Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Институт информационных технологий

Кафедра информационных систем и технологий

Ю. А. Скудняков, И. И. Шпак

ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ И СИСТЕМЫ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для выполнения контрольных работ для специальности 1-36 04 02 «Промышленная электроника»

Минск БГУИР 2020

УДК [004.38+004.7](076)
ББК 32.973.202я73
С46

Р е ц е н з е н т ы:

кафедра программного обеспечения информационных систем и технологий
Белорусского национального технического университета,
(протокол №9 от 20.03.2019);

доцент кафедры энергоэффективных технологий учреждения образования
«Международный государственный экологический институт
имени А. Д. Сахарова» Белорусского государственного университета
кандидат технических наук, доцент В. И. Красовский

Скудняков, Ю. А.
С46 Вычислительные машины и системы : учеб.-метод. пособие / Ю. А. Скудняков, И. И. Шпак. – Минск : БГУИР, 2020. – 79 с. : ил.
ISBN 978-985-543-551-9.

Приведены краткие сведения по теоретическим основам вычислительных машин и систем, теоретическим основам организации памяти современных ЭВМ и вычислительных систем, логическим основам ЭВМ, методикам анализа и синтеза комбинационных схем. Рассмотрена общая характеристика контрольной работы, предложено по три индивидуальных задания для каждого студента.

Предназначено для студентов специальности 1-36 04 02 «Промышленная электроника» заочной формы получения высшего образования, может быть полезно студентам других специальностей, а также магистрантам, аспирантам и специалистам, занимающимся проектированием, созданием и эксплуатацией современных ЭВМ и вычислительных систем.

УДК [004.38+004.7](076)
ББК 32.973.202я73

ISBN 978-985-543-551-9

Содержание

Введение

Информатизация всех отраслей экономики и оснащение современных производств новейшим технологическим оборудованием, вычислительными машинами и системами высокого быстродействия являются важнейшим необходимым условием научно-технического прогресса, в том числе и в области промышленной электроники. Эффективное использование вычислительных машин и систем и создание на их основе современных средств автоматизации и управления основываются на знании принципов действия как аппаратных компонентов вычислительной техники, так и программного обеспечения вычислительной техники.

Целью изучения дисциплины «Вычислительные машины и системы» (ВМиС) является получение студентами знаний о принципах организации и построения цифровых вычислительных систем и сетей, логических основах их работы, функциональных возможностях современных контроллеров, программируемых логических интегральных схем, вычислительных машин и систем и их структурных компонентов, а также навыков использования современных средств автоматизированного проектирования.

В результате изучения дисциплины «Вычислительные машины и системы» обучаемый должен знать:

- многообразие архитектур вычислительного комплекса;
- принципы построения и функционирования операционных устройств вычислительных машин;
- историю и перспективные направления развития вычислительных машин, систем и сетей;
- модели и методы организации процессоров и памяти современной вычислительной техники;
- принципы работы устройств и блоков современных вычислительных машин, систем и сетей.

В результате изучения дисциплины студент должен уметь:

- использовать системы автоматизированного проектирования узлов вычислительных комплексов;
- применять методы и модели различной степени детализации проекта;
- применять технические средства систем автоматизированного проектирования;
- применять современные технологические решения при использовании микропроцессоров, модулей памяти и периферийных устройств вычислительных систем;
- грамотно формулировать постановку задачи проектирования и тестирования компонентов контроллеров и вычислительных систем;
- применять базовые научно-теоретические знания для решения теоретических и практических задач.

Выполнение контрольных работ является одной из важнейших форм самостоятельной работы студента по усвоению материала дисциплины.

ОБЩАЯ ХАРАКТЕРИСТИКА КОНТРОЛЬНОЙ РАБОТЫ

Учебной программой дисциплины ВМиС предусмотрено выполнение одной контрольной работы (КР). Варианты заданий указываются преподавателем индивидуально каждому студенту во время установочной сессии и соответствуют порядковому номеру студента в группе на момент выдачи задания.

Выполнение индивидуального задания по КР предусматривает не только изучение студентами учебной и методической литературы, но и самостоятельную работу над справочной и специальной научно-технической литературой, патентными и рекламно-информационными источниками.

Основная цель выполнения контрольной работы состоит в изучении учебного материала по всем темам дисциплины: в изучении выданных преподавателем теоретических вопросов и поиске ответов на них и решении задач.

КР по дисциплине ВМиС предусматривает выполнение трех заданий.

Для выполнения первого задания требуется изучение и усвоение материала дисциплины, необходимого для ответа на один или группу вопросов по всем разделам программы.

Второе задание предусматривает изучение и усвоение материала по различным видам организации памяти современных ЭВМ и систем. Для этого студенту необходимо усвоить информацию о назначении, принципах построения и функционирования, результатах проведенного сравнительного анализа и методиках выбора видов памяти в зависимости от класса решаемых задач с помощью современных ЭВМ и компьютерных систем, рассмотреть пути перспективного развития устройств памяти.

Для выполнения третьего задания студент должен изучить логические основы ЭВМ, методику анализа и синтеза комбинационных схем, решить задачу их построения. Результаты выполнения задания необходимо представить в письменном и (или) электронном виде.

1 Задание №1

Краткие сведения по теоретическим основам вычислительных машин и систем

Данный раздел содержит только краткие сведения о принципах построения и функционирования вычислительных машин и систем. Более полная информация по данному разделу представлена в приведенных ниже литературных источниках.

Вычислительная машина (**ВМ**), или компьютер, – комплекс технических и программных средств, предназначенный для решения задач пользователя путем автоматической обработки информации.

Вычислительная система (**ВС**) – совокупность взаимодействующих процессоров или вычислительных машин, периферийного оборудования и программного обеспечения, предназначенная для решения задач пользователей.

1.1 История развития электронных вычислительных машин

Впервые классическую архитектуру ВМ предложил Джон фон Нейман.

Основные принципы организации неймановской ВМ:

- использование двоичной системы счисления для представления информации;

- программы и данные записываются в двоичном коде с использованием форматов одинаковой длины с сохранением их в общих запоминающих устройствах, следует выполнять операции над командами программы как над числами;

- управление вычислительным процессом осуществляется централизованно путем последовательного выполнения команд, а каждая команда руководит выполнением одной операции и передает управление следующей команде;

- память ВМ имеет линейную организацию – она состоит из последовательно пронумерованных ячеек.

Укрупненная схема неймановской ВМ представлена на рисунке 1.

Рисунок 1 – Укрупненная схема неймановской ВМ

ВМ состоит из нескольких основных устройств:

- арифметико-логического устройства (АЛУ);
- устройства управления (УУ);
- памяти;
- устройства ввода и вывода.

АЛУ выполняет логические и арифметические операции для переработки информации, хранящейся в памяти.

УУ обеспечивает управление и контроль всех устройств ВМ (управляющие сигналы указаны пунктирными стрелками).

Данные, хранящиеся в запоминающем устройстве, представлены в двоичной форме.

Программа работы ВМ и данные хранятся в одном и том же устройстве памяти.

Для ввода и вывода информации используются устройства ввода и вывода (периферийные устройства).

В архитектуре фон Неймана применяется однородная память микропроцессора, в которую могут записываться различные программы. При этом специальная программа-загрузчик работает с ними как с данными. Затем управление может быть передано этим программам, и они уже начинают выполнять свой алгоритм. При подобном подходе к управлению микропроцессором удается достигнуть максимальной гибкости работы ВМ.

Недостатком архитектуры фон Неймана является возможность непреднамеренного нарушения работоспособности **ВМ** (программные ошибки) и преднамеренное уничтожение ее работы (вирусные атаки).

В гарвардской архитектуре, в отличие от неймановской, имеют место раздельное хранение и обработка команд и данных. Для работы с памятью программ и с памятью данных организуются отдельные шины обмена данными (системные шины) (рисунок 2).

Рисунок 2 – Структура гарвардской архитектуры

Гарвардская архитектура применяется в микроконтроллерах и сигнальных процессорах, где требуется обеспечить высокую надежность работы аппаратуры. К недостаткам такой архитектуры можно отнести высокую стоимость и большое количество внешних выводов микропроцессора.

Современные ВМ в основном базируются на неймановской архитектуре в силу ее более простой реализации.

Кратко история развития вычислительной техники и информационных технологий отображена на рисунке 3.

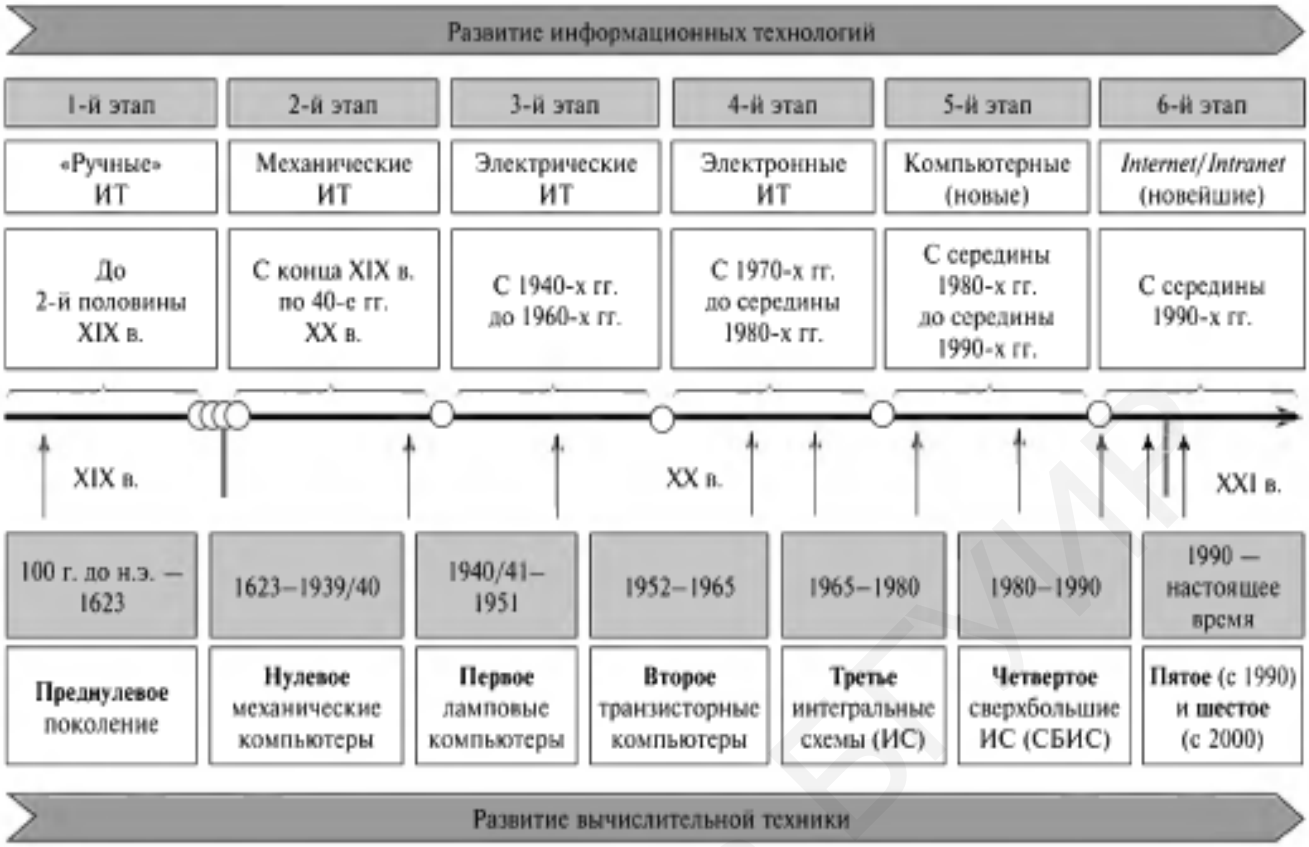


Рисунок 3 – История развития вычислительной техники и информационных технологий

Из рисунка видно, что благодаря развитию элементной базы и компьютерных технологий в настоящее время широко используются наиболее компактные, надежные, экономичные и комфортные персональные компьютеры (ПК), на базе которых создаются и используются современные компьютерные комплексы, системы и сети, позволяющие решать задачи разной сложности в различных сферах человеческой деятельности благодаря использованию информационных ресурсов, имеющих место как в отдельном ПК, так и в компьютерах сети Интернет. Кроме того, современные ПК, обладая относительно высокими показателями качества, такими как производительность, время обработки данных, надежность и приемлемая стоимость, в настоящее время вполне отвечают требованиям большинства пользователей.

Структура современного ПК представлена на рисунке 4 и содержит основные взаимосвязанные функциональные модули компьютера.

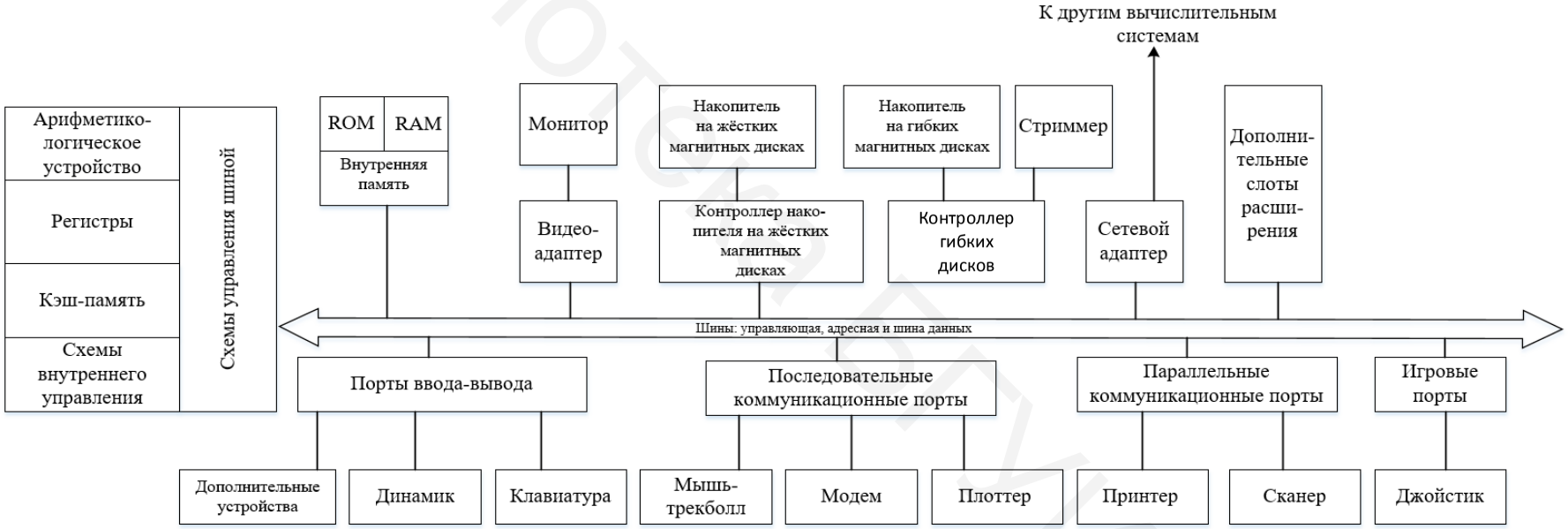


Рисунок 4 – Структура современного ПК

1.2 Элементы и узлы ЭВМ

Каждый модуль, представленный в структуре ПК, состоит из ряда элементов и узлов, необходимых для полноценного функционирования компьютера (см. рисунок 4). К таким элементам и узлам можно отнести: триггеры, элементы различных видов памяти, сумматоры, мультиплексоры, дешифраторы, компараторы, регистры, счетчики и т. д.

Основным узлом ПК является материнская плата, представляющая собой многофункциональный модуль, содержащий ряд основных компонентов, в том числе и микропроцессор.

Микропроцессор состоит:

- из арифметико-логического устройства (АЛУ), предназначенного для выполнения арифметических и логических операций;
- устройства управления (УУ), выполняющего функции управления работой отдельных модулей и ПК в целом;
- микропроцессорной памяти (МП), предназначенной для хранения обрабатываемых данных и программ.

На рисунке 5 представлен общий вид конструктивного исполнения материнской платы ПК с указанием ее основных функциональных компонентов.

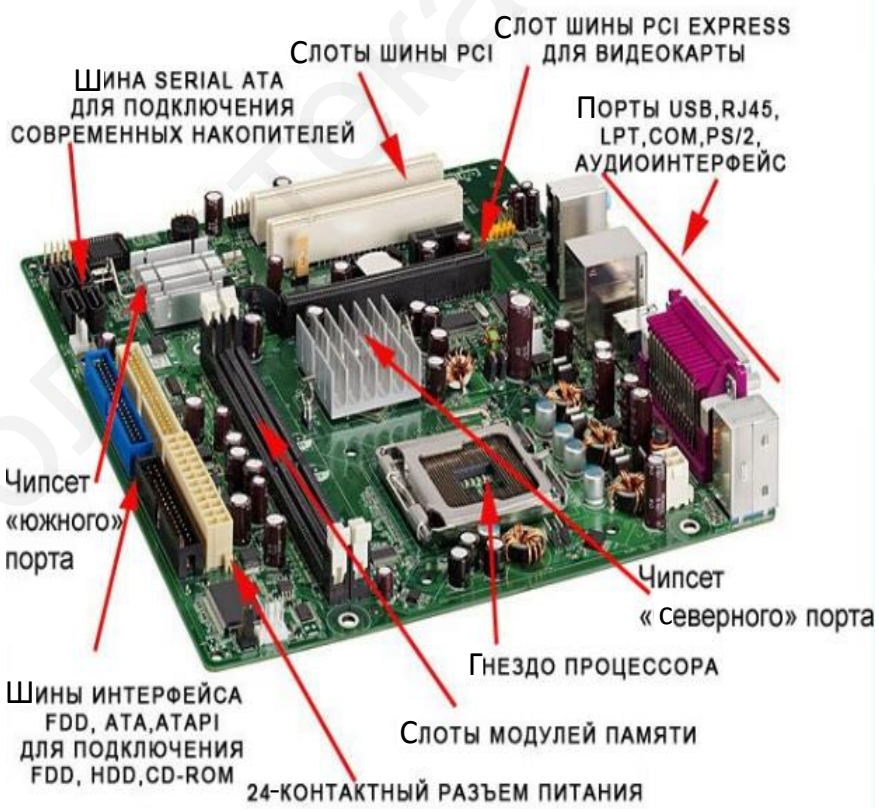


Рисунок 5 – Внешний вид конструкции материнской платы ПК

Внешний вид конструкций отдельных основных функциональных компонентов системного блока ПК показан на рисунке 6.

Рисунок 6 – Конструкции отдельных компонентов системного блока ПК

1.3 Форматы хранения информации

1.3.1 Системы счисления

Система счисления – способ представления любого числа посредством некоторого алфавита символов, называемых цифрами.

Существуют **непозиционные** и **позиционные** системы счисления.

В непозиционных системах счисления каждая цифра сохраняет свое значение независимо от ее положения в числе.

К таким системам относятся: **египетская**, в которой для изображения чисел используются иероглифы; **римская** и **славянская**, в которых для представления цифр используются буквы латинского и славянского алфавитов соответственно. Данные системы в силу своей сложности реализации не получили практического применения.

В позиционных системах значение цифр зависит от их места, или позиции, в числе.

В настоящее время для функционирования ЭВМ применяются позиционные системы счисления с основанием 2, 8, 16, так как в данных системах счисления при той же точности требуется минимальное количество двоичных знаков для представления одного числа.

1.3.2 Форматы хранения графических изображений

Для сохранения изображений в памяти создаются и используются графические файлы.

Кодировка графических изображений осуществляется с помощью определенного способа – формата.

В настоящее время для хранения изображений используются различные форматы файлов, однако лишь часть из них практически применяется в подавляющем большинстве программ. Как правило, несовместимые форматы имеют файлы растровых, векторных, трехмерных изображений, хотя существуют форматы, позволяющие хранить данные разных классов. Многие приложения ориентированы на собственные «специфические» форматы, перенос их файлов в другие программы вынуждает использовать специальные фильтры или экспортировать изображения в стандартный формат.

Большое количество сфер применения изображений с различными требованиями к файлам приводит к многообразию графических форматов.

При выборе формата файлов необходимо учитывать, что данный формат должен поддерживаться заданной сферой применения.

Для редактирования изображения могут использоваться маски, если редактирование выполняется повторно с применением формата, в котором помимо изображения будут сохраняться и маски.

Для хранения векторных изображений используются следующие форматы:

- *AutoCADDXF* и *MicrosoftSYLK* (наиболее распространенные форматы);
- *WMF* – формат, используемый графическими программами ОС *Windows*;
- *AI* – внутренний формат программы *Illustrator*, может открываться программой *Photoshop*;
- *CDR* – внутренний формат программы *Corel Draw*.

Хранение растровых изображений осуществляется с помощью следующих форматов:

- *MicrosoftBMP*, *PCX*, *TIFF* и *TGA* (наиболее распространенные форматы);
- *TIFF* – лучший выбор при передаче изображений и растровой графики в векторные программы и издательские системы;
- *JPG* – формат для сжатия изображения в десятки раз, в котором используется метод сжатия *jpeg*;
- *GIF* – используется для создания веб-страниц, анимаций;
- *PNG* – формат, использующий метод сжатия без потерь качества, который обозначается *deflate*;
- *PDF* – не зависящий от графических программ формат для создания электронной документации, презентаций, а также для передачи графики через компьютерные сети;
- *PSD* – внутренний формат программы *Photoshop*, который поддерживается все большим количеством графических программ. Этот формат позволяет

записывать изображение со многими слоями и дополнительными альфа-каналами, а также с каналами простых цветов и контурами и другой специфической информацией;

- *PCX* является самым известным форматом. Практически любая программа, работающая с графикой, поддерживает этот формат. Формат *PCX* поддерживает метод сжатия, который обозначается *RLE*. Этот формат используется для штрихованных изображений и для изображений с небольшой глубиной цвета;

- *BMP* – формат, который является родным графическим форматом *Windows*. Поддерживается всеми редакторами. В этом формате хранятся небольшие растровые изображения, предназначенные для использования в системе *Windows*. Этот формат невысокого качества и с низкой степенью сжатия. Его не рекомендуется использовать ни для веб-дизайна, ни для передачи.

Метафайловые форматы могут хранить и растровые, и векторные данные, среди которых наиболее распространенными являются *WPG, Macintosh PICT* и *CGM.*

Цветовая модель – математическая модель описания представления цветов в виде кортежей чисел (обычно из трех, реже – четырех значений), называемых цветовыми компонентами или цветовыми координатами. Все возможные значения цветов, задаваемые моделью, определяют цветовое пространство.

Цветовая модель задает соответствие между воспринимаемыми человеком цветами, хранимыми в памяти, и цветами, формируемыми на устройствах вывода.

Наиболее широко применяемы следующие цветовые модели:

1 ***RGB*** – аддитивная (от англ. *add* – добавлять) цветовая модель, как правило, предназначенная для вывода изображения на экраны мониторов и другие электронные устройства. Состоит из синего, красного и зеленого цветов (аббревиатура английских слов ***red, green, blue*** – красный, зеленый, синий), которые образуют все промежуточные, и обладает большим цветовым охватом. Аддитивная цветовая модель предполагает, что вся палитра цветов складывается из светящихся точек, т. е. на бумаге, например, невозможно отобразить цвет в цветовой модели *RGB*, поскольку бумага цвет поглощает, а не светится сама по себе. Итоговый цвет можно получить, прибавляя к исходной черной (несветящейся) поверхности проценты от каждого из ключевых цветов.

Наглядная интерпретация данной модели приведена на рисунке 7.

2 ***Lab*** – цветовая модель, в которой значение светлоты отделено от значения хроматической составляющей цвета (тон, насыщенность). Светлота задана координатой *L* (изменяется от 0 до 100, т. е. от самого темного до самого светлого), хроматическая составляющая – двумя декартовыми координатами *a* и *b*. Первая обозначает положение цвета в диапазоне от зеленого до пурпурного, вторая – от синего до желтого.

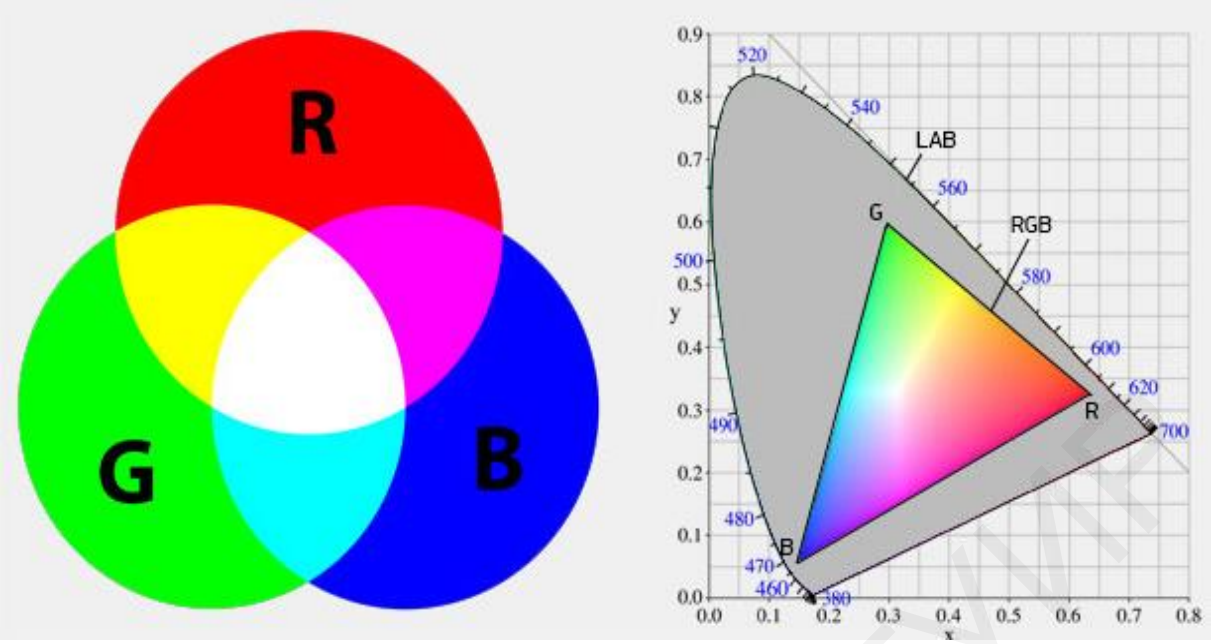


Рисунок 7 – Изображение модели *RGB*

В отличие от цветовых пространств *RGB* или *CMYK*, которые являются, по сути, набором аппаратных данных для воспроизведения цвета на бумаге или на экране монитора (цвет может зависеть от типа печатной машины, марки красок, влажности воздуха на производстве или производителя монитора и его настроек), *Lab* однозначно определяет цвет. Поэтому *Lab* нашла широкое применение в программном обеспечении для обработки изображений в качестве промежуточного цветового пространства, через которое происходит конвертирование данных между другими цветовыми пространствами (например, из *RGB* сканера в *CMYK* печатного процесса). При этом особые свойства *Lab* сделали редактирование в этом пространстве мощным инструментом цветокоррекции. Благодаря характеру определения цвета в *Lab* появляется возможность отдельно воздействовать на яркость, контраст изображения и на его цвет. Во многих случаях это позволяет ускорить обработку изображений, например, при допечатной подготовке. *Lab* предоставляет возможность избирательного воздействия на отдельные цвета в изображении, усилении цветового контраста. Незаменимыми являются и возможности, которые это цветовое пространство предоставляет для борьбы с шумом на цифровых фотографиях.

