент каждому элементу изображения – пикселу – ставит в соответствие значение напряжения, которое затем при помощи АЦП преобразуется в двоичный код (оцифровывается). Упрощенная структура сканера представлена на рисунке. Здесь РД – редуцирующая (уменьшающая) линза, ПЗС–элемент, АЦП – аналого-цифровой преобразователь.



Разрядность АЦП n определяет разрешающую способность сканера при восприятии им интенсивности светового потока, т.е. разрешающую способность при восприятии полутонового изображения. Например, при n=8 сканер обеспечивает восприятие 256 уровней интенсивности света: от min=0 до max=255. Следует отметить, что вместо одного ПЗС – элемента обычно используется линейка или матрица ПЗС - элементов. Это позволяет упростить механизм сканирования (движение – только в одном направлении) и ускорить процесс восприятия.

Цветные сканеры отличаются от полутоновых лишь тем, что изображение (документ) освещается не белым светом, а тремя цветами: красным (R), зеленым (G), синим (B). В простейшем случае освещение осуществляется последовательно: сначала красным, потом зеленым, а затем синим светом, т.е. восприятие производится за три прохода одного документа, т. е. медленно. Выход: параллельное (одновременное восприятие) сразу по трем каналам (от трех источников света). При этом затраты оборудования утраиваются: три фильтра, три ПЗС–структуры (три линейки из ПЗС-элементов для параллельного восприятия строки), три АЦП. Следует отметить, что ПЗС-элементы в линейке организованы в виде регистра сдвига, к крайнему элементу которого (левому или правому) подключается усилитель и АЦП.

Таким образом, в цветном сканере каждому элементу изображения ставится в соответствие три кода: код R, код G, код B.

Пример: цветной сканер фирмы HP имеет в своем составе трехполосную ПЗС-структуру, на каждую из полос которой через специальный фильтр подается свет R, G, B (К, З, С), и восприятие изображения осуществляется за один проход. Размер ПЗС-элемента в линейке (шаг) 8 мкм. Этот параметр и определяет вместе с редуцирующей линзой разрешающую способность сканера - количество точек (пикселей) на дюйм – dpi (ppi). Пример: лист формата А4, ширина листа 210 мм, разрешающая способность 12,5 точек на мм. Размер линейки ПЗС-элементов – 8мкм х 210мм х 12.5 = 21000мкм = 21мм. Поэтому РД должна уменьшить изображение строки элементов изображения (точек) в 10 раз, т. е. сфокусировать строку точек длиной 210мм на линейку ПЗС-элементов длиной 21мм.

Следует отметить, что в последнее время в сканерах вместо ПЗС-элементов для восприятия информации стали использовать КДИ-элементы (Contact Image Sensor - CIS – контактный датчик изображения). Основой КДИ-элемента также является фототранзистор, только выполненный по МОП-технологии (металл – окисел – полупроводник). КДИ-элементы в устройствах типа ФАКС используются давно.

Отличительные особенности сканеров на основе КДИ-элементов:

- не требуется сложная оптическая система, т. к. размеры КДИ-элементов больше, чем ПЗС-элементов и не нужно использовать редуцирующую линзу, разрешающая способность – 300 dpi,

- простые источники света – светодиоды.

Тема 15. Прочие виды периферийных устройств

Прочие периферийные устройства

Устройства автоматического ввода текстовой информации - читающие автоматы (ЧА) – предназначены для автоматического ввода текста с первичного документа – оригинала, привычного для человека. Работа ЧА обычно основана на оптическом восприятии (чтении) информации с листа: интенсивность отраженного от листа светового потока различна на различных участках бумаги – чистых (белых) и не “чистых”.

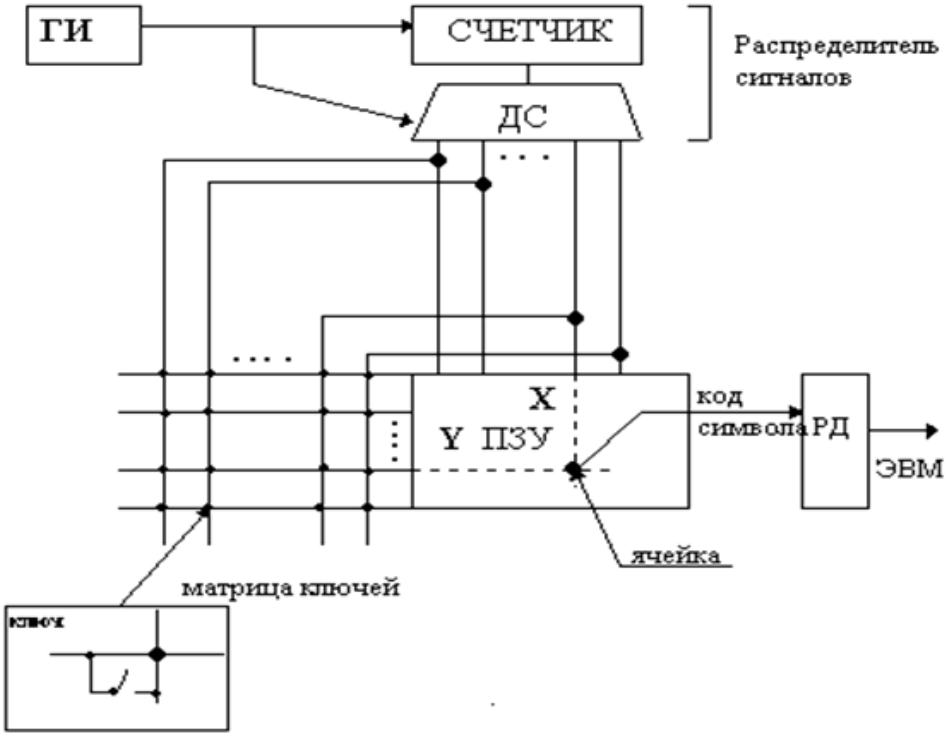
Помимо уже рассмотренных ранее, по функциональному назначению внешние устройства могут быть условно разделены на следующие классы:

1. Устройства ручного ввода и оперативного управления (клавиатура, мышь, джойстик, сенсорные устройства и устройства тактильного ввода);
2. Терминальные средства визуализации, реализованные в ПЭВМ на различных типах дисплеев – от монохромных до графических дисплеев высокой разрешающей способности, световых индикаторах;
3. Средства ввода-вывода речевых сообщений.

Устройство клавиатуры

Назначение клавиатуры – ручной ввод текстовой (символьной) информации. Количество клавиш на клавиатуре всегда меньше количества символов. Так сложилось исторически: для ввода информации в компьютер использовали традиционную клавиатуру пишущих машинок, в которых количество клавиш было примерно вдвое меньше количества символов. Ввод большего количества символов в них обеспечивается специальными режимами: верхний регистр/ нижний регистр. Если каждому символу поставить в соответствие клавишу, то клавиатура станет плохо обозримой, ее трудно будет запомнить и печатать «вслепую» (для увеличения скорости ввода).

Организация клавиатуры. Нажатие клавиши на клавиатуре вызывает замыкание ключа, связанного с этой клавишей (отпускание – размыкание). В момент замыкания ключа блок управления клавиатурой формирует код символа, изображенного на клавише. Простейшая схема кодирования имеет вид, представленный на рисунке.



Распределитель сигналов- состоит из ГИ, счетчика и дешифратора

Работа схемы: распределитель импульсов периодически опрашивает (сканирует) состояние матрицы ключей. Если некоторая клавиша нажата (ключ замкнут), то сигнал с ГИ (импульс) проходит по соответствующим шинам X,Y (вертикальным, горизонтальным) и выбирает одну ячейку ПЗУ, из которой и выбирается код символа. Этот код заносится в РД клавиатуры, а из него в ЭВМ (обычно по сигналу прерывания, по которому вызывается подпрограмма, обеспечивающая ввод кода символа из РД клавиатуры в ячейку памяти ЭВМ). Такой вариант организации, при котором в ПЗУ размещаются две таблицы ASCII-кодов – английская (набор 0) и русская (набор 1) подходит для России, а для международного применения – нет. Поэтому в современных клавиатурах на аппаратном уровне формируется не ASCII-код, а номер нажатой (отжатой) клавиши в матрице ключей – скан-код. Номер клавиши определяется путем сканирования матрицы ключей – отсюда название кода. Формирование же ASCII-кода осуществляется на программном уровне.

Дисплеи

По типу применяемых индикаторов дисплеи делятся на два класса: на основе электронно-лучевых трубок (ЭЛТ) и на основе плоских матричных экранов (ПМЭ).

В силу известных недостатков индикаторов типа ЭЛТ – не плоский экран (до недавнего времени), большие габариты и потребляемая мощность, мерцание, вредные для человека излучения – они со временем должны уступить место индикаторам типа ПМЭ.

В качестве ПМЭ можно использовать тонкоплёночные электролюминисцентные экраны, газоплазменные экраны, экраны на базе матричных светодиодных структур, жидкокристаллические индикаторы (ЖКИ) и др. Наибольшее распространение среди ПМЭ в настоящее время имеют ЖКИ, набирают силу газоплазменные ПМЭ.

Индикаторы на основе ЭЛТ. Отображение точек на экране ЭЛТ (CRT – Catode Ray Tube) осуществляется путем их «высвечивания» электронным «лучом». Экран ЭЛТ с внутренней стороны покрыт слоем люминофора, который начинает светиться под действием электронного луча. Время послесвечения небольшое, поэтому требуется периодическая «подсветка» точек – регенерация изображения.

Для вывода изображения на экран ЭЛТ используется растровый метод – растровая развертка: траектория движения луча постоянна (не зависит от изображения). Луч сначала «бежит» слева – направо по верхней строке матрицы, высвечивая точки изображения. Затем луч вхолостую возвращается в начало следующей строки. После этого он опять бежит слева – направо, вычерчивая строку, и т. д. по всем строкам матрицы. После вычерчивания последней строки растра луч вхолостую возвращается в начало первой строки. Такая траектория движения луча обеспечивает однократное отображение изображения и называется кадром. После этого она периодически повторятся с целью регенерации и обновления картинки.

Для формирования цветных точек используется покрытие экрана люминофором трех цветов – красного, зеленого, синего и три луча. Путем управления интенсивностью трех лучей обеспечивается высвечивание трех точек (субпикселей), которые воспринимаются как одна цветная – пиксель. Пиксель – это совокупность (описаний и изображений) трех точек – красной (red), зеленой (green), синей (blue) как единого целого. Принцип формирования цвета точки – суммирование трех основных цветов (RGB) с разной интенсивностью на черном фоне.

Плоские матричные экраны. ПМЭ – это прямоугольная матрица ячеек (пикселей), с помощью которых формируется растровое изображение. В зависимости от типа ПМЭ его ячейки либо сами испускают свет, либо изменяют свою прозрачность для света от внешнего (постоянного) источника, либо изменяют свою отражательную способность. В ЖКИ изображение формируется от внешнего источника света, в других типах – в плазменных, электролюминисцентных и др. – на основе управляемого процесса испускания света. Свечение пикселей в ПМЭ более устойчиво, чем в дисплеях на основе ЭЛТ, поэтому в ПМЭ отсутствует мерцание, характерное для ЭЛТ.

Жидкокристаллические индикаторы. ЖКИ или ЖК дисплей (LCD - Liquid Crystal Display) состоит из жидкого вещества, которое обладает некоторыми свойствами кристалла. Молекулы этого вещества под действием электрических сигналов могут изменять свои физические свойства и тем самым влиять на световой луч, проходящий сквозь них. Пример такого вещества – бифенил.

Работа ЖКИ основана на физическом явлении поляризации светового потока. Известно, что кристаллы-поляроиды пропускают только ту часть светового потока, вектор электромагнитной индукции которой лежит в плоскости, параллельной оптической плоскости поляроида. Для другой (не параллельной) части светового потока поляроид не будет прозрачным. В результате он как бы «просеивает» световой поток. Этот эффект и называется поляризацией света. Поскольку в жидко-кристаллических веществах молекулы подвижны и чувствительны к внешнему электрическому полю, постольку имеется возможность управлять поляризацией света. При нулевом уровне сигнала угол поляризации обычно равен 90°. При увеличении напряжения изменяется ориентация молекул-поляроидов и угол поляризации светового потока уменьшается до 0° при некотором максимальном напряжении.

Конструктивно ЖКИ представляет собой две прозрачные (стеклянные) пластины, две поляризационные решетки, каждая из которых обеспечивает поворот луча на 90°, а между ними тонким слоем размещается ЖК вещество – управляемый поляроид. На передней и задней пластинах нанесены поляризационные решетки с взаимно перпендикулярным направлением поляризации. Этот «бутерброд» из трех поляроидов освещается источником света, расположенным сзади, если индикатор работает на просвет, или сбоку, если – на отражение. Вся панель разделена на ячейки рядом горизонтальных (прозрачных) электродов на одной пластине и рядом вертикальных электродов – на другой. Между электродами (на их пересечении) образуются фактически конденсаторные пластины, зазор между которыми заполнен ЖК веществом (бифенилом). Вся эта конструкция при нулевом уровне напряжения прозрачна для проходящих лучей света, т. к. четырежды обеспечивает поворот угла поляризации на 90°, т. е. на 360°. Если на пластины

конденсатора подавать электрический потенциал, то изменяется (уменьшается) угол поляризации и уменьшается прозрачность ячейки ЖКИ.

Растровое изображение формируется путем последовательной (строка за строкой) подачи управляющих сигналов на электроды ЖКИ. Сначала подаются сигналы на все ячейки первой (верхней) строки матрицы, затем – второй и т. д. до последней строки. Так создается кадр.

Различают ЖКИ с пассивной и активной матрицей. В пассивной матрице электрическая часть матрицы состоит из электродов. Ячейки (конденсаторы) матрицы имеют достаточно большую емкость, поэтому напряжение на них не может изменяться быстро, отсюда проблема с отображением движущихся изображений. Время обновления экрана с пассивной матрицей – сотни миллисекунд.

В активной матрице в каждую ячейку встраивается усилительный элемент – прозрачный тонкопленочный транзистор (TFT). Он существенно уменьшает время изменения напряжения на обкладках конденсатора и, следовательно, время изменения прозрачности ячеек матрицы. В результате время обновления экрана сокращается до величины порядка 20-30 миллисекунд (20 мс – 50 кадров в сек).

Для формирования цветного изображения используется более сложная конструкция ЖКИ, в которой для формирования каждого пикселя используется три TFT-ячейки (три субпикселя), по одной на каждый пиксель, формируемый из трех цветов RGB.

Плазменные экраны (панели) PDP – Plasma Display Panel. Принцип отображения базируется на использовании электрического разряда в газе (газ ксенон или неон). Разряд – холодный. В результате разряда газ переходит в состояние холодной плазмы. Пиксель матрицы организуется на основе трех элементов, обеспечивающих высвечивание трех цветов R,G,B. Пиксели, как и в ЖКИ, образуются на пересечении прозрачных проводников, промежуток между которыми заполнен газом. Для того, чтобы «зажечь» ячейку, на электроды подается высокое напряжение, вызывающее разряд в газе и ультрафиолетовое излучение. Флюорисцирующее покрытие в зоне разряда под действием ультрафиолетового излучения начинает излучать свет в видимом диапазоне. Интенсивность, яркость свечения зависит от интенсивности ультрафиолетового излучения, которое, в свою очередь, зависит от величины управляющего напряжения.

Дисплеи на основе **светоизлучающих полимерных полупроводников** материалов (LEP – Light Emitting Plastics). Если такой материал поместить между электродами и подать напряжение (не долее 3 вольт), он начинает излучать свет. Основные преимущества LEP-панелей: простота конструкции и дешевизна производства. Основной недостаток – небольшой (пока) срок службы, порядка полутора лет при температуре 80°C. Если панель поместить в специальный корпус, срок возрастает до 5 лет.

Полевые эмиссионные панели FED (Field Emission Display). Их еще называют плоскими ЭЛТ, т. к. в них как и в ЭЛТ используется люминофор. В них свечение каждого пикселя вызывается электронами, которые излучаются с помощью нескольких тысяч субмикронных остроконечных элементов поверхности, напротив которой расположен люминофор. При этом не требуется напряжений высокого уровня, как в ЭЛТ. Основное достоинство - отсутствие вредных излучений, основной недостаток – небольшой срок службы.

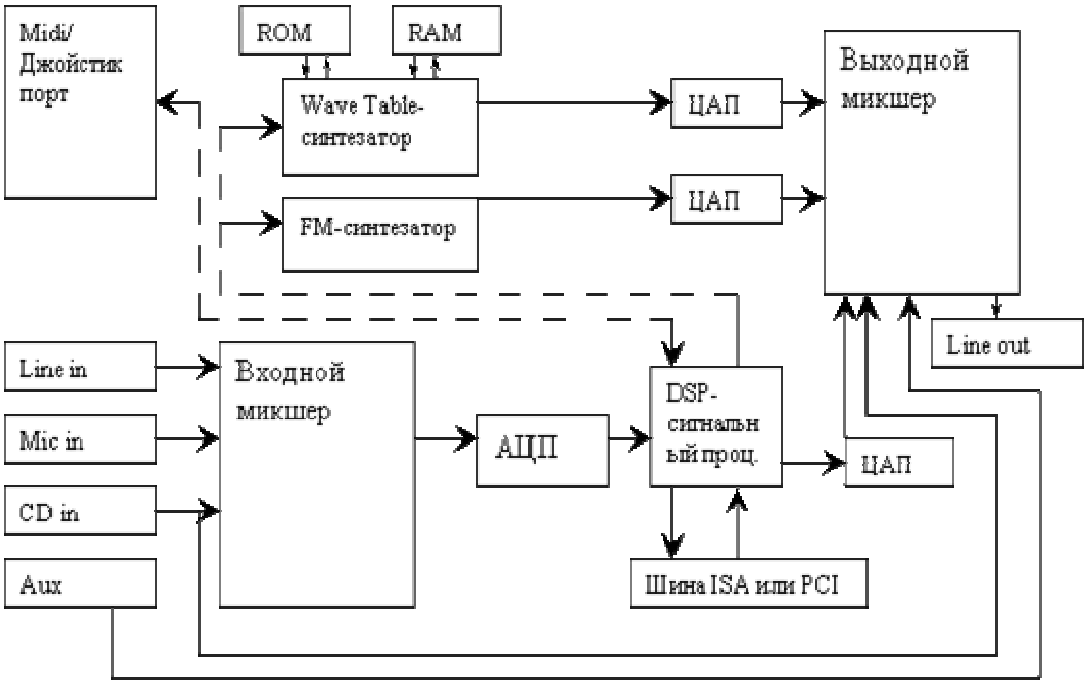
В зависимости от применяемого метода вывода графических изображений различают: **растровые ГД** (с постоянным сканированием), в которых реализуется матричный метод вывода изображений, и **ГД с произвольным сканированием**, в которых реализуется векторный метод вывода изображений (траектория движения луча зависит от изображения).

Звуковая информация

Исходная форма звукового сигнала – непрерывное изменение амплитуды во времени – представляется в цифровой форме с помощью перекрестной дискретизации по времени

и по уровню. Одновременно с временной дискретизацией выполняется амплитудная – измерение мгновенных значений амплитуды и их представление в виде числовых величин. Полученный поток чисел (серии двоичных чисел) называют импульсно-кодовой модуляцией – PCM.

Устройство звуковой карты:



Line in, Mic in – линейный и микрофонный входы.

Aux: сигнал с этого входа минует все устройства и сразу идет на выход.

CD in используется для CD-ROM.

У всех разъем mini-Jack.

На задней панели платы есть 15-пиновый разъем midi/джойстик порта, используется для подключения синтезаторов, клавиатур или джойстика.

Все сигналы с внешних аудиоустройств поступают во входной микшер, он служит для усиления.

АЦП – аналогово-цифровой преобразователь. Замеряет амплитуду поступающего сигнала и кодирует соотношения.

ЦАП – цифро-аналоговый преобразователь. Заменяет коды, преобразует в аналоговый сигнал.

DSP-сигнальный процессор управляет обменом данных со всеми остальными устройствами компьютера через шину ISA или PCI.

Синтезатор – имитация музыкальных инструментов.

FM (Frequency Modulation – частотная модуляция) синтезатор для сохранения совместимости с Sound Blaster.

Wave Table – синтезатор для получения качественного звука.

RAM – оперативная память используется для загрузки звука.

ROM – постоянная память, в ней хранятся образцы звучания.

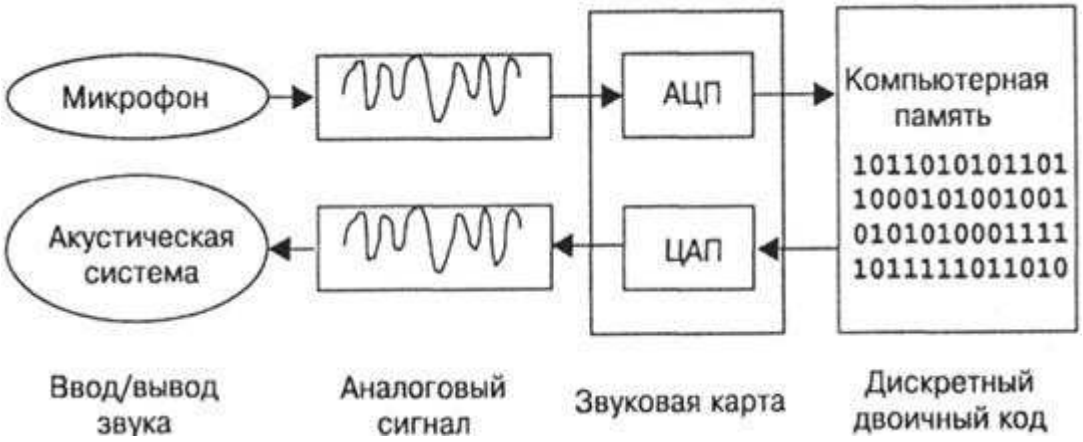
Устройства вывода звуковой информации: колонки, наушники, электроакустические аппараты для воспроизведения речи, музыки и прочее.

По способу звукоизлучения различают: рупорные (наиболее распространены, т. к. обладают большей отдачей), безрупорные.

К устройствам ввода звуковой информации относятся микрофоны. Эти устройства преобразуют звуковые колебания в электрические.

Микрофон используется для ввода звука в компьютер. Непрерывные электрические колебания, идущие от микрофона, преобразуются в числовую последовательность. Эту

работу выполняет устройство, подключаемое к компьютеру, которое называется аудиоадаптером, или звуковой картой. Воспроизведение звука, записанного в компьютерную память, также происходит с помощью аудиоадаптера, преобразующего оцифрованный звук в аналоговый электрический сигнал звуковой частоты, поступающий на акустические колонки или стереонаушники. Из сказанного следует, что звуковая карта совмещает в себе функции ЦАП и АЦП. Рисунок иллюстрирует описанный процесс.



Методическое пособие для лабораторных работ по курсу «Организация ЭВМ и периферийные устройства»

Оглавление

1.ВВЕДЕНИЕ В АРХИТЕКТУРУ ЭВМ.

1.1.Структура персонального компьютера.

Под термином “Архитектура ЭВМ” - будем понимать представление о машине в терминах функциональных основных модулей, языка, структуры данных (без деталей их аппаратной реализации).

С точки зрения пользователя, можно выделить следующие группы характеристик, определяющие архитектуру ЭВМ:

Структура машинного кода и система команд, определяющие алгоритмические возможности процессора.

Технические и эксплуатационные характеристики ЭВМ: производительность, емкость памяти, тактовая частота, стоимость и т.д.

Состав и характеристики функциональных модулей базовой конфигурации ЭВМ, наличие возможности подключения дополнительных модулей.

Структура и состав системного программного обеспечения и принципы его взаимодействия с аппаратурой.

Структурная схема ПК приведена на рисунке 1.1.

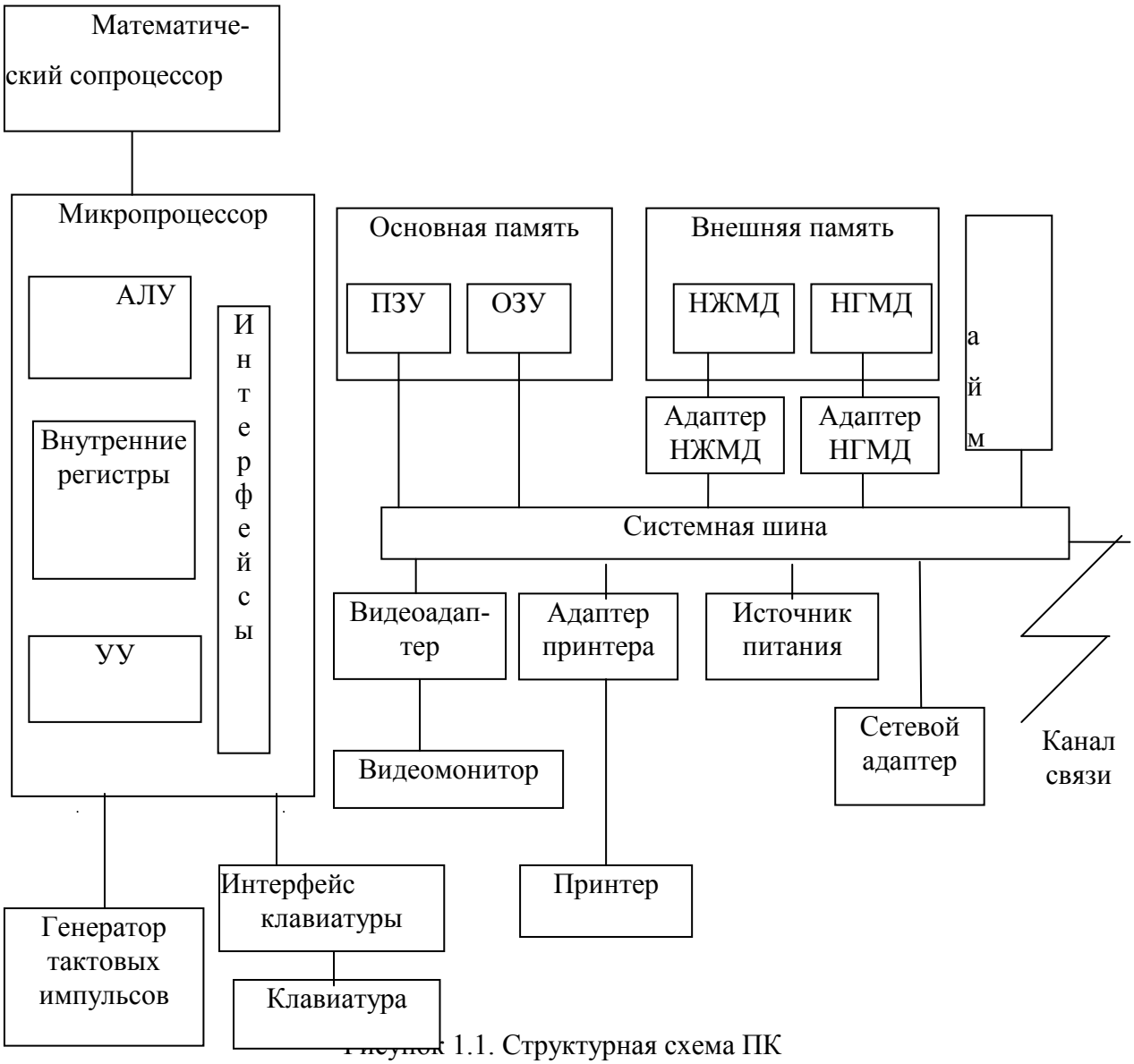


Рисунок 1.1. Структурная схема ПК

Микропроцессор (МП) – центральный блок ПК, предназначенный для управления работой всех блоков машины и для преобразования информации, то есть. выполнения арифметических и логических операций над информацией. Микропроцессор состоит из следующих схем:

- Арифметико-логическое устройство (АЛУ) предназначено для выполнения всех арифметических и логических операций над числовой и символьной информацией, для ускорения операций с вещественными числами к АЛУ подключается математический сопроцессор.
- *Внутренние регистры* – служат для кратковременного хранения, записи и выдачи информации, непосредственно используемой в вычислениях в ближайшее время работы ПК. Регистры – быстродействующие ячейки памяти различной длины (8÷64 разрядов).
- *Устройство управления* (УУ) – формирует и подает на все блоки машины символы управления (управляющие символы); формирует адреса ячеек памяти и передает их для выполнения операции.
- *Интерфейс МП* – реализует сопряжение и связь с другими устройствами ПК; включает в себя связь между устройствами МП, буферные регистры, схемы управления портами ввода-вывода и системной шиной.

Интерфейс – совокупность средств сопряжения и связи устройств компьютера, обеспечивающая их эффективное взаимодействие. Порт ввода-вывода – аппаратура сопряжения, позволяющая подключить к микропроцессору другое устройство ПК.

Генератор тактовых импульсов генерирует последовательность электрических импульсов; синхронизирующих работу всех устройств ПК; частота генерируемых импульсов определяет тактовую частоту ПК, которая является одной из основных характеристик ПК.

Такт – промежуток времени между соседними импульсами.

Системная шина – основная интерфейсная система ПК.

Системная шина включает в себя:

- *шину данных*, содержащую провода и схемы сопряжения для параллельной передачи всех разрядов числового кода операнда;
- *шину адреса*, содержащую провода и схемы сопряжения для параллельной передачи всех разрядов числового кода адреса ячейки ОП или порта ввода-вывода внешнего устройства;
- *шину управления*, содержащую провода и схемы сопряжения для передачи управляющих сигналов во все блоки ПК;

• *шину питания*, имеющую провода и схемы сопряжения для подключения блоков ПК к системе электропитания.

Системная шина обеспечивает 3 направления передачи информации:

между МП и ОП;

между МК и портами ввода-вывода внешних устройств;

между ОП и портами ввода-вывода внешних устройств (в режиме прямого доступа к памяти).

Все блоки ПК (точнее их порты ввода-вывода) через соответствующие унифицированные разъемы подключаются к шине единообразно: непосредственно или через *адаптеры.*

Управление системной шиной осуществляется либо непосредственно, либо через *контроллер шины*. Обмен информацией между внешними устройствами и системной шиной выполняется с использованием ASCII-кодов.

Функции хранения информации выполняет память внутренняя и внешняя.

Основная память (ОП) – предназначена для хранения и оперативного обмена информацией с другими блоками ПК. ОП состоит из ПЗУ и ОЗУ. В последнее время в состав ОП входят также КЭШ-память и FLASH- память

ПЗУ – служит для хранения постоянной программной и справочной информации, позволяет оперативно только считывать хранящуюся в нем информацию (изменить информацию в ПЗУ нельзя).

ОЗУ – предназначено для оперативной записи, хранения и считывания информации (программ и данных), непосредственно участвующей в информационно-вычислительном процессе, выполняемом ПК в текущий период времени.

Внешняя память – используется для долговременного хранения любой информации, которая может когда-либо потребоваться. Во внешней памяти хранится все программное обеспечение ПК.

Источник питания – блок, содержащий системы автономного и сетевого энергопитания ПК.

Таймер – внутримашинные электронные часы, обеспечивающие автоматический съем значения текущего момента времени (год, месяц, день, часы, минуты, секунды и доли секунды).

Таймер подключается к автономному источнику питания – аккумулятору и при отключении ПК от сети продолжает работать.

Внешние устройства (ВУ) – обеспечивают взаимодействие ПК с окружающей средой: пользователями, объектами управления и другими ЭВМ.

ВУ разделяются на следующие классы:

- внешние запоминающие устройства (ВЗУ);
- устройства ввода информации;
- устройства вывода информации;
- средства связи и телекоммуникации.

Устройства ввода информации:

- клавиатура;
- графические планшеты (диджитайзеры) – ручной ввод графической информации путем перемещения по планшету специального указателя (пера);
- сканеры;
- манипуляторы (джойстик – рычаг, мышь, трекбол – шар в оправе, световое перо – для ввода графической информации на экран дисплея путем управления движением курсора по экрану);
- сенсорные экраны – для ввода отдельных элементов изображения, программ или команд в ПК;
- устройство речевого ввода-вывода – быстроразвивающиеся средства мультимедиа;
- видеомонитор – устройство для отображения вводимой и выводимой информации.

Устройства вывода информации:

- принтеры – печатающие устройства для регистрации информации на бумажный носитель.
- графопостроители (плоттеры) – устройства для вывода графической информации из ПК на бумажный носитель. Векторные плоттеры – вычерчивают изображение при помощи пера. Растровые бывают электростатические, струйные и лазерные. По конструкции плоттеры подразделяются на планшетные и барабанные.

Устройства связи и телекоммуникации используются для связи с другими приборами и подключения ПК к каналам связи и другим ЭВМ и вычислительным сетям (сетевые интерфейсные платы, мультиплексоры передачи данных, модемы).

Сетевой адаптер – служит для подключения ПК к каналу связи для работы в составе вычислительной сети. В глобальных сетях функции сетевого адаптера выполняет модулятор-демодулятор (модем).

Средства мультимедиа (multimedia – многосредовость) – комплекс аппаратных и программных средств, позволяющих человеку общаться с ПК, используя самые разные , естественные для себя среды: звук, видео, графику, тексты, анимацию и др.

К средствам мультимедиа относятся: устройства речевого ввода-вывода, сканеры, высококачественные видео- и звуковые платы, платы видео-захвата (снимают изображения с видеомагнитофона или видеокамеры и вводят его в ПК), высококачественные акустические и видеовоспроизводящие системы с усилителями, звуковыми колонками, большими видеоэкранами. К средствам мультимедиа относят также внешние запоминающие устройства большой емкости на оптических дисках, часто используемые для записи звуковой и видео информации.

Математический сопроцессор – используется для ускоренного выполнения операций над числами с плавающей запятой. Имеет свою систему команд и работает параллельно с основным МП, но под его управлением. Последние модели МП, начиная с МП 80486DX, включают сопроцессор в свой состав в качестве устройства с плавающей точкой (FPU).

Контроллер прямого доступа к памяти освобождает МП от прямого управления НМД, что существенно повышает эффективность ПК.

Сопроцессор ввода-вывода – за счет параллельной работы с МП значительно ускоряет выполнение процедур ввода-вывода при обслуживании нескольких внешних устройств, освобождая МП от обработки процедур ввода-вывода, в том числе реализует и режим прямого доступа к памяти.

Контроллер прерываний – обслуживает процедуры прерывания, принимает запрос на прерывание от ВУ, определяет уровень приоритета этого запроса и выдает сигнал прерывания в МП. МП, получив этот сигнал, приостанавливает выполнение текущей программы и переходит к выполнению специальной программы обслуживания того прерывания, которое запросило ВУ. После завершения программы обслуживания восстанавливается выполнение прерванной программы. Контроллер прерываний является программируемым.

Конструктивно ПК выполнены в виде центрального системного блока, к которому через разъемы подключаются ВУ: дополнительная память, клавиатура, дисплей, принтер и т.д.

Системный блок обычно включает в себя системную плату (материнскую плату), блок питания, накопители на дисках, разъемы для дополнительных устройств и *платы расширения* с контроллерами-адаптерами ВУ.

На системной плате размещаются:

- микропроцессор;
- математический сопроцессор;
- генератор тактовых импульсов;
- блоки (микросхемы) ОЗУ и ПЗУ;
- адаптеры клавиатуры, НЖМД, НГМД ;

• контроллер прерываний;

• таймер.

Микропроцессоры.

Микропроцессор (центральный процессор или CPU) – функционально законченное программно-управляемое устройство обработки информации, выполненное в виде одной или нескольких больших (БИС) или сверхбольших (СБИС) интегральных схем.

Функции МП.

Чтение и дешифрация команд из ОП.

Чтение данных из ОП и регистров адаптеров ВУ.

Прием и обработка запросов и команд от адаптеров на обслуживание ВУ.

Обработка данных и их запись в ОП и регистры адаптеров ВУ.

Выработка управляющих сигналов для всех узлов и блоков ПК.

Разрядность шины данных МП определяет разрядность ПК в целом; разрядность шины адреса МП – его адресное пространство.

Адресное пространство – это максимальное количество ячеек ОП, которое может быть непосредственно адресовано микропроцессором.

Первый МП был выпущен в США в 1971 году фирмой Intel – МП 4004. В настоящее время МП фирмы Intel и Intel-подобные являются наиболее популярными и распространенными.

Все МП можно разделить на 3 группы:

МП типа CISC (с полным набором команд);

МП типа RISC (с сокращенным набором команд);

МП типа MISC (с минимальным набором команд и весьма высоким быстродействием) – находятся в стадии разработки.

Большинство современных ПК типа IBM PC используют МП типа CISC.

Основные модели процессоров фирмы INTEL приведены в таблице.

Модель МП	Разрядность, бит		Тактовая частота, МГц	Адресное пространство, байт	Число команд	Число элтов	Год выпуска
	Данных	дреса					

4004	4		4,77	4×10^3	5	2300	1971
8080	8		4,77	64×10^3		10000	1974
8086	16	6	4,77 и 8	10^6	34	70000	1982
8088	8, 16	6	4,77 и 8	10^6	34	70000	1981
80186	16	0	8 и 10	10^6		140000	1984
80286	16	4 (20)	10-33	4×10^6 (4×10^9)		180000	1985
80386	32	2	25-50	16×10^6 (4×10^9)	40	275000	1987
80486	32	2	33-100	16×10^6 (4×10^9)	40	$1,2 \times 10^6$	1989
Pentium	64	2	50-150	4×10^9	40	$3,1 \times 10^6$	1993
Pentium Pro	64	2	66-200	4×10^9	40	$5,5 \times 10^6$	1995

Особенности некоторых процессоров:

Начиная с МП 80286 имеется возможность многозадачной работы и работы МП в вычислительной сети.

Начиная с МП 80386 используется конвейерное выполнение команд, что позволило увеличить быстродействие ПК в 2-3 раза, а также обеспечивается поддержка системы виртуальных машин, при которой в одном МП моделируется как бы несколько ПК, работающих параллельно и имеющих разные ОС.

МП Pentium – это МП 80586(P5), товарная марка Pentium запатентована фирмой Intel. Эти МП имеют пятиступенчатую конвейерную структуру и КЭШ-буфер для команд условной передачи управления, позволяющие предсказать направление ветвления программы; по эффективному быстродействию они приближаются к RISC МП. Обмен данными с системой может выполняться со скоростью 1 Гбайт/с. У всех МП Pentium имеется встроенная КЭШ - память, отдельно для команд, отдельно для данных; имеются специализированные конвей-

ерные аппаратные блоки сложения, умножения и деления, значительно ускоряющие выполнение операций с плавающей запятой.

В качестве примера на рисунке 1.2 приведена упрощенная структурная схема микропроцессора i486.

.

