МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ И СВЯЗИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

С.Н. Мамойленко
Ю.С. Майданов

АРХИТЕКТУРА ЭВМ

Учебное пособие

Новосибирск
2024

УДК 681.3.068(075)

Мамойленко С.Н., Майданов Ю.С. Архитектура ЭВМ: Учебное пособие. – Новосибирск: СибГУТИ, 2024. – 90 с.

Учебное пособие предназначено для студентов, обучающихся по направлениям подготовки 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии», 02.03.01 «Фундаментальная информатика и информационные технологии» и изучающих дисциплину «Архитектура ЭВМ». В нём содержится материал, предназначенный для проведения практических занятий по указанному учебному курсу с целью изучения основных принципов построения электронно-вычислительных машин.

Кафедра вычислительных систем

Ил. – 32, список лит. – 3 наим.

Рецензент: доцент кафедры телекоммуникационных сетей и вычислительных средств СибГУТИ Моренкова О.И.

Для студентов, обучающихся по направлениям 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии», 02.03.01 «Фундаментальная информатика и информационные технологии».

Утверждено редакционно-издательским советом СибГУТИ в качестве учебного пособия.

© Мамойленко С.Н., Майданов Ю.С., 2024

© СибГУТИ, 2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Электронная вычислительная машина (ЭВМ) или компьютер (англ. computer – вычислитель) – это аппаратурно-программный комплекс, предназначенный для обработки информации. Под обработкой понимается: преобразование (т.е. выполнение некоторых вычислительных операций), а также ввод, вывод и хранение информации.

Подобные устройства нашли применение практически во всех областях деятельности человечества. Изначально использовались большие сложные вычислительные машины, затем более широкое распространение получили персональные компьютеры.

Персональный компьютер (ПК) (англ. personal computer, PC) – это массово применяемая ЭВМ, имеющая небольшие габаритные размеры и невысокую стоимость.

Архитектура ЭВМ – это описание принципа действия, конфигурации и взаимного соединения основных логических узлов ЭВМ.

В учебном пособии представлен материал для организации практических занятий по изучению основных принципов работы персональных компьютеров, а также получению базовых навыков системного программирования в UNIX-подобных операционных системах.

В приложении приведены задания для практических занятий, выполняемых в рамках курса «Архитектура ЭВМ» студентами, обучающимися на Кафедре вычислительных систем ФГБОУ ВО «Сибирский государственный университет телекоммуникаций и информатики».

ГЛАВА 1. КРАТКАЯ ИСТОРИЯ СОЗДАНИЯ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Первым шагом, сделанным на пути создания ПК, считается ЭВМ Altair-8800, созданная в 1975 году компанией Micro Instrumentation and Telemetry Systems (MITS), расположенной в Соединённых Штатах Америки (в городе Альбукерке, штат Нью-Мексико) (рис. 1).

Рисунок 1. Микрокомпьютер Altair-8800

ЭВМ Altair-8800 (тогда она называлась микрокомпьютером) была основана на микропроцессоре 8080 фирмы Intel, оснащена блоком питания, лицевой панелью с множеством индикаторов и оперативной памятью ёмкостью 256 байтов (!). Весь комплект тогда стоил 397 долларов и продавался в виде набора деталей, которые покупатель должен был самостоятельно собрать, используя паяльник. Программы в ЭВМ вводились переключением тумблеров на передней панели, а результаты считывались со светодиодных индикаторов.

Микрокомпьютер Altair 8800 был построен по принципу открытой архитектуры с использованием общей шины, что позволило пользователям самостоятельно разрабатывать дополнительные платы для подключения внешних устройств.

Название «Altair» придумала дочь Эда Робертса, возглавлявшего в то время компанию MITS. Так называлась звезда из популярного тогда телесериала «Звёздный поход».

В 1976 году компания Apple Computers выпустила свою модель персонального компьютера – Apple I (рис. 2), стоимостью 695 долларов. Его системная плата была прикручена к куску фанеры, и полностью отсутствовали корпус и блок питания.

Настоящий успех к компании Apple Computers пришел с выводом на рынок модели компьютера Apple II (рис. 3). Это был первый в истории персональный компьютер в пластиковом корпусе и с цветным экраном. Стоил он тогда больше тысячи долларов. В начале 1978 года на рынок вышел недорогой дисковод для дискет Apple Disk II, который ещё больше увеличил популярность этого ПК.

В начале 80-х годов XX столетия компания International Business Machines (IBM) основала в штате Флорида в городе Бока-Ратон (Boca-Raton) отделение Entry System Division, объединившее группу из 12 человек под руководством Филиппа Дона Эстриджа (Phillip Don Estridge). Главным конструктором был назначен Льюис Эггебрехт (Lewis Eggebrecht).

Рисунок 2. Компьютер Apple I

Рисунок 3. Компьютер Apple II

Целью создания этой лаборатории была разработка ЭВМ, доступной широкому кругу потребителей, которая должна иметь небольшие размеры, невысокую стоимость и производительность, позволяющую решать определённый набор задач. Другими словами, целью создания это лаборатории была разработка персонального компьютера.

Первый IBM PC (рис. 4) появился 12 августа 1981 года и стал родоначальником нового стандарта построения ЭВМ.

С тех пор уже прошло более двадцати лет, и, конечно же, многое изменилось (см., например, внешний вид современного ПК, рис. 5). Например, производительность (число операций, выполняемых в единицу времени) современных ПК по сравнению с первыми возросла в тысячи и более раз.

На сегодняшний день фирма IBM не является единственным производителем персональных компьютеров. Однако, используя термин *персональный компьютер,* часто имеют в виду именно IBM-совместимый ПК.

Рисунок 4. Первый IBM PC

Рисунок 5. Современный персональный компьютер семейства IBM PC

Контрольные вопросы

1. Расскажите об истории возникновения персонального компьютера.
2. Что представляла собой ЭВМ Altair-8800?
3. Когда была выпущена модель персонального компьютера Apple I?
4. В чём главные отличия Apple II от Apple I?
5. Какую цель ставили перед собой создатели лаборатории Entry System Division?
6. Когда появился первый IBM PC?
7. Какие изменения претерпели ПК за свою двадцатилетнюю историю?

ГЛАВА 2. БАЗОВЫЕ ЭЛЕМЕНТЫ ЭВМ. АЗЫ БУЛЕВОЙ АЛГЕБРЫ

Основными элементами, на которых строятся ЭВМ, являются простые цифровые устройства – вентили, которые могут принимать два значения – «нуль» и «единица». Чаще всего состоянию «единица» соответствует положительное напряжение (например, + 5 В) на его выходе, состоянию «нуль» – нулевое напряжение. Схемы из вентилей могут вычислять различные функции от этих двух значений. Эти элементы формируют основу для построения сложных цифровых схем.

2.1. Базовые элементы для построения ЭВМ

В основу работы вентиля положен принцип работы транзисторов, которые в определённых условиях могут работать как бинарные переключатели. На рисунке 38а изображён биполярный транзистор, имеющий три контакта: *коллектор, эмиттер* и *базу*. Если входное напряжение V_{in} на базе ниже критического значения, т.е. соответствует логическому нулю, то транзистор закрывается и выступает как сопротивление. При этом напряжение V_{out} на выходе высокое, т.е. соответствует логической единице. Если V_{in} превышает критическое значение, т.е. равно логической единице, то транзистор открывается, и V_{out} стремится к нулю. Такая схема называется **инвертером**, т.е. преобразовывает входное значение в противоположное. Для переключения транзистора из одного состояния в другое требуется некоторое время, например, несколько наносекунд.

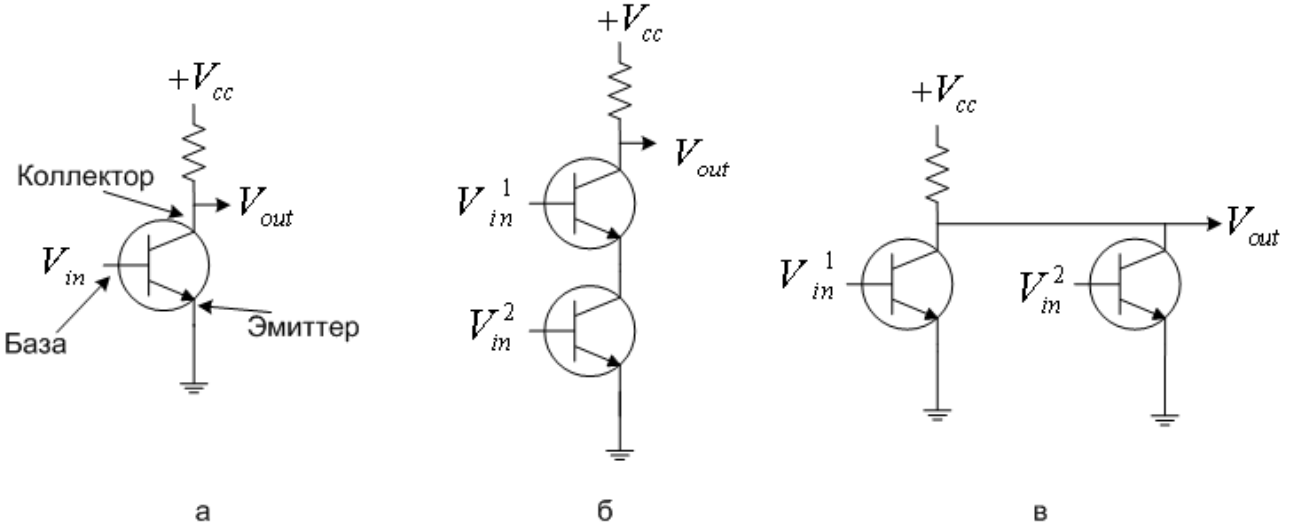


Рисунок 6. Схемы вентилей НЕ (а), И-НЕ (б), ИЛИ-НЕ (в)

Если два транзистора соединить последовательно (рис. 6б), то схема начинает выполнять функции логического сложения с инверсией. В этом случае напряжение V_{out} будет высоким только в том случае, если закрыты оба транзистора, т.е. напряжения V^1_{in} и V^2_{in} высокие и соответствуют логическим единицам.

Если два транзистора соединить параллельно (рис. 6в), то схема начинает выполнять функции логического умножения с инверсией. Если напряжения V^1_{in} и V^2_{in} высокие, то оба транзистора открыты, и напряжение V_{out} стремится к 0. Если же хотя бы одно из напряжений V^1_{in} или V^2_{in} низкое, то соответствующий

транзистор находится в закрытом состоянии, и напряжение V_{out} отлично от 0, т.е. соответствует логической единице.

Эти три схемы образуют три простейших вентиля и называются НЕ, И-НЕ, ИЛИ-НЕ соответственно. Если выход схем И-НЕ и ИЛИ-НЕ подключить на вход схеме НЕ, то получатся схемы И и ИЛИ. В первой схеме на выходе будет единица только в том случае, если на оба входа схемы подаётся единица. На выходе второй схемы получится единица, если хотя бы на один из входов подаётся единица.

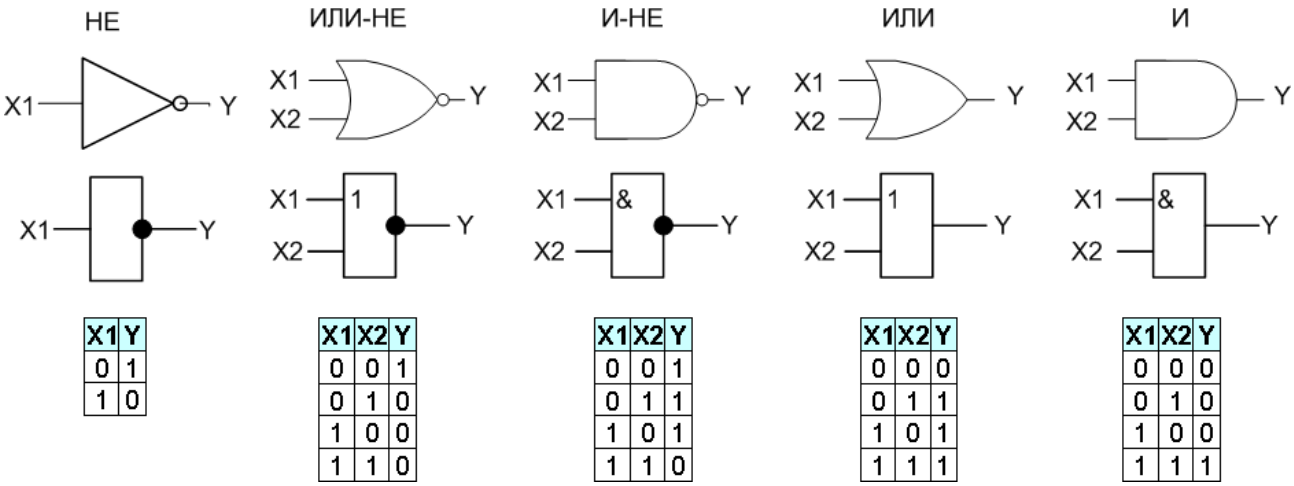
Из выше сказанного ясно, что для реализации схем И-НЕ и ИЛИ-НЕ требуется всего два транзистора, а для схем И и ИЛИ – три. По этой причине чаще всего используются только вентили НЕ, И-НЕ и ИЛИ-НЕ.

2.2. Элементы булевой алгебры

Чтобы математически описать схемы, которые строятся путём сочетания различных вентилей, используется особый тип алгебры, называемой **булевой алгеброй**, в которой все переменные и функции могут принимать только значения 0 или 1. Название эта алгебра получила в честь английского математика Джорджа Буля.

Как и в обычной алгебре, правила преобразования входных значений в выходные называются *функциями*, которые могут зависеть от одной или нескольких переменных, и получать результат (0 или 1), основываясь только на их значениях. Если функция содержит n переменных, то существует 2^n возможных наборов значений функции.

Булева функция может быть представлена двумя способами: в виде таблицы истинности и с помощью алгебраической записи.



X1	Y
0	1
1	0

X1	X2	Y
0	0	1
0	1	0
1	0	0
1	1	0

X1	X2	Y
0	0	1
0	1	1
1	0	1
1	1	0

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Рисунок 7. Значки для изображения вентилей на схемах и булевы функции, описывающие их работу (сверху вниз – европейское обозначение, российское обозначение, булева функция в виде таблицы истинности)

Если написать таблицу, в которой перечислить всевозможные комбинации входных переменных и соответствующие этим комбинациям значения функций (выходное значение), то получится **таблица истинности**. Очевидно, что размер таблицы определяется числом переменных в функции и может быть огромным (2^n, где n – число входных переменных).

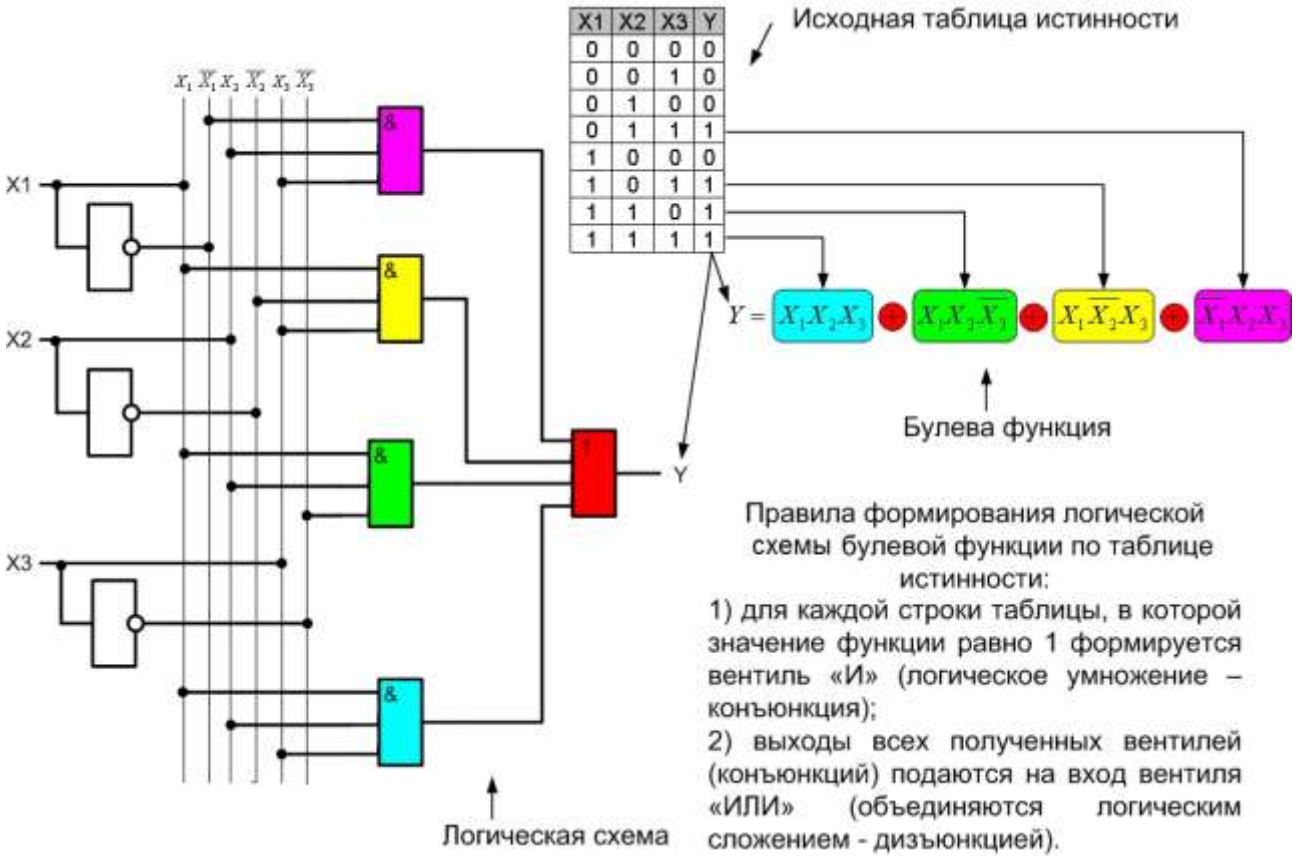


Рисунок 8. Формирование булевой функции и синтез логических схем

Как альтернатива таблицы истинности используется **алгебраическая запись**, в которой перечисляются все комбинации переменных, дающие единичное (или нулевое) значение функции. При этом знаком умножения обозначается операция И, а знаком сложения – операция ИЛИ, черта над переменной означает операцию НЕ. Например, для таблицы истинности вида:

X_1	X_2	X_3	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

функция примет вид: $Y = \overline{X_1}\,\overline{X_2}\,\overline{X_3} + \overline{X_1}\,\overline{X_2}X_3 + X_1\overline{X_2}\,\overline{X_3} + X_1\overline{X_2}X_3$. Здесь первое слагаемое образуется из инверсных значений входных переменных, так как единица на выходе соответствует их нулевым значениям, второе слагаемое – из двух инверсных и одного прямого, так как единица на выходе соответствует нулевым значениям двух переменных и единичному значению третьей и т.д.

2.3. Простейший синтез цифровых схем

Схематически вентили НЕ, И-НЕ, ИЛИ-НЕ, И, ИЛИ обозначаются, как показано на рисунке 7. Для отличия вентилей И-НЕ и ИЛИ-НЕ от И и ИЛИ на схемах первых выход обозначается кругом.