Наглядные данные модели *Lab* представлены на рисунке 8.

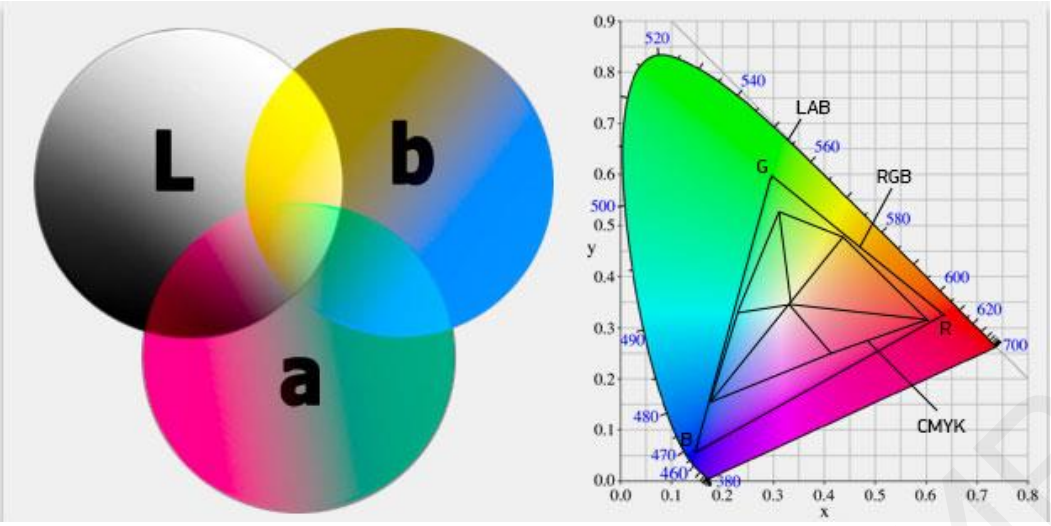


Рисунок 8 – Изображение модели *Lab*

З ***HSB*** – модель, которая в принципе является аналогом *RGB* и основана на ее цветах, но отличается системой координат. Любой цвет в этой модели характеризуется тоном (*Hue*), насыщенностью (*Saturation*) и яркостью (*Brightness*). Тон – это собственно цвет. Насыщенность – процент добавленной к цвету белой краски. Яркость – процент добавленной к цвету черной краски. *HSB* – трехканальная цветовая модель. Любой цвет в *HSB* получается добавлением к основному спектру черной или белой, т. е. фактически серой краски. В основе модели *HSB* лежит *RGB*. В любом случае *HSB* конвертируется в *RGB* для отображения на мониторе и в *CMYK* для печати, а любая конвертация не обходится без потерь.

Принцип формирования модели *HSB* показан на рисунке 9.

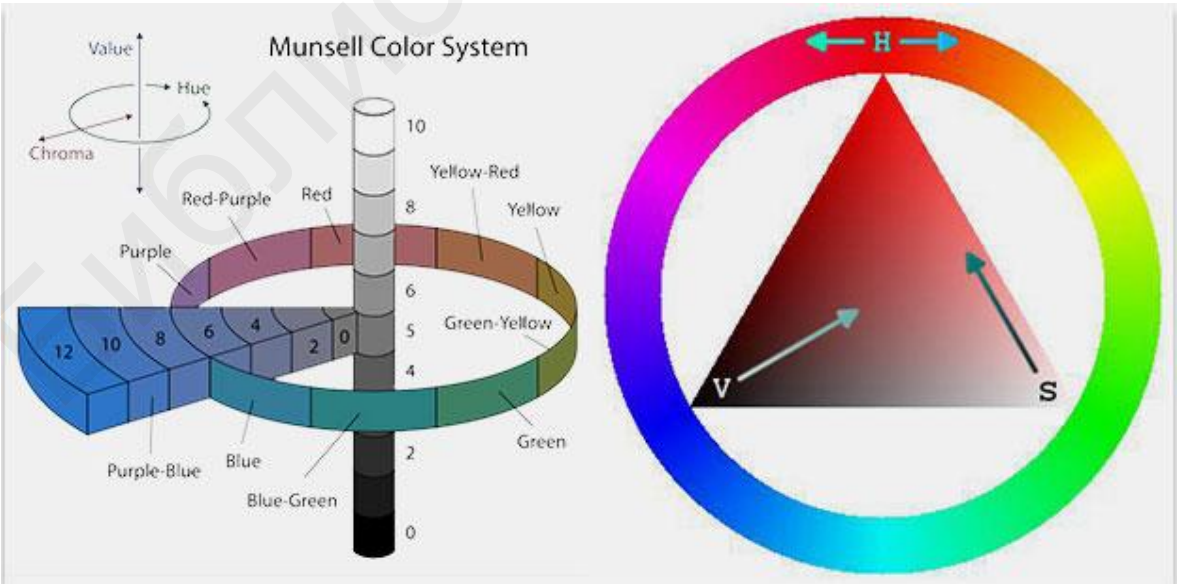


Рисунок 9 – Принцип формирования модели *HSB*

4 ***CMYK – Cyan, Magenta, Yellow, Key color*** – субтрактивная (от англ. *subtract* – вычитать) схема формирования цвета, используемая в полиграфии для стандартной триадной печати. Обладает меньшим, в сравнении с *RGB*, цветовым охватом. *CMYK* – субстрактивная модель, так как бумага и прочие печатные материалы являются поверхностями, отражающими свет.

Удобнее считать, какое количество света отразилось от той или иной поверхности, нежели сколько поглотилось. Таким образом, если вычесть из белого три первичных цвета *RGB*, то получится тройка дополнительных цветов *CMY*. Субтрактивный означает «вычитаемый» т. е. из белого вычитаются первичные цвета. ***Key Color*** (черный) используется в этой цветовой модели в качестве замены смешению в равных пропорциях красок триады *CMY*, поскольку только в идеальном варианте при смешении красок триады получается чистый черный цвет. На практике же он получится, скорее, грязно-коричневым – в результате внешних условий, впитываемости краски материалом и неидеальности красителей.

Модель *CMYK* представлена на рисунке 10.

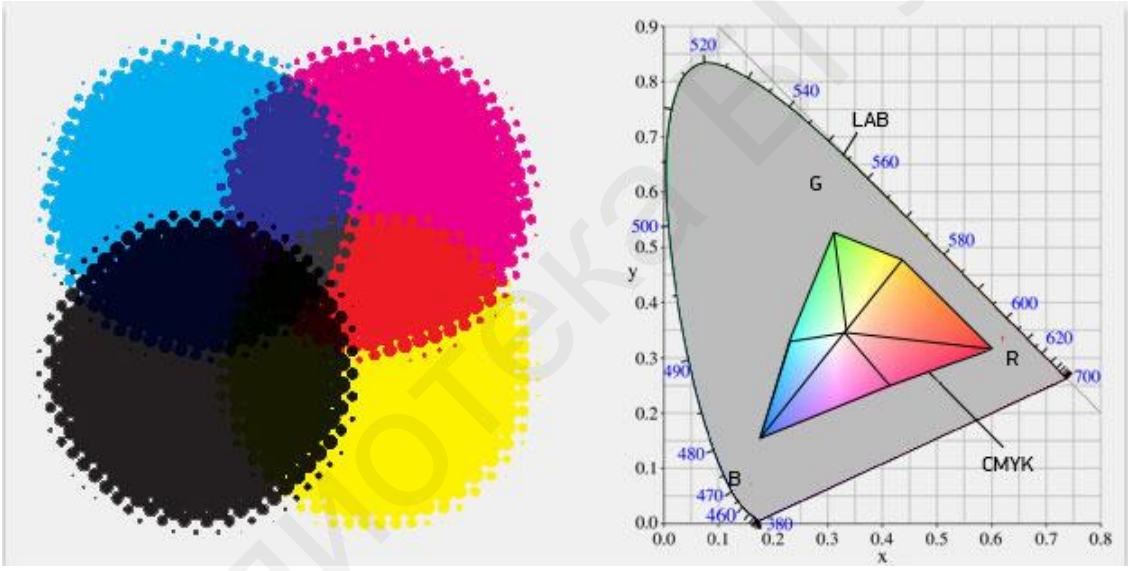


Рисунок 10 – Модель *CMYK*

1.4 Периферийные устройства ЭВМ

Периферийные устройства (ПУ) ЭВМ – набор средств, непосредственно не входящих в центральную часть компьютера.

ПУ делятся:

- на устройства хранения данных;
- устройства ввода/вывода (мышь, клавиатура, монитор, принтер, сканер);
- коммуникационные устройства (сетевые платы, модемы, *Bluetooth*-адаптеры).

Между центральной частью ЭВМ и ПУ существует иерархическая система подключения (рисунок 11).

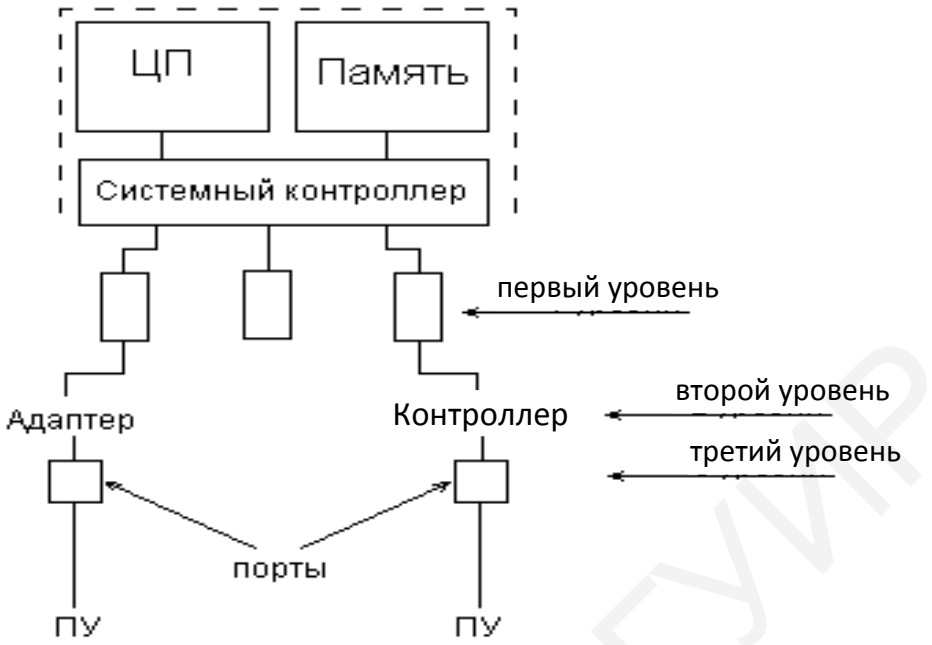


Рисунок 11 – Иерархическая система подключения ПУ

Центральная часть иерархии ЭВМ взаимодействует с ПУ через системный контроллер.

Верхним уровнем иерархии является шина расширения ввода/вывода. К шине расширения подключаются адаптеры и контроллеры.

Приведенные на рисунке 11 уровни являются интерфейсами, осуществляющими передачу информации между устройствами.

Существует множество ПУ различного назначения:

1 ***Монитор*** – устройство, через которое пользователь воспринимает всю визуальную информацию от компьютера. Данные, отображаемые на экране монитора, хранятся в определенном блоке памяти компьютера (видеопамять). Управляет работой монитора устройство, размещенное в системном блоке и называемое видеокартой или видеоадаптером. Видеокарта вместе с монитором образуют видеосистему. Процессор помещает в видеопамять данные, а видеокарта монитора просматривает данные и рисует соответствующее их содержанию изображение на экране. Сейчас наиболее широкое применение получили жидкокристаллические, или *LCD*-мониторы.

LCD-мониторы, выполняются на основе жидких кристаллов. Молекулы жидких кристаллов меняют свойства проходящего сквозь них светового луча, создавая таким образом на мониторе изображение. В *LCD*-мониторах отсутствует вредное электромагнитное излучение, а также такие устройства потребляют незначительное количество энергии.

2 ***Принтеры*** – печатающие устройства, работающие на основе разных физических принципов. Наиболее широко применяемые – лазерные принтеры, использование которых позволяет получать высокое качество печати.

3 ***Манипулятор «мышь»*** – устройство ввода и управления информацией в ЭВМ. По принципу действия мыши делятся на оптико-механические и оптические. Оптические мыши отличаются высокой надежностью и точностью позиционирования на экране.

4 ***Клавиатура*** – устройство ввода информации в ЭВМ. После нажатия клавиши клавиатура посылает процессору сигнал прерывания и заставляет процессор приостановить свою работу и переключиться на программу обработки прерывания клавиатуры. При этом клавиатура в своей собственной специальной памяти запоминает, какая клавиша была нажата. После передачи кода нажатой клавиши процессору эта информация из памяти клавиатуры исчезает.

5 ***Сканер*** – устройство, выполняющее функции оцифровки и ввода в компьютер изображений с бумажных копий. Современные сканеры позволяют оцифровывать изображения даже объемных предметов и диапозитивов (слайдов).

6 ***USB-накопитель на флэш-памяти*** – самое универсальное средство переноса информации, имеет небольшие габариты, высокую механическую прочность, относительно нечувствительно к электромагнитным излучениям и изменениям состояния окружающей среды, а также работает без наличия электропитания.

7 ***Цифровая камера*** формирует любые изображения сразу в компьютерном формате.

8 ***Микрофон*** осуществляет ввод звуковой информации, при этом звуковая карта преобразует звук из аналоговой формы в цифровую.

9 ***Веб-камера*** служит для ввода динамического изображения в компьютер и звука, например, для общения людей через телеконференции.

Сейчас, кроме вышеперечисленных, используется достаточно большое количество и других ПУ.

Для подключения к ЭВМ ПУ на системном блоке имеются разъемы различных портов:

- ***COM*** – последовательный порт, передающий последовательно электрические импульсы, несущие информацию. К ним обычно подключают мышь и модем;
- ***LPT*** – параллельный порт, передающий одновременно восемь электрических импульсов. Реализует более высокую скорость передачи информации;
- ***USB*** (*Universal Serial Bus*) – последовательная универсальная шина, обеспечивающая высокоскоростное подключение нескольких ПУ (сканера, цифровой камеры и т. д.).

1.5 Интерфейс компьютерной системы

Информационный обмен между процессором и другими электронными компонентами ЭВМ осуществляется через интерфейс с использованием системных и локальных шин.

Системная шина (СШ) – это совокупность сигнальных линий, объединенных по их назначению (данные, адреса, управление).

По СШ осуществляется адресация устройств и происходит обмен специальными служебными сигналами. Передачей информации по шине управляет одно из подключенных к ней устройств или специально выделенный для этого узел, называемый арбитром шины.

Для работы с внешними устройствами в этой шине предусмотрены линии аппаратных прерываний (***Interupt ReQuest*** – ***IRQ***) и линии для требования внешними устройствами прямого доступа в память (***Direct Memory Access*** – ***DMA***). Для подключения плат расширения используются специальные разъемы.

Локальные (*local*) шины (ЛШ) непосредственно связывают процессор с контроллерами ПУ. В качестве ЛШ можно привести шины: ***VL-bus*** (***VLB***), предложенная ассоциацией *VESA* (*Video Electronics Standards Association*), и ***PCI*** (***Peripheral Component Interconnect***), разработанная фирмой *Intel*. Обе эти шины были предназначены для увеличения быстродействия компьютера, позволяя таким ПУ, как видеоадаптеры и контроллеры накопителей, работать с тактовой частотой до 33 МГц и выше. Обе шины используют разъемы типа *MCA*.

Спецификация шины *PCI* обладает несколькими преимуществами перед основной версией *VL-bus*. Так, использовать *PCI* можно вне зависимости от типа процессора. Специальный контроллер заботится о разделении управляющих сигналов ЛШ процессора и *PCI*-шины и, кроме того, осуществляет арбитраж на *PCI*. Именно поэтому данная шина может использоваться и в иных компьютерных платформах. Гибкость и быстродействие этой шины предполагают и большие аппаратные затраты, чем для *VL-bus*. Тем не менее шина *PCI* стала практическим стандартом для систем на базе *Pentium* и не менее успешно используется в других компьютерах, даже и не *PC*-совместимых.

Шина *PCI* представляет собой мощное средство взаимодействия различных компонентов ПК, как расположенных внутри системного блока, так и находящихся за его пределами. Использование данной шины позволяет разделить различные функции, возложенные на чипсет материнской платы, на две группы. При этом имеет место интеграция одной группы функций в микросхему с названием ***North bridge*** **(Северный мост)**. Другая группа функций была интегрирована (фирмой *Intel* и некоторыми из ее конкурентов) в микросхему под названием ***South bridge*** **(Южный мост)**. Эти микросхемы соединяются шиной *PCI*.

1.6 Архитектура команд вычислительной системы

1.6.1 Формы представления данных

Наименьшая единица информации – двоичный разряд (нуль или единица), получивший название бит. Для представления символов в ЭВМ используется байт (восемь бит).

Наименьшей адресуемой структурной единицей информации в современных ЭВМ принят байт и байтовая организация информации в оперативной памяти.

Для представления в ЭВМ алфавитно-цифровой информации обычно применяется машинное слово, представляющее совокупность символов, которая извлекается из оперативной памяти или записывается в нее за одно обращение. Машинное слово обычно содержит целое число байтов, которые образуются последовательно, начиная с нуля.

В ЭВМ и микроЭВМ в качестве операндов используются следующие форматы данных: байт, полуслово, слово, двойные слово и поле – информационная единица переменной длины, но не более 256 байт. Это объясняется тем, что число возможных комбинаций допустимых значений нулей или единиц в одном байте равно 2^8= 256 (от 0 до 255), т. е. максимально возможное двоичное число равно 1111 1111_2. И в десятичном виде оно будет иметь следующее значение:

1111 1111_2 = (1111 1111_2 + 1) – 1 = 1000 000 – 1 = 2^8 – 1 = 255.

Для микропроцессоров и микроЭВМ применяются форматы данных с длиной слова 4, 8, 12, 16, 32 и 64 бита. Побайтовая организация информации в памяти ЭВМ обеспечивает информационную совместимость форматов данных между различными микропроцессорами и микроЭВМ.

На рисунках 12 и 13 представлены форматы данных фиксированной и переменной длины соответственно.

Рисунок 12 – Формат данных фиксированной длины

Байт	Байт	Байт	Байт	Байт	...	Байт
0	1	2	3	4	...	255

Рисунок 13 – Формат данных переменной длины

При размещении обрабатываемой информации в ЭВМ следует учитывать необходимость контроля ее обработки и адресации в ячейке оперативной памяти. Это вызывает определенные требования к организации разрядной сетки ЭВМ. Под разрядной сеткой ЭВМ понимают количество разрядов, необходи-

мых для размещения в ячейке оперативной памяти полного машинного слова. Для каждого типа ЭВМ она имеет строго определенное количество разрядов.

В ЭВМ используются две формы представления чисел в разрядной сетке:

- с фиксированной запятой (точкой) (ФЗ);
- с плавающей запятой (точкой) (ПЗ).

Представление с ФЗ – это естественная форма представления числа, которая характеризуется тем, что положение его запятой в разрядной сетке всегда остается постоянным (фиксируется), независимо от величины самого числа. Обычно запятая фиксируется перед старшим или после младшего разряда. Если запятая фиксируется перед старшим разрядом, то числа в ЭВМ представляются как правильные дроби, если после младшего, то как целые числа.

При выполнении на ЭВМ вычислений необходимо, чтобы все исходные и получающиеся в процессе вычислений промежуточные и конечные данные не выходили за диапазон чисел, представимых в данной разрядной сетке. В противном случае в вычислениях могут возникнуть грубые ошибки. Для этого при программировании задачи данные, участвующие в вычислениях, берутся с соответствующими масштабными коэффициентами.

Использование представления чисел в ФЗ позволяет упрощать схемы ЭВМ, повысить ее быстродействие, но создает трудности при программировании. Первые ЭВМ были с ФЗ.

В настоящее время представление чисел с ФЗ используется как основное и единственное лишь в сравнительно небольших по своим вычислительным возможностям масштабах, применяемых для управления технологическими процессами и обработки измерительной информации в реальном масштабе времени.

В ЭВМ, предназначенных для решения широкого круга вычислительных задач, основным является представление чисел с плавающей запятой, поскольку оно не требует масштабирования данных. В ЭВМ форма с ПЗ используется при переполнении чисел ФЗ. Причем перевод выполняется автоматически по подпрограмме (микропрограмме).

В ЭВМ часто наряду с ПЗ используется и форма с ФЗ, поскольку операции с такими числами выполняются за меньшее время. При этом в большинстве случаев формат чисел с ФЗ служит для представления целых двоичных чисел (запятая фиксируется справа от младшего разряда числа) и выполнения операций над ними, что, в частности, необходимо для операций над кодами адресов (операции индексной арифметики).

Представление числа с ПЗ в общем случае имеет вид

$N = mr^{\pm p}$,

где m – мантисса числа N, $|m| < 1$;

p – порядок числа;

r – основание с/с;

$r^{\pm p}$ – характеристика числа N.

Это представление числа называется также *полулогарифмическим*, так как часть числа – характеристика (r^p) выражена в логарифмической форме.

Мантисса числа *m* является правильной дробью.

Порядок числа *p*, который может быть положительным или отрицательным числом, определяет положение запятой в числе *N*:

$129{,}467 = 12{,}9467 \cdot 10^{+1} = 0{,}129467 \cdot 10^{+3} = 0{,}000129467 \cdot 10^{+6} = 12946{,}7 \cdot 10^{-2}$.

Для двоичных чисел $N = m \cdot 2^{\pm p}$.

1.6.2 Семейство процессоров *Intel*

Первый микропроцессор *I*4004 был изготовлен в 1971 году, и с тех пор фирма ***Intel* (*INTegrated ELectronics*)** прочно удерживает лидирующее положение на данном сегменте рынка.

Реализация ряда проектов фирмы *Intel* по разработке однокристальных микропроцессоров (***i*4040**, ***i*8008**) возвестила о наступлении эры персональных компьютеров. Наиболее успешным был, пожалуй, проект разработки микропроцессора ***i*8080**. Именно на этом микропроцессоре был основан компьютер «Альтаир», для которого молодой Билл Гейтс написал свой первый интерпретатор Бейсика. Он был выполнен по *n*-канальной МОП-технологии (*n-MOS*), а его тактовая частота не превышала 2 МГц. Классическая архитектура *i*8080 оказала огромное влияние на дальнейшее развитие однокристальных микропроцессоров. Несмотря на заслуженный успех *i*8080, настоящим промышленным стандартом для персональных компьютеров стал другой микропроцессор фирмы *Intel*.

Микропроцессор *i*8088 был анонсирован *Intel* в июне 1979 года, а в 1981 году фирма ***IBM***, выбрала этот микропроцессор для своего первого ПК. Новый чип содержал примерно 29 тыс. транзисторов. Одним из существенных достоинств микропроцессора *i*8088 была возможность (благодаря 20 адресным линиям) физически адресовать область памяти в 1 Мбайт. Для *IBM PC* в этом пространстве программам было отведено всего лишь 640 Кбайт.

Хотя с внешними периферийными устройствами (дисками, видео) *i*8088 был связан через свою внешнюю 8-разрядную шину данных, его внутренняя структура (адресуемые регистры) позволяла работать с 16-разрядными словами. Как известно, на системной шине *IBM PC* для передачи данных было отведено восемь линий (1 байт). Первоначально микропроцессор *i*8088 работал на частоте 4,77 МГц и имел быстродействие около 0,33 *MIPS* (*MillionInstruction-PerSecond*), однако впоследствии были разработаны его модификации, рассчитанные на более высокую тактовую частоту (например, 8 МГц).

В июне 1978 года появился процессор ***i*8086**, ставший популярным в основном благодаря компьютеру ***Compaq Desk Pro***. Программная модель (доступные регистры) этого микропроцессора полностью совпадала с моделью *i*8088. Основное отличие данных микропроцессоров состоит в различной разрядности внешней шины данных: 8 разрядов у *i*8088 и 16 разрядов у *i*8086. Понятно, что более высокой производительности с новым микропроцессором можно было достичь только

при использовании компьютера, на системной шине которого под данные предусмотрено 16 линий. Адресная шина микропроцессора *i*8086 по-прежнему позволяла адресовать 1 Мбайт памяти.

Опираясь на архитектуру *i*8086 и учитывая запросы рынка, фирма *Intel* в феврале 1982 года выпустила свой новый микропроцессор – ***i*80286**. На кристалле было реализовано около 130 тыс. транзисторов. Надо сказать, что этот чип появился практически одновременно с новым компьютером фирмы ***IBM PC/AT***. Наряду с увеличением производительности этот микропроцессор (*i*80286) мог теперь работать в двух режимах: реальном и защищенном. Если первый режим был (за рядом исключений) похож на обычный режим работы *i*8088/86, то второй использовал более изощренную технику управления памятью. В частности, защищенный режим работы позволял, например, таким программным продуктам, как *Windows* 3.0 и *OS*/2, работать с оперативной памятью выше 1 Мбайта. Благодаря 16 разрядам данных на новой системной шине, которая была впервые использована в *IBM PC*/*AT*286, микропроцессор *i*80286 мог обмениваться с периферийными устройствами 2-байтными сообщениями. 24 адресные линии позволяли в защищенном режиме обращаться уже к 16 Мбайтам памяти. В микропроцессоре *i*80286 впервые на уровне микросхем были реализованы мультизадачность и управление виртуальной памятью. При тактовой частоте 8 МГц достигалась производительность 1,2 *MIPS*.

В октябре 1985 года фирмой *Intel* был анонсирован первый 32-разрядный микропроцессор ***i*80386**. Новый чип содержал примерно 275 тыс. транзисторов. Полностью 32-разрядная архитектура (32-разрядные регистры и 32-разрядная внешняя шина данных) в новом микропроцессоре была дополнена расширенным устройством управления памятью *MMU* (*Memory Management Unit*), которое помимо блока сегментации (*Segmentation Unit*) было дополнено блоком управления страницами (*Paging Uni*t). Это устройство позволяло легко переставлять сегменты из одного места памяти в другое (свопинг) и освобождать драгоценные килобайты памяти. На тактовой частоте 16 МГц быстродействие нового процессора составило примерно 6 *MIPS*.