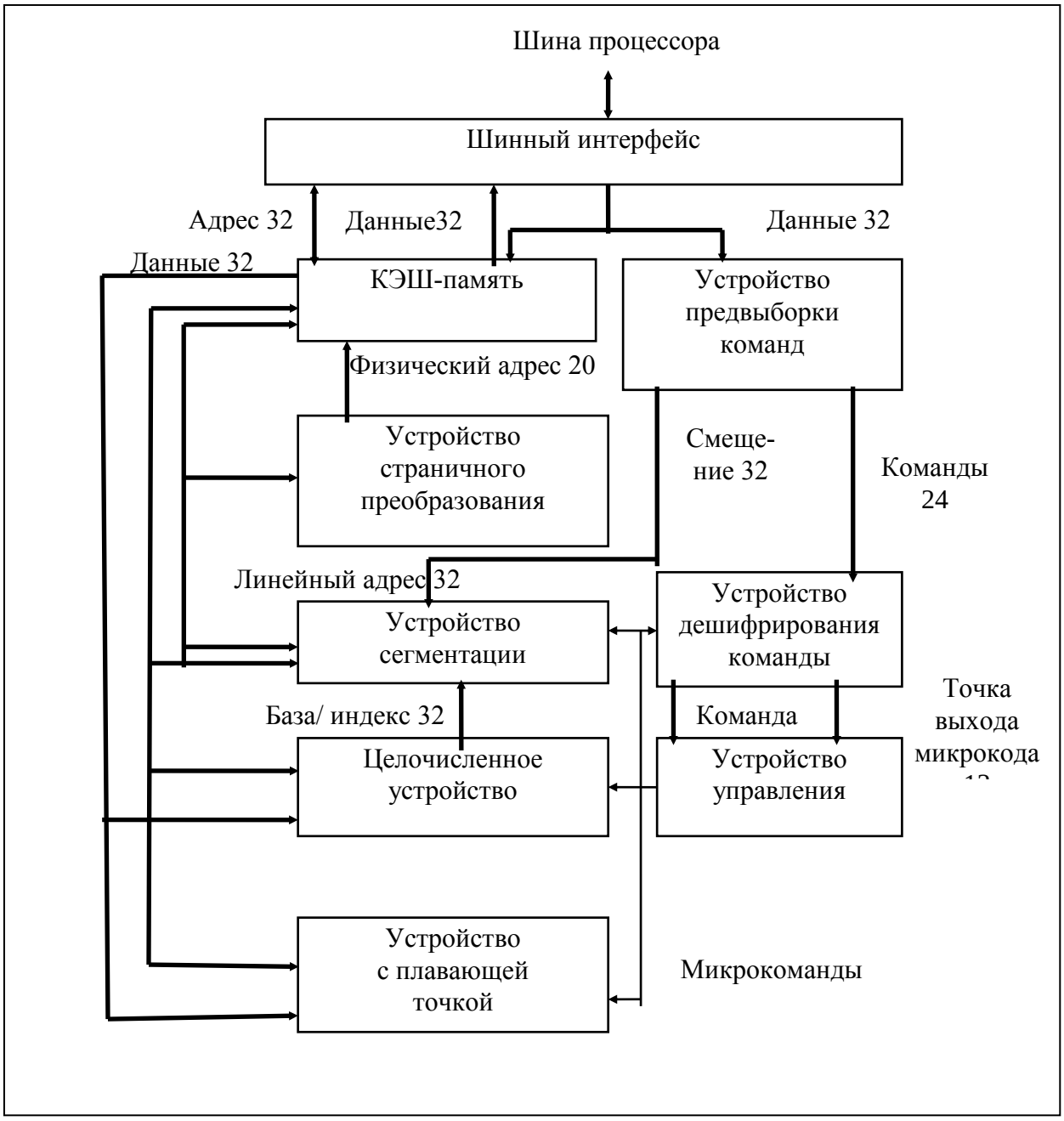


Рисунок 1.2. Упрощенная структурная схема микропроцессора i486.

В состав микропроцессора входят девять внутренних функциональных устройств, которые работают параллельно:

шинный интерфейс;

внутренняя кэш-память;

устройство предвыборки (опережающей выборки) команд;

устройство дешифрирования команд;

устройство управления;

целочисленное устройство;

устройство с плавающей точкой;

устройство сегментации;

устройство страничного преобразования адреса.

Сигналы внешней 32-битной шины процессора подаются во внутренние устройства через шинный интерфейс. Данные передаются из кэш-памяти в шинный интерфейс по 32-битной шине данных. Тесно связанные кэш-память и устройство предвыборки одновременно воспринимают выбранные с опережением команды из шинного интерфейса по 32-битной шине данных, которую кэш-память использует также для получения операндов. Находящиеся в кэш-памяти команды доступны устройству предвыборки, которое имеет 32-байтную очередь команд, ожидающих выполнения.

Когда внутренние запросы данных или команд можно удовлетворить из кэш-памяти, обращение к внешней шине процессора не производится.

Дешифратор команд преобразует команды в управляющие сигналы низкого уровня и точки входа в микрокод.

Устройство управления выполняет микрокод и управляет целочисленным устройством, устройством с плавающей точкой и устройством сегментации. Результаты вычислений помещаются во внутренние регистры целочисленного устройства и устройства с плавающей точкой или кэш-память.

Формирование адреса производят устройства сегментации и страничного преобразования (используется только в защищенном режиме работы процессора). Логические адреса преобразуются устройством сегментации в физические адреса для реального режима работы или в линейные адреса для защищенного режима работы процессора. В свою очередь линейные адреса передаются в устройство страничного преобразования и кэш-память по 32-битной шине адреса. Устройство страничного преобразования превращает линейные адреса в физические, которые направляются в кэш-память по 20-битной шине.

2.АРХИТЕКТУРА РЕАЛЬНОГО РЕЖИМА РАБОТЫ М/П СЕМЕЙСТВА 8086

2.1.Форматы данных микропроцессора

2.1.1.Числа

Микропроцессор работает с двоичными числами со знаком и без знака, длиной 8 бит (1 байт), 16 бит (2 байта) или 32 бита (4 байта), с двоично-десятичными числами длиной 8 бит (BCD - числа) и с десятичными числами длиной 8 бит.

Байт - это число без знака в диапазоне от 0 до 255 или число со знаком в диапазоне от -128 до +127.

Слово - это число без знака в диапазоне от 0 до 65535 или число со знаком, то от -32768 до +32767.

Для 32-разрядных процессоров определены операции над двойными словами. Двойное слово - это число без знака в диапазоне от 0 до 4294967295, или число со знаком в диапазоне от -2147483648 до +2147483647.

Для чисел со знаком старший бит является знаковым: для положительных чисел он равен нулю, а для отрицательных чисел - единице.

Отрицательные числа представлены в дополнительном коде, который получается путем инвертирования всех разрядов положительного числа и суммирования полученного кода с единицей в младшем разряде.

Микропроцессор может выполнять арифметические операции над двоично-десятичными числами, хранящимися в упакованном (2 цифры в байте) формате, и над десятичными числами в неупакованном (1 цифра в байте) формате. Упакованный формат предполагает, что байт содержит две десятичные цифры, занимающие старший и младший полубайты. Диапазон представимых чисел составляет от 0 до 99. В неупакованном формате байт содержит одну десятичную цифру, которая обычно кодируется в символьном коде ASCII. Цифра 0 кодируется как 30H, а цифра 9 - как 39H.

2.1.2.Символы

Символьные данные хранятся в стандартном символьном коде ASCII, каждый символ занимает 1 байт. Микропроцессор ничего не знает об ASCII-коде и рассматривает символьные данные как цепочки произвольных байтов, то есть последовательность кодов символов.

2.1.3.Указатели

Указатели применяются для обращения к некоторым объектам в памяти, например адресам процедур или адресам меток. Близкий (NEAR) внутрисегментный указатель - это 16-

битное или 32-битное смещение (в зависимости от разрядности микропроцессора) от базового адреса того сегмента, в котором находится указатель.

Далекий (FAR) межсегментный указатель применяется в тех случаях, когда программа осуществляет передачу управления в другой сегмент. Такой указатель определяет адрес сегмента и смещение внутри сегмента.

2.1.4.Цепочки

Микропроцессоры семейства 8086 могут оперировать цепочками(строками) байт, слов и двойных слов (32-разрядные процессоры). Под цепочкой понимается последовательность взаимосвязанных элементов, хранящихся по соседним адресам. Длина цепочки не может превышать длину сегмента. Система команд процессора имеет набор цепочечных примитивов, позволяющих эффективно работать с цепочками.

2.2.Адресация памяти

Микропроцессор делит адресное пространство на произвольное количество сегментов, каждый из которых содержит не более 64 Кб. Адрес первого байта сегмента всегда кратен 16 байтам и называется адресом сегмента (параграфом сегмента, номером сегмента). Для обращения к памяти внутри сегмента используется дополнительный адрес, называемый смещением, который указывает расположение байта относительно начала сегмента (относительный адрес или относительное смещение или исполнительный адрес). Адрес сегмента и смещение составляют логический адрес.

Физический адрес образуется с помощью объединения 16 разрядного адреса сегмента и 16 разрядного смещения. Если адрес сегмента сдвинуть влево на 4 бита и дополнить справа 4 нуля, то получится 20-разрядный базовый адрес сегмента. Если сложить базовый адрес сегмента с 16 разрядным смещением, то получится 20 разрядный физический адрес. Следовательно, адресуемая память составляет 2^{20} байт = 1024 К = 1 М. Адрес сегмента - пятизначное шестнадцатеричное число, последняя цифра = нулю, так как адрес получается в результате умножения на 16. Смещение - четырехзначное шестнадцатеричное число. В памяти слово хранится в двух соседних байтах. Младший байт - младший адрес. Старший байт - старший адрес.

2.3.Внутренние регистры процессора

2.3.1.Регистры общего назначения

В микропроцессоре есть четыре 16-битных регистра общего назначения: AX, BX, CX, DX. Младший и старший байты регистров общего назначения доступны для работы и имеют свои уникальные имена: AL и AH, BL и BH, CL и CH, DL и DH соответственно. Регистры

общего назначения в основном используются для временного хранения данных, особенно операндов арифметических операций. Данные регистры взаимозаменяемы, но тем не менее каждый из них имеет специальные функции:

AX - основной регистр, используемый в арифметических операциях над словом, также используется в операциях ввода/вывода и в некоторых операциях со строками.

AL - используется в аналогичных операциях над байтами, при преобразовании 10-х чисел и в операциях над ними.

AH - используется при умножении и делении байтов.

BX - используется при адресации данных.

CX - счетчик числа повторений циклов, номер позиции элемента данных при операциях сдвига и циклического сдвига на несколько битов.

DX - используется как расширение аккумулятора при операциях, дающих 32-разрядный результат и в операциях ввода/вывода.

Начиная с микропроцессора 80386, в дополнение к вышеперечисленным регистрам, существуют еще 32-разрядные регистры общего назначения EAX, EBX, ECX, EDX, младшим словом которых являются регистры AX, BX, CX, DX соответственно.

2.3.2.Сегментные регистры

Все сегментные регистры имеют размер 16 битов (слово).

CS - содержит адрес сегмента кода исполняемой программы.

SS - указывает на сегмент стека - область данных, предназначенную для временного хранения параметров и адресов, используемых программой.

DS - указывает на сегмент данных.

ES - указывает на дополнительный сегмент данных, используется при операциях с цепочками (если данные программы превышают 64 Кб).

Если программа не изменяет значения регистра CS, то размер ее кода не может превышать 64 Кб. Переключение адреса сегмента кода программы выполняется с помощью специальных команд JMP и CALL, когда управление передается на дальнюю метку или в дальнюю процедуру. Эти формы передачи управления неявно изменяют значение регистра CS и позволяют создавать программы любого размера.

Начиная с микропроцессора 80386, в дополнение к вышеперечисленным регистрам, существуют еще два регистра для адресации дополнительных сегментов данных FS и GS.

2.3.3.Регистры смещения

Микропроцессор содержит пять основных 16-битных регистров смещения:

SP и BP - содержат смещение в сегменте стека. SP-указывает на вершину стека, BP используется как базовый регистр для того, чтобы зафиксировать положение стека в какой-то момент времени и в дальнейшем адресоваться к данным, расположенным в стеке;

SI и DI в основном используются для формирования сложных адресов, состоящих из смещения начала блока данных в сегменте, и относительного смещения элемента данных внутри блока. При этом смещение начала блока обычно хранится в регистре BX или непосредственно в команде, а SI или DI задают смещение внутри блока. Кроме того, они участвуют в выполнении команд работы с цепочками;

IP - указывает на следующую выполняемую команду в сегменте кода, адрес ее хранится в CS. Программа не может явно получить или изменить значение этого регистра. Однако команды JMP и CALL неявно изменяют его значение, а также сохраняют его в стеке и восстанавливают из стека.

Начиная с микропроцессора 80386, в дополнение к перечисленным регистрам, существуют еще 32-разрядные регистры смещения ESP, EBP, ESI, EDI, EIP, младшее слово которых составляют регистры SP, BP, SI, DI и IP соответственно.

2.3.4.Регистр флагов

Для 16-разрядных микропроцессоров регистр флагов имеет размер 16 битов и использует в реальном режиме работы 9 флагов: 6-статусных флагов (отражают результаты арифметических и логических операций) и 3-управляющих флага (изменяют режим работы процессора).

Начиная с микропроцессора 80386 регистр флагов имеет размер 32-бита, но в реальном режиме используются тоже только 9 флагов из младшего слова:

CF - флаг переноса, указывает на арифметический перенос разряда. CF также может содержать значение бита, который при сдвиге или циклическом сдвиге вышел за границы регистра. CF=1, если перенос был (0 разряд).

PF - флаг четности, указывает на четность единичных разрядов в результате операции. Используется в основном в операциях обмена данными (2 разряд).

AF - вспомогательный флаг переноса, указывает на корректировку, необходимую при двоично-десятичных арифметических операциях (фиксирует факт переноса из младшего полубайта в старший полубайт). Упакованное число - 2 цифры в байте. В микропроцессоре нет десятичного сумматора. Сложение и вычитание десятичных чисел производится как сложение и вычитание двоичных чисел. Результат может быть неверным десятеричным числом, поэтому существуют команды преобразования результата в правильное десятеричное число. Чтобы это сделать, надо знать, был ли перенос (4 разряд).

ZF - флаг нуля. Указывает на нулевой результат или равенство при сравнении (6 разряд).

SF - флаг знака. Указывает на отрицательный результат в арифметической операции, логической, операциях сдвига или циклического сдвига (7 разряд).

OF - флаг переполнения. Указывает на арифметическое переполнение при операции (выход за пределы допустимых значений) (11 разряд).

TF - флаг трассировки (ловушки). Управляет одношаговыми операциями (при использовании отладчика), генерируя программные прерывания в конце каждой команды (8 разряд).

IF - флаг прерываний. Управляет разрешением и запретом прерываний от внешних устройств. Блокирует все прерывания, за исключением немаскируемых прерываний (NMI) (9 разряд).

DF - флаг направления. Управляет направлением влево и вправо в операциях со строками. При DF=1 микропроцессор уменьшает на 1 содержимое регистров индекса SI и DI после выполнения команды, а при DF=0 увеличивает на 1 содержимое регистра индексов (10 разряд).

Регистр флагов не имеет имени, однако, часто 16-разрядный регистр флагов называют FLAGS, а 32-разрядный регистр флагов - EFLAGS.

Формат 16-разрядного регистра флагов

6				1	0										
				F	F	F	F	F	F		F		F		F

2.4.Режимы адресации

Обычно выделяют семь режимов адресации, которые определяют способ вычисления смещения адреса операнда :

Регистровая адресация.

Непосредственная адресация.

Прямая адресация.

Косвенная регистровая адресация.

Базовая адресация.

Прямая адресация с индексированием.

Базовая адресация с индексированием.

2.4.1.Регистровая адресация.

При таком способе адресации значения операндов содержатся в регистрах. Допускаются операции с любыми регистрами, кроме регистра IP. Регистр CS не может являться операндом - приемником.

Примеры использования регистровой адресации:

MOV AX, CX

ADD AX, BX

2.4.2.Непосредственная адресация

При таком способе адресации операндом - источником является непосредственное значение (константа).

Пример использования непосредственной адресации:

MOV CX, 5 или K EQU -8

MOV CX, K

2.4.3.Прямая адресация

При прямой адресации смещение адреса операнда (исполнительный адрес) содержится в самой команде, например:

.data

T DB 3

.code

MOV AL, T

T - имя ячейки памяти , значение которой загружается в регистр AX.

2.4.4.Косвенная регистровая адресация

В данном случае регистр содержит не значение операнд, а его смещение в памяти. При этом для адресации операнда могут использоваться регистры BX, DI и SI, если операнд берется из сегмента данных (DS). Регистр BP используется для косвенной адресации, если операнд находится в сегменте стека (SS). При использовании команд работы с цепочками регистр DI всегда адресует операнд в дополнительном сегменте памяти (ES).

Начиная с микропроцессора 80386 для косвенной регистровой адресации могут быть использованы все 32-разрядные регистры общего назначения: EAX, EBX, ECX, EDX.

Примеры использования косвенной регистровой адресации:

MOV AX,[BX].

Смещение адреса может быть загружено в регистр при помощи операции OFFSET, а адрес сегмента - операцией SEG.

MOV BX, OFFSET T

MOV DS, SEG T

Если требуется, можно изменить адрес сегмента ячейки памяти, используя префикс сегмента, например:

MOV AX, ES:[BX] - пересылает в AX операнд, адрес которого складывается из содержимого регистров ES и BX.

2.4.5.Базовая адресация

При этом способе адресации смещение адреса операнда образуется как сумма:

[BX] + сдвиг, если операнд в сегменте данных(DS);

[BP] + сдвиг, если операнд в стеке (SS).

Базовый адрес массива (записи) помещается в регистр BX или BP, а сдвиг указывает на элемент относительно базы. Для доступа к разным записям или к элементам разных массивов внутри массива достаточно только перегрузить регистр BX (BP).

Примеры базовой адресации:

MOV AX, [BX] + 4

MOV AX, [BX + 4].

2.4.6.Прямая адресация с индексированием

При такой адресации смещение адреса операнда образуется как сумма значений смещения ячейки памяти и смещения в регистре SI или DI. Данный способ используется для доступа к массивам данных. Например:

.data

T DB 6, 5, 4, 3

MOV DI, 3

MOV AL, T[DI] - Загружает в AL 3-й элемент массива T, т.е. 4.

При операциях со строками регистр DI по умолчанию указывает на дополнительный сегмент данных (CS).

2.4.7.Базовая адресация с индексированием

Смещение адреса операнда образуется как сумма трех компонент:

BX + SI + сдвиг или BX + DI + сдвиг, если операнд находится в основном сегменте данных (DS),

BP + SI + сдвиг или BP + DI + сдвиг, если операнд находится в сегменте стека (SS).

Удобен для адресации двумерных массивов, когда BX или BP содержат смещение адреса начала массива, а в индексных регистрах и сдвиге - индексы элементов по строке и столбцу, например:

MOV AX, [BX + DI + 2]

MOV AX, [DI + BX + 2]

MOV AX, T[DI][SI]

2.5.Система команд микропроцессора

Система команд микропроцессора обычно делится на семь групп команд:

Команды пересылки данных.

Арифметические команды.

Логические команды или команды манипулирования битами.

Команды передачи управления.

Команды обработки строк (цепочные команды).

Команды управления процессором.

Команды прерывания.

2.5.1.Команды пересылки данных.

Перечень команд пересылки данных приведен в таблице 2.1.

Таблица 2.1. Перечень команд пересылки данных.

Мнемоника команды	Описание команды
Общие	
MOV (переслать)	источник - приемник
MOVSX (с м/п 80386)	источник - приемник
MOVZX (с м/п 80386)	источник - приемник
XCHG (обменять)	источник - приемник
PUSH (включить в стек)	источник - стек
POP (извлечь из стека)	Стек - приемник
PUSHA (включить в стек все)	регистры - стек
POPA (извлечь из стека все)	Стек - регистры
XLAT (преобразовать)	M[AL] > AL
Аккумуляторные (ввода-вывода)	
IN (ввести)	Порт → AL или AX
OUT (вывести)	AL или AX → порт

XLAT (преобразовать)	f(AL) → AL
Адресные	
LEA (смещение)	Смещение источника → регистр
LDS (загрузить полный адрес в регистр DS)	Источник, источник + 1 → регистр Источник + 2, источник + 3 → DS
LES (загрузить полный адрес в регистр ES)	Источник, источник + 1 → регистр Источник + 2, источник + 3 → ES
LSS (с м/п 80386)	
LFS (с м/п 80386)	
LGS (с м/п 80386)	
Флажковые	
LAHF (загрузить флажки в AH)	SF, ZF, AF, PF, CF → AH
SAHF (запомнить AH во флажках)	AH → SF, ZF, AF, PF, CF
PUSHF (включить в стек флажки)	Флаги → стек
POPF (извлечь из стека флаж-ки)	Стек → флаги

2.5.1.1. Общие команды.

Команда MOV - переслать данные, основная команда группы.

Команда MOV осуществляет пересылку байт, слов или двойных слов.

Форматы команды MOV приведены в таблице 2.2.

Таблица 2.2. Форматы команды MOV.

Передача данных	Слово	Байт
Регистр в регистр	MOV AX, BX	MOV AH, BH

Операнд в регистр или память	MOV CX, 850
	
	MOV PW, 850
		MOV BL, 35
		
		MOV PB, 35

Память в регистр	MOV DX, PW	MOV CL, PB
Регистр в память	MOV PW, DX	MOV PB, CL

Регистр в сегментный регистр	MOV ES, BX	
Сегментный регистр в регистр	MOV AX, DS	
Сегментный регистр в память	MOV PW, CS	

Исключения:

Нельзя пересылать данные из одной ячейки памяти в другую.

Нельзя загрузить в регистр сегмента операнд с непосредственной адресацией.

Нельзя переслать значение одного регистра сегмента в другой.

Нельзя использовать регистры CS и IP в качестве приемника в команде MOV.

Команды MOVSX и MOVZX - новые команды пересылки данных, введенные в систему команд микропроцессора 80386. Форматы команд:

$$\left.\begin{matrix}\text{MOVSX}\\ \\ \text{MOVZX}\end{matrix}\right\} \begin{cases}\text{reg16, reg/mem8; расширение байта в слово}\\ \text{reg32, reg/mem16; расширение слова в двойное слово}\\ \text{reg32, reg/mem8; расширение байта в двойное слово}\end{cases}, \text{где}$$

reg- только регистры общего назначения, mem - ячейка памяти.

MOVSX - пересылает данные и расширяет регистр знаковым разрядом операнда - источника,

MOVZX - пересылает данные и расширяет регистр нулями.

Команда XCHG - осуществляет обмен байт или слов. Один из ее операндов может быть в регистре или памяти, другой - в регистре. Различий между приемником и источником нет. Примеры использования команды прведены в таблице 2.3.

Таблица 2.3. Примеры использования команды XCHG.

Обмен данными	Слово	Байт
Регистр с регистром	XCHG CX, DX	XCHG AL, AH
Регистр с памятью	XCHG BX, PW	XCHG BL, PB

Исключение:

Нельзя выполнить обмен значений регистров сегментов.

2.5.1.2. Команды PUSH и POP .

PUSH - передает слово из источника в стек, а команда POP осуществляет противоположное действие: передает слово из стека в приемник. Регистр SP содержит смещение последнего включенного в стек слова (вершину стека).

! Стек растет по направлению уменьшения адресов !

PUSH начинается с уменьшения содержимого SP на 2, а команда POP завершается увеличением содержимого SP на 2.

Операндами команд PUSH и POP могут быть сегментный регистр, несегментный (шестнадцатиразрядный) регистр или слово в памяти. Кроме того, в команде PUSH можно указывать непосредственный операнд, что не допускается в команде PUSH для процессора 8086. POP- не может содержать непосредственный операнд. Форматы команд приведены в таблице 2.4.

Таблица 2.4. Форматы команд PUSH и POP.

Операнд	Включение	Извлечение
Регистр	PUSH AX	POP BX
Память	PUSH PW	POP PW
Сегментный регистр	PUSH DS	POP ES
Непосредственный операнд	PUSH 856	
Все 16-битные регистры	PUSHA	POPA
Все 32-битные регистры	PUSHAD	POPAD

Исключение:

POP CS и POP IP - недействительные операции.

PUSHA и POPA являются эффективным средством для сохранения содержимого всех регистров (кроме сегментных и IP) в начале выполнения процедуры и восстановления их в конце работы. PUSHA включает в стек регистры в следующем порядке: AX, CX, DX, BX, SP, BP, SI, DI. Значение SP то, что было в нем до выполнения команды PUSHA. PUSHA уменьшает содержимое SP на 2 при включении в стек содержимого каждого регистра. POPA вызывает увеличение содержимого SP на ту же величину, что и PUSHA, ей не требуется запомненное в стеке содержимое регистра SP и она его просто уничтожает. Команды PUSHAD, POPAD работают аналогично с 32-битными регистрами процессора.

Команда XLAT - преобразует значение в регистре AL: она его заменяет на байт из таблицы, адресуемой регистром BX, причем индексом таблицы служит исходное содержимое регистра AL. Удобна для преобразования из одного кода в другой.

Пример использования команды для перевода цифр из 10 системы счисления в код “2 из 5” - любой код, содержащий 2 единичных бита.

Код “2 из 5” приведен в таблице.

Цифра	Код

0	11000
1	00011
2	00101
3	00110
~	~
9	10100

TABLE DW 11000B, 00011B, 00101B, 00110B, 01001B, 01010B, 01100B, 10001B, 10010B, 10100B, 11000B

MOV BX, OFFSET TABLE

MOV AL, 7

XLAT TABLE

2.5.1.3. Команды ввода-вывода.

Для связи с разными частями ЭВМ и управления ими микропроцессор использует порты ввода-вывода. Любой порт идентифицируется шестнадцатиразрядным номером порта в диапазоне от 0 до 65535. Как и при доступе к памяти, процессор для связи использует шины данных и адреса. При доступе к порту он посылает сначала по управляющей шине сигнал, который оповещает все устройства ввода-вывода, что адрес на шине является адресом определенного порта, а затем посылает сам адрес. То устройство, адрес порта которого совпадает, дает ответ. Номер порта - это адрес ячейки памяти, являющейся частью устройства ввода-вывода, а не частью основной памяти. Для указания на доступ к порту и пересылки информации к устройствам ввода-вывода и обратно, используются специальные команды ввода-вывода.

IN - передает данные (байт, слово или двойное слово) из исходного порта в аккумулятор (AL, AX или EAX).

OUT - передает данные из аккумулятора в исходный порт.

Номер порта можно указывать либо в самой команде, либо в регистре DX(0-65535).

Примеры использования команд ввода-вывода приведены в таблице 2.5.

Таблица 2.5. Примеры использования команд ввода-вывода.

Команда	Байт	Слово	Дв. Слово
IN (непосредственная, операнд)	IN AL, 20h	IN AX, 20h	IN EAX, 20h
OUT(непосредс	OUT 20h,	OUT 20h,	OUT 20h,

твенная, операнд)	AL	AX	EAX
IN (регистр)	IN AL, DX	IN AX, DX	IN EAX, EDX
OUT(регистр)	OUT DX, AL	OUT DX, AX	OUT EDX, EAX

2.5.1.4. Адресные команды (пересылки адреса)

В микропроцессоре существуют три основные команды пересылки адреса: LEA, LDS, LES.

LEA - загрузить смещение в регистр. Форматы команды:

LEA { reg16, mem16
reg32, mem16 - расширение нулем
reg16, mem32 - младшее слово
reg32, mem32

LDS -загрузить полный адрес в DS и регистр;

LES - загрузить полный адрес (указатель) в ES и регистр.

С микропроцессора 80386 в систему команд добавляются команды LSS, LFS, LGS, которые используются для загрузки полного адреса в регистр-приемник и регистры SS, FS, GS соответственно. Форматы команд LDS, LES, LSS, LFS, LGS:

LSS
LFS
LGS
LDS
LES
} { reg16, mem32
reg32, mem48

Каждая из команд имеет два операнда - один в памяти, другой в 16-битном регистре или 32-битном регистре. Примеры использования команд:

LEA BX, MEMLOC=MOV BX, OFFSET MEMLOC

LDS CX, MEMLOC

LES DX, MEMLOC

LEA вычисляет 16-битное или 32-битное смещение операнда-источника и загружает его в любой 16-битный или 32-битный регистр общего назначения или в регистр смещения (кроме IP). Она удобна для передачи смещения переменной в качестве параметра из одной процедуры в другую. В качестве операнда-источника в команде LEA может быть элемент массива, например: LEA BX, TABLE[DI].

LDS передает полный адрес операнда-источника в 16-битный или 32-битный регистр (любой регистр общего назначения) и в регистр DS.

```
HERE DB 100                    =    MOV BX, OFFSET HERE

LDS BX, HERE                        MOV AX, SEG HERE
                                    MOV DS, AX
```

LES, LSS, LFS, LGS аналогичны LDS, но загружают адрес сегмента в регистры ES, SS, FS, GS соответственно.

Замечание: 32-разрядная адресация используется только в защищенном режиме работы микропроцессора. В реальном режиме работы команды используют только 16-битные регистры.

2.5.1.5. Флажковые команды (команды пересылки флагов).

Команды этого класса обеспечивают доступ к флагам процессора.

Команды LAHF (загрузить флаги в AH) и SAHF (запомнить AH во флагах). LAHF передает 5 флагов SF, ZF, AF, PF и CF в определенные биты регистра AH, а SAHF - реализует противоположную передачу. На рисунке 2.1 показано соответствие флагов разрядам регистра AH.

SF	ZF		AF		PF		CF	флаги
7	6	5	4	3	2	1	0	AH

Рисунок 2.1. Соответствие флагов разрядам регистра AH.

Эти флаги выделены потому, что они были в микропроцессоре 8080. LAHF и SAHF предусмотрены в основном, для преобразования программ микропроцессора 8080 в программы микропроцессора 8086. Команда SAHF оказалась полезной при использовании арифметического сопроцессора для использования результатов расчета в команде условного перехода.

Существуют команды для пересылки содержимого регистра флагов в стек и обратно. Это команды PUSHF (включить в стек флаги) и POPF (извлечь из стека флаги). Рисунок 2.2 иллюстрирует работу этих команд.

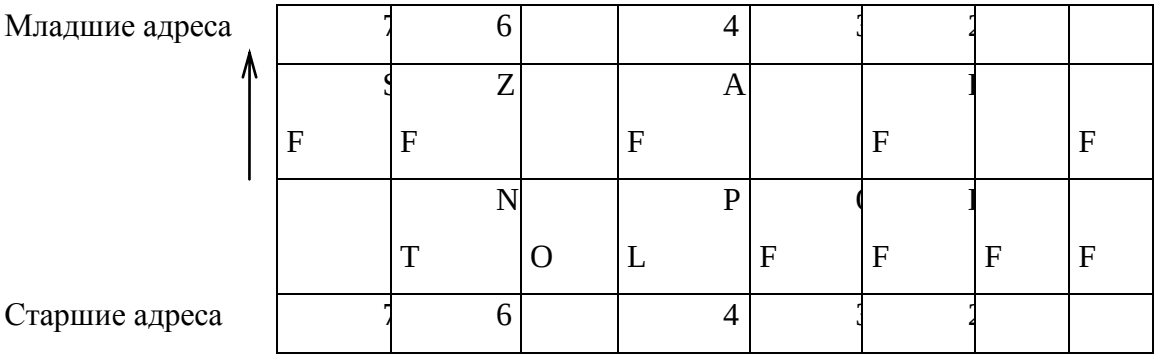


Рисунок 2.2. Размещение в стеке флагов после команды PUHF.

Начиная с микропроцессора 80386 добавляются команды PUSHFD и POPFD для 32-битного регистра EFLAGS.

PUSHFD- включает в стек 32-разрядный регистр флагов.

POPFD - извлекает из стека 32-разрядный регистр флагов.

2.5.2.Арифметические команды.

Перечень арифметических команд приведен в таблице 2.6.

Таблица 2.6. Перечень арифметических команд.

Мнемоника команд	Описание команды
Сложение	
ADD - сложение	приемник + источник → приемник
ADC - сложить с переносом	приемник + источник + CF → приемник
INC - увеличить на 1	приемник + 1 → приемник
. Вычитание	
SUB - вычесть	приемник - источник → приемник
SBB - вычесть с займом	приемник - источник - CF → приемник
DEC - уменьшить на 1	приемник - 1 → приемник
NEG - изменить знак	0 - приемник → приемник
CMP - сравнить	приемник - источник →
Умножение	
MUL - умножить	AL * источник (8) → AX AX * источник (16) → DX, AX
IMUL - умножить со знаком	так же, как и в команде MUL, но операнды со знаком
Деление.	
DIV - разделить	AX/источник (8): целая часть → AL, остаток → AH
	[DX, AX]/источник (16): целая часть → AX, остаток → DX
IDIV - разделить со знаком	так же, как и в команде DIV, но операнды со знаком
Команды расширения	
CBW - преобразовать байт в слово	знаковый бит AL → в AH
CWD - преобразовать слово в	знаковый бит AX → в DX

двойное слово	
CWDE - преобразовать слово в расширенное двойное слово	знаковый бит AX → в EAX
CDQ - преобразовать двойное слово в учетверенное двойное слово	знаковый бит EAX → в EDX

В арифметических командах устанавливаются или сбрасываются 6 флажков состояния:

CF - устанавливается, если операция дала беззнаковый результат вне диапазона (т.е. есть перенос в знаковый разряд). Заём (7,15,31) вызывает выход из разрядной сетки.

OF - устанавливается, если в операции получился знаковый результат, находящийся вне диапазона (т.е. перенос в знаковый разряд) не создаёт переноса из разрядной сетки или перенос из разрядной сетки происходит без переноса в знаковый разряд.

ZF - устанавливается, если результат операции (знаковый или беззнаковый) равен нулю.

SF - устанавливается, если старший бит результата операции содержит 1, показывая отрицательное число.

PF - устанавливается, если результат операции содержит четное число единичных битов.

AF - устанавливается, если в десятичных операциях требуется коррекция.

2.5.2.1. Команды сложения.

В системе команд существует три команды сложения: ADD, ADC, INC.

ADD - команда сложения, один из операндов которой может находиться в регистре или в самой команде (непосредственный операнд)

ADC - команда сложения с переносом, аналогична ADD, но использует в качестве третьего слагаемого начальное значение CF.

INC - команда увеличения на единицу, имеет один операнд. Прибавляет 1 к содержимому операнда и помещает результат в этот же операнд (не устанавливает флаг CF). INC идентична ADD с непосредственным операндом 1, но требует меньше байт в памяти.

Форматы команды ADD:

ADD reg/ mem, imn

ADD reg, reg/ mem

ADD mem/ reg, reg,

где reg - имя регистра, mem - имя ячейки памяти, imn - непосредственное значение размером 8, 16 или 32- разряда:

Начиная с микропроцессора 80386 введены следующие форматы:

ADD reg/ mem16, imn 8

ADD reg/ mem 32, imn 8 (расширение недостающих разрядов знаком).

2.5.2.2.Команды вычитания.

Команды вычитания SUB, SBB, DEC аналогичны командам ADD, ADC, INC, только производят операцию вычитания, а не сложения.

Команда CMP аналогична команде SUB, но результат не запоминается в приемнике, а устанавливаются только флаги в соответствии с результатом. CMP - это команда сравнения, после которой обычно следует команда условного перехода. Состояние флагов после выполнения команды CMP приведено в таблице 2.7.

Таблица 2.7. Состояние регистра флагов после выполнения команды CMP

	Знаковые	Беззнаковые
приемник>источника	ZF=0&SF=OF	CF=0&ZF=0
приемник≥источника	SF=OF	CF=0
приемник=источника	ZF=1	ZF=1
приемник≤источника	ZF=1&SF≠OF	CF=1&ZF=1
приемник<источника	SF≠OF	CF=1

Команда NEG изменяет знак своего операнда, то есть вычитает значение операнда-приёмника из 0 и тем самым формирует его дополнение до двух. Полезна для вычитания значения регистра или ячейки памяти из непосредственного значения.

Пример:

SUB 100, AL - запрещена

Можно сделать следующее:

NEG AL

ADD AL, 100

NEG - даёт дополнительный код операнда

2.5.2.3.Команды умножения и деления.

Умножение двух однобайтовых чисел может дать произведение длиной в слово. Аналогично умножение двух слов может дать результат длиной в двойное слово, а умножение двух двойных слов может дать результат длиной в учетверенное слово. Приведенный пример иллюстрирует данное утверждение.

8 бит

8 бит

16 бит

При умножении двух байтов командой MUL произведение будет находиться в регистрах AH (старший байт) и AL (младший байт), при умножении двух слов произведение будет находиться в регистрах DX (старшее слово) и AX (младшее слово), , а при умножении двух двойных слов произведение будет находиться в регистрах EDX и EAX.

Команда деления DIV делит 16-битовое число из регистра AX (или 32-битное число из регистров DX и AX, или 64-битное число из регистров EDX, EAX) на операнд половинного размера, определяемый в команде. Частное помещается в регистр AL (AX, EAX), а остаток в AH (DX, EDX). Для знаковых чисел требуются специальные команды умножения и деления (IMUL и IDIV).

Форматы команд умножения и деления приведены в таблице 2.8:

Таблица 2.8. Общие форматы команд умножения и деления.

Операнд	Слово	Байт
AX (AL) с регистром	КОП BX	КОП CL
AX (AL) с памятью	КОП MEMW	КОП MEMB

КОП: MUL, IMUL, DIV, IDIV.

В микропроцессоре 80286 команда IMUL имеет ещё один дополнительный формат: разрешается определить множитель как непосредственный операнд. При этом множимое не обязательно должно быть в регистре AX (AL), а может находиться в любом шестнадцатиразрядном регистре или 16-битной ячейке памяти. Результат длиной только 16 бит допускается разместить в любом шестнадцатиразрядном регистре.

Примеры:

Слово в регистре IMUL DX, BX, 115; BX * 115 DX

Слово в памяти IMUL CX, MEMW, 632; MEMW * 632 CX

Начиная с микропроцессора 80386 добавляются следующие форматы команды IMUL:

Двухоперандные:

IMUL reg16, imn 8;

IMUL reg16, imn 16;

IMUL reg 32, imn 8;

IMUL reg 32, imn 32;

IMUL reg16, reg/ mem 16.

Старшая часть произведения теряется.

Трехоперандные:

IMUL reg 16, reg/ mem 16, imn 8/ 16;

IMUL reg 32, reg/ mem 32, imn 8/ 32.

Старшая часть произведения теряется.

Однооперандные:

Второй операнд и приемник в регистрах AL, AX, EAX.

IMUL reg/ mem (8/ 16/ 32-бита)

Команды умножения и деления действуют так, чтобы результат двойной длины при умножении можно было использовать в последующем делении. Если требуется поделить два числа одинакового размера, то в этом случае необходимо искусственно увеличить размер делимого. Если число без знака, то можно просто обнулить содержимое регистра AH, а если число со знаком, то 8-битное число должно быть преобразовано в 16-битное, 16-битное - в 32-битное и 32-битное в 64-битное. Для проведения операций расширения знака числа в микропроцессоре существуют специальные команды: CBW , CWD, CWDE, CDQ.