Синтез логической схемы осуществляется аналогично получению булевой функции из таблицы истинности (рис. 9). Для каждой строки таблицы истинности (или для каждого слагаемого булевой функции, если схема строится на основе неё) формируется вентиль И. Все полученные вентили объединяются через вентиль ИЛИ, в результате чего получается выход схемы.

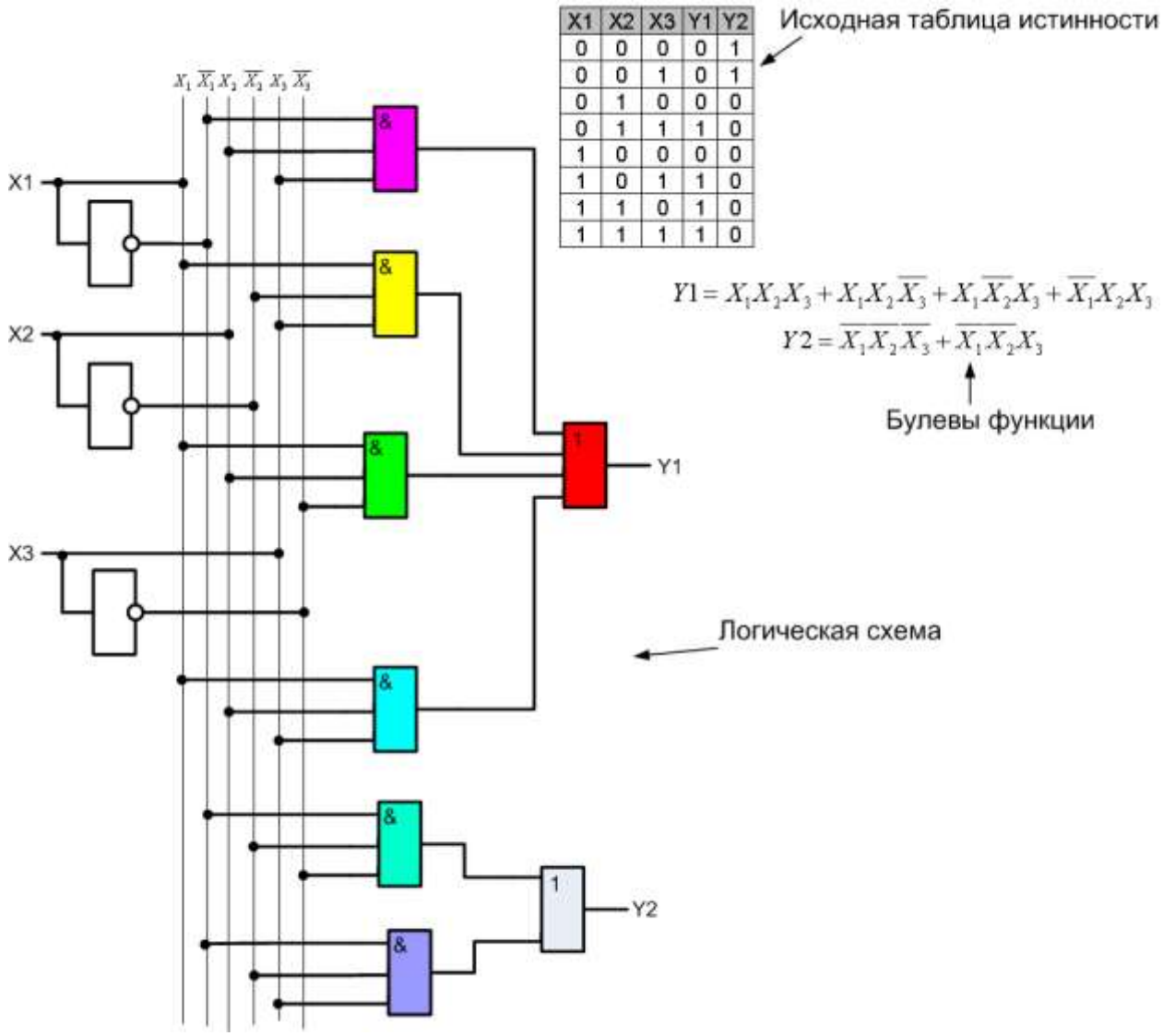


Рисунок 9. Синтез схемы для устройства, имеющего два выхода

Конечно, представленный подход не гарантирует получения оптимальных схем (имеющих минимальное число вентилей и построенных на основе минимального количества типов вентилей), однако позволяет построить схему для любой булевой функции. Вопросы организации оптимальных схем и минимизации булевых функций выходят за рамки данного пособия и поэтому рассматриваться не будут.

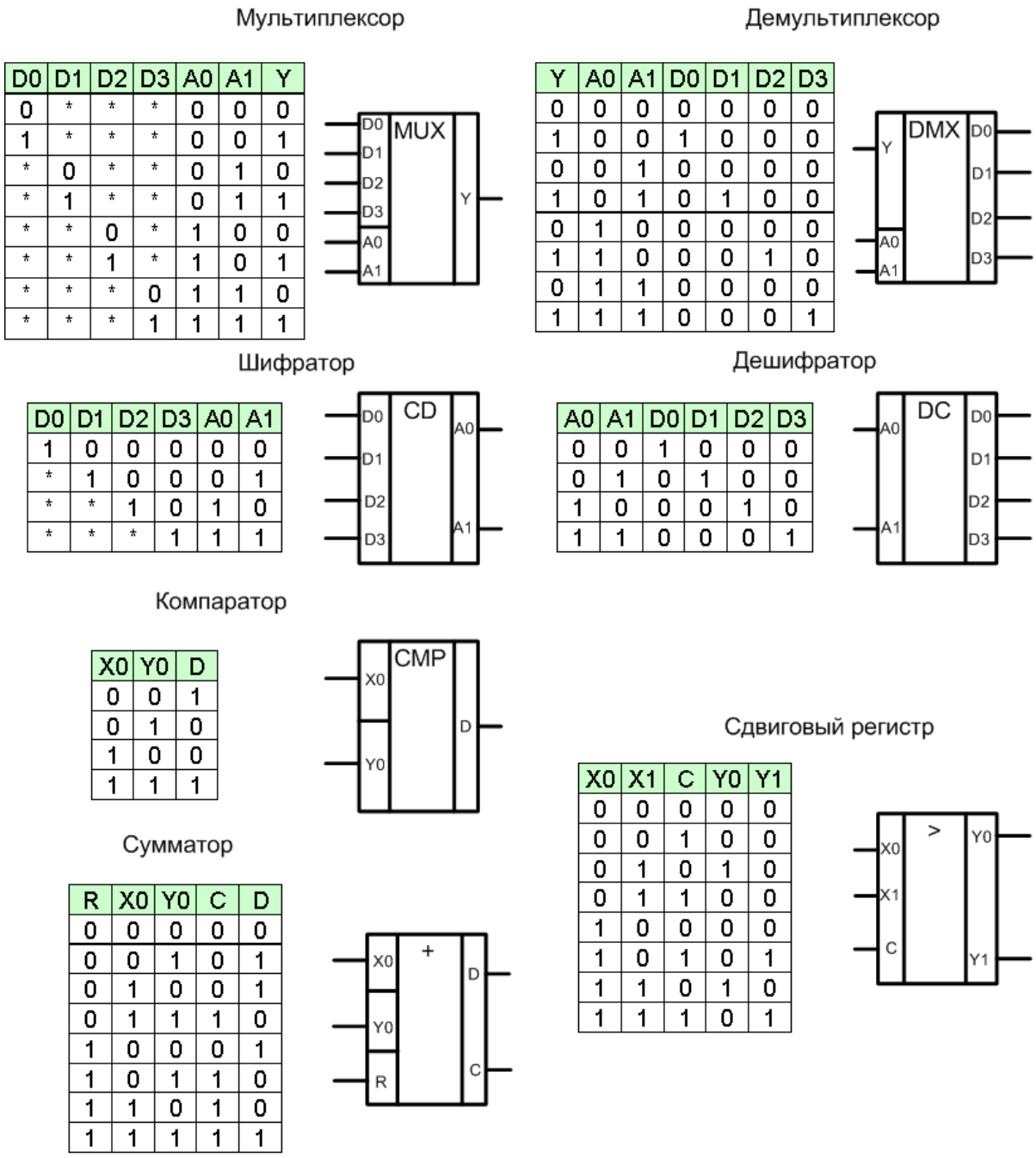


Рисунок 10. Комбинационные схемы, их таблицы истинности и обозначение на логических схемах (символ * в таблице истинности означает любое значение)

2.4. Комбинационные цифровые логические схемы

Многие применения цифровой логики требуют схемы с несколькими входными и несколькими выходными сигналами, которые получаются только исходя из значений входных сигналов. Такие схемы называются **комбинационными.** В качестве примеров можно назвать:

– мультиплексоры – устройства с 2^n информационными входами, одним выходом и n линиями управления, которые выбирают один из входов системы и соединяют его с выходом. Другими словами, мультиплексор – это цифровая схема канала для поступающей информации. Схема, обратная мультиплексору (т.е. имеющая один вход, соединяемый с несколькими выходами), называется демультиплексором;

– шифраторы – устройства, имеющие 2^n входов и n выходов, на которые выдаётся двоичное число, соответствующее номеру входа с единичным значением. Схема, обратная шифратору (т.е. сопоставляющая n-разрядное число выходу с соответствующим номером), называется дешифратором. Обычно, если единица подаётся на несколько входов шифратора, то во внимание берётся только та, которая располагается в старшем разряде;
– компараторы – устройства, имеющие $2n$ входов (т.е. входов для поступления двух n-разрядных чисел) и один выход, на котором устанавливается единичное значение, если входные числа равны (т.е. у них совпадают все разряды);
– схемы, выполняющие арифметические действия:
  - регистры сдвига – устройства, имеющие n информационных и один управляющий вход и n выходов, на которых формируется n-разрядное число, равное входному n-разрядному числу, сдвинутому вправо, если на управляющий вход подаётся единица, и сдвинутому влево, если на управляющий вход подаётся нуль. Кроме информационных выходов, имеется дополнительный выход, сигнализирующий о **переполнении**, т.е. ситуации, в которой результат операции выходит за n разрядов;
  - сумматоры – устройства, имеющие $2n$ входов и n выходов, на которых формируется результат арифметического сложения. Кроме этого, предусматривается выход, сигнализирующий о переполнении, и вход, учитывающий перенос с предыдущего разряда (R).

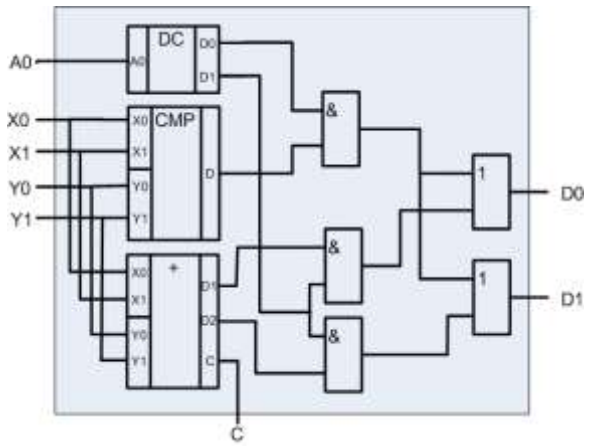


Рисунок 11. Пример схемы АЛУ, выполняющего два действия

Таблицы истинности для рассмотренных комбинационных схем приведены на рисунке 42.

В ЭВМ схемы подобного рода объединяются в одно функциональное устройство, называемое **арифметико-логическим устройством** (АЛУ). Все операции, которые может выполнять АЛУ, имеют свой уникальный номер, называемый **кодом операции**. В зависимости от того, какой код операции подаётся на вход АЛУ, будет выбрана соответствующая часть схемы (используя дешифратор и схему И) и выполнено требуемое действие. Например, на рисун-

ке 43 изображено АЛУ, выполняющее два действия – сложение и сравнение двух чисел.

2.5. Типы данных, используемые для хранения переменных в программах, написанных на языке Си

В стандарте ANSI языка Си определены следующие базовые типы переменных:

Тип	Описание
int	Знаковый целый
char	Символьный
float	Вещественный одинарной точности с плавающей запятой
double	Вещественный двойной точности с плавающей запятой

Каждый из типов определяет формат хранения данных в переменных и, соответственно, диапазон допустимых значений, которые эти переменные могут принимать. Для изменения формата хранения данных используются модификаторы типа, определяющие наличие или отсутствие в переменной знака (signed или unsigned), увеличенный или сокращённый диапазон значений (long, long long (начиная со стандарта C99) или short).

Для ввода и вывода значения переменных используются функции *scanf* и *printf* из стандартной библиотеки ввода-вывода. Чтобы определить, в каком виде выводить значения переменных, в функцию передаётся требуемый формат в виде последовательности символов % (процент) и буквы, определяющей необходимый формат:

%d – означает вывести (или ввести) целое десятичное число со знаком;

%o – целое без знака в восьмеричной системе счисления;

%x – целое без знака в шестнадцатеричной системе счисления;

%f – число с плавающей запятой и т.д.

Например, если функция *printf* вызвана следующим образом: *printf (“Значение x = %x\n”, x);*, то в стандартный поток вывода будет помещена фраза «Значение переменной x = », после чего туда же будет выведено значение переменной x в шестнадцатеричной системе счисления.

2.6. Разрядные и логические операции в языке Си. Маскирование

На практике часто приходится определять или задавать состояние устройств или управлять выполнением программы с использованием **двоичных флагов** – переменных, в которых может храниться только 0 или 1. Причем флаг считается установленным, если он содержит 1 и неустановленным – в противном случае. Примером использования флагов можно назвать программное управление выключателями: если необходимо выключатель перевести в состояние «включено», то устанавливаем флаг в 1, в противном случае – в 0.

Конечно, для представления флага можно использовать одну целую переменную. Однако чаще всего приходится иметь дело одновременно с большим количеством флагов (управлять большим количеством выключателей). В этом случае целесообразно в качестве флага использовать один разряд целой пере-

менной. Таким образом, при использовании 32-разрядных переменных, одной переменной может быть описано сразу 32 флага. Именно такой способ применяется в современных процессорах для описания их состояния (регистр флагов – ячейка памяти, содержащая несколько двоичных разрядов-флагов).

Для манипуляции отдельными битами целых переменных в языке Си используются **поразрядные операции:** И, ИЛИ, НЕ, ИСКЛЮЧАЮЩЕЕ ИЛИ, СДВИГ ВЛЕВО, СДВИГ ВПРАВО. Операция И обозначается символом &, операция ИЛИ – |, операция НЕ – ~, операция ИСКЛЮЧАЮЩЕЕ ИЛИ – ^, СДВИГ ВЛЕВО – <<, СДВИГ ВПРАВО – >>. Выполнение этих операций происходит в двоичной системе счисления, несмотря на то, в каком виде представлены её операнды. Например, результат выполнения операции 2 & 5 будет равен 0 (т.к. 010 & 101 = 0).

Чтобы получить значение определённого флага (т.е. располагающегося в определённом разряде числа) необходимо выполнить следующую последовательность действий:

flag = (registr >> (k – 1)) & 0x1,

где registr – это переменная, хранящая флаги, k – номер разряда (по порядку), в котором находится требуемый флаг. В результате этих действий переменная registr будет сдвинута вправо таким образом, что требуемый флаг окажется в младшем разряде, после чего будет произведена операция поразрядного И с единицей (т.е. все разряды числа, кроме младшего, будут умножены на 0). В переменную flag будет возвращена либо 1, либо 0, в зависимости от того, в каком состоянии был флаг.

Если требуется установить значение флага в единицу, то необходимо выполнить следующие действия:

registr = registr | (1 << (k – 1)).

Другими словами, необходимо выполнить операцию поразрядного ИЛИ между регистром флагов и числом, в котором в нужном разряде установлена 1, а в остальных разрядах числа содержатся нули.

Если требуется установить значение флага в нуль, то необходимо выполнить следующие действия:

registr = registr & (~(1 << (k – 1))).

Другими словами, необходимо выполнить операцию поразрядного И между регистром флагов и числом, в котором в нужном разряде установлен 0, а в остальных содержатся единицы. Такое число получается путем инвертирования числа, содержащего единицу в том разряде, в котором нам нужен 0.

Такая операция называется **маскированием**, а число, определяющее разряд (второй операнд) – **маской**. Ясно, что выделение разряда и установка его в нуль выполняется одинаковым образом, с тем лишь отличием, что в первом случае используется маска с единицей в требуемом разряде и остальными разрядами, равными нулю, а во втором случае – инверсная ей маска.

Очевидно, что одновременно можно работать с несколькими разрядами. Необходимо только подобрать соответствующим образом второй операнд (маску).

Контрольные вопросы

1. Что такое вентиль? Нарисуйте схемы вентилей НЕ, И-НЕ и ИЛИ-НЕ.
2. Что такое булева функция? Назовите виды представления булевых функций.
3. Напишите таблицы истинности для элементов НЕ, И-НЕ и ИЛИ-НЕ.
4. Запишите булевы функции в алгебраическом виде для тех же элементов.
5. Как осуществляется синтез логической схемы? Объясните на примере.
6. Что такое комбинационная логическая схема?
7. Запишите таблицу истинности для мультиплексора и демультиплексора. Объясните назначение этих устройств.
8. Запишите таблицу истинности для шифратора и дешифратора. Объясните назначение этих устройств.
9. Запишите таблицу истинности для сдвигового регистра. Объясните назначение этого устройства.
10. Что такое АЛУ? Нарисуйте схему АЛУ, выполняющего действия сравнения и сдвига.

ГЛАВА 3. ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ В ЭВМ

Любая информация в ЭВМ, включая текст, аудио и видео, представляется в виде последовательности нулей и единиц. Используя определённые правила, двоичные разряды формируются в числа, которые, в свою очередь, преобразуются в нужную для человека форму (текст, звук, изображение и т.п.). Важным этапом в процессе изучения принципов работы ЭВМ является понимание того, каким же образом внутри них хранится информация.

3.1. Краткая история возникновения чисел, алгебры и систем счисления

Потребность человечества в числах определялась необходимостью счёта и измерения, возникавшей в непосредственной практической деятельности. Затем число становится основным понятием математики, и дальнейшее развитие этого понятия определяется потребностями этой науки.

Первоначально числа обозначались чёрточками на материале, служащем для записи (папирус, глиняные таблички и т.д.). Числа соотносились с конкретными предметами, например, три камня, четыре палочки и т.п. Арифметические операции являлись действиями по объединению нескольких совокупностей предметов в одну или разделения одной совокупности на части. Лишь в многовековом опыте сложилось представление об отвлечённом характере этих действий, о независимости количественного результата действия от природы предметов, составляющих совокупности, о том, что, например, два предмета и три предмета составят пять предметов независимо от природы этих предметов. Тогда стали разрабатывать правила действий, изучать их свойства, создавать методы для решения задач, т.е. начинается развитие науки о числах – арифметики.

Со временем потребность человека в арифметических операциях возрастала. Для их выполнения были разработаны различные приспособления, одними из которых являются электронные вычислительные машины.