В реальном режиме (после включения питания) микропроцессор *i*80386 работал как «быстрый *i*8088» (адресное пространство 1 Мбайт, 16-разрядные регистры). Защищенный режим был полностью совместим с аналогичным режимом в *i*80286. Тем не менее в этом же режиме *i*80386 мог выполнять и свои «естественные» (*native*) 32-разрядные программы. Вспомним, что 32 адресные линии микропроцессора позволяют физически адресовать 4 Гбайта памяти. Кроме того, был введен новый режим виртуального процессора (***V*86**). В этом режиме могли одновременно выполняться несколько задач, предназначенных для *i*8086.

Более дешевая альтернатива 32-разрядному процессору *i*80386, который впоследствии получил окончание *DX*, появилась только в июне 1988 года. Это был микропроцессор ***i*80386*SX***. В отличие от своего старшего «собрата» новый микропроцессор использовал 16-разрядную внешнюю шину данных и 24-разрядную адресную (адресуемое пространство 16 Мбайт). Это было особенно удобно для

стандартных *PC/AT*, системная шина которых использует, как известно, только 16 линий данных. Благодаря дешевизне нового изделия многие производители аппаратных средств стали заменять теперь уже устаревший микропроцессор *i*80286 на более производительный *i*80386*SX*. Одним из решающих факторов для замены была полная совместимость 32-разрядных микропроцессоров: программное обеспечение, написанное для *i*80386*DX*, корректно работало и на *i*80386*SX*. Дело в том, что внутренние регистры их были полностью идентичны. К концу 1988 года микропроцессор *i*80386*SX* выпускался в количествах, существенно превосходящих рекордные показатели для *i*80386*DX*. Кстати, считается, что индекс *SX* произошел от слова *SiXteen* (шестнадцать), поскольку разрядность внешней шины данных нового тогда процессора была именно такой. В дальнейшем, правда, для 486-х процессоров *SX* стал означать отсутствие математического сопроцессора.

В 1989 году фирма ***Intel*** впервые анонсировала микропроцессор ***i486DX***, который содержал более миллиона транзисторов (а точнее, 1,2 млн) на одном кристалле и был полностью совместим с процессорами ряда *x*86. Напомним, что на кристалле первого члена этого семейства – микропроцессора *i*8088 – насчитывалось около 29 тыс. транзисторов. В борьбе с «микропроцессорами-клонами» фирма *Intel* намеренно убрала из названия нового устройства число 80. Новая микросхема впервые объединила на одном чипе такие устройства, как центральный процессор, математический сопроцессор и кэш-память. Использование конвейерной архитектуры, присущей ***RISC***-процессорам, позволило достичь четырехкратной производительности обычных 32-разрядных систем. Это связано с уменьшением количества тактов для реализации каждой команды. Встроенная кэш-память 8 Кбайт ускоряет выполнение программ за счет промежуточного хранения часто используемых команд и данных. На тактовой частоте 25 МГц микропроцессор показал производительность 16,5 *MIPS*. Созданная в июне 1991 года версия микропроцессора с тактовой частотой 50 МГц позволила увеличить производительность еще на 50 %. Встроенный математический сопроцессор существенно облегчил и ускорил математические вычисления. Однако впоследствии стало ясно, что подобный сопроцессор необходим всего лишь 30 % пользователей.

Появление нового микропроцессора ***i486SX*** фирмы ***Intel*** вполне можно было считать одним из важнейших событий 1991 года. Уже предварительные испытания показали, что компьютеры на базе *i*486*SX* с тактовой частотой 20 МГц работают быстрее (примерно на 40 %) компьютеров, основанных на *i*80386*DX* с тактовой частотой 33 МГц. Микропроцессор *i*486*SX*, подобно оригинальному *i*486*DX*, содержал на кристалле кэш-контроллер и кэш-память, а вот математический сопроцессор у него был заблокирован. Значительная экономия (благодаря исключению затрат на тестирование сопроцессора) позволила фирме *Intel* существенно снизить цены на новый микропроцессор. Если микропроцессор *i*486*DX* был ориентирован на применение в сетевых серверах и рабочих станциях, то *i*80486*SX* послужил отправной точкой для создания мощных настольных компьютеров.

В семействе микропроцессоров *i*486 предусматривалось несколько новых возможностей для построения мультипроцессорных систем: соответствующие

команды поддерживают механизм семафоров памяти, аппаратно реализованное выявление недостоверности строки кэш-памяти обеспечивает согласованность между несколькими модулями кэш-памяти и т. д. Для микропроцессоров семейства *i*486 допускалась адресация физической памяти размером 4 Гбайта и виртуальной памяти размером 64 Тбайта.

К концу 1991 года 32-разрядные микропроцессоры стали стандартными для компьютеров типа лэптоп и ноутбук, однако обычные микросхемы *i*80386*DX*/*SX* не полностью отвечали требованиям разработчиков портативных компьютеров. Для удовлетворения потребностей этого сегмента рынка фирмой ***Intel*** в 1990 году был разработан микропроцессор ***i*80386*SL***, который содержал примерно 855 тыс. транзисторов. Данный микропроцессор представлял собой интегрированный вариант микропроцессора *i*80386*SX*, базовая архитектура которого была дополнена еще несколькими вспомогательными контроллерами. По существу, все компоненты, необходимые для построения портативного компьютера, сосредоточены в двух микросхемах: микропроцессоре *i*80386*SL* и периферийном контроллере *i*82360*SL*. В набор ***i*386*SL*** впервые было введено новое прерывание, называемое *System Management Interrupt* (*SMI*), которое использовалось для обработки событий, связанных, например, с управлением потребляемой мощности. Вместе с математическим сопроцессором ***i*80387*SL*** данный набор микросхем позволял создать 32-разрядный компьютер на площади, ненамного превышающей размер игральной карты.

1.6.3 Конвейерное выполнение команды

Для повышения производительности процессора при выполнении операций его операционное устройство (ОУ) может строиться по блочному принципу. В таких блочных ОУ реализуется несколько функционально независимых исполнительных устройств, выполняющих различные операции (или различные группы операций, например: три блока целочисленного сложения, два – целочисленного умножения, по одному блоку деления, сложения и умножения с плавающей запятой и т. д.).

Эти устройства работают параллельно: каждое обрабатывает свои операнды.

Управление этими устройствами осуществляется с помощью так называемых длинных командных слов **(*Very Long Instruction Word – VLIW*)**. Командные слова включают инструкции для каждого из исполнительных устройств, а также операнды или указатели на них.

Преимуществом блочных ОУ является более высокая производительность, достигаемая за счет распараллеливания вычислений. В то же время использование таких устройств не всегда эффективно, поскольку не всегда есть возможность загрузить все исполнительные устройства в каждом такте, в результате часть из них простаивает.

Более эффективными часто оказываются конвейерные операционные устройства, поскольку конвейеризовать вычисления в ряде случаев проще, чем распараллелить, что связано с повторением однотипных вычислений в алгоритмах.

Для конвейеризации вычислений необходимо:

- разбить вычисления на последовательность одинаковых по времени этапов;
- реализовать каждый этап аппаратно в виде ступени конвейера;
- обеспечить фиксацию промежуточных результатов вычислений на выходе каждой ступени в регистрах-защелках.

Эффективность конвейера будет тем выше, чем больше задач будет поступать на его вход.

Типичным примером конвейерных операционных устройств могут служить матричные умножители, включающие матрицу операционных элементов (сумматоров) и применяемые в основном для умножения матриц.

1.6.4 Архитектура системы команд *RISC* и *CISC* процессоров

Под архитектурой системы команд (*Instruction Set Architecture – ISA*) понимают состав и возможности системы команд (СК), общий взгляд на СК и связанную с ней микроархитектуру процессора с точки зрения программиста. Во многом именно архитектура СК определяет трактовку архитектуры компьютера вообще, как «…абстрактного представления о вычислительной машине с точки зрения программиста».

Исторически первые микропроцессоры, появившиеся в 70-х годах прошлого века, имели относительно простую систему команд, что объяснялось небольшими возможностями интегральной схемотехники. По мере увеличения степени интеграции ИМС разработчики микропроцессоров (МП) старались расширять систему команд и делать команды более функциональными, «семантически нагруженными». Это объяснялось, в частности, двумя моментами:

- во-первых, требованиями экономить память для размещения программ, оставлять больше памяти под данные и т. д.;
- во-вторых, возможностью реализовать внутри кристалла процессора сложные инструкции быстрее, чем при их программной реализации.

В результате появились процессоры с большими наборами команд, причем команды эти также зачастую являлись достаточно сложными. Впоследствии эти МП назвали *CISC* (*Complete Instruction Set Computer* – компьютер с полным набором команд или *Complex ISC* – со сложным набором команд).

Типичным примером *CISC*-процессоров являются процессоры семейства *x*86 корпорации *Intel* и ее конкурентов (а также *Motorola* 68*K* и др.).

Наряду с отмеченными преимуществами процессоры *CISC* обладали и рядом недостатков. В частности, команды оказывались сильно неравнозначны-

ми по времени выполнения (разное количество тактов), плохо конвейеризовывались, требовали сложного и длительного декодирования и выполнения.

Для повышения производительности стали использовать жесткую логику управления, что отразилось на регулярности и сложности кристаллов (нерегулярные кристаллы менее технологичны при изготовлении). На кристалле оставалось мало места для регистров общего назначения (РОН) и кэш-памяти.

Кроме того, исследования показали, что производители компиляторов и просто программисты не используют многие сложные инструкции, предпочитая использовать последовательность коротких.

Разработчики подошли к концепции более простого и технологичного процессора с некоторым возвратом назад к простым и коротким инструкциям. С конца 70-х до середины 80-х годов появляются проекты таких процессоров Стэнфордского университета и университета Беркли (Калифорния) – *MIPS* и *RISC*.

В основу архитектуры *RISC* (*Reduced Instruction Set Computer* – компьютер с сокращенным набором команд) положены, в частности, принципы отказа от сложных и многофункциональных команд, уменьшения их количества, а также концентрация на обработку всей информации преимущественно на кристалле процессора с минимальными обращениями к памяти.

Основные особенности архитектуры RISC:

- уменьшение числа команд (до 30–40);
- упрощение и унификация форматов команд;
- в системе команд преобладают короткие инструкции (например, часто в СК отсутствуют умножения);
- отказ от команд типа «память – память» (например, *MOVSB* в *x*86);
- работа с памятью сводится к загрузке и сохранению регистров (поэтому другое название *RISC – Load-Store Architecture*, т. е. архитектура типа «загрузка – сохранение»);
- преимущественно реализуются трехадресные команды, например: *add r*1, *r*2, *r*3 – значит, сложить *r*2 с *r*3 и поместить результат в *r*1;
- большой регистровый файл до 32-64 РОН;
- предпочтение отдается жесткой логике управления.

Достоинства архитектуры RISC:

- облегчается конвейерная, суперскалярная и другие виды параллельной обработки, планирование загрузки, переупорядочивание, предвыборка и т. д.;
- более эффективно используется площадь кристалла (больше памяти: РОН, кэш);
- быстрее выполняется декодирование и исполнение команд, что приводит к повышению тактовой частоты.

Примерами семейств процессоров с *RISC*-архитектурой могут служить *DEC Alpha*, *SGI MIPS*, *Sun SPARC* и др. Большинство современных суперскалярных и *VLIW*-процессоров (в том числе и *Intel*) либо имеют архитектуру *RISC*, либо реализуют похожие на *RISC* принципы, либо поддерживают

CISC-инструкции, но внутри транслируют их в *RISC*-подобные команды для облегчения загрузки конвейеров и решения других задач.

1.7 Центральный процессор вычислительной машины

Процессор – центральное устройство компьютера. Он выполняет находящиеся в оперативной памяти команды программы и «общается» с внешними устройствами благодаря шинам адреса, данных и управления, выведенными на специальные контакты корпуса микросхемы.

К обязательным компонентам процессора относятся арифметико-логическое (исполнительное) устройство (АЛУ) и устройство управления (УУ).

Выполнение процессором команды предусматривает:

- арифметические действия;
- логические операции;
- передачу управления (условную и безусловную);
- перемещение данных из одного места памяти в другое;
- координацию взаимодействия различных устройств ЭВМ.

Выделяют четыре этапа обработки команды процессором:

1) выборка;
2) декодирование;
3) выполнение и запись результата.

В ряде случаев пока первая команда выполняется, вторая может декодироваться, а третья – выбираться.

Функции процессора:

1) обработка данных по заданной программе путем выполнения логических и арифметических операций;

2) программное управление работой устройств компьютера.

Процессор состоит из ячеек. В ячейках процессора данные не хранятся, а обрабатываются. Во время обработки они могут изменяться самыми разными способами. Ячейки процессора называются **регистрами.**

Регистр выполняет функцию кратковременного хранения числа или команды. Над содержимым некоторых регистров специальные электронные схемы могут выполнять некоторые манипуляции. Например, «вырезать» отдельные части команды для последующего их использования или выполнять определенные арифметические операции над числами. Основным элементом регистра является электронная схема, называемая триггером, которая способна хранить одну двоичную цифру (разряд).

Разные регистры процессора имеют разное назначение:

- регистры общего назначения используются для операций с данными (байтами, словами и двойными словами);
- адресные регистры служат для хранения в них адресов, по которым процессор находит данные в памяти;
- существуют специальные регистры для самопроверок процессора;

– биты флагового регистра служат как бы флажками, которые включаются или выключаются в особых случаях. Когда от меньшего числа отнимают большее, то занимают одну единицу в старшем разряде. На этот случай во флаговом регистре есть специальный флажок, который включается при таком событии. Существуют флажки, включающиеся при переполнении регистров или их обнулении, а также еще несколько специальных флажков.

Некоторые важные регистры имеют свои названия, например:

- сумматор – регистр АЛУ, участвующий в выполнении каждой операции;
- счетчик команд – регистр УУ, содержимое которого соответствует адресу очередной выполняемой команды. Служит для автоматической выборки программы из последовательных ячеек памяти;
- регистр команд – регистр УУ для хранения кода команды на период времени, необходимый для ее выполнения. Часть его разрядов используется для хранения кода операции, остальные – для хранения кодов адресов операндов.

У компьютеров четвертого поколения функции центрального процессора выполняет микропроцессор – сверхбольшая интегральная схема (СБИС), реализованная в едином полупроводниковом кристалле (кремния или германия) площадью меньше 0,1 см^2.

Микропроцессоры различаются рядом важных характеристик:

- тактовой частотой обработки информации;
- разрядностью;
- интерфейсом с системной шиной;
- адресным пространством (адресацией памяти).

Тактовая частота обработки информации. Тактом называют интервал времени между началом подачи двух последовательных импульсов электрического тока, синхронизирующих работу различных устройств компьютера. Специальные импульсы для отсчета времени для всех электронных устройств вырабатывает тактовый генератор частоты, расположенный на системной плате. Его главный элемент представляет собой кристалл кварца, обладающий стабильностью резонансной частоты. Тактовая частота определяется как количество тактов в секунду и измеряется в мегагерцах (1 МГц = 1 млн тактов/с). Тактовая частота влияет на скорость работы, быстродействие микропроцессора. Переход к микропроцессору с большей тактовой частотой означает скорое повышение обработки информации.

Быстродействие процессора определяется количеством операций, выполняемых им в секунду.

Один из способов повышения быстродействия микропроцессора – использование кэш-памяти. Это позволяет избежать циклов ожидания в работе МП, пока информация из соответствующих схем памяти установится на системной шине данных компьютера. Таким образом, кэш-память функционально предназначена для согласования скорости работы сравнительно медленных устройств с относительно быстрым микропроцессором.

Для улучшения показателей при выполнении операций с плавающей запятой создано и используется специальное устройство – математический сопроцессор, выполненный в виде интегральной схемы, работающей во взаимодействии с центральным микропроцессором и предназначенной только для выполнения математических операций.

Разрядность процессора – это число одновременно обрабатываемых процессором битов, т. е. количество внутренних битовых (двоичных) разрядов – важнейший фактор производительности микропроцессора.

Процессор может быть 8-, 16-, 32- и 64-разрядным. Разрядность характеризует объем информации, перерабатываемый процессором компьютера за единицу времени.

Интерфейс с системной шиной – разрядность внутренней шины данных микропроцессора может не совпадать с количеством внешних выводов для линии данных. Например, микропроцессор с 32-разрядной внутренней шиной данных может иметь только 16 внешних линий данных. Это означает, что разрядность интерфейса с внешней шиной данных равна 16. Аналогичная ситуация может наблюдаться с другой частью системной шины – адресной шиной.

Выполнение процессором команды предусматривает наряду с арифметическими действиями и логическими операциями передачу управления и перемещение данных из одного места памяти в другое.

Поэтому важна не только разрядность внутренних шин процессора, но и его интерфейс с системной шиной.

Адресное пространство (адресация памяти) – одна из функций процессора, которая состоит в перемещении данных, в организации их обмена с ПУ и оперативной памятью. При этом процессор формирует код устройства, а для ОЗУ – адрес ячейки памяти. Код адреса передается по адресной шине. Объем физически адресуемой микропроцессором оперативной памяти называется его адресным пространством, которое определяется разрядностью внешней шины адреса. Если N – разрядность адресной шины, то количество различных двоичных чисел, которые можно по ней передать, равно 2.

Число, передаваемое по адресной шине при обращении процессора к оперативной памяти, есть адрес ячейки ОЗУ (ее порядковый номер). Значит, 2^N – это количество ячеек оперативной памяти, к которым, используя адресную шину, может обратиться (адресоваться) процессор, т. е. 2^N – объем адресного пространства процессора. Следовательно, при 16-, 20-, 24- или 32-разрядной шине адреса создается адресное пространство соответственно 2^{16} = 64 Кбайта, 2^{20} = 1 Мбайт, 2^{24} = 16 Мбайт, 2^{32} = 4 Гбайта.

1.7.2 Суперскалярные архитектуры процессоров ЭВМ

Использование конвейера команд позволяет в лучшем случае снизить показатель производительности *CPI* до 1, т. е. на каждом такте с конвейера должна «сходить» новая обработанная команда. В этом случае производительность

процессора должна увеличиться в четыре раза, при его длительности такта в 10 нс (тактовая частота 100 МГц) имеем производительность в 100 *MIPS*. Но, во-первых, у *Celeron* такой показатель равен примерно 250, а во-вторых, достижение показателя 1 *CPI* не всегда возможно из-за конфликтов при конвейеризации. В действительности в лучшем случае будет 1,5–2 *CPI*.

Для достижения такой высокой производительности в *Celeron* и других процессорах с архитектурой P6 используется суперскалярная обработка, т. е. обработка с многопотоковым конвейером команд, когда процессор выполняет больше одной команды за такт.

Фактически в суперскалярном процессоре несколько потоков проходят через несколько исполнительных устройств, а остальные ступени так или иначе работают с одним потоком. Для согласования разных скоростей потоков декодирования, выборки, трансляции в *RISC*-подобные инструкции, переупорядочивания и потоков в исполнительных устройствах применяют различные очереди инструкций (буферы *FIFO*), которые есть у каждого из исполнительных устройств.

В процессорах с длинным командным словом (*Very Long Instruction Word*) (рисунок 14) используется альтернативный суперскалярной обработке принцип распараллеливания последовательного алгоритма.

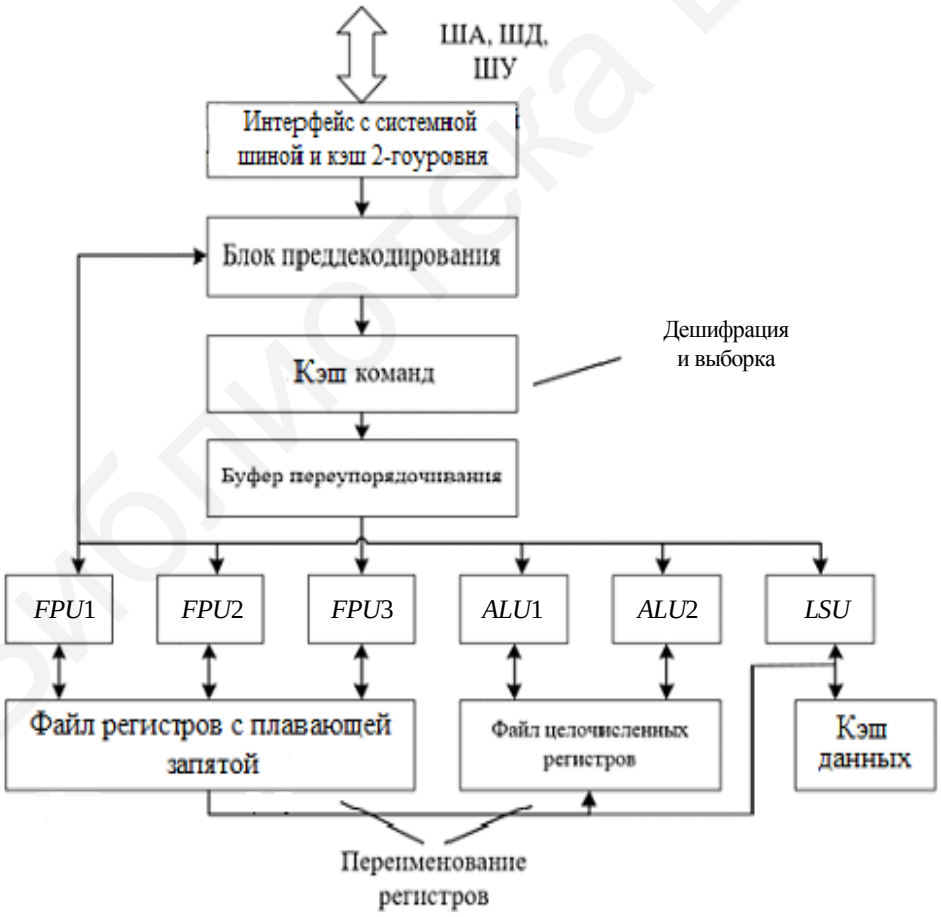


Рисунок 14 – Структура процессора с длинным командным словом *VLIW*

В основном вся тяжесть планирования загрузки большого числа исполнительных устройств в таком процессоре (а у него блочное операционное устройство) ложится на программиста или оптимизирующий компилятор. В процессор поступают уже сформированные триады для всех исполнительных устройств, так что ему только остается выполнять эти длинные команды. В результате он не ограничен размером окна исполнения, так как и программист, и компилятор видят весь код программы и могут извлечь из него максимальный параллелизм.

Такой подход позволяет достичь принципиально более высокой производительности (например, тестирование процессоров *Itanium* с архитектурой *IA*-64, использующей принципы *VLIW*, указывает на 10-кратное ускорение при выполнении ряда вычислений).

Но такие процессоры обладают и рядом недостатков:

- загрузка исполнительных устройств в целом менее эффективная, так как не всегда можно сформировать достаточное количество команд для параллельного исполнения;
- сложности обработки условных переходов;
- сложность программирования и др.

Последнее обстоятельство ограничивает применение процессоров *VLIW*, даже *Intel*, в ПК, так как для этого придется кардинально переписывать все программное обеспечение, поскольку в существующем виде оно не даст прироста производительности на этих процессорах.

Сфера применения *VLIW*-процессоров пока ограничена серверами, производительными рабочими станциями, а также многопроцессорными ЭВМ.

Необходимо отметить, что в суперскалярных процессорах происходит очень сложное и нетривиальное преобразование последовательного потока команд исходной программы в параллельный поток триад «операция + операнды + назначение результата», параллельно продвигающихся по очередям команд в исполнительные устройства.

Процессор ограничен в возможностях такого преобразования, а также в возможностях спекулятивного исполнения (подготовки загрузки альтернативных ветвей ветвления) и прогнозирования ветвлений размером так называемого «окна исполнения», т. е. той частью программного кода, который процессор «видит» в процессе выполнения в данном такте. Все это ограничивает возможности распараллеливания потоков команд до величин порядка 6–8.

1.8 Тенденции развития вычислительной техники

Тенденции развития вычислительной техники следующие:

- переход от отдельных ЭВМ к многофункциональным вычислительным системам и комплексам с широким спектром характеристик, параметров и конфигураций;
- дальнейшее развитие систем телекоммуникаций;
- оптимальная организация использования ресурсов сети Интернет;

- оптимизация характеристик и параметров суперкомпьютеров и ПК;

- разработка и применение микропроцессоров с быстродействием не менее 1000 *MIPS* и встроенной памятью 16 Мбайт;

- разработка и применение встроенных беспроводных сетевых и видеоинтерфейсов;

- разработка и применение тонких крупноформатных дисплеев с высокой разрешающей способностью;

- разработка миниатюрных магнитных дисков огромной емкости, применение которых позволит сделать ненужным стирание старой информации;

- широкое использование широкополосных мультиканальных радио-, волоконно-оптических и оптических каналов обмена информацией между ЭВМ для обеспечения достаточно большой пропускной способности;

- широкое внедрение средств мультимедиа, например видео- и аудио-средств ввода и вывода информации, для решения задач в различных сферах человеческой деятельности;

- дальнейшее развитие компьютерных систем искусственного интеллекта и робототехники для их использования в различных областях человеческой деятельности.

1.9 Содержание задания №1

В таблице 1 представлены номера вопросов, составляющих контрольное задание для каждого варианта.

Таблица 1 – Номера вопросов для каждого из вариантов задания №1

Номер варианта	Номера вопросов	Номер варианта	Номера вопросов
1	15, 32	16	17, 47
2	8, 43	17	34, 56
3	5, 33	18	18, 41
4	9, 40	19	23, 42
5	6, 48	20	25, 37
6	11, 45	21	27, 36
7	30, 59	22	19, 32
8	3, 54	23	20, 38
9	7, 58	24	35, 53
10	4, 44	25	24, 57
11	29, 52	26	21, 51
12	13, 31	27	22, 55
13	2, 46	28	28, 50
14	14, 26	29	12, 39
15	10, 16	30	1, 60

1.10 Вопросы к заданию №1

1 Эволюция вычислительных машин.

2 Неймановская и гарвардская архитектуры, сравнение архитектур.

3 Принципы построения фон-неймановской архитектуры.

4 Поколения ЭВМ.

5 Конвергенция вычислительных систем и систем мобильной связи.

6 Элементы и узлы ЭВМ.

7 Мультиплексор, дешифратор, компаратор, сумматор. Назначение и функциональные возможности.

8 Операция сдвига, счетчик на синхронном триггере.

9 Реализация операции умножения целых чисел.

10 Форматы хранения информации. Системы счисления.

11 Форматы хранения графических изображений. Хранение векторных изображений.

12 Форматы хранения графических изображений. Хранение растровых изображений.

13 Цветовые модели. Модель *RGB*.

14 Цветовые модели. Модель *Lab*.

15 Цветовые модели. Модель *HSB*.

16 Цветовые модели. Модель *CMYK*.

17 Периферийные устройства ЭВМ. Классификация периферийных устройств.

18 Иерархическая система подключения периферийных устройств к ЭВМ.

19 Периферийные устройства ЭВМ. Мониторы, классификация, назначение и принципы функционирования.

20 Периферийные устройства ЭВМ. Принтеры, классификация, назначение и принципы функционирования.