2.5.2.4.Команды расширения знака.

CBW преобразует байт в регистре AL в слово в регистре AX путем расширения знакового бита AL во все биты регистра AH. Команда CWD преобразует слово в регистре AX в двойное слово , расположенное в паре регистров DX, AX путем расширения знакового бита регистра AX во все биты регистра DX.

Новые команды расширения знака

Начиная с микропроцессора 80386, существуют еще две команды расширения знака CWDE и CDQ. CWDE преобразует слово в расширенное двойное слово путем расширения знакового разряда AX во все старшие разряды регистра EAX. CDQ преобразует двойное слово в регистре EAX в учетверенное слово в паре регистров EDX, EAX путем расширения знакового разряда EAX во все разряды EDX.

2.5.2.5.Десятичная арифметика.

До сих пор мы рассматривали арифметические операции под двоичными числами, так как компьютеры работают только с двоичными числами, но для людей более привычны десятичные числа. Поэтому возникает проблема преобразования десятичных чисел в двоичные. Можно десятичное число представить в двоичной системе полностью, кодом, например 37 - 00100101, а можно закодировать отдельно каждую цифру 3 и 7 и получить код 0011 0111. Такое двоичное изображение десятичных чисел называется двоично-десятичным кодированием (BCD - кодом). Для выполнения арифметических операций над числами в данном формате потребовалось бы ввести соответствующие команды сложения, вычитания, умножения и деления. Возможен и второй вариант: применить к таким числам команды двоичной арифметики, заранее зная о неправильном результате, а затем выполнить команду коррек-

ции, которая сформирует правильный результат в BCD формате. Именно такой вариант был выбран в процессорах семейства 8086.

Рассмотрим сложение чисел 23 и 14 в BCD формате с помощью двоичного сложения:

5	
4	
9	Результат правильный, коррекция не нужна

Сложим 29 и 14:

9	
4	
?	Ответ неверен, так как код 1101 не соответствует десятичной цифре, требуется коррекция.

Коррекция заключается в том, чтобы добавить 6 к сумме в тех разрядах, где получена запрещённая комбинация, компенсируя этим, 6 запрещённых комбинаций для десятичных чисел (4 разряда - 16 комбинаций, 10 цифр правильных, 6 - лишних).

?	
6	
3	Результат правильный

Более сложная ситуация возникает, когда сумма “проскакивает” запрещённый диапазон и становится допустимой цифрой.

Сложим 29 и 18:

9	
8	

1	Результат неверный, так как младшая цифра проскочила запрещенный диапазон.

При коррекции требуется добавить 6 и получить правильный результат 47. Однако необходимость такой коррекции невозможно определить по самому результату. Признаком “проскока” цифрой запрещённого диапазона служит перенос из соответствующего бита (разряда). В приведённом примере им будет перенос из младшего (десятичного) разряда в старший. Флаг CF показывает, что при сложении возник перенос из старшего бита (разряда), флаг вспомогательного переноса AF предназначен только для регистрации переноса из младшего 10-го разряда, зная который можно осуществить коррекцию. После сложения в нашем примере CF = 0 и AF = 1 (если CF = 1, то при следующем сложении надо учитывать его и сумму).

Десятичную коррекцию сложения осуществляет команда DAA, в которой предполагается, что сумма находится в регистре AL. С учётом содержимого AL и состояний флагов AF и CF команда DAA определяет необходимость коррекции и реализует её для AL.

Аналогично команда DAS корректирует результат после операции вычитания.

Для умножения чисел в формате BCD произвести коррекцию невозможно, так как в результате “замешаны” перекрёстные члены произведения. Аналогично и для команды деления. Следовательно, для умножения и деления необходимо перейти к другому представлению десятичных чисел. BCD формат называется упакованным, а в неупакованном формате байт содержит всего одну десятичную цифру. Она находится в 4-х младших битах, а старшие биты не влияют на значение цифры. Примером такого формата служит код ASCII, в котором символы представлены 8 битами. ASCII-коды десятичных цифр представлены в таблице 2.9.

Таблица 2.9. ASCII-коды десятичных цифр.

Цифра	Код
0	00110000
1	00110001
2	00110010
3	00110011
4	00110100
5	00110101

6	00110110
7	00110111
8	00111000
9	00111001

Четыре бита 0011 не влияют на значение цифры, однако, должны быть обнулены до выполнения арифметических операций.

Результаты двоичного сложения и вычитания ASCII-чисел можно скорректировать аналогично коррекции в BCD формате, причём корректируется только младшая цифра. В системе команд микропроцессора существуют специальные команды коррекции:

AAA - ASCII коррекция сложения.

AAS - ASCII коррекция вычитания.

AAM - ASCII коррекция умножения.

AAD - ASCII коррекция деления.

Пример: умножим 9 * 4, 9 - находится в регистре BL, а 4 - в регистре AL.

BL: 00001001 = 9 | MUL BL - даёт в AX 16-битный результат, равный 36

AL: 00000100 = 4 | 36 = 0000 0000 0010 0100

Команда коррекции AAM должна “разложить” результат на 3 (00000011) в регистре AH и 6 в регистре AL. Для этого нужно просто раз делить содержимое AL на 10 и поместить частное в AH, а остаток в AL.

Поэтому команда AAM имеет длину 2 байта, так как второй байт - это представление 10. В рассмотренном примере старшие биты были нулевыми, иначе результат нельзя скорректировать. Поэтому перед умножением неупакованных десятичных чисел следует сбросить четыре старших бита в 0.

Рассмотрим деление неупакованных десятичных чисел, например 42/6. 42 находится в AL (0000 0100 в AH и 0000010 в AL), а 6 (00000110) в BL. Неупакованное представление одноразрядного числа 6 является его двоичным представлением, следовательно, нужно преобразовать 42 в двоичное число. Для этого AH следует умножить на 10 и сложить с содержимым регистра AL. Тогда при делении в AL получится число 7, двоичное представление которого совпадает с неупакованным представлением. Команда коррекции деления имеет свои особенности:

AAD - двухбайтовая команда (второй байт - 10).

Коррекция AAD предшествует делению, а в сложении, вычитании и умножении производится после операции.

Делимое и делитель (множимое и множитель) должны иметь 0 в старших 4-х битах.

2.5.3.Логические команды.

Перечень логических команд и их описание приведено в таблице 2.10.

Таблица 2.10. Перечень логических команд.

	Мнемоника	Описание
Булевы команды	AND (и)	приемник и источник → приемник
	TEST	приемник и источник →
	OR (или)	приемник или источник → приемник
	XOR (исключающее или)	приемник ⊕ источник → приемник
	NOT (инверсия)	не приемник → приемник
Команды	SHL (логический сдвиг влево)	CF ← приемник ← 0
сдвигов	SHR (логический сдвиг вправо)	0 → приемник → CF
	SAL (арифметический сдвиг влево)	CF ← приемник ← 0
	SAR (арифметический сдвиг вправо)	знак → приемник → CF
Команды циклических	ROL (циклический сдвиг влево)	
сдвигов	ROR (циклический сдвиг вправо)	
	RCL (циклический сдвиг влево через перенос)	
	RCR (циклический сдвиг вправо через перенос)	
Начиная с м/п 80386 дополнительные команды:		
Команды двойного сдвига	SHLD (сдвиг с двойной точностью влево)	
	SHRD (сдвиг с двойной точностью вправо)	
Команды работы с	BT (поиск бита и запись	

двоичными цепочками	его в CF)	
	BTC (поиск и инвертирование бита в цепочке)	
	BTS (поиск и установка бита в)	
	BTR (поиск и сброс бита)	
	BSF (сканирование цепочки битов вперед)	
	BSR (сканирование цепочки битов назад)	

2.5.3.1.Булевы команды.

К булевым командам относятся команды AND, OR, XOR, NOT, TEST.

Команды AND, OR, XOR выполняют логическую функцию над соответствующими битами источника и приёмника, помещая результат в приёмник. Все команды двухоперандные , разрешённые форматы операндов такие же, как у команд ADD, ADC, SUB, SBB.

Команда NOT - одноопернадная, она выполняет инверсию каждого бита операнда и помещает результат в то же место. Формат команды такой же, как у команд DEC, NEG, INC.

Команда AND удобна для обнуления указанных разрядов числа: один операнд определяет разряды, а второй число. Например: обнуление старших 4-х битов в байте с именем MEMB (AND MEMB, 00001111B).

Команда OR используется для установки указанных битов, а команда XOR используется для инвертирования указанных разрядов в числе. XOR позволяет также сбросить содержимое регистра в нуль (регистр должен быть и источником и приёмником).

Команда TEST - двухоперандная, формат совпадает с командами ADD, ADC, SUB, SBB. Объединяет возможности команд AND и CMP, Как AND она выполняет объединение по “и” соответствующих бит операндов, как CMP она сохраняет только состояния флагов, а не результат. Удобна для проверки того, есть ли в указанных разрядах числа хотя бы одна 1. Один операнд определяет разряды, второй - число. Если результат неравен 0, то, по крайней мере, один разряд равен 1. Все логические команды кроме NOT изменяют флаги SF, ZF, PF. Флаг AF - не определён, а CF = 0 и OF = 0, DF, IF, TF - не изменяются.

2.5.3.2.Команды сдвигов.

Команды сдвигов являются эффективным средством увеличения или уменьшения числа в 2 раза (меньше памяти и быстрее, чем в командах умножения и деления). Для умножения числа на 2, надо сдвинуть все биты на 1 разряд влево, а в освобождённый правый бит поместить 0. Если выдвинутый слева бит передать во флаг CF, то можно зафиксировать выход за диапазон, проверив условие CF = 1. Аналогично уменьшение беззнакового числа вдвое осуществляется сдвигом всех бит на один разряд вправо, а в освобождающийся бит помещается 0. Вдвигаемый справа бит передаётся во флаг CF, если CF = 1, то число нечётное. С беззнаковыми числами работает команда SHL и SHR, а команды SAL и SAR предназначены для знаковых чисел. SAR сохраняет знаковый бит неизменным, команда SHR помещает в знаковый бит 0 (но заносит 1 во флаг OF, если знак имеется). Отметим, что сдвиг вправо нечётного числа всегда даёт результат, который меньше половины числа, например:

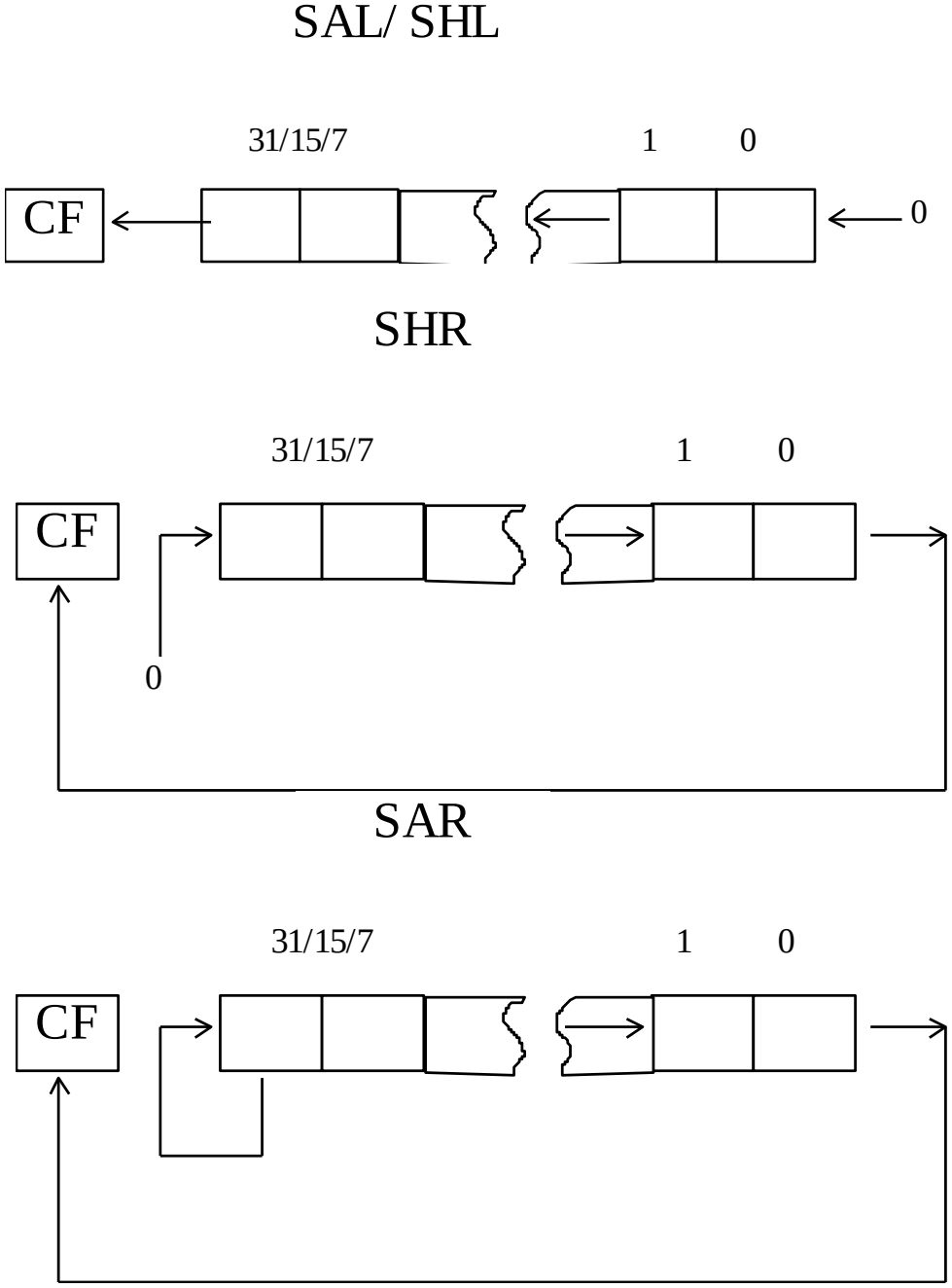
-5 (1111 1011) SAR -3 (1111 1101)

-3 < -2.5

При делении -5 на 2 командой DIV результат будет = -2. Различий между удвоением знакового и беззнакового чисел нет.

Рисунок 2.3.

Графическое представление работы команд сдвигов



2.5.3.3.Команды циклических сдвигов.

Команды циклических сдвигов позволяют переставить биты в числе. ROL - циклический сдвиг влево и ROR - циклический сдвиг вправо, обеспечивают циклический сдвиг. При этом выдвигающийся бит подаётся в освобождающийся бит. В командах RCL и RCR в кольцо сдвига включается флаг CF: выдвигающийся бит подаётся во флаг CF, а состояние флага CF передаётся в освобождающийся бит. Операнд команд сдвигов и циклических переносов может находиться в памяти или в регистре, длина операнда равна 8 или 16 бит. Сдвиг осуществляется на предопределённое число бит (фиксированный сдвиг). В первом случае число

сдвигов определяется в команде, а во втором - содержимым регистра CL (счётчика). В микропроцессоре 8086 фиксированный сдвиг осуществляется только на 1 бит.

Примеры команд приведены в таблице 2.11.

Таблица 2.11. Примеры команд циклических сдвигов.

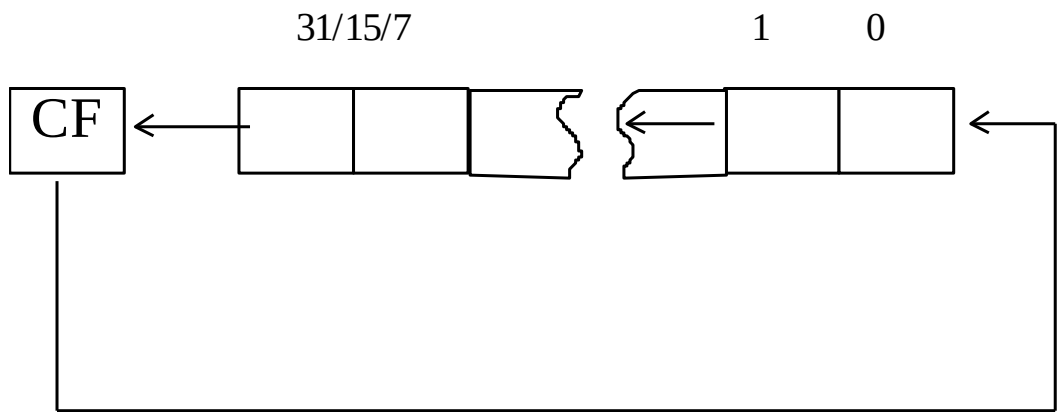
Операнд	Слово	Байт
Фиксированный сдвиг		
Регистр	КОП BX, 13	КОП DL, 1
Память	КОП MEMW, 15	КОП MEMB, 7
Переменный сдвиг		
Регистр	КОП AX, CL	КОП BL, CL
Память	КОП MEMW, CL	КОП MEMB, CL

КОП: SHR, SHL, SAL, SAR, ROR, ROL, RCR, RCL.

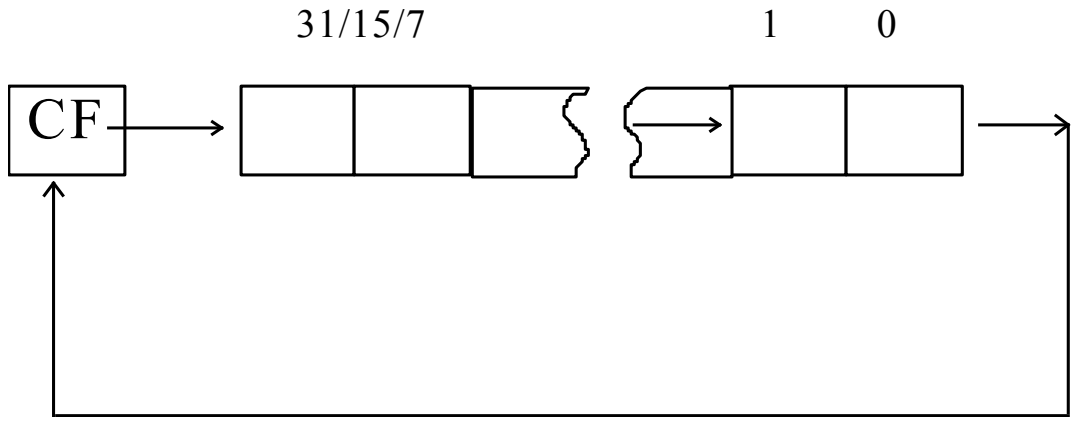
Графическое представление работы команд циклических сдвигов приведено на рисунке 2.4.

Рисунок 2.4. Графическое представление работы команд циклических сдвигов

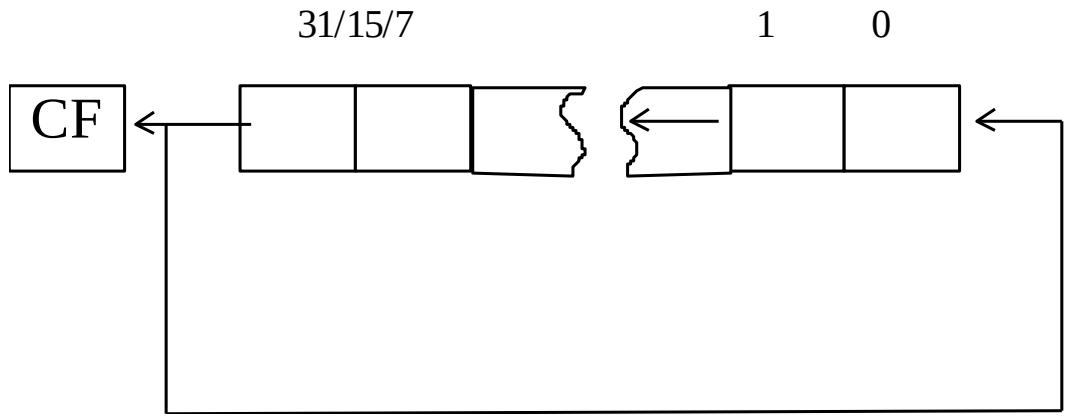
RCL



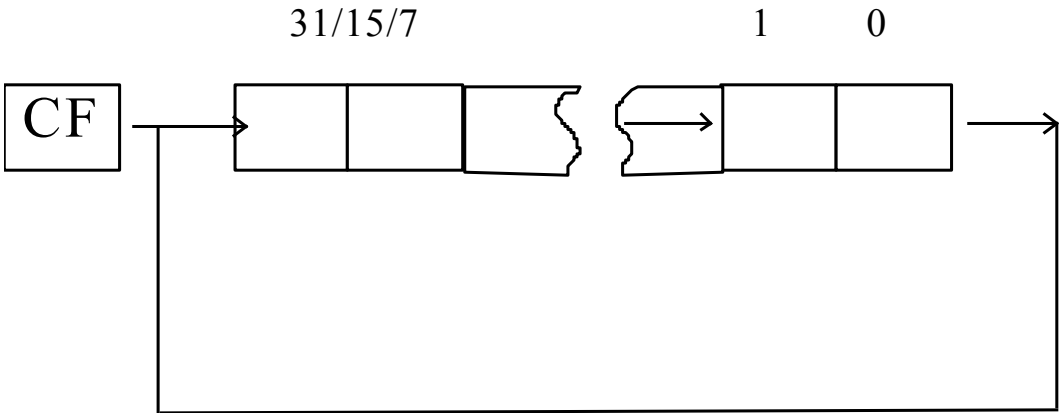
RCR



ROL



ROR



2.5.3.4.Команды двойного сдвига

Команды SHLD, SHRD введены в систему команд микропроцессора 80386. Предназначены для сдвигов двух 16-битных или двух 32-битных операндов. Форматы команд:

SHLD, SHRD reg/ mem 16, reg, imn

SHLD, SHRD reg/ mem 32, reg, imn, где

imn - непосредственное значение, которое определяет число сдвигов.

Внутри процессора два первых операнда объединяются в промежуточном регистре двойной длины, код которого сдвигается. После сдвига результат в приемнике, а источник не изменяется.

2.5.3.5.Команды работы с двоичными цепочками

Команды работы с двоичными цепочками введены в систему команд микропроцессора 80386 включают две группы :

1) Команда поиска бита BT выбирает бит в двоичной цепочке и передает его значение в CF. BTC- поиск и инвертирование бита в цепочке, BTS- поиск и установка бита, BTR- поиск и сброс бита. Форматы команд этой группы одинаковые, поэтому приведены только для команды BT:

BT приемник (двоичная цепочка), источник (номер бита в цепочке)

$$BT \begin{cases} reg/\ mem\ 16,\ reg\ 16 \\ reg/\ mem\ 32,\ reg\ 32 \\ reg/\ mem\ 16,\ imn\ 8 \\ reg/\ mem\ 32,\ imn\ 8 \end{cases}$$

2) Команды сканирования битов

BSF- вперед	предназначены для поиска в слове или двойном слове позиции первого единичного бита
BSR- назад	

Формат команд :

BSF,BSR приемник, источник

$$\text{BSF,BSR} \begin{cases} \text{reg 16, reg / mem 16} \\ \text{reg 32, reg / mem 32} \end{cases}$$

Приемник- номер первого единичного разряда.

Источник- битовая цепочка.

Результат исполнения команды фиксирует флаг ZF: ZF=0, если есть 1, и ZF=1, если источник равен 0 (в этом случае приемник не определен).

2.5.4.Команды передачи управления.

2.5.4.1.Команды безусловной передачи управления.

К командам настоящей группы относятся команды переходов, вызовов (процедур) и возвратов (из процедур). Переходы загружают значение в указатель команды IP, нарушая тем самым последовательное выполнение команд. Вызовы выполняют то же самое, но в начале они запоминают текущее значение содержимого указателя IP в стеке, так что при возврате можно было возобновить выполнение программы с этой точки.

Переходы, вызовы и возвраты бывают двух видов - внутрисегментные и межсегментные. Первые из них передают управление внутри текущего сегмента кода, вторые - в произвольный сегмент кода (изменяя содержимое регистра CS), который становится текущим сегментом кода. Межсегментный вызов сохраняет в стеке текущее содержимое регистров CS и IP. Межсегментный возврат соответственно восстанавливает из стека содержимое регистров CS и IP. В случае внутрисегментного вызова и возврата в стеке сохраняется и восстанавливается только указатель команды IP. Отметим, что все вызовы одной и той же процедуры должны быть либо внутрисегментными либо межсегментными, так как при возврате из процедуры должно извлекаться из стека столько байт, сколько было включено в стек при вызове.

Команда CALL осуществляет вызов (межсегментный и внутрисегментный) а команда RET соответствующий ей возврат. Для того чтобы сообщить ассемблеру, какие типы вызовов и возвратов генерировать для процедуры, последнюю ограничивают операторами PROC и ENDP.

Например:

```
UPCOUNT PROC NEAR (FAR)
ADD CX, 1
RET
```

UPCOUNT ENDP

Так как процедура объявлена как NEAR, то вызов и возврат будут внутрисегментными, если бы мы объявили её как FAR, то вызов и возврат были бы межсегментными, и команда CALL загрузила бы в стек сначала содержимое регистра CS, а затем IP.

Команда JMP осуществляет внутрисегментные и межсегментные переходы. Для того чтобы указать, какой переход осуществить, необходимо указать тип метки перехода (NEAR или FAR). При внутрисегментном переходе команда JMP занимает 3 байта, а при межсегментном - 5 байт. Существует разновидность команды JMP для коротких внутрисегментных переходов (-128 +127 байт от адреса команды JMP). Она занимает в памяти 2 байта. Для определения такой команды следует указать, что её операнд имеет тип SHORT. Например, JMP SHORT LABEL1.

До сих пор мы рассматривали только прямые переходы и вызовы. Однако команды JMP и CALL могут осуществлять и косвенный переход или вызов через регистр или ячейку памяти. Если для косвенной передачи управления используется регистр, то в нём должно быть смещение процедуры или метки перехода относительно регистра CS (регистра кодов). Если для косвенной передачи управления используется ячейка памяти, то микропроцессор 8086 (80286) по умолчанию будет считать, что она содержится в сегменте DS (если не указан префикс смены сегмента или не используется регистр BP для адресации ячейки); если используется BP - то микропроцессор будет считать, что ячейка памяти находится в сегменте стека и адресуется регистром SS.

Примеры команд безусловной передачи управления приведены в таблицах 2.12 и 2.13.

Таблица 2.12. Примеры внутрисегментых команд передачи управления.

Внутрисегментные

Прямые	Косвенные
JMP LABEL	JMP MEMLOC
CALL LABEL	JMP BX
RET	CALL WORD PTR MEMLOC
RET 6	CALL CX
	CALL WORD PRT ES: MEMLOC

Таблица 2.13. Примеры межсегментных команд передачи управления.

Прямые	Косвенные
JMP LABEL	JMP DWORD PTR MEMLOC
CALL LABEL	JMP DWORD PTR [BP]
RET	JMP DWORD PTR CS:[BX]

RET 6	CALL MEMLOC CALL DWORD PTR SS: MEMLOC

Тип перехода или вызова при косвенных передачах через ячейку памяти определяется по типу объявленной переменной (WORD или DWORD) или явно в команде посредством операции WORD PTR или DWORD PTR. Возврат не может быть косвенным, так как управление всегда возвращается в место вызова, однако существует разновидность возврата, которая после восстановления значений содержимого регистра IP, прибавляет к значению указателя стека константу, содержащуюся в команде как непосредственный операнд. В результате из стека извлекаются и уничтожаются дополнительные элементы (то есть, например, параметры, переданные процедуре). Удобнее, чем чистить стек командой INC SP после возврата из процедуры, так как при многократных вызовах процедуры это надо будет делать каждый раз.

2.5.4.2. Команды условных переходов.

Команды условных переходов вместе с командой CMP реализуют передачу управления в зависимости от отношения между двумя числами. Сначала процессор выполняет команду сравнения и устанавливает по результату флаги, а затем выполняет команду условного перехода, которая проверяет флаги и производит переход, если числа удовлетворяют заданному отношению. Числа могут проверяться на равенство или требуется узнать, какое из них больше или меньше. Какое число больше 11111111 или 00000000? Если число без знака, то первое = 255, а второе = 0 и первое больше второго. Если число со знаком, то первое = (-1) и меньше второго. Следовательно, отношения “меньше” или “больше” зависят от того, знаковые или беззнаковые числа. Поэтому целесообразно ввести новые термины, позволяющие различать эти два случая. При сравнении знаковых чисел пользуемся терминами “больше” и “меньше”, а при сравнении беззнаковых чисел терминами “выше” и “ниже” (так как беззнаковые числа обычно используются для сравнения адресов). Каждое из условий можно определить по состояниям флагов, поэтому в микропроцессоре есть команды, ориентированные на проверку отношений и команды, ориентированные на проверку состояния флагов. Перечень команд условных переходов приведен в таблице 2.14.

Таблица 2.14. Перечень команд условных переходов.

Команда	Описание	Состояние флагов
JE / JZ	Перейти, если равно / если ноль	ZF = 1
JNE /	Перейти, если не равно / если не	ZF = 0

JNZ	ноль	
JL / JNGE	Перейти, если меньше / если не больше и не равно	SF ≠ OF
JNL / JGE	Перейти, если не меньше / если больше или равно	SF = OF
JG / JNLE	Перейти, если больше / если не меньше и не равно	ZF = 0 & SF = OF
JNG / JLE	Перейти, если не больше / если меньше или равно	ZF = 1 & SF ≠ OF
JB / JNAE	Перейти, если ниже / если не выше и не равно	CF = 1
JNB / JAE	Перейти, если не ниже / если выше или равно	CF = 0
JA / JNBE	Перейти, если выше / если не ниже и не равно	CF = 0 & ZF = 0
JNA / JBE	Перейти, если не выше / если выше или равно	CF = 1 & ZF = 1
JC / JNC	Перейти, если есть перенос / если переноса нет	CF = 1 / CF = 0
JS / JNS	Перейти, если есть знак / если нет знака	SF = 1 / SF = 0
JO / JNO	Перейти, если переполнение / если не переполнение	OF = 1 / OF = 0
JP / JNE	Перейти, если есть паритет / если паритет четный	PF = 1
JNP / JPO	Перейти, если паритета нет / если паритет нечетный	PF = 0
JCXZ	Перейти, если CX = 0	CX = 0
JECXZ	Перейти, если ECX = 0	CX = 0

Команды условного перехода могут использоваться не только с командой CMP, но и с любой другой командой, воздействующей на флаги. Каждая из команд условного перехода в 80286 состоит из 8-битного кода операции и 8-бит, определяющих место перехода (то есть имеет размер 2 байта). 8 бит определяют относительное смещение места перехода и команды

условного перехода, то есть в диапазоне -128 +127 байт от команды. “Близкий” или “далёкий” условный переход всегда можно сделать при помощи двух команд: “короткого” условного перехода и “близкого” или “далёкого” безусловного перехода. В микропроцессоре 80386 команды условного перехода могут быть с типом NEAR.

2.5.4.3.Команды SetCondition

SET CC	- код условия совпадает с кодами условий в командах JCC
CC	условного перехода

Формат команд:

SETCC reg/ mem 8- если условияCC удовлетворяется, то в байт- операнд помещается код 01H, иначе - код 00H.

2.5.4.4 Команды управления циклами.

Команды управления циклами обеспечивают передачи управления при организации циклов. У микропроцессора регистр CX служит счётчиком 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 уменьшает содержимое регистра CX на 1 и передаёт управление операнду 'близкая' метка, если содержимое регистра CX не равно 0.

Например:

```
MOV CX, 100 ; Загрузить счётчик циклов (CX ≠ 0 в начале цикла!!)
START:          ...;
...;
LOOP START    ; Если CX ≠ 0, то перейти к метке START, иначе выйти из цикла
...;
```

Выход из цикла до достижения CX=0 обеспечивают команды LOOPE/LOOPZ (повторять, если равно) и LOOPNE/LOOPNZ (повторять, пока не равно). Команда LOOPE / LOOPZ уменьшает содержимое CX на 1, затем осуществляет переход, если CX ≠ 0 и флаг нуля ZF=1. Обычно LOOPE используют для поиска первого ненулевого результата в серии операций. Команда LOOPNE / LOOPNZ уменьшает CX на 1, затем осуществляет переход, если CX ≠ 0 и флаг нуля ZF=0. Обычно LOOPNE используется для поиска первого нулевого результата в серии операций. Для того чтобы определить, по какому условию произошел выход из цикла можно использовать команду JCXZ (CX=0).

2.5.5.Цепочечные (строковые) команды.

Под цепочкой (строкой) понимается последовательность байт, слов или двойных слов в памяти, а цепочечной операцией называется операция, которая выполняется над каждым элементом цепочки. Микропроцессор имеет набор команд, которые сокращают время выполнения операций по обработке цепочек. Сокращение времени достигается за счёт мощного набора примитивных команд, ускоряющих обработку каждого элемента цепочки и за счёт устранения служебных действий, которые обычно требуются между обработкой последовательных элементов. Перечень цепочечных команд приведен в таблице 2.15.

Таблица 2.15. Перечень цепочечных команд.

Мнемоника	Описание	Модифицирует
MOVS (передать цепочку)	источник → приемник	SI, DI (ESI,EDI)
CMPS (сравнить цепочки)	источник- приемник →	SI, DI (ESI,EDI)
SCAS (сканировать цепочку)	AL (AX,EAX)- приемник →	DI (EDI)
*INS (ввести цепочку)	входной порт → приемник	DI (EDI)
*OUTS (вывести цепочку)	источник → выходной порт	SI (ESI)
LODS (загрузить цепочку)	источник → AL (AX,EAX)	SI (ESI)
STOS (запомнить цепочку)	AL (AX,EAX) → приемник	DI (EDI)

* В операциях над словами вместо AL используется AX, для двойных слов- EAX. По умолчанию микропроцессор предполагает, что цепочка-приёмник находится в дополнительном сегменте и адресуется парой регистров ES : DI (или для 32 разрядной адресации - ES:EDI), а цепочка-источник в текущем сегменте данных и адресуется парой регистров DS : SI (для 32 разрядной адресации - ES:ESI). Цепочечные команды автоматически модифицируют указатели для адресации следующего элемента цепочки. Флаг направления DF определяет направление обработки цепочки. Если DF = 0, то значения регистров DI и SI увеличиваются, а если DF = 1, то значения регистров DI и SI уменьшаются. В первом случае мы двигаемся по цепочке в прямом направлении (слева - направо), во втором случае - в обратном (справа - налево).

Состоянием флага DF можно управлять с помощью двух команд: CLD (DF ← 0) и STD (DF ← 1). Одна команда обработки цепочки обрабатывает один элемент цепочки длиной 8, 16 или 32 бит.

Для того чтобы одна команда обработала группу последовательных элементов перед ней надо указать префикс повторения. При этом число повторений извлекается из регистра CX. Префикс REP даёт указание повторять команду, пока не обнаружится конец цепочки, то есть CX = 0. Префиксы REPE и REPZ (повторять пока равно) повторяют цепочечную команду, пока флаг ZF = 1 и CX ≠ 0.

Например:

MOV CX, 100; поэлементное сравнение

REPE CMPS DEST, SOURCE; DEST и SOURCE до первого несовпадения

REPNE / REPNZ; (повторять, пока не равно) обеспечивают повторение команды, пока ZF = 0 и CX ≠ 0. Если в предыдущем примере REPNE CMPS DEST, SOURCE будет повторяться до первого совпадения.

Префиксы REPE / REPZ и REPNE / REPNZ используются с командами сравнения или сканирования цепочки, так как они воздействуют на флаг ZF.

2.5.5.1.Команды пересылки цепочки.

Формат общей команды: MOVS цепочка-приёмник, цепочка-источник

Дополнительные команды:

MOVSB

MOVSW

MOVSD

В команде MOVS операнды нужны только для того, чтобы ассемблер узнал, что нужно пересылать байты или слова, так как смещение цепочки-приёмника предварительно должно быть загружено в регистр DI, а смещение цепочки-источника в регистр SI. Ассемблер преобразует команду MOVS в одну из команд MOVSB, MOVSW или MOVSD, следовательно, можно сразу их употреблять в программе.

Примеры:

```
MEMW1 DW 10 DUP (1), 5 DUP (0)
MEMW2 DW 15 DUP (?)
MOV AX, @DATA
MOV DS, AX
MOV ES, AX
CLD
MOV SI, OFFSET MEMW1
MOV DI, OFFSET MEMW2
MOV CX, 15
REP MOVS MEMW2, MEMW1; ⇔ REP MOVSW
```

2.5.5.2.Команды сравнения цепочек.

Формат общей команды: CMPS цепочка-приёмник, цепочка-источник

Дополнительные команды:

```
CMPSB
CMPSW
CMPSD
```

Подобна CMP, однако CMPS производит обратное по отношению с CMP вычитание, вычитает из источника приёмник, а CMP, наоборот - из приёмника источник. CMPSB и CMPSW - соответствующие версии команды сравнения строк байтов или строк слов. Если мы используем префиксы повторения REPE / REPZ или REPNE / REPNZ, то операция сравнения может завершиться в двух случаях: CX=0 или ZF=0 для REPE (ZF=1 для REPNE). Для того чтобы узнать, какая ситуация имела место, следует указать после CMPS команду условной передачи управления, проверяющую флаг ZF, а именно JE (JZ) или JNE (JNZ).

Пример:

```
CLD
MOV CX, 100
REPNE CMPS DEST, SOURCE
```

JNE NOT_FOUND; Переход к метке NOT_FOUND, если нет ни одной совпадающей пары.

...

NOT_FOUND:

...

2.5.5.3.Команды сканирования цепочек.

Команды сканирования цепочек позволяют осуществить поиск заданного значения в цепочке, находящейся в дополнительном сегменте. Смещение адреса первого элемента цепочки должно быть помещено в регистр DI. При сканировании цепочки байтов искомое значение должно находиться в регистре AL, при сканировании цепочки слов - в регистре AX, а при сканировании цепочки двойных слов - в регистре EAX.

Формат основной команды: SCAS цепочка-приёмник.

Дополнительные команды:

SCASB,

SCASW,

SCASD.

Если при сканировании обнаружен заданный элемент, то в DI смещение адреса следующего за ним элемента, а ZF = 0. С командами данной группы можно использовать префиксы REPE / REPNE.

2.5.5.4.Команды загрузки.

Команда LODS пересылает операнд цепочка-источник, адресованный регистром SI из сегмента данных в регистр AL (AX или EAX) а затем изменяет регистр SI так, чтобы он указывал на следующий элемент цепочки.

Формат основной команды: LODS цепочка-источник.

Дополнительные команды:

LODSB,

LODSW,

LODSD.

2.5.5.5.Команды сохранения цепочек.