3.2. Системы счисления

Для записи чисел человечеством придуманы различные правила, называемые *системами счисления*. По этим правилам любое число представляется в виде набора специальных символов – *цифр* (от араб. сифр – нуль, буквально – пустой; арабы этим словом называли знак отсутствия разряда в числе). Получение количественного эквивалента числа осуществляется по *алгоритму замещения*, согласно которому сначала цифры заменяются их количественными эквивалентами, а затем эквивалент числа получается путём арифметических операций над эквивалентами цифр.

В зависимости от того, меняет ли свое количественное значение цифра при разном положении в числе, системы счисления можно классифицировать как *непозиционные* и *позиционные* системы.

В *системах счисления первого типа (непозиционных)* число образуется из цифр, значение которых не изменяется при различном положении цифр в числе. Примером таких систем служит *римская* система счисления. В ней в качестве цифр для составления чисел используются буквы латинского алфавита. I озна-

чает единицу, V – пять, X – десять, L – пятьдесят, C – сто, D – пятьсот, M – тысячу. Для получения числа требуется просто просуммировать количественные эквиваленты входящих в него цифр, с учётом того, что если младшая цифра идет перед старшей цифрой, то она входит в сумму с отрицательным знаком. Например, DLXXVII = пятьсот + пятьдесят + десять + десять + пять + один + один = пятьсот семьдесят семь. Или CDXXIX = минус сто + пятьсот + десять + десять + минус один + десять = четыреста двадцать девять.

В *позиционных системах счисления* количественное значение цифры определяется её позицией в числе. Номер позиции называется *разрядом*. Число цифр, используемых для представления чисел, называется *основанием*. Количественное значение числа в позиционной системе счисления, состоящего из n цифр $\{a_i\}$, $i \in \overline{0,n-1}$ (т.е. числа, имеющего вид $a_{n-1}a_{n-2} \ldots a_1a_0$), может быть получено следующим образом:

$$A_{(p)} = a_{n-1}p^{n-1} + a_{n-2}p^{n-2} + \ldots + a_1p^1 + a_0p^0, \qquad (1)$$

число n называется *разрядностью* и определяет максимальный эквивалент, который можно получить для такого числа:

$$A_{(p)}^{\max} = p^n.$$

В силу того, что ЭВМ строится на базе логических схем, которые могут иметь только два состояния – включено и выключено, то все числа в них представлены в *двоичной системе счисления*, которая по своей сути является позиционной. Набор цифр в этой системе состоит из 0 и 1 (основание равно 2). Например, число в двоичной системе может иметь вид $10100111_{(2)}$. Количественный эквивалент такого числа равен ста шестидесяти семи ($167_{(10)}$).

Очевидно, что для человека выполнение арифметических операций с двоичными числами затруднительно и неестественно. Поэтому для ввода и вывода чисел используются другие системы счисления. Наибольшее распространение получили восьмеричная, десятичная и шестнадцатеричная системы счисления и представление целых чисел в двоично-десятичной форме (англ. Binary Coded Decimal, BCD).

В *десятичной системе счисления* набор цифр включает 0, 1, 2, 3, 4, 5, 6, 7, 8 и 9. Например, число $10_{(10)}$ имеет количественный эквивалент десять ($1 \cdot 10^1 + 0 \cdot 10^0$). Эта система используется нами в повседневной жизни. Такое распространение она получила из-за того, что первоначально счёт предметов производился с использованием пальцев на человеческих руках. Как известно, у обычного человека на руках ровно десять пальцев.

В *восьмеричной системе счисления* набор цифр включает 0, 1, 2, 3, 4, 5, 6 и 7. Например, число $77_{(8)}$ имеет количественный эквивалент шестьдесят три ($7 \cdot 8^1 + 7 \cdot 8^0$), а $10_{(8)}$ равно восьми ($1 \cdot 8^1 + 0 \cdot 8^0$).

В *шестнадцатеричной системе счисления* набор цифр включает арабские цифры от 0 до 9 и 6 букв латинского алфавита – A (десять), B (одиннадцать), C (двенадцать), D (тринадцать), E (четырнадцать), F (пятнадцать). Например, число $FF_{(16)}$ имеет количественный эквивалент двести пятьдесят пять

$(15 \cdot 16^1 + 15 \cdot 16^0)$, а $100_{(16)}$ равно двумстам пятидесяти шести $(1 \cdot 16^2 + 0 \cdot 16^1 + 0 \cdot 16^0)$.

Двоично-десятичные числа – это специальный вид представления числовой информации, в основу которого положен принцип кодирования каждой десятичной цифры группой из четырёх бит. При этом каждый байт содержит одну или две цифры. Первый способ называется *неупакованное число*, второй – *упакованное*. Например, число 3456 может быть записано как неупакованное в виде $00000011\ 00000100\ 00000101\ 00000110_{(2)}$, или как упакованное – $0011\ 0100\ 0101\ 0110_{(2)}$.

3.3. Перевод чисел из одной системы счисления в другую

Для выполнения арифметических операций над числами они должны быть представлены в одной системе счисления.

Перевод чисел из *любой системы счисления в десятичную систему* осуществляется простым получением их количественных эквивалентов и записи в виде десятичного числа.

Перевод чисел из *десятичной системы в любую другую* осуществляется путём деления исходного числа на основание требуемой системы и записи остатков от деления в обратном порядке. Например, смотрите рисунок 17.

Перевод числа из шестнадцатеричной системы в двоичную систему может быть осуществлён путём представления каждой его цифры в виде двоичной тетрады и последовательной записи этих тетрад. Например, число $FF_{(16)}$ представляется как $1111\ 1111_{(2)}$. А число $3A_{(16)}$ – как $0011\ 1010_{(2)}$. Соответственно, перевод числа из двоичной системы в шестнадцатеричную систему осуществляется путём деления его на тетрады и представления каждой тетрады в виде шестнадцатеричной цифры. Например, $10111001_{(2)}$ представляется как $1011\ 1001_{(2)}$ и в шестнадцатеричной системе имеет вид $B9_{(16)}$.

Перевод числа из восьмеричной системы в двоичную систему осуществляется аналогично переводу числа из шестнадцатеричной системы, только цифры заменяются не тетрадами, а двоичными триадами. Например, $77_{(8)}$ будет представлено как $111\ 111_{(2)}$, а число $10_{(8)}$ – как $001\ 000_{(2)}$. Двоичное число при переводе в восьмеричную систему счисления сначала делится на триады, а затем каждая триада представляется в виде восьмеричной цифры. Например, $11100111_{(2)}$, делится на $011\ 100\ 111_{(2)}$ и получается $347_{(8)}$.

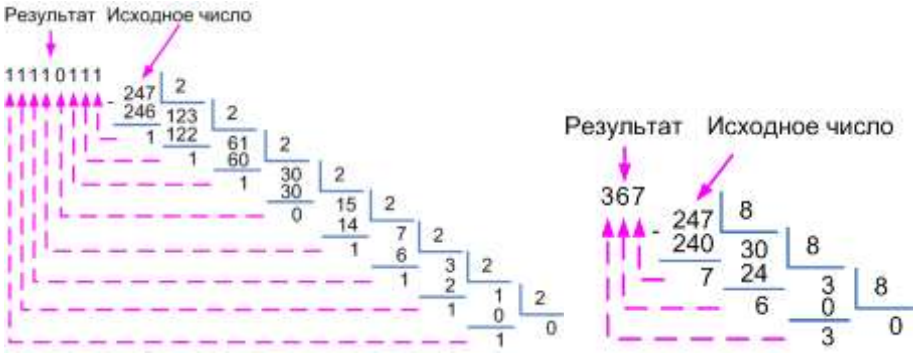


Рисунок 12. Перевод числа 247 из десятичной системы счисления в двоичную (слева) и восьмеричную (справа) системы

Перевод чисел из шестнадцатеричной системы счисления в восьмеричную систему и наоборот осуществляется в два этапа. Сначала исходное число переводится в двоичную или десятичную систему, а затем из двоичной или десятичной системы число переводится в требуемую систему счисления. Например, число $FF_{(16)}$ в двоичной системе имеет вид $1111\ 1111_{(2)}$, и в восьмеричной системе оно примет вид $377_{(8)}$.

3.4. Представление отрицательных целых и вещественных чисел в ЭВМ

Отрицательные целые числа в ЭВМ представляются в специальном виде, называемом *дополнительным кодом*, который позволяет при выполнении операций исключить отличия отрицательных чисел от положительных.

Дополнительный код отрицательного числа представляет собой результат инвертирования (замены в числе нулей на единицы и наоборот) каждого бита двоичного числа (модуля отрицательного числа) и прибавления к нему единицы.

Обратное преобразование числа из дополнительного кода в обычный вид осуществляется аналогичным образом (сначала инвертируется, затем прибавляется 1).

Например, рассмотрим число $-185_{(10)}$. Его модуль в двоичном виде имеет вид $-10111001_{(2)}$. Чтобы его перевести в дополнительный код, надо произвести его инвертирование. Причём инвертируется вся ячейка памяти, отведённая под число (будем считать, что мы работаем с 16-разрядными ячейками). В результате получится число $1111111101000110_{(2)}$. Добавляя к нему единицу, получаем число в дополнительном коде $1111111101000111_{(2)}$.

Если требуется определить, является ли число отрицательным, то необходимо проанализировать его старший разряд. Если он равен 1, то число отрицательное, если 0 – то положительное. Если считать, что результат предыдущего примера только положительный, то в десятичном виде он равен 65351. Именно поэтому изменяется диапазон отрицательных чисел, которые можно записать в n-разрядную ячейку. Например, в 8-разрядную ячейку можно записать целые положительные числа из диапазона от $0_{(10)}$ до $255_{(10)}$, а целые отрицательные из диапазона $-128_{(10)}$ до $127_{(10)}$.

Вещественные числа в ЭВМ могут быть представлены в форме с фиксированной или плавающей запятой.

Представление числа *в форме с фиксированной запятой*, которую иногда называют также *естественной формой*, включает в себя знак числа и его модуль в q-ичном коде. Здесь q – это *основание системы* или *база*. В ЭВМ чаще всего используется двоичная система, однако существуют ещё 8- и 16-ричные формы. Числам с фиксированной запятой соответствует запись вида $a_{n-1}a_{n-2}\ldots a_1a_0a_{-1}a_{-2}\ldots a_{-m+1}a_{-m}$. При этом положение запятой никак не фиксируется, а лишь подразумевается в процессе выполнения арифметических операций. Точность числа в формате с фиксированной запятой определяется числом разрядов, отводимых под дробную часть.

Получить количественный эквивалент числа с фиксированной запятой можно по формуле:

$$A_{(p)} = a_{n-1}p^{n-1} + a_{n-2}p^{n-2} + \ldots + a_1p^1 + a_0p^0 + a_{-1}p^{-1} + a_{-2}p^{-2} + \ldots + a_{-m}p^{-m}.$$

Перевод из двоичной системы счисления в шестнадцатеричную осуществляется аналогично целым числам – деление на тетрады и представление каждой тетрады в виде шестнадцатеричной цифры. Деление на тетрады осуществляется от запятой (влево для целой части и вправо для дробной).

Перевод числа с фиксированной запятой из десятичной системы в двоичную осуществляется в два этапа. На первом этапе переводится целая часть обычным образом. На втором этапе переводится дробная часть умножением её на 2 и выделением целой части на каждом шаге (рис. 13). Умножение производят до тех пор, пока не будет достигнута требуемая точность (будет получено требуемое количество разрядов после запятой) или при очередном умножении получается дробная часть, равная нулю.

Вещественные числа в формате с плавающей запятой представляются в виде двух групп цифр – мантиссы и порядка. Число представляется в виде произведения $X = \pm mq^{\pm p}$, где m – мантисса числа X, q – основание системы счисления, p – порядок числа. Форму записи чисел с плавающей запятой также называют *нормальной*. Точность числа в формате с плавающей запятой зависит от числа разрядов, отводимых под мантиссу и порядок.

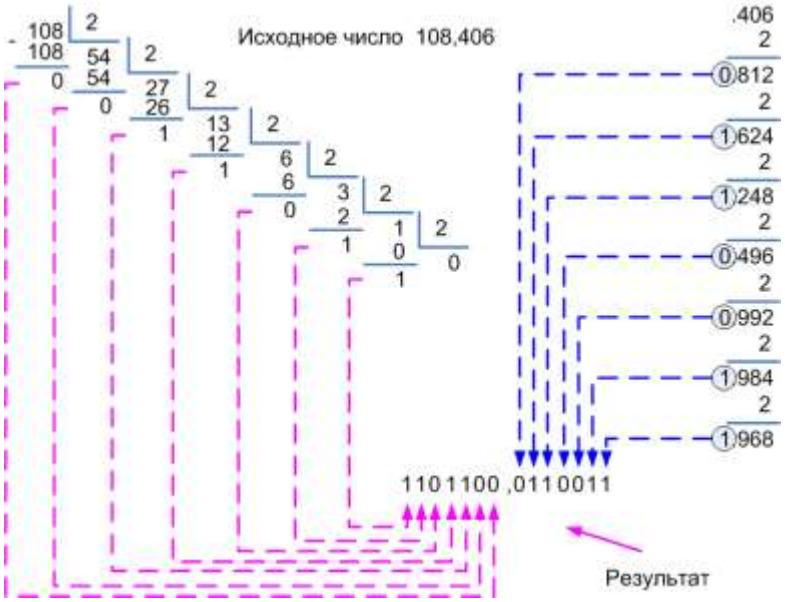


Рисунок 13. Перевод числа 108,406 из десятичной системы в двоичную систему счисления

Для представления чисел с плавающей запятой в ЭВМ был разработан стандарт IEEE 754, согласно которому выделяют два типа чисел: 32-битовое с 8-ю разрядами под порядок и 64-битовое с 11-ю разрядами под порядок. Первый тип называется одинарным или простым, второй – двойным или двойной точностью.

3.5. Представление символьной информации в ЭВМ

Каждому символу однозначно сопоставляется некоторая двоичная последовательность, называемая его **кодом**. Совокупность возможных символов и их кодов образуют **таблицу кодировки**.

В настоящее время применяется огромное количество различных кодировок символов. Общим (хотя и не обязательным) для всех систем кодирования является **весовой принцип,** согласно которому коды цифр (т.е. двоичные числа) увеличиваются по мере увеличения цифры, а коды латинских букв увеличиваются в алфавитном порядке. Например, код символа ‘1’ на единицу меньше кода символа ‘2’, а код символа ‘b’ на единицу меньше кода символа ‘c’. Требований к порядку расположения специальных символов (символов национальных языков, знаков препинания, математических символов и т.п.) обычно не предъявляется.

3.5.1. Кодировочные таблицы с фиксированной длинной кода

Самыми первыми и наиболее распространёнными являются кодировки, представляющие символы восьмиразрядными двоичными числами. При этом в таблице может содержаться не более 256 кодов различных символов. Примерами таких таблиц являются:

- расширенный двоично-кодированный код (англ. Extended Binary Coded Decimal Interchange Code, EBCDIC). Также он известен под названием ДКОИ. Кодировка EBCDIC используется в ЭВМ, производимых фирмой IBM;
- американский стандарт кода для представления информации (англ. American Standard Code for Information Interchange, ASCII), разработанный Институтом стандартизации США (ANSI).

Таблица ASCII делится на две части: базовую и расширенную. Базовая таблица закрепляет значение кодов от 0 до 127 (т.е. использует только 7 бит), а расширенная относится к символам с номерами от 128 до 255. Изначально существовала только базовая часть таблицы ASCII, а старший (восьмой) бит использовался для контроля чётности (т.е. принимал единичное значение, если в 7-битной части чётное число единиц). Основная таблица описывает 128 символов, из которых (рис. 14):

- первые 32 кода отданы производителям аппаратурных средств (в первую очередь производителям печатающих устройств). В этой области размещаются так называемые управляющие коды, которым не соответствуют никакие символы языков, и эти коды не выводятся ни на экран, ни на устройства печати, но ими можно управлять тем, как производится вывод прочих данных;
- коды с 32 по 127 описывают символы английского алфавита, знаков препинания, цифр, арифметических действий и некоторых вспомогательных символов.

Стандарт ASCII с 8 битами не определяет содержание верхней половины таблицы кодировки, которая используется разработчиками разных стран для представления символов соответствующих языков. Однако, для того чтобы обеспечить единообразие правил представления символов этой части кодовой таблицы, Международная организация по стандартизации (ISO) взяла ответственность по определению семейства стандартов, известных как семейство ISO 8859-X. Это семейство представляет собой совокупность 8-битных кодиро-

вок, где младшая половина каждой кодировки (символы с кодами 0–127) соответствует ASCII, а старшая половина определяет символы для различных языков. В зависимости от использования кодов 128–255 различают следующие вариации стандарта ISO 8859 (табл. 1).

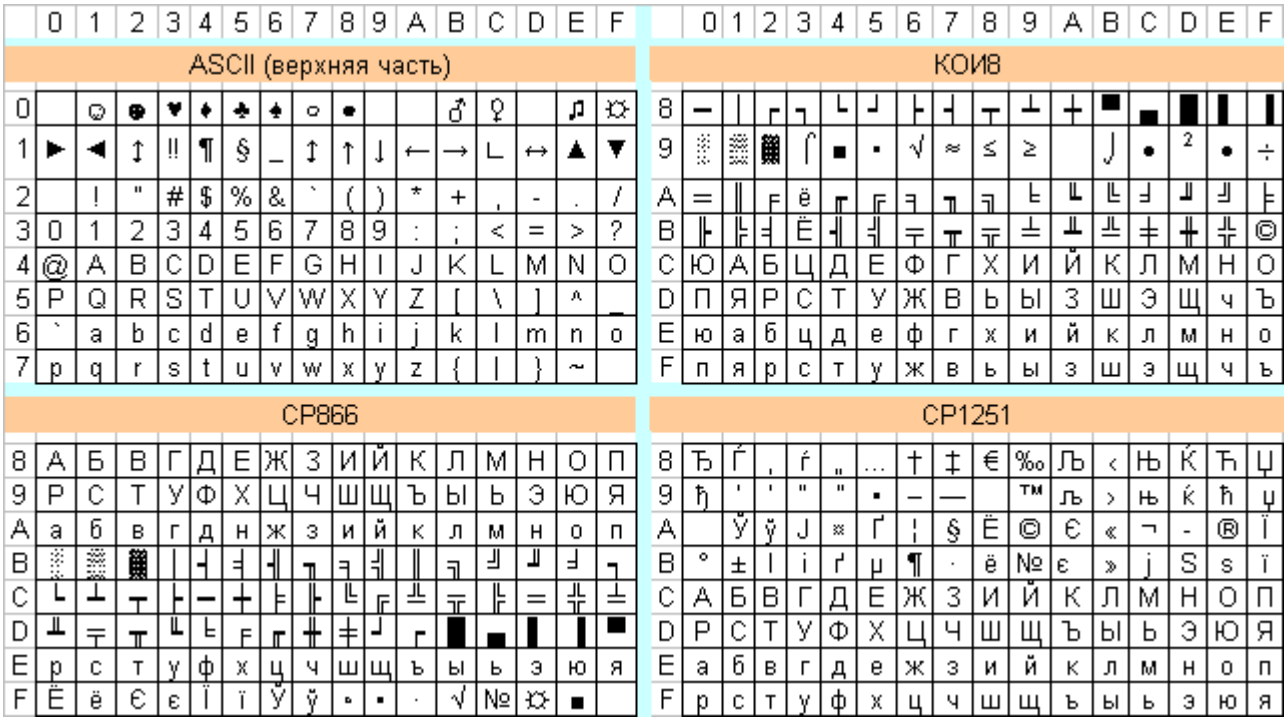


Рисунок 14. Таблицы символов

Таблица 1. Варианты стандарта ISO 8859-X

Стандарт	Характеристика
ISO 8859-0	Новый европейский стандарт (так называемый Latin-0)
ISO 8859-1	Языки западной Европы и Латинской Америки (Latin-1)
ISO 8859-2	Языки стран центральной и восточной Европы
ISO 8859-3	Языки стран южной Европы, мальтийский и эсперанто
ISO 8859-4	Языки стран северной Европы
ISO 8859-5	Языки славянских стран с символами кириллицы
ISO 8859-6	Арабский язык
ISO 8859-7	Современный греческий язык
ISO 8859-8	Языки иврит и идиш
ISO 8859-9	Турецкий язык
ISO 8859-10	Языки стран северной Европы (лапландский, исландский)
ISO 8859-11	Тайский язык
ISO 8859-13	Языки балтийских стран
ISO 8859-14	Кельтский язык
ISO 8859-15	Комбинированная таблица для европейских языков
ISO 8859-16	Специфические символы для языков: албанского, хорватского, английского, финского, французского, немецкого, венгерского, ирландского, итальянского, польского, румынского и словенского

Несмотря на то, что в ISO разработали стандарты для представления языков многих стран, разработчики программного обеспечения предпочитают пользоваться другими (собственными) кодировочными таблицами.