21 Периферийные устройства ЭВМ. Клавиатура, назначение и принципы функционирования.

22 Периферийные устройства ЭВМ. Манипулятор «мышь», назначение и принципы функционирования.

23 Периферийные устройства ЭВМ. Сканер, назначение и принципы функционирования.

24 Периферийные устройства ЭВМ. *USB*-накопитель на флэш-памяти, назначение и принципы функционирования.

25 Периферийные устройства ЭВМ. Цифровая камера, назначение и принципы функционирования.

26 Периферийные устройства ЭВМ. Микрофон и веб-камера, назначение и принципы функционирования.

27 Порты ЭВМ, их классификация, назначение и функциональные возможности.

28 Интерфейс компьютерной системы. Системная шина, назначение и принципы функционирования.

29 Интерфейс компьютерной системы. Локальная шина, назначение и принципы функционирования.

30 Формы представления данных. Форматы данных фиксированной длины.

31 Формы представления данных. Форматы данных переменной длины.

32 Семейство процессоров *Intel*. Микропроцессор *i*8080, архитектура и функциональные возможности.

33 Семейство процессоров *Intel*. Микропроцессор *i*8088, архитектура и функциональные возможности.

34 Семейство процессоров *Intel*. Микропроцессор *i*8086, архитектура и функциональные возможности.

35 Семейство процессоров *Intel*. Микропроцессор *i*80286, архитектура и функциональные возможности.

36 Семейство процессоров *Intel*. Микропроцессор *i*80386, архитектура и функциональные возможности.

37 Семейство процессоров *Intel*. Микропроцессор *i*80386*SX*, архитектура и функциональные возможности.

38 Семейство процессоров *Intel*. Микропроцессор *i*80386*DX*, архитектура и функциональные возможности.

39 Семейство процессоров *Intel*. Микропроцессор *i*486*DX*, архитектура и функциональные возможности.

40 Семейство процессоров *Intel*. Микропроцессор *i*486*SX*, архитектура и функциональные возможности.

41 Семейство процессоров *Intel*. Микропроцессор *i*80486*SX*, архитектура и функциональные возможности.

42 Семейство процессоров *Intel*. Микропроцессор *i*80386*SL*, архитектура и функциональные возможности.

43 Семейство процессоров *Intel*. Математический сопроцессор *i*80387*SL*, архитектура и функциональные возможности.

44 Конвейерное выполнение команды в процессоре ЭВМ. Блочное операционное устройство, назначение, архитектура и функциональные возможности.

45 Конвейерное выполнение команды в процессоре ЭВМ. Конвейерное операционное устройство, назначение, архитектура и функциональные возможности.

46 Архитектура системы команд *RISC*-процессоров, назначение и функциональные возможности.

47 Архитектура системы команд *CISC*-процессоров, назначение и функциональные возможности.

48 Центральный процессор вычислительной машины, назначение и принципы функционирования.

49 Суперскалярные архитектуры процессоров ЭВМ, назначение и функциональные возможности.

50 Процессор с длинным командным словом *VLIW*, назначение, структурные решения и функциональные возможности.

51 Алгоритм сжатия рисунков, функциональные возможности.

52 Коды *ASC*II и *Unicode* для символов, назначение и функциональные возможности.

53 Многоядерные вычислительные системы, назначение, архитектура и функциональные возможности.

54 Кластерные системы, назначение, архитектура и функциональные возможности.

55 Видеокарта, назначение, архитектура и функциональные возможности.

56 Видеопроцессор, назначение, архитектура и функциональные возможности.

57 Технология *DMA*, назначение и функциональные возможности.

58 Процессор *Itanium*, назначение, архитектура и функциональные возможности.

2 Задание №2

Краткие сведения по теоретическим основам организации памяти современных ЭВМ и систем

2.1 Организация памяти вычислительной системы

2.1.1 Виды памяти вычислительной системы

Компьютерная память (устройство хранения информации, запоминающее устройство) – часть вычислительной машины, физическое устройство или среда для хранения данных, используемых в вычислениях, в течение определенного времени. Память в вычислительных устройствах имеет иерархическую структуру и обычно предполагает использование нескольких запоминающих устройств, имеющих различные характеристики.

Любая информация может быть измерена в битах и потому, независимо от того, на каких физических принципах и в какой системе счисления функционирует цифровой компьютер (двоичной, троичной, десятичной и т. п.), числа, текстовая информация, изображения, звук, видео и другие виды данных можно представить последовательностями битовых строк или двоичными числами. Это позволяет компьютеру манипулировать данными при условии достаточной емкости системы хранения.

Для современных типов запоминающих устройств характерна следующая закономерность: чем больше емкость, тем ниже стоимость хранения бита, но больше время доступа.

При создании системы памяти ВМ приходится находить компромисс между параметрами стоимости, емкости и быстродействия. Построение памяти осуществляется по иерархическому принципу (рисунок 15).

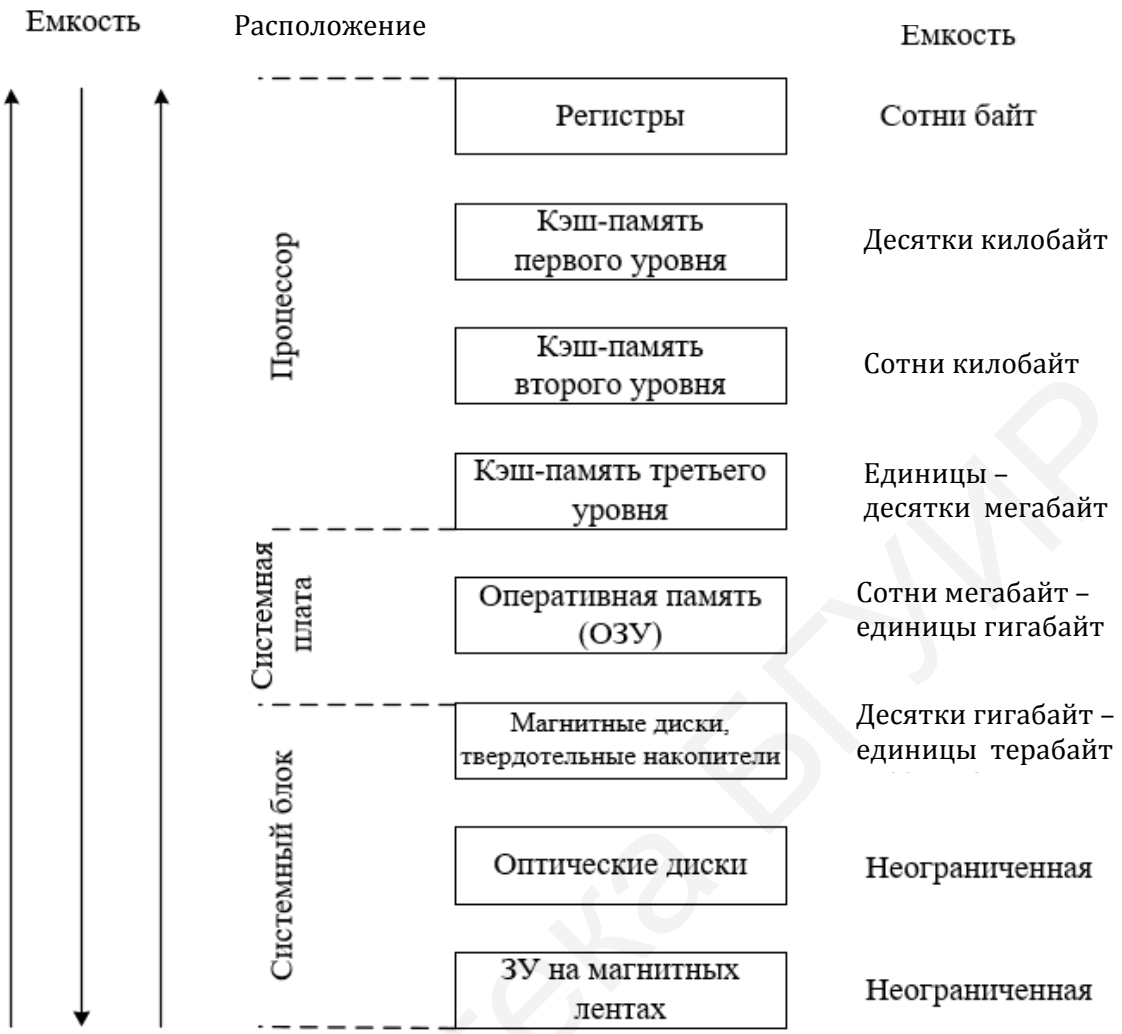


Рисунок 15 – Иерархическая структура средств хранения информации

Техническая реализация памяти в ЭВМ осуществляется с помощью использования запоминающих устройств (ЗУ).

По устойчивости записи и возможности перезаписи ЗУ делятся:

- на постоянные ЗУ (ПЗУ), содержание которых не может быть изменено конечным пользователем (например, *BIOS*). ПЗУ в рабочем режиме допускает только считывание информации;
- записываемые ЗУ (ППЗУ), в которые конечный пользователь может записать информацию только один раз (например, *CD-R*);
- многократно перезаписываемые ЗУ (ПППЗУ) (например, *CD-RW*);
- оперативные ЗУ, обеспечивающие режим записи, хранения и считывания информации в процессе ее обработки.

По типу доступа ЗУ делятся:

- на устройства с последовательным доступом (например, магнитные ленты);
- устройства с произвольным доступом (*RAM*) (например, оперативная память);

- устройства с прямым доступом (например, жесткие магнитные диски);
- устройства с ассоциативным доступом (специальные устройства для повышения производительности).

Энергонезависимая память (от англ. *Non Volatile Random Access Memory – NVRAM*) – перезаписываемая или оперативная память в электронном устройстве, сохраняющая свое содержимое вне зависимости от подачи основного питания на устройство.

В общем случае **энергонезависимая память** – любое устройство или его часть, сохраняющие данные вне зависимости от подачи питающего напряжения.

Постоянное запоминающее устройство (ПЗУ) (от англ. *Read-Only Memory – ROM*) – энергонезависимая память, которая используется для хранения массива неизменяемых данных.

Оперативная память (также оперативное запоминающее устройство, **ОЗУ**) – часть системы памяти ЭВМ, в которую процессор может обратиться за одну операцию (*jump, move* и т. д.). Предназначена для временного хранения данных и команд, необходимых процессору для выполнения им операций. ОЗУ передает процессору данные непосредственно либо через кэш-память. Каждая ячейка оперативной памяти имеет свой индивидуальный адрес.

ОЗУ может изготавливаться как отдельный блок или входить в конструкцию однокристальной ЭВМ или микроконтроллера.

Сегнетоэлектрическая память (*FRAM*) (от англ. *Ferroelectric RAM*) – статическая оперативная память с произвольным доступом, ячейки которой сохраняют информацию, используя сегнетоэлектрический эффект. Ячейка памяти представляет собой две токопроводящие обкладки и пленку из сегнетоэлектрического материала. В центре сегнетоэлектрического кристалла имеется подвижный атом. Приложение электрического поля заставляет его перемещаться. В случае если поле «пытается» переместить атом в положение, например, соответствующее логическому нулю, а он в нем уже находится, то через сегнетоэлектрический конденсатор проходит меньший заряд, чем в случае переключения ячейки. На измерении проходящего через ячейку заряда и основано считывание. При этом процессе ячейки перезаписываются и информация теряется (требуется регенерация). Исследованиями в этом направлении занимаются фирмы *Hitachi* совместно с *Ramtron*, *Matsushita* с фирмой *Symetrix*. По сравнению с флэш-памятью ячейки *FRAM* практически не деградируют – гарантируется до 10^{10} циклов перезаписи.

Память, реализованная на основе ЗУ, – записи, в которых стираются при снятии электропитания. К этому типу памяти относится память, реализованная на ОЗУ, – кэш-память.

2.1.2 Статическая оперативная память с произвольным доступом *SRAM*

Статическая оперативная память с произвольным доступом ***SRAM*** (*Static Random Access Memory – SRAM*) – полупроводниковая оперативная память, строящаяся на триггерах, в которой каждый двоичный или троичный разряд хранится в схеме с положительной обратной связью, позволяющей поддерживать состояние сигнала без постоянной перезаписи. Тем не менее сохранять данные без перезаписи *SRAM* может только пока есть питание, т. е. *SRAM* является энергозависимым типом памяти. Произвольный доступ (*Random Access Memory – RAM*) – возможность выбирать для записи/чтения любой из битов (тритов) (чаще байтов (трайтов)), зависит от особенностей конструкции), в отличие от памяти с последовательным доступом (*Sequental Access Memory – **SAM***).

Двоичная *SRAM*

Типичная ячейка статической двоичной памяти (двоичный триггер) на КМОП-технологии состоит из двух перекрестно (кольцом) включенных инверторов и ключевых транзисторов для обеспечения доступа к ячейке (рисунок 16).

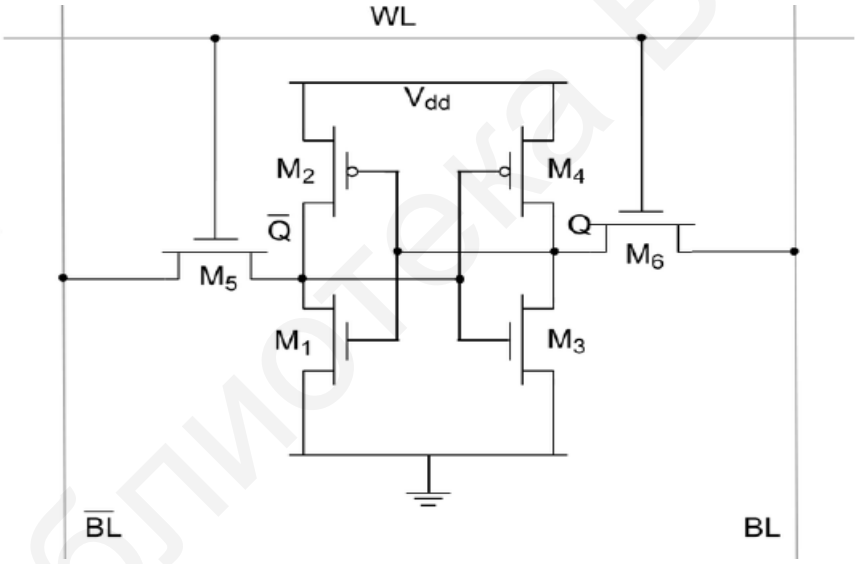


Рисунок 16 – Шеститранзисторная ячейка
статической двоичной памяти (бит) *SRAM*

Часто для увеличения плотности упаковки элементов на кристалле в качестве нагрузки применяют поликремниевые резисторы. Недостатком такого решения является рост статического энергопотребления.

Линия *WL* (*Word Line*) управляет двумя транзисторами доступа.

Линии $\overline{BL}$ и *BL* (*Bit Line*) – битовые линии, использующиеся и для записи данных, и для чтения данных.

Запись. При подаче нуля на линию *BL* или $\overline{BL}$ параллельно включенные транзисторные пары (*M*5 и *M*1) и (*M*6 и *M*3) образуют логические схемы ИЛИ,

последующая подача единицы на линию *WL* открывает транзистор *M*5 или *M*6, что приводит к соответствующему переключению триггера.

Чтение. При подаче единицы на линию *WL* открываются транзисторы *M*5 и *M*6, уровни, записанные в триггере, выставляются на линии *BL* и *BL* и попадают на схемы чтения.

Переключение триггеров через транзисторы доступа является неявной логической функцией приоритетного переключения, которая в явном виде для двоичных триггеров строится на двухвходовых логических элементах 2ИЛИ-НЕ или 2И-НЕ. Схема ячейки с явным переключением является обычным *RS*-триггером. При явной схеме переключения линии чтения и записи разделяются, отпадает нужда в транзисторах доступа (по два транзистора на одну ячейку), но в самой ячейке требуются двухзатворные транзисторы.

В настоящее время есть усовершенствованная схема с отключаемой сигналом записи обратной связью, которая не требует транзисторов нагрузки и соответственно избавлена от высокого потребления энергии при записи.

Троичная *SRAM*

Схема троичной ***SRAM*** на трехразрядных однозначных троичных триггерах показана на рисунке 17.

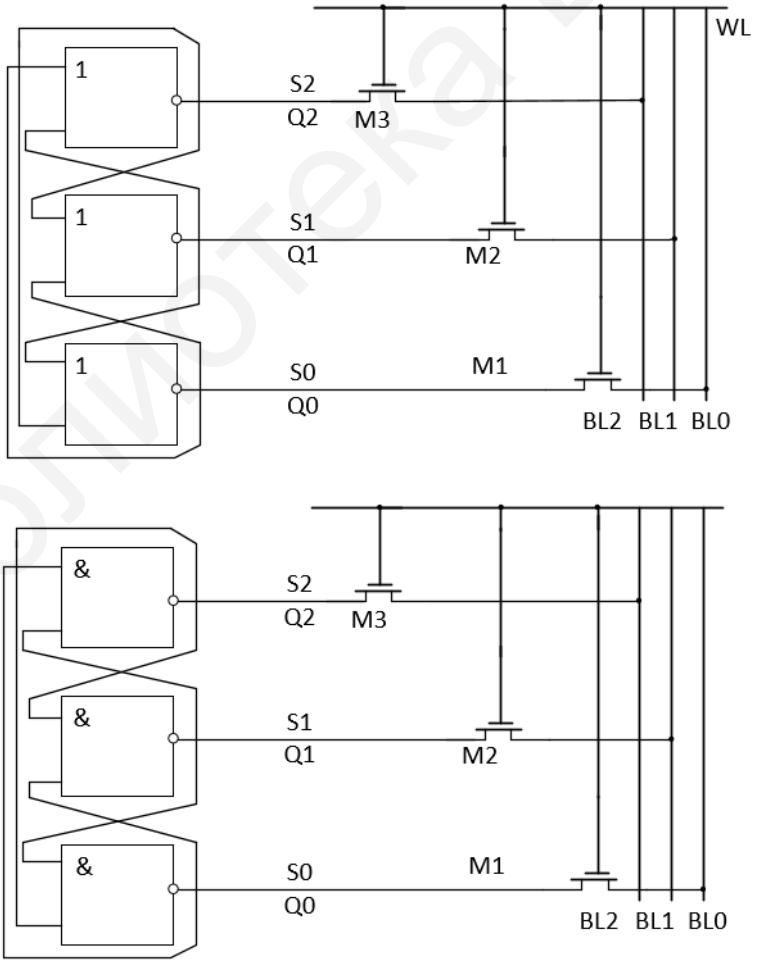


Рисунок 17 – Троичная *SRAM* на трехразрядных однозначных троичных триггерах

Один логический элемент 2ИЛИ-НЕ состоит из двух двухзатворных транзисторов, три – из шести, есть еще три транзистора доступа, т. е. всего девять транзисторов на одну трехразрядную ячейку памяти.

Устройство матрицы статической памяти

Подобно ячейкам динамической памяти триггеры объединяются в единую матрицу, состоящую из строк (*row*) и столбцов (*column*), последние из которых так же называются битами (*bit*).

В отличие от ячейки динамической памяти, для управления которой достаточно всего одного ключевого транзистора, ячейка статической памяти управляется как минимум двумя. Это не покажется удивительным, если вспомнить, что триггер, в отличие от конденсатора, имеет раздельные входы для записи логического нуля и единицы соответственно.

Ячейка статической памяти показана на рисунке 18. Как видно из схемы, на создание одной ячейки используется целых шесть транзисторов: четыре идут, собственно, на сам триггер, а еще два – на управляющие «защелки».

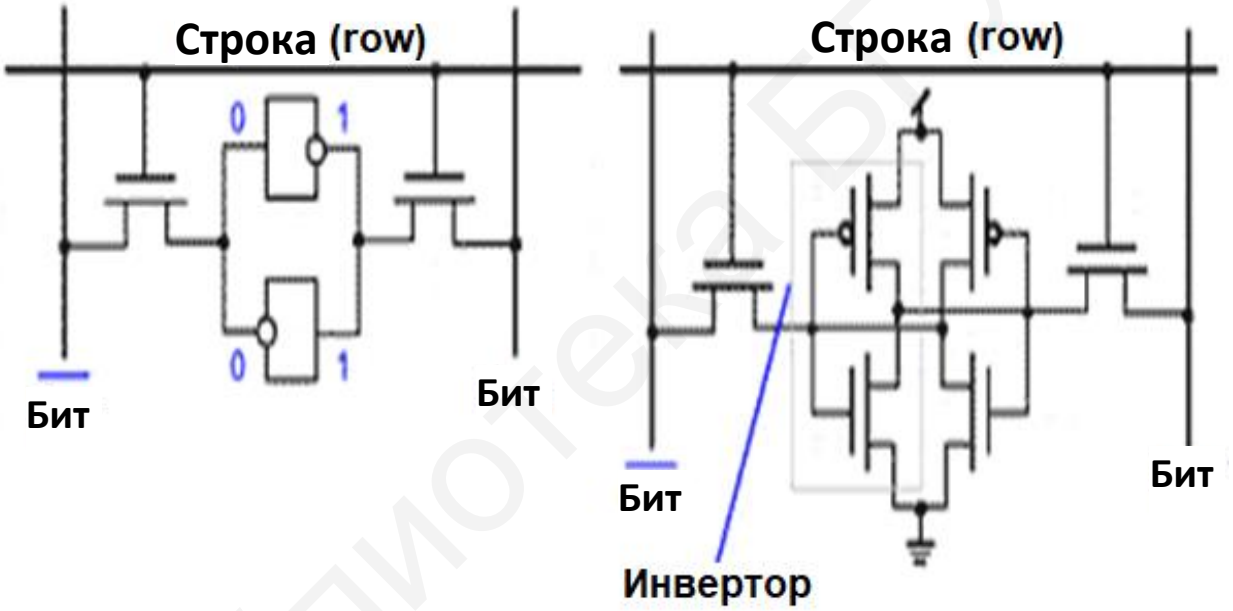


Рисунок 18 – Устройство шеститранзисторной однопортовой ячейки *SRAM*-памяти

Основной недостаток шеститранзисторной ячейки заключается в том, что в каждый момент времени может обрабатываться всего лишь одна строка матрицы памяти. Параллельное чтение ячеек, расположенных в различных строках одного и того же банка, невозможно, равно как невозможно и чтение одной ячейки одновременно с записью другой.

Этого ограничения лишена многопортовая память. Каждая ячейка многопортовой памяти содержит один-единственный триггер, но имеет несколько комплектов управляющих транзисторов, каждый из которых подключен к «своим» линиям *ROW* и *BIT*, благодаря чему различные ячейки матрицы могут обрабатываться независимо. Такой подход намного более прогрессивен, чем деление памяти на банки. Ведь в последнем случае параллелизм достигается

лишь при обращении к ячейкам различных банков, что не всегда выполнимо, а многопортовая память допускает одновременную обработку любых ячеек, избавляя программиста от необходимости вникать в особенности ее архитектуры. Наиболее часто встречается двухпортовая память, устройство ячейки которой изображено на рисунке 19. Для создания одной ячейки двухпортовой памяти расходуется восемь транзисторов! Если емкость кэш-памяти составляет 32 Кбайта, тогда только на одно ядро уйдет свыше 2 млн транзисторов. Реализация динамической ячейки в кристалле представлена на рисунке 20.

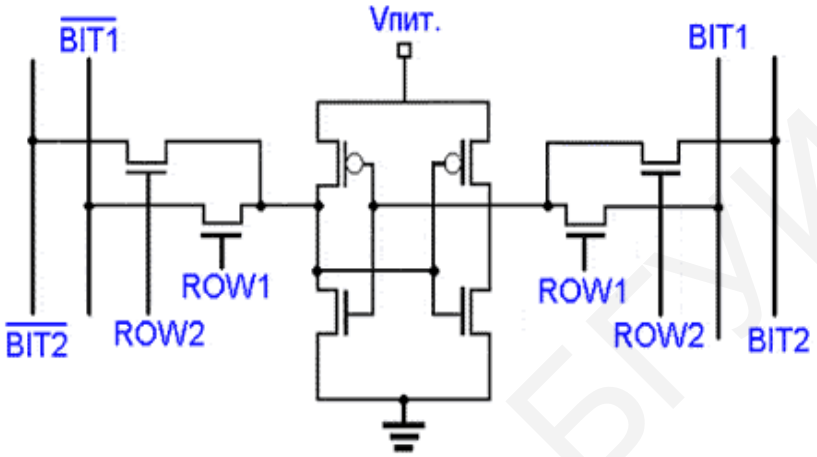


Рисунок 19 – Устройство восьмитранзисторной двухпортовой ячейки *SRAM*-памяти

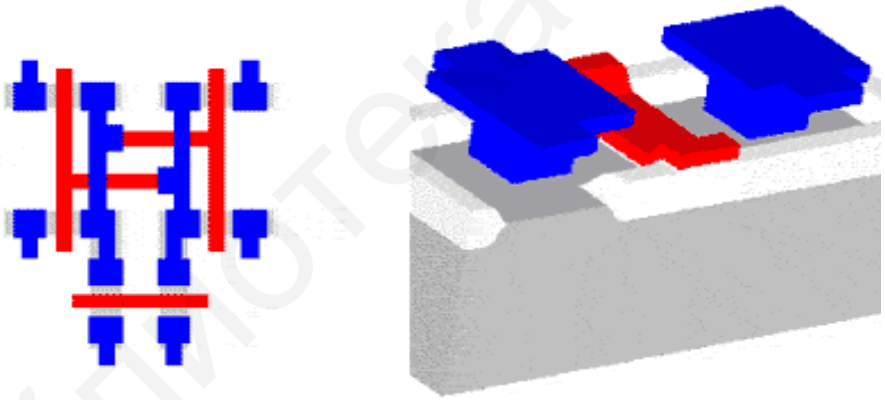


Рисунок 20 – Ячейка динамической памяти, воплощенная в кристалле

Устройство интерфейсной обвязки

По своему устройству интерфейсная обвязка матрицы статической памяти практически ничем не отличается от аналогичной ей обвязки матрицы динамической памяти. Единственное различие в интерфейсах статической и динамической памяти заключается в том, что микросхемы статической памяти, имея значительно меньшую емкость (а следовательно, и меньшее количество адресных линий) и геометрически располагаясь гораздо ближе к процессору, могут не использовать мультиплексирование. Для достижения наивысшей производительности номера строк и столбцов чаще всего передаются одновременно.

Если статическая память выполнена в виде отдельной микросхемы и не располагается непосредственно на кристалле процессора, линии ее входа, как

правило, объединяются с линиями выхода и требуемый режим работы приходится определять по состоянию специального вывода *WE* (*Write Enable*). Высокое состояние вывода *WE* готовит микросхему к чтению данных, а низкое – к записи. Статическую память, выполненную на кристалле процессора, обычно не мультиплексируют, и в этом случае содержимое одной ячейки можно читать параллельно с записью другой, так как линии входа и выхода – раздельные.