Команда STOS пересылает цепочку байт, слов или двойных слов из регистра AL (AX или EAX) в элемент операнда цепочка-приёмник, адресуемый парой регистров ES : DI.

Формат основной команды: STOS цепочка-приемник.

Дополнительные команды:

STOSB,

STOSW,

STOSD.

С командами данной группы можно использовать префикс REP.

2.5.5.6.Команды ввода и вывода цепочек.

Обеспечивают считывание данных из входного устройства в последовательные ячейки памяти и запись данных из последовательных ячеек в выходное устройство. Они упрощают передачи больших блоков данных между памятью и внешними устройствами. INS передаёт данные из входного порта, определяемого содержимым регистра DX в байт или слово, смещение которого находится в ES:DI (ES:EDI) и производит уменьшение или увеличение DI на 1 (или 2), или EDI на 4. Аналогично OUTS передаёт байт, слово или двойное слово, смещение которого находится в DS:SI (DS:ESI), в выходной порт, адресуемый регистром DX и производит увеличение или уменьшение содержимого регистра SI на 1 (или 2) или ESI на 4. Команд INS и OUTS в 8086 нет. Дополнительные команды:

INSB,

INSW,

INSD,

OUTSB,

OUTSW,

OUTSD.

С командами данной группы можно использовать префикс REP.

2.5.5.7.Замена сегмента.

Можно ли изменить адресацию регистра SI (ESI) с сегмента данных на дополнительный сегмент? Можно, если использовать префикс замены сегмента, например:

LEA SI, ES : H1; копирует байт из строки H1 в строку H2. Обе строки в дополнительном сегменте

LEA DI, H2

MOVSB

Нельзя заменить сегмент, к которому адресуется регистр DI (EDI). Для того чтобы работать со строками в сегменте данных, надо в регистр ES загрузить значение, равное содержимому регистра DS.

Например:

MOV AX, @DATA

MOV ES, AX

MOV DS, AX

или:

PUSH DS

POP ES

2.5.6.Команды управления микропроцессором.

2.5.6.1.Команды управления флагами.

Микропроцессор имеет команды для установки и сброса флага переноса (STC, CLC), флага направления (STD, CLD) и флага прерывания (STI, CLI). Есть также команда инвертирования флага переноса (CMC). Перечень команд управления процессором приведена в таблице 2.16.

Таблица 2.16. Перечень команд управления процессором.

Мнемоника	Описание
CLC сбросить флаг переноса	0 → CF
CMC инвертировать флаг переноса	1 - CF → CF
STC установить флаг переноса	1 → CF
CLD сбросить флаг направления	0 → DF
STD установить флаг направления	1 → DF
CLI сбросить флаг разрешения прерывания	0 → IF
STI установить флаг разрешения прерывания	1 → IF

CLI микропроцессор игнорируют маскируемые прерывания, но обрабатывает немаскируемые прерывания.

2.5.6.2.Команды синхронизации.

Одним из средств синхронизации процессора с внешними устройствами являются прерывания, но в его архитектуре реализованы ещё две формы синхронизации: первая относится к использованию сопроцессора, вторая - к разделению ресурсов с другими процессорами в мультипроцессорной системе. Например, для реализации вещественной арифметики целесообразно использовать специальный арифметический сопроцессор. Он работает только во взаимодействии с основным процессором .

В частности, когда основной процессор встречает специальную команду ESC, он передает ее на исполнение сопроцессору. Команда ESC показывает, какую операцию должен

выполнить сопроцессор (код операции), эту информацию основной процессор игнорирует, но команда ESC задаёт для сопроцессора операнд в памяти (или регистре). Эту информацию основной процессор использует так: он вычисляет адрес операнда, а затем считывает его или записывает в память по запросу процессора. ESC применяется вместе с командой ожидания WAIT, обеспечивающей синхронизацию процессоров; с её помощью основной процессор проверяет, когда сопроцессор заканчивает свою операцию.

Если в многопроцессорной системе возникает проблема разделения общей памяти, то для избежания использования не полностью откорректированной информации вторым процессором при проведении операций коррекции первым в системе команд микропроцессора существует префикс блокировки шины LOCK. Например, чтобы нельзя было прервать команду пересылки элементов строки можно написать: LOCK REP MOVS. Такая команда выдает сигнал LOCK на всё время выполнения команды MOVS, что запрещает другим процессорам доступ к памяти.

2.5.6.3.Команда холостого хода.

Команда NOP (нет операции) - не выполняет никакой операции, она только увеличивает значение указателя команд IP. Эта команда находит много применений: удобна при тестировании в качестве точки останова трассировки, используется для организации задержки и так далее.

2.5.6.4.Команды прерываний.

Средства прерывания работы процессора внешними устройствами освобождают его от периодической проверки необходимости обслуживания устройств (циклического опроса). Микропроцессор имеет два входа, по которым внешние устройства могут привлечь его внимание: вход NMI немаскируемого прерывания и вход INTR маскируемого прерывания.

Когда внешнее устройство формирует сигнал на входе NMI, процессор прекращает свои действия (но не в середине команды) и реагирует на прерывание. По входу NMI должны появляться только прерывания из-за катастрофических событий (например, отказ сети, ошибка памяти).

Если катастрофических событий нет, внешнее устройство может прервать процессор по входу INTR. Процессор может игнорировать этот сигнал. Это зависит от состояния флага IF: если IF = 0 (процессор не реагирует на прерывания), если IF = 1 - реагирует.

Кроме сигнала на входе INTR внешнее устройство должно сообщить процессору причину прерывания. По запросу процессора внешнее устройство сообщает число из диапазона 0..255, соответствующее причине прерывание, которое называется номером прерывания. Для

каждого номера прерывания существует своя программа, которую процессор должен выполнить до возобновления прерванной задачи. Адреса этих программ находятся в 256-элементной таблице векторов прерываний. Каждый её элемент состоит из 4-х байт и содержит полный адрес процедуры обработки соответствующего прерывания (вектор прерывания).

При поступлении сигнала INTR (IF = 1) процессор завершает текущую команду и готовится выполнить процедуру прерывания. Прежде всего он сохраняет в стеке текущее состояние регистра флагов и регистров CS и IP. Затем процессор получает от внешнего устройства номер прерывания и помещает в IP и CS вектор прерывания по соответствующему номеру. Когда приходит прерывание по входу NMI (независимо от флага IF), процессор производит аналогичные действия, за исключением получения номера прерывания, так как причина прерывания по входу NMI всегда одна. Для процедуры прерывания NMI отведён элемент с номером 2 в таблице прерываний.

Выше речь шла о прерываниях, формируемых внешними устройствами, так называемых внешних прерываниях. Но процессор сам генерирует внутренние прерывания, если при выполнении некоторых команд происходит что-то неожиданное. Это так называемые особые случаи, которые свидетельствуют о серьёзной ошибке. Первый 32 элемента в таблице прерываний зарезервированы для внутренних прерываний и прерывания NMI. Процессор 8086 и все последующие в реальном режиме работы генерирует только первые 5 номеров прерываний. Процедура прерывания заканчивается выполнением команды IRET (возврат из прерывания), которая восстанавливает из стека содержимое IP, CS и флагов.

Некоторые процедуры прерываний удобно вызывать и тогда, когда прерываний нет (например функции DOS или BIOS). Поэтому было бы хорошо иметь команду, которая делала бы то, что делает процессор при распознавании прерываний, за исключением одного - номер прерывания содержится в команде, а не выдаётся внешним устройством. Такой командой и является команда INT n, где n - целое число от 0 до 255. Перечень команд прерываний приведен в таблице 2.17.

Таблица 2.17. Перечень команд прерываний.

Мнемоника	Описание
INT n	Прерывание типа n (длина 2 байта)
INT	Прерывание типа 3 (один байт) применяется в отладчиках для реализации останова)
INTO (IF = 1, OF = 1)	Прерывание по переполнению (если OF=1) генерирует прерывание №4

IRET	Возврат из прерывания
HLT	Останов. Прекращает все действия процессора, пока не будет произведен аппаратный сброс или не придет внешнее прерывание (NMI всегда выводит из состояния прерывания, остальные-только если IF=1)

При установке точки останова в отладчике, требуется поместить в точку останова какую-либо команду, которая передала бы управление отладчику. Для этой цели подошла команда INT, которая сохраняет регистры CS и IP. Так как тип прерывания для этой цели не надо передавать, то используют однобайтную форму команды. Отладчик использует команду INT для генерирования прерывания типа 3 при отладке программ, следовательно в наших программах не следует использовать это прерывание.

Прерывания компьютеров семейства PC могут быть разделены как правило на 3 группы: внутренние прерывания микропроцессора, внешние (аппаратные) прерывания, программые прерывания.

2.5.7.Новые команды микропроцессора 80486

В системе команд микропроцессора 80486 появились новые команды BSWAP, XADD, CMPXCHG.

1. BSWAP reg 32- обмен байт в операнде.

(0÷7) разряды ↔ (24÷31) разрядами,

(8÷15) разряды ↔ (16÷23) разрядами.

Используется для преобразования форматов данных с различной адресацией: с младшей стороны и со старшей стороны.

2. XADD mem 8/ 16/ 32(приемник), reg 8/ 16/ 32(источник)

источник= приемнику,

приемник= приемник + источник

3. CMPXCHG mem 8/ 16/ 32 (приемник), reg AL/ AX/ EAX (ист.)

Если приемник= AX, то приемник= источнику, иначе AX=приемнику и ZF=0. Используется для проверки и модификации семафоров.

3.ДИРЕКТИВЫ И ОПЕРАТОРЫ АССЕМБЛЕРА

Каждая программа на языке ассемблера помимо команд процессора содержит еще и специальные инструкции, указывающие самому ассемблеру, как организовать специальные секции программы, где располагаются данные, а где команды, позволяющие выбирать тип системы команд, налаживать связи между процедурами и так далее. Разные ассемблеры используют разные наборы директив, но TASM и MASM (два самых популярных ассемблера для DOS и Windows) поддерживают общий набор директив. Все дальнейшие примеры директив приведены для этих ассемблеров (точнее, TASM поддерживает набор директив MASM наряду с собственным набором директив Ideal Mode).

3.1.Структура программы

Программа на языке ассемблера состоит из строк, имеющих следующий вид:

Метка команда/директива операнды; комментарий.

Все поля необязательны. Метка может быть любой комбинацией букв английского алфавита, цифр и символов: _, $, @, ?, но цифра не может быть первым символом метки, а символы $ и ? иногда имеют специальное значение и не рекомендуются к использованию.

Большие и маленькие буква по умолчанию не распознаются, но различие можно включить при помощи опций при вызове ассемблера. Во втором поле, поле команды, может располагаться команда процессора, которая транслируется в исполняемый код, или директива ассемблера, которая не приводит к генерации кода, а управляет работой самого ассемблера. В поле операндов располагаются требуемые командой или директивой операнды. В поле комментария, начало которого отмечается символом ; (точка с запятой), можно написать все что угодно.

Если метка располагается перед командой процессора, сразу после нее всегда ставится оператор : (двоеточие), который указывает ассемблеру, что надо создать переменную с этим именем, содержащую адрес текущей команды:

```
mov cx, 10; занести в cx число элементов массива mas
xor si, si; обнулить индексный регистр si
xor ax, ax; обнулить регистр ax, в котором будет сумма элементов массива слов mas
m1:     add ax, mas[si]; сложить содержимое ax и очередной элемент массива
        inc si; увеличить индекс элемента массива на 2,
        inc si; так как элемент массива - слово
  loop  m1; вернуться на начало цикла, если cx>0.
```

Если метка стоит перед директивой ассемблера, то двоеточие не ставится, так как метка в этом случае является операндом директивы. Рассмотрим директивы, работающие напрямую с метками и их значениями, - LABEL, EQU и =.

метка LABEL тип.

Директива LABEL определяет метку и задает ее тип: BYTE (байт), WORD (слово), DWORD (двойное слово), FWORD (6 байт), QWORD (учетверенное слово), TBYTE (10 байт), NEAR (ближняя метка), FAR (дальняя метка). Метка получает значение, равное следующим за ней данным (для всех типов, кроме двух последних) или равное адресу следующей команды (для типов NEAR и FAR). С помощью директивы LABEL удобно организовывать доступ к одним и тем же данным, как к байтам, так и словам, определив перед ними две метки с разными типами. Метки типа NEAR или FAR можно использовать для вызова процедуры:

call метка.

Директива EQU присваивает метке значение, которое определяется как результат целочисленного выражения в правой части. Результатом этого выражения может быть целое число, адрес или любая строка символов:

```
one     equ     1
mes1    equ     ‘Ошибка $’
var2    equ     4[si]
        cmp     ax, one ; cmp ax, 1
        db      mes1    ; db ‘Ошибка $’
        mov     ax, var2        ; mov ax, 4[si].
```

Директива EQU используется для введения параметров, общих для всей программы.

Директива = эквивалентна EQU, но определяемая ею метка может принимать только целочисленные значения, кроме того, указанная этой директивой метка может быть переопределена. Существует предопределенная метка $, которая соответствует текущему адресу. Данную метку удобно использовать для определения длины строки текста или числа элементов в массиве:

```
.data
path1 db ‘D:\students\641\*.*’
length equ $ - path1; метка length будет иметь значение, равное числу символов в переменной path1.
```

Псевдокоманды определения данных

Псевдокоманда – это директива ассемблера, которая приводит к включению данных или кода в программу, хотя никакой команде процессора не соответствует. Псевдокоманды определения переменных указывают ассемблеру, что соответствующем месте программы располагается переменная, устанавливают ее тип (байт, слово, вещественное число и так далее) , задают начальное значение и ставят в соответствие переменной метку, которая будет использоваться для обращения к этим данным. Псевдокоманды определения данных записываются в общем виде следующим образом:

имя_переменной d* значение,

где d* - одна из псевдокоманд:

DB – определить байт;

DW – определить слово (2 байта);

DD – определить двойное слово (4 байта);

DF – определить 6 байт (адрес в формате 16-битный селектор; 32-битное смещение);

DQ – определить учетверенное слово (8 байта);

DT – определить 10 байт (80 –битные типы данных, используемые FPU).

Пример использования псевдокоманд определения переменных:

```
txt_str  db    ‘Hello, world!’
number   dw    ?
table    db    10 dup(0)
float_n  dd    3.1415.
```

Первая строка содержит 13 байт ASCII- кодов строки символов Hello, world! и переменная txt_str указывает на первую букву в этой строке. Во второй строке знак ? указывает на то, что переменная является неинициализированной и ее значение на момент запуска может оказаться любым. Для заполнения участка памяти повторяющимися данными используется оператор DUP, имеющий формат счетчик dup(значение). В третьей строке примера создается массив из 10 слов, инициализированных нулевым значением. Переменная table указывает на первый элемент массива. В качестве аргумента в операторе DUP могут выступать несколько значений, разделенных запятыми, и даже дополнительные вложенные операторы DUP.

3.2.Организация программы.

3.2.1. Модели памяти

Модели памяти задаются директивой .MODEL.

model модель, язык, модификатор, где модель – одно из следующих слов:

TINY – код, данные и стек размещаются в одном и том же сегменте размером до 64 Кб. Эта модель памяти используется для написания программ на языке ассемблера в формате COM файла;

SMALL – код размещается в одном сегменте, а данные и стек – в другом (для их описания могут применяться разные сегменты, но объединенные в одну группу). Эта модель памяти используется для написания программ на языке ассемблера в формате EXE файла;

COMPACT – код размещается в одном сегменте, а для хранения данных могут использоваться несколько сегментов;

MEDIUM – код размещается в нескольких сегментах, а все данные – в одном;

LARGE, HUGE – и код, и данные могут занимать несколько сегментов;

FLAT – то же, что и TINY, но используются 32 – битные сегменты, так что максимальный размер сегмента, содержащего и данные, и код, и стек, - 4 Мб.

Язык – необязательный операнд, принимающий значения различных языков программирования, при указании которого подразумевается, что процедуры рассчитаны на вызов из программ на соответствующем языке программирования.

Модификатор – это необязательный операнд, принимающий значения NEARSTACK (по умолчанию) или FARSTACK. Во втором случае сегмент стека не будет объединяться в одну группу с сегментами данных.

Упрощенные директивы определения сегментов

Каждая программа, написанная на любом языке программирования, состоит из одного или нескольких сегментов. Область памяти, в которой находятся команды, называется сегментом кода, область памяти с данными – сегментом данных и область памяти, отведенная под стек – сегментом стека. Ассемблер позволяет помещать данные в сегмент кода, разносить код по нескольким сегментам, помещать стек в один сегмент с данными или вообще использовать один сегмент для всего. Для описания сегментов программы используются директивы SEGMENT и ENDS. Однако, чаще всего используются упрощенные директивы определения сегментов, которые вступают в силу после установления модели памяти.

Директива .CODE описывает сегмент кода:

.code.

Директива .STACK описывает сегмент стека:

.stack размер.

Директива .DATA описывает основной сегмент данных:

.data.

3.2.2. Процедуры

Процедурой в ассемблере является всё то, что в других языках называют подпрограммами, функциями, процедурами и т.д. Ассемблер не накладывает на процедуры никаких ограничений- на любой адрес программы можно передать управление командой CALL, и оно вернётся к вызвавшей процедуре, как только встретится команда RET. Такая свобода выражения легко может приводить к трудночитаемым программам, и в язык ассемблера были включены директивы логического оформления процедур.

Метка proc язык тип USES регистры

…

ret

метка endp

Все операнды PROC необязательны.

Тип может принимать значения NEAR и FAR, и если он указан, все команды RET в теле процедуры будут заменены соответственно на RETN и RETF. По умолчанию подразумевается, что процедура имеет тип NEAR в моделях памяти TINY, SMALL и COMPACT.

Конец программы

End start_label

Этой директивой завершается любая программа на ассемблере. В роли необязательного операнда здесь выступает метка (или выражение), определяющая адрес, с которого начинается выполнение программы. Если программа состоит из нескольких модулей, только один файл может содержать начальный адрес, так же как в С только один файл может содержать функцию main().

3.2.3. Директивы задания набора допустимых команд

По умолчанию ассемблеры используют набор команд процессора 8086 и выдают сообщения об ошибках, если выбирается команда, которую этот процессор не поддерживал. Для того чтобы ассемблер разрешил использование команд, появившихся в более новых процессорах, и команд расширений, предлагаются следующие директивы:

.8086- используется по умолчанию. Разрешены только команды 8086;

.186- разрешены команды 80186;

.286 и .286c- разрешены непривилегированные команды 80286;

.286p- разрешены все команды 80286;

.386 и .386c- разрешены непривилегированные команды 80386;

.386p- разрешены все команды 80386;

.486 и .486c- разрешены непривилегированные команды 80486;

.486p- разрешены все команды 80486;

.586 и .586c- разрешены непривилегированные команды P5 (Pentium);

.586p- разрешены все команды P5 (Pentium);

.686 - разрешены непривилегированные команды P6 (Pentium Pro, Pentium II);

.686p- разрешены все команды P6 (Pentium Pro, Pentium II);

.8087 – разрешены команды арифметического сопроцессора 8087;

.287 – разрешены команды арифметического сопроцессора 80287;

.387 – разрешены команды арифметического сопроцессора 80387;

.487 – разрешены команды FPU 80486;

.587 – разрешены команды FPU 80586;

.MMX – разрешены команды IA MM;

.K3D – разрешены команды AMD 3D;

Если присутствует директива .386 и выше, ассемблер TASM определяет все сегменты как 32-битные при условии, что директива указана перед директивой .model.

3.3.Примеры использования директив в программах типа .EXE и .COM.

; Файл с текстом программы hello-1.asm

; Выводит на экран сообщение ‘Hello World!’ и завершается.

; Генерируется исполняемый модуль типа .EXE при помощи вызова ассемблера TASM и редактора TLINK:

```
; tasm hello-1.asm
; tlink hello-1.obj
  .model small   ;        Модель памяти, используемая для EXE
  .stack  100h   ;                Сегмент стека размером в 256 байт.
  .data
message db      'Hello World!',0Dh,0Ah,'$'
  .code
start:    mov   ax, @data      ; Настройка сегментного регистра
  mov     ds,ax          ; ds на начало сегмента данных.
  mov     dx,offset message
  mov     ah,9
  int     21h            ; Вызов функции DOS для вывода строки.
  mov     ax,4C00h
  int     21h            ; Вызов функции DOS для завершения программы.
```

```
  end     start
; Файл с текстом программы hello-2.asm
; Выводит на экран сообщение ‘Hello World!’ и завершается.
; Генерируется исполняемый модуль типа .COM при помощи вызова ассемблера TASM и редактора TLINK:
; tasm hello-2.asm
; tlink /t hello-2.obj
  .model tiny             ; Модель памяти, используемая для .COM
  .code                   ; Начало сегмента кодов
  org     100h            ; Начальное значение программного счетчика-
; внутренней переменной ассемблера, равная
;смещению относительно начала сегмеента - 100h
  jmp start               ; Переход на начало программы
message db      'Hello World!',0Dh,0Ah,'$'    ; Строка для вывода
start:  mov     ah,9            ; -Номер функции DOS - в AH
  mov     dx,offset message     ;  Смещение адреса строки - в DX
  int     21h             ; Вызов системной функции DOS
  ret                     ; Завершение COM- программы
  end     start           ; Конец текста программы.
```

4.АРХИТЕКТУРА И СИСТЕМА КОМАНД АРИФМЕТИЧЕСКОГО СОПРОЦЕССОРА

Арифметический сопроцессор предназначен для вычислений над числами с плавающей точкой и может работать только в паре с основным процессором. Применение сопроцессора в задачах, использующих сложные вычисления (машинная графика, статические и инженерные расчеты) позволяет повысить быстродействие системы.

Первый из семейства – это восьмиразрядный сопроцессор 8087, затем появились шестнадцатиразрядные модели 80287 и 80387. Сопроцессор 80387 - имеет расширенный набор команд по сравнению с моделью 80287 и большее быстродействие. В микропроцессорах 80486 – DX, а затем и во всех последующих моделях существует встроенное устройство для выполнения операций с числами с плавающей точкой (FPU), являющееся одним из модулей основного процессора. Это позволило еще больше увеличить производительность персонального компьютера.

4.1.Форматы чисел сопроцессора

Сопроцессор поддерживает представление чисел в следующих форматах: целые двоичные, целые двоично-десятичные и вещественные двоичные числа.

4.1.1.`Целые числа

Целые двоичные числа могут быть представлены в трех различных форматах:

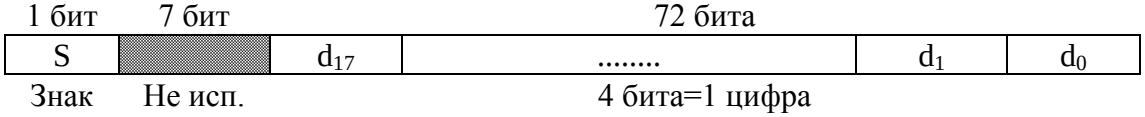
1. Целое слово - 16 бит (DW);

2. Короткое целое слово - 32 бита (DD);

3. Длинное целое слово - 64 бита (DQ);

Все они представимы в обычном формате целых чисел со знаком.

Двоично-десятичные целые числа представлены в упакованном двоично-десятичном формате размером 80 бит (DT), при этом каждая десятичная цифра кодируется четырьмя двоичными разрядами.



4.1.2. Вещественные числа

Вещественные числа могут быть представлены тремя различными форматами:

1. Короткое вещественное (одинарная точность) -32 бита (DD);

бит

	По- рядок	мантисса

DD

2. Длинное вещественное (двойная точность) - 64 бита (DQ);

11 бит 52 бита

бит

	Порядок	Мантисса

DQ

3. Временное вещественное (расширенная точность) - 80 бит (DT);

15 бит 64 бита

бит

	Порядок	мантисса

]

T

1) **S**- поле знака: равно нулю, если число положительное и равно 1, если число отрицательное;

Для представления нуля используются нулевой порядок и нулевая мантисса, следовательно существует два нуля: положительный и отрицательный. Обычно этот факт скрыт от программиста - в командах сравнения оба пуля считаются одним числом.

2) Поле мантиссы.

В сопроцессоре принята “научная нотация“ представления чисел. При этом предполагается, что поле мантиссы представимо в виде X.XX...X, где старший бит, находящийся слева от точки должен быть не равен нулю. Для двоичной системы счисления это означает, что старший бит должен быть равен единице. Следовательно, его можно не хранить, что и сделано в форматах одинарной и двойной точности для расширения диапазона представления чисел. В формате с расширенной точностью старший бит хранится явно, так как дополнительная точность не требуется, а вычисления происходят быстрее.

3) Поле порядка. Порядок - это степень числа два, на которое надо умножить мантиссу.

Для представления отрицательных порядков в поле порядка хранится сумма истинного порядка и смещение. Для одинарной точности смещение равно 127. Для двойной точности смещение равно 1023. Для расширенной точности смещение равно 16383.

Если поле порядка составляет 8 бит как в формате с одинарной точностью, то максимальное значение порядка составляет 2^8-1 = 255 , то есть диапазон хранимых порядков рассчитывается следующим образом:

255 - 127 = 128 – максимальный порядок;

0 - 127 = -127 – минимальный порядок.

Аналогично можно рассчитать диапазоны хранимых порядков для форматов чисел с двойной и расширенной точностью.

Так как во всех форматах чисел минимальное и максимальное значения порядков зарезервированы для обработки особых случаев, то реальный диапазон порядка нормализованных чисел меньше на два для всех форматов чисел, то есть для чисел с одинарной точностью максимальный порядок равен 127 ,а минимальный порядок равен 126.

Формат чисел с двойной точностью позволяет получить произведение двух чисел с одинарной точностью без потери точности.

Формат чисел с расширенной точностью предназначен для представления промежуточных результатов при работе с числами в формате с двойной точностью. Существует три причины выбора 80 – разрядного представления чисел в формате с расширенной точностью:

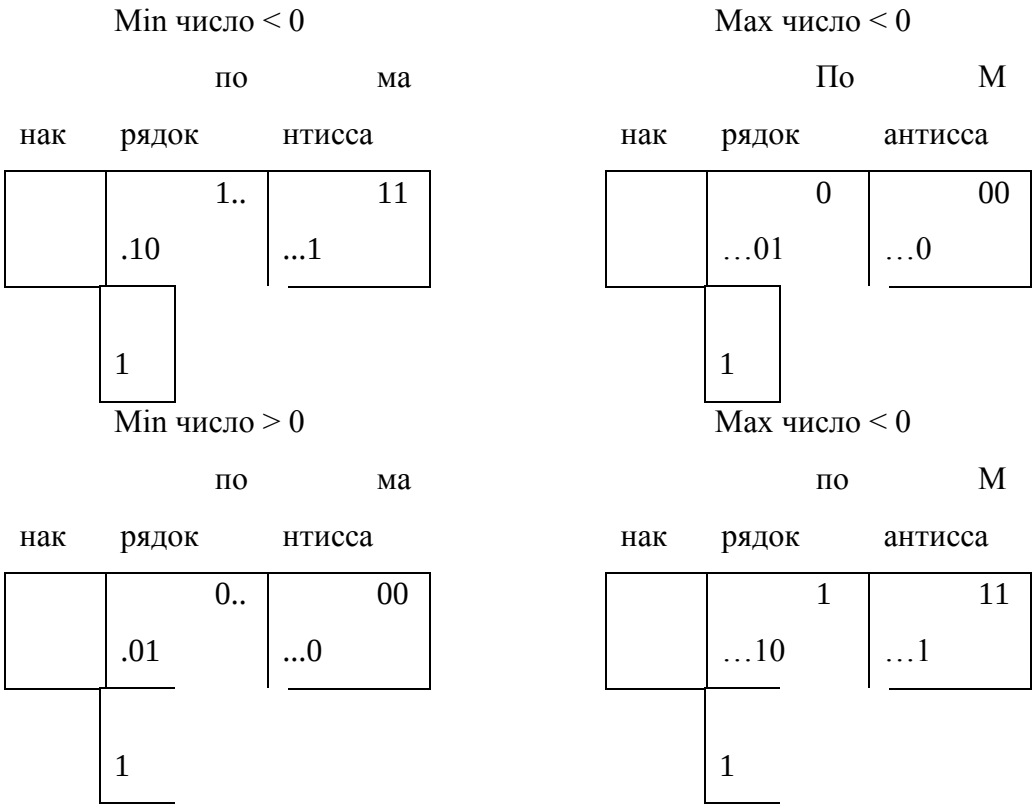
Исследования показали, что для самых сложных для вычислений математических функций не происходит потери точности при получении окончательных результатов в формате чисел с двойной точностью, если размер мантиссы составляет не менее 64 бит, при этом порядок составляет 15 бит и общий минимальный размер двоичного числа составляет 80 бит;

Для того чтобы использовать более привычное 16-байтное представление чисел с расширенной точностью (128 бит) требуются более сложные схемы процессора или при тех же схемах будет меньшая производительность процессора;

Цель формата расширенной точности - защита промежуточных результатов. Если взять удобный 16-байтный формат, то может случиться, что все вычисления будут производиться в нем и тогда надо расширять точность для промежуточных результатов и так далее.

4.1.3.Диапазоны вещественных чисел в x87.

Внутреннее представление нормализованных чисел:



Единица подразумевается в старшем разряде мантиссы для чисел в формате с одинарной и двойной точностью. Дивпазоны представлеения десятичных чисел в двоичных вещественных форматах:

а) с одинарной точностью:

$-3.37*10^{38}$ / $-1.17*10^{-37}$

$+1.17*10^{-37}$ / $+3.37*10^{38}$

б) с двойной точностью:

$-1.67*10^{308}$ / $-2.23*10^{-307}$

$+2.23*10^{-307}$ / $+1.67*10^{308}$

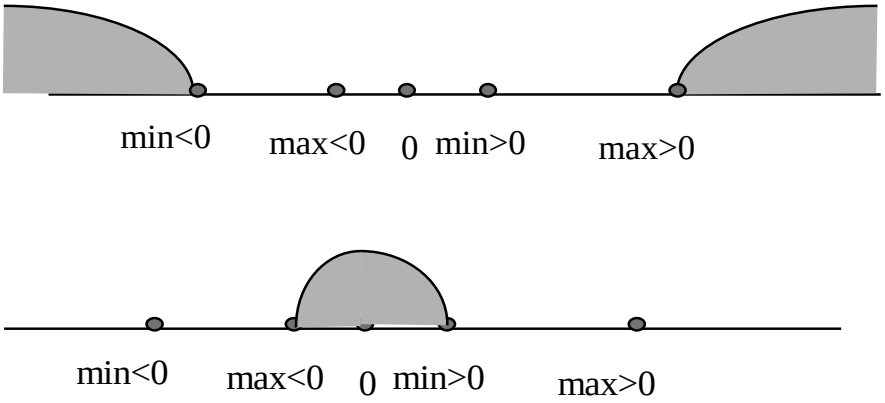
в) с расширенной точностью:

$-1.2*10^{4932}$ / $-3.37*10^{-4931}$

$+3.37*10^{-4931}$ / $+1.2*10^{4932}$

4.2.Особые случаи вещественной арифметики

При реализации операций с вещественными числами возникают внутренние прерывания сопроцессора, называемыми особыми случаями сопроцессора (ОС). Если результат арифметической операции меньше минимального отрицательного числа или больше максимального положительного числа в данном формате, то такое состояние называется переполнением.



Если результат арифметической операции меньше минимального положительного числа и больше максимального отрицательного числа в данном формате, то такое состояние называется антипереполнением.

Сопроцессор генерирует шесть особых случаев вещественной арифметики:

1. Неточный результат. Возникает, когда результат операции невозможно точно представить в формате приемника. Например, при делении 1 на 3 получается бесконечная дробь;

2. Численное антипереполнение. Ненулевой результат слишком мал по абсолютной величине для представления в формате приемника. Например, при делении 1 на максимальное число с расширенной точностью $1.2*10^{4932}$ получается очень маленькое число, которое не

может быть представлено в формате с расширенной точностью. В данном случае результатом является денормализованное число. Денормализованными называются числа с плавающей точкой, имеющие в поле порядка все нули и нулевой старший бит мантиссы.

3. Денормализованный операнд. Возникает, когда операндом команды оказывается денормализованное число.

4. Деление на ноль. Результатом операции является бесконечность.

5. Численное переполнение. Возникает, когда результат слишком велик по абсолютной величине, чтобы быть представленным в формате приемника. Например, при сложении двух максимальных чисел в формате с расширенной точностью.

6. Недействительная операция. Включает в себя все арифметические операции, при которых нет приемлемого результата. Например, при делении нуля на ноль, при извлечении квадратного корня из отрицательного числа.

Обычная обработка особых случаев заключается в том, чтобы немедленно остановить операцию и передать управление процедуре обработки особого случая (внутреннего прерывания процессора).

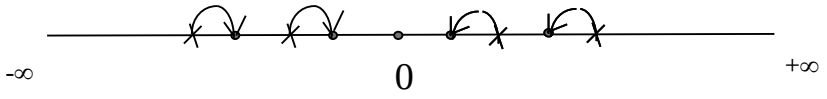
Второй подход состоит в том, чтобы заставить операцию возвратить специальное значение особого случая и продолжить вычисления.

Оба подхода применяются в арифметическом сопроцессоре, программист может сам выбрать режим обработки особых случаев вещественной арифметики. При инициализации сопроцессора по умолчанию устанавливается режим формирования специальных значений. Для формирования специальных значений можно использовать числа с минимальными и максимальными значениями порядков в каждом формате, что не сильно сокращает диапазон значений нормализованных чисел. Кроме того, наборы специальных значений выбраны так, что большинство сравнений на равенство или неравенство выполняется правильно (эти сравнения редки для плавающей арифметики). Сделано это для того, чтобы при получении в качестве промежуточных результатов специальных значений, программа имела возможность сформировать правильный окончательный результат.

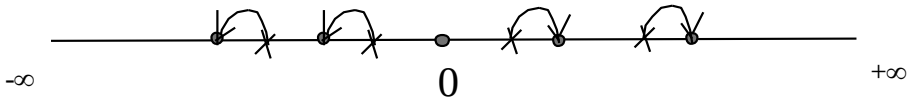
4.3.Формирование специальных значений в особых случаях

4.3.1. Случай неточного результата.

В качестве специального значения используется приближенное значение числа с плавающей точкой. Существуют четыре режима округления:

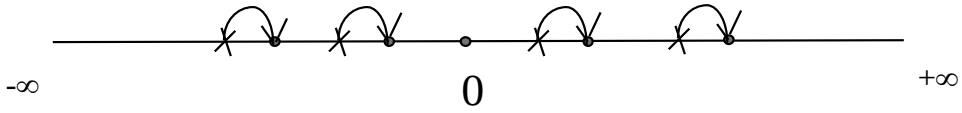


в направлении к 0 (обычно для целочисленной арифметики);

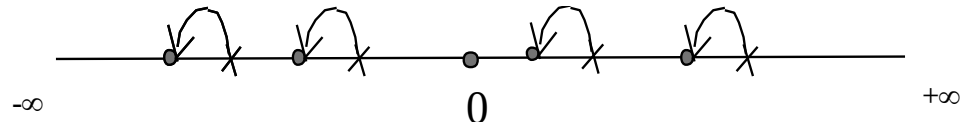


до ближайшего числа (если результат посередине, то выбирается ближайшее четное число).

округление к + ∞



округление к - ∞



Режимы c) и d) используются для реализации интервальной арифметики : операция выполняется дважды: с округлением к +∞ и с округлением к-∞. Истинный результат находится между двумя результатами, следовательно не требуется анализ ошибок округления. При инициализации сопроцессора по умолчанию устанавливается режим округления к ближайшему числу, однако программист может его изменить.

4.3.2.Численное антипереполнение.

Принято соглашение, что для денормализованного числа поле порядка 00..00 считается равным 00...01, но старший бит мантиссы равен 0 (а не 1). Для одинарной точности смещение равно 127, следовательно, поле порядка 00000001 соответствует порядку - 126.

Пример :

нак	Порядок	Старший бит мантиссы	Мантисса – 23 бита	
	0000 0001	1	10...00	$= 1.1_2*2^{-126}$ - норм.
	0000 0001	1	00...00	$=1.0_2*2^{-126}$ - норм.

(	0000 0000	0	10...00	$=0.1_2*2^{-126}$ - денорм.
(	0000 0000	0	01...00	$=0.1_2*2^{-127}$ - денорм.
	....		.....	....
(	0000 0000	0	00...01	$=0.1_2*2^{-149}$ – денорм.

Точность минимального числа равна 1 биту, то есть происходит потеря точности ради расширения диапазона. Специальное значение - денормализованное число, а если результат мал для денормализованого числа, то в качестве результата формируется нуль.

На числовой оси плотность представимых чисел выше у нуля и ниже у максимальных значений.

++ +++++	++- +++-+++-+	++-++-++-+-+- +-+-+-	+- -+- -+- - -+- - -+
0 Денорм.	←——— Нормализованные числа		

4.3.3. Денормализованный операнд.

Для денормализованных операндов выполняются следующие правила формирования результатов операций:

Результат равен 0, если он слишком мал для представления денормализованным числом (например, при умножении двух денормализованных чисел).

Результат - денормализованное число, если он представим в формате денормализованных чисел (например, при сложении двух денормализованных чисел).

Результат - нормализованное число, если неточность денормализованного числа влияет на результат меньше, чем ошибка округления (например, при сложении денормализованного и большого нормализованного числа).

Результат - ненормализованное число, если результат слишком велик для денормализованного числа, неточный, а приемник имеет расширенную точность (например, при сложении денормализованного и большого нормализованного числа).

Ненормализованным числом называется число с расширенной точностью, имеющее обычное (ненулевое и неединичное) поле порядка и нулевой старший бит мантиссы.

Ненормализованные операнды в операциях подчиняются правилам для денормализованных чисел. Особый вид ненормализованного числа - псевдоноль. Псевдоноль имеет ненулевое поле порядка и нулевую мантиссу. В сравнениях псевдонули считаются нулями. Данный формат чисел поддерживается сопроцессором модели 80287 и не поддерживается моделью 80387 и устройством с плавающей точкой процессора 80486 и последующих моделей, так как уточнился стандарт представления чисел с плавающей точкой.

Результат - особый случай недействительной операции (наример, при делении на денормализированное число или попытке извлечения квадратного корня из денормализованного числа).

4.3.4. Деление на ноль.

Бесконечность - число, имеющее в поле порядка все единицы, а в поле мантиссы - все нули . Единица в старшем разряде хранится явно в расширенном формате. Так как поле знака может быть равным единице или нулю, то существуют положительная и отрицательная бесконечности.