Одной из альтернатив стандарту ISO 8859-5 стала кодовая таблица KOI8, разработчики которой поместили символы русской кириллицы в верхней части расширенной ASCII-таблицы таким образом, что позиции кириллических символов соответствуют их фонетическим аналогам в английском алфавите в нижней части таблицы. Это означает, что если в тексте, написанном в KOI8, убрать восьмой бит каждого символа, то текст останется читаемым, хотя и выглядеть будет забавно. Например, слово «привет» будет выглядеть как «privet». Такая кодировка широко используется (например, при передаче SMS-сообщений). Следует отметить, что KOI8-R подходит только для русских текстов, и как следствие был создан украинский вариант KOI8-U.

Компания Microsoft в своих операционных системах MS-DOS и MS Windows использует свои кодовые страницы, называемые OEM (Original Equipment Manufacturer) (табл. 2):

Таблица 2. Наиболее распространенные страницы OEM

CP437	США, страны западной Европы и Латинской Америки
CP708	Арабские страны
CP737	Греция
CP866	Российская кодировка для MS-DOS
CP932	Япония
CP936	Китай
CP1251	Российская кодировка для MS Windows

К сожалению, стандарты ISO, KOI8 и OEM хотя и предназначены для одного и того же (кодирования символов национальных языков), но не согласованы между собой. Поэтому при передаче информации необходимо чётко оговаривать, с использованием какой таблицы она закодирована.

3.5.2. Универсальная кодовая таблица и системы кодирования символов

Стандарты кодирования символов с фиксированной длинной кода (ASCII и все аналогичные ему стандарты) в силу 8-битной кодировки имеет существенные ограничения по числу символов, которые могут быть закодированы. Огромное разнообразие таких кодовых таблиц привело к хаусу и полной неразберихе, которая дала понять, что необходимо как-то унифицировать кодирование символов в ЭВМ.

По этой причине в 1989 году международная организации по стандартизации (ISO) начала разрабатывать стандарт ISO/IEC 10646, получивший название Универсальной системы кодирования символов (Universal Coded Character Set[1]). В этом стандарте записываются все существующие символы и знаки

[1] Список Unicode символов - https://symbl.cc/en/unicode-table/

и каждому из них присваивается некоторый универсальный код. Кодовое пространство разбито на 17 плоскостей (англ. planes) по 2^{16} (65 536) символов. Итого стандарт позволяет закодировать 1 114 112 символов (пока этого достаточно ☺). Плоскость с номером 0 называется базовой (Basic Multilingual Plane, BMP) и содержит символы наиболее употребительных письменностей.

Для описания кода символа стали использовать общепринятую нотацию с префиксом U+ и с указанием кода символа в шестнадцатеричной форме. При этом, для символов базовой плоскости формат записи кода символа будет выглядеть как U+0030 (символ цифры 0) или U+0412 (заглавная буква В) и т.п. Для кодовой плоскости 1 запись кода принимает вид от U+10000 до U+FFFFF, и т.д. Система кодирования развивается бурно и на сегодня уже выпущено более 20 версий этого стандарта (на конец 2023 года версия стандарта – 15.1.0).

Параллельно разработкой универсальной кодовой таблицей не утихал спор на тему «а как же представлять коды символов в памяти ЭВМ». Очевидным стал тот факт, что кодирование символов с фиксированной длинной кода на практике продемонстрировало свою неэффективность. В результате появились правила форматирования кодов символов (англ. Unicode Transformation Format). Совокупность универсальной таблицы символов и правил форматирования кодов символов стали называть одним словом – **Unicode**.

Наибольшую популярность приобрели правила UTF-8. Согласно этим правилам, универсальный код символа в зависимости от его значения представляется последовательностью из 1, 2, 3 или 4 байтов. При этом, в старших разрядах первого байта кода содержится подсказка из скольки байт состоит эта последовательность, а все остальные байты последовательности в своих двух старших разрядах содержат префикс 10.

Количество требуемых байт определяется по следующему правилу:

Диапазон кодов универсальных символов	Количество байт	Префикс первого байта
00000000 -0000007F	1	0xxxxxxx
00000080-000007FF	2	110xxxxx
00000800-0000FFFF	3	1110xxxx
00010000-0010FFFF	4	11110xxx

Такая система правил позволяет получить совместимость с базовой таблицей ASCII (коды символов из этой таблицы в USC находятся в самом начале и их коды совпадают). Под символы кириллицы в UCS выделены области знаков с кодами от U+0400 до U+052F, от U+2DE0 до U+2DFF, от U+A640 до U+A69F.

Рассмотрим пример как кодируются символы в UTF-8. Например, символ Ю (большая кириллическая буква Ю) имеет код U+042E. Код символа попадает во второй диапазон, поэтому будет записываться двумя байтами (рис. 15).

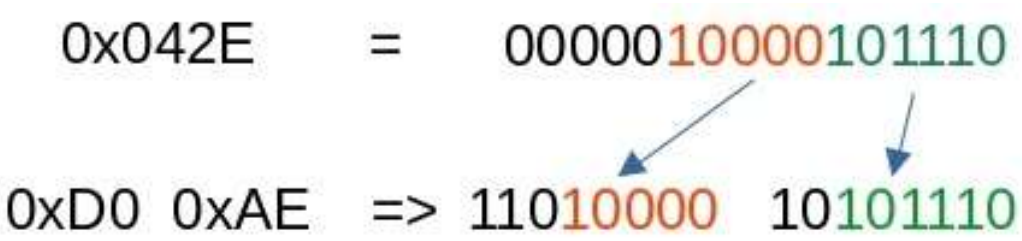


Рисунок 15. Пример кодирования универсального символа в UTF-8.

Для того, чтобы понять какая система правил форматирования символов использовалась для создания текстового файла, в самом начале файла записывается «магическая последовательность», указывающая на соответствующую UTF. Для UTF-8 такой последовательностью являются три байта со значениями: 0xEF 0xBB 0xBF.

3.6. Вывод символьной информации. Шрифты

При кодировании символьной информации в ЭВМ никак не описывается, как будет выглядеть каждый символ на экране или на бумаге. А только лишь определяются соглашения вида «если это число X, то будем понимать его как символ «буква а», а если это число Z, то будем понимать его как символ «запятая» и т.д. Для того чтобы описать, как должны выглядеть символы на экране или на бумаге, используются дополнительные правила – **шрифты**, в которых для каждого числа однозначно определяется вид соответствующего символа.

Шрифты бывают *растровые* и *векторные*. В первом случае в памяти ЭВМ хранится образ символов (растр), который при необходимости выбирается из неё и выводится пользователю. Растр (рис. 16) – это матрица определённого размера, в которой в тех ячейках, которые должны быть закрашены, помещается 1, а в остальных – 0. Во втором случае в памяти ЭВМ хранятся команды, которые надо выполнить устройству отображения, чтобы вывести требуемый символ.

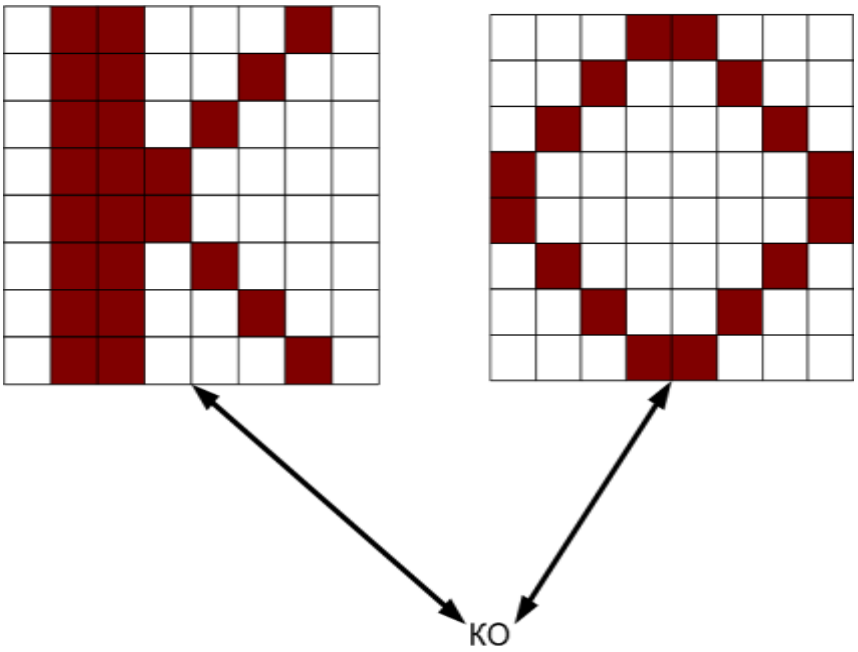


Рисунок 16. Пример растрового шрифта

В текстовом режиме вся область для вывода информации поделена на ячейки, называемые **знакоместом**. В каждую ячейку может быть выведен только один символ.

Контрольные вопросы

1. Что такое система счисления? Назовите отличия позиционных систем счисления от непозиционных. Приведите примеры систем счисления обоих видов.
2. Как перевести число из двоичной системы счисления в десятичную и наоборот? Приведите примеры.
3. Как перевести число из восьмеричной системы счисления в десятичную и наоборот? Приведите примеры.
4. Как перевести число из шестнадцатеричной системы счисления в десятичную и наоборот? Приведите примеры.
5. Как перевести число из шестнадцатеричной системы в двоичную и наоборот? Приведите примеры.
6. Как перевести число из восьмеричной системы в двоичную и наоборот? Приведите примеры.
7. Как перевести число из шестнадцатеричной системы в восьмеричную и наоборот? Приведите примеры.
8. Как представляются отрицательные числа в ЭВМ? Что представляет собой дополнительный код? Приведите примеры перевода отрицательных десятичный чисел в двоичную систему счисления.
9. Расскажите о представлении вещественных чисел в ЭВМ. В чем отличие представления таких чисел в форме с фиксированной и плавающей запятой?
10. Как представляется символьная информация в ЭВМ? Какие виды кодировок символов Вы знаете?
11. Объясните отличия стандарта ASCII и ISO 8859-X.
12. Какие кодовые страницы используются в операционных системах компании Microsoft?
13. Что такое шрифт? Какие виды шрифтов Вы знаете?
14. Какие типы данных используются в языке программирования Си для хранения переменных?
15. Что такое двоичный флаг? Каково его назначение?
16. Какие поразрядные операции существуют в языке программирования Си?
17. Что такое маска и маскирование?

ГЛАВА 4. УСТРОЙСТВА ВВОДА-ВЫВОДА ИНФОРМАЦИИ. ТЕРМИНАЛЫ

4.1. Клавиатура

Для ввода информации в ЭВМ обычно используется устройство, называемое клавиатурой. В общем случае оно представляет собой набор переключателей – клавиш, расположенных в виде прямоугольной матрицы, присоединённой к специальному процессору (рис. 17).

4.1.1. Общее устройство клавиатуры

Нажимая на клавишу, пользователь замыкает соответствующий переключатель, и, тем самым, на определённые входы процессора подаёт положительные сигналы (логические единицы). Процессор шифрует значения этих входов, получая цифровой номер нажатой клавиши, и передаёт это значение в ЭВМ. Цифровой номер клавиши называется её **скан-кодом**. Такое название принято вследствие того, что для определения нажатой клавиши процессор как будто сканирует значения своих входов, а лишь затем выполняет операцию шифрации.

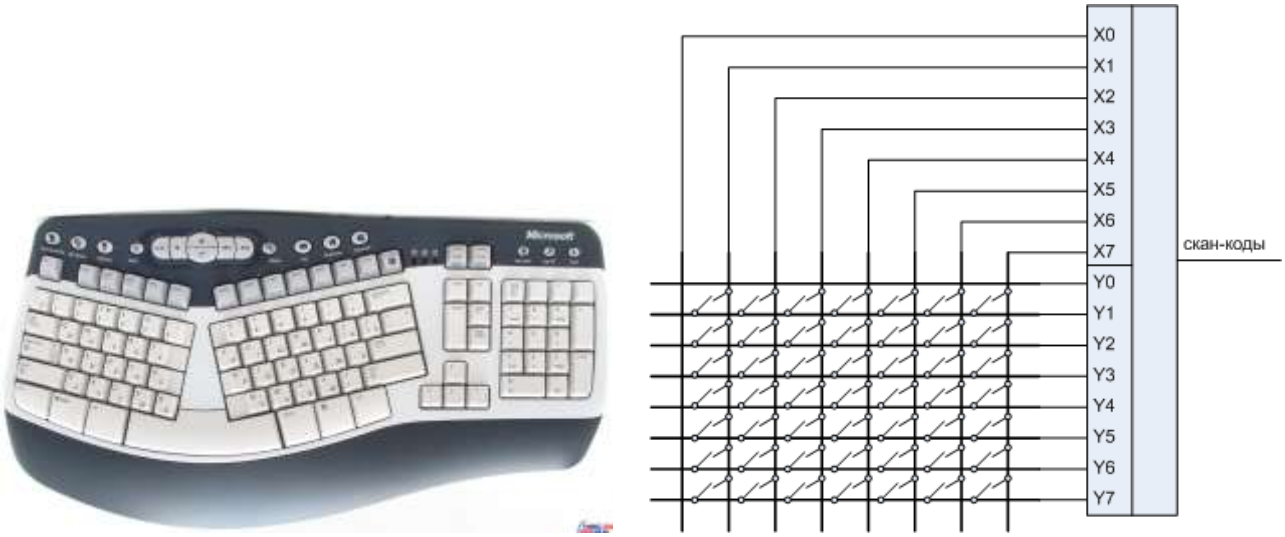


Рисунок 17. Пример клавиатуры (слева) и её простейшая схема (справа)

Скан-код клавиши может состоять из одного или нескольких байтов (см. приложение). Обычно клавиши, на которых нанесены символьные обозначения, имеют однобайтовые коды, а управляющие клавиши – многобайтовые. Таблица сопоставления кодов клавишей зависит от типа клавиатуры. В персональных компьютерах принято кодировать клавиши способом, приведённым в приложении.

Номер клавиши однозначно связан только с её местоположением в матрице (клавиатуре), и никак не зависит от нанесённых на неё обозначений. Например, скан-код 0x2B соответствует клавише с ASCII обозначениями «f», «F», «а», «А».

Обратите внимание (!!!), что скан-код клавиши и код символа – это разные вещи. Одному скан-коду в зависимости от режима работы клавиатуры мо-

жет соответствовать несколько символов или вообще может не соответствовать никакого символа, как, например, у управляющих клавиш.

Если пользователь продолжает держать клавишу нажатой и, соответственно, переключатель в замкнутом состоянии, то процессор после некоторого ожидания повторяет процедуру шифрации нажатой клавиши и передачи её кода в ЭВМ. Это сделано для того, чтобы при необходимости многократного нажатия на клавишу (например, для перемещения курсора на несколько строк или столбцов) пользователю не приходилось «долбить» по клавиатуре.

Когда пользователь отпускает клавишу или размыкает переключатель, процессор также сообщает об этом ЭВМ, используя определённое числовое значение. Обычно это значение равно значению, получаемому при нажатии клавиши только с единицей в старшем бите.

После получения скан-кода клавиши ЭВМ (а точнее соответствующее программное обеспечение – драйвер клавиатуры или пользовательская программа) сопоставляет ей определённую символьную последовательность, называемую **кодом клавиши.** Очевидно, что правила сопоставления, зависят не только от того, какая клавиша нажата, но и от того, в каком режиме работает клавиатура. Например, вводится ли русский или английский текст, была ли нажата одна клавиша или их было нажато несколько, является ли нажатая клавиша символьной (т.е. на неё нанесён символ) или управляющей (например, клавиша перемещения курсора) и т.п.

По сути, эта последовательность является числовыми кодами символов, изображённых на соответствующей клавише. Для управляющих клавиш, у которых нет символьного изображения (например, клавиши управления курсором или функциональные клавиши) формируется последовательность из нескольких символов (байтов), первым из которых идёт специальный символ, сигнализирующий о том, что нажата специальная клавиша.

Для некоторых клавиш вообще не формируются символьные последовательности, а их нажатие приводит к тому, что ЭВМ переводит клавиатуру в новый режим. Например, клавиша Shift приводит к тому, что при её удержании символьные клавиши в дальнейшем будут сопоставляться заглавным буквам.

Часто кроме клавиш клавиатура оснащена дополнительными индикаторами, предназначенными для отображения её состояния или текущего режима работы. Например, нажатие клавиши NumLock вызывает изменение функциональных назначений цифровой части клавиатуры, о чём сигнализирует соответствующий светодиодный индикатор. Для изменения состояния этих индикаторов ЭВМ взаимодействует в обратном порядке с процессором клавиатуры, передавая ему специальные команды.

4.1.2. Конструкции клавиш

В современных клавиатурах используются несколько типов клавиш:

- с механическими переключателями;
- с замыкающими накладками;
- с резиновыми колпачками;
- мембранные.

В механических переключателях происходит замыкание металлических контактов. В них для создания «осязательной» обратной связи зачастую устанавливается дополнительная конструкция из пружины и смягчающей пластинки. При этом при нажатии ощущается сопротивление клавиш, и слышится щелчок. Такие переключатели очень надёжны, их контакты обычно самоочищающиеся. Они выдерживают до 20 миллионов срабатываний и стоят вторыми по долговечности после ёмкостных датчиков (см. ниже).

Клавиши с замыкающими накладками (рис. 18) широко применялись в старых клавиатурах, например фирмы Keytronics и др. В них прокладка из пористого материала с приклеенной снизу фольгой соединяется с кнопкой клавиши. При нажатии клавиши фольга замыкает печатные контакты на плате. Когда клавиша отпускается, пружина возвращает её в исходное положение. Пористая прокладка смягчает удар при отпускании, но клавиатура при этом становится слишком «мягкой». Основной недостаток этой конструкции заключается в отсутствии щелчка при нажатии (нет обратной связи), поэтому в системах с такой клавиатурой часто приходится программным образом выводить на встроенный динамик компьютера какие-нибудь звуки, свидетельствующие о наличии контакта. Еще один недостаток такой конструкции состоит в том, что она весьма чувствительна к коррозии фольги и загрязнению контактов на печатной плате.