Номера столбцов и строк поступают на декодеры столбца и строки соответственно (рисунок 21). После декодирования расшифрованный номер строки поступает на дополнительный декодер, вычисляющий принадлежащую ей матрицу. Оттуда он попадает непосредственно на выборщик строки, который открывает «защелки» требуемой страницы. В зависимости от выбранного режима работы чувствительный усилитель, подсоединенный к битовым линейкам матрицы, либо считывает состояние триггеров соответствующей *raw*-линейки, либо «перещелкает» их согласно записываемой информации.

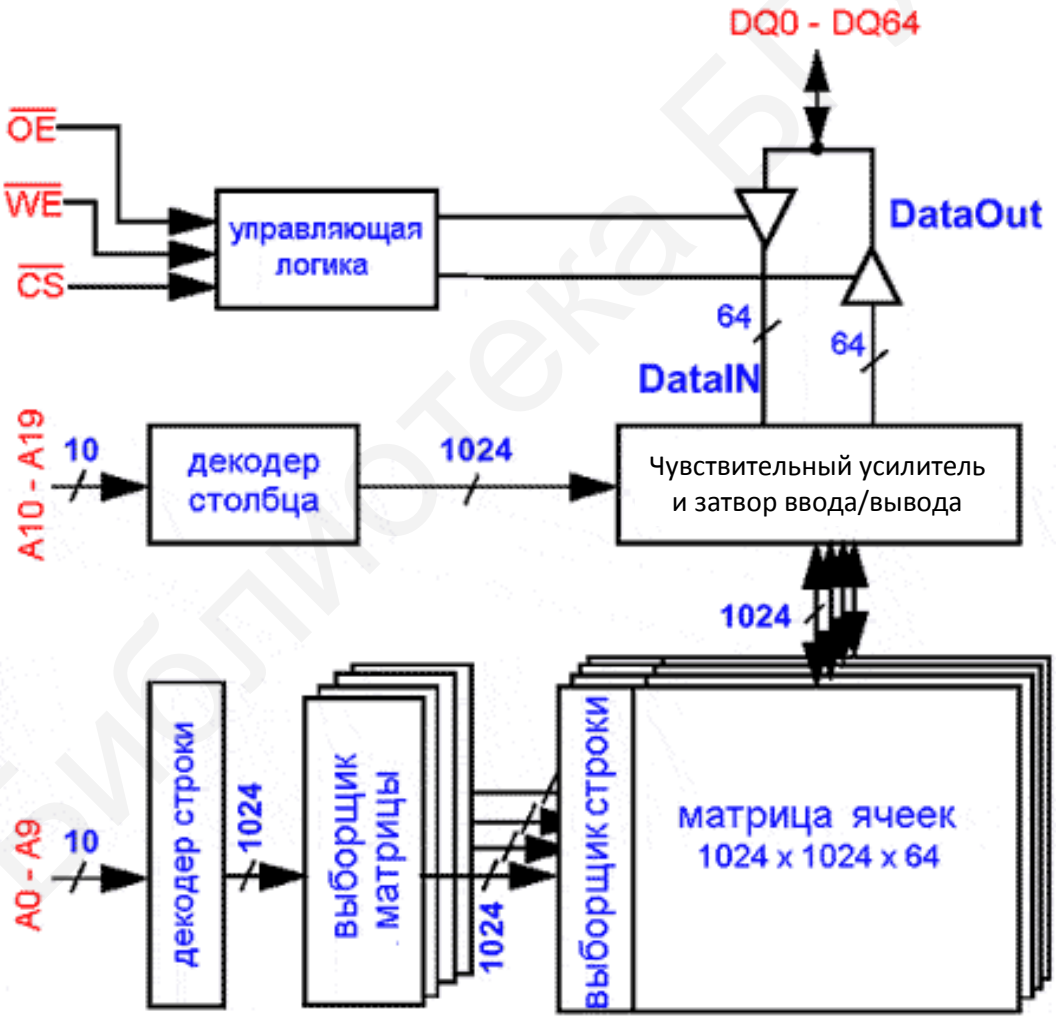


Рисунок 21 – Устройство типовой микросхемы *SRAM*-памяти

Временные диаграммы чтения/записи

Временные диаграммы чтения/записи статической памяти практически ничем не отличаются от аналогичных им диаграмм микросхем динамической памяти (что и неудивительно, так как интерфейсная обвязка в обоих случаях схожа) (рисунок 22).

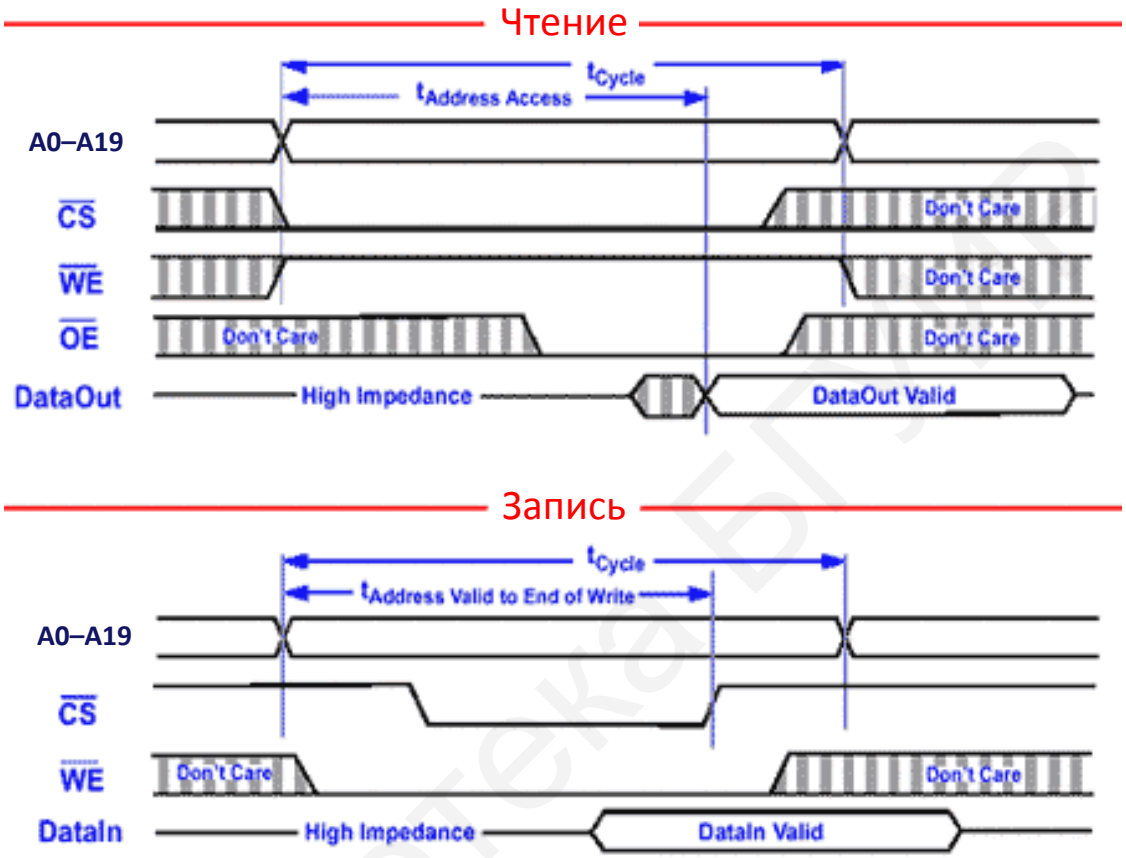


Рисунок 22 – Временные диаграммы чтения/записи асинхронной статической памяти

Цикл чтения

Цикл чтения начинается со сброса сигнала *CS* (*Chip Select* – выбор чипа) в низкое состояние, давая понять тем самым микросхеме, что чип выбран и сейчас с ним будут работать.

К тому моменту, когда сигнал стабилизируется, на адресных линиях должен находиться готовый к употреблению адрес ячейки (т. е. номера строки и столбца), а сигнал *WE* должен быть переведен в высокое состояние (соответствующее операции чтения ячейки). Уровень сигнала *OE* (*Output Enable* – разрешение вывода) не играет никакой роли, так как на выходе пока ничего не содержится, точнее выходные линии находятся в так называемом высокоимпедансном состоянии.

Спустя некоторое время ($t_{Address\ Access}$), определяемое быстродействием управляющей логики и быстротечностью переходных процессов в инверторах, на линиях выхода появляются ожидаемые данные, которые вплоть до оконча-

ния рабочего цикла (t_{Cycle}) могут быть непосредственно считаны. Обычно время доступа к ячейке статической памяти не превышает 1–2 нс.

Цикл записи

Цикл записи происходит в обратном порядке. Сначала выставляется на шину адрес записываемой ячейки, и одновременно с этим сбрасывается сигнал *WE* в низкое состояние. Затем, дождавшись, когда наш адрес декодируется, усилится и поступит на соответствующие битовые линии, сбрасывается *CS* в низкий уровень, указывая микросхеме подать сигнал высокого уровня на требуемую линию *row*. Защелка, удерживающая триггер, откроется, и в зависимости от состояния *bit*-линии триггер переключится в то или иное состояние.

Типы статической памяти

Существует как минимум три типа статической памяти: асинхронная, синхронная и конвейерная. Все они практически ничем не отличаются от соответствующих им типов динамической памяти.

Асинхронная статическая память

Асинхронная статическая память работает независимо от контроллера, и потому контроллер не может быть «уверен», что окончание цикла обмена совпадет с началом очередного тактового импульса. В результате цикл обмена удлиняется по крайней мере на один такт, снижая тем самым эффективную производительность. Следовательно, в настоящее время асинхронная память практически нигде не применяется (последними компьютерами, на которых она еще использовались в качестве кэша второго уровня, были ЭВМ, построенные на базе процессора *Intel* 80386).

Синхронная статическая память

Синхронная статическая память выполняет все операции одновременно с тактовыми сигналами, в результате чего время доступа к ячейке укладывается в один-единственный такт. Именно на синхронной статической памяти реализуется кэш первого уровня современных процессоров.

Конвейерная статическая память

Конвейерная статическая память представляет собой синхронную статическую память, оснащенную специальными «защелками», удерживающими линии данных, что позволяет читать (записывать) содержимое одной ячейки параллельно с передачей адреса другой.

Также конвейерная память может обрабатывать несколько смежных ячеек за один рабочий цикл. Достаточно передать лишь адрес первой ячейки пакета, а адреса остальных микросхема вычислит самостоятельно – необходимо лишь подавать (забирать) записанные (считанные) данные.

За счет большей аппаратной сложности конвейерной памяти, время доступа к первой ячейке пакета увеличивается на один такт, однако это практически не снижает производительности, так как все последующие ячейки пакета обрабатываются без задержек.

Конвейерная статическая память используется, в частности, в кэше второго уровня микропроцессоров *Pentium II*, и ее формула выглядит так: 2 – 1 – 1 – 1.

Простейшим примером организации памяти является постоянное запоминающее устройство. В такой памяти с долговременным хранением информации определяющим элементом служат перемычки в двумерном массиве проводников. Изначально все строки и столбцы соединены между собой с помощью перемычек – токопроводящих мостиков. В процессе записи данных в матрицу хранения часть перемычек остается нетронутыми, и они соединяют линии строк и линии столбцов этого массива, а остальные разрушаются.

Замкнутому с помощью перемычки состоянию проводника можно присвоить значение логического нуля, а разомкнутому – логической единицы. Измеряя сопротивление между конкретными линиями строк и столбца, по его высокому значению (разомкнутому состоянию проводников) можно считать логическую единицу, а по короткозамкнутому состоянию – принять решение о хранении там логического нуля. Такая оценка электрических параметров коммутируемого соединения становится формой хранения информации и определяет метод доступа к содержимому заданной ячейки памяти (рисунок 23).

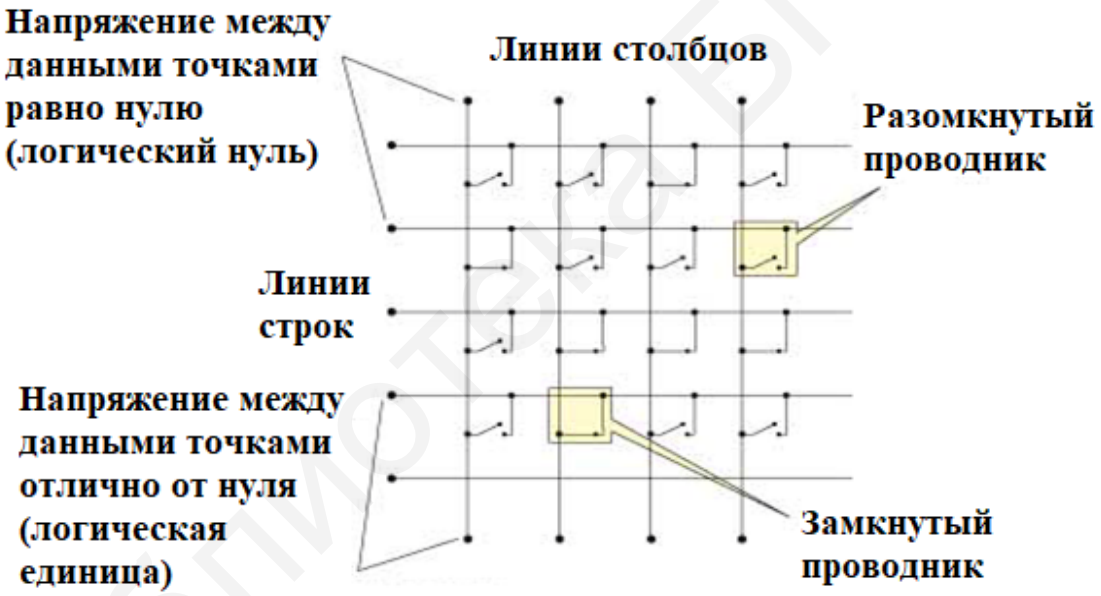


Рисунок 23 – Оценка электрических параметров коммутируемого соединения как форма доступа к содержимому заданной ячейки памяти

Отказ от необратимого способа записи требует пересмотра схем управления запоминающей матрицы, но не отменяет сам принцип хранения информации. Оперативная память использует все тот же механизм доступа по строкам и столбцам, полагаясь на электронные ключи в узлах коммутации вместо тривиальных перемычек, характерных для постоянных запоминающих устройств.

Достоинства SRAM:

- быстрый доступ (*SRAM* – память произвольного доступа, доступ к любой ячейке памяти в любой момент занимает одно и то же время);

- простая схемотехника (*SRAM* не требуются сложные контроллеры);

- возможны очень низкие частоты синхронизации, вплоть до полной остановки синхроимпульсов.

Недостатки SRAM:

- высокое энергопотребление;

- невысокая плотность записи (шесть элементов на бит), вследствие чего – высокая стоимость килобайта памяти.

Тем не менее высокое энергопотребление не является принципиальной особенностью *SRAM* – оно обусловлено высокими скоростями обмена с данным видом внутренней памяти процессора. Энергия потребляется только в момент изменения информации в ячейке *SRAM*.

Применение

SRAM применяется в микроконтроллерах и ПЛИС, в которых объем ОЗУ невелик (единицы килобайт), но требуются низкое энергопотребление (за счет отсутствия сложного контроллера динамической памяти), предсказываемое с точностью до такта время работы подпрограмм и отладка прямо на устройстве. ***SRAM*** реализуется в виде регистров и кэш-памяти.

2.1.3 Динамическая оперативная память с произвольным доступом *DRAM*

***DRAM* (*Dynamic Random Access Memory*)** – тип энергозависимой полупроводниковой памяти с произвольным доступом (*RAM*), также запоминающее устройство, наиболее широко используемое в качестве ОЗУ большого объема современных компьютеров.

Физически память *DRAM* представляет собой набор запоминающих ячеек, которые состоят из конденсаторов и транзисторов, расположенных внутри полупроводниковых микросхем. Совокупность ячеек такой памяти образует условный «прямоугольник», состоящий из определенного количества строк и столбцов. Один такой «прямоугольник» называется *страницей*, а совокупность страниц – *банком*. Весь набор ячеек условно делится на несколько областей.

DRAM-память – это модуль различных конструктивов, состоящий из электрической платы, на которой расположены микросхемы памяти и разъем, необходимый для подключения модуля к материнской плате.

При отсутствии подачи электроэнергии к памяти этого типа происходит разряд конденсаторов, и память опустошается (обнуляется). Для поддержания необходимого напряжения на обкладках конденсаторов ячеек и сохранения их содержимого необходимо периодически подзаряжать конденсаторы, прилагая к ним напряжение через коммутирующие транзисторные ключи.

Такое динамическое поддержание заряда конденсатора является основополагающим принципом работы памяти типа *DRAM*. Конденсаторы заряжают в случае, когда в ячейку записывается единичный бит, и разряжают в случае, когда в ячейку необходимо записать нулевой бит.

Важным элементом памяти этого типа является чувствительный усилитель (от англ. *sense amp*), подключенный к каждому из столбцов «прямоуголь-

ника». Он, реагируя на слабый поток электронов, движущихся через открытые транзисторы с обкладок конденсаторов, считывает всю страницу целиком. Именно страница является минимальной порцией обмена динамической памятью, так как обмен данными с отдельно взятой ячейкой невозможен.

В отличие от статической памяти типа *SRAM* (от англ. S*tatic Random Access Memory*), которая является конструктивно более сложным и более дорогим типом памяти и используется в основном в кэш-памяти, память *DRAM* изготавливается на основе конденсаторов небольшой емкости, которые быстро теряют заряд, поэтому информацию приходится обновлять через определенные промежутки времени во избежание потерь данных. Этот процесс называется **регенерацией памяти**. Он реализуется специальным контроллером, установленным на материнской плате или же на кристалле центрального процессора. На протяжении времени, называемого шагом регенерации, в *DRAM* перезаписывается целая строка ячеек, и через 8–64 мс обновляются все строки памяти.

Процесс регенерации памяти в классическом варианте существенно тормозит работу системы, поскольку в это время обмен данными с памятью невозможен. Регенерация, основанная на обычном переборе строк, не применяется в современных типах *DRAM*. Существует несколько более экономичных вариантов этого процесса – расширенный, пакетный, распределенный; наиболее экономичной является *скрытая* (*теневая*) *регенерация.*

К ЗУ относятся **триггеры** – класс электронных устройств, обладающих способностью длительно находиться в одном из двух или более устойчивых состояний и чередовать их под воздействием внешних сигналов. Каждое состояние триггера легко распознается по значению выходного напряжения.

По характеру действия триггеры относятся к импульсным устройствам – их активные элементы (например, транзисторы) работают в ключевом режиме, а смена состояний длится очень короткое время.

ОЗУ, построенное на триггерах, называется *статической памятью с произвольным доступом, или просто статической памятью*.

Достоинство этого вида памяти – скорость. Поскольку триггеры собраны на вентилях, а время задержки вентиля очень мало, то и переключение состояния триггера происходит очень быстро. Поэтому данное решение используется для сверхбыстрого ОЗУ.

К недостаткам данного вида памяти можно отнести:

- транзисторы, входящие в состав триггера, обходятся дороже, даже если они вытравляются миллионами на одной кремниевой подложке;

- группа транзисторов занимает гораздо больше места, поскольку между транзисторами, которые образуют триггер, должны быть вытравлены линии связи.

Триггер по сравнению с конденсатором имеет следующие преимущества:

- состояния триггера устойчивы и при наличии питания могут сохраняться бесконечно долго, в то время как конденсатор требует периодической регенерации;

- триггер, обладая мизерной инертностью, надежно работает на частотах вплоть до нескольких гигагерц, тогда как конденсаторы имеют частотный предел уже на 75–100 МГц.

К недостаткам триггеров следует отнести их высокую стоимость и низкую плотность хранения информации. Если для создания ячейки динамической памяти достаточного всего одного транзистора и одного конденсатора, то ячейка статической памяти состоит как минимум из четырех, а в среднем – шести-восьми транзисторов, поэтому мегабайт статической памяти оказывается по меньшей мере в несколько раз дороже.

Работа ячейки *DRAM*

Одной из широко распространенных реализаций оперативных запоминающих устройств является ячейка с динамическим способом хранения данных – *DRAM*. Для ячейки динамического запоминающего устройства необходимы два элемента: управляющий транзистор, выполняющий функции коммутирующего ключа, и конденсатор, хранящий заряд. Именно по его состоянию и можно принять решение, хранит ячейка логическую единицу (конденсатор заряжен) или нуль (разряжен). Главная особенность такого подхода – постоянная забота о регенерации конденсатора, так как электрический заряд на его пластинах имеет свойство со временем рассеиваться или, как говорят, «стекать».

Если рассматривать внутреннюю структуру кристалла динамической памяти (рисунок 24), то в контексте однобитной его организации сложилась терминологическая традиция: горизонтальные строки запоминающей матрицы считать (условно) *wordline*, где слово *word* подразумевает все биты строки; тогда *bitline* – это столбец (вертикальная строка) матричной организации памяти. Такая абстракция хорошо иллюстрирует механизм произвольного доступа, когда битовые линии образуют шину данных, а горизонтальные строки обеспечивают запись и чтение адресуемого слова. Как мы увидим ниже, схемотехника модуля памяти и его обработка центральным процессором имеют более сложную архитектуру.

Чтение ячейки, сформированной парой «транзистор – конденсатор», деструктивно по своей природе: выборка информации разряжает конденсатор в процессе определения его заряда. Вследствие этого *DRAM*-ячейки представляют собой короткоживущую память, да к тому же и разового действия. Во избежание потери информации считанный бит информации необходимо сразу же «освежить в памяти» – перезаписать заново. Это входит в обязанности самой микросхемы *DRAM* под четким контролем контроллера памяти.

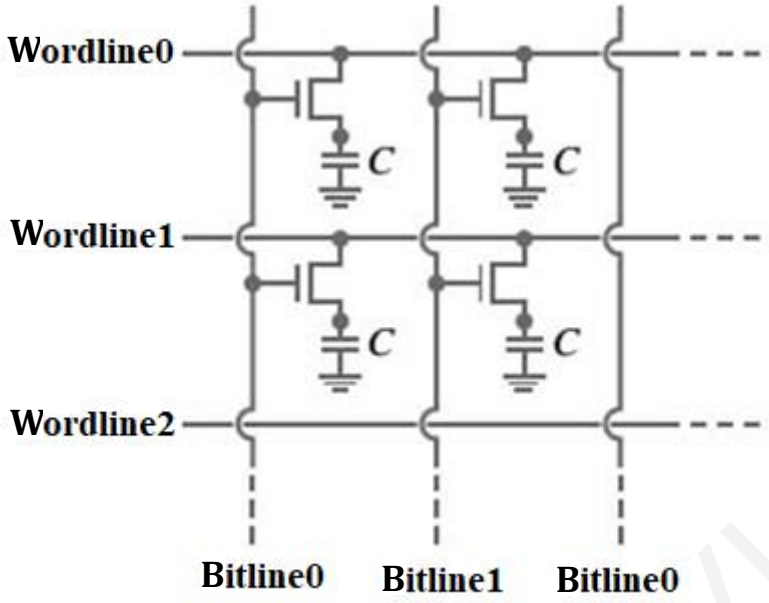


Рисунок 24 – Внутренняя структура кристалла *DRAM* в контексте однобитной его организации

Фактор короткоживущей памяти – еще одна проблема в архитектуре *DRAM*. Валидность информации в памяти компьютера зависит от того, как долго конденсатор сможет хранить заряд. Саморазряд, причиной которого является миниатюрность конденсатора на подложке управляющего транзистора, приводит к тому, что достоверность данных ограничена тысячными долями секунды. Регенерация этого процесса делегируется соответствующим схемам обслуживания, которые вынуждены периодически считывать информацию из ячеек с последующей их перезаписью.

Схемы регенерации претерпели существенные изменения в процессе эволюции персональной платформы. Ранее регенерация оперативной памяти тактировалась тиками системного таймера с частотой, равной 18,2 Гц. Современные кристаллы динамической памяти содержат схемы регенерации внутри самой микросхемы. Это позволяет сократить время ожидания валидных данных. Чип до очередного цикла регенерации копирует содержимое обновляемой строки в буфер, доступный контроллеру памяти, параллельно выполняя *Refresh* ячеек. Такая организация получила название *синхронной динамической памяти*, или *SDRAM*.

Во время выполнения циклов регенерации, запоминающая матрица недоступна для операций чтения и записи, и это обстоятельство неизбежно сказывается на производительности. Статические элементы оперативной памяти (***SRAM***), в узлах которой используются более сложные схемы на основе триггеров с двумя устойчивыми состояниями, лишены этого недостатка и обладают лучшим быстродействием. Эффективным компромиссом производительности и стоимости является использование оперативного запоминающего устройства на основе *DRAM* в сочетании с кэш-памятью небольшого размера на основе *SRAM*.

В кэш-память оперативно загружаются данные, скорость доступа к которым наиболее критична для обеспечения производительности.

Аппаратная реализация модуля *DRAM*, с которым оперирует контроллер памяти, также базируется на интеграции ячеек, хранящих один или несколько байтов информации, – в прямоугольную матрицу. Ее горизонтальные линии образуют *row* (строки), а вертикальные – *column* (столбцы). Итак, в узлах запоминающей матрицы полупроводниковый прибор (транзистор), находясь в закрытом состоянии, предотвращает потерю конденсатором очень важного кванта информации – заряда. Но стоит подать на заданную строку запоминающей матрицы специальный импульс, транзисторные ключи всех узлов строки сработают. Открывшись, каждый из них соединит свой конденсатор с конкретным столбцом. Схема детектирования, мониторящая состояние столбцов матрицы, отреагирует на возникший ток и считает всю строку целиком (в упомянутый выше транзитный буфер, который принято называть страницей (*page*)).

Доступ к отдельно взятой ячейке динамической памяти невозможен, в отличие от простейшего ПЗУ, где чтение данных могло выполняться по конкретному адресу. В данном случае запрос информации по одной строке матрицы откроет все ключи и приведет к считыванию состояния всех, подключенных к этой линии конденсаторов.

Современная подсистема памяти использует 64-битную шину данных. Она осталась за кадром, уступив место двумерной адресации 64-битных ячеек (рисунок 25).

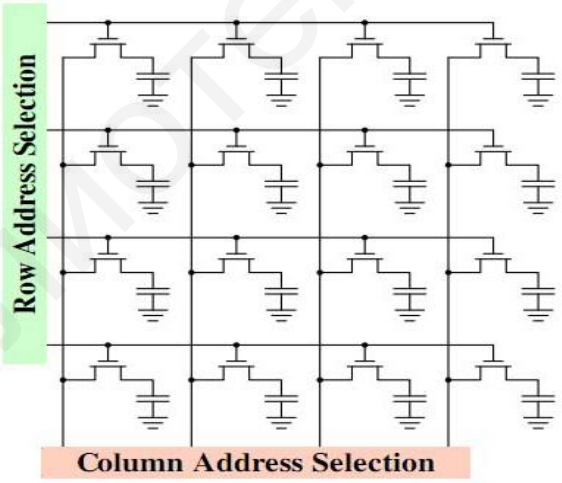


Рисунок 25 – Двумерная адресация ячеек динамической памяти по строкам и столбцам

Для одномерной их адресации в современных условиях пришлось бы использовать более сорока адресных линий, что невероятно накладно для конструкции компьютерной платформы и к тому же негативно сказалось бы на производительности ОЗУ. Раздельная адресация по строкам и столбцам позволяет оперировать со страницами памяти по адресной шине меньшей размерности. Для этого, правда, приходится выставлять адрес требуемой страницы посредством двух посылок, т. е. дважды.