Если x - конечное положительное число, не равное нулю, то разрешены следующие операции с бесконечностью:

1) $x : +0 = +\infty$

$-x : +0 = -\infty$

$x : -0 = -\infty$

$-x : -0 = +\infty$

2) $x : +\infty = +0$

$-x : +\infty = -0$

$x : -\infty = -0$

$-x : -\infty = +0$

3) $x * +\infty = +\infty$

$-x * +\infty = -\infty$

$x * -\infty = -\infty$

$x * -\infty = +\infty$

4) $+\infty : x = +\infty$

$+\infty : -x = -\infty$

$\infty : x = -\infty$

$\infty : -x = +\infty$

5) $+\infty * +\infty = +\infty$

$+\infty * -\infty = -\infty$

$-\infty * +\infty = -\infty$

$-\infty * -\infty = -\infty$

Запрещены следующие операции с бесконечностью (в качестве результата они дают особый случай недействительной операции):

$$\begin{cases} +\infty * +0 \\ -\infty * +0 \\ +\infty * -0 \\ -\infty * -0 \end{cases}$$

$$\begin{cases} +\infty : +0 \\ -\infty : +0 \\ +\infty : -0 \\ -\infty : -0 \end{cases}$$

$$\begin{cases} +\infty : +\infty \\ -\infty : +\infty \\ +\infty : -\infty \\ -\infty : -\infty \end{cases}$$

Для операций сложения, вычитания и сравнения бесконечностей появляются трудности. Математический сопроцессор 80287 имеет два режима управления бесконечностью : проективный и аффинный.

В проективном режиме - скрыт факт наличия двух бесконечностей и двух нулей. Сравнение бесконечности с конечным числом в качестве результата вызывает особый случай недействительной операции, сравнение двух бесконечностей дает результат “равны”. Сложение и вычитание бесконечностей дает в результате особый случай недействительной операции. Сложение и вычитание бесконечности и конечного числа дает в результате бесконечность.

В аффинном режиме существуют положительная и отрицательная бесконечности и существуют положительный и отрицательный нули. Этот режим разрешает сложение бесконечностей с одинаковыми знаками и вычитание бесконечностей с разными знаками:

$(+\infty) + (+\infty) = +\infty$;

$(-\infty) - (+\infty) = -\infty$.

В аффинном режиме также разрешены сложение и вычитание бесконечности и конечного числа, сравнение конечного числа с бесконечностью и сравнение бесконечностей друг с другом.

Проективный режим упразднен, начиная с математического сопроцессора 80387, у него и последующих версий устройства с плавающей точкой существует только аффинный режим управления бесконечностью.

4.3.5.Численное переполнение.

Численное переполнение наступает тогда, когда результат превышает наибольшее конечное число в формате приемника. Переполнение опасно, так как знак может оказаться неверным, а результат будет не бесконечность, а слишком большое для представления число.

Пример: деление большого числа на малое число (близкое к 0). Знак результата будет зависеть от знака малого числа, хотя он может быть неверным в результате ошибки округления.

Специальное значение формируется в зависимости от режима округления, установленного в регистре управления сопроцессора:

1) если установлен режим округления к ближайшему числу - то результат- плюс или минус бесконечность;

2) если установлен режим округления к минус бесконечности, то для положительных чисел результат - максимальное число больше нуля, для отрицательных чисел - минус бесконечность;

3) если установлен режим округления к плюс бесконечности, то для положительных чисел результат – плюс бесконечность, а для отрицательных чисел - максимальное число меньше нуля;

4) если установлен режим округления к нулю, то для положительных чисел результат - максимальное число больше нуля, для отрицательных чисел - максимальное число меньше нуля.

4.3.6.Недействительная операция.

Возникает, когда нет приемлемого результата в арифметической операции, операции сравнения или при стековых операциях (при этом флаг SF в регистре состояния равен единице):

при реализации операций сложения, вычитания, умножения и деления может возникнуть особый случай недействительной операции (например, при запрещенных операциях с бесконечностью);

- при реализации операции сравнения результатом является код условия. Обычно он показывает отношения “<”, “=”, “>”. Если при этом возникает особый случай недействительной операции (например, в проективном режиме при сравнении бесконечности и конечного числа, или при сравнении нечисла и конечного числа), то код условия принимает специальное значение “не сравнимы”. При этом нарушается закон трихотомии арифметики : $\forall a,b => a > b \vee a = b \vee a < b$. Это вызывает трудности в языках высокого уровня при исполнении оператора IF, так как, например, отрицанием условия больше или равно является не только меньше, но и “не сравнимы”;.

особый случай стека: переполнение, то есть попытка включения числа в заполненный стек или антипереполнение, то есть попытка извлечения числа из пустого стека;

Во всех случаях результатом недействительной операции будет нечисло (NAN).

Нечисло – это двоичное вещественное число, у которого в поле порядка все единицы, а в поле мантиссы - все, кроме значения, зарезервированного за бесконечностью, то есть 1.000000...0.

Если один или оба операнда в операции -нечисло - то результатом тоже является нечисло. Таким образом, полученное в промежуточном результате операции нечисло распространяется в вычислениях и выдается как окончательный результат.

Особый случай нечисла - неопределенность. Сопроцессор формирует неопределенность как результат недействительной операции (остальные нечисла формируются программистом).

Для вещественных чисел неопределенность - это нечисло у которого в поле мантиссы две старшие цифры – единицы, а все остальные – нули (1.1...00), а поле порядка одни единицы (11...11). Поле знака также равно единице.

Для формата десятичных чисел неопределенность имеет следующее представление :

Знак	1	2	3	4	мусор
79	7	7	6	63 -	0 - 62
	5 - 78	1 - 74	7 - 70	66	
1	1	1	1	1111	xx...x
	111	111	111	x	

Десятичную неопределенность нельзя преобразовать в вещественное число, результат при этом - неопределенность.

Для двоичных целых форматов неопределенность представляется максимальным по модулю отрицательным числом, то есть имеет код

1 0000...00

3

нак

Целочисленная неопределенность при преобразовании в формат с плавающей точкой дает обычное отрицательное число.

Попытка преобразования нечисла, денормализованного, ненормализованного числа или бесконечности в формат целых чисел вызывает особый случай недействительной операции, результатом которой и является неопределенность.

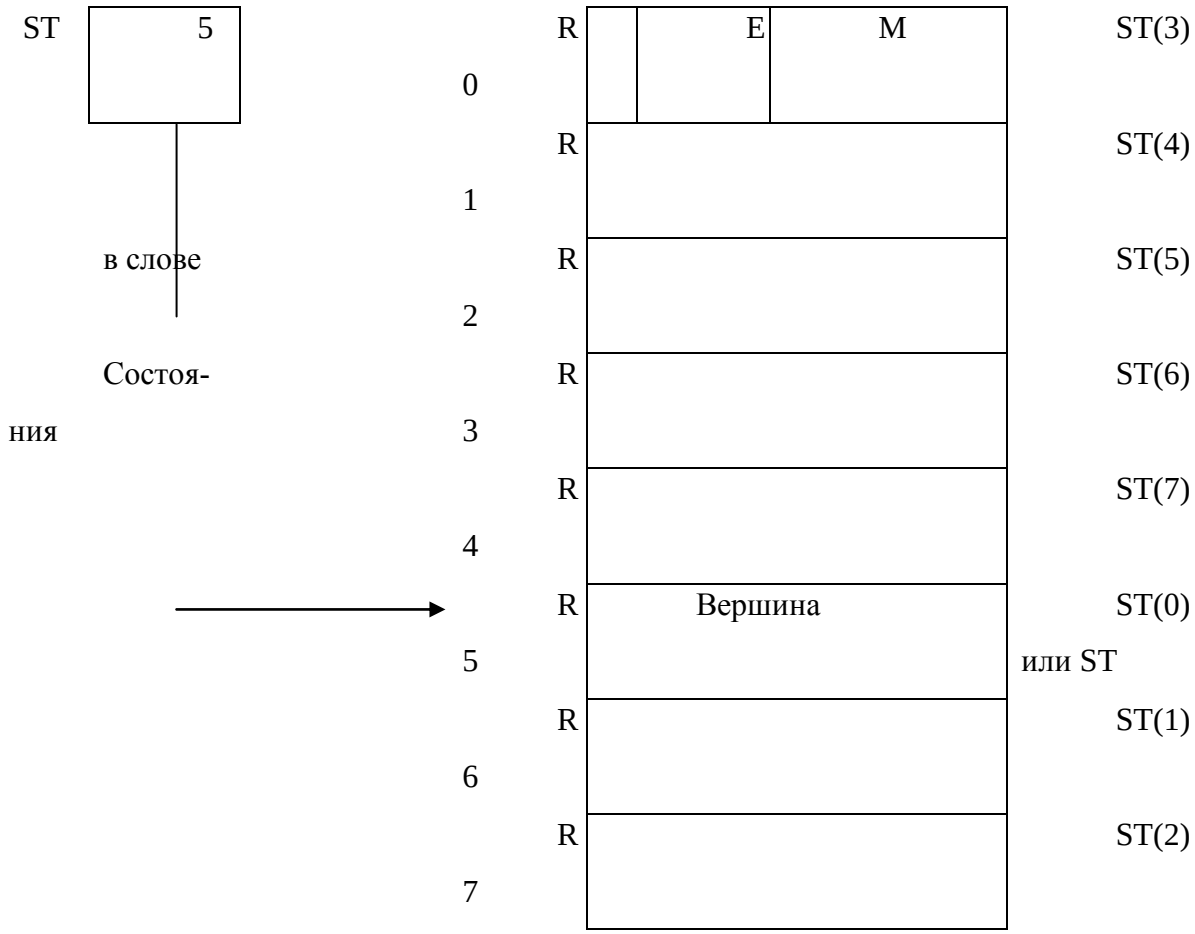
4.4.Регистры математического сопроцессора.

4.4.1.Численные регистры (регистровый стек).

Предназначен для выполнения вычислительных операций. Регистровый стек состоит из восьми регистров по 80 бит каждый, в которых представлены вещественные числа в расширенном формате. Адресация численных регистров реализована следующим образом: логический номер регистра, указанного в команде, процессор суммирует с содержимым поля ST (вершина стека) в регистре состояний. Сумма по модулю 8 определяет используемый численный регистр. Для удобства математических расчетов код и результат операции могут замещать значения операндов и использоваться дальше в вычислениях.

1 7 63

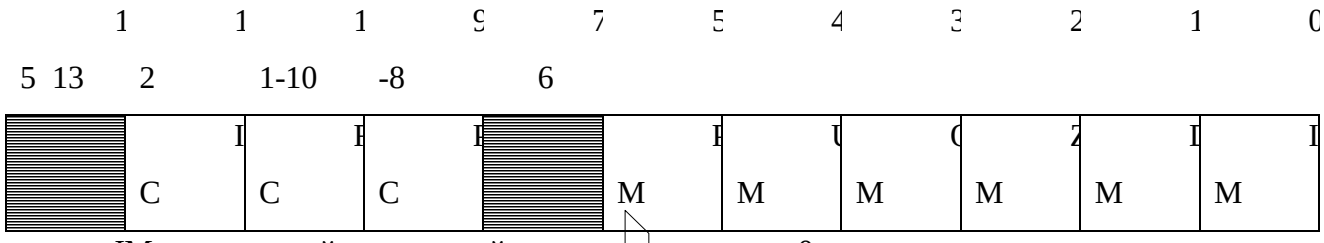
01 9 8



Числовой стек имеет кольцевую организацию. Если стек заполнен полностью, то есть ST = 111, то операция включения в стек вызывает перезапись в вершину стека ST = 000 (при обработке особых случаев сопроцессора посредством обращения к системе пррываний центрального процессора) или особый случай недействительной операции (при формировании специальных значений в особых случаях сопроцессора).

4.4.2.Регистр управления (cw)

Предназначен для управления работой сопроцессора. Имеет размер – 16 бит.



IM -маска недействительной операции

DM -маска денормализованного операнда

ZM -маска деления на нуль

OM -маска переполнения

UM -маска антипереполнения

= 0 - прерывания разрешены

= 1 – формирование специальных значений

PM -маска неточного результата

PC - поле управления точностью:

11 - расширенная точность (по умолчанию);

10 - двойная точность;

00 - одинарная точность.

RC - поле управления округлением:

00 - к ближайшему числу (по умолчанию);

01 - округление к минус бесконечности;

10 - округление к плюс бесконечности;

11 - округление к нулю.

IC - поле управления бесконечностью: (для 80287)

0 - проективный режим (по умолчанию);

1 - аффинный режим.

С 80387 не используется - по умолчанию всегда аффинный режим.

4.4.3.Регистр состояния.

Флаги (0-6 биты):

IE - недействительная операция;

DE - денормализованный операнд;

ZE - деление на нуль;

OE – переполнение;

UE – антипереполнение;

PE - точность (неточный результат);

SF- флаг стека (с 80387 сопроцессора).

При возникновении особого случая, не зависимо от того, замаскирован он или нет, автоматически выставляется флаг особого случая, равный единице. Явно сбросить флаги должен программист, загружая в регистр состояния новое значение соответствующих флагов.

ES - бит суммарной ошибки, устанавливается в единицу, когда команда порождает любой особый случай. C_0, C_1, C_2, C_3 - коды условий, являющиеся результатом сравнения или команды нахождения остатка. Интерпретация кодов условий зависит от конкретной команды.

ST - поле вершины стека, содержит физический номер регистра, являющийся вершиной стека.

B - бит занятости, равен единице, когда процессор выполняет команду или сигнализирует прерывание, если сопроцессор свободен, то B равен нулю. Бит занятости показывает занятость численного операционного устройства сопроцессора, в которое входит числовой стек и регистр тэгов.

4.4.4. Регистр тэгов (признаков).

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Тэ гR7	Тэ гR6	Тэ гR5	Тэ гR4	Тэ гR3	Тэ гR2	Тэ гR1	Тэ гR0

Каждое поле соответствует состоянию численного регистра (соответствие устанавливается на уровне физических имен регистров - RO:R7):

00 – в регистре действительное число (любое конечное число , не равное нулю, ненормализованное число считается действительным в 80287);

01 – нуль;

10 - недействительное число (нечисло, бесконечность, денормализованное число или ненормализованное число в 80387 и последующих версиях устройства с плавающей точкой);

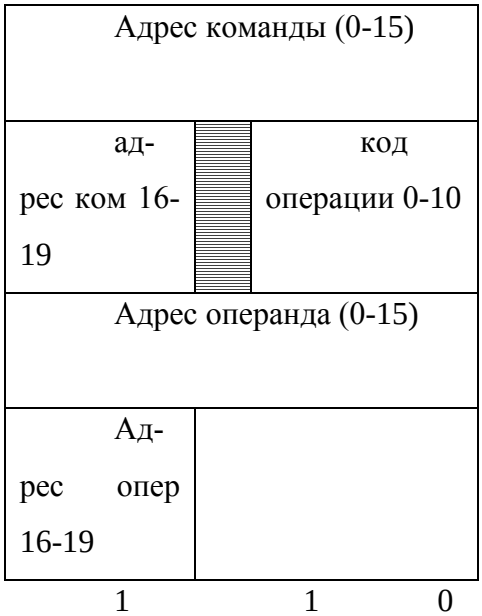
11 - пустой регистр.

Регистр считается пустым, если он не был инициализирован, или его содержимое было “извлечено” из стека. Процессор использует это состояние тэга для обнаружения переполнения стека (слишком много включений) или антипереполнения стека (слишком много извлечений). При попытке извлечь число из пустого регистра возникает особый случай недействительной операции. Когда этот особый случай замаскирован (прерывания запрещены), процессор корректирует ST обычным способом и возвращает неопределенность как результат операции.

4.4.5.Указатели особого случая.

Предназначены для процедур обработки особых случаев. Они имеют два формата в зависимости от работы 80287 в реальном или защищенном режиме.

Реальный режим



Защищенный режим

20-разрядные адреса команды и операнда и 11 младших разрядов кода операции	Селектор плюс смещение подозрительной команды и ее операнда

4.5.Система команд арифметического сопроцессора.

4.5.1.Команды передачи данных.

ТИП КОМАНДЫ	МНЕМОНИКА И ФОРМАТ	ОПЕРАНД/ДЕЙСТВИЕ
КОМАНДЫ ЗАГРУЗКИ В СТЕК	FLD REG/MEM	Вещественное число
	FILD MEM	Целое число
	FBLD MEM	Десятичное число
КОМАНДЫ ИЗВЛЕЧЕНИЯ ИЗ СТЕКА	FSTP REG/MEM	Вещественное число
	FISTP MEM	Целое число
	FBSTP MEM	Десятичное число
КОМАНДЫ КОПИРОВАНИЯ	FST REG/MEM	Вещественное число
	FIST MEM	Целое число
КОМАНДА ОБМЕНА	FXGH REG	Обмен содержимым между регистрами стека
КОМАНДЫ ЗАГРУЗКИ КОНСТАНТ	FLDZ	Загрузка 0
	FLD1	Загрузка 1
	FLDPI	Загрузка π
	FLDLG2	Загрузка log по основанию 10 из 2
	FLDLN2	Загрузка ln 2
	FLDL2T	Загрузка log по основанию 2 из 10
	FLDL2E	Загрузка log по основанию 2 из E

КОМАНДЫ ЗАГРУЗКИ – все команды одноперандные, записывают в вершину стека заданный операнд. При загрузке операнд преобразуется в формат с заданной точностью, значение поля ST в регистре состояния уменьшается на единицу и операнд записывается в новую вершину стека. Преобразование формата осуществляется в соответствии с полем PC регистра управления.

КОМАНДЫ ИЗВЛЕЧЕНИЯ - все команды одноперандные, извлекают содержимое вершины стека и запоминают его в операнд-приемник. При этом содержимое вершины стека преобразуется в формат приемника, значение поля ST в регистре состояния увеличивается на единицу. Преобразование формата для двоичных чисел выполняется в соответствии с полем RC регистра управления, для десятичных чисел - путем суммирования числа и 0.5 и отбрасывания дробной части результата.

КОМАНДЫ КОПИРОВАНИЯ - все команды одноперандные, копируют содержимое вершины стека в операнд-приемник. При этом содержимое вершины стека преобразуется в формат приемника аналогично командам извлечения. Поле вершины стека данными командами не изменяется.

КОМАНДА ОБМЕНА - одноперандная команда, предназначена для обмена содержимого вершины стека и другого численного регистра. Поле вершины стека не изменяется.

КОМАНДЫ ЗАГРУЗКИ КОНСТАНТ - безоперандные команды, предназначены для быстрой загрузки констант в вершину стека. Поле вершины стека уменьшается на единицу.

Коды операций всех команд начинаются с бит 11011, которые соответствуют коду команды ESC в основном процессоре (заставляет извлечь содержимое указанного в ней операнда и передать его на шину данных). Ассемблерная мнемоника команд сопроцессора начинается с буквы F.

4.5.2.Арифметические команды

Для безоперандных арифметических команд операндами по умолчанию являются регистры ST(0), ST(1). Для одноперандных арифметических команд - один из операндов - это вершина стека ST(0).

Для всех арифметических команд существуют обычные (прямые) формы команд: FADD, FSUB, FMUL, FDIV , а для команд вычитания и деления существуют ещё и обратные формы команд: FSUBR , FDIVR.

Для обратных форм команд выполняется следующая схема:

источник - приемник=приемник или источник / приемник = приемник. Для прямых команд уменьшаемое или делимое берется из приемника.

Существует шесть форм команд:

FXXX;

FXXX память;

FIXXX память;

FXXX ST(0), ST(i);

FXXX ST(i), ST(0);

FXXXP ST(i), ST(0).

Здесь XXX- мнемокод арифметической команды.

При реализации команд первого типа процессор извлекает из стека два верхних операнда, а затем включает в стек результат операции. В результате указатель вершины стека увеличивается на единицу.

При реализации команд второго типа операнд в памяти (одинарной или двойной точности) является источником, а регистр ST(0)- приемником. Результат записывается в вершину стека , то есть указатель вершины стека не изменяется.

Реализации команд третьего типа присходит аналогично командам второго типа, но операнды в памяти целые (16 или 32 бита).

При реализации команд четвертого типа любой регистр ST(i) служит источником, а вершина стека – приемником, при этом указатель вершины стека не изменяется.

При реализации команд пятого типа любой регистр ST(i) служит приемником, а вершина стека - источником, при этом указатель вершигы стека не изменяется.

При реализации команд шестого типа любой регистр ST(i) служит приемником, а вершина стека - источником, по окончании операции источник извлекается из стека, то есть указатель вершины стека увеличивается на единицу.

Арифметические операции над целыми числами выполняется медленнее, чем в основном процессоре.

4.5.3.Дополнительные арифметические команды

МНЕМОКОД	ОПИСАНИЕ
FSQRT	Извлечение квадратного корня
FSCALE	Масштабирование на степень 2
FPREM	Нахождение частичного остатка
FPREM1	Нахождение остатка
FRNDINT	Округление до целого
FXTRACT	Выделение порядка и мантиссы

FABS	Нахождение абсолютной величины
FCHS	Изменение знака

FSQRT: $\sqrt{ST(0)} \rightarrow ST(0)$ - извлекает квадратный корень из ST(0) и записывает результат в ST(0). При попытке извлечения корня из отрицательного числа возникает особый случай недействительной операции. Начиная с процессора 80486, разрешены операции : $\sqrt{0} = 0, \quad \sqrt{\infty} = \infty$.

FSCALE: $ST(0) * 2^{ST(1)} \rightarrow ST(0)$, где $2^{-15} < ST(1) < 2^{15}$. Если ST(1) не целое и больше 1 по абсолютной величине, то берется ближайшее меньшее целое, если ST(1) меньше 1 или находится вне диапазона, то результат неопределенный, особый случай не генерируется. Начиная с сопроцессора 80387 ограничений на операнды команды нет.

FXTRACT: разлагает ST(0) на два числа с плавающей точкой: несмещенный порядок и знаковую мантиссу. Порядок заменяет старое значение ST(0), а мантисса включается сверху, то есть указатель стека уменьшается на единицу. Порядок истинный, в форме вещественного числа.

FRNDINT: округляет ST(0) до целого числа в формате с плавающей точкой согласно полю RC регистра состояния.

FABS: заменяет содержимое регистра ST(0) на его абсолютное значение (изменяет знаковый разряд).

FCHS: инвертирует знак содержимого ST(0).

FPREM: вычисляет остаток от деления содержимого ST(0) на число из ST(1), знак остатка совпадает со знаком ST(0). Остаток заменяет содержимое ST(0). FPREM формирует частичный остаток, так как операция может прекратиться, не завершив операции деления. FPREM работает, производя повторяющиеся машинные вычитания, при этом производится не более 64 вычитаний. Если 64 вычитания достаточно, чтобы число ST(0) < ST(1), то в ST(0) будет находится истинный остаток, а бит C_2 кода условия сбрасывается в нуль. Если 64 вычитаний недостаточно, то C_2 устанавливается в единицу, а в ST(0) находится уменьшенный результат после 64 вычитаний. C_2 проверяется командой JP, после передачи регистра состояния в AX и командой SAHF. Повторное вычитание FPREM до образования $C_2 = 0$ дает точный остаток. 64 вычитания делают для того, чтобы между повторными исполнениями команды можно было прервать сопроцессор (64 вычитания гарантируют, что FPREM не будет длиться дольше команды деления, то есть не превысит длительности максимальной по времени команды). Применяется в основном в вычислениях тригонометрических функций.

FPREM1 (введена в систему команд для сопроцессора 80387): вычисляет величину REM = ST - ST(1)*Q, где Q – целая часть числа ST/ST(1); C_0,C_3,C_2 – содержат младшие три бита частного.

4.5.4.Команды сравнений

МНЕМОНИКА	ОПИСАНИЕ
FCOM операнд/(без операндов)	Сравнение
FICOM операнд	Целочисленное сравнение
FCOMP операнд/(без операндов)	Сравнение с извлечением из списка
FICOMP операнд	Целочисленное сравнение с извлечением
FCOMPP (без операндов)	Сравнение и двойное извлечение
FTST (без операндов)	Сравнение с нулем
FXAM (без операндов)	Анализ
FUCOM (без операндов)	Сравнение мантисс операндов
FUCOMP (без операндов)	Сравнение мантисс операндов с извлечением
FUCOMPP (без операндов)	Сравнение мантисс операндов с двойным извлечением

Во всех операциях сравнения всегда существует ST(0), поэтому требуется один операнд, если нет операндов, то второй операнд берется из ST(1).

FCOM X сравнивает содержимое ST(0) с операндом X и устанавливает следующие коды условия :

Условие сравнения	3	2	0
ST(0) > X			
ST(0) < X			
ST(0) = X			
ST(0) и X “не сравнимы”			

X- численный регистр, или число в памяти одинарной или двойной точности. Указатель вершины стека - не изменяется.

FCOM - без операндов - это FCOM ST(1).

FCOMP - это FCOMP ST(1) и извлекается из стека содержимое регистра ST(0), то есть указатель вершины стека уменьшается на единицу.

FCOMPP – это FCOMPP ST(1) и извлекается из стека содержимое регистров ST(0) и ST(1), то есть указатель вершины стека уменьшается на два.

FICOM - аналогична FCOM, но операндом является целое число (16 или 32 бита) в памяти.

FCOMP - аналогична FCOM, но после операции сравнения производится одно извлечение из стека, то есть указатель вершины стека уменьшается на единицу.

FICOMP - аналогична FCOMP для целых чисел

FCOMPP (без операндов) действует как FCOM (без операндов), но после сравнения она извлекает из стека оба операнда, то есть указатель вершины стека уменьшается на два.

FTST – сравнивает нулем и устанавливает следующие коды условия:

Условие сравнения	3	2	0
ST(0) > 0			
ST(0) < 0			
ST(0) = 0			
ST(0) и 0 “не сравнимы”			

FXAM - аналогична FTST, но установка кодов условий другая:

$C_1 = 0$	$C_1 = 1$
ST(0) >0	ST(0) <0

$C_0 = 0$	$C_0 = 1$
ST(0) конечное	ST(0) не конечное

3 2 3 2

		ненормализованное
		нормализованное
		0
		денормализованное

		нечисло
		бесконеч-ность
		пустой ре-гистр
		пустой ре-гистр

FXAM выполняется примерно в два раза быстрее чем FTST.

Команды сравнения FPU 80486.

FUCOM, FUCOMP, FUCOMPP - сравнивают мантиссы. Допускается сравнение нечисел.

Бит $C_2 = 1$, если один или оба операнда - нечисла, а биты C_0 и C_3 устанавливает соотношение между мантиссами.

FUCOM
FUCOMP } - безоперандные- сравнивают ST(0) и ST(1).
FUCOMPP

Коды условий после команд сравнения мантисс:

3	2	0	
			ST(0) > 0
			ST(0) < 0
			ST(0) = 0
			?

Как используются коды условий сопроцессора для условных переходов основного процессора?

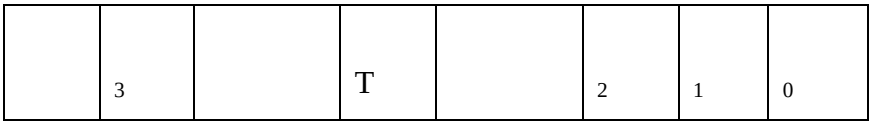
Ассемблерная команда сопроцессора FSTSW AX копирует содержимое регистра состояния сопроцессора в регистр AX основного процессора.

Затем коды условий проверяются в среде основного процессора.

Команда SAHF (запомнить AH во флагах) позволяет получить следующий результат:

SAHF

5



FST W AX

Результат FCOM по установке флажков такой же, как и у команды CMP основного процессора за исключением условия “не сравнимы”.

FCOM	Переход к MET, если
FSTSW AX	ST(0) = ST(1)
FWAIT, если нет FPU	Команды JE, JNE, JA, JAE, JB, JBE,
SAHF	JP, JNP, JC, JNC, JZ, JNZ – можно ис-
JE MET	пользовать для условных переходов

4.5.5.Трансцендентные команды

МНЕМОНИКА	ОПИСАНИЕ
FPTAN	Частичный tg
FPATAN	Частичный arctg
FYL2X	$y*\log_2(x)$
FYL2XP1	$y*\log_2(x)$
F2XM1	2^x-1

FPTAN дает в качестве результата два таких числа x и y, что y/x=tg (ST(0)). У - заменяет старое содержимое ST(0), а x включается сверху.

Для модели математического сопроцессора 80287 аргумент команды FPTAN должен быть нормализован; денормализованные и ненормализованные числа, бесконечность и нечисла недопустимы в качестве аргументов у всех трансцендентных команд. Кроме того, аргумент должен находиться в диапазоне $0<ST(0)<\pi/4$. Если аргумент недопустим или находится вне диапазона, FPTAN дает неправильный результат, не сигнализируя об особом слу-

чае (то же самое верно и для других трансцендентных команд). Программист сам должен заботится о допустимости аргумента и приведении его в диапазон. Допустимость может быть проверена с помощью команды FXAM. Для модели математического сопроцессора 80387 и устройств FPU команда FPTAN в ST(1) возвращает тангенс исходного угла, а в ST(0) возвращается единица. Сделано так из-за совместимости с сопроцессором 80287.

FPATAN - вычисляет ST(0)=arctg (ST(1)/ST(0)). Два верхних элемента извлекаются из стека, результат включается в стек. Упрощает вычисление остальных тригонометрических функций. Для аргументов должно выполняться следующее условие: $0<ST(1)<ST(0)<\infty$ для модели математического сопроцессора 80287, с модели математического сопроцессора 80387 ограничений на аргумент нет.

FYL2X - вычисляет функцию $ST(0) = ST(1)*\log_2 ST(0)$. Два операнда извлекаются из стека, а затем результат включается в стек. Условие для аргументов $ST(0) > 0$ должно быть выполнено.

FYL2XP1 - вычисляет функцию $ST(0) = ST(1)*\log_2(ST(0)+1)$. Должно быть выполнено следующее условие: $0 \le |ST(0)| < 1 - \frac{1}{\sqrt{2}}$. Причина появления этой команды - более высокая точность вычисления функции $\log_2(1+x)$ при малых x. Удобна для вычисления обратных гиперболических функций.

F2XM1 вычисляет $ST(0) = 2^{ST(0)}-1$, причем ST(0) должно находиться в диапазоне: $0 < ST(0) < 0.5$ модели математического сопроцессора 80287. Для модели математического сопроцессора 80387 и устройств FPU условие для аргумента функции $-1 < ST(0) < +1$. Вычисление 2^x-1 вместо 2^x позволяет избежать потери точности, при x близких к нулю (при этом 2^x близко к 1). Удобна для вычисления гиперболических функций.

Новые трансцендентные команды

Для модели математического сопроцессора 80387 были разработаны новые трансцендентные команды:

FSIN - без операндов. Вычисляет синус действительного числа из вершины стека, результат записывается в вершину стека. Для величины угла нет ограничений.

FSINCOS - без операндов. Одновременно вычисляет синус и косинус угла в вершине стека. Синус - в ST(1), косинус- в ST(0). Ограничений величины угла нет.

FCOS - без операндов. Вычисляет косинус действительного числа из вершины стека. Результат помещается в новую вершину стека. Ограничений величины угла нет.

4.5.6.Административные команды

Эта группа команд обеспечивает управление режимом работы сопроцессора. Для программиста особенно интересны команды работы с регистрами управления и состояния.

МНЕМОНИКА	ОПИСАНИЕ КОМАНДЫ
FNSTCW(FSTCW)	Запомнить регистр управления
FLDCW	Загрузить регистр управления
FNSTSW(FSTSW)	Запомнить регистр состояния
FNSTSW AX(FSTSW AX)	Запомнить регистр состояния в AX
FNCLEX(FCLEX)	Сбросить особые случаи
FNINIT(FINIT)	Инициализировать сопроцессор
FNSTENV(FSTENV)	Запомнить среду
FLDENV	Загрузить среду
FNSAVE(FSAVE)	Запомнить полное состояние
FRSTOR	Восстановить полное состояние
FINCSTP	INC указателя стека
FDECSTP	DEC указателя стека
FFREE	Освободить регистр
FNOP	Нет операции FWAIT - ожидание
FSETPM	Установить защищенный режим
FENI(FNENI)	Разрешить прерывания (IEM = 0)
FDISI(FNDISI)	Запретить прерывания (IEM = 1)

FNSTCW (FSTCW) - содержимое регистра управления записывается в ячейку памяти, указанную в качестве операнда.

FLDCW - загружает регистр управления из ячейки памяти, указанной в качестве операнда.

Эти команды применяются для изменения режима работы сопроцессора (можно переопределять режим округления, управления бесконечностью, маскирование особых случаев).

FNSTSW (FSTSW) - передает содержимое регистра состояния в ячейку памяти, указанную в качестве операнда. FNSTSW AX (FSTSW AX) передает содержимое регистра состояния в регистр AX основного процессора.

FNCLEX (FCLEX) - сбрасывает (устанавливает в нуль) флажки всех особых случаев и биты ES и B в регистре состояния.

FNINIT (FINIT) - инициализирует регистры : управления, состояния и тэгов (такое же действие производит аппаратный сигнал сброса):

Регистр управления :

режим управления бесконечностью – для модели математического сопроцессора 80287 – проективный, для последующих моделей - аффинный;

режим округления - округление к ближайшему числу;

режим точности - расширенная точность;

все особые случаи - замаскированы (то есть формируются специальные значения).

Регистр состояния :

a) B = 0 (не занят)

b) C_0 C_1 C_2 C_3 - не определены

c) ST = 0

d) ES = 0

e) флаги особых случаев = 0

Все тэги = 11 (“пустые”)

FNSTENV (FSTENV) - записывает в память содержимое регистров управления, состояния, тэгов и указателей особого случая, начиная с адреса указателя в операнде. Формат информации в памяти следующий :

15 бит 0 бит

Регистр управления	0 (смещение от начала)
Регистр состояния	2
Регистр тэгов	4
Указатель команды	6
Указатель операнда	10 старший адрес

FLDENV - загружает среду, ранее запомненную FNSTENV. Эти команды применяются в процедурах обработки особых случаев, чтобы получить доступ к указателям особого случая.

FNSAVE (FSAVE) действует аналогично FNSTENV, но дополнительно сохраняет в памяти содержимое численных регистров, располагая их после указателя операнда :

Смещение от начала буфера

S T(0)	4

Всего в память передается 94 байта (или 108, если 387 процессор
После выполнения команды сопроцессор переходит в нач. сост. как после FINIT

ST(1)	4
ST(2)	4
ST(3)	4
ST(4)	4
ST(5)	4
ST(6)	4
ST(7)	4

FRSTOR восстанавливает все регистры сопроцессора из памяти. Эта команды в основном применяется для переключения задач (в защищенном режиме). Обратная ей – команда FSAVE.

FINCSTP и FDECSTP - осуществляют инкремент или декремент указателя стека. Не влияют на регистр тэгов и численные регистры, не эквивалентны командам извлечения или включения в стек.

FFREE ST(i) - устанавливают тэг регистра ST(i) в состояние “пустой” (11), но содержание численного регистра не изменяют.

FNOP - не производит никакой операции, действует как FST ST(0), ST(0).

FENI и FDISI- не используются с сопроцессора 80387.

FSETPM - переводит сопроцессор в защищенный режим.

Аппаратный сброс переводит основной процессор и сопроцессор в реальный режим. Команды с мнемоникой FNXXX не проверяют незамаскированные особые случаи (если прерывания разрешены) и выполняется немедленно. В таком случае ассемблер автоматически не вставляет перед ней команду FWAIT, то есть не проверяет наличия особого случая сопроцессора. Все остальные команды синхронизируются автоматически.

4.6.Совместная работа двух процессоров в системе.

Выборку команд из общей очереди команд осуществляет центральный процессор. Если выбранная команда оказывается командой центрального процессора, он выполняет ее обычным образом, сопроцессор такие команды игнорирует. Если выбирается команда сопроцес-

сора, действия центрального процессора зависят от специфики конкретной команды. Если команда не связана с обращением к памяти, центральный процессор ее игнорирует и переходит к следующей команде. Если команда требует обращения к памяти, центральный процессор вычисляет физический адрес операнда и обращается к памяти, при этом сопроцессор перехватывает с общей шины адреса физический адрес операнда, а в операции со считыванием из памяти - еще и данные с общей шины данных. После этого сопроцессор выполняет команду. Выполнение команды сопроцессором происходит параллельно с работой центрального процессора, что повышает эффективность системы. Существуют случаи, когда требуется синхронизация действий двух процессоров :

4.6.1.Синхронизация по командам.

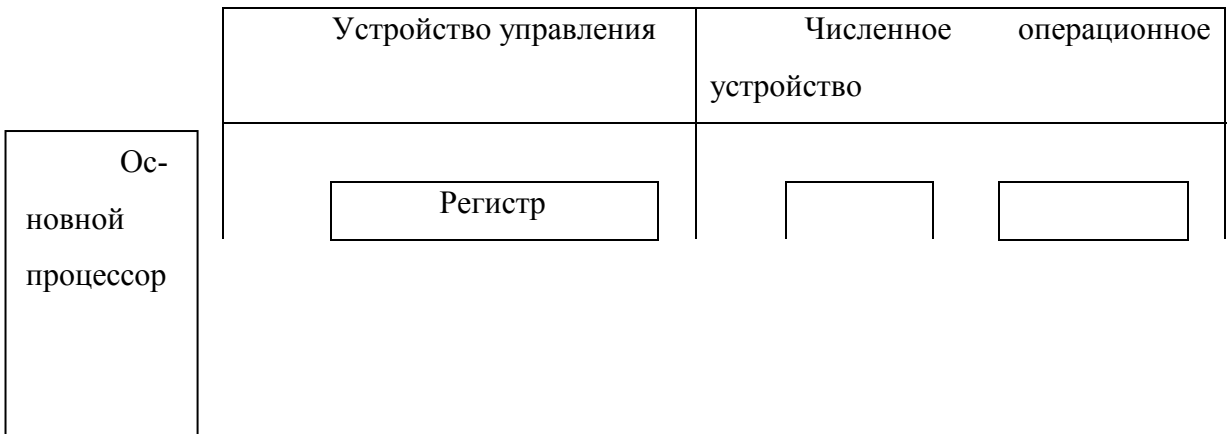
Если сопроцессор занят выполнением команды, а центральный процессор выбирает для выполнения следующую команду - опять команду сопроцессора, то центральный процессор не должен ее передавать сопроцессору, а должен подождать. Следовательно, перед каждой командой сопроцессора в программе должна быть специальная команда центрального процессора, которая проверяет занятость сопроцессора и переводит его в состояние ожидания, если надо. Такой командой является команда FWAIT, а вставляет ее ассемблер (или компилятор языка высокого уровня).

Команда FWAIT не вставляется перед административными командами в мнемокоде которых есть буква N, FNXXX... Это так называемые команды без ожидания.

Происходит это по следующей схеме (см. рисунок 1). Некоторые административные команды могут выполняться устройством управления, а не численным операционным устройством. Поэтому перед такими командами не надо проверять бит B в регистре состояния, который показывает занятость численного операционного устройства.