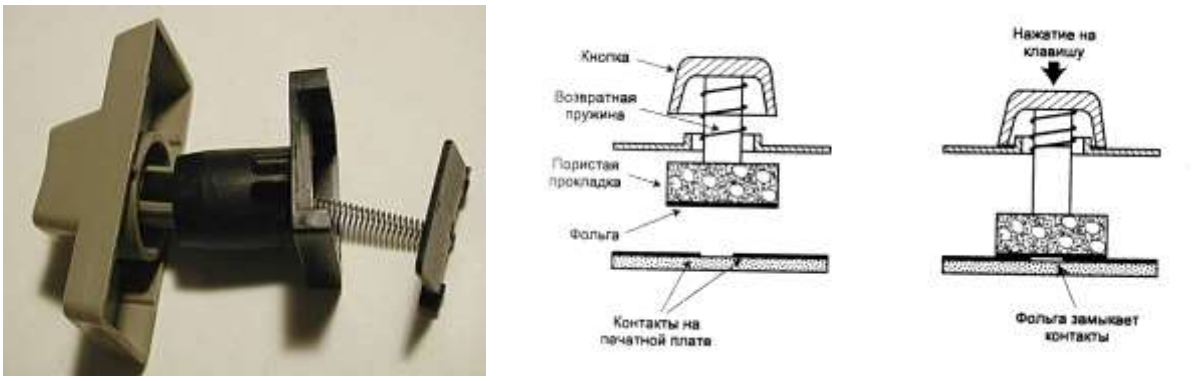


Рисунок 18. Клавиши с замыкающимися накладками

Клавиатура с резиновыми колпачками (рис. 19) вместо пружины использует резиновый колпачок с замыкающей вставкой из той же резины, но с угольным наполнителем. При нажатии клавиши шток надавливает на резиновый колпачок, деформируя его. Деформация колпачка сначала происходит упруго, а затем он «проваливается». При этом угольный наполнитель замыкает проводники на печатной плате. При отпускании резиновый колпачок принимает свою первоначальную форму и возвращает клавишу в исходное состояние. Замыкающие вставки делаются из очищенного угля, потому они не подвержены коррозии и сами по себе очищают металлические контакты, к которым прижимаются. Колпачки обычно прессуются все вместе в виде листов резины, покрывающих плату целиком и защищающих её от пыли, грязи и влаги.

Мембранная клавиатура является разновидностью предыдущей, но в ней нет отдельных клавиш: вместо них используется лист с разметкой, который укладывается на пластину с резиновыми колпачками. При этом ход каждой клавиши ограничен.

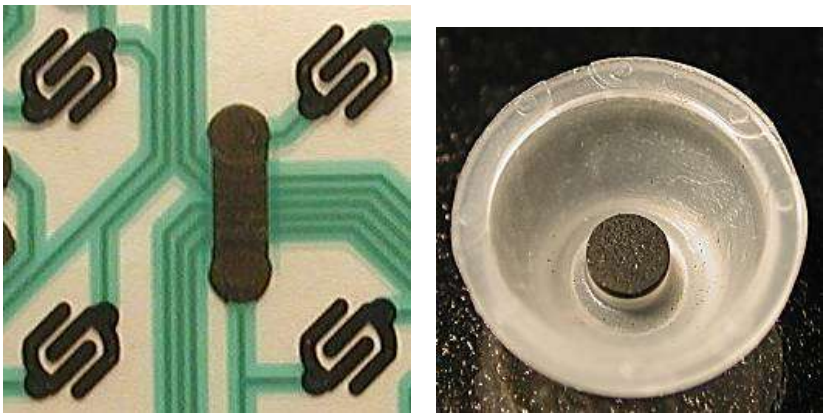


Рисунок 19. Клавиши с резиновыми колпачками

Ёмкостные датчики (рис. 20) являются единственными бесконтактными переключателями. В таких клавиатурах нет замыкающихся контактов. Их роль выполняют две смещающиеся относительно друг друга пластинки и специальная схема, реагирующая на изменение ёмкости между ними. Клавиатура представляет собой набор таких датчиков. При нажатии клавиши шток смещает верхнюю пластину ближе к неподвижной нижней пластине. Клавиши сконструированы так, что переход между пластинами происходит скачкообразно, и при этом слышен щелчок. Когда верхняя пластинка приближается к нижней, ёмкость между ними увеличивается, что регистрируется схемой компаратора, установленной в клавиатуре.

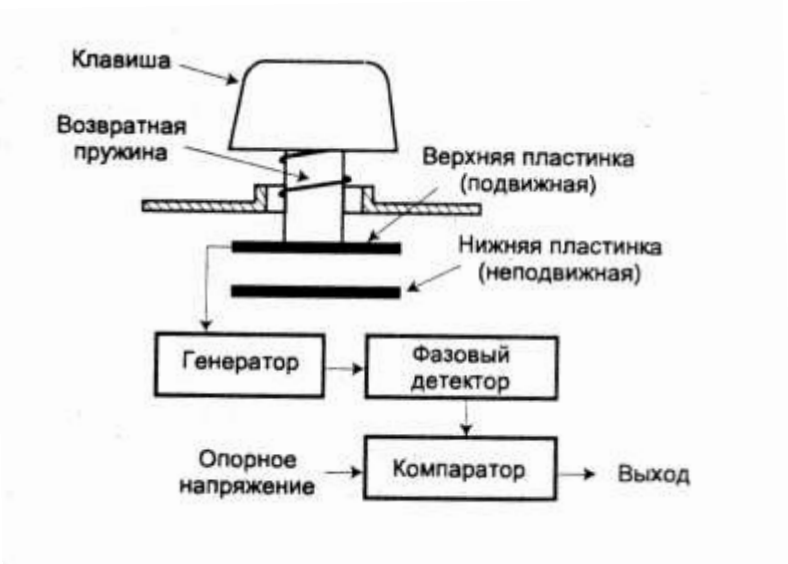


Рисунок 20. Устройство ёмкостной клавиши

Из-за отсутствия электрических контактов такая клавиатура устойчива к коррозии и загрязнению. В ней практически отсутствует дребезжание (явление, когда при одном нажатии на клавишу символ вводится несколько раз подряд). Её долговечность составляет до 25 миллионов срабатываний. Единственным недостатком такой клавиатуры является её высокая стоимость, но она во многом компенсируется удобством и долговечностью.

4.2. Монитор

Монитор, также называемый дисплеем, – это устройство, предназначенное для отображения информации. По физическому принципу получения изображения мониторы можно разделить на: электронно-лучевые (ЭЛТ, англ. Cathode Ray Tube, CRT) и жидкокристаллические (ЖКД, англ. Liquid Crystal Display, LCD).

В первом случае картинка выводится с использованием стеклянной вакуумной электронно-лучевой трубки (рис. 21).

Рисунок 21. Монитор с электронно-лучевой трубкой

С фронтальной стороны внутренняя часть стекла трубки покрыта специальным веществом – люминофором, испускающим свет при попадании на него заряжённых частиц. В качестве люминофоров для цветных ЭЛТ используются довольно сложные составы на основе редкоземельных металлов – иттрия, эрбия и т.п.

Люминофор начинает светиться под воздействием ускоренных электронов, которые создаются тремя электронными пушками, расположенными в противоположной стороне трубки. Каждая из трёх пушек соответствует одному из основных цветов и посылает пучок электронов на различные люминофорные частицы, чьё свечение основными цветами с различной интенсивностью комбинируется, и в результате формируется изображение с требуемым цветом.

Наборы точек люминофора располагаются по треугольным триадам. Триада образует пиксель – точку, из которой формируется изображение (англ. pixel – picture element – элемент картинки).

Расстояние между центрами триад называется точечным шагом монитора. Это расстояние существенно влияет на чёткость изображения. Чем меньше шаг, тем выше чёткость. Обычно в цветных мониторах шаг составляет 0,28 мм и меньше. При таком шаге глаз человека воспринимает точки триады как одну точку «сложного» цвета.

Чтобы электроны беспрепятственно достигали экрана, из трубки откачивается воздух, а между пушками и экраном создаётся высокое электрическое напряжение, ускоряющее электроны. Перед экраном на пути электронов ставится маска — тонкая металлическая пластина с большим количеством отверстий, расположенных напротив точек люминофора. Маска обеспечивает попадание электронных лучей только в точки люминофора соответствующего цвета.

На ту часть колбы, где расположены электронные пушки, надевается отклоняющая система, заставляющая электронный пучок пробегать поочерёдно всю поверхность экрана. Количество отображённых строк в секунду называется **строчной частотой развёртки**. А частота, с которой меняются кадры изображения, называется **кадровой частотой развёртки**.

Изображение на мониторах второго типа (жидкокристаллических) формируется при помощи специальных жидких кристаллов (англ. liquid crystals) – органических веществ, способных под напряжением изменять величину пропускаемого света.

Жидкокристаллический монитор представляет собой две стеклянных или пластиковых пластины, между которыми находится суспензия (рис. 22). Кристаллы в этой суспензии расположены параллельно по отношению друг к другу, тем самым они позволяют свету проникать через панель. При подаче электрического тока расположение кристаллов изменяется, и они начинают препятствовать прохождению света.

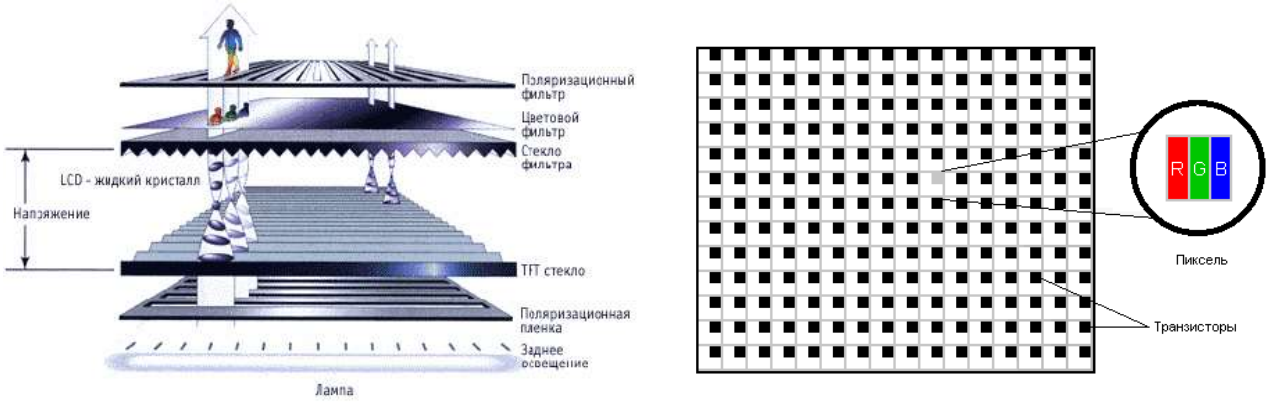


Рисунок 22. Монитор с жидкокристаллическим экраном

Существует два вида жидкокристаллических мониторов: DSTN (англ. Dual-Scan Twisted Nematic – кристаллические экраны с двойным сканированием) и TFT (англ. Thin Film Transistor – на тонкоплёночных транзисторах), также их называют соответственно пассивными и активными матрицами. Такие мониторы состоят из следующих слоёв: поляризующего фильтра, стеклянного слоя, электрода, слоя управления, жидких кристаллов, ещё одного слоя управления, электрода, слоя стекла и поляризующего фильтра.

Экран LCD-монитора представляет собой массив маленьких сегментов – пикселей. Как и в электроннолучевых трубках, пиксель формируется из трёх участков – красного, зелёного и синего. Различные цвета получаются в результате изменения величины соответствующего электрического заряда (что приводит к повороту кристалла и изменению яркости проходящего светового потока).

4.3. Видеоадаптер

Для того чтобы монитор вывел картинку на экран, её надо сформировать. Для этого в персональном компьютере используется специальное устройство, называемое видеоадаптером (рис. 23).

Рисунок 23. Вид видеоадаптера (ATI RADEON X850 XT PE)

Структура и состав видеоадаптера зависит от фирмы-производителя, однако в общем виде это устройство состоит из следующих компонентов: памяти, микропроцессора, шинного интерфейса, цифро-аналогового преобразователя (ЦАП, англ. Digital-to-Analog Converter, DAC) и постоянного запоминающего устройства (ПЗУ).

Память служит непосредственно для хранения изображения, представляемого в виде набора точек – экранных пикселей. От объёма памяти зависит максимально возможное разрешение видеокарты, определяемое как произведение трёх величин: количества столбцов, воспроизводимых на экране, количества строк и количества возможных цветов каждой точки. Таким образом, для разрешения 640x480x16 достаточно всего 256 КБ. А для разрешения 1024x768x65536 уже требуется как минимум 2 МБ.

Многие современные мониторы используют для получения управляющих команд аналоговые сигналы, для формирования которых используется ЦАП.

Для начального запуска видеоадаптера и организации дальнейшего управления им используются специальные программы и данные, находящиеся в ПЗУ видеоадаптера.

Многие современные видеоадаптеры кроме функции генерации сигналов для монитора выполняют функции по обработке самого изображения. Зачастую это наложение нескольких изображений друг на друга. Например, курсора, текстуры и т.д. Для выполнения таких операций видеоадаптер оснащён собственным микропроцессором.

Шинный интерфейс предназначен для организации взаимодействия видеоадаптера с остальной частью ЭВМ. Наибольшее распространение получили интерфейсы типа AGP, PCI, PCI-X.

Для подключения современных мониторов используется два вида разъёмов (рис. 24): D-sub и DVI. Первый используется для подключения аналоговых мониторов (которые получают сигнал в аналоговой форме), второй – для подключения цифровых мониторов (получающих сигнал в цифровой форме).

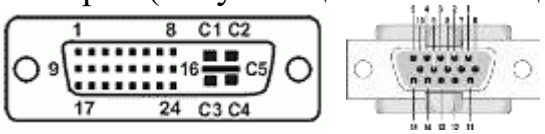


Рисунок 24. Разъёмы для подключения мониторов (DVI – слева, D-sub – справа)

4.4. Терминалы – устройства ввода и вывода информации

Часто для взаимодействия с ЭВМ, в составе которых не предусмотрены собственные средства для взаимодействия с оператором (или они есть, но имеют скудный набор возможностей), используются специальные устройства, называемые **терминалами**.

Чаще всего терминалы образуют единое устройство, соединённое с ЭВМ (или каким-то другим устройством) через кабельные или телефонные каналы (рис. 25). При этом к одной ЭВМ может подключаться несколько терминалов одновременно.

Примерами терминалов могут служить: POS-терминалы, операторские консоли по управлению промышленным оборудованием и т.п.

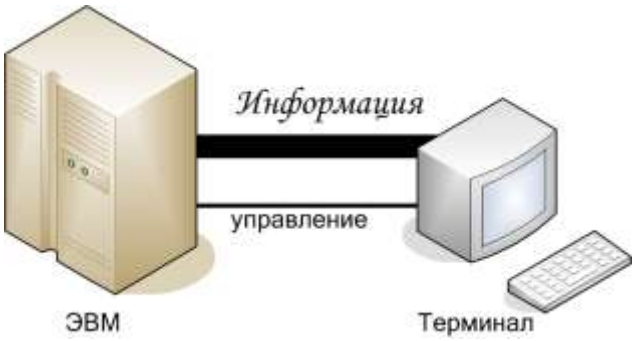


Рисунок 25. Использование терминала для доступа к ЭВМ

Терминалы различаются возможностями устройств, входящих в их состав (т.е. сколько клавиш на клавиатуре, может ли монитор выводить графическую информацию или только текст, используется ли цвет для вывода информации на монитор и т.п.).

Первые терминалы были вроде дистанционно управляемых пишущих машинок, которые могли только «отображать» (печатать на бумаге) символьный поток, посланный им из компьютера. Самые ранние модели назывались телетайпами, они могли выполнять перевод строки и возврат каретки точно так же, как обыкновенная пишущая машинка. Такие терминалы стали называть пассивными, так как они только выводят информацию и не могут самостоятельно обрабатывать её.

Современные терминалы могут выполнять значительно больше действий, включающих ввод и вывод текстовой и графической информации, использование дополнительных каналов ввода информации (например, манипулятор «мышь»), выполнять дополнительные функции (например, распознавание штрих-кода) и т.п. Состав и структура терминала определяется областью применения. Терминалы, способные проводить первичную обработку информации, называются активными. Далее будет рассматриваться именно такие терминалы.

Независимо от назначения и, соответственно, структуры терминала принцип его работы следующий (рис. 25).

Человек-оператор, нажимая клавиши на клавиатуре терминала, вводит информацию, которая поступает в устройство управления терминалом (УУТ). Далее она передаётся в ЭВМ в цифровом (численном) виде (скан-коды или

управляющие последовательности), и поступает на вход специальной программе – драйверу, который её обрабатывает и передаёт выполняющейся программе (взаимодействующей с терминалом и ожидающей прихода от него данных). При необходимости информация, поступающая от клавиатуры, может сразу дублироваться на экране терминала. Такой режим называется «ЭХО».

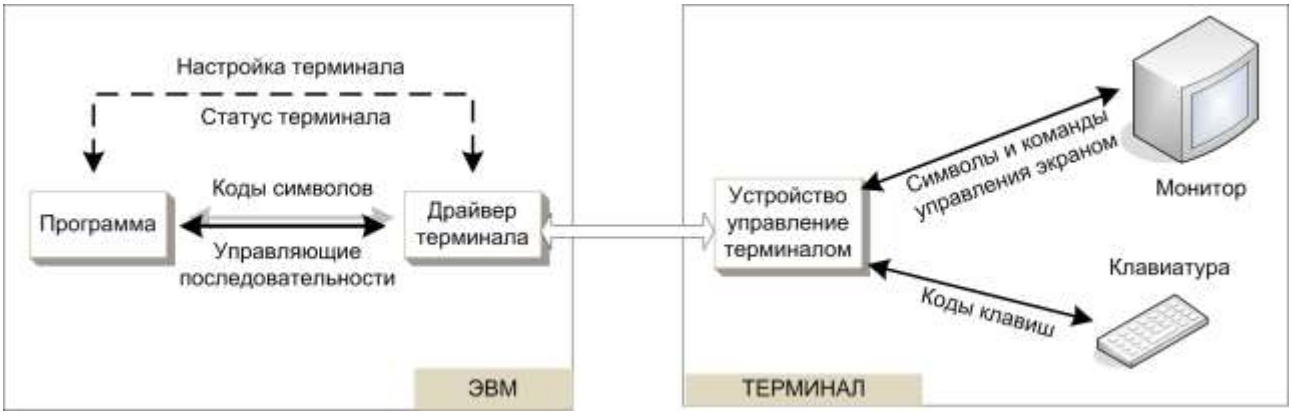


Рисунок 26. Принцип работы терминала

Программа пользователя выводит результаты своей работы на терминал через драйвер, который передаёт её УУТ, а тот, в свою очередь, вырабатывает необходимые управляющие сигналы для монитора и выводит полученную информацию.