За адресацию ОЗУ отвечает контроллер памяти, который с недавних пор перекочевал из чипсета системной логики в центральный процессор. Ему в этом помогают схемы синхронизации, формирующие дополнительные сигналы управления *RAS* (*Row Access Strobe*) и *CAS* (*Column Access Strobe*).

Запоминающая шина банка динамической памяти и временная диаграмма ее работы представлены соответственно на рисунках 26 и 27.

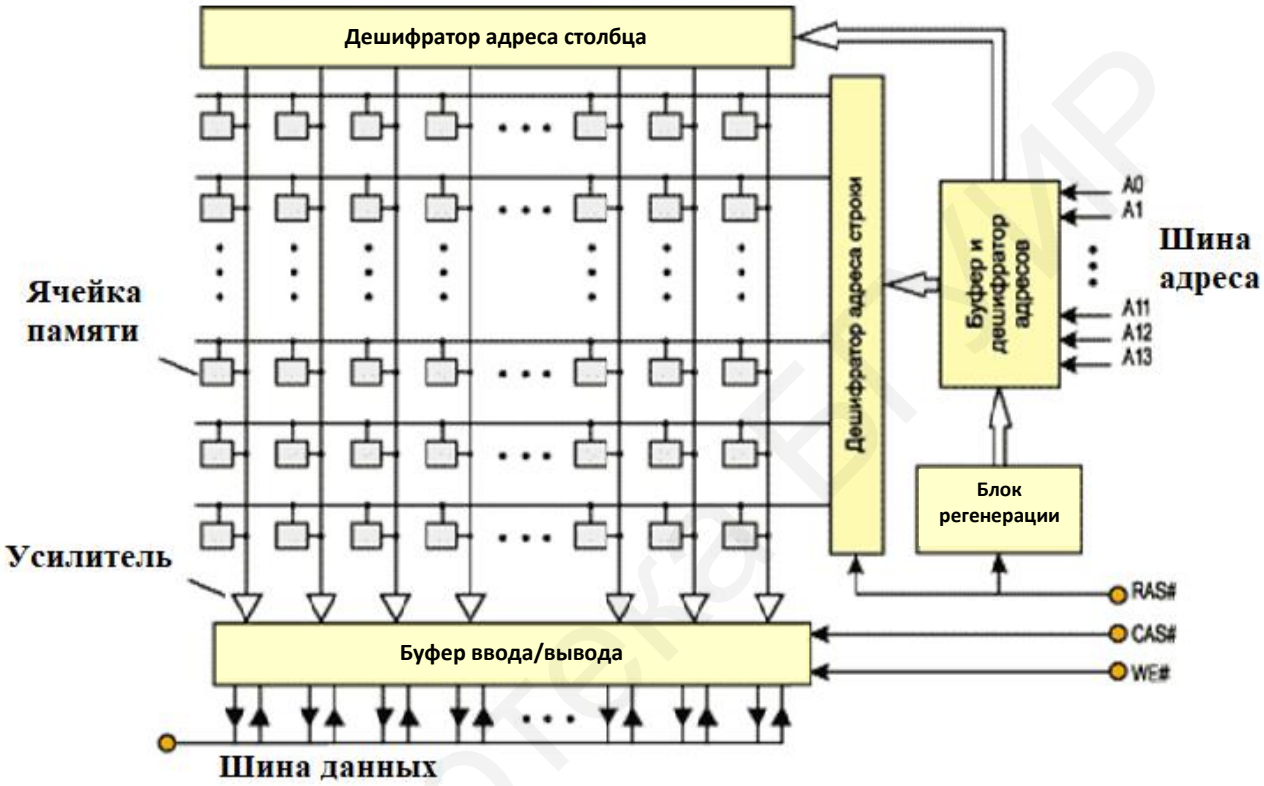


Рисунок 26 – Запоминающая матрица банка динамической памяти

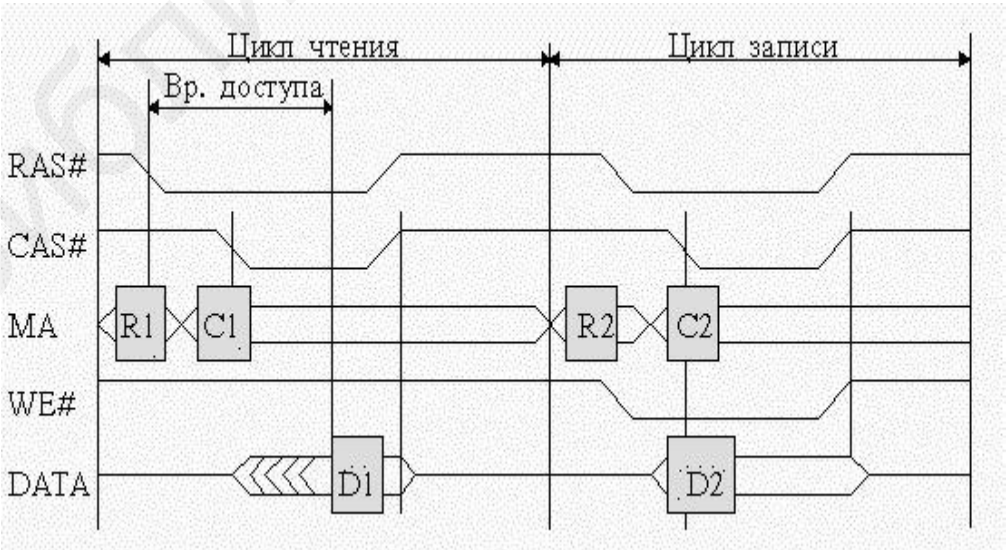


Рисунок 27 – Временная диаграмма работы классической запоминающей матрицы динамической памяти

Сначала на адресную шину выставляется адрес строки, затем – столбца. В классической модели *SDRAM* первая из посылок выполняется по спаду (заднему фронту) тактирующего импульса *RAS*, вторая – по спаду *CAS*. После этого в течение заданного времени контроллер может оперировать данными в оперативной памяти (читать их или писать).

Окно возможностей закрывается с началом упомянутого уже регулярного цикла регенерации (*Memory Refresh*). Для поддержания сохранности данных необходимо выполнить обход ячеек памяти, освежив в них информацию холостыми циклами перезаписи. Регенерация памяти происходит сразу по всей строке в силу транзисторной архитектуры матрицы. Оперативная память видеоадаптера специальных циклов регенерации не требует: формирование развертки для вывода на экран изображения создает условия, при которых периодичность обращения к памяти обеспечивает своевременную перезарядку запоминающего конденсатора.

Даже при наличии буфера обмена чип динамической памяти не в состоянии обеспечить мгновенный доступ к затребованной контроллером информации. Чтобы избежать пауз, создаваемых операцией *Refresh*, современные микросхемы *DRAM* поддерживают несколько независимых запоминающих матриц (они называются *банками памяти*). Обмен данными выполняется с минимальными задержками, если требуемая информация хранится в разных банках одного и того же чипа.

Страничная память *DRAM*

Страничная память (от англ. *pagemode DRAM, PMDRAM*) является одним из первых типов выпускаемой компьютерной оперативной памяти. Память такого типа выпускалась в начале 1990-х годов, но с ростом производительности процессоров и ресурсоемкости приложений требовалось увеличивать не только объем памяти, но и скорость ее работы.

Быстрая страничная память *DRAM* – данный тип памяти в основном применялся для компьютеров с процессорами *Intel* 80486 или аналогичными процессорами других фирм. Память могла работать на частотах 25 и 33 МГц с временем полного доступа 70 и 60 нс и с временем рабочего цикла 40 и 35 нс соответственно.

2.1.4 Кэш-память

Кэш (от англ. *cache*) – промежуточный буфер с быстрым доступом, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Доступ к данным в кэше идет быстрее, чем выборка исходных данных из ОЗУ и быстрее внешней (жесткий диск или твердотельный накопитель) памяти, за счет чего уменьшается среднее время доступа и увеличивается общая производительность компьютерной системы.

Ряд моделей центральных процессоров (ЦП) обладает собственным кэшем, для того чтобы минимизировать доступ к **ОЗУ**, которая медленнее, чем регистры.

Кэш-память может давать значительный выигрыш в производительности в случае, когда тактовая частота ОЗУ значительно меньше тактовой частоты ЦП. Тактовая частота для кэш-памяти обычно ненамного меньше частоты ЦП.

Кэш центрального процессора разделен на несколько уровней, для универсальных процессоров – до трех.

Кэш-память уровня N+1, как правило, больше по размеру и медленнее по скорости обращения и передаче данных, чем кэш-память уровня N.

Самой быстрой памятью является кэш первого уровня – *L*1-*кэш*. Она расположена на одном с процессором кристалле, входит в состав функциональных блоков и состоит из кэша команд и кэша данных. Некоторые процессоры без *L*1-кэша не могут функционировать.

*L*1-кэш работает на частоте процессора, и в общем случае обращение к нему может производиться каждый такт, т. е. имеется возможность выполнять несколько чтений/записей одновременно.

Латентность доступа обычно равна 2−4 тактам ядра, а объем памяти составляет не более 128 Кбайт.

Вторым по быстродействию является *L*2-*кэш* – кэш второго уровня, который обычно расположен либо на кристалле, как и *L*1, либо рядом с ядром, например, в процессорном картридже (только в слотовых процессорах). Объем *L*2-*кэша* – от 128 Кбайт до 1−12 Мбайт.

В современных многоядерных процессорах кэш второго уровня, находясь на том же кристалле, является памятью раздельного пользования: при общем объеме кэша в 8 Мбайт на каждое ядро приходится по 2 Мбайта. Обычно латентность *L*2-кэша, расположенного на кристалле ядра, составляет от 8 до 20 тактов ядра. В отличие от *L*1-кэша, его отключение может не повлиять на производительность системы. Однако в задачах, связанных с многочисленными обращениями к ограниченной области памяти, например СУБД, производительность может упасть в десятки раз.

Кэш третьего уровня наименее быстродействующий и обычно расположен отдельно от ядра ЦП, но его размер может быть более 32 Мбайт.

*L*3-кэш медленнее предыдущих кэшей, но все равно значительно быстрее, чем ОЗУ.

Диаграмма кэш-памяти ЦПУ представлена на рисунке 28.

Функционирование кэш-памяти осуществляется в следующей последовательности:

- когда клиент кэша (ЦПУ, веб-браузер, операционная система) обращается к данным, то прежде всего исследуется кэш;

- если в кэше найдена запись с идентификатором, совпадающим с идентификатором затребованного элемента данных, то используются элементы данных в кэше;

- если в кэше не найдена запись, содержащая затребованный элемент данных, то он читается из основной памяти в кэш и становится доступным для последующих обращений.

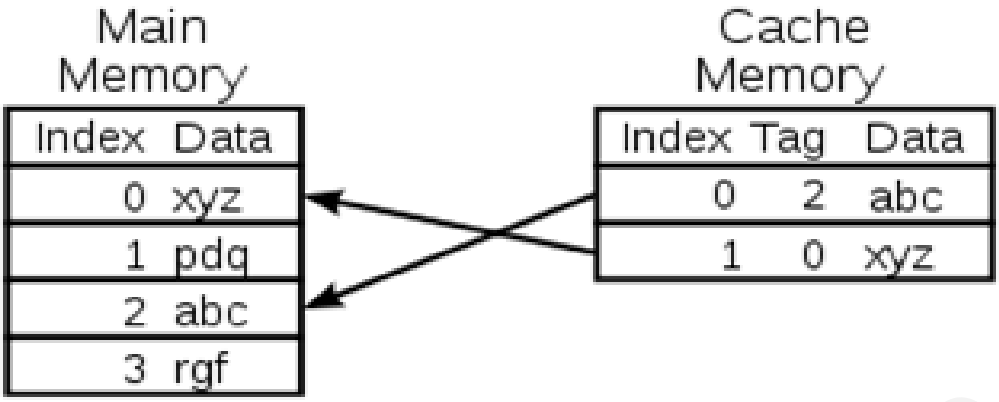


Рисунок 28 – Диаграмма кэш-памяти ЦПУ

2.2 Содержание задания №2

В таблице 2 представлены номера вопросов, составляющих контрольное задание для каждого варианта.

Таблица 2 – Номера вопросов для каждого из вариантов задания №2

Номер варианта	Номера вопросов	Номер варианта	Номера вопросов
1	15, 32	16	17, 47
2	8, 43	17	34, 56
3	5, 33	18	18, 41
4	9, 40	19	23, 42
5	6, 48	20	25, 37
6	11, 45	21	27, 36
7	30, 59	22	19, 32
8	3, 54	23	20, 38
9	7, 58	24	35, 53
10	4, 44	25	24, 57
11	29, 52	26	21, 51
12	13, 31	27	22, 55
13	2, 46	28	28, 50
14	14, 26	29	12, 39
15	10, 16	30	1, 60

2.3 Вопросы к заданию №2

1 Компьютерная память. Определение, параметры памяти, иерархический принцип построения памяти.

2 Запоминающие устройства (ЗУ). Назначение, классификация, функциональные возможности.

3 Энергозависимая, энергонезависимая память. Назначение, функциональные возможности.

4 Постоянное и оперативное запоминающие устройства. Назначение, функциональные возможности.

5 Сегнетоэлектрическая память. Назначение, функциональные возможности.

6 Статическая оперативная память с произвольным доступом *SRAM*. Назначение, функциональные возможности.

7 Шеститранзисторная ячейка статической двоичной памяти *SRAM*. Назначение, принципы функционирования.

8 Триггеры. Назначение, функциональные возможности, принципы функционирования.

9 Устройство матрицы статической памяти. Назначение, принципы функционирования.

10 Аппаратная реализация памяти *SRAM*. Примеры.

11 Устройство типовой микросхемы *SRAM*-памяти. Назначение, принципы функционирования.

12 Типы статической памяти. Назначение, функциональные возможности.

13 Области применения памяти *SRAM*.

14 Динамическая оперативная память с произвольным доступом *DRAM*. Назначение, функциональные возможности.

15 Описание работы ячейки *DRAM*.

16 Внутренняя структура кристалла *DRAM*. Состав, принципы функционирования.

17 Синхронная динамическая память. Назначение, функциональные возможности.

18 Схема двумерной адресации ячеек динамической памяти по строкам и столбцам. Состав, принципы функционирования.

19 Схема запоминающей матрицы банка динамической памяти. Состав, принципы функционирования.

20 Описание временной диаграммы работы классической запоминающей матрицы динамической памяти.

21 Страничная память *DRAM*. Назначение, функциональные возможности.

22 Кэш-память. Назначение, функциональные возможности.

23 Описание диаграммы кэш-памяти процессора.

24 Описание алгоритма работы кэш-памяти процессора.

25 Регистры. Назначение, принципы функционирования.

26 Описание характеристик памяти ЭВМ.

27 Описание принципа локализации памяти для выполнения приложения.

28 Основная память ЭВМ. Конструктивное исполнение ОЗУ.

29 Основная память ЭВМ. Конструктивное исполнение ПЗУ.

30 Внешняя память ЭВМ. Назначение и принципы работы накопителя на жестком магнитном диске.

31 Внешняя память ЭВМ. Назначение и принципы работы накопителя на оптическом диске.

32 Внешняя память ЭВМ. Назначение и принципы работы накопителя на магнито-оптическом диске.

33 Запоминающая матрица банка динамической памяти.

34 Внешняя память ЭВМ. Назначение и принципы работы накопителя на магнитной ленте.

35 Аппаратная реализация памяти *DRAM*. Примеры.

36 Аппаратная организация ОЗУ ЭВМ. Примеры.

37 Аппаратные реализации обмена с памятью *SRAM*. Примеры.

38 *BIOS.* Назначение, функциональные возможности и принципы работы.

39 Описание организации кэш-памяти ЭВМ.

40 Описание методов ускорения обмена с памятью *DRAM.*

41 Описание режимов обмена с основной памятью ЭВМ.

42 Флэш-память. Назначение и принципы работы.

43 Флэш-память. Конструктивное исполнение.

44 Реализации сегментной памяти.

45 Реализации страничной памяти.

46 Видеопамять. Назначение и функциональные возможности.

47 Что такое контроллер прямого доступа к памяти (*DMA*), каково его назначение?

48 Описание физической структуры основной памяти ЭВМ.

49 Структурная схема модуля основной памяти. Принцип работы.

50 Модуль оперативной памяти *SIMM.* Назначение, функциональные возможности, конструктивное исполнение.

51 Модуль оперативной памяти *DIMM.* Назначение, функциональные возможности, конструктивное исполнение.

52 Модуль оперативной памяти *RIMM.* Назначение, функциональные возможности, конструктивное исполнение.

53 Динамическая память *FPM DRAM.* Назначение, функциональные возможности, конструктивное исполнение.

54 Динамическая память *RAM EDO.* Назначение, функциональные возможности, конструктивное исполнение.

55 Динамическая память *BEDO DRAM.* Назначение, функциональные возможности, конструктивное исполнение.

56 Динамическая память *SDRAM.* Назначение, функциональные возможности, конструктивное исполнение.

57 Динамическая память *DDR SDRAM.* Назначение, функциональные возможности, конструктивное исполнение.

58 Динамическая память *DRDRAM.* Назначение, функциональные возможности, конструктивное исполнение.

59 Описание таблицы страниц виртуальной памяти.

60 Описание характеристик дисковых накопителей.

3 Задание №3

Краткие сведения по логическим основам ЭВМ, методикам анализа и синтеза комбинационных схем

3.1 Логические основы ЭВМ

Работу любой схемы ЭВМ можно представить следующим образом: на входы схемы поступает некоторая последовательность нулей и единиц, которая вызывает появление на выходах вполне определенной последовательности нулей и единиц. Такие схемы называются *логическими.*

Схемы, в которых значения выходных сигналов в момент времени t однозначно определяются значениями входных сигналов в момент времени $t_1 \leq t$, называются **комбинационными схемами** (КС) (рисунок 29).

Реализуемый в этих схемах способ информации называется комбинационным, так как результат обработки информации зависит только от комбинации входных сигналов и вырабатывается сразу при подаче входной информации.

Математическим аппаратом для описания КС являются ФАЛ (функции алгебры логики). ФАЛ от n переменных $f(x_1, x_2, \ldots, x_n)$ можно задать с помощью таблицы истинности (таблица 3).

Таблица 3 – Таблица истинности для основных бинарных логических операций

A	***B***	$\wedge$	$\vee$	$\rightarrow$	$\oplus$	~	$\mid$	$\downarrow$
0	0	0	0	1	0	1	1	1
0	1	0	1	1	1	0	1	0
1	0	0	1	0	1	0	1	0
1	1	1	1	1	0	1	0	0

Схемы, в которых выходные сигналы определяются не только значениями входных сигналов в данный момент времени, но и состоянием схемы, зависящим от сигналов, поданных на ее входы в предыдущие моменты времени, называются *цифровыми автоматами* (ЦА) (рисунок 29). ЦА содержит память, состоящую из запоминающих элементов (ЗЭ) – триггеров, элементов задержки, фиксирующих состояние, в котором он находится.

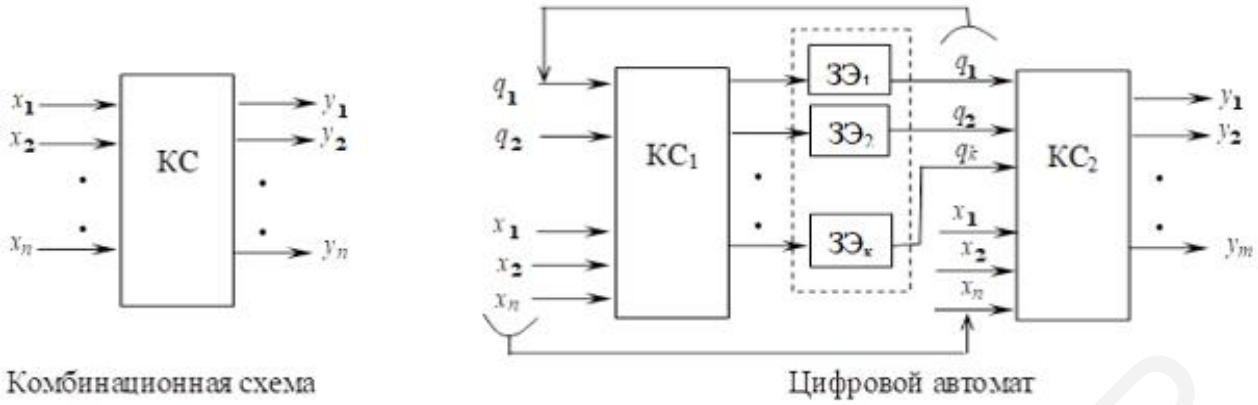


Рисунок 29 – Комбинационная схема, цифровой автомат

Сложные логические схемы ЭВМ состоят из соединенных между собой определенным образом простейших схем, называемых **логическими элементами**. Выходной сигнал логического элемента однозначно определяется комбинацией входных сигналов $x_1, x_2, \ldots, x_n$, каждый из которых равен нулю или единице, и поэтому может быть представлен в виде некоторой функции $f(x_1, x_2, \ldots, x_n)$, которая также принимает только два значения: нуль или единицу.

Функция $f(x_1, x_2, \ldots, x_n)$ называется *переключательной функцией* (ПФ) или *булевой функцией*, если она, так же как и ее аргументы, может принимать только два значения: нуль или единицу. Любая переключательная функция может быть задана таблицей ее значений в зависимости от значений аргументов.

Технические вопросы, связанные с состоянием логических схем ЭВМ, можно решать с помощью аппарата математической логики, впервые использовавшей переключательные функции. Основные понятия алгебры логики были развиты Джорджем Булем.

В алгебре логики, называемой также *исчислением высказываний или булевой алгеброй*, объектом исследования являются высказывания.

В алгебре логики производятся операции над символами, но не арифметические, а специальные логические операции. Кроме того, под символами понимаются не цифры, или числа, а высказывания.

Под высказыванием понимается всякое утверждение, которое может быть истинным или ложным. Поэтому в алгебре логики все высказывания делятся на два класса: *истинные* и *ложные*. Никакие другие утверждения не допускаются.

Согласно принятым обозначениям истинному утверждению соответствует значение 1, а ложному – значение 0. Эти цифровые выражения истинных и ложных утверждений позволяют производить их алгебраический анализ, на котором основан систематический расчет логических схем.

Отдельные высказывания принято обозначать латинскими буквами.

Пример

A = (2*2 = 4) $A = 1$;

B = «Париж – столица Франции» $B = 1$;

D = «Рим – столица Франции» $D = 0$;

C = «трижды три – семь» $C = 0$.

Истинностные значения новых высказываний определяются при этом только истинностными значениями входящих в них высказываний. Построение из данных высказываний (или из данного высказывания) нового высказывания называется *логической операцией*. Знаки логических операций называются *логическими связками*.

Пример

Из высказываний «$x > 2$», «$x < 3$» при помощи связки И можно получить высказывание «$x > 2$ И $x < 3$»; из высказываний «$y > 10$», «$x < 3$» при помощи связки ИЛИ можно получить высказывание «$y > 10$ ИЛИ $x < 3$».

Истинность или ложность получаемых таким образом высказываний зависит от истинности и ложности исходных высказываний и соответствующей трактовки связок как операций над высказываниями.

Одной из основных операций алгебры логики является *операция отрицания*. Отрицание высказывания A (т. е. НЕ A) обозначается $\overline{A}$ и читается: «отрицание A», «не A» или «A с чертой».

В таблице 4 приведены основные бинарные логические операции и связки.

Пример

Дано высказывание A = «Минск – столица Бельгии».

Тогда НЕ A = НЕ «Минск – столица Бельгии». Высказывание НЕ A означает: не верно, что A, т. е. не верно, что «Минск – столица Бельгии».

В таблице 4 представлены все наборы значений переменных A и B и значения операций на этих наборах.

Таблица 4 – Основные бинарные логические операции и связки

Обозначение логической операции	Другие обозначения логической операции	Название логической операции и связки	Логические связки
1	2	3	4
$A \wedge B$	$A\&B$ $A \cdot B$ AB	**Конъюнкция**, логическое умножение, логическое И	A **и** B; (A ***** B)
$A \vee B$	$A + B$	**Дизъюнкция**, логическое сложение, логическое ИЛИ	A **или** B; ($A + B$)
$A \rightarrow B$	$A \supseteq B$ $A \Rightarrow B$	**Импликация**, логическое следование	**если** A, **то** B;

Продолжение таблицы 4

1	2	3	4
$A \oplus B$	$A \Delta B$	Сумма по модулю 2, **разделительная дизъюнкция**, разделительное «ИЛИ»	**либо** A, **либо** B
$A \sim B$	$A \equiv B$ $A \leftrightarrow B$ $A \Leftrightarrow B$	**Эквиваленция**, тождественность равнозначность	A **тогда и только тогда, когда** B
$A \mid B$	$\overline{A \wedge B}$	Штрих Шеффера, **антиконъюнкция**	**не верно, что** A **и** B
$A \downarrow B$	$\overline{A \vee B}$	Стрелка Пирса, **антидизъюнкция**,	**ни** A, **ни** B

Конъюнкция. Результатом операции конъюнкции для высказывания $A \wedge B$ будет *истина* только тогда, когда истинны одновременно оба высказывания.

Пример

Даны высказывания A = «Минск – столица Беларуси» и B = «Париж – столица Франции».

Сложное высказывание $A \wedge B$ = «Минск – столица Беларуси» $\wedge$ «Париж – столица Франции » истинно, так как истинны оба высказывания.

Дизъюнкция. Результатом операции дизъюнкции для высказывания $A \vee B$ будет *истина* тогда, когда истинно хотя бы одно высказывание, входящее в него.

Пример

Даны высказывания A = «2 + 3 = 5» и B = «3 + 3 = 5».

Сложное высказывание $A \vee B$ = «2 + 3 = 5 $\vee$ 3 + 3 = 5» истинно, так как истинно высказывание A.

Эквиваленция. Результатом операции эквиваленции для высказывания $A \sim B$ будет *истина* тогда, когда истинны или ложны одновременно оба высказывания. Отличие эквиваленции от конъюнкции состоит в том, что вне зависимости от смысла, равнозначными являются как истинные, так и ложные высказывания.

Пример

Даны высказывания A = «2 + 2 = 7» и B = «1 – 8 = 5».

Сложное высказывание $A \sim B$= «2 + 2 = 7 тогда и только тогда, когда 1 – 8 = 5» истинно, так как оба высказывания ложны.