Данными командами следует пользоваться аккуратно, так как. они не вызывают обработчика особых случаев через систему прерываний, то есть не проверяют наличие сигнала прерывания от сопроцессора, а если он произошел, то процесс восстановления из прерывания не произойдет, так как. выполнение команды управления может помешать процедурам обработки определить причину особого случая и метод его обработки.

Структурная схема сопроцессора



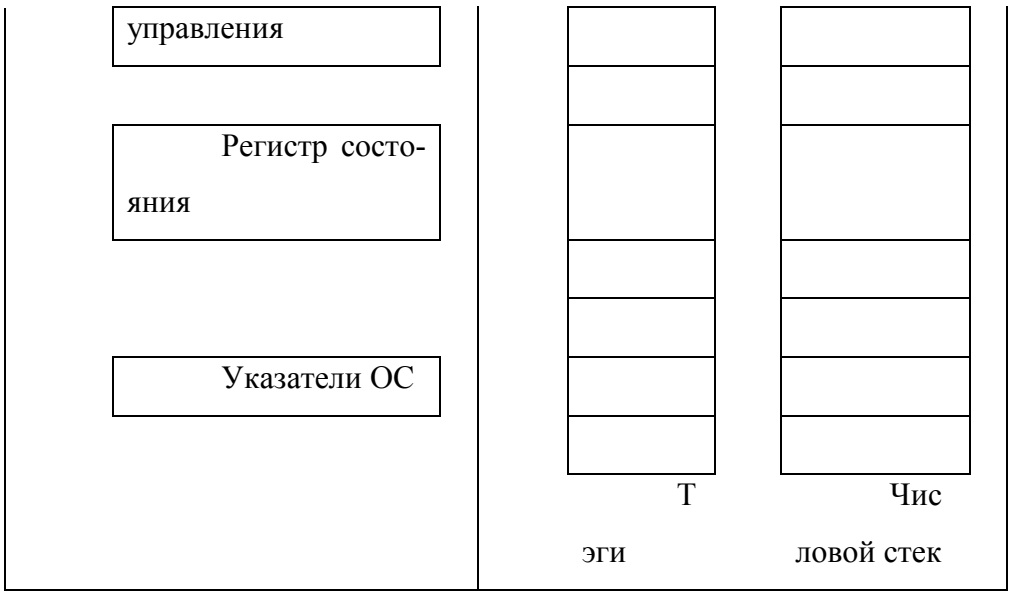


Рисунок 4.1

4.6.2.Синхронизация по данным.

Если сопроцессорная команда записывает операнд в ячейку памяти, а следующая команда центрального процессора использует этот операнд в дальнейших вычислениях, ЦП должен ожидать завершения операции сопроцессора. автоматически учесть такие ситуации сложно, поэтому вся работа по синхронизации в этом случае ложится на программиста. Для реализации этой синхронизации можно использовать команду FWAIT или любую другую “пустую” команду сопроцессора, например : FNOP, FST ST(0). В устройстве FPU - есть синхронизация по данным, которая автоматически реализуется ассемблером или компилятором языка высокого уровня.

5.ПРИМЕРЫ ПРОГРАММ

```
;Программа очистки экрана через видеобуфер
.model small
.486
.stack 100h
.code
begin: mov ax,@data
mov ds,ax
mov ax,0b800h; адрес видеобуфера текстовых режимов
mov es,ax; настройка регистра es на видеобуфер
mov di,00h; настройка регистра di  на начало видеобуфера
mov ax,0700h; определение атрибутов фона и символа: черный фон , белый символ
mov cx,2000; счетчик числа выводимых символов- выводим на экран 2000 пробелов
rep stosw
mov ax,4c00h; вызов функции завершения программы
int 21h
end begin
;*****************************************************************
;Программа демонстрации суммирования элементов двух массивов целых чисел
.model small
.486
.stack 100h
.data
  M1 db 1,2,3,4,5; первый исходный массив
  M2 db 6,7,8,9,10;второй исходный массив
  M3 db 5 dup(0); массив результата
.code
begin:mov ax,@data; настройка сегментного регистра ds
mov ds,ax; на сегмент данных
xor di,di;обнуление индексного регистра
mov cx,5;определение счетчика цикла
met:     mov al,M1[di];пересылка элемента M1 в регистр
add al,M2[di];суммирование элементов массивов
```

```
mov M3[di],al;запись в память результата
inc di; увеличение индекса элемента массива
loop    met; возврат на цикл
kon:    mov ah,4ch
int 21h
end begin
;*****************************************************************
;Программа перекодировки символов из 16 системы счисления в ASCII или BCD код
.model small
.486
.stack 100h
.data
SH db 0h,1h,2h,3h,4h,5h,6h,7h,8h,9h,0ah,0bh,0ch,0dh,0eh,0fh
ASCII db '0123456789abcdef'
BCD db 0,1,2,3,4,5,6,7,8,9,10h,11h,12h,13h,14h,15h
.code
begin:  mov ax,@data
mov ds,ax
mov al,SH[3]; загрузка 3 элемента массива в al
mov dl,al; пересылка его в dl
lea bx,ASCII; настройка адреса массива ASCII кодов на регистр bx
xlat ; перекодировка: в al вместо 16 кода – ASCII -код
mov ch,al; сохранение его в ch
mov al,dl; восстановление 16 кода в al
lea bx,BCD; настройка регистра bx на адрес массива BCD
xlat ; перекодировка: в al вместо 16 кода – BCD -код
mov cl,al; сохранение его в cl
mov ah,4ch; завершение программы
int 21h
end begin
;*****************************************************************
;Программа ввода и перевода вещественных границ интервала определения функции
.Model Small
.486
```

```
.Stack 200h
.Data
buf db 18; буфер для ввода вещественного числа
db 0; MS DOS возвращает число введенных символов
db 18 dup(0) ; максимальный размер вводимой строки
ten dw 10; константа 10
XL dq 0 ; левая граница интервала
XP  dq 0 ; правая граница интервала
mes1 db 13,10,' Введите левую границу'  ,13,10,'$'
mes2 db 13,10,' Введите правую границу'  ,13,10,'$'
mes3 db 13,10,' Левая граница не может быть равна правой!',13,10,'$'
mes4 db 13,10,' Левая граница не может быть больше правой!',13,10,'$'
c dw 0; переменная для двоичного кода одной цифры числа
;*********************************************************
.Code
 GRAN PROC; Процедура ввода и перевода вещественного числа
pusha
vvod:
mov ah, 0Ah; вызов системной функции для ввода строки
lea dx, buf
int 21h
fldz; загрузка в стек сопроцессора 0
mov si, 2; настройка si на первый символ введенной строки в буфере
cmp buf[si],'-'; первый символ - минус?
je  M1; да – переход на M1
cmp buf[si],'+'; первый символ – плюс?
jne M2; если нет – то переход на M2
M1:     inc si; переход к следующему символу в строке
M2:     cmp buf[si],'.'; очередной символ – десятичная точка
je  Drob; если да – то переход на обработку дробной части
cmp buf[si],0dh ; иначе – сравнение с кодом клавиши Enter – признак конца строки
je  Kon_enter; если да – то переход на метку Kon_enter
cmp buf[si],'0'; иначе – проверка: символ это цифра?
jb vvod; если код символа меньше кода нуля – то повторный ввод
```

```
cmp buf[si],'9';
ja vvod; если код символа больше кода девяти – то повторный ввод
mov al,buf[si]; если цифра – то загрузить ее код в al
sub al,'0'; перевод кода символа цифры в двоичный код
xor ah,ah; обнуление регистра
mov c, ax; сохранение двоичного кода цифры
fimul ten; умножение числа в вершине стека на 10
fild c; загрузка кода цифры в вершину стека
fadd; сложение с предыдущей частью числа
inc si
jmp M2; переход к следующему символу в целой части числа
Drob:; обработка дробной части
fldz
xor bx,bx
mov bl,buf[1]; загрузка в bx числа введенных символов
mov si,bx; настройка si на последний символ
inc si
L1:; цикл обработки дробной части
mov al,buf[si]
cmp al,'.'; сравнение с кодом десятичной точки
je Kon_drob;  если точка – то завершение обработки дробной части
cmp buf[si],'0'; проверка кода символа на цифры
jb vvod
cmp buf[si],'9'
ja vvod
sub al,'0'; перевод в двоичный код
xor ah,ah
mov c, ax; сохранение двоичного кода цифры
fild c; загрузка кода цифры в вершину стека
fadd; сложение с предыдущей дробной частью числа
fidiv ten; деление на 10
dec si
jmp L1; переход к следующей цифре
Kon_drob:       fadd; сложение целой и дробной частей числа
```

```
Kon_enter:
cmp buf[2],'-'; определение отрицательного числа
jne Kon; если число положительное – то на метку Kon
fchs; иначе – инвертировать число в вершине стека
Kon:
popa
ret
GRAN endp
;****************************************
Start:; начало основной программы
mov  ax,@data
mov  ds,ax
repeat:
finit; инициализация сопроцессора
mov ah,9; вывод первого сообщения
lea dx,mes1
int 21h
Call GRAN
fst XL; сохранение левой границы интервала
mov ah,9; вывод второго сообщения
lea dx,mes2
int 21h
Call GRAN
fst XP; сохранение правой границы интервала
fcom; сравнение левой и правой границ
fstsw ax; передача кодов условия в ЦП
fwait
sahf
je ravno; если границы равны – то ошибка
jb menshe; если правая граница меньше левой  – то ошибка
jmp end_progr; если границы введены верно – то выход из программы
ravno:
mov  ah,9
lea  dx,mes3
```

```
int  21h
jmp  repeat
menshe:
mov  ah,9
lea  dx,mes4
int  21h
jmp  repeat
end_progr:
mov ah,4ch
int  21h
End Start
```

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Григорьев В.Г. Микропроцессор i486. Архитектура и программирование (в 4-х книгах). Книга 1. Программная архитектура. - М., ГРАНАЛ, 1993. - с.346, ил.87.
2. Григорьев В.Г. Микропроцессор i486. Архитектура и программирование (в 4-х книгах). Книга 2. Аппаратная архитектура. Книга 3. Устройство с плавающей точкой. Книга 4. Справочник по системе команд. - М., ГРАНАЛ, 1993. - с.382, ил.54.
3. Юров В. Assembler, -СПб: Питер, 2001, - 624 с.
4. Зубков С.В. Assembler для Dos, Windows и UNIX. 2 – е изд., испр. и доп. – М., ДМК, 2000. – 608 с.

VHDL

1 Структура проекта

Проект в системе проектирования на основе VHDL представляется совокупностью иерархически связанных текстовых фрагментов, называемых ***проектными модулями***. Различают первичные и вторичные проектные модули, при этом:

<первичный модуль> ::= <декларация сущности> | <декларация пакета> | <декларация конфигурации>

<вторичный модуль> ::= <архитектурное тело> | <тело пакета>

Декларация **сущности** (entity) определяет имя проекта и его интерфейс, т.е. порты и параметры настройки. **Архитектурное тело** сущности описывает тем или иным образом функционирование устройства и (или) его структуру.

Каждой сущности сопоставляется одно или несколько архитектурных тел. Несколько вторичных модулей, соответствующих одному первичному, составляют набор возможных альтернативных реализаций объекта, представленного первичным модулем. Например, одной сущности может соответствовать несколько архитектурных тел, отличающихся степенью детализации описания и даже алгоритмом преобразования данных.

Первичные и соответствующие им вторичные модули могут сохраняться в различных файлах или записываться в одном файле. Важно лишь, чтобы они были скомпилированы в общую **проектную библиотеку**, причём первичный модуль компилируется раньше подчинённого ему вторичного. При записи первичного и вторичного модулей в одном файле первичный модуль записывается ранее соответствующего ему вторичного.

Типовой текст программы на VHDL имеет следующую структуру:

<VHDL-программа> ::=

{

{<объявление библиотеки>} {<объявление использования>} <первичный модуль>

}

{<вторичный модуль>}

<объявление библиотеки> ::= library <имя библиотеки>;

<объявление использования> ::= use <имя библиотеки>.<имя пакета>.<имя модуля>;

Иными словами, текст представляет собой произвольный набор первичных и вторичных модулей, причём каждому первичному модулю может предшествовать указание на библиотеки и пакеты, информация из которых требуется для построения этого модуля. После этого необходимо указать, что они будут использованы при помощи соответствующего объявления. В качестве имени модуля часто употребляют all — в этом случае будут использованы все имеющиеся модули. Отметим, что даже если несколько первичных модулей ссылаются на одни и те же библиотеки и модули, декларация использования должна предшествовать каждому

первичному модулю в отдельности.

2 Сущности и архитектурные тела

Синтаксис декларации сущности имеет вид:

```
<Декларация сущности> ::= entity <имя проекта> is
[<объявление параметров настройки>] [<объявление портов>] [<раздел деклараций>]
[begin <раздел опреаторов>] end [entity] <имя проекта>;
<объявление параметров настройки> ::= generic (<имя>:<тип> [:=<выражение>]
{;<имя>:<тип> [:=<выражение>]});
<объявление портов> ::=
port (<имя>:<режим> <тип> [:=<выражение>] {;<имя>:<режим> <тип>
[:=<выражение>]};
<режим> ::= in | out | inout
```

Объявление **параметров настройки** включается в декларацию сущности для создания проектов, которые предполагается использовать как фрагменты в различных других проектах, причём возможна модификация некоторых свойств данного компонента, точнее выбор параметра из множества значений, определённого типом.

Определение портов задаёт имена входных (in), выходных (out) и двунаправленных (in-out) линий передачи информации и тип данных, передаваемых через порты. Объявление портов, как следует из представленных правил синтаксиса, не обязательно. Возникает вопрос: зачем может понадобиться устройство, не имеющее входов и выходов? Оказывается, такая конструкция позволяет эффективно сочетать описание собственно проектируемого устройства и алгоритм его тестирования.

Для параметров и входных портов можно задавать значения по умолчанию. Они принимаются, если соответствующим единицам информации не присвоены другие значения в модулях высшего уровня иерархии.

Раздел деклараций сущности имеет такое же содержание т такой же смысл, что и раздел деклараций архитектурного тела (см. ниже) — объявляются локальные типы данных, подпрограммы, сигналы и т.п. При этом объявленные объекты доступны во всех архитектурных телах, подчинённых этой сущности.

Раздел операторов сущности может содержать только весьма ограниченный набор служебных операторов, которые на практике применяются редко. Поэтому обычно этот раздел остаётся пустым.

Архитектурные тела представляют содержательное описание проекта. Архитектурное тело в некотором смысле подчинено соответствующей сущности. Различают **структурные архитектурные тела** (описывают проект в виде совокупности компонентов и их соединений), **поведенческие архитектурные тела** (описывающие проект как совокупность исполняемых действий) и **смешанные тела**. Формальны признаков, по которым тело можно было бы отнести к тому или иному типу, не существует — речь идёт скорее не о чёткой классификации, а о стилях

описания, для каждого из которых характерными являются определённые операторы и которые лучше отражают разные аспекты одного и того же объекта (структурный и поведенческий соответственно).

Определены следующие правила записи архитектурных тел:

```
<Архитектурное тело> ::=
architecture <имя архитектуры> of <имя сущности> is <раздел деклараций>
begin
<раздел операторов>
end  [architecture]  <имя архитектуры>;
```

Здесь имя архитектуры — это любое индивидуальное имя проектного модуля. Имя сущности задаёт первичный модуль, которому подчиняется архитектурное тело. В разделе деклараций объявляются локальные для этого модуля информационные единицы — типы данных, сигналы, подпрограммы и т.д. Раздел операторов описывает правила функционирования и (или) конструирования устройства в терминах языка.

3 Типы данных

Язык VHDL основан на **концепции строгой типизации данных**, т.е. любой единице информации в программе обязательно присваивается имя, и для неё должен быть определён тип. Определение информационной единицы размещается в разделе деклараций программного модуля, в котором оно используется, или иерархически предшествующего модуля. **Тип данных** определяет набор значений объектов, отнесённых к этому типу, а также набор допустимых преобразований этих данных. Данные разных типов несовместимы в одном выражении.

Язык VHDL предоставляет некоторый базовый набор типов данных, которые не требуют объявления в программе пользователя. Кроме того, пользователь может определить свои типы данных и подключить существующие из библиотек. Различают скалярные типы данных и агрегатные типы. Объект, отнесённый к скалярному типу, рассматривается как законченная единица информации. Агрегат представляет упорядоченную совокупность скалярных единиц, объединённых единым именем. Классификация типов данных в VHDL представлена на рис. 1.

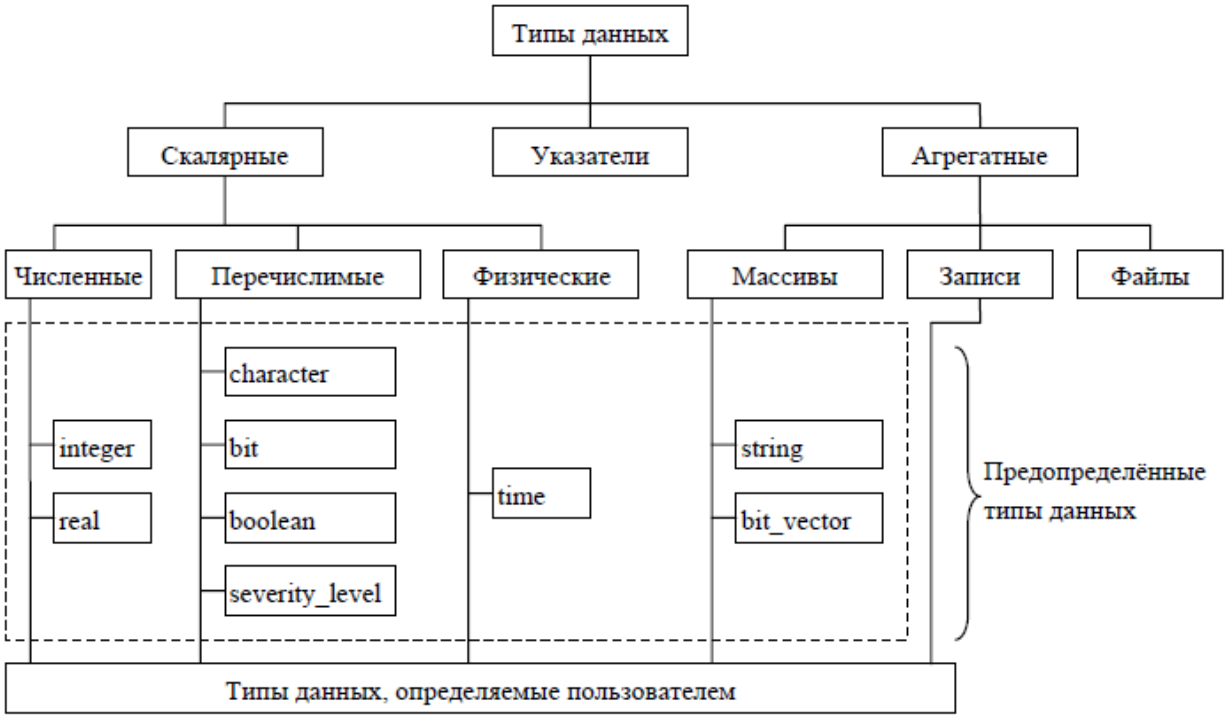


Рис. 1 – Классификация типов данных в VHDL

Данные, используемые в программах, относятся к одной из категорий:

- константы;
- переменные;
- сигналы.

Различие между переменными и сигналами будет рассмотрено в разделе 4. Декларация объектов имеет следующий вид:

```
<декларация объекта> ::= <категория> <имя>{,<имя>}:<тип> [:=<выражение>];
<категория> ::= constant | variable | signal
```

Одна декларация может определить несколько объектов. Выражение в декларации должно совпадать по типу с декларируемым объектом и задаёт значения константы либо начальные значения сигналов и переменных.

3.1 Предопределённые типы данных

```
<предопределённые типы> ::= integer | real | bit | boolean | character | string
| time | but_vector | severity_level | file_open_status | file_open_kind
```

Типы integer и real определяют численные данные — целые и действительные соответственно. Количество байт, отводимых для хранения чисел может зависеть от реализации, но стандартом считается 4 байта, что соответствует типам данных int и float для стандарта ANSI C. Для них определены следующие бинарные арифметические операции:

Таблица 1 Бинарные арифметические операции

Операция	Пример	
	Выражение	Результат
Сложение	5+6	11
Вычитание	8-3	5
Умножение	3*6	18
Деление (для целых)	9/4	2
Деление (для вещественных)	9/4	2.25
Деление по модулю (для целых)	9 mod 4 -14 mod 5	1 1
Остаток от деления (для целых)	9 rem 4 -14 rem 5	1 -4
Абсолютное значение (модуль)	abs (-5)	5
Возведение в степень	2**4	16

Также определены унарные арифметические операции сохранения и смены знака. В арифметических выражениях предполагаются традиционные способы определения приоритетов операций, включая применение скобок. Поддерживаются также операторы отношения (сравнения): =, /=, <, <=, >, >=. Результатом сравнения является значение типа boolean.

Данные типа bit могут принимать значения из множества {'0', ' 1'}. Для данных типа BIT определены логические операции:

- not — инверсия;
- or — операция ИЛИ;
- nor — операция ИЛИ-НЕ;
- and — операция И;
- nand — операция И-НЕ;
- xor — неравнозначность;
- xnor — равнозначность.

Данные типа boolean также могут принимать два значения: {true, false}, и для них определены те же операции, что и над данными типа bit. Разница между типами bit и boolean состоит в том, что первые применяются для представления уровней логических сигналов в аппаратуре, а вторые для представления обобщенных условий, например результатов сравнения. Данные разных типов несовместимы.

Тип character объединяет символы, определенные в используемой операционной системе — буквы, цифры, специальные символы. VHDL'87 допускает применение только первых 128 символов кодов ASCII (латинские буквы, цифры, специальные символы). В тексте программы символьная константа записывается как стандартный символ, заключенный в одинарные кавычки. Отметим, что символы '0' и

'1' имеют двойное — и как символ, и как логическое значение. В каждом конкретном случае тип данных определяется по контексту.

Тип time служит для задания задержек элементов и времени приостанова процессов при моделировании. Запись временной константы имеет вид:

```
<целое>  <единица измерения времени>
```

Определены следующие единицы измерения времени:

- fs — фептосекунда;
- ps — пикосекунда;
- ns — наносекунда;
- us — микросекунда;
- ms — миллисекунда;
- s — секунда.

Над данными типа "время" определены операции отношения, сложения и вычитания, а также умножения и деления на целое.

Данные этого типа применяются для задания задержек элементов и длительности интервалов останова. Важную роль при моделировании имеет определенная на этом типе функция now, возвращающая значение текущего времени на момент ее вызова. Например, выражение

```
now>  1  us
```

примет значение true через 1 микросекунду модельного времени от начала моделирования и может обеспечить формирование последовательности входных воздействий или контроль-ных точек при моделировании.

Тип severity_level задает следующее множество значений: {note, warning, error, failure} и используется для управления работой компилятора или программы моделирования. С помощью переменных и констант этого типа в операторах assert определяются действия, которые следует выполнить при обнаружении некоторых условий. Фактическая трактовка действий в стандарте не оговорена и оставлена на усмотрение разработчиков системы моделирования.

Типы FILE_OPEN_STATUS и FILE_OPEN_KIND обеспечивают возможность контроля процедур обмена между программой моделирования и файловой системой инструментального компьютера.

Типы string и bit_vector относятся к агрегатным и фактически определены как неограниченный массив символов и массив битов соответственно. Более подробно правила работы с массивами и их элементами будут рассмотрены далее. В тексте программы строковая константа заключается в двойные кавычки.

1.3.2 Скалярные типы, вводимые пользователем

Пользователь имеет возможность определить собственные типы данных, **используя декларацию типа данных**:

```
<декларация типа данных> ::=  TYPE  <имя типа>  IS  <определение типа>;
<определение типа> ::= <определение перечислимого типа> |<определение целого типа>
|<определение действительного типа>  |<определение физического типа>
```

|<определение типа массивов> |<определение типа записей>

Начнём рассмотрение пользовательских типов данных со скалярных типов. Определение **перечислимого типа** имеет вид:

<Определение перечислимого типа> ::= (перечислимое значение {,перечислимое значение});

<перечислимое значение> ::= <идентификатор> | <символьная константа>

Например:

```
TYPE state IS (SO,SI,S2,S3);
```

может представлять, например, набор допустимых состояний системы, для каждого состояния определяются выполняемые действия и правила перехода в другое состояние.

Важнейшим приложением перечислимого типа данных являются расширенные алфавиты моделирования, в частности тип std_logic и соответствующий ему тип std_logic_vector, описанный в пакете std_logic_1164. Данный девятизначный алфавит моделирования рассмотрен в приложении 1 настоящего методического указания.

Определение **численных типов пользователя** целесообразно, во-первых, для контроля совместимости данных в программах, а во-вторых, для точного задания разрядности слов, представляющих данные в проектируемом объекте. В общем случае определение ограниченного типа подчиняется синтаксическому правилу:

<Определение ограниченного типа> ::= [<базовый тип>] <диапазон>;

<диапазон> ::= range <ограничение><направление><ограничение> |range<>

<направление> ::= downto | to

Направление (to — увеличение, downto — уменьшение) должно быть согласовано с соотношением ограничений.

Примеры:

```
type unsigned_short is integer range 0 to 255;
type my_data is integer range -2**(n-l)+l to 2**(n-l)-l;
type input__level is -10.0 to +10.0;
```

Тип unsigned_short объединяет целые положительные числа, которые могут быть представлены в байтовом формате.

Тип my_data объединяет целые в диапазоне, который объявляет пользователь через разрядность данных n. В этом случае пользователь точно указывает компилятору число разрядов, необходимое для представления данных, обеспечивая экономию ресурсов микросхемы по сравнению с неограниченным типом.

При объявлении типа input_level базовый тип явно не задан, тип ограничений устанавливается в соответствии с типом их фактических значений.

3.3 Физические типы

Наряду с предопределенным типом time пользователь может определить другие физические типы, которые будут отражать физические (механические, электрические или иные) свойства носителя информации.

<Определение физического типа> :: = range <диапазон> units

```
<имя базовой единицы> {<имя вторичной единицы> = <значение единицы>} end units
[<имя типа>];
```

Например:

```
type voltage is range -5e6 to +5e6;
units uv; -- базовая единица - микровольт
mv = 1000 uv; -- милливольт
v=1000 mv; -- вольт
end units voltage;
```

Введение такого типа позволяет описывать и моделировать сопряжение цифровой логической схемы с аналоговыми источниками. В VHDL-AMS этот тип введен в пакет electrical_systems и фактически так же может считаться предопределенным.

3.4 Агрегатные типы

3.4.1 Массивы

Массив, как и в других языках программирования высокого уровня, — это набор данных, объединенных общим именем и различаемых по порядковым номерам (индексам). Для того чтобы ввести объект типа "массив", необходимо предварительно объявить соответствующий тип на основе следующих синтаксических правил:

```
<определение типа массива> ::=
array (<диапазон> {, <диапазон>}) of <тип элемента массива>
```

Диапазон задает множество допустимых значений индекса. Число измерений массива формально не ограничено. Если диапазон задан конструкцией range<>, то это является объявлением неограниченного массива. В этом случае определяется не диапазон значений индекса, а только тип индексной переменной. Такое определение предполагает задание диапазона при определении конкретного экземпляра объекта, относимого к такому типу, например, при вызове подпрограмм. В подобных случаях диапазон устанавливается динамически в соответствии с диапазоном подставляемого фактического параметра.

Примеры:

```
type ram1 is array (length-1 downto 0) of integer range 2**width-1 downto 0;
type ram2 is array (length-1 downto 0,width-1 downto 0) of std_logic;
type ram3 is array (integer range<>, integer range<>) of std_logic;
```

Во всех приведенных декларациях объявляется в сущности одно и то же, а именно — матрица ячеек памяти емкостью length слов по width разрядов в каждом, причем предполагается, что эти параметры были ранее определены. Однако выполнено это разными способами, а значит, и ссылаться на эти типы следует по-разному, ram1 и ram2 определены как ограниченные типы массивов, ram1— как одномерный массив целых, а ram2 — как двумерный массив битов; ram3 определен как неограниченный тип и требует задания границ индексов при декларации объектов выбранного типа.

Декларации объектов, принадлежащих приведенным типам, могут выглядеть следующим образом:

```
variable ram1_instance: ram1; variable ram2_instance: ram2;
variable ram3_instance: ram3 (length-1 downto 0, width-1 downto 0);
```

При обращении к элементам массива в программе индексы помещаются в скобках следом за именем массива. Тип индексного выражения должен соответствовать типу индекса, объявленного при декларации типа массива. При обращении к элементу многомерного массива индексные выражения записываются через запятые в порядке, определенном в декларации типа.

Для одномерных массивов определено несколько групповых операций, в которых массив рассматривается как единое целое. Это, прежде всего, операция конкатенации & (объединение строк). Например, приведенная ниже последовательность операторов присваивает сигналу b значение "11011001".

```
a := "1001"; b <= "1101" & a;
```

Здесь a и b — строки или битовые векторы, причем a — переменная, а b — сигнал. **Операции сдвига** определены для одномерных массивов типа bit или boolean и записываются следующим образом:

```
<имя массива> <символ операции сдвига> <целое>
```

В VHDL'93 определены следующие операции сдвига (в VHDL'87 их нет):

- логические сдвиги влево и вправо sll и srl;
- арифметические сдвиги влево и вправо sla и sra;
- циклические сдвиги влево и вправо rol и ror.

Целое в записи выражения для сдвига определяет число разрядов, на которые осуществляется сдвиг кода.

В составе большинства современных САПР поставляются пакеты, определяющие ариф-метические операции над битовыми массивами (кодами). Как правило, они поставляются в виде пакета std_logic_arith.

3.4.2 Записи

Запись — эта структура данных, каждая информационная единица которой, называемая полем записи, имеет индивидуальное имя и может быть индивидуального типа. Записи удобны для агрегатирования различных данных, характеризующих один объект. Для использования записей как переменных сначала надо объявить соответствующий тип:

```
<определение типа записи> ::=
record <список полей записи> : < тип>; {<список полей записи> : < тип>;}
end record;
```

Рассмотрим пример. Определим тип pixel, представляющий цветовые составляющие отображения точки на экране в формате цветопередачи, предусматривающей восьмиразрядное представление трех цветовых составляющих:

```
type pixel is
```

```
record red, green, blue: integer range 0 to 255; end record;
```

Тогда тип "видеопамять" может быть определен как:

```
type video_ram is array (integer range<>, integer range<>) of pixel;
```

Экземпляр видеопамяти будет определяться, например, следующим образом:

```
signal VRAM : video_ram (479 downto 0, 639 downto 0);
```

Этот экземпляр может сохранять информацию об изображении размером 480 строк по 640 элементов в строке. Выборка значения красной составляющей верхнего левого элемента изображения из такой памяти описывается оператором

```
Out_red <= VRAM (0,0).red;
```

Следующий пример определяет обобщенный тип для представления конечных автоматов. Автомат, как известно, определяется множеством входов, множеством состояний и множеством выходов, а также соответствующими функциями на этих множествах. Значит, можно ввести универсальный тип:

```
type state_machine is record
s : state;
x : machine_input; y : machine_output; end record;
```

Здесь state, machine_input и machine_output — ранее определенные перечислимые типы. Функции переходов и выходов конкретного экземпляра автомата будут определяться в разделе операторов соответствующего архитектурного тела.

3.5 Подтипы

Специфическим понятием языка VHDL является подтип. Объекты, отнесенные к подтипу, сохраняют совместимость с данными типа, из которого выделяется подтип так называемого базового типа. Однако введение подтипа:

- определяет множество допустимых значений данных подтипа как подмножество допустимых значений базового типа;
- позволяет вводить дополнительные функции преобразования, определяемые только для данных подтипа.

Синтаксис декларации подтипа следующий:

```
<декларация подтипа> ::=
subtype <имя подтипа> is [<имя функции разрешения>] <имя базового типа или подтипа> [<ограничение>];
```

Пример:

```
subtype bit_in_word_number is integer range 31 downto 0;
```

Определен подтип типа integer. Данные этого подтипа предполагается использовать для индексации бита в 32-разрядном коде. Данные совместимы с данными типа integer. Однако присвоение этим данным значений вне указанного диапазона вызывает сообщение об ошибке.

Использование функции разрешения будет описано в разделе 9.

4 Сигналы и переменные

Любой проект является описанием явлений в дискретных системах. Эти явления могут представляться тремя различными категориями данных: **константы**, **переменные** и **сигналы**.

SIGNAL — это информация, передаваемая между модулями проекта или представляющая входные и выходные данные проектируемого устройства. Сигналу присваиваются свойства изменения во времени.

VARIABLE — это вспомогательная информационная единица, используемая для описания внутренних операций в программных блоках. Присвоение значения сигналу отображается знаком <=, а переменной — знаком :=.

Для того чтобы представить различия сигналов и переменных, следует сделать несколько предварительных замечаний. В языке VHDL введены два типа операторов: последовательные и параллельные.

Последовательные операторы выполняются последовательно друг за другом в порядке записи. Такие операторы во многом подобны операторам традиционных языков программирования высокого уровня и описывают набор действий, которые последовательно выполняются над исходными данными с целью получения результата. К этому классу операторов относят оператор присваивания переменной, последовательный оператор присваивания сигналу, условные операторы, оператор выбора и ряд других.

Исполнение **параллельных операторов** инициируется не по последовательному, а по событийному принципу, т. е. они исполняются тогда, когда реализация других операторов программы создала условия для их исполнения. Параллельные операторы представляют части алгоритма, которые в реальной системе могут исполняться одновременно. Эти части взаимодействуют между собой и с окружением проектируемой системы.

Наиболее явно разница между сигналами и переменными проявляется при интерпретации операторов последовательных присвоений. Для обоих видов сохраняется общее для последовательных операторов правило начала исполнения: первый оператор в **процессе** (см. раздел 6) исполняется после выполнения условий инициализации процесса, а каждый следующий сразу после исполнения предыдущего. Однако результат присвоения переменной непосредственно доступен любому последующему оператору в теле процесса. Трактовка оператора последовательного присвоения сигналу существенно отличается от трактовки присвоения переменной или операторов присваивания в традиционных языках программирования. Присвоение сигналу не приводит непосредственно к изменению его значения. Новое значение сначала заносится в специальный буфер, называемый **драйвером сигнала**, и следующие операторы в теле процесса оперируют со старыми значениями. Фактическое изменение значения сигнала выполняется только после исполнения до конца процессов и других параллельных операторов, инициированных общим событием, или после исполнения оператора

останова wait (см. раздел 7.3).

Сформулируем наиболее существенные различия сигналов и переменных:

Переменные меняют значения сразу после присвоения, и новые значения непосредственно учитываются во всех преобразованиях, записанных в теле процесса после такого присвоения.

Значение сигнала меняется не сразу после выполнения присвоения. Оператору присваивания сопоставляется некий буфер, называемый контейнером или, чаще, драйвером сигнала. Оператор присваивания передает новое значение драйверу сигнала, и лишь после того, как выполнены преобразования во всех процессах, инициированных общим событием, содержание драйвера передается сигналу. Передача значения сигналу может быть еще более задержана, если оператор присваивания содержит выражение задержки after.

Переменная определена только внутри тела процесса, сигнал — во всем архитектурном теле.

Переменной можно многократно присваивать значение в теле процесса. Сигнал внутри одного процесса может иметь только один драйвер, т. е. присвоение значения сигналу может быть выполнено только один раз в теле процесса (на различных несовместимых путях реализации алгоритма могут быть несколько операторов присваивания значений одному сигналу).

5 Атрибуты

Атрибуты — скаляры, отражающие некоторые свойства объектов, используемых в программных модулях (типов, переменных, агрегатов). Например, атрибуты типа предназначены для сжатого представления информации о множестве значений, объединенных типом, а атрибуты сигнала — для представления временных свойств сигнала. В разделах операторов нельзя присваивать значение атрибуту, способ его определения задается декларацией атрибута. Атрибуту присваивается имя и тип, имя является обычной переменной в выражениях того типа, который присвоен атрибуту. Имя атрибута записывается следующим образом:

<имя атрибута> ::= <имя атрибутируемого объекта>'<определитель атрибута> [(<выражение>)]

Определитель атрибута (attribute designator) определяет свойство объекта, представляемое атрибутом. Необязательное выражение может задавать дополнительные данные для вычисления значения атрибута. Пользователь может создавать свои атрибуты, однако здесь мы ограничимся рассмотрением только наиболее употребительных предопределенных атрибутов.

Предопределенные атрибуты типов приведены в таблице 2. Здесь T — имя типа, N — целое, а X — вспомогательное выражение, тип которого совпадает с типом T. Тип пере-численных атрибутов, кроме T'pos и T'image, совпадает с атрибутируемым типом. Атрибут T'pos — принимает целое значение, а T'image — строка.

Таблица 2 Предопределенные атрибуты типов

Вид атрибута	Вычисляемое значение	Атрибутируемый тип
`T'left`	Левая граница значений T	Любой скалярный
`T'right`	Правая граница значений T	Любой скалярный
`T'low`	Нижняя граница значений T	Численный, физический
`T'high`	Верхняя граница значений T	Численный, физический
`T'image(X)`	Строка символов, представляющая значение X	Любой
`T'pos(X)`	Позиция значения X в наборе значений T	Перечислимый
`T'val(N)`	Значение элемента в позиции N в наборе значений T	Перечислимый, физический, целый
`T'leftof(X)`	Значение в наборе значений T, записанное в позиции слева от X	Перечислимый, физический, целый
`T'rightof(X)`	Значение в наборе значений T, записанное в позиции справа от X	Перечислимый, физический, целый
`T'pred(X)`	Значение в наборе значений T на одну позицию меньшее X	Перечислимый, физический, целый
`T'succ(X)`	Значение в наборе значений T на одну позицию большее X	Перечислимый, физический, целый

Предопределенные атрибуты массивов, приведенные в таблице 3, упрощают запись подпрограмм и описаний настраиваемых модулей. Они, в частности, позволяют записывать границы обработки безотносительно к фактическому размеру массива. В таблице 3 A — имя типа массива, а N — порядковый номер измерения многомерного массива. Для одномерного массива N=1, но можно выражение в скобках при записи атрибута вообще опускать. Тип результата всегда совпадает с типом индекса. Смысл определителей left, low, right, high такой же, как у определителей типов.

Таблица 3 Предопределенные атрибуты массивов

Имя атрибута	Результат
`A'left(N)`	Левая граница диапазона индексов N-й координаты массива A
`A'right(N)`	Правая граница диапазона индексов N-й координаты массива A
`A'low(N)`	Нижняя граница диапазона индексов N-й координаты массива A

A'high(N)	Верхняя граница диапазона индексов N-й координаты массива A
A'range(N)	Диапазон индексов N-й координаты массива A
A'reverse_range(N)	Обратный диапазон индексов N-й координаты массива A
A'length(N)	Диапазон индексов N-й координаты массива A

Атрибуты сигналов являются эффективным средством анализа поведения сигнала во времени. В таблице 4 символ S означает имя сигнала.