Ввод информации через терминал может осуществляться в одном из **двух режимов**: *каноническом*, в котором информация передаётся в ЭВМ только в виде законченных строк (т.е. после нажатия клавиши «ВВОД» или «ENTER»); и *неканоническом*, при котором вводимая информация сразу поступает в ЭВМ.

Помимо информационного потока между драйвером и УУТ передаются управляющие сообщения, которые позволяют настраивать УУТ на нужный режим работы и получать его текущие настройки. Программа пользователя также может управлять терминалом (например, передвинуть курсор на экране или изменить цвет выводимых символов) путём передачи ему специальных последовательностей символов, называемых **командами терминала** или **управляющими кодами**. Взаимодействовать напрямую с клавиатурой и монитором программа пользователя не может.

Формат и назначение команд терминала определяются его типом и (обычно) описываются производителем терминала в руководстве пользователя и в соответствующей программной среде (например, в ОС семейства UNIX (см. приложение) используется база данных настроек терминала, называемая terminfo).

Управляющие коды (или управляющие символы) обычно состоят из первых 32 байтов алфавита ASCII. Они включают коды таких символов: возврат каретки (переместить курсор к левому краю экрана), перевод строки (переместить курсор вниз на одну строку), возврат на один символ, символ ESC, табуляция и звонок. Эти символы обычно не показываются на экране.

Так как не имеется достаточного количества управляющих кодов, чтобы делать всё, используются команды терминала, чаще всего называемые **Escape-**

последовательностями. Они состоят из нескольких подряд идущих символов, первым из которых является символ с кодом ASCII, равным 27, называемый «Escape» или ESC. Именно из-за первого символа команды терминала называются Escape-последовательностями.

После получения символа ESC, УУТ исследует символы после него так, чтобы интерпретировать их и выполнить соответствующую команду. Когда распознаётся конец последовательности, дальнейшие полученные символы отображаются на экране (если дальше опять не следует команда). Некоторые Escape-последовательности могут иметь параметры (или аргументы), например, координаты на экране, куда надо переместить курсор. Параметры являются частью Escape-последовательности.

Современные персональные компьютеры имеют клавиатуру и монитор, непосредственно подключённые к их системному блоку. Функции по управлению этими устройствами возлагаются на операционную систему, которая воплощает в себе УУТ и драйвер терминала. Хотя производительность ПК достаточно высока, и возможности по вводу и выводу информации практически неограниченны, они могут использоваться как терминалы для доступа к другим ПК или иным устройствам, к которым они подключены (например, видеосерверам, сетевым принтерам, модемам и т.п.). Для этого используются специальные программы, которые представляют ПК в виде псевдо- или виртуального терминала. Примером таких программ служат HyperTerminal, Remote Desktop, PuTTY, xterm и т.п. В некоторых операционных системах, например Linux, эмуляция терминала используется для того, чтобы позволить запустить несколько программ, выводящих различную информацию. При этом пользователь как будто работает за несколькими терминалами, поочерёдно переключаясь (например, нажимая комбинацию клавиш ALT+F1, или CTRL+ALT+F1) то на один, то на другой.

Используя графический режим вывода информации на монитор и программы эмуляции текстовых терминалов, пользователь может одновременно запустить несколько виртуальных терминалов.

4.4.1. Терминальные управляющие последовательности

Для описания всех доступных команд терминала и его возможностей в ОС Linux используется специальная база данных, называемая «termcap» (или «terminfo»). Она содержит разделы (отдельные файлы) для каждой модели терминала, в которых перечисляются все управляющие последовательности данного терминала и их синтаксис. Все разработчики терминалов должны придерживаться правил, указанных в этой базе.

Чтобы получить список доступных команд для определённого терминала можно использовать команду **infocmp** с параметрами –1L тип_терминала. Формат её вывода имеет вид:

поле = значение,

где «поле» – это название команды (например, clear_screen), «значение» – символьная последовательность, которую нужно отправить терминалу, чтобы выполнить указанную команду. Подробный перечень полей, которые могут быть

описаны в базе termcap, можно найти в электронном справочнике man (man terminfo).

Символьная последовательность в поле «значение» может быть указана полностью, или приведены правила для её формирования. Для представления специальных символов используются записи вида \E (символ с кодом в ASCII 27 – ESCAPE) или ^X (символ с кодом ASCII, рассчитываемом по формуле ‘X’ – ‘A’ + 1). Для указания правил используются форматные строки, аналогичные тем, что применяются в функции *printf* для указания формата вывода. Рассмотрим пример базы для терминала linux (текстового терминала).

Формат команды для получения содержимого базы следующий:

```
infocmp -1L linux
```

Результат выполнения следующий (часть базы):

```
#     Reconstructed via infocmp from file: /usr/share/terminfo/l/linux
linux|linux console,
      acs_chars=+\020\054\021-
\030.^Y0\333`\004a\261f\370g\361h\260i\316j\331k\277l\332m\300n\305o~p\30
4q\304r\304s_t\303u\264v\301w\302x\263y\363z\362{\343|\330}\234~\376,
      bell=^G,
      carriage_return=^M,
      clear_screen=\E[H\E[J,
      cursor_invisible=\E[?25l\E[?1c,
      cursor_visible=\E[?25h\E[?8c,
      enter_alt_charset_mode=\E[11m,
      enter_blink_mode=\E[5m,
      enter_bold_mode=\E[1m,
      exit_alt_charset_mode=\E[10m,
      key_f1=\E[[A,
      orig_colors=\E]R,
      restore_cursor=\E8,
      set_a_background=\E[4%p1%dm,
      set_a_foreground=\E[3%p1%dm
```

В результате получен список управляющих последовательностей, которые «понимает» терминал. Например, поле clear_screen (очистка экрана) содержит значение: \E[H\E[J. Это значит, что, если вывести на экран последовательность из 6 символов (\E, [, H, \E, [, J), то произойдёт очистка экрана, и курсор переместится в левый верхний угол экрана. Интерес вызывает команда set_a_background (установить фон), значение которой равно: \E[4%p1%dm, что означает, что команда должна формироваться с использованием одного параметра (%p1), имеющего целый тип (%d) и располагающегося между символами ‘4’ и ‘m’. Чаще всего текстовые терминалы поддерживают до 8 цветов, каждый из которых имеет свой номер. Например, если требуется установить цвет фона в белый (код 7), то команда должна иметь вид: \E[47m.

4.4.2. Основная и дополнительная таблица кодировок символов в терминалах

Существует множество различных типов терминалов, каждый из которых имеет свои особенности. Например, использует различные кодировочные таблицы, обрабатывает разный набор управляющих последовательностей и т.п. Одной из общих для всех текстовых терминалов проблем являются правила вы-

вода символов псевдографики (символов, позволяющих выводить текстовые рамки). Суть проблемы заключается в том, что расположение символов псевдографики в таблицах кодировок никак не регламентировано. Для того чтобы позволить выводить символы псевдографики в любых типах терминалов (если они, конечно, имеют такую возможность), разработали универсальное правило, согласно которому в терминалах используется дополнительная кодировочная таблица, в которой располагаются требуемые символы. Чтобы переключить терминал на использование дополнительной таблицы, необходимо послать ему управляющую команду, указанную в поле enter_alt_charser_mode. Команда, необходимая для обратного переключения, хранится в поле exit_alt_charset_mode.

Чтобы определить место расположения символов псевдографики, используют строку соответствия символов (поле acs_chars) тому расположению, которое применяется в терминале типа VT100. В нём для вывода символом псевдографики используются символы из строки `afgijklmnopqrstuvwxyz{|}~. Значение каждого символа можно найти в электронном справочнике по команде terminfo.

4.5. Взаимодействие с терминалом в ОС Linux

Для взаимодействия пользователей с ПК, работающим под управлением операционной системы Linux, используется эмуляция нескольких текстовых и графических терминалов. Т.е. другими словами, даже если пользователь сидит непосредственно за ПК, то он как будто работает за терминалом, подключённым к ЭВМ.

В ОС Linux пользователи «изолируются» от аппаратурной части персонального компьютера. Для доступа к устройствам используется единый интерфейс в виде специальных файлов устройств, которые связывают приложения с соответствующими драйверами. Вся работа с устройством происходит через этот файл, а соответствующий ему драйвер обеспечивает выполнение операций ввода-вывода согласно конкретным протоколам обмена данными между ЭВМ и устройством. Причём правила работы с файлами устройств такие же, как и правила работы с файлами на запоминающем устройстве, с некоторыми дополнениями, позволяющими выдавать управляющие воздействия.

Все устройства, подключаемые к ЭВМ, условно можно разделить на два класса:

- блочные устройства, т.е. передающие и принимающие данные большими фрагментами, называемыми блоками или пакетами. При этом ядро операционной системы производит необходимую буферизацию. Примером физических устройств, соответствующих этому типу файлов, являются жёсткие диски;
- символьные устройства, т.е. использующие побайтную передачу данных. Терминалы относятся именно к таким устройствам.

Заметим, что это разделение условно, так как одно и то же устройство может иметь и символьный, и блочный интерфейс (т.е. уметь передавать данные как побайтно, так и блоками).

Все специальные файлы устройств хранятся в каталоге /dev. Например, файл, соответствующий устройству «жёсткий диск», имеет имя hda, а файл, соответствующий первому виртуальному терминалу – tty0.

Для взаимодействия со специальными файлами устройств используются функции прямого доступа к файлам – *open*, *read*, *write*, *close*. Для управления устройством применяется системный вызов *ioctl*. Для работы с терминалами – дополнительные функции *isatty*, *tcgetattr*, *tcsetattr*, позволяющие произвести настройку драйвера на необходимый режим работы и определить его текущее состояние.

Обратите внимание (!!!), что все эти функции являются системными, в отличие от функций *fopen*, *fread*, *fwrite*, *fclose*.

4.5.1. Вызов open

Системный вызов *open* используется для открытия файла. В качестве параметров в функцию передаются строковая константа, соответствующая имени открываемого файла, режим открытия и дополнительные параметры.

Описание функций *open* и *close*

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open (const char *pathname, int flags);
int close (int fd);
```

В результате работы функции возвращается положительное целое число – номер дескриптора открытого файла, или –1, если во время открытия файла произошла ошибка. Дескриптор файла – это структура, описывающая файл. Хранится она в памяти пользовательской программы и может быть доступна по номеру дескриптора, который передаётся в качестве аргумента для функций, работающих с устройством.

Параметр *flags* определяет, в каком режиме требуется открыть файл. Он является целым числом, в котором за каждым битом однозначно закреплён один требуемый режим. Для наглядности определения требуемых режимов в заголовочном файле *fcntl.h* библиотеки функций работы с файлами заданы соответствующие макросы. Например, режим открытия для чтения описывается макросом *O_RDONLY*, режим записи – *O_WRONLY*, режим одновременной и записи и чтения (т.е. произвольного доступа) – *O_RDWR*. Некоторые режимы могут комбинироваться. Например, режим записи может комбинироваться с режимом дополнения файла, т.е. *flags* будет выглядеть как *O_APPEND | O_WRONLY*.

Обратной по действию функции *open* является функция *close*, которая закрывает файл с указанным дескриптором.

Обратите внимание (!!!), что системные вызовы, в отличие от вызовов стандартной библиотеки Си, в качестве параметров принимают целое значение – номер дескриптора, а не указатель на структуру FILE.

При запуске каждой программы автоматически открываются три файла, соответствующие устройствам: ввода, вывода и вывода ошибок. В обычном

случае все эти файлы соответствуют терминалу, с которого запущена программа. При необходимости пользователь может изменить эти устройства, например, перенаправив вывод программы в файл на диске или на принтер. Дескрипторы этих устройств имеют номера 0, 1 и 2, соответственно. Поэтому все дескрипторы, создаваемые программами пользователей, нумеруются с цифры 3.

4.5.2. Вызовы isatty, ttyname

Для того чтобы проверить, является ли файл, описываемый некоторым дескриптором, специальным файлом терминала, используется вызов *isatty*. В качестве входного параметра эта функция принимает номер дескриптора файла, который надо проверить, и возвращает 1, если этот дескриптор связан с файлом терминала и 0 – в противном случае.

Описание функции *isatty* и *ttyname*

```
#include <unistd.h>

int    isatty  (int fd);
char  *ttyname (int fd);
```

Чтобы определить точное имя файла терминала, который был открыт под определённым номером, используется вызов *ttyname*. В качестве входного значения эта функция принимает номер дескриптора и возвращает указатель на строку, содержащую имя соответствующего файла, или NULL, если возникает ошибка (например, дескриптор не связан с файлом терминала).

Например, программа, проверяющая, какие терминалы автоматически открыты для потоков ввода, вывода и ошибок при её выполнении, представлена в листинге 1.

Листинг 1. Определение терминалов для потоков ввода, вывода и ошибок

```
1)  #include <stdio.h>
2)  #include <unistd.h>

3)  /* Основная функция программы */
4)  int main (void){
5)    /* Проверяем, является ли дескриптор 0 файлом терминала */
6)    if (isatty(0)){
7)      printf  ("Поток  ввода  связан  с  терминалом  [%s]\n",
                  ttyname(0));
8)    } else {
9)      printf ("Поток ввода не связан с терминалом.\n");
10)   }
11)   /* Проверяем, является ли дескриптор 1 файлом терминала */
12)   if (isatty(1)){
13)     printf  ("Поток  вывода  связан  с  терминалом  [%s]\n",
                  ttyname(1));
14)   } else {
15)     printf ("Поток вывода не связан с терминалом.\n");
16)   }
17)   /* Проверяем, является ли дескриптор 2 файлом терминала */
18)   if (isatty(2)){
```

```
19)    printf ("Поток ошибок связан с терминалом [%s]\n",
                  ttyname(2));
20)   } else {
21)     printf ("Поток ошибок не связан с терминалом.\n");
22)   }
23)   return (0);
24) }
25)
```

В строках 1–2 подключаются заголовочные файлы библиотек *stdio* и *unistd*, в которых описываются функции *printf*, *isatty*, *ttyname* используемые в программе.

Строка 4 определяет основную функцию программы. Её имя *main*. Она не принимает ни одного параметра и возвращает в операционную систему целое значение (результат работы программы).

В строках 6–10 проверяется, связан ли дескриптор стандартного потока ввода (он имеет номер 0) с файлом терминала (строка 6). Если он связан, то в стандартный поток вывода передаётся строка «Поток ввода связан с терминалом» и выводится полное имя этого файла (строка 7). В противном случае в поток вывода передаётся строка «Поток ввода не связан с терминалом» (строка 9).

В строках 11–22 аналогичным образом проверяется соответствие дескрипторов стандартных потоков вывода и вывода ошибок к подключению к терминалу.

Строка 23 завершает работу программы с результатом 0 (т.е. программа завершилась корректно).

4.5.3. Вызовы read, write

Чтение данных из устройства (точнее из его специального файла) производится с помощью функции *read*, которая в качестве параметров получает номер дескриптора, адрес буфера, куда необходимо поместить прочитанную информацию и максимальный размер этого буфера (т.е. может быть прочитано не более этого значения). Эта функция возвращает число прочитанных байтов или –1 в случае ошибки.

Для передачи данных устройству (т.е. записи данных в специальный файл) используется вызов *write*. Параметры его аналогичны параметрам вызова *read*, и возвращает он также число переданных байтов.

Описание функции *read* и *write*

```
#include <unistd.h>

ssize_t read (int fd, void *buf, size_t count);
ssize_t write (int fd, void *buf, size_t count);
```

В качестве примера работы этих функций рассмотрим программу, считывающую последовательность данных с одного терминала и записывающую её на другой.

Обратите внимание (!!!), чтобы проверить, как работает эта программа, её необходимо запускать на одном из текстовых терминалов системы Linux. Для того чтобы переключиться из графического терминала в текстовый, необходимо нажать комбинацию клавиш

CTRL+ALT+F1. Чтобы вернуться в графический терминал, нужно нажать комбинацию клавиш ALT+F7. В ОС Linux имеется возможность работы с несколькими текстовыми и графическими терминалами (что и демонстрирует программа). Чтобы переключаться между текстовыми терминалами, используются комбинации клавиш ALT+F1, ALT+F2 и т.д. Соответственно ALT+F1 – переключает на первый терминал, ALT+F2 – на второй и т.п. Для демонстрации работы программы её необходимо запустить на любом текстовом терминале, а на втором текстовом терминале зайти под своим учётным именем (так как программа попытается вывести информацию на 2-й терминал, а сделать она это сможет только, если 2-й терминал будет «принадлежать» Вам).

Листинг 2. Взаимодействия с терминалами

```
1)  #include <stdio.h>
2)  #include <sys/types.h>
3)  #include <sys/stat.h>
4)  #include <fcntl.h>
5)
6)  int main (void){
7)      int fd, read_chars;
8)      char buf[200];
9)
10)    /* Открываем файл для терминала 2 на запись */
11)     fd = open (“/dev/tty2”, O_WRONLY);
12)     if (fd == -1){
13)       fprintf (stderr, “Ошибка открытия терминала.\n”);
14)       return (1);
15)     }
16)
17)    /*  Читаем  с  клавиатуры  последовательность  символов  и
           выводим их на терминал 2 */
18)     if ((read_chars = read (0, buf, 199)) > 0){
19)        write (fd, buf, read_chars);
20)     } else {
21)        write (fd, “Ошибка”, 6);
22)     }
23)     close (fd);
24)     return (0);
25) }
```

4.5.4. Функции ioctl, tcgetattr, tcsetattr

Управление терминалом производится либо управляющими воздействиями, либо установкой значений атрибутов терминала. Первый способ осуществляется с использованием вызова *ioctl* (Input Output ConTroL), второй – вызовами *tcsetattr* и *tcgetattr*.

Описание функции *ioctl*

```
#include <sys/ioctl.h>

int ioctl (int fd, int request, ...);
```

Вызов *ioctl* в качестве входных значений принимает номер дескриптора открытого файла терминала, которым надо управлять, номер требуемой операции и дополнительные параметры, необходимые для выполнения этой опера-

ции. Вызов *ioctl* является универсальным и не зависит от типа устройства (т.е. он применяется для управления не только терминалами). Поэтому сама функция *ioctl* описана в заголовочном файле *sys/ioctl.h*, а номера команд, которые она может выполнять над устройствами, – в других заголовочных файлах. Функции взаимодействия с терминалами описаны в заголовочном файле *termios.h*. Примером управляющего воздействия является определение размера экрана терминала. Результат работы *ioctl* помещается в структуру *winsize*, которая имеет вид:

Описание структуры *winsize*

```
1)  struct winsize {
2)     unsigned short ws_row;
3)     unsigned short ws_col;
4)     unsigned short ws_xpixel;
5)     unsigned short ws_ypixel;
6)  }
```

Листинг 3. Определение размера экрана терминала

```
1)  #include <stdio.h>
2)  #include <termios.h>
3)  #include <sys/ioctl.h>
4)
5)  int main (void){
6)      struct winsize ws;
7)
8)      if (!ioctl(1, TIOCGWINSZ, &ws)){
9)        printf ("Получен размер экрана.\n");
10)       printf ("Число строк – %d\nЧисло столбцов – %d\n",
11)               ws.ws_row, ws.ws_col);
12)     } else {
13)       fprintf (stderr, "Ошибка получения размера экрана.\n");
14)     }
15)     return (0);
16) }
```

Параметры работы терминалов описываются структурой *termios*, состоящей из четырёх регистров флагов, описывающих работу драйвера терминала при обработке входной информации (от клавиатуры), при выводе информации на экран, при передаче информации в ЭВМ, дополнительных режимов, а также массива специальных управляющих символов:

Описание структуры *termios*