Импликация. Результатом операции импликации для высказывания $A \rightarrow B$ будет ложь только тогда, когда первое высказывание (A) истинно, а второе (B) – ложно. При этом A – предпосылка, а B – следствие. В остальных случаях результатом операции всегда будет истина.

Пример

Даны высказывания A = «2 + 2 = 4» и B = «1 – 8 = 5».

Сложное высказывание $A \rightarrow B$ = «если 2 + 2 = 4, то 1 – 8 = 5» ложно, так как высказывание A истинно, а B – ложно.

Антиконъюнкция. Результатом операции антиконъюнкции для высказывания $A \mid B$ будет ложь только тогда, когда оба высказывания истинны. В остальных случаях результатом операции всегда будет истина.

Пример

Даны высказывания A = «Варшава – столица Польши» и B = «Анкара – столица Турции».

Сложное высказывание $A \mid B$ = «не верно, что Варшава – столица Польши и Анкара – столица Турции» ложно, так как истинны оба высказывания.

Антидизъюнкция. Результатом операции антидизъюнкции для высказывания $A \downarrow B$ будет истина только тогда, когда оба высказывания ложны. В остальных случаях результатом операции всегда будет ложь.

Пример

Даны высказывания A = «Пекин – столица Беларуси» и B = «Минск – столица Китая».

Сложное высказывание $A \downarrow B$=«ни Пекин – столица Беларуси, ни Минск – столица Китая» истинно, так как ложны оба высказывания.

Связки и частица НЕ рассматриваются в алгебре логики как операции над величинами, принимающими значения 0 (ложь/*false*) и 1 (истина/*true*), и результатом применения этих операций также являются числа 0 или 1.

Любая булева функция может быть задана таблицей ее значений в зависимости от значений переменных (аргументов) или таблицей истинности.

Пример

Дана переменная A = 1(истина). После применения операции инверсии для переменной A ее значение станет равным 0 (ложь).

На этапе конструирования аппаратных средств алгебра логики позволяет значительно упростить логические функции, описывающие функционирование схем компьютера и, следовательно, уменьшить число элементарных логических элементов, из десятков тысяч которых состоят основные узлы компьютера.

В логической схеме компьютера выделяют логические элементы. **Логический элемент** компьютера – это часть электронной логической схемы, которая реализует элементарную логическую формулу.

Логическими элементами компьютеров являются электронные схемы И, ИЛИ, НЕ, И-НЕ, ИЛИ-НЕ. С помощью этих схем можно реализовать любую логическую формулу, описывающую работу устройств компьютера.

Каждый логический элемент имеет свое условное обозначение, которое выражает его логическую формулу, но не указывает на то, какая именно электронная схема в нем реализована. Это упрощает запись и понимание сложных логических схем.

Схема И реализует конъюнкцию двух или более логических значений. Условное обозначение структурной схемы И представлено на рисунке 30, *а*.

На выходе схемы И значение 1 будет тогда и только тогда, когда на всех входах будут 1. Когда хотя бы на одном из входов будет 0, то на выходе также будет 0.

Операция конъюнкции на функциональных схемах обозначается знаком & (читается как «амперсанд»), являющимся сокращенной записью английского слова *and*.

Схема ИЛИ реализует дизъюнкцию двух логических значений. Условное обозначение схемы ИЛИ представлено на рисунке 30, *б*.

На выходе схемы ИЛИ значение 0 будет тогда и только тогда, когда на всех входах будут 0. Когда хотя бы на одном из входов будет 1, то на выходе также будет 1.

Операция дизъюнкции на функциональных схемах обозначается знаком «1».

Схема НЕ (инвертор) реализует операцию отрицания. Условное обозначение схемы НЕ представлено на рисунке 30, *в*.

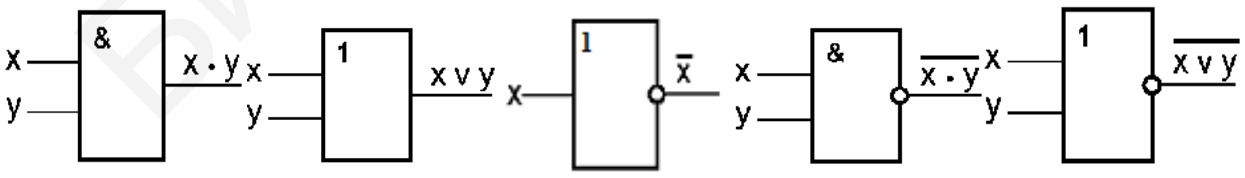
Если на входе схемы 0, то на выходе будет 1. Когда на входе – 1, на выходе будет 0.

Схема И-НЕ состоит из элемента И и инвертора и осуществляет отрицание результата схемы И. Условное обозначение схемы И-НЕ представлено на рисунке 30, *г*.

На выходе схемы И-НЕ значение 0 будет тогда и только тогда, когда на всех входах будут 1.

Схема ИЛИ-НЕ состоит из элемента ИЛИ и инвертора и осуществляет отрицание результата схемы ИЛИ. Условное обозначение схемы ИЛИ-НЕ представлено на рисунке 30, *д*.

На выходе схемы ИЛИ-НЕ значение 1 будет тогда и только тогда, когда на всех входах будут 0.



а б в г д

а – схема И; б – схема ИЛИ; в – схема НЕ; г – схема И-НЕ;
д – схема ИЛИ-НЕ

Рисунок 30 – Условные обозначения логических схем

Алгебра логики позволяет решать следующие задачи:

- производить логический анализ схем, т. е. по имеющейся логической схеме составить описание ее работы с помощью логических выражений;

- выполнить синтез логической схемы, т. е. построить оптимальную логическую схему из простых схем;

- определить функционально полный набор логических элементов, т. е. такой набор, с помощью которого можно построить любую заранее заданную логическую схему.

Логический элемент, имеющий n входов и m выходов полностью задан, если известен закон функционирования элементов, который определяет значения выходных сигналов $y_1, y_2, \ldots, y_m$ в зависимости от значений входных сигналов $x_1, x_2, \ldots, x_n$.

Закон функционирования любого логического элемента можно описать системой ПФ:

$$\left.\begin{aligned} y_1 &= y_1(x_1, x_2, \ldots, x_n), \\ y_2 &= y_2(x_1, x_2, \ldots, x_n), \\ \ldots \quad &\ldots \quad \ldots, \\ y_m &= y_m(x_1, x_2, \ldots, x_n) \end{aligned}\right\}$$

Таким образом, логический элемент рассматривается по существу, как «черный ящик» с заданным законом функционирования, но с неизвестной внутренней структурой.

Логической системой называется совокупность логических элементов, соединенных между собой так, что выполняется заданный закон функционирования схемы.

При изучении логических схем важную роль играет их внутренняя структура, так как один и тот же закон функционирования можно реализовать схемами, имеющими различное число элементов и различные способы их соединения между собой.

Одна из основных задач синтеза схем заключается в выборе типов элементов, из которых должны собираться логические схемы. В ЭВМ применяется большое количество самых разнообразных по способам функционирования схем. Поэтому основное требование, предъявляемое к набору логических элементов, состоит в том, чтобы с помощью этого набора можно было построить любую сколь угодно сложную схему. Ввиду того, что законы функционирования элементов однозначно описываются ПФ, сформулированное требование сводится к определению набора таких ПФ, с помощью которых можно получить любую сколь угодно сложную ПФ.

При составлении логических схем из элементов используют два приема:

- последовательное соединение элементов;

- перестановку входов.

Этим приемам соответствуют математические операции суперпозиции и подстановки аргументов.

Суперпозиция состоит в подстановке вместо аргументов булевой функции других булевых функций и в перенумерации (переименовании) аргументов. Указанная подстановка возможна, так как и аргументы и сами булевые функции могут принимать только два значения – 0 и 1. Принцип суперпозиции показан на рисунке 31.

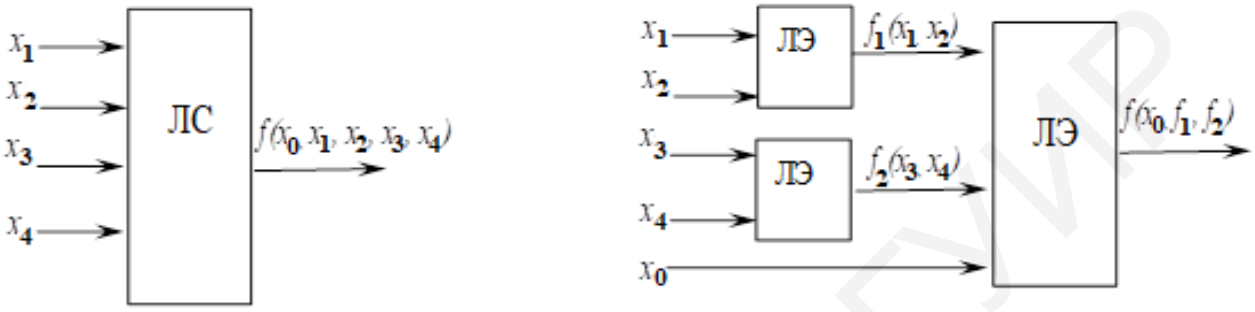


Рисунок 31 – Принцип суперпозиции

Перестановка входов:

$$f(x_0,x_1,x_2,x_3,x_4)=f\left[x_0\cdot f_1(x_1,x_2)\cdot f_2(x_3,x_4)\right]=f\left[f_1(x_0,x_4)\cdot x_1\cdot f_2(x_2,x_3)\right]=$$

$$=f\left[f_1(x_0,x_4)\cdot f_2(x_1,x_3)\cdot x_2\right].$$

Техническая задача определения набора логических элементов, с помощью которых можно построить любую логическую схему, сводится к математической задаче отыскания функционально полного набора ПФ.

Система ПФ называется функционально полной, если с помощью функций, сходящих в эту систему, применяя операции суперпозиции и подстановки, можно получить любую сколь угодно сложную ПФ.

Выбор функционально полной системы ПФ с технической точки зрения эквивалентен выбору типов логических элементов, из которых может быть построена любая логическая схема ЭВМ. Поэтому в основу выбора функционально полных систем ПФ должны быть положены технические соображения.

Набор логических элементов называется функционально полным, если из элементов, входящих в этот набор, можно построить любую логическую систему ЭВМ.

К логическим элементам предъявляют ряд технических требований:

1 Типы логических элементов должны быть выбраны так, чтобы из этих элементов можно было построить любую сложную схему. Это требование сводится к требованию функциональной полноты системы ПФ, реализуемых логическими элементами.

2 Схемы логических элементов должны быть максимально простыми – они должны содержать минимальное число радиоэлектронных компонентов.

3 Количество различных типов элементов должно быть по возможности минимальным, так как однотипность элементов имеет ряд существенных достоинств: простота и дешевизна изготовления, взаимозаменяемость элементов, простота эксплуатации и т. п. Поэтому желательно, чтобы функционально полная система ПФ содержала минимальное число различных функций. Однако с другой стороны, количество различных типов элементов в ряде случаев не должно быть слишком малым, так как разнообразие элементов позволяет более гибко проводить синтез сложных схем. При этом может оказаться, что сложная схема, собранная из разнообразных элементов, будет содержать меньшее количество деталей, чем схема, собранная из однотипных элементов.

Наиболее удобной для решения задачи синтеза схем ЭВМ является функционально полная система ПФ, содержащая конъюнкцию, дизъюнкцию и инверсию. Кроме того, для повышения гибкости этой системы для построения сложных схем в нее включают константу 0, константу 1 и переменную *x*. Эта система ПФ позволяет построить любую схему ЭВМ. Элементы, реализующие другие ПФ, фактически всегда состоят из элементов основной системы.

Построение КС в базисе {И, ИЛИ, НЕ} осуществляется по следующему алгоритму [7]:

- для реализации каждой логической операции используется соответствующий логический элемент;
- логические операции выполняются в следующем порядке: НЕ, И, ИЛИ.
- выражения в скобках и под знаком отрицания реализуются в первую очередь.

Для функции $f = \overline{x_3 \cdot \left(x_2 \vee \overline{x_1} \right)}$ в базисе {И, ИЛИ, НЕ} приведен пример построения схемы (рисунок 32).

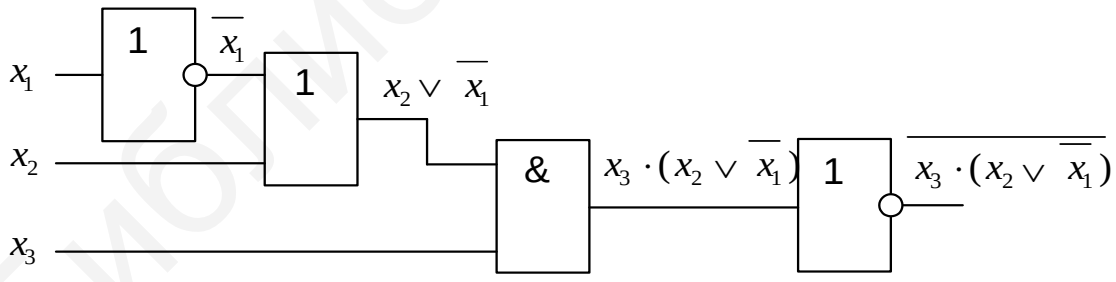
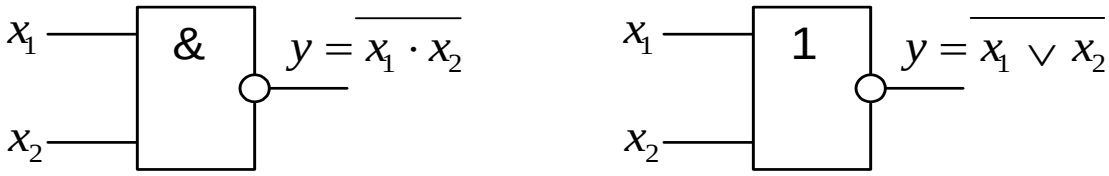


Рисунок 32 – Пример построения КС в базисе {И, ИЛИ, НЕ}

Для построения КС, заданной логической формулой, необходимо иметь столько типов логических элементов, сколько функций содержится в базисе, в котором записана ФАЛ.

Реализация КС может быть осуществлена также на базе одного логического элемента из нижеследующих:

- элемента И-НЕ (рисунок 33, *а*), реализующего функцию Шеффера;
- элемента ИЛИ-НЕ (рисунок 33, *б*), реализующего функцию Вебба.



а б

а – функция Шеффера; б – функция Вебба

Рисунок 33 – Примеры реализации функций Шеффера и Вебба на логических элементах

Каждый из этих элементов составляет функционально полную систему алгебры логики. На рисунке 34 показана реализация основных ФАЛ на элементе ИЛИ-НЕ. На рисунке 35 показана реализация основных ФАЛ на элементе И-НЕ.

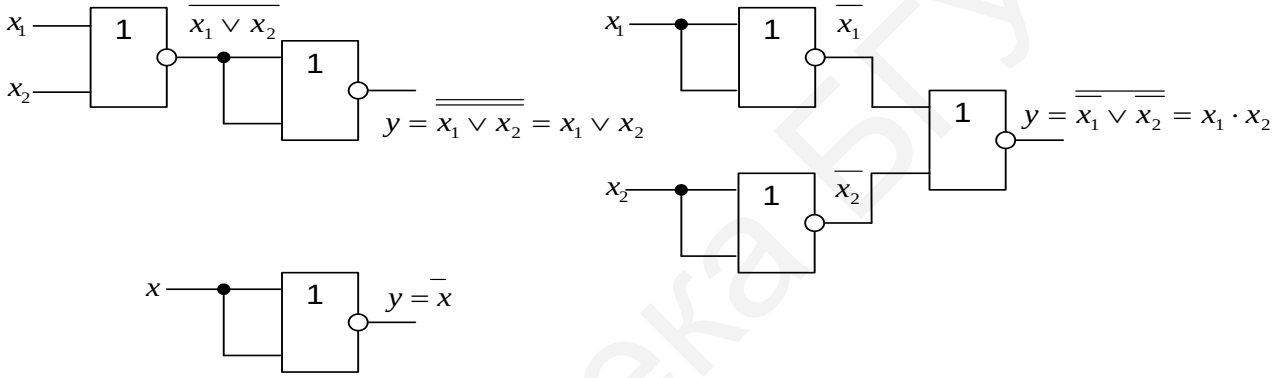


Рисунок 34 – Реализация функций И, ИЛИ, НЕ с использованием элемента Вебба

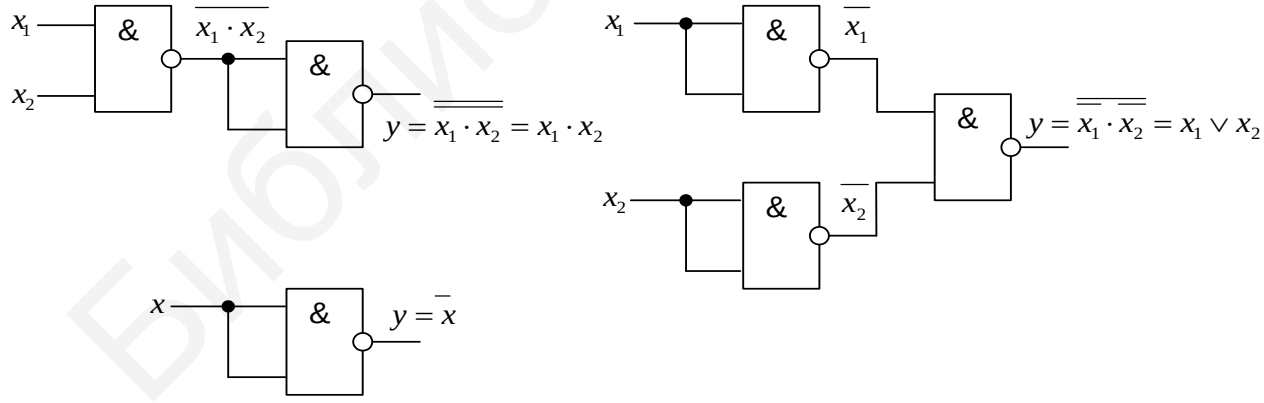


Рисунок 35 – Реализация функций И, ИЛИ, НЕ с использованием элемента Шеффера

Построение схем в базисах {И-НЕ} и {ИЛИ-НЕ} производится путем замены каждого элемента схемы, реализованной в базисе {И, ИЛИ, НЕ}, на эквивалентный элемент, реализованный в соответствующем базисе.

Минимизация ФАЛ с помощью карт Карно

Так как одна и та же ФАЛ может быть записана с помощью различных формул, то возникает задача получения минимальной формы записи (с минимальным числом букв), которая потребует минимальных затрат аппаратуры при реализации.

Рассмотрим процесс минимизации ФАЛ с помощью карт Карно. Карта Карно может быть составлена для любого количества переменных, однако удобно работать при количестве переменных не более шести. Карта Карно (представлена на рисунке 36) является другой формой представления таблицы истинности (таблица 5).

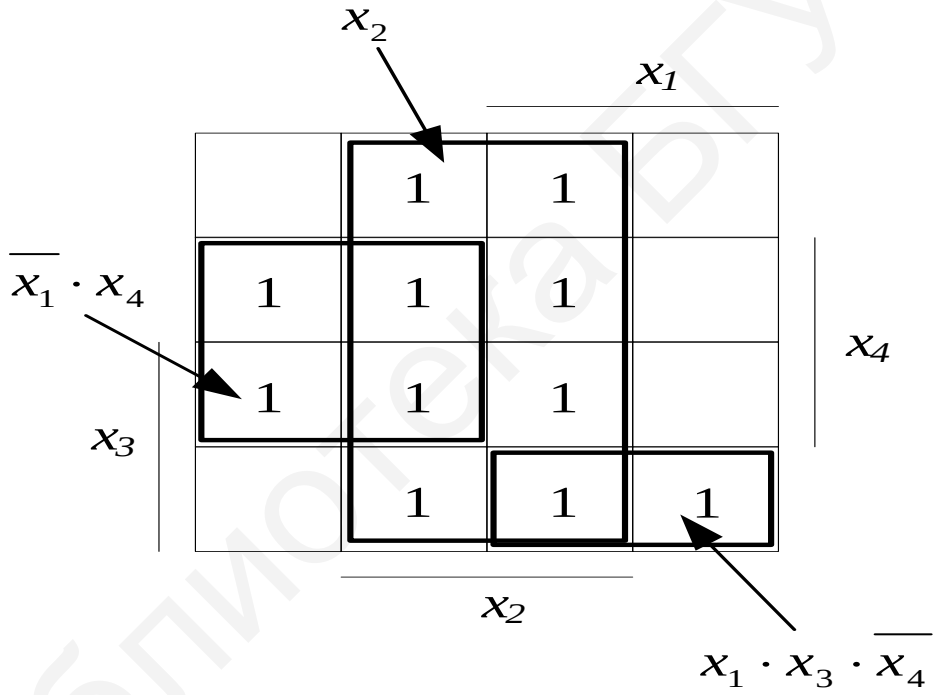


Рисунок 36 – Карта Карно

Каждая клетка карты соответствует строке таблицы (двоичному набору). Часть клеток (половина), которым соответствует значение $x_i = 1$, отмечается чертой, соответственно в областях, не обозначенных чертой, переменная $x_i = 0$.

Для задания ФАЛ в карте Карно надо проставить 1 в тех клетках, которые соответствуют разрешенным наборам. В карте на рисунке 36 задана ФАЛ из таблицы 5.

Таблица 5 – Таблица истинности

Шаг	x_1	x_2	x_3	x_4	y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Минимизация ФАЛ по карте Карно заключается в объединении соседних клеток в прямоугольные контуры, причем соседними считаются и клетки, разделенные внешней границей карты.

Минимизация производится по следующим правилам:

1 Все единицы должны быть включены в прямоугольные контуры.
2 Во всех клетках контура должны стоять единицы.
3 Число клеток в контуре кратное степени числа 2: 1, 2, 4, 8, … .
4 Контуры могут накладываться друг на друга.

5 Контуры, все клетки которых уже вошли в другие контуры, являются лишними.

6 Для получения наиболее простой формулы надо выбирать контуры с максимальным числом клеток.

7 Каждому контуру соответствует конъюнкция, составленная из переменных, значения которых не изменяются во всех клетках контура.

8 Конъюнкции контуров объединяются знаками дизъюнкции.

Ниже представлена ФАЛ, соответствующая карте Карно, представленной на рисунке 36:

$$f = x_2 \vee \overline{x_1} \cdot x_4 \vee x_1 \cdot x_3 \cdot \overline{x_4}.$$

Минимизируем функцию, представленную своими разрешенными наборами, с использованием карт Карно (см. рисунок 36):

$$f = \{0, 1, 2, 3, 4, 5, 6, 7, 14, 15, 16, 17, 22, 23, 30, 31\}.$$

В карте Карно от пяти переменных два контура, удовлетворяющие указанным правилам, объединяются в один контур, если они расположены симметрично относительно центральной оси. Например, контур на рисунке 37, соответствующий конъюнкции $x_3 \cdot x_4$.

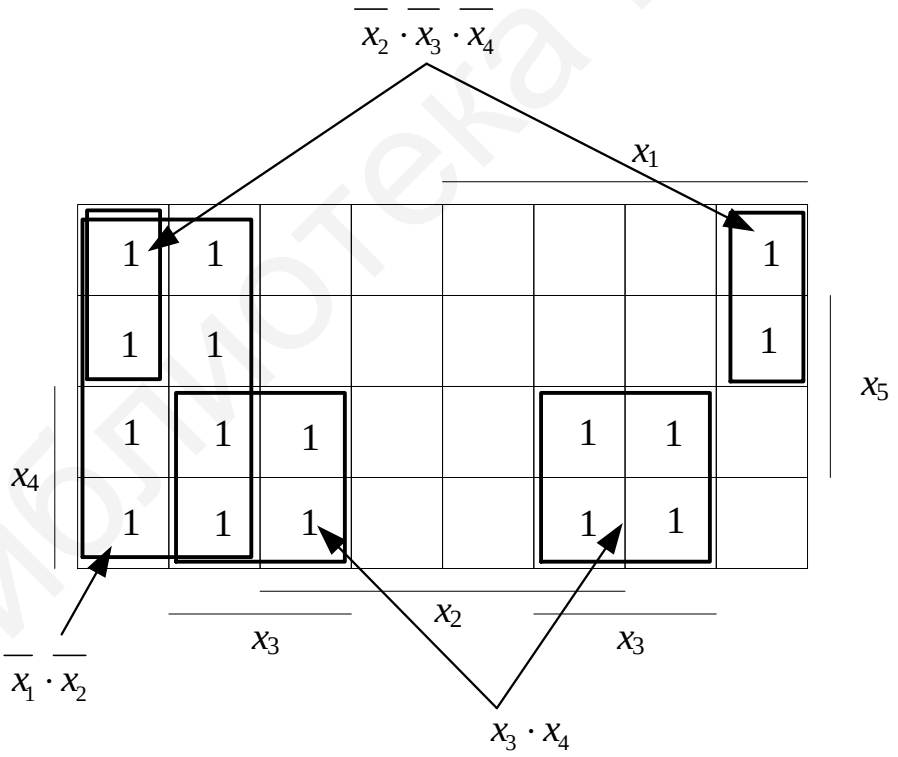


Рисунок 37 – Минимизация по карте Карно

В итоге мы получаем следующую функцию:

$$f = \overline{x_1} \cdot \overline{x_2} \vee x_3 \cdot x_4 \vee \overline{x_2} \cdot \overline{x_3} \cdot \overline{x_4}.$$

3.2 Методика выполнения задания №3

Приступая к выполнению задания №3 контрольной работы необходимо:
- ознакомиться с содержанием методических указаний;
- получить вариант задания у преподавателя.

Для функции f_1, соответствующей номеру варианта, выполнить следующие действия:
- построить комбинационную схему;
- составить таблицу истинности;
- записать СДНФ и СКНФ;
- построить схему в базисе {И, ИЛИ, НЕ};
- построить схему в базисе Шеффера {И-НЕ};
- построить схему в базисе Вебба {ИЛИ-НЕ};
- записать исходную формулу в базисе И, НЕ;
- записать исходную формулу в базисе ИЛИ, НЕ.

Для функции f_2, соответствующей номеру варианта, выполнить следующие действия:
- осуществить минимизацию по карте Карно;
- записать минимизированную ФАЛ.

Пример выполнения задания [7]

Исходные данные:

$$f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4;\ f_2 = \{1, 2, 3, 6, 7, 10, 11, 13, 14\}.$$

Построение комбинационной схемы

Исходя из правил построения релейно-контактных схем, функцию f_1 необходимо преобразовать, исключив знаки отрицания над выражением (используем формулы де Моргана):

$$f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4 = \overline{x_1} \cdot \overline{x_2} \cdot x_3 \vee x_4.$$

Применим дополнительное реле для построения схемы:

$$f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4 = \overline{y} \cdot x_3 \vee x_4;$$

$$y = x_1 \vee x_2.$$

Построение таблицы истинности

Построение таблицы удобно проводить поэтапно, выделяя столбец для отдельных выражений исходной формулы (таблица 6).