Таблица 4 Атрибуты сигналов

Имя атрибута	Тип атрибута	Значение
S'delayed (T)	То же, что у S	Значение S, существовавшее на время T перед вычислением атрибута
S'event	boolean	Сигнализирует об изменении сигнала
S'stable	boolean	S'stable = not S'event
S'active	boolean	true, если присвоение сигналу выполнено, но значение еще не изменено (не закончен временной интервал, заданный выражением after)
S'quiet	boolean	S'quiet = not S'active
S'last_event	time	Время от момента вычисления атрибута до последнего перед этим изменения сигнала
S'last_active	time	Время от момента вычисления атрибута до последнего присвоения значения сигналу (не совпадает с last_event при наличии слова after в определяющем выражении)

6 Процессы

Как уже отмечалось выше, параллельные операторы представляют части алгоритма, которые в реальной системе могут исполняться одновременно. Эти части взаимодействуют между собой и с окружением проектируемой системы. Параллельные операторы могут быть простыми и составными. Составной оператор включает несколько простых операторов, для которых определены общие условия инициализации. Такая совокупность операторов называется телом составного оператора. Важнейшим составным оператором является **оператор процесса** process, синтаксис которого определен следующим образом:

<оператор процесса> ::=

[<метка процесса>:] process[(<список чувствительности>)] [is] <раздел деклараций>

begin

<раздел операторов>

end process [<метка процесса>];

<раздел операторов>::= {<последовательный оператор>}

Ключевое слово is в версии VHDL'93 является необязательным, а в VHDL'87 недопустимо в данной конструкции.

Последовательные операторы могут записываться только в теле оператора process. При моделировании фрагменты алгоритма, заключенные в оператор process, будут исполняться друг за другом после возникновения в системе **инициализирующего события** — изменении одного из сигналов, перечисленных в **списке чувствительности**, или в заранее определенный момент времени. Параллельные операторы в теле процесса не определены. Переменные могут быть определены только в теле процесса, а сигналы во всем архитектурном теле.

Общие правила интерпретации оператора process можно свести к следующим:

- Процесс "запускается" при изменении любого сигнала, перечисленного в списке чувствительности. Если список чувствительности пуст, то процесс безусловно исполняется при начальном запуске, а также сразу за исполнением последнего оператора в разделе операторов этого процесса. При этом надо иметь в виду, что оператор процесса без списка чувствительности обязательно должен содержать в своем теле оператор ожидания wait (см. раздел 7.3). Иначе исполнение любых других операторов в программе блокируется.
- Все операторы раздела операторов выполняются подряд друг за другом от начала до конца, за исключением случаев приостановки исполнения действий оператором wait. Тогда после приостановки может быть инициировано исполнение других процессов и параллельных операторов, а реализация операторов, следующих за оператором wait, продолжится после наступления события, объявленного в этом операторе.

7 Последовательные операторы

Последовательные операторы (Sequential Statement) по характеру исполнения подобны операторам традиционных языков программирования высокого уровня. Операторы этого типа обязательно вложены в оператор process или подпрограмму и выполняются последовательно друг за другом в порядке записи. Результаты исполнения последовательных операторов недоступны прочим программным модулям, по крайней мере, до того, как будет выполнен оператор ожидания wait, или не будут выполнены до конца все процессы, инициированные общим событием. Это можно трактовать так, что с точки зрения окружения все операторы в теле процесса от его начала до оператора wait, а при отсутствии wait — до конца тела, исполняются единомоментно. При использовании выражения задержки сигнала after изменение сигнала прогнозируется на еще более отстоящий момент времени.

Ниже приведен полный список последовательных операторов языка.

<Последовательный оператор> ::= <оператор ожидания> | <оператор проверки> | <последовательное сигнальное присваивание> | <присваивание переменной> | <вызов

процедуры> | <условный оператор> | <оператор выбора> | <оператор повторения> | <оператор перехода к новому циклу> | <оператор выхода из цикла> | <оператор возврата> | <пустой оператор>

В данном разделе будут рассмотрены основные последовательные операторы VHDL.

7.1 Операторы присваивания

Синтаксическая формула **оператора присваивания** значения сигналу имеет вид:

<оператор присваивания сигналу> ::= <приёмник> <= [<модель задержки>] <прогноз поведения>;

<модель задержки> :: = transport | [reject <выражение времени>] inertial

<прогноз поведения> ::= элемент поведения> {, <элемент поведения>}

<элемент поведения> ::= <значащее выражение> [after <выражение времени>]

Приёмник — объект сигнальной категории, представленный простым именем или компонентом агрегатного сигнала. Прогноз поведения (в стандартах VHDL — Waveform, временная диаграмма) задает порядок изменения сигнала после события, инициирующего исполнение этого оператора. При этом временные интервалы для определения переходов задаются относительно времени возникновения инициирующего события. **Значащее выражение** — любое выражение, дающее результат того же типа, что и приемник.

Если **временное выражение** опущено, полагается нулевая задержка (так называемая **дельта-задержка**): изменение (если оно действительно предсказано при вычислении значащего выражения) заносится в календарь событий с той же отметкой времени, что и инициирующее событие.

Если в некоторый момент модельного времени выполняется присвоение сигналу, для которого ранее были предсказаны переходы, часть этих переходов может быть исключена. Такая ситуация возникает, например, когда процесс, содержащий оператор присваивания, инициируется несколькими сигналами, изменения которых отстоят друг от друга на время меньшее, чем определено в прогнозе поведения (напомним, что каждое изменение любого сигнала из списка чувствительности вызывает исполнение тела процесса).

Порядок исключения зависит от принимаемой **модели задержки**. Различают транспортную и инерционную модели задержки. Если в операторе присваивания присутствует ключевое слово transport, предполагается транспортная задержка.

Транспортная модель предполагает идеализацию поведения устройства так, что любой импульс, сколь коротким он бы ни был, воспроизводится на выходе. В этом случае из временной диаграммы (фактически, из календаря событий) исключаются все переходы, которые были предсказаны на время, позднее первого из новых объявляемых переходов, и добавляются новые переходы.

По умолчанию, а также если использовано объявление inertial, предполагается инерционная задержка. В VHDL'87 слово inertial не определено, инерционная

задержка выбирается по умолчанию при отсутствии опции модели задержки.

Инерционная модель применяется для описания устройств, не реагирующих на им-пульсы, длительность которых меньше некоторого наперед заданного значения. В этом слу-чае, подобно транспортной задержке, в календарь событий добавляются новые предсказан-ные переходы и удаляются все переходы, предсказанные в предшествующих присвоениях на время большее времени нового прогнозируемого перехода. После этого просматривается интервал модельного времени, который предшествует новому предсказываемому переходу и длительность которого определена временным выражением в подстроке reject. Все переходы в этом интервале, которые приводят к значению, отличающемуся от нового предсказания, удаляются. Если ключевое слово reject отсутствует, то интервал, в котором выполняется такое удаление, определяется значением, указанным после слова after.

Иными словами, инерционная модель задержки предполагает, что элемент не реагирует на сигналы, длительность которых меньше порога, равного времени задержки элемента.

Рассмотрим различные варианты модели задержки на следующем примере. Для этого будем рассматривать объект с задержкой как совокупность двух компонентов: идеального элемента и элемента задержки. В частности, модель элемента 2И состоит из идеального вен-тиля и блока задержки (см. рис. 2.):

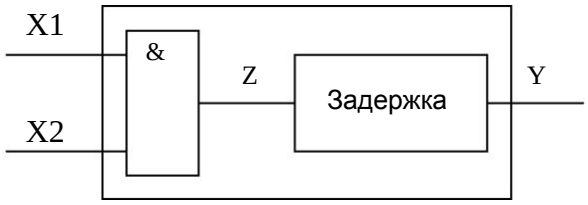


Рис. 2 Модель элемента с задержкой

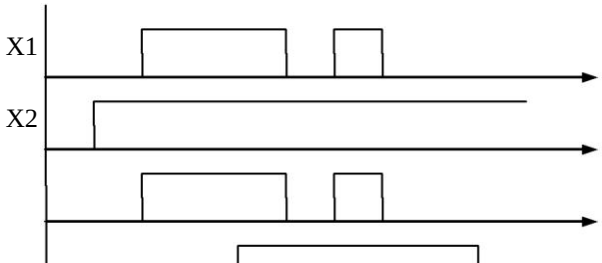
Опишем данный элемент:

```
entity  and2  is
port (X1, X2 : in bit; Y : inout bit); end and2;
```

Для сущности and2 создадим 2 архитектурных тела, в которых реализуем инерционную и транспортную задержки:

```
architecture and2_inertial of and2 is begin
Y  <=  X1  and  X2  after  10  ns;
end;
architecture and2_transport of and2 is begin
Y  <=  transport  X1  and  X2  after  10  ns;
end;
```

Сравним результаты моделирования для данных архитектурных тел при подаче сигналов, длительность которых меньше времени задержки (см. рис. 3.):



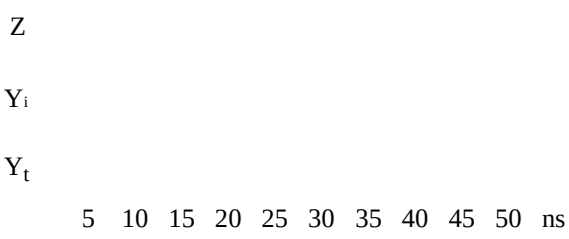


Рис. 3 Временная диаграмма для инерционной и транспортной моделей задержки

Для инерционной модели возможно использование дополнительной опции — можно игнорировать сигналы, длительность которых меньше определённого порога, отличного от времени задержки всего элемента. Такой вариант поведения называется резекцией.

Пример резекции сигнала длительность менее 1 ns (более длительные импульсы проходят на выход):

```
Y <= reject 1 ns inertial X1 and X2 after 10 ns;
```

Фактически, этот оператор эквивалентен следующим двум:

```
TMP <= X1 and X2 after 1 ns;
Y <= transport TMP after 9 ns;
```

7.2 Оператор условия и оператор выбора

Операторы условия if и выбора case позволяют описывать совокупности действий, некоторые из которых исполняются при возникновении определенных условий в реальном устройстве и при моделировании, а иные при тех же условиях не исполняются.

Синтаксис **оператора условия** имеет вид:

```
<оператор условия> ::= if <булевское выражение> then <оператор> {<оператор>}
{elsif <булевское выражение> then <оператор> {<оператор>}}
[else <оператор> {<оператор>}] end if;
```

В качестве операторов в приведенной конструкции могут выступать любые последовательные операторы, в том числе и операторы условия или выбора. В этом случае говорят об иерархическом вложении операторов. Формальных ограничений на глубину вложений не вводится, хотя надо иметь в виду, что некоторые компиляторы могут оказаться неспособны выполнить прямую реализацию в аппаратуре синтаксических конструкций с большим числом уровней вложения.

Проиллюстрируем применение уловного оператора описанием поведения вентиля И2, рассмотренного в предыдущем разделе, для которого время задержки для фронта и среза не совпадают:

```
architecture and2_if of and2 is
begin
      process (X1, X2)
      variable Z : bit;
      begin
            Z := X1 and X2;
            if Z='1' and Y='0' then
                  Y <= '1' after 7 ns;     --фронт
            elsif Z='0' and Y='1' then
```

```
                Y  <=  '0'  after  3  ns;     --срез
            end  if;
      end  process;
end;
```

Синтаксис **оператора выбора** имеет вид:

```
<оператор выбора> ::=
case  <ключевое  выражение>  is
when  <вариант>  {|<вариант>}  =>  <оператор>  {<оператор>}  {when  <вариант>
{|<вариант>} => <оператор> {<оператор>}} end case;
<вариант> ::=  <константное выражение>  |  <диапазон>  |  others
```

Разделителем в списках выбираемых вариантов является вертикальная черта, т. е. в определении оператора выбора вертикальная черта это не метасимвол БНФ, в отличие от любых других определений, а синтаксический элемент определяемой конструкции.

Тип константного выражения или диапазона в записи варианта совпадают с типом ключевого выражения. В частности, в качестве этого выражения может использоваться строка, которая соответствует значению битового вектора или вектора типа std_logic.

При каждом исполнении оператора выбора реализуется единственная последовательность вложенных операторов, а именно та, которой предшествует вариант, совпадающий со значением ключевого выражения в момент исполнения оператора. Если вариант представлен диапазоном, то соответствующая последовательность операторов исполняется при условии, что значение ключевого выражения принадлежит этому диапазону.

Ключевое слово others определяет операторы, которые исполняются, если значение ключевого выражения не совпадает ни с одним вариантом и не входит в объявленные диапазоны. Если алгоритм предусматривает варианты, при которых не производится никаких действий, то в операторной части таких вариантов записывается пустой оператор.

Практически любой разветвленный фрагмент можно описать с использованием и оператора условия, и оператора выбора. Применение одного из способов записи — дело вкуса программиста. Тем не менее, оператор case предпочтительнее, когда выбор ветви алгоритма связан с одной переменной, принимающей дискретное множество значений. Иногда набор условий легко приводится к одной такой переменной.

7.3 Оператор ожидания

Исполнение операторов, записанных в теле процесса, приостанавливается, если очередной оператор является **оператором ожидания** (фактически — **оператором приостанова**) wait. При этом результаты исполнения предшествующих операторов заносятся в календарь событий и могут быть инициализированы другие процессы. Прекращение состояния приостанова процесса зависит от условий, определенных в операторе wait. Определено несколько модификаций оператора wait:

```
<оператор ожидания> ::= wait;
```

```
| wait on <имя сигнала> {,<имя сигнала>}; | wait until <условие>;
| wait  for  <выражение времени>;
```

Вариант оператора wait без дополнительных уточняющих конструкций соответствует бесконечному останову. В этом случае после достижения такого оператора процесс никогда больше не будет исполняться. Указанная версия пригодна для описания процедур инициализации систем, а также фрагментов, работа которых при некоторых условиях прекращается навсегда. Обычно такой оператор завершает программы генерации тестов, означая окончание тестовой последовательности.

Список сигналов в варианте wait on эквивалентен списку чувствительности процесса: исполнение будет продолжено после того, как один из сигналов списка изменит свое значение.

Приостанов, заданный конструкцией wait until, заканчивается, когда выполнено определенное оператором условие, т. е. соответствующее выражение принимает значение true.

Вариант ожидания по времени иллюстрируется процессом generator. Данный процесс выполняется бесконечно, приостанавливаясь каждые 50 ns модельного времени, причем перед приостановом уровень сигнала clock меняется на противоположный. В момент при-останова могут быть инициированы параллельные операторы программы, в том числе другие процессы.

```
generator: process begin
clock <= '0'; wait for 50 ns;
clock <= not clock; end process generator;
```

Процесс, содержащий оператор wait, не может иметь списка чувствительности. Это связано с тем, что трудно описать систему, в которой может произойти повторная инициализация действий, в то время как реакция на предыдущее событие еще не реализована, например, произошло изменение одного из инициирующих сигналов, когда время ожидания еще не вышло. Еще раз напомним, что оператор wait, как и другие последовательные операторы, может размещаться только в теле процесса или теле подпрограммы.

7.4 Операторы повторения

Операторы повторения loop позволяют сокращенно записывать совокупности однотипных действий.

```
<оператор повторения> ::=
[<метка оператора повторения>:] [<итерационная схема>] loop <оператор>
{<оператор>}
end  loop  [<метка оператора повторения>];
<итерационная схема>::= while <условие>
|  for  <имя переменной>  in  <диапазон>
```

Последовательность операторов (здесь могут быть только последовательные операторы), заключенная между ключевыми словами loop и end loop, называется **телом оператора повторения** или **телом цикла**. Операторы в теле цикла выполняются друг другом в порядке записи, причем такое выполнение повторяется многократно. Число повторений определяется **итерационной схемой**.

Оператор повторения, не содержащий явного объявления итерационной схемы, предполагает **бесконечное повторение** последовательностей вложенных в него операторов. Такая модель, в целом, соответствует поведению реальных дискретных устройств, повторяющих некоторую последовательность действий вплоть до отключения питания. В то же время эта конструкция имеет логический смысл, только если тело цикла содержит оператор ожидания wait или оператор выхода из цикла exit. В противном случае бесконечное безусловное повторение блокировало бы исполнение любых других операторов и процессов.

Оператор с ключевым словом while обязательно должен содержать в теле цикла операторы, изменяющие описанное в итерационной схеме условие. Операторы цикла повторяются, пока при вычислении условия не получается значения false. Условие проверяется каждый раз перед исполнением тела цикла.

Оператор повторения с ключевым словом for повторяется для всех значений переменной из заданного итерационной схемой диапазона.

Отметим, что присвоенные в теле цикла значения переменных могут быть исходными данными для очередного цикла. Если же в цикле выполнено присвоение значения сигналу, то в следующих операторах тела и очередных повторениях того же цикла используются старые значения, если только тело цикла не содержит операторов ожидания.

Если при моделировании и, в частности, при описании тестовых воздействий смысл операторов повторения практически не отличается от смысла подобных конструкций в традиционных языках программирования высокого уровня, то при интерпретации в аппаратуре имеются существенные отличия. Предусматривается не просто последовательное во времени повторение набора преобразований, а **реализация набора устройств**, выполняющих однотипные действия, причем эти устройства работают параллельно.

Число устройств определяется итерационной схемой. Для оператора с ключевым словом for — это просто число значений переменной в объявленном диапазоне. Для варианта с ключевым словом while условие не может быть связано с сигнальными данными, способными изменяться в реальном устройстве. Операторы повторения должны иметь логически постоянные границы, т.к. в противном случае не ясно, сколько повторяющихся блоков в устройстве реально потребуется.

Кроме раздела итерационная схема порядок реализации повторений может задаваться дополнительными операторами: оператором перехода к следующему циклу next и оператором выхода из цикла exit.

Оператор next блокирует исполнение всех последующих операторов в текущем цикле и обеспечивает автоматический переход к следующей итерации.

Оператор записывается следующим образом:

<оператор перехода к следующей итерации> ::=

```
[<метка>:] next [<метка оператора повторения>] [when <условие>]
```

Оператор exit прекращает исполнение не только текущего цикла, но и всех последующих циклов, заданных итерационной схемой исполняемого оператора.

Синтаксис оператора exit имеет вид:

```
<оператор прекращения цикла> ::=
[<метка>:] exit [<метка оператора повторения>] [when <условие>]
```

Необязательная метка в операторах next и exit используется при записи вложенных циклов. Такая метка указывает, что прерывается не только данный цикл, но и все иерархически предшествующие ему циклы, вплоть до цикла, оператор которого помечен этой меткой.

7.5 Операторы проверки

Оператор проверки assert относится к категории конструкций, не подлежащих реализации в аппаратуре. Оператор служит для выявления специфических ситуаций, которые могут возникать в процессе компиляции и моделирования (т. е. программной интерпретации описания проекта), и выдачи в этих ситуациях сообщения разработчику. Синтаксис оператора проверки определен следующим образом:

```
<оператор проверки> ::= assert <булевское выражение> [report <строка сообщения>]
[severity <уровень важности>];
```

При выполнении этого оператора в процессе моделирования проверяется условие и, если получено значение true, выполняется переход к следующему оператору программы. В противном случае на терминал выводится строка сообщения. Если опция report <строка сообщения> отсутствует, выдается стандартное сообщение Assertion violation (нарушение условий проверки). После этого поведение программы моделирования определяется значением уровня важности. Уровень важности — это выражение (обычно константа) типа severity_level. Напомним, что данные этого типа могут принимать четыре значения, причем действия, которые выполняются при значении булевского выражения false задаются экспериментатором как опции симулятора. По умолчанию (т. е. при отсутствии в тексте указания важности) подразумевается уровень error.

8 Параллельные операторы

Параллельные операторы это такие, каждый из которых выполняется при любом изменении сигналов, используемых в качестве его исходных данных. Результаты исполнения оператора доступны для других параллельных операторов не ранее, чем будут выполнены все операторы, инициализированные общим событием (а может быть и позже, если присутствуют выражения задержки). В языке VHDL к классу параллельных операторов относятся:

```
<параллельный оператор> ::= <оператор процесса>
| <оператор параллельного присваивания> | <параллельный вызов процедуры> |
<параллельный оператор проверки> | <оператор блока> | <оператор вхождения
компонента> | <оператор генерации>
```

Оператор процесса уже рассматривался в предыдущих разделах. Здесь важно отметить, что этот оператор определен именно как **составной оператор параллельного типа**. Под составным оператором понимается оператор, имеющий тело, которое содержит несколько вложенных операторов. Оператор процесса

начинает исполняться при изменении сигналов, входящих в список чувствительности (при отсутствии такого списка — безусловно после выполнения всех вложенных операторов), а результаты его исполнения доступны другим параллельным операторам только после исполнения всех операторов, инициируемых теми же событиями, в том числе процессов.

8.1 Параллельное присваивание

Параллельное присваивание определено в трех различных формах:

<параллельное присваивание> ::= [<метка>:]<безусловное параллельное присваивание> | [<метка>:]<условное присваивание> | [<метка>:]<присваивание по выбору>

По синтаксису и правилам исполнения **безусловное параллельное присваивание** совпадает с последовательным присваиванием сигналу. Варианты различаются по локализации в программе и характеризуются различными условиями исполнения. Допускается введение ключевого слова guarded перед правой частью оператора присваивания сигналу, этот вопрос будет рассмотрен ниже.

Выделим наиболее существенные различия безусловного параллельного присваивания и последовательного присваивания сигналу:

- параллельное присваивание локализуется в общем разделе архитектурного тела, а последовательное — только в теле процесса;
- последовательное присваивание сигналу выполняется после того, как инициировано исполнение процесса и выполнены все предшествующие операторы в теле процесса;
- оператор параллельного присваивания выполняется сразу (с точки зрения модельного времени) после изменения сигналов в правой части этого оператора.

В обоих случаях результаты присвоения сначала фиксируются в драйвере сигнала и пе-редаются сигналу, т. е. могут влиять на другие операторы, только после исполнения всех операторов и процессов, инициированных одним событием, или через интервал модельного времени, заданный опцией after.

Условное присваивание и **присваивание по выбору**. Эти операторы во многом сходны с условным оператором и оператором выбора, соответственно — описанные действия вы-полняются при определенных условиях. Различие, кроме единых для всех параллельных и последовательных операторов свойств, состоит в том, что условный оператор и оператор вы-бора являются составными, т. е. в них условие может задавать исполнение последовательно-сти действий, а в операторах присваивания возможно только присвоение одного значения.

<условное присваивание> ::=
<приёмник> <= [guarded] [<модель задержки>]
{<прогноз поведения> when <условие> else} <прогноз поведения>;
<присваивание по выбору> ::=
with <ключевое выражение> select <приемник> <= [guarded] [< модель задержки>]

«<прогноз поведения> when <вариант>,» <прогноз поведения> when <вариант>;

Смысл и синтаксис конструкции **вариант** точно совпадает с соответствующим элементом оператора выбора.

Важно отметить, что если условный оператор if и оператор выбора case не могут выполняться над данными, вырабатываемыми модулями, представленными различными операторами процесса, то условное присваивание и присваивание по выбору позволяют описывать такие ситуации.

Параллельные операторы проверки и **вызова подпрограмм** соотносятся с соответствующими последовательными операторами проверки и вызова подобно соотношению параллельного и последовательного присваивания, а именно: они имеют одинаковый синтаксис и правила выполнения, но различаются локализацией и условиями запуска к исполнению.

8.2 Оператор блока

Оператор блока block, подобно оператору process, является составным оператором, тело которого включает несколько операторов, но, в данном случае, параллельных. Операторы тела блока, как и другие параллельные операторы, обеспечивают возможность представления параллелизма в моделируемой системе. Эти операторы инициируются не по последовательному, а по событийному принципу, а результаты их исполнения становятся доступны другим операторам как включенным в блок, так и размещенным в других блоках или по отдельности, только после исполнения всех операторов, инициированных одним событием. В этом смысле операторы, включенные в блок, не отличаются от обычных параллельных операторов.

Объединение операторов в блоки обеспечивает следующие возможности:

- структуризация текста описания, т. е. возможность явного и наглядного выделения совокупности операторов, описывающих законченный функциональный узел;
- возможность объявления в блоке локальных типов, сигналов, подпрограмм и некоторых других локальных понятий;
- возможность приписывания всем или некоторым операторам блока общих условий инициализации.

Упрощенные правила записи оператора блока определены таким образом:

```
<оператор блока> ::=
<метка блока>: block [(охранное выражение)] [is] [<раздел деклараций блока>]
begin
<раздел операторов блока> end block [<метка блока>];
```

Наиболее специфическими аспектами блочной организации являются понятия охранного выражения и охраняемого оператора присваивания.

Охранное выражение — это любое выражение логического типа, аргументами которого являются сигналы. Любое изменение сигналов, входящих в охранное выражение, вызывает вычисление значения этого выражения и присвоение полученного значения предопределенной переменной guard. Область действия этой

переменной — все тело блока, и она может использоваться как обычная логическая переменная во вложенных операторах блока. Например, узел выборки данных из тридцатидвухразрядного регистра на восьмиразрядную линию, в котором транслируется байт, указанный двухразрядным кодом номера byte_sel, может быть представлен таким блоком:

```
select_byte: block (select='1' and read='1')
is begin
dbus <= reg(7 downto 0) when guard and byte_sel="00" else
      reg(15 downto 0) when guard and byte_sel="01" else
      reg{23 downto 16) when guard and byte_sel ="10"
      else
      reg(31 downto 24) when guard and byte_sel ="11" else
"zzzzzzzz"; end block select_byte;
```

Охраняемый оператор присваивания использует значение переменной guard без явного указания условия в программе. Если guard ='0', то исполнение операторов присваивания, содержащих ключевое слово guarded, в таком блоке запрещено.

9 Разрешение сигналов и шины

В предыдущих разделах мы обходили рассмотрение случаев, когда несколько источников подключаются к одной линии. Если сигнал имеет один драйвер, то его значение определяется достаточно просто. После исполнения или перехода в состояние ожидания всех процессов и параллельных операторов, вызванных общим событием, предсказанные изменения передаются из драйверов сигналов, являющихся, в сущности, программными буферами, в поле данных системы моделирования. Это и определяет новое значение сигнала.

Однако во многих системах **к одной линии подключено несколько источников**. Например, шина данных компьютера может принимать сигналы от процессора, памяти, периферийных устройств, а к линии данных матрицы запоминающего устройства подключается достаточно много ячеек памяти, а также буферы ввода-вывода. Этому соответствуют программные модели, в которых один сигнал назначается в нескольких параллельных операторах и процессах. В VHDL не любые сигналы способны принимать значение в соответствии со значениями нескольких источников. Сигналы, значения которых автоматически определяются исходя из состояний нескольких источников (драйверов), называют **разрешаемыми** (resolved). Разрешаемым может быть объявлен конкретный сигнал или **подтип данных**, к которому такой сигнал отнесен при его декларации. Мы ограничимся объявлением сигнала как разрешаемого (resolved) через использование декларации разрешаемого подтипа. Напомним, что декларация подтипа может содержать имя **функции разрешения** (resolution function). Функция разрешения определяет правило вычисления сигнала, формируемого несколькими независимыми источниками. В общем случае функция разрешения зависит от конкретных условий приложения проекта, в том числе технологии изготовления проектируемого устройства, и если в системе интерпретации отсутствует подходящий аналог, требуется разработка

соответствующей программы.

Функция разрешения локализуется в том же программном модуле, что и декларация подтипа, и вызывается всякий раз, когда меняет состояние любой из драйверов разрешаемого сигнала. Можно сказать, что по умолчанию предполагается наличие в программном модуле параллельного вызова этой функции, причем драйверы сигналов являются ее фактическими параметрами, а возвращаемое значение присваивается сигналу.

Рассмотрим в качестве примера так называемое **монтажное ИЛИ**. В качестве базового типа данных выберем bit. Введём функцию разрешения wired_OR:

```
function wired_OR (inputs : in bit_vector) return bit is
variable X : bit := '0';
begin
      for i in inputs'range loop
            if inputs(i)='1'
            then
                  X:='1';
                  exit;
            end if;
      end loop;
      return X;
end;
```

Пусть эта функция определена в некотором архитектурном теле или пакете. Там же вводим подтип:

```
subtype  WO  is  wired_OR  bit;
```

Объявим сигнал и в разделе операторов объединим выходы элементов (в данном случае — 2-х элементов И):

```
signal  Y  :  WO;
Y <= X1 and X2; Y <= X3 and X4;
```

Драйверы сигнала inputs неявно рассматриваются как битовый массив, границы которого определяются стандартным атрибутом range.

На практике обычно используется девятизначный алфавит std_logic или std_ulogic. Первый отличается тем, что для него заранее предусмотрена функция разрешения и он фактически является подтипом для std_ulogic. В случае, если предлагаемая функция разрешения не устраивает разработчика, он может определить её самостоятельно и создать свой подтип на основе std_ulogic.

10 Подпрограммы

Подпрограммы в VHDL, как и в других алгоритмических языках, обеспечивают, во-первых, структуризацию описания проекта за счет разделения его на законченные внутренне определенные блоки, а во-вторых, являются средством экономии времени проектировщика, позволяя заменить несколько описаний сходных фрагментов алгоритма одним объявлением подпрограммы и соответствующими ссылками на нее (вызовами) в основном тексте.

Каждая подпрограмма, встречающаяся в проектном модуле, должна быть представлена телом подпрограммы в разделе деклараций этого модуля или

проектного модуля, иерархически старшего по отношению к данному.

Различают два вида подпрограмм: процедуры (procedure) и функции (function).

Оба вида содержат в своем теле **набор последовательных операторов**, которые задают совокупность действий, исполняемых после вызова этой подпрограммы. Процедура возвращает результаты либо путем непосредственного преобразования объектов, определенных в вызывающей программе (глобальных сигналов или переменных), либо за счет сопоставления объектов через список соответствий. Функция же определяет единственное значение, используемое в выражениях, в которые включен вызов этой функции.

Объявления подпрограмм отображаются в текстах телами подпрограмм, которые подчиняются следующему синтаксическому правилу:

```
<тело подпрограммы> ::= <спецификация подпрограммы> is <раздел деклараций подпрограммы> begin
{<последовательный оператор>}
end [procedure | function] <имя подпрограммы>;
<спецификация подпрограммы> ::=
procedure <имя подпрограммы> [(интерфейсный список>)]
| function <имя подпрограммы> [(интерфейсный список>)] return <тип>
<интерфейсный список> ::= <элемент интерфейсного списка> {;<элемент интерфейсного списка>}
<элемент интерфейсного списка> ::=
[constant | variable | signal] <формальный параметр>
{,<формальный параметр>} :<направление> <тип> [:=<константное выражение>]
```

Спецификация подпрограммы определяет ее **интерфейс** (имя, входные и выходные данные). Формальный параметр следует понимать как имя, присваиваемое на время исполнения подпрограммы фактическому параметру, т. е. объекту, сопоставленному этому формальному параметру в списке соответствий оператора вызова. Входные данные подпрограммы специфицируются направлением in, выходные — направлением out, а данные, которые воспринимаются подпрограммой и возвращаются в вызывающую программу измененными, — как inout. Указание категории элемента списка (constant, variable или signal) обеспечивает контроль корректности использования подпрограммы. По умолчанию определено, что сопоставляемый объект относится к категории variable. Несоответствие типа или категории фактического или формального параметра является ошибкой. Необязательное константное выражение, завершающее представление элемента интерфейсного списка, допустимо только для параметров категории variable. Оно определяет значение по умолчанию, т. е. принимаемое соответствующей переменной, если при каких-либо условиях вызова подпрограммы присвоение иного значения не предусматривается.

В разделе деклараций подпрограммы могут определяться **локальные**, т. е. определенные только в теле подпрограммы, объекты: вложенные подпрограммы, типы и подтипы данных, переменные, константы, атрибуты. Раздел операторов содержит только последовательные операторы.

Вызов подпрограмм подчиняется единому для процедур и функций синтаксическому правилу:

```
<вызов подпрограммы> ::= <имя подпрограммы> [<список соответствий>]
```

<список соответствий> **::=** ([<Формальный параметр>] =><фактический параметр>
{, [«Формальный параметр>] => <фактический параметр>)})
<фактический параметр> **::=** <выражение> | <константа>
| <имя сигнала> | <имя переменной> | <вызов функции>

Вызов процедуры записывается в программе как отдельный оператор, а вызов функции используется в выражениях того же типа, что и тип возвращаемого параметра, как обычная переменная.

Предусмотрено две формы списка соответствий. Форма, в которой каждый элемент списка содержит формальный параметр и сопоставляемый ему фактический параметр, разделенные знаком =>, называется **сопоставлением по имени**. Альтернативная форма содержит только фактические параметры и называется **позиционным сопоставлением**. Позиционное сопоставление требует точного совпадения порядка записи фактических параметров в списках соответствия и порядка записи формальных параметров в интерфейсном списке подпрограммы. Если какой-либо параметр не используется или принимается значение входа, определенное по умолчанию, соответствующая позиция в списке отмечается как пустая. При сопоставлении по имени порядок записи не имеет значения, важно лишь совпадение имени формального параметра с именем, указанным в декларации подпрограммы.

Язык VHDL, в отличие от традиционных языков программирования, различает **последовательный** и **параллельный вызов подпрограммы**. Синтаксически они одинаковы, но различна их локализация и правила исполнения. Вызов функции трактуется как параллельный, если входит в параллельный оператор, чаще всего — в оператор параллельного присваивания, и как последовательный, если входит в последовательный оператор.

Оператор вызова процедуры является последовательным, если локализован в теле процесса или теле другой подпрограммы. В иных случаях оператор вызова подпрограммы интерпретируется как параллельный оператор. Одна и та же подпрограмма может вызываться как параллельным, так и последовательным оператором. Как и другие последовательные операторы, оператор последовательного вызова выполняется после исполнения всех операторов, предшествующих ему в теле процесса или теле подпрограммы. Параллельный оператор вызова исполняется после изменения любого из сигналов, перечисленных в списке соответствий этого оператора. Иными словами, параллельный вызов процедуры эквивалентен процессу, тело которого совпадает с телом процедуры с точностью до обозначений, а список чувствительности содержит входные фактические параметры оператора вызова.

Большое значение в подпрограммах имеет **оператор возврата** return. В процедуре этот оператор прекращает ее исполнение, передавая управление вызывающей программе. Если исполнен оператор return в процедуре, вызванной последовательным оператором, то после него выполняется оператор вызывающей программы, следующий за оператором вызова. После исполнения оператора return в

процедуре, вызванной параллельным оператором, интерпретатор программы обращается к календарю событий и инициирует исполнение оператора, связанного со следующим событием в календаре. При отсутствии оператора возврата исполнение процедуры завершается последним оператором в порядке записи.

Оператор возврата в теле функции обязателен. Он также прекращает исполнение подпрограммы, но, кроме того, выполняет присвоение значения результату, который подставляется в вызывающей программе на месте вызова функции.

11 Структурное представление проекта

В предыдущих разделах рассмотрено несколько языковых понятий, служащих структуризации описания проекта: понятия процесса, блока, подпрограммы.

Язык VHDL предоставляет еще одну возможность структуризации описания — так называемые **структурные архитектурные тела** (Structural Architectural Body, SAB). Структурное архитектурное тело представляет проект в виде набора компонентов и их связей, т. е. приближает описание к реальной конфигурации проектируемого устройства, как минимум, к представлению разработчика об этой конфигурации. Все используемые в проекте компоненты должны иметь свое entity и однозначно заданное поведение, например поведенческое архитектурное тело, которое следует заранее скомпилировать в библиотеку проекта или иную библиотеку, объявленную перед entity составного проекта.

SAB представляется наиболее эффективным средством создания **иерархических проектов**. Применение концепции SAB облегчает совместную работу нескольких исполнителей над одним проектом. Упрощается включение в новые проекты ранее созданных модулей, а также использование стандартных библиотек проектных модулей.

По форме структурные и поведенческие архитектурные тела не различаются. Различие состоит в составе включенных операторов и деклараций.

В разделе объявлений SAB, кроме характерных для любых архитектурных тел деклараций, таких как декларации типов, сигналов, констант, включаются специфические подразделы: обязательный подраздел **декларации прототипов используемых компонентов** и необязательный подраздел **объявления конфигураций**. Раздел операторов SAB содержит **операторы вхождения компонентов**, которые, собственно, и определяют порядок соединения компонентов. Могут включаться и другие параллельные операторы, а если таких операторов относительно много, подобное архитектурное тело называют **смешанным**, или структурно-поведенческим.

Каждому модулю, входящему в качестве структурного компонента в SAB, должно сопутствовать объявление прототипа. Синтаксис декларации прототипа компонента имеет вид:

<декларация прототипа компонента> ::= component <имя entity компонента> is [<образ настройки>] <образ порта>

```
end component [<имя entity компонента>];
```

Здесь <образ настройки> и <образ порта> — прямая копия высказываний generic и port из текста объявления entity соответствующего компонента.

Если в устройстве есть несколько одинаковых модулей, то прототип декларируется только один раз. Например, если в проект входит несколько однотипных регистров, представленных в библиотеке одним и тем же entity, то в структурном теле размещается единственная декларация прототипа регистра, даже если конкретные экземпляры имеют различные параметры. Тем не менее, каждому экземпляру, включаемому в проект, называемому также **вхождением модуля** (instance), при структурном описании присваивается собственное имя. Это собственное имя предъявляется в разделе операторов архитектурного тела в виде метки оператора вхождения компонента.

В подразделе объявления сигналов обязательно объявление всех соединений между блоками. Каждый сигнал относят к типу, определенному в разделе port соответствующего модуля.

Основными операторами SAB являются **операторы вхождения компонентов** (Component Instantiation Statement), которые определяют порядок соединения включаемых в проект модулей и, возможно, их параметры.

```
<оператор вхождения компонента> ::=
<метка вхождения>: [component] <имя компонента>
[generic map (<список соответствий параметров настройки>)] port map (<список соответствий порта>);
<список соответствий> ::= ([<формальный параметр>] =><фактический параметр>
{,[<формальный параметр>] => <фактический параметр>})
```

По форме списки соответствий настроек и порта совпадают со списком соответствий вызова подпрограмм. Но для параметров настройки фактическим параметром может быть только константное выражение, а для порта — только имя сигнала либо имя порта главного модуля. Нельзя объявлять в качестве фактического параметра в списке соответствий порта имя входа или выхода другого компонента. **Все связи осуществляются только через** объекты, объявленные в текущем архитектурном теле как **сигналы**. (В принципе, объявление сигналов может осуществляться в общедоступном модуле package.)

Подобно вызову подпрограмм возможны полная и сокращенная формы записи списка соответствий. В сокращенной записи (без формальных параметров) все элементы списков соответствий записываются в том же порядке, как в списках параметров и сигналов порта данного компонента. В полной форме списка соответствий порядок записи элементов в списке произволен.