```
1)  struct termios{
2)     tcflag_t c_iflag;
3)     tcflag_t c_oflag;
4)     tcflag_t c_lflag;
5)     tcflag_t c_cflag;
6)     tcflag_t c_cc[NCCS];
7)  };
```

В зависимости от типа используемого терминала значение каждого из регистров флагов может изменяться. Здесь мы рассмотрим только наиболее важ-

ные из режимов работы текстового терминала ОС Linux. Более подробную информацию о настройке терминалов можно получить в электронном справочнике man по ключевому слову tty.

Регистр *c_lflag* описывает, как будет вести себя терминал при обработке и передаче информации в ЭВМ. Примером таких действий являются:

- определение режима работы (канонический или неканонический). Флаг называется ICANON;
- будут ли сразу отображаться на экране терминала вводимые с клавиатуры символы. Флаг называется ECHO;
- будут ли обрабатываться управляющие символы, например прерывание работы программы (CTRL+C) или приостановка работы программы (CTRL+Z). Флаг называется ISIG.

Если установлен флаг ICANON, то включается канонический режим работы терминала. Как уже было сказано выше, это позволяет использовать символы редактирования строки в процессе построчного ввода. Если флаг ICANON не установлен, то терминал находится в режиме прямого доступа (англ. raw mode). Вызовы *read* будут при этом получать данные непосредственно из очереди ввода. Другими словами, основной единицей ввода будет одиночный символ, а не логическая строка. Программа при этом может считывать данные по одному символу или блоками фиксированного размера.

Если установлен флаг ISIG, то разрешается обработка клавиш прерывания (intr) и аварийного завершения (quit). Обычно это позволяет пользователю завершить выполнение программы. Если флаг ISIG не установлен, то проверка не выполняется, и символы intr и quit передаются программе без изменений. Значения управляющих символов задаются в массиве *c_cc*.

Если установлен флаг ECHO, то символы будут отображаться на экране по мере их набора. Сброс этого флага полезен для процедур проверки паролей и программ, которые используют клавиатуру для особых функций, например, для перемещения курсора или команд экранного редактора.

Массив *c_cc* содержит перечень символов, которые будут интерпретироваться как управляющие. Примером таких символов может служить символ с кодом 26 (комбинация клавиш CTRL+D), который интерпретируется как завершение ввода информации в каноническом режиме и т.д. Подробнее о назначении элементов массива *c_cc* можно прочитать в электронном справочнике man по ключевому слову tcgetattr.

Кроме этого, массив *c_cc* содержит ещё два элемента, описывающие поведение драйвера терминала при получении запроса на чтение данных в неканоническом режиме. А именно: какое количество символов должно быть в очереди, чтобы вызов *read* завершился (элемент, обозначаемый **VMIN**), и сколько времени (в десятых долях секунды) ждать появления хотя бы одного символа в очереди (параметр **VTIME**). Значения этих элементов определяются только для неканонического режима. В каноническом режиме они равны соответственно размеру буфера для строки и 0 (т.е. ожидать бесконечно долго).

Существуют четыре возможных комбинации параметров VMIN и VTIME:

- оба параметра VMIN и VTIME равны нулю. При этом возврат из вызова *read* обычно происходит немедленно. Если в очереди ввода терминала присутствуют символы (напомним, что попытка ввода может быть осуществлена в любой момент времени), то они будут помещены в буфер;
- параметр VMIN больше нуля, а параметр VTIME равен нулю. В этом случае *read* завершится только после того, как будут считаны VMIN символов. Это происходит далее в том случае, если вызов *read* запрашивал меньше, чем VMIN символов (т.е. ожидаться будут VMIN символов, а в буфер помещено требуемое число символов из полученных);
- параметр VMIN равен нулю, а параметр VTIME больше нуля. В этом случае вызов *read* завершится по приходе первого же символа или по истечении времени VTIME;
- оба параметра VMIN и VTIME больше нуля. В этом случае таймер запускается после получения первого символа, а не при входе в вызов *read*. Если VMIN символов будут получены до истечения заданного интервала времени, то происходит возврат из вызова *read*. Если таймер срабатывает раньше, то в программу пользователя возвращаются только символы, находящиеся при этом в очереди ввода.

Чтобы получить текущие настройки терминала используется вызов *tcgetattr*, который в качестве параметров получает номер дескриптора файла и адрес памяти, куда поместить структуру, описывающую режимы работы терминала. Результатом вызова будет либо 0, если параметры получены успешно, либо –1, если возникла какая-то ошибка.

Описание функций *tcgetattr* и *tcsetattr*

```
#include <termios.h>
#include <unistd.h>

int tcgetattr (int fd, struct termios *tsaved);
int tcsetattr (int fd, int actions, const struct termios *tnew);
```

Для установки новых параметров драйвера терминала используется вызов *tcsetattr*, который в качестве параметров принимает номер дескриптора, новые значения флагов и правила их замены. Правила могут быть следующими:

- **TCSANOW**. Немедленное выполнение изменений, что может вызвать проблемы, если в момент изменения флагов драйвер терминала выполняет вывод на терминал.
- **TCSADRAIN**. Выполняет ту же функцию, что и TCSANOW, но перед установкой новых параметров ждёт опустошения очереди вывода.
- **TCSAFLUSH**. Аналогично TCSADRAIN ждёт, пока очередь вывода не опустеет, а затем также очищает и очередь ввода перед установкой для параметров дисциплины линии связи значений, заданных в структуре *tnew*.

Контрольные вопросы

1. Для чего используется клавиатура? Что такое скан-код? Расскажите, как ЭВМ узнаёт, какая клавиша была нажата.
2. Что представляет собой код клавиши? Какие типы клавиш Вы знаете?
3. Для чего предназначен монитор? На какие типы можно разделить мониторы по физическому принципу получения изображения?
4. Расскажите об устройстве электронно-лучевых мониторов. Что такое строчная и кадровая частота развёртки?
5. Расскажите об устройстве жидкокристаллического монитора.
6. Для чего используется видеоадаптер? Расскажите о структуре и составе видеоадаптера.
7. Что такое терминал? Приведите примеры терминалов. Расскажите о принципе работы терминала.
8. Какие режимы ввода информации через терминал существуют?
9. Как может управлять терминалом программа пользователя?
10. Что такое Escape-последовательность? Как получить список доступных команд для терминала?
11. Как перейти от основной таблицы кодировок символов в терминале к дополнительной?
12. Какие устройства относятся к блочным, а какие – к символьным?
13. Какие функции используются для прямого доступа к файлам?
14. Для чего используются функции *isatty, ttyname*?
15. Какие функции используются для управления терминалом?
16. В какой структуре описаны параметры работы терминалов? Объясните назначение полей этой структуры.
17. Для чего используются функции *tcgetattr* и *tcsetattr*?

ГЛАВА 5. ПОДСИСТЕМА ПРЕРЫВАНИЙ ЭВМ

Основной проблемой, возникшей в системах, основанных на принципе общей шины, стало простаивание процессора при взаимодействии с медленно работающими устройствами. Чтобы повысить эффективность использования процессора, реализовали механизм его переключения на выполнение другой программы. Чтобы сообщить процессору, что медленное устройство снова готово взаимодействовать с ним, используется механизм прерываний.

Прерывание – это происходящее в ЭВМ событие, при котором процессор временно приостанавливает выполнение одной (текущей) программы и переключается на выполнение другой программы, необходимой для обработки этого события. После окончания выполнения обработчика события, вызвавшего прерывание, процессор возобновляет выполнение приостановленной программы. Такой подход позволяет устройствам, входящим в состав ЭВМ, функционировать независимо от процессора и сообщать последнему о своей готовности взаимодействовать с ним. Кроме этого, использование прерываний позволяет реагировать на особые состояния, возникающие при работе самого процессора (т.е. при выполнении им программ).

5.1. Механизм обработки прерываний

Механизм прерываний реализуется аппаратурно-программными средствами. Для организации аппаратурной системы прерываний используется контроллер прерываний, который подключается к соответствующему входу микропроцессора. К нему в свою очередь подключаются внешние устройства (рис. 27). Обычно контроллер может обрабатывать запросы от 8-ми источников, что на сегодняшний день является явно недостаточным. Поэтому используют каскадное подключение нескольких контроллеров, увеличивая тем самым число обслуживаемых внешних устройств. Чаще всего используют каскадное соединение только двух микросхем, обеспечивая 15 линий для генерации прерываний (одна используется для подключения ведомого контроллера). Программные прерывания реализуются самим микропроцессором.

Обработка прерываний (как аппаратурных, так и программных) микропроцессором производится как минимум в четыре этапа:

- прекращение выполнения текущей программы;
- определение источника прерывания;
- выполнение обработчика прерывания;
- возврат к прерванной программе.

Первый этап должен обеспечить временное прекращение работы текущей программы таким образом, чтобы потом возможно было продолжить её выполнение. Любая программа, исполняемая процессором, располагается в своей области оперативной памяти и никак не затрагивает память других программ (если такие имеются). Разделяемым ресурсом (т.е. одновременно используемым) является сам процессор. Поэтому сохранить необходимо как минимум его состояние (точнее состояние его внутренних регистров).

На втором этапе процессор определяет источник прерывания и сопоставляет ему адрес программы обработчика. Для этого используется специальная таблица прерываний, располагаемая в оперативной памяти. В этой таблице каждому источнику прерывания (а точнее его номеру) однозначно сопоставляется адрес оперативной памяти, где располагается программа-обработчик.

После того как определена программа-обработчик, процессор начинает её исполнять, как и обычные программы. После её завершения автоматически загружается ранее прерванная программа, и продолжается её выполнение.

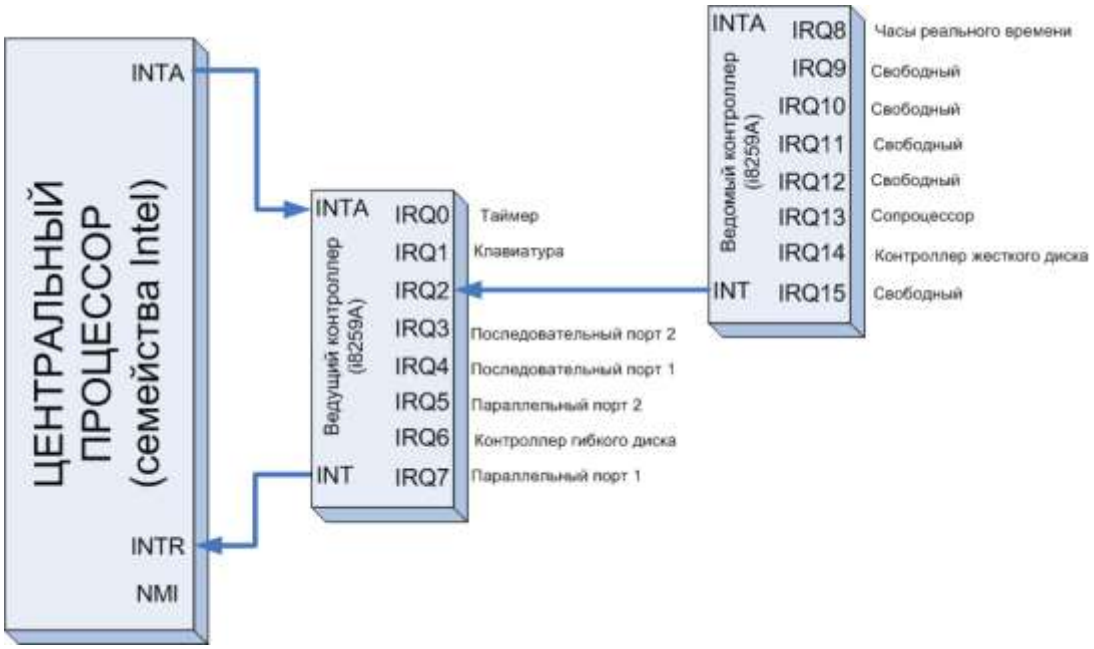


Рисунок 27. Схема каскадного подключения контроллеров прерываний в ПК на базе процессоров семейства Intel

5.2. Обработка программных прерываний в UNIX-подобных системах. Сигналы

Аналогом программных прерываний в UNIX-подобных операционных системах (например, Linux) служат сигналы. **Сигнал** – это способ взаимодействия программ, позволяющий сообщать о наступлении определённых событий, например, о появлении в очереди управляющих символов или возникновении ошибки во время работы программы (например, Segmentation Fault – выход за границы памяти).

Как и аппаратное прерывание, сигналы описываются номерами, которые описаны в заголовочном файле *signal.h*. Кроме цифрового кода, каждый сигнал имеет соответствующее символьное обозначение, например, SIGINT.

Большинство типов сигналов предназначены для использования ядром операционной системы, хотя есть несколько сигналов, которые посылаются от процесса к процессу. Полный список доступных сигналов приведён в электронном справочнике man (man 7 signal). Перечислим некоторые из них:

- **SIGABRT** – сигнал прерывания процесса (англ. process abort signal). Посылается процессу при вызове им функции *abort()*. В результате сигнала SIGABRT произойдет аварийное завершение

(англ. abnormal termination) и запись образа памяти (англ. core dump, иногда переводится как «дамп памяти»). Образ памяти процесса сохраняется в файле на диске для изучения с помощью отладчика;

- **SIGALRM** – сигнал таймера (англ. alarm clock). Посылается процессу ядром при срабатывании таймера. Каждый процесс может устанавливать не менее трёх таймеров. Первый из них измеряет прошедшее реальное время. Этот таймер устанавливается самим процессом при помощи системных вызовов *alarm* или *setitimer* (см. ниже);
- **SIGILL** – недопустимая команда процессора (англ. illegal instruction). Посылается операционной системой, если процесс пытается выполнить недопустимую машинную команду. Иногда этот сигнал может возникнуть из-за того, что программа каким-либо образом повредила свой код. В результате сигнала SIGILL происходит аварийное завершение программы;
- **SIGINT** – сигнал прерывания программы (англ. interrupt). Посылается ядром всем процессам, связанным с терминалом, когда пользователь нажимает клавишу прерывания (другими словами, в потоке ввода появляется управляющий символ, соответствующий клавише прерывания). Примером клавиши прерывания может служить комбинация CTRL+C. Это также обычный способ остановки выполняющейся программы;
- **SIGKILL** – сигнал уничтожения процесса (англ. kill). Это довольно специфический сигнал, который посылается от одного процесса к другому и приводит к немедленному прекращению работы получающего сигнал процесса. Иногда он также посылается системой (например, при завершении работы системы). Сигнал SIGKILL – один из двух сигналов, которые не могут игнорироваться или перехватываться (то есть обрабатываться при помощи определённой пользователем процедуры);
- **SIGPROF** – сигнал профилирующего таймера (англ. profiling time expired). Как было уже упомянуто для сигнала SIGALRM, любой процесс может установить не менее трёх таймеров. Второй из этих таймеров может использоваться для измерения времени выполнения процесса в пользовательском и системном режимах. Сигнал SIGPROF генерируется, когда истекает время, установленное в этом таймере, и поэтому может быть использован средством профилирования (планирования работы) программы;
- **SIGQUIT** – сигнал о выходе (англ. quit). Очень похожий на сигнал SIGINT. Этот сигнал посылается ядром, когда пользователь нажимает клавишу выхода в используемом терминале. Значение клавиши выхода по умолчанию соответствует символу F6 таблицы ASCII или CTRL+Q. В отличие от SIGINT этот сигнал приводит к аварийному завершению и сбросу образа памяти;

- **SIGSEGV** – обращение к некорректному адресу памяти (англ. invalid memory reference). Сокращение SEGV в названии сигнала означает нарушение границ сегментов памяти (англ. segmentation violation). Сигнал генерируется, если процесс пытается обратиться к неверному адресу памяти. Получение сигнала SIGSEGV приводит к аварийному завершению процесса;
- **SIGTERM** – программный сигнал завершения (англ. software termination signal). Используется для завершения процесса. Программист может использовать этот сигнал для того, чтобы дать процессу время для «наведения порядка», прежде чем посылать ему сигнал SIGKILL. Команда kill по умолчанию посылает именно этот сигнал;
- **SIGWINCH** – сигнал, генерируемый драйвером терминала при изменении размеров окна;
- **SIGUSR1** и **SIGUSR2** – пользовательские сигналы (англ. user defined signals 1 and 2). Так же как и сигнал SIGTERM, эти сигналы никогда не посылаются ядром и могут использоваться для любых целей по выбору пользователя.

При получении сигнала процесс может выполнить одно из трёх действий:

- выполнить действие по умолчанию. Обычно действие по умолчанию заключается в прекращении выполнения процесса. Для некоторых сигналов, например, для сигналов SIGUSR1 и SIGUSR2, действие по умолчанию заключается в игнорировании сигнала. Для других сигналов, например, для сигнала SIGSTOP, действие по умолчанию заключается в остановке процесса;
- игнорировать сигнал и продолжать выполнение. В больших программах неожиданно возникающие сигналы могут привести к проблемам. Например, нет смысла позволять программе останавливаться в результате случайного нажатия на клавишу прерывания, в то время как она производит обновление важной базы данных;
- выполнить определённое пользователем действие. Программист может задать собственный обработчик сигнала. Например, выполнить при выходе из программы операции по «наведению порядка» (такие как удаление рабочих файлов), что бы ни являлось причиной этого выхода.

Чтобы определить действие, которое необходимо выполнить при получении сигнала, используется системный вызов **signal:**

Описание функции *signal*

```
#include <signal.h>

typedef void (*sighandler_t) (int);
sighandler_t signal (int signum,  sighandler_t handler);
```

Вызов *signal* определяет действие программы при поступлении сигнала с номером *signum*. Действие может быть задано как: адрес пользовательской функции (в таком случае в функцию в качестве параметра передаётся номер полученного сигнала) или макросы *SIG_IGN* (для игнорирования сигнала) и *SIG_DFL* (для использования обработчика по умолчанию).

Если действие определено как пользовательская функция, то при поступлении сигнала программа будет прервана, и процессор начнёт выполнять указанную функцию. После её завершения выполнение программы, получившей сигнал, будет продолжено, и обработчик сигнала будет установлен в *SIG_DFL*.

Чтобы вызвать сигнал, используется системный вызов **raise**:

Описание функции *raise*

```
#include <signal.h>

int raise (int signum);
```

Процессу посылается сигнал, определённый параметром *signum*, и в случае успеха функция *raise* возвращает нулевое значение.

Чтобы приостановить выполнение программы до тех пор, пока не придёт хотя бы один сигнал, используется вызов **pause**:

Описание функции *pause*

```
#include <unistd.h>

int pause (void);
```

Вызов *pause* приостанавливает выполнение вызывающего процесса до получения любого сигнала. Если сигнал вызывает нормальное завершение процесса или игнорируется процессом, то в результате вызова *pause* будет просто выполнено соответствующее действие (завершение работы или игнорирование сигнала). Если же сигнал перехватывается, то после завершения соответствующего обработчика прерывания вызов *pause* вернёт значение –1 и поместит в переменную *errno* значение EINTR.

Пример программы, обрабатывающей прерывания, приведён в листинге 4.

Листинг 4. Обработка сигналов