Таблица 6 – Таблица истинности для заданной функции

Шаг	x_1	x_2	x_3	x_4	$x_1 \vee x_2$	$\overline{x_1 \vee x_2}$	$\overline{x_1 \vee x_2} \cdot x_3$	$f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4$
0	0	0	0	0	0	1	0	0
1	0	0	0	1	0	1	0	1
2	0	0	1	0	0	1	1	1
3	0	0	1	1	0	1	1	1
4	0	1	0	0	1	0	0	0
5	0	1	0	1	1	0	0	1
6	0	1	1	0	1	0	0	0
7	0	1	1	1	1	0	0	1
8	1	0	0	0	1	0	0	0
9	1	0	0	1	1	0	0	1
10	1	0	1	0	1	0	0	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	0	0
13	1	1	0	1	1	0	0	1
14	1	1	1	0	1	0	0	0
15	1	1	1	1	1	0	0	1

Составление СДНФ и СКНФ

СДНФ исходной функции:

$$f_1 = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} \cdot x_4 \vee \overline{x_1} \cdot \overline{x_2} \cdot x_3 \cdot \overline{x_4} \vee \overline{x_1} \cdot \overline{x_2} \cdot x_3 \cdot x_4 \vee \overline{x_1} \cdot x_2 \cdot \overline{x_3} \cdot x_4 \vee$$
$$\vee \overline{x_1} \cdot x_2 \cdot x_3 \cdot x_4 \vee x_1 \cdot \overline{x_2} \cdot \overline{x_3} \cdot x_4 \vee x_1 \cdot \overline{x_2} \cdot x_3 \cdot x_4 \vee x_1 \cdot x_2 \cdot \overline{x_3} \cdot x_4 \vee x_1 \cdot x_2 \cdot x_3 \cdot x_4.$$

СКНФ исходной функции:

$$f_1 = \left(x_1 \vee x_2 \vee x_3 \vee x_4\right) \cdot \left(x_1 \vee \overline{x_2} \vee x_3 \vee x_4\right) \cdot \left(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4\right) \&$$
$$\&\left(\overline{x_1} \vee x_2 \vee x_3 \vee x_4\right) \cdot \left(\overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4\right) \cdot \left(\overline{x_1} \vee \overline{x_2} \vee x_3 \vee x_4\right) \&$$
$$\&\left(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee x_4\right).$$

Построение комбинационной схемы в базисе {И, ИЛИ, НЕ}

Схема, реализующая функцию $f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4$ в базисе {И, ИЛИ, НЕ}, представлена на рисунке 38.

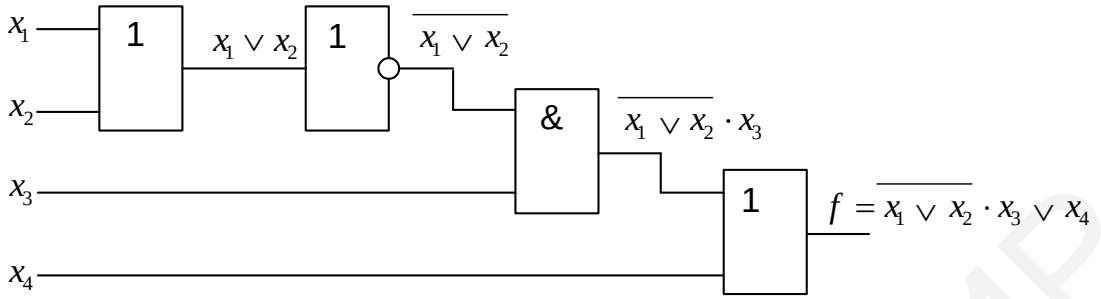


Рисунок 38 – Реализация функции $f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4$ в базисе {И, ИЛИ, НЕ}

Построение комбинационной схемы в базисе Шеффера {И-НЕ}

Схема, реализующая функцию $f = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4$ в базисе Шеффера, представлена на рисунке 39.

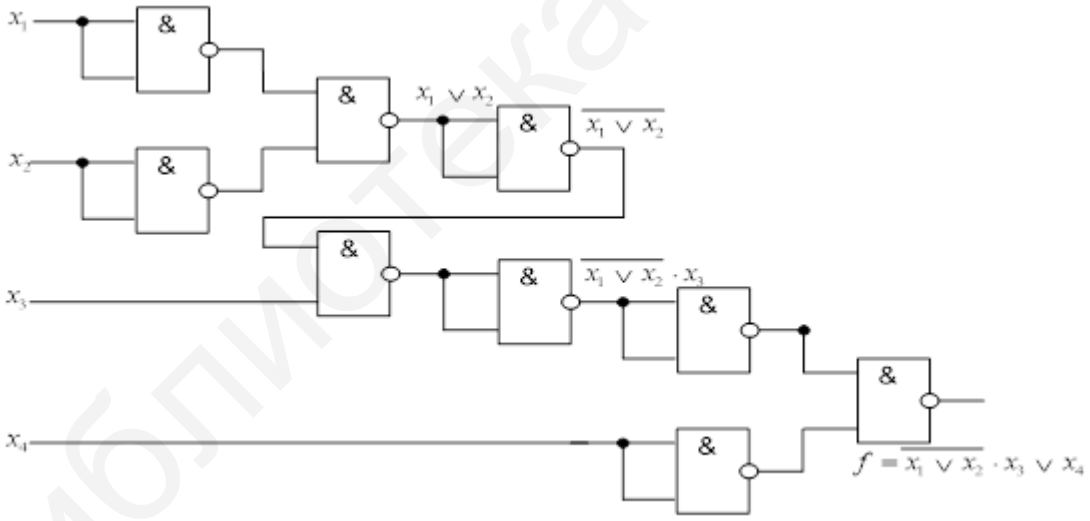


Рисунок 39 – Схема реализации функции $f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4$ в базисе Шеффера

Построение комбинационной схемы в базисе Вебба {ИЛИ-НЕ}

Схема, реализующая функцию $f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4$ в базисе Вебба, представлена на рисунке 40. Пунктиром показана возможность исключения двух элементов по закону двойного отрицания.

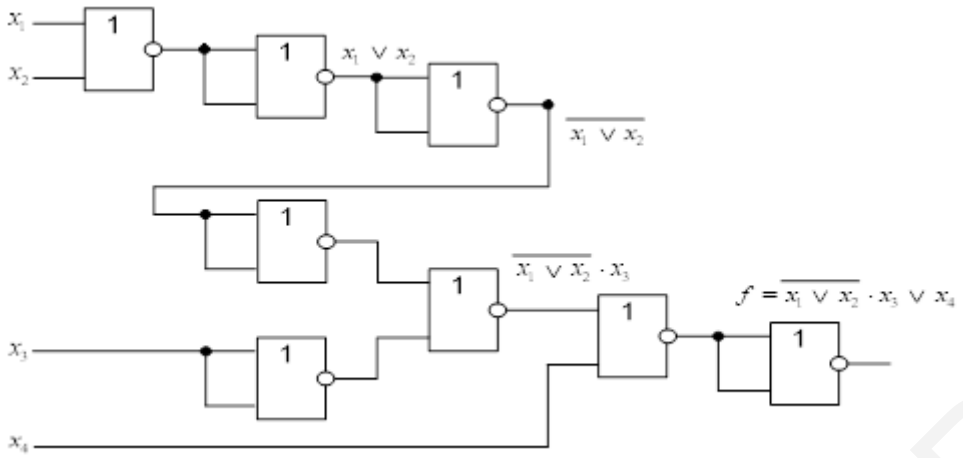


Рисунок 40 – Схема реализации функции $f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4$ в базисе Вебба

Запись исходной формулы в базисе И, НЕ

Воспользовавшись законами алгебры логики, преобразуем исходную функцию к виду, содержащему только функции И, НЕ:

$$f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4 = \overline{x_1} \cdot \overline{x_2} \cdot x_3 \vee x_4 = \overline{\overline{y}} \vee x_4 = \overline{\overline{y}} \vee x_4 = \overline{y} \cdot x_4 =$$

$$= \overline{\overline{\overline{x_1} \cdot \overline{x_2} \cdot x_3}} \cdot \overline{x_4}.$$

Запись исходной формулы в базисе ИЛИ, НЕ

Воспользовавшись законами алгебры логики, преобразуем исходную функцию к виду, содержащему только функции ИЛИ, НЕ:

$$f_1 = \overline{x_1 \vee x_2} \cdot x_3 \vee x_4 = y \cdot x_3 \vee x_4 = \overline{\overline{y}} \cdot x_3 \vee x_4 = \overline{y} \vee \overline{x_3} \vee x_4 =$$

$$= \overline{\overline{\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}}} \vee x_4 = \overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee x_4.$$

Минимизация по картам Карно

Минимизируем функцию $f_2 = \{1, 2, 3, 6, 7, 10, 11, 13, 14\}$, заданную своими разрешенными наборами, по картам Карно.

Результат минимизации представлен на рисунке 41.

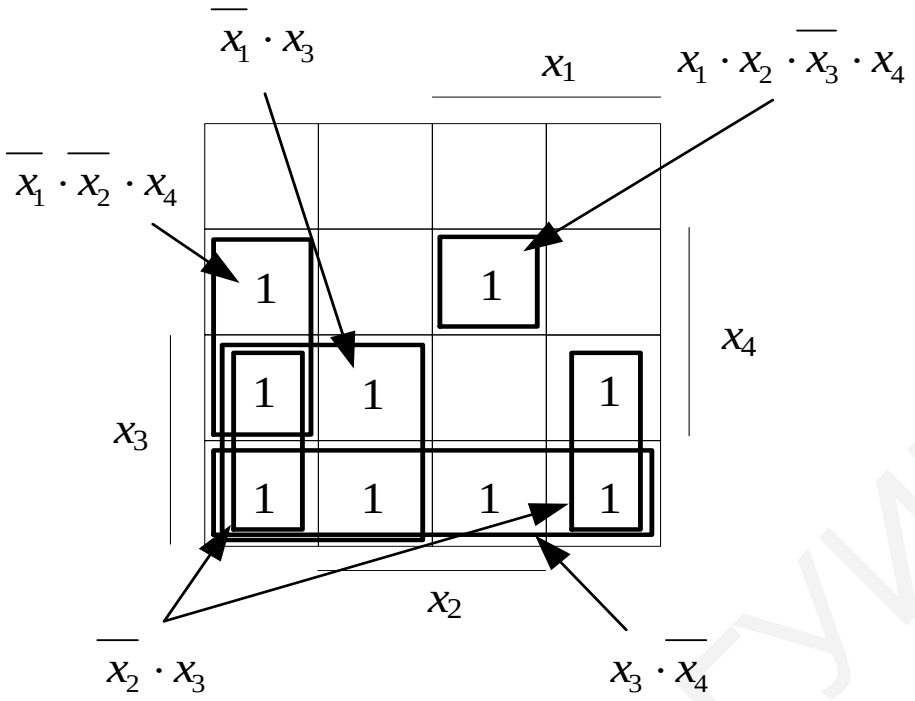


Рисунок 41 – Минимизация f_2 по карте Карно

Запишем ФАЛ, соответствующую произведенной минимизации:

$$f_2 = x_1 \cdot x_2 \cdot \overline{x_3} \cdot x_4 \vee \overline{x_2} \cdot x_3 \vee x_3 \cdot \overline{x_4} \vee \overline{x_1} \cdot x_3 \vee \overline{x_1} \cdot \overline{x_2} \cdot x_4.$$

3.3 Содержание задания №3

Варианты заданий представлены в таблице 7.

Таблица 7 – Функции f_1 и f_2 для каждого из вариантов задания №3

Номер варианта	Функция f_1	Функция f_2
1	2	3
01	$f_1 = x_3 \cdot (x_2 \vee \overline{x_4}) \vee \overline{x_1 \cdot x_2}$	$f_2 = \{0,4,6,8,12,13,14,15\}$
02	$f_1 = \overline{x_1} \cdot x_2 \cdot \overline{(\overline{x_4} \vee x_3)} \vee x_2 \cdot x_3$	$f_2 = \{1,3,7,9,13\}$
03	$f_1 = \overline{x_1} \cdot \overline{x_2 \vee x_4} \vee x_4 \cdot \overline{x_3}$	$f_2 = \{1,3,7,9,13\}$
04	$f_1 = \overline{x_1} \cdot \overline{x_4 \vee x_1 \cdot \overline{x_3}} \vee x_1 \cdot x_2$	$f_2 = \{0,6,7,8,9,11,12,13,15\}$
05	$f_1 = \overline{x_4 \cdot x_2 \vee \overline{x_1 \cdot \overline{x_3}}} \vee \overline{x_4} \cdot x_2$	$f_2 = \{5,7,9,11,13,15\}$
06	$f_1 = \overline{\overline{x_1} \vee x_4} \vee \overline{x_2} \cdot \overline{x_1 \vee \overline{x_4} \cdot \overline{x_3}}$	$f_2 = \{2,10,11,12,14\}$
07	$f_1 = x_1 \cdot \overline{x_3} \vee \overline{x_2 \cdot \overline{x_1} \vee x_4}$	$f_2 = \{2,4,5,8,10,11,14,15\}$
08	$f_1 = (\overline{x_1} \vee \overline{x_2}) \cdot \overline{\overline{x_4} \cdot x_3}$	$f_2 = \{0,4,13,14,15\}$
09	$f_1 = (x_3 \vee \overline{x_1}) \cdot (\overline{x_2} \cdot \overline{x_4} \vee \overline{x_3})$	$f_2 = \{0,4,13,14,15\}$

Продолжение таблицы 7

1	2	3
10	$f_1 = \overline{(x_2 \vee \overline{x_3} \cdot x_4)} \cdot (\overline{x_1} \vee x_4)$	$f_2 = \{0,4,5,6,7,8,11,15\}$
11	$f_1 = (\overline{x_1 \cdot x_2} \vee x_3) \cdot x_4$	$f_2 = \{2,3,6,7,10,11,13,14,15\}$
12	$f_1 = (\overline{x_1 \cdot x_2} \vee x_3) \cdot x_4$	$f_2 = \{2,3,6,7,10,11,13,14,15\}$
13	$f_1 = \overline{x_1 \cdot \overline{x_2}} \cdot (x_3 \vee \overline{x_1} \cdot x_4)$	$f_2 = \{1,5,8,10,12,14\}$
14	$f_1 = (x_1 \vee \overline{x_2 \cdot x_3}) \cdot (x_2 \vee \overline{x_4})$	$f_2 = \{1,2,3,5,6,7,14,15\}$
15	$f_1 = \overline{(x_2 \vee \overline{x_3} \cdot x_4} \cdot (\overline{x_1} \vee x_2)$	$f_2 = \{0,1,3,4,5,7,12,13\}$
16	$f_1 = \overline{x_3 \cdot \overline{x_4} \vee x_1 \cdot (x_2 \vee \overline{x_4})}$	$f_2 = \{1,3,4,5,7,11,15\}$
17	$f_1 = x_1 \vee \overline{x_2 \vee x_3} \vee \overline{x_1} \cdot x_4$	$f_2 = \{1,5,8,9,12,13,14,15\}$
18	$f_1 = (x_3 \vee \overline{x_1}) \cdot (\overline{x_2} \cdot \overline{x_4} \vee \overline{x_3})$	$f_2 = \{0,1,4,5,7,10,11,13,14,15\}$
19	$f_1 = \overline{x_1 \cdot (\overline{x_2} \vee x_3)} \cdot (\overline{x_4} \vee x_1)$	$f_2 = \{0,1,2,4,5,8,9,12,13,14,15\}$
20	$f_1 = \overline{(x_2 \vee \overline{x_3} \cdot x_4)} \cdot (\overline{x_1} \vee x_4)$	$f_2 = \{0,4,5,6,7,8,11,15\}$
21	$f_1 = \overline{x_1 \vee x_2} \cdot x_4 \vee \overline{x_2 \vee x_3}$	$f_2 = \{4,8,9,10,11,12,13,14,15\}$
22	$f_1 = \overline{\overline{x_3} \vee x_4} \cdot x_2 \cdot (x_1 \vee \overline{x_4})$	$f_2 = \{0,1,6,7,8,9\}$
23	$f_1 = \overline{(x_1 \vee \overline{x_2}) \cdot x_3} \vee \overline{x_3} \cdot x_4$	$f_2 = \{1,5,7,9,13,14,15\}$
24	$f_1 = (\overline{x_1 \cdot x_3} \vee \overline{x_4}) \cdot (x_2 \vee x_3)$	$f_2 = \{2,3,5,6,7,8,9,12,13,15\}$
25	$f_1 = \overline{x_2} \vee x_3 \cdot \overline{x_4} \vee \overline{x_1 \vee \overline{x_4}}$	$f_2 = \{1,3,5,6,7,9,10,11,13,15\}$
26	$f_1 = \overline{(x_1 \vee x_2) \cdot (x_3 \vee \overline{x_4})} \vee \overline{x_1}$	$f_2 = \{0,1,2,3,4,5,6,7,8,9,11,12,13\}$
27	$f_1 = (\overline{\overline{x_1 \vee \overline{x_3}} \vee x_2}) \cdot \overline{x_4}$	$f_2 = \{3,7,8,10,12,14\}$
28	$f_1 = \overline{(x_1 \cdot \overline{x_2} \vee x_3) \cdot x_4}$	$f_2 = \{2,3,4,5,6,7,11,15\}$
29	$f_1 = (\overline{x_1 \cdot x_4} \vee x_2) \cdot \overline{x_1} \cdot \overline{x_3}$	$f_2 = \{0,1,4,5,7,8,9,11,12,13\}$
30	$f_1 = \overline{x_1 \cdot x_4} \cdot x_3 \vee \overline{x_2 \cdot x_4}$	$f_2 = \{0,1,2,3,4,5,6,7,12,15\}$
31	$f_1 = \overline{\overline{x_1} \vee x_3} \vee x_2 \vee \overline{x_1} \cdot \overline{x_4}$	$f_2 = \{1,5,6,8,9,10,11,12,13,14,15\}$
32	$f_1 = (x_2 \vee \overline{x_3}) \cdot (\overline{x_1 \cdot x_4} \vee \overline{x_2})$	$f_2 = \{0,1,2,3,4,5,8,9\}$
33	$f_1 = (x_1 \cdot \overline{x_4} \vee x_3) \cdot (\overline{x_2} \vee x_1)$	$f_2 = \{0,1,10,11,12,13,14,15\}$
34	$f_1 = \overline{x_4 \cdot \overline{x_2} \vee x_3} \vee \overline{x_2} \cdot x_3$	$f_2 = \{0,2,4,6,11,15\}$
35	$f_1 = (\overline{x_1 \cdot x_4} \vee x_3) \cdot x_2$	$f_2 = \{3,6,7,11,12,13,14,15\}$
36	$f_1 = \overline{(x_2 \vee \overline{x_4}) \cdot x_1 \vee x_3}$	$f_2 = \{2,6,9,10,11,13,14,15\}$
37	$f_1 = \overline{x_1 \cdot \overline{x_2} \vee \overline{\overline{x_3} \vee x_4}}$	$f_2 = \{0,1,3,5,6,7,14,15\}$

Продолжение таблицы 7

1	2	3
38	$f_1 = (x_1 \vee \overline{x_4}) \cdot \overline{x_2} \vee \overline{x_1} \cdot x_3$	$f_2 = \{0,2,3,6,7,9,11,13,15\}$
39	$f_1 = \overline{x_2 \vee \overline{x_3}} \cdot x_4 \vee \overline{x_1} \cdot x_2$	$f_2 = \{1,3,6,7,8,10,14,15\}$
40	$f_1 = (\overline{x_2 \vee x_4} \vee \overline{x_2}) \cdot \overline{x_1} \cdot \overline{x_3}$	$f_2 = \{0,6,8,9,11,13,14,15\}$
41	$f_1 = (\overline{\overline{x_2} \cdot x_3} \vee \overline{x_1} \cdot x_4) \cdot \overline{x_2}$	$f_2 = \{0,1,2,3,5,7,8,10\}$
42	$f_1 = x_1 \cdot \overline{\overline{x_2} \vee x_3} \vee \overline{x_1} \cdot x_4$	$f_2 = \{0,2,4,5,8,10,12,13\}$
43	$f_1 = x_3 \cdot (x_2 \vee \overline{x_4}) \vee \overline{x_1 \cdot x_2}$	$f_2 = \{0,4,6,8,12,13,14,15\}$
44	$f_1 = \overline{x_1} \cdot \overline{\overline{x_2} \vee x_3} \vee x_4$	$f_2 = \{3,5,7,9,11,14,15\}$
45	$f_1 = \overline{x_1} \cdot x_2 \cdot \overline{(\overline{x_4} \vee x_3)} \vee x_2 \cdot x_3$	$f_2 = \{1,3,7,9,13\}$
46	$f_1 = x_2 \cdot \overline{x_4 \vee x_3} \vee \overline{x_4 \cdot x_2}$	$f_2 = \{4,6,8,12,14\}$
47	$f_1 = \overline{x_1} \cdot \overline{x_2 \vee x_4} \vee x_4 \cdot \overline{x_3}$	$f_2 = \{1,3,9,13,14,15\}$
48	$f_1 = x_1 \cdot \overline{x_2 \vee x_4} \vee x_3 \cdot \overline{x_2 \cdot x_4}$	$f_2 = \{5,6,7,8,9,11,13\}$
49	$f_1 = \overline{x_1} \cdot \overline{x_4 \vee x_1 \cdot \overline{x_3}} \vee x_1 \cdot x_2$	$f_2 = \{0,6,7,8,9,11,12,13,15\}$
50	$f_1 = \overline{x_1 \cdot \overline{x_2} \cdot x_4 \vee x_4 \cdot x_3}$	$f_2 = \{4,5,7,9,10,11,15\}$
51	$f_1 = \overline{x_4 \cdot x_2 \vee x_1 \cdot \overline{x_3}} \vee \overline{x_4} \cdot x_2$	$f_2 = \{5,7,9,11,13,15\}$
52	$f_1 = \overline{x_1 \cdot \overline{x_2}} \cdot (x_3 \vee x_1 \cdot x_4)$	$f_2 = \{3,4,5,7,9,11,12\}$
53	$f_1 = \overline{\overline{x_1} \vee x_4} \vee \overline{x_2} \cdot x_1 \vee \overline{\overline{x_4} \cdot \overline{x_3}}$	$f_2 = \{2,10,11,12,14\}$
54	$f_1 = \overline{x_3 \cdot \overline{x_4} \vee x_1 \cdot (x_2 \vee \overline{x_4})}$	$f_2 = \{3,7,8,9,10,11,12,13\}$
55	$f_1 = x_1 \cdot \overline{x_3} \vee \overline{x_2 \cdot \overline{x_1} \vee x_4}$	$f_2 = \{2,4,5,8,10,11,14,15\}$
56	$f_1 = x_1 \vee \overline{x_2 \vee x_3} \vee \overline{x_1} \cdot x_4$	$f_2 = \{6,8,10,12,14\}$
57	$f_1 = (\overline{x_1} \vee \overline{x_2}) \cdot \overline{\overline{x_4} \cdot x_3}$	$f_2 = \{0,4,13,14,15\}$
58	$f_1 = \overline{x_1 \cdot x_2 \cdot x_3} \vee x_1 \cdot \overline{x_4} \vee x_4 \cdot x_2$	$f_2 = \{5,6,8,9,11,13,14\}$
59	$f_1 = (x_3 \vee \overline{x_1}) \cdot (\overline{x_2} \cdot \overline{x_4} \vee \overline{x_3})$	$f_2 = \{3,4,5,9,10,11,14,15\}$
60	$f_1 = \overline{x_1 \cdot (\overline{x_2} \vee x_3)} \cdot (\overline{x_4} \vee x_1)$	$f_2 = \{1,2,3,4,7,8,9,10,11,12\}$

Литература

Основная

1 Олифер, В. Г. Компьютерные сети. Принципы, технологии, протоколы : учебник для вузов / В. Г. Олифер, Н. А. Олифер. – 5-е изд. – СПб. : Питер, 2016. – 992 с.

2 Бройдо, В. Л. Архитектура ЭВМ и систем : учебник для вузов / В. Л. Бройдо, О. П. Ильина. – СПб. : Питер, 2006. – 718 с.

3 Таненбаум, Э. Архитектура компьютера / Э. Таненбаум. – 5-е изд. – СПб. : Питер, 2009. - 844 с.

4 Буза, М. К. Архитектура компьютеров : учебник / М. К. Буза. – Минск : Новое знание, 2007. – 559 с.

5 Хеннесси, Д. Л. Компьютерная архитектура. Количественный подход / Д. Л. Хеннесси, Д. А. Паттерсон. – 5-е изд. – М. : Техносфера, 2016. – 936 с.

6 Степанов, А. Н. Архитектура вычислительных систем и компьютерных сетей : учеб. пособие для вузов / А. Н. Степанов. – СПб. : Питер, 2007. – 509 с.

Дополнительная

7 Старков, В. В. Компьютерное железо : архитектура, устройство и конфигурирование [справочник] / В. В. Старков. – 2-е изд., стер. – М. : Горячая линия – Телеком, 2007. – 424 с.

8 Колдаев, В. Д. Архитектура ЭВМ : учеб. пособие / В. Д. Колдаев, С. А. Лупин. – М. : ФОРУМ : ИНФРА-М, 2009. – 384 с.

9 Гук, М. Ю. Аппаратные средства IBM PC. Энциклопедия / М. Ю. Гук. – 3-е изд. – СПб. : Питер, 2006. – 1072 с.

10 Цилькер, Б. Я. Организация ЭВМ и систем : учебник для вузов / Б. Я. Цилькер, С. А. Орлов. – СПб. : Питер, 2004. – 668 с.

11 Горюнов, А. Г. Подсистема памяти микропроцессорной системы / А. Г. Горюнов. – Томск : Национальный исследовательский томский политехнический университет, 2012. – 54 с.

12 Сапожников, В. В. Теория дискретных устройств железнодорожной автоматики, телемеханики и связи : учебник для вузов / В. В. Сапожников, Ю. А. Кравцов, Вл. В. Сапожников. – 2-е изд. – М. : УМК МПС России, 2001. – 312 с.

13 Сапожников, В. В. Анализ и синтез комбинационных схем : метод. указания по курсу «Теория дискретных устройств» / В. В. Сапожников, Вл. В. Сапожников, Д. В. Ефанов. – СПб. : Питер, 2011. – 21 с.

Св. план 2019, поз. 69

Учебное издание

Скудняков Юрий Александрович
Шпак Иван Ильич

ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ И СИСТЕМЫ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Е. В. Иванюшина*
Корректор *Е. И. Костина*
Компьютерная правка, оригинал-макет *Е. Г. Бабичева*

Подписано в печать 14.10.2020. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 4,77. Уч.-изд. л. 5,0. Тираж 50 экз. Заказ 373.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
Ул. П. Бровки, 6, 220013, г. Минск