Каждый компонент в архитектурном теле представлен своим оператором вхождения, метка которого определяет его имя в проекте, вслед за чем записано имя прототипа и списки соответствий.

Оператор вхождения можно трактовать как вызов процедуры со специальным, наглядным синтаксисом. Но есть и более существенные отличия:

- Вызов подпрограммы инициирует исполнение тела подпрограммы, являющегося набором последовательных операторов. Оператор

вхождения вызывает к исполнению архитектурное тело, которое содержит параллельные операторы, и сам является параллельным оператором, исполняемым при каждом изменении его входных сигналов.

- Внутренние переменные и сигналы встраиваемого модуля определяются как статические (в отличие от переменных подпрограмм), т. е. сохраняют свои значения между инициализациями.

12 Настройка и конфигурирование компонентов

Очень часто устройства проектируются не только как изделия с наперед заданными свойствами, но и для возможности их применения в различных приложениях, требующих однотипных преобразований. Соответственно, текстовое описание желательно создавать в формах, допускающих модификацию в определенных пределах свойств представляемых объектов проектирования. Есть два основных пути создания программ, описывающих множество модулей с идентичными функциями, иначе — перестраиваемых модулей:

- использование параметров настройки (generic);
- разработка нескольких архитектурных тел, подчиненных общему entity, иными словами, имеющих одинаковую алгоритмическую сущность при различии способа описания или способа реализации.

Модуль, содержащий декларацию параметров настройки (generic), называют **параметризованным**. Фактическое значение задается в списке соответствий оператора вхождения.

Параметры, определяющие количественные свойства реализаций (например, разрядность данных, время задержки), в выражениях внутри подчиненных архитектурных тел являются обычными константами. Но, кроме того, часто применяются параметры структурного характера, уточняющие функции, реализуемые конкретными вхождениями параметризованного модуля и/или его структуру. Чаще всего такие параметры объявляются в операторах генерации, синтаксис которых определен как:

```
<оператор генерации> ::=
<метка оператора генерации>: <схема генерации> generate <параллельный оператор>
{<параллельный оператор>}
end  generate  [<метка оператора генерации>];
<схема генерации> ::=
if <булевское выражение> | for <имя> in <диапазон>
```

Схема генерации с ключевым словом if разрешает (или блокирует) создание компонентов, представленных вложенными параллельными операторами. Схема с ключевым словом for определяет число компонентов, создаваемых в структуре модуля в зависимости от значения параметра настройки.

В случаях, когда вариативность компонента достигается разработкой нескольких архитектурных тел, т. е. первичному проектному модулю (entity) этого компонента в библиотеке проекта соответствует несколько различных архитектурных тел, проектный модуль высшего уровня иерархии должен содержать

объявление конфигурации компонента. Объявление конфигурации определяет, какое именно архитектурное тело компонента используется в текущем проекте или сеансе отладки. Объявление конфигурации компонента подчиняется следующему синтаксическому правилу:

```
<конфигурация компонента> ::=
for  <спецификация компонента>  use  <индикатор привязки>;
<список вхождений> ::= <метка вхождения> {,<метка вхождения>} | others
|  all
<индикатор привязки> ::=
entity  <имя сущности>  (<имя архитектурного тела>)
<спецификация компонента> ::=  <список вхождений>  :  <имя компонента>
```

13 Пакеты

Мы уже неоднократно обращались к понятию пакета (package) в VHDL. Пакет — это программный модуль, содержащий описание объектов, доступных нескольким другим программным модулям. В пакете могут быть объявлены глобальные типы, константы, функции, сигналы и т. п. Фактически, общие для нескольких подпроектов типы и сигналы можно объявить только в пакете.

Рассмотрим более формально правила записи пакетов.

Пакет представляется двумя структурными единицами — обязательным первичным модулем **декларации пакета** (package declaration) и необязательным вторичным модулем **тела пакета** (package body).

```
<декларация пакета> ::= package <имя пакета> is <раздел деклараций пакета> end
[package] [<имя пакета>];
```

Раздел деклараций пакета может содержать спецификации подпрограмм, декларации типов и подтипов, констант, сигналов, атрибутов, компонентов и ряда других объектов.

Тело пакета содержит конкретизацию способов вычисления функций и записывается в соответствии со следующим синтаксическим правилом:

```
<тело пакета> ::=
package body <имя пакета> is <раздел деклараций пакета>
end  [package  body][<имя пакета>];
```

Раздел деклараций тела пакета содержит тела подпрограмм (спецификация этих подпрограмм обязательно присутствует в разделе деклараций пакета), а также дополнительные декларации объектов, используемых в представленных подпрограммах. Могут декларироваться типы, константы, вложенные подпрограммы, объекты некоторых иных классов. Эти объекты недоступны для других проектных модулей.

Литература

1. Е.П. Угрюмов. Цифровая схемотехника. СПб.: БХВ-Петербург, 2005.
2. Р.И. Грушвицкий, А.Х. Мурсаев, Е.П. Угрюмов. Проектирование систем на микросхемах с программируемой структурой. СПб.: БХВ-Петербург, 2006.
3. А.К. Поляков. Языки VHDL и VERILOG в проектировании цифровой аппаратуры. М.: СОЛОН-Пресс, 2003.
4. П.Н. Бибило. Системы проектирования интегральных схем на основе

языка VHDL. М.: СОЛОН-Пресс, 2005.
5. В.Ю. Зотов. Проектирование цифровых устройств на базе ПЛИС фирмы XILINX в САПР WebPACK ISE.
6. Л.З. Бобровников. Электроника. СПб.: Питер, 2004.
7. А.К. Нарышкин. Цифровые устройства и микропроцессоры. М.: ACADEMIA, 2006.
8. И.М. Мышляева. Цифровая схемотехника. М.: ACADEMIA, 2005.
9. Берчун Ю.В. Язык описания электронной аппаратуры VHDL: Методические указания. - М.: МГТУ им. Н.Э. Баумана, 2006. - 46 с.

Приложение 1. Алфавит моделирования

Важной характеристикой метода моделирования цифровых устройств является количество различимых состояний сигнала. Каждому состоянию сопоставляется индивидуальный символ, совокупность символов составляет **алфавит моделирования**. Естественно, каждое состояние специфически воспринимается приемниками сигналов, поэтому в системе моделирования определяется набор правил преобразования сигналов типовыми цифровыми элементами.

Простейший алфавит — двоичный, содержащий набор {'0', '1'}. Функционирование элементов описывается по правилам алгебры логики. Моделирование на базе этого алфавита весьма экономично, но его возможности ограничены. Невозможно описание шинной логики, в том числе схем, имеющих высокоимпедансное состояние на выходе (Z-состояние), схем с открытым коллектором и подобных. Затруднено воспроизведение сбойных ситуаций, например, вызванных подачей управляющих сигналов на триггеры во время, когда информационные сигналы еще не установлены.

Весьма распространен **алфавит из четырех символов** {'O', 'X', '1' 'Z'}.

Здесь 'X' означает неопределенное состояние. Такой символ присваивается, в частности, сигналу на выходе логического элемента во время переходного процесса. Например, неопределенное состояние принимает выход триггера после подачи активизирующего сигнала на синхронизирующий вход при запрещенной или неопределенной комбинации сигналов на его информационных входах. Символ 'Z' представляет высокоимпедансное состояние порта или отключенную линию.

Дальнейшее расширение возможностей — **девятиэлементный алфавит**, в котором приняты следующие символы для представления состояний связей:

- 'U' — не инициализировано (сигналу в программе вообще не присваивались другие значения; обеспечивает контроль корректности инициализации);
- 'Z' — отключено (все источники, подключенные к связи в высокоимпедансном состоянии);
- 'X' —активное неопределенное состояние;
- '0' —активный ноль;
- '1' —активная единица;
- 'L' —слабый ноль;
- 'H' — слабая единица;
- 'W' — слабое неопределенное состояние;
- '–' — не важно (разработчик может запрограммировать переход в это состояние, если реализация алгоритма не зависит от результата; выбор конкретного значения предоставляется компилятору с целью оптимизации реализации устройства).

Разница между слабыми и активными состояниями состоит в том, что слабый

сигнал формируется от источников (называемых драйверами), имеющих повышенное выходное сопротивление по сравнению с активными источниками. В этом случае источник, генерирующий активный сигнал, подавляет слабый, если не отключен. Пример элемента, генерирующего слабую единицу, — буфер с открытым коллектором. На выходе у него может быть активный ноль, но слабая единица.

При записи программ в VHDL пользователь может априорно задать алфавит моделирования тех или иных языковых конструкций, определяя тип сигналов — от простого двоичного, задаваемого как тип bit (битовый), до девятикомпонентного типа std_logic. В принципе, пользователь может создавать свои типы с большим или меньшим числом символов для представления логических данных, или, что то же самое, — числом воспроизводимых в модели состояний сигналов.

При выборе алфавита (если это допускает система моделирования) следует учитывать, что расширенный алфавит, обеспечивая во многих случаях большую адекватность моделирования, требует больших затрат машинного времени на проведение сеансов моделирования.

Задания на лабораторные работы

Лабораторная работа №1. Программирование сопроцессора

Ввести с клавиатуры границы интервала значений аргумента функции, перевести их в формат вещественного числа, рассчитать шаг и вычислить значение функции в заданном интервале. Вывести на экран график функции, таким образом, чтобы он весь умещался на экране, для чего произвести масштабирование вычисленных значений функции в соответствии с размерами экрана в выбранном графическом режиме.

$Y = \sin^2 x + \cos^4 x$.

$Y = (\cos^2 x + \sin x)/2$.

$Y = \sin^2 x + \cos^3 x$.

$Y = (\cos^2 x + \sin^3 x)/3$.

$Y = (x*\sin^3 x)/5$.

$Y = x*\cos^2 x * \sin^2 x$.

$Y = (x^2 * \sin x)/2$.

$Y = x^2 * \cos^2 x * \sin x$.

$Y = x^3 * \sin x * \cos x$.

$Y = (x^3 * \cos^3 x)/3$.

$Y = x^3 * \cos x * \sin^3 x$.

$Y = 3*\cos^2 x * x^4$.

$Y = 4*\sin^3 x * x$.

$Y = 2*\cos^3 x * x^2$.

$Y = x* \sin^2 x + x^2 * \cos x$.

$Y = x*\cos^2 x + x^3 * \sin^3 x$.

$Y = (\sin x + x^2)/2$.

$Y = 2*(\cos^2 x + x^2)$.

$Y = x*(\sin^2 x + x)/2$

$Y = 2x*(\cos^2 x + x^2)$

$Y = x^2 * (\cos x + x)$.

$Y = x^2 * (\sin x + x)/2$.

$Y = x^3 * (\sin x + \cos x)$.

$Y = x^3 * (\sin x + \cos x)^2$.

$Y = x^2 * (\cos x + x)^2$.

$Y = x^2 * (\sin x + x)^3$.

Лабораторная работа №2. Работа с файлами

Создать текстовый файл, который содержит фамилию и имя студента. Открыть его с использованием функций прерывания 21h и вывести содержимое файла на экран.

Лабораторная работа №3. Работа с видеопамятью

Вывести свое имя и фамилию на экран в текстовом режиме, используя прямой доступ в видеопамять (сегмент 0B800h).

Лабораторная работа №4. Вывод текстовой информации в графическом режиме

Воспользовавшись одной из доступных битовых карт шрифтов, вывести в графическом режиме на экран свое имя и фамилию.

Лабораторная работа №5. Резидентные программы

Перехватив прерывание от клавиатуры, поменять между собой вывод двух произвольных букв в верхнем и нижнем регистре.

Лабораторная работа №6. Разработка моделей устройств на языке VHDL

Используя язык описания VHDL, спроектировать схему в соответствии с вариантом:

1. Одноразрядный сумматор на элементах И-НЕ.
2. Схема выделения старшей единицы.
3. Мультиплексор 4-1 на элементах И-НЕ.
4. Мультиплексор 4-1 на элементах И-НЕ с инверсным выходом.
5. Мультиплексор 4-1 на элементах И-НЕ с парафазным выходом.
6. Мультиплексор 4-1 на элементах ИЛИ-НЕ с инверсным выходом.
7. Одноразрядный сумматор на элементах ИЛИ-НЕ.
8. Схема сравнения двухразрядных кодов на равенство на элементах И-НЕ.
9. Схема свертки четырехразрядного кода для проверки на четность на элементах И-НЕ.
10. Схема сравнения двухразрядных кодов на равенство на элементах ИЛИ-НЕ.
11. Схема свертки четырехразрядного кода для проверки на четность на элементах ИЛИ-НЕ.

Вопросы по курсу «ЭВМ и ПУ»

1. Введение
2. Вычислительная машина и вычислительная система
3. Архитектура и уровни детализации
4. Концепция машины с хранимой в памяти программой
5. Принцип двоичного кодирования. Принцип программного управления. Принцип однородности памяти. Принцип адресности
6. Фон-неймановская архитектура
7. Структуры вычислительных машин. Структуры вычислительных систем
8. Архитектура системы команд. Классификация архитектур системы команд
9. Классификация по составу и сложности команд. Классификация по месту хранения операндов
10. Типы и форматы операндов. Числовая информация. Числа в форме с фиксированной запятой. Упакованные целые числа. Десятичные числа
11. Числа в форме с плавающей запятой. Упакованные числа с плавающей запятой
12. Символьная информация. Логические данные. Строки
13. Типы команд. Команды пересылки данных. Команды арифметической и логической обработки. SIMD-команды. Команды для работы со строками. Команды преобразования
14. Команды ввода/вывода. Команды управления системой. Команды управления потоком команд
15. Форматы команд. Длина команды
16. Непосредственная адресация. Прямая адресация. Косвенная адресация. Регистровая адресация. Косвенная регистровая адресация.
17. Относительная адресация. Базовая регистровая адресация. Индексная адресация. Страничная адресация. Блочная адресация
18. Система операций
19. Функциональная схема фон-неймановской ВМ
20. Счетчик команд. Регистр команды. Указатель стека
21. Регистр адреса памяти. Регистр данных памяти. Дешифратор кода операции
22. Микропрограммный автомат. Арифметико-логическое устройство. Операционный блок. Регистры операндов. Регистр признаков. Аккумулятор
23. Основная память
24. Модуль ввода/вывода. Порты ввода и порты вывода
25. Цикл команды. Этап выборки команды. Этап формирования адреса следующей команды
26. Этап декодирования команды. Этап вычисления адресов операндов. Этап выборки операндов. Этап исполнения операции. Этап записи результата
27. Машинный цикл с прерыванием
28. Основные показатели вычислительных машин
29. Организация шин
30. Память. Характеристики систем памяти
31. Иерархия запоминающих устройств
32. Основная память. Блочная организация основной памяти. Расслоение памяти
33. Организация микросхем памяти
34. Последовательный режим. Конвейерный режим. Регистровый режим
35. Страничный режим. Режим быстрого страничного доступа. Пакетный режим. Режим удвоенной скорости
36. Функции центрального устройства управления
37. Модель устройства управления
38. Структура устройства управления
39. Микропрограммный автомат с жесткой логикой
40. Микропрограммный автомат с программируемой логикой

41. Принцип управления по хранимой в памяти микропрограмме
42. Операционные устройства вычислительных машин. Структуры операционных устройств
43. Операционные устройства с жесткой структурой
44. Операционные устройства с магистральной структурой
45. Классификация операционных устройств с магистральной структурой
46. Организация узла РОН магистрального операционного устройства. Организация операционного блока магистрального операционного устройства
47. Языки описания электронной аппаратуры
48. Уровни описания электронной аппаратуры
49. История развития HDL
50. Варианты использования HDL. Преимущества HDL
51. HDL с точки зрения схемотехника
52. HDL с точки зрения программиста
53. Шины и слоты
54. Разъемы для видеокарт, мониторов.
55. Порты для устройств ввода. Разъем для локальных сетей
56. Разъемы для жестких дисков
57. Основные характеристики и классификация ВЗУ
58. Физические основы магнитной записи
59. Физические основы оптической записи информации
60. Организация накопителей на оптической основе
61. Методы регистрации текстовой информации
62. Устройства автоматического ввода текстовой информации
63. Организация сканеров
64. Устройство клавиатуры
65. Дисплеи
66. Звуковая информация

Контрольный тест №1

1. Какие способы построения ВМ существуют?

а) С непосредственными связями

б) Иерархический

в) С общей шиной

г) С общим чипсетом

2. Выберите корректные типы команд

а) Команды пересылки данных

б) Команды управления данными

в) Команды управления потоками данных

г) Команды управления потоком команд

3. Что не является основным свойством алгоритма?

а) Массовость

б) Дискретность

в) Успешность

г) Результативность

д) Определенность

4. Какое определение счетчика команд является самым удачным?

а) Счетчик команд

б) Программный счетчик

в) Программный указатель

г) Указатель команды

5. Как называется АСК с полным набором команд?

а) RISC

б) DISC

в) CISC

г) PISC

6. В каком виде организованы регистры данных в сопроцессоре?

а) Нумерованный список

б) Именованный список

в) Стек

г) Область с произвольным доступом

7. Как физически реализуются линии шины?

а) Витая пара

б) Медные проводящие дорожки на кристалле микросхемы

в) Полоски проводящего материала на монтажной плате

г) Жгут проводов в индивидуальной изоляции

8. В каком методе доступа к данным можно продолжать операцию чтения по предыдущему адресу в процессе запроса по следующему?

а) Регистровый

б) Конвейерный

в) Последовательный

г) Удвоенной скорости

9. Методы доступа к данным:

а) Непосредственный

б) Последовательный

в) Прямой

г) Произвольный

д) Адаптивный

е) Ассоциативный

10. В зависимости от способа формирования микрокоманд различают микропрограммные автоматы:

а) С жесткой или аппаратной логикой

б) Со смешанной логикой

в) С программируемой логикой

г) С вариативной логикой

11. Операционный узел устройства управления является частью

а) Адресной части УУ

б) Управляющей части УУ

в) Исполнительной части УУ

12. Какие виды ОУ существуют?

а) С жесткой структурой

б) С программируемой структурой

в) С магистральной структурой

13. К основным достоинствам HDL следует отнести следующие:

а) Интерпретируемость

б) Стандартность

в) Многоаспектность и иерархичность

г) Пригодность для восприятия человеком и обработки на ЭВМ

д) Переносимость

14. Выберите параллельные интерфейсы:

а) PCI

б) USB

в) SATA

г) SCSI

15. Из каких частей состоит ВЗУ?

а) Носитель информации

б) Привод носителя

в) Стабилизатор

г) Порт ввода-вывода

д) Контроллер накопителя

16. Накопители на магнитных дисках являются ЗУ такого типа:

а) Последовательного

б) Произвольного

в) Смешанного

17. В каких принтерах для перенесения изображения символов на бумагу также используется электрофотографический метод?

а) Матричные

б) Струйные

в) Лазерные

г) Термические

18. В каких сканерах носитель неподвижен?

а) Планшетные

б) Рулонные

в) Проекционные

19. К какому классу ПУ относятся мышь и клавиатура?

а) Устройства ручного ввода и оперативного управления

б) Терминальные средства визуализации

в) Средства ввода-вывода речевых сообщений

20. Какова длина типа данных dw в битах?

Ответы к тесту №1

1. а), в)

2. а), г)

3. в)

4. г)

5. в)

6. в)

7. б), в)

8. б)

9. б), в), г), е)

10. а), в)

11. а)

12. а), в)

13. б), в), г)

14. а), г)

15. а), б), д)

16. в)

17. в)

18. а), в)

19. а)

20. 16

Контрольный тест №2

1. Какие структуры вычислительных систем существуют?

а) Система с общей памятью

б) Распределенная система

в) Система с общей шиной

г) Частично независимая система

2. Какие этапы цикла команды предшествуют выборке операндов?

а) Запись результата

б) Декодирование команды

в) Формирование адреса следующей команды

г) Исполнение операции

3. Что не является таблицей кодировки?

а) EBCDIC

б) ISO

в) ASCII

г) UNICODE

4. На каком этапе команда извлекается из памяти и размещается в регистре команды?

а) Выборка команды

б) Дешифрация команды

в) Запись результата

г) Выборка операндов

5. Как называется АСК с сокращенным набором команд?

а) CISC

б) RISC

в) MISC

г) PISK

6. Для чего предназначен служебный регистр SWR?

а) Хранение информации о текущем состоянии сопроцессора

б) Управление режимами работы сопроцессора

в) Контроль за состоянием регистров данных

г) Запоминание информации об адресе команды

7. Основные виды транзакций на шине

а) Транзакция чтения

б) Транзакция передачи

в) Транзакция записи

г) Транзакция ввода-вывода

8. Как разделяют ЗУ по месту расположения?

а) Внутренние

б) Процессорные

в) Шинные

г) Интерфейсные

д) Внешние

9. Основную память образуют устройства

а) С прямым доступом

б) С адаптивным доступом

в) С произвольным доступом

г) С последовательным доступом

10. Входной информацией для устройства управления служат:

а) Тактовые импульсы, код операции, флаги, признаки результата

б) Тактовые импульсы, код операции, признаки результата, сигналы из системной шины

в) Тактовые импульсы, код операции, флаги, сигналы из системной шины

11. Отличительная особенность микропрограммного автомата с программируемой логикой

а) Использование счетчика команд

б) Наличие устройства управления

в) Хранение микрокоманд в специализированном запоминающем устройстве

12. Какие типы операционных блоков ОУ вы знаете?

а) Последовательный

б) Параллельный

в) Гибридный

г) Смешанный

13. Какие уровни описания ВС вы знаете?

а) Системный уровень

б) Аналитический уровень

в) Комбинационный уровень

г) Функционально-логический уровень

14. Выберите последовательные интерфейсы:

а) USB

б) SATA

в) PCI

г) SCSI

15. Какие из нижеперечисленных оптических дисков существуют?

а) Однослойные

б) Двуслойные

в) Трехслойные

г) Многослойные

16. Оптические ЗУ делятся на:

а) Постоянные ЗУ типа CD-ROM

б) ЗУ с однократной записью типа CD-R

в) ЗУ с многократной записью

г) ЗУ с ограниченной записью

17. В каких принтерах изображение символа на бумагу наносится с помощью капелек жидкого красителя?

а) Лазерные

б) Струйные

в) Матричные

г) Термические

18. Какие фоточувствительные элементы используются в сканерах?

а) ПЗС-элементы

б) ЦМД-элементы

в) ПДИ-элементы

г) КДИ-элементы

19. Какие виды дискретизации используются при оцифровке аудиоданных?

а) Временной

б) Амплитудный

в) Частотный

г) Спектральный

20. Какова разрядность регистра ah в 32-битных процессорах?

Ответы к тесту №2

1. а), б)

2. б), в)

3. б)

4. а)

5. б)

6. а)

7. а), в), г)

8. а), б), д)

9. в)

10. в)

11. в)

12. а), б)

13. а), г)

14. а), б)

15. а), б)

16. а), б), в)

17. б)

18. а), г)

19. а), б)

20. 8

Контрольный тест №3

1. Какой подход не используется при описании эволюции ВТ?

а) Хронологический

б) Технологический

в) Классификационный

2. Принципы фон-неймановской концепции вычислительной машины

а) Двоичного кодирования

б) Программного управления

в) Изоморфности

г) Однородности памяти

д) Адресности

3. Что такое SIMD-команда?

а) Одна инструкция - один операнд

б) Одна инструкция - много данных

в) Много инструкций - один операнд

г) Много инструкций - много данных

4. Что является источником информации для ВМ?

а) Порты ввода-вывода

б) Основная память

в) Периферийные устройства

5. Какие виды АСК по месту хранения операндов существуют?

а) Стековые

б) Регистровые

в) Аккумуляторные

г) С обменом через порты вводы-вывода

д) С выделенным доступом к памяти

6. Какова разрядность регистра данных сопроцессора семейства 32-битных ЦП x86?

а) 40

б) 64

в) 80

г) 128

7. Структура взаимосвязей должна обеспечивать обмен информацией между:

а) Центральным процессором и памятью

б) Памятью и базой данных

в) Центральным процессором и модулями ввода/вывода

г) Памятью и модулями ввода/вывода

д) Базой данных и модулями ввода/вывода

8. Какие виды адресации Вы знаете?

а) Непосредственная

б) Посредственная

в) Базовая регистровая

г) Косвенная регистровая

д) Прямая

е) Относительная регистровая

ж) Абсолютная

з) Блочная

9. Свойства иерархической памяти:

а) Все данные на одном уровне могут быть также найдены на более низком уровне

б) Более быстрая память физически располагается выше более медленной

в) Более высокий уровень меньше по емкости, быстрее и имеет большую стоимость

г) Из процессора прямой доступ возможен к любому уровню памяти

10. Как называется микропрограмма на английском языке?

а) Software

б) Firmware

в) Software

11. Процесс синтеза схемы МПА с жесткой логикой разделяется на следующие этапы:

а) Выбор типа логических и запоминающих элементов

б) Кодирование состояний автомата

в) Выбор соединительных элементов

г) Синтез комбинационной схемы, формирующей выходные сигналы

12. Каких ОУ не существует в классической фон-неймановской ВМ по типу обрабатываемых данных?

а) ОУ целочисленной арифметики

б) ОУ для реализации логических операций

в) ОУ для комплексных чисел

г) ОУ десятичной арифметики

д) ОУ для чисел с плавающей запятой

13. Какие языки описания электронной аппаратуры вы знаете?

а) AHDL

б) CHDL

в) VHDL

г) Verilog

14. Какой разъем используется для локальных сетей?

а) PS/2

б) AGP

в) PATA

г) 8P8C

15. Какие способы магнитной записи на носитель вы знаете?

а) Прямой

б) Вертикальный

в) Горизонтальный

г) Диагональный

16. По способу доступа к информации ВЗУ бывают:

а) ЗУ с произвольным доступом

б) ЗУ с последовательным доступом

в) ЗУ со смешанным доступом

17. В каких принтерах используется быстрый нагрев красителя, когда минуется жидкая фаза?

а) Термосублимационные

б) Лазерные

в) Струйные

г) Термические

18. Читающие автоматы принято характеризовать:

а) Вероятностью ошибок распознавания

б) Вероятностью отказов от распознавания

в) Степенью уверенности в распознанном символе

19. Какой код формируется клавиатурой на аппаратном уровне?

а) ASCII

б) Скан-код

в) EBCDIC

г) UTF

20. Какое максимальное количество адресов используется в командах x86 процессоров?

Ответы к тесту №3

1. в)

2. а), б), г), д)

3. б)

4. в)

5. а), б), в), д)

6.в)

7. а), б), г)

8. а), в), г), д), е), з)

9. а), в)

10. б)

11. а), б), г)

12. в)

13. в), г)

14. г)

15. б), в)

16. а), б), в)

17. а)

18. а), б)

19. б)

20. 2

Контрольный тест №4

1. Основные показатели вычислительных машин

а) Производительность

б) Помехозащищенность

в) Стоимость

г) Надежность

2. Числовые типы данных

а) Целые числа

б) Вещественные числа

в) Комплексные числа

г) Адреса

3. Что такое функциональная полнота системы операций?

а) Достаточность системы операций для описания любых алгоритмов

б) Степень соответствия системы операций заданному классу алгоритмов

в) Степень соответствия функции вычислительной машины ее аппаратным средствам

4. Что является центральным узлом устройства управления ВМ?

а) Арифметико-логическое устройство

б) Микропрограммный автомат

в) Операционный блок

г) Сопроцессор

5. Какое название получила проблема того, что сложные операторы, характерные для ЯВУ, существенно отличаются от простых машинных операций?

а) Комбинаторный взрыв

б) Семантический разрыв

в) Технологический срыв

г) Понятийный обрыв

6. Сколько регистров данных в сопроцессоре семейства 32-битных ЦП x86?

а) 4

б) 5

в) 8

г) 10

7. Чего не требуется описывать, чтобы охарактеризовать шину?

а) Совокупность сигнальных линий

б) Физические, механические и электрические характеристики шины

в) Сигналы арбитража, состояния, управления и синхронизации

г) Типы портов, которым подходит данная шина

д) Протокол шины

8. Какие режимы доступа к памяти получили наибольшее распространение?

а) Последовательный

б) Параллельный

в) Регистровый

г) Страничный

д) Пакетный

е) Конвейерный

ж) Зависимый

з) Удвоенной скорости

9. Что лежит в основе расслоения памяти?

а) Принцип последовательного доступа

б) Принцип чередования адресов

в) Принцип однородности памяти

10. Микропрограммный автомат формирует следующую выходную информацию:

а) Внутренние управляющие сигналы

б) Результаты вычислений

в) Сигналы в системную шину

г) Флаги результата

11. Что не входит в состав типичного МПА?

а) Регистр адреса микрокоманды

б) Регистр микрокоманды

в) Транслятор микрокоманды

г) Дешифратор микрокоманд

12. Структурный базис ОУ включает в себя:

а) Регистры, обеспечивающие кратковременное хранение слов данных

б) Управляемые шины, предназначенные для передачи слов данных

в) Операционный узел

г) Комбинационные схемы

13. Команды, алгоритмы устройств, микрооперации относятся к аспектам

а) Поведения

б) Времени

в) Структуры

14. Какой разъем не относится к видеокартам и мониторам?

а) VGA

б) PCMCIA

в) DVI

г) HDMI

15. Какие форматы перезаписываемых DVD существуют?

а) DVD+RW

б) DVD-RW

в) DVD*RW

г) DVD:RW

16. Преимущества оптических дисков перед магнитооптическими:

а) Не требуется предварительное стирание старой информации

б) Отсутствие магнитных полей

в) Более низкая температура работы

г) Использование лазера меньшей мощности

17. Методы нанесения изображения символа на бумагу разделяются на:

а) Ударные

б) Бесконтактные

в) Слабоконтактные

г) Полноконтактные

18. Читающий автомат должен выполнять следующие функции:

а) Осмотр и восприятие изображения

б) Выделение существенных признаков из первичного описания

в) Формирование основных структурных блоков изображения

г) Распознавание символа

19. По типу применяемых индикаторов дисплеи делятся на следующие классы:

а) На основе электронно-лучевых трубок

б) На основе приборов с зарядовой связью

в) На основе поляризационных ячеек

г) На основе плоских матричных экранов

20. Какое число нужно поместить в регистр cx, чтобы зациклить выполнение операций 10 раз с помощью команды loop?

Ответы к тесту №4

1. а), в), г)

2. а), б)

3. а)

4. б)

5. б)

6. в)

7. г)

8. а), в), г), д), е), з)

9. б)

10. а), в)

11. г)

12. а), б), г)

13. а)

14. б)

15. а), б)

16. а), б), г)

17. а), б), в)

18. а), б), г)

19. а), г)

20. 10

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(СГАУ)

СОГЛАСОВАНО

Управление образовательных программ

_____/ А.В. Дорошин /

"_____"_____ 20___ г.

УТВЕРЖДАЮ

Проректор по учебной работе

_____/ Ф.В. Гречников /

"____" _____ 20___ г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Наименование модуля (дисциплины)

ЭВМ и периферийные устройства

Цикл, в рамках которого происходит освоение модуля (дисциплины)	Общепрофессиональные дисциплины
Часть цикла	Базовая часть
Код учебного плана	230100.2.62-2011-О-П-4г00м
Факультет	Информатики
Кафедра	Инф. систем и технологий

Курс	3		
Семестр	6		
Лекции (СЛ)	36		
Семинарские и практические занятия (СП)	0		
Лабораторные занятия (СЛР)	54	Экзамен	6
Контроль самостоятельной работы / Индивидуальные занятия (КСР / ИЗ)	0	Зачет	
Самостоятельная работа (СРС)	54	Согласовано: 110 FAQ	
Всего (Всего с экзаменами)	144		

Наименование стандарта, на основании которого составлена рабочая программа:

230100 "Информатика и вычислительная техника"

Соответствие содержания рабочей программы, условий ее реализации, материально-технической и учебно-методической обеспеченности учебного процесса по дисциплине всем требованиям государственных стандартов подтверждаем.

Составители:

Лёзин Илья Александрович, асс., к.т.н. _____
(подпись)

Заведующий кафедрой:

Прохоров Сергей Антонович, проф., д.т.н. _____
(подпись)

Рабочая программа обсуждена на заседании кафедры

Инф. систем и технологий

Протокол №___ от "____" _____ 20___ г.

Наличие основной литературы в фондах научно-технической библиотеки (НТБ) подтверждаем:

Директор НТБ _____ /_____/
(подпись) (расшифровка подписи)

Согласовано:

Декан _____ /_____/
(подпись) (расшифровка подписи)

1 Цели и задачи модуля (дисциплины), требования к уровню освоения содержания

1.1 Перечень развиваемых компетенций

ОК-1, ОК-2, ОК-6, ОК-7, ОК-8, ОК-12, ОК-13
ПК-2, ПК-4, ПК-5

1.2 Цели и задачи изучения модуля (дисциплины)

Целью дисциплины является изучение основ построения и функционирования аппаратных средств вычислительной техники, приобретение студентами теоретических знаний об архитектуре ЭВМ и периферийных устройствах, а также практических навыков системного программ
ирования на языке ассемблера при написании приложений, взаимодействующих с периферийными устройствами.

Задачи дисциплины:
1) изучить принципы построения процессоров, интерфейсов передачи данных, устройств управления, арифметико-логических, запоминающих и периферийных устройств;
2) сформировать у студентов теоретические знания об архитектуре ЭВМ и разных типах периферийных устройств;
3) развить практические навыки в области системного программирования на языке низкого уровня.

1.3 Требования к уровню подготовки студента, завершившего изучение данного модуля (дисциплины)

Студенты, завершившие изучение данной дисциплины, должны
знать: классификацию, назначение и принципы построения ЭВМ и периферийных устройств, их организацию и функционирование;
уметь: выполнять основные процедуры проектирования вычислительных устройств, применять теоретические знания в области создания программ на языке ассемблера к решению конкретных инженерных задач;
владеть: средствами анализа вычислительных узлов и блоков.

1.4 Связь с предшествующими модулями (дисциплинами)

Для успешного усвоения материала дисциплины студенты должны обладать знаниями и навыками, полученными в ходе изучения следующих дисциплин:
- информатика;
- операционные системы;
- системное программирование;
- электротехника, электроника и схемотехника.

1.5 Связь с последующими модулями (дисциплинами)

Дисциплина является базовой для изучения курса «Интерфейсы информационно-вычислительных систем».

2 Содержание рабочей программы (модуля)

Семестр 1		
СЛ 0,25 36 часов 1 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 1	Становление и эволюция цифровой вычислительной техники
		Принципы построения и функционирования ЭВМ и вычислительных систем
		Функциональная организация фон-неймановской ВМ
		Архитектура системы команд
		Архитектура и программирование сопроцессора
		Организация шин
		Память
		Устройства управления
		Операционные устройства вычислительных машин
		Языки описания электронной аппаратуры
		Большие и малые интерфейсы IBM PC
		Внешние запоминающие устройства
		Принтеры и способы печати
		Сканеры и технология распознавания символов
		Прочие виды периферийных устройств
СП 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	
СЛР 0,375 54 часов 1,5 ЗЕТ	Активные 1	Программирование сопроцессора
		Работа с файлами
		Работа с видеопамятью
		Вывод текстовой информации в графическом режиме
		Резидентные программы
		Разработка моделей устройств на языке VHDL

	Интерактивные 0	
	Традиционные 0	
КСР 0 0 часов 0 ЗЕТ	Активные 0	
	Интерактивные 0	
	Традиционные 0	
СРС 0,375 54 часов 1,5 ЗЕТ	Активные 0,5	Подготовка к лабораторным занятиям
	Интерактивные 0	
	Традиционные 0,5	Подготовка к сдаче экзамена

3 Инновационные методы обучения

1. Использование при самостоятельной подготовке электронных средств коммуникаций, в том числе специализированных сайтов и форумов.
2. Выполнение лабораторных работ с помощью современного программного обеспечения.
3. Использование тестирования для оценки знаний студентов.
4. Применение рейтинговой системы оценки знаний студентов.

4 Технические средства и материальное обеспечение учебного процесса

1. Компьютерный класс, используемый для проведения лабораторных занятий.
2. Программное обеспечение, используемое при проведении лабораторных занятий.

5 Учебно-методическое обеспечение

5.1 Основная литература

1. Пирогов, Владислав Юрьевич.Ассемблер [Текст] : учеб. курс / В. Ю. Пирогов. - 2-е изд., перераб. и доп. - СПб. : БХВ-Петербург, 2003. - IX, 1036 с. - ISBN 5-94157-328-6 : 221.84 р. - 5 экз.
2. Солдатова, Ольга Петровна.Системное программирование [Текст] : Курс лекций / О. П. Солдатова, С. В. Востокин ; Самар. гос. аэрокосм. ун-т им. С. П. Королева. - Самара : [б. и.], 2002. - 123 с. - ISBN 5-7883-0226-9 : 102.12 р. - 94 экз.
3. Цилькер, Борис Яковлевич.Организация ЭВМ и систем [Текст] : [учеб. для вузов по направлению "Информатика и вычисл. техника"] / Б.Я. Цилькер, С.А. Орлов. - СПб. и др. : Питер : Питер принт, 2004. - 667 с. - (Учебник для вузов). - ISBN 5-94723-759-8 : 180.00 р., 194.00 р., 174.00 р

5.2 Дополнительная литература

1. Догадин, Николай Борисович.Архитектура компьютера [Текст] : учеб. пособие : [для вузов по направлению 050200 "Физ.-мат. образование"] / Н. Б. Догадин. - М. : Бином. Лаб. знаний , 2008. - 271 с. - (Педагогическое образование). - ISBN 978-5-94774-728-7 : 219.00 р. - 1 экз.

5.3 Электронные источники и интернет ресурсы

1. Ассемблер [Электронный ресурс]. – http://www.codenet.ru/cat/Languages/Assembler/.
2. VHDL – язык описания аппаратуры [Электронный ресурс]. – http://www.allhdl.ru/vhdl.php.
3. VHDL портал для студентов и разработчиков [Электронный ресурс]. – http://www.bsuir.by/vhdl/index.php?section=main.

5.4 Методические указания и рекомендации

Текущий контроль знаний студентов в семестре завершается на отчетном занятии, результатом которого является допуск или недопуск студента к экзамену по дисциплине. Основанием для допуска к экзамену является выполнение всех лабораторных работ. Пропуск лекционных занятий может быть основанием для дополнительного вопроса на экзамене.
Промежуточный контроль знаний студентов проводят в семестре в виде экзамена. Экзамен проводится согласно положению о текущем и промежуточном контроле знаний студентов, утвержденному ректором университета. Оценка на экзамене ставится на основании письменного решения задачи в виде текста программы на языке ассемблера и устного ответа студента на вопрос из списка контрольных вопросов, а также, при необходимости, ответов на дополнительные вопросы. В качестве дополнительного задания может быть предложен как теоретический вопрос, так и задача.