```
1)  #include <stdio.h>
2)  #include <signal.h>
3)
4)  /* Функция-обработчик сигнала */
5)  void sighandler (int signo){
6)    printf ("Получен сигнал !!! Ура !!!\n");
7)  }
8)  /* Основная функция программы */
9)  int main (void){
10)   int x = 0;
11)
12)   /* Регистрируем обработчик сигнала */
13)   signal (SGUSR1, sghandler);
14)   do {
15)     printf ("Введите X = "); scanf ("%d", &x);
```

```
16)     if (x & 0x0A) raise (SIGUSR1);
17)   } while (x != 99);
18) }
```

5.3. Работа с таймером

Как было сказано ранее, каждая программа в UNIX-подобных операционных системах может устанавливать три таймера:

- **ITIMER_REAL** уменьшается постоянно и подаёт сигнал SIGALRM, когда значение таймера становится равным 0.
- **ITIMER_VIRTUAL** уменьшается только во время работы процесса и подаёт сигнал SIGVTALRM, когда значение таймера становится равным 0.
- **ITIMER_PROF** уменьшается во время работы процесса, и, когда система выполняет что-либо по заданию процесса. Совместно с ITIMER_VIRTUAL этот таймер обычно используется для профилирования времени работы приложения в пользовательской области и в области ядра. Когда значение таймера становится равным 0, подаётся сигнал SIGPROF.

Когда на одном из таймеров заканчивается время, процессу посылается сигнал, и таймер обычно перезапускается.

Для установки таймеров используется функция *setitimer*. Величина, в которую устанавливается таймер, определяется следующими структурами.

Описание структур *itimerval* и *timeval*

```
struct itimerval {
   struct timeval it_interval; /* следующее значение */
   struct timeval it_value; /* текущее значение */
};

struct timeval {
   long tv_sec;  /* секунды */
   long tv_usec; /* микросекунды */
};
```

Значение таймера уменьшается от величины *it_value* до нуля, после чего генерируется соответствующий сигнал, и значение таймера вновь устанавливается равным *it_interval*. Таймер, установленный в нуль (его величина *it_value* равна нулю, или время вышло, и величина *it_interval* равна нулю), останавливается. Величины *tv_sec* и *tv_usec* являются основными при установке таймера.

Если устанавливаемое время срабатывания таймера измеряется в полных секундах, то для установки таймера может использоваться системный вызов *alarm*:

Описание функций *alarm* и *setitimer*

```
#include <unistd.h>

unsigned int alarm(unsigned int secs);

#include <sys/time.h>
```

```
int setitimer(int which, const struct itimerval *value, struct itimerval *ovalue);
```

Функция *alarm* запускает таймер, который через *secs* секунд сгенерирует сигнал SIGALRM. Поэтому вызов *alarm(60)*; приводит к посылке сигнала SIGALRM через 60 секунд. Обратите внимание, что вызов *alarm* не приостанавливает выполнение процесса, как вызов *sleep*, вместо этого сразу же происходит возврат из вызова *alarm*, и продолжается нормальное выполнение программы, по крайней мере, до тех пор, пока не будет получен сигнал SIGALRM. «Выключить» таймер можно при помощи вызова *alarm* с нулевым параметром: *alarm(0)*.

Вызовы *alarm* не накапливаются. Другими словами, если вызвать *alarm* дважды, то второй вызов отменит предыдущий. Но при этом возвращаемое вызовом *alarm* значение будет равно времени, оставшемуся до срабатывания предыдущего таймера.

Пример программы, настраивающей терминал и обрабатывающей сигнал от него, приведён в листинге 5.

Листинг 5. Использование таймера

```
1)  #include <stdio.h>
2)  #include <signal.h>
3)  #include <sys/time.h>
4)
5)  void signalhandler (int signo){
6)      printf ("Сработал таймер\n");
7)  }
8)
9)  int main (void)
10) {
11)   struct itimerval nval, oval;
12)
13)   signal (SIGALRM, signalhandler);
14)
15)   nval.it_interval.tv_sec = 3;
16)   nval.it_interval.tv_usec = 500;
17)   nval.it_value.tv_sec = 1;
18)   nval.it_value.tv_usec = 0;
19)
20)   /* Запускаем таймер */
21)   setitimer (ITIMER_REAL, &nval, &oval);
22)
23)   while (1){
24)     pause();
25)   }
26)
27)   return (0);
28) }
```

Контрольные вопросы

1. Что такое прерывание? Какие механизмы обработки прерываний Вы знаете?

2. Объясните назначение сигналов. Назовите известные Вам типы сигналов, объясните, для чего они используются.
3. Какая функция используется для определения действия, которое необходимо выполнить при получении сигнала?
4. Как вызвать сигнал из программы?
5. Для чего используется функция *pause*?
6. Что такое таймер? Какие типы таймеров можно устанавливать в программе?
7. Какой сигнал подаёт таймер *ITIMER_REAL*?
8. Какие функции используются для установки таймеров?
9. Расскажите о назначении полей структур *itimerval* и *timeval*.

СПИСОК ЛИТЕРАТУРЫ

1. Архитектура средств вычислительной техники. Организация памяти ЭВМ и методы ее защиты. Методы и средства защиты информации в ЭВМ : учебное пособие. — Новосибирск : НГТУ, 2021. — 70 с. — ISBN 978-5-7782-4469-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/216275 (дата обращения: 29.03.2024). — Режим доступа: для авториз. пользователей.
2. Введение в архитектуру ЭВМ : учебное пособие / А. М. Собина, Н. Ю. Фаткуллин, В. Ф. Шамшович, Е. Н. Шварева. — Уфа : УГНТУ, 2020. — 110 с. — ISBN 978-5-7831-2151-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/245174 (дата обращения: 29.03.2024). — Режим доступа: для авториз. пользователей.
3. Журавлев, А. Е. Организация и архитектура ЭВМ. Вычислительные системы / А. Е. Журавлев. — 3-е изд., стер. — Санкт-Петербург : Лань, 2023. — 144 с. — ISBN 978-5-507-48089-0. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/341138 (дата обращения: 29.03.2024). — Режим доступа: для авториз. пользователей.

ПРИЛОЖЕНИЕ 1. ТАБЛИЦА СКАН-КОДОВ

КОДЫ КЛАВИШ ДЛЯ 101-,102- и 104-КЛАВИШНЫХ КЛАВИАТУР ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

КЛАВ.	КОД	ОТЖАТИЕ		КЛАВ.	КОД	ОТЖАТИЕ		КЛАВ.	КОД	ОТЖАТИЕ
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32		`	0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	FO,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	FO,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	FO,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	05	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	06	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	04	F0,04		KP 4	6B	F0,6B
X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	03	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	01	F0,01		]	5B	F0,5B
3	26	F0,26		F10	09	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78		'	52	F0,52
5	2E	F0,2E		F12	07	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-				

КОДЫ КЛАВИШ УПРАВЛЕНИЯ ПИТАНИЕМ

КЛАВ.	КОД	ОТЖАТИЕ
Power	E0, 37	E0, F0, 37
Sleep	E0, 3F	E0, F0, 3F
Wake	E0, 5E	E0, F0, 5E

КОДЫ КЛАВИШ РАСШИРЕНИЯ WINDOWS

КЛАВ.	КОД	ОТЖАТИЕ
Next Track	E0, 4D	E0, F0, 4D
Previous Track	E0, 15	E0, F0, 15
Stop	E0, 3B	E0, F0, 3B
Play/Pause	E0, 34	E0, F0, 34
Mute	E0, 23	E0, F0, 23
Volume Up	E0, 32	E0, F0, 32
Volume Down	E0, 21	E0, F0, 21
Media Select	E0, 50	E0, F0, 50
E-Mail	E0, 48	E0, F0, 48
Calculator	E0, 2B	E0, F0, 2B
My Computer	E0, 40	E0, F0, 40
WWW Search	E0, 10	E0, F0, 10
WWW Home	E0, 3A	E0, F0, 3A
WWW Back	E0, 38	E0, F0, 38
WWW Forward	E0, 30	E0, F0, 30
WWW Stop	E0, 28	E0, F0, 28
WWW Refresh	E0, 20	E0, F0, 20

ПРИЛОЖЕНИЕ 2. ПРАКТИЧЕСКИЕ ЗАДАНИЯ

П2.1. Общее описание разрабатываемой модели SimpleComputer

В рамках выполнения практических заданий и курсового проектирования в процессе изучения дисциплины «Архитектура ЭВМ» необходимо разработать программную модель простейшей вычислительной машины Simple Computer и набор утилит по созданию для неё программного обеспечения.

Архитектура Simple Computer

Архитектура Simple Computer представлена на рисунке 28 и включает следующие функциональные блоки: оперативную память, устройство ввода-вывода (терминал), центральный процессор и генератор тактовых импульсов.

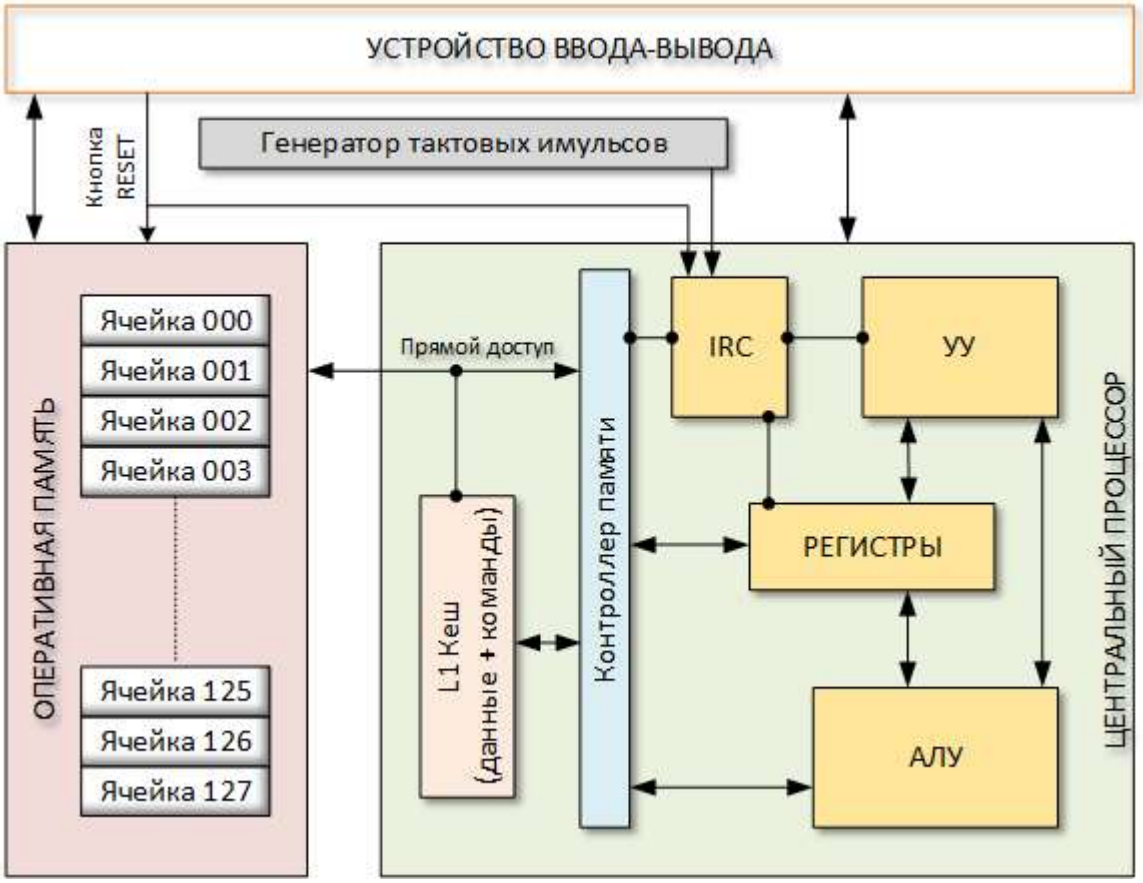


Рисунок 28 – Архитектура вычислительной машины Simple Computer

Блок «Оперативная память»

Оперативная память – это часть Simple Computer, где хранятся программа и данные. **Память состоит из ячеек, каждая из которых хранит 15 двоичных разрядов**.

Ячейка – минимальная единица, к которой можно обращаться при доступе к памяти. Все ячейки последовательно пронумерованы целыми числами. Номер ячейки является её адресом и задается 7-ми разрядным числом. Предполагаем,

что Simple Computer **оборудован памятью из 128 ячеек** (с адресами от 0_{10} до 127_{10} или 0_{16} до $7F_{16}$).

Прямой доступ к оперативной памяти имеет центральный процессор (через его контроллер памяти, о нем ниже) и устройство ввода-вывода. Оба эти устройства могут по заданному адресу оперативной памяти (номеру ячейки) поместить значение или считать его.

Блок «Центральный процессор»

Выполнение программ осуществляется центральным процессором Simple Computer. Процессор состоит из следующих функциональных блоков:

- регистры (аккумулятор, счетчик команд, регистр флагов, счетчик тактов простоя процессора);
- арифметико-логическое устройство (АЛУ);
- управляющее устройство (УУ);
- контроллер оперативной памяти;
- кэш L1 данных и команд;
- обработчик прерываний (IRC).

Регистры являются внутренней памятью процессора и используются для хранения данных или отражения состояния процессора. В состав регистров входят:

- ***аккумулятор***, используемый для временного хранения данных и результатов операций, имеет разрядность 15 бит;

- ***счетчик команд***, указывающий на адрес ячейки памяти, в которой хранится текущая выполняемая команда, разрядность регистра – 15 бит;

- ***регистр флагов***, сигнализирующий об определённых событиях. Содержит 5 разрядов, каждый из которых показывает находится ли процессор в соответствующем состоянии (произошло ли соответствующее событие): переполнение при выполнении текущей операции, ошибка деления на 0 при выполнении операции, ошибка выхода за границы памяти, игнорирование тактовых импульсов, получена неверная команда;

- ***счетчик тактов простоя процессора*** – используется для подсчета тактов, которые должен ждать процессор пока происходит обмен с оперативной памятью, разрядность регистра – 8 бит.

Арифметико-логическое устройство (англ. arithmetic and logic unit, ALU) — блок процессора, который служит для выполнения логических и арифметических преобразований над данными. В качестве данных могут использоваться значения, находящиеся в аккумуляторе, заданные в операнде команды или хранящиеся в оперативной памяти. Результат выполнения операции сохраняется в аккумуляторе или может помещаться сразу в оперативную память.

В ходе выполнения операций АЛУ устанавливает значения флагов «деление на 0» и «переполнение при выполнении текущей операции». Считается, что все операции (кроме доступа к памяти) АЛУ выполняет за один такт работы процессора.

Управляющее устройство (англ. control unit, CU) координирует работу центрального процессора. По сути, именно это устройство отвечает за выполнение программы, записанной в оперативной памяти. Логика работы управляющего устройства следующая: при поступлении сигнала от контроллера прерываний:

1. Если это сигнал Reset, то управляющее устройство задает регистрам процессора значения по умолчанию и обнуляет всю оперативную память. По умолчанию регистры процессора Simple Computer имеют следующие значения:
    a. Аккумулятор = 0;
    b. Счетчик команд = 0;
    c. Счетчик тактов простоя процессора = 0;
    d. Флаг «переполнение при выполнении операции» = 0;
    e. Флаг «ошибка деления на 0» = 0;
    f. Флаг «Ошибка выхода за границы памяти» = 0;
    g. Флаг «Получена неверная команда» = 0;
    h. Флаг «Игнорирование тактовых импульсов» = 1.
2. Если пришел сигнал от генератора тактовых импульсов, то управляющее устройство:
    a. получает текущую выполняемую команду из памяти (адрес текущей команды указан в регистре счетчика команд IC);
    b. декодирует её. Если в процессе декодирования произошла ошибка, то устанавливает флаги «указана неверная команда» и «игнорирование тактовых импульсов» и работа управляющего устройства завершается;
    c. в зависимости от полученной команды:
        i. если это арифметическая или логическая операция, то команда и операнд передается в АЛУ и ожидается обработка команды;
        ii. если это команда управления, то соответствующим образом меняется состояние регистров процессора;
        iii. если это команда взаимодействия с устройством ввода/вывода, то команда и операнд передается устройству ввода/вывода и ожидается выполнение соответствующей операции;
    d. определяется следующая выполняемая команда (меняется состояние регистра счетчика команд, если это необходимо в соответствии с текущей обрабатываемой командой). По умолчанию считается, что ячейки памяти обрабатываются последовательно. Исключение – команды управления, которые задают в своем операнде номер ячейки памяти следующей команды.

Блок «Генератор тактовых импульсов», кнопка Reset и контроллер прерываний

Генератор тактовых импульсов используется для синхронизации действий элементов Simple Computer. **Импульсы генерируются с частотой 2 импульса**

в секунду и направляются в центральный процессор, а точнее в его модуль – контроллер прерываний.

Получив очередной импульс контроллер прерываний проверяет состояние флага «Игнорирование тактовых импульсов». Если флаг установлен, то импульс игнорируется и на этом обработка прерывания останавливается.

Далее проверяется содержимое счетчика тактовых импульсов простоя процессора. Если это значение больше 0, то оно уменьшается на 1 и обработка прерывания завершается. Если значение этого счетчика равно 0, то сигнал передается блоку управляющего устройства для выполнения очередной команды.

Если приходит прерывание от нажатия кнопки Reset, то этот сигнал безусловно передается управляющему устройству для соответствующей реакции на него.

Блок «Контроллер памяти»

Контроллер памяти предназначен для организации доступа блоков процессора к оперативной памяти с целью извлечения оттуда или сохранения туда данных (команд). Оперативная память работает значительно медленнее, чем центральный процессор и **прямой доступ центрального процессора к оперативной памяти занимает 10 тактов**.

С целью оптимизации доступа в центральном процессоре используется **кэш, в котором хранится 5 строк по 10 значений**. Прежде, чем делать запрос к оперативной памяти контроллер памяти проверяет наличие запрашиваемых данных (команд) в кэше. Если они там есть, то запрос выполняется за один такт и значение сразу передается контроллером памяти блоку, который их запрашивал. Если запрашиваемой ячейки нет в кэше, то контроллер запрашивает из оперативной памяти строку (10 значений, выровненных по границе 10) и помещает её в кэш. При этом, если в кэше нет свободных срок, то самая неиспользуемая строка (доступ к ячейкам которой не было дольше всего) выгружается обратно в оперативную память (если она изменялась) и освобождается, а на её место загружается новая строка из оперативной памяти. Для упрощения ситуации считаем, что операция записи выгружаемых данных в оперативную память и чтение новых данных из оперативной памяти осуществляется одним запросом контроллера памяти.

При записи значений в оперативную память данные сохраняются в кэше. При этом если соответствующей строки в кэше не было, то она предварительно загружается в него по аналогии с загрузкой данных при запросе из памяти с учетом простоя процессора.

Если контроллеру памяти передан некорректный адрес ячейки памяти, то устанавливается флаг выхода за границы памяти и работа контроллера прекращается.

Блок «Устройство ввода-вывода»

Блок «Устройство ввода-вывода» используется для организации взаимодействия с пользователем Simple Computer и для управления самой моделью. Интерфейс устройства ввода вывода представлен на рисунке 2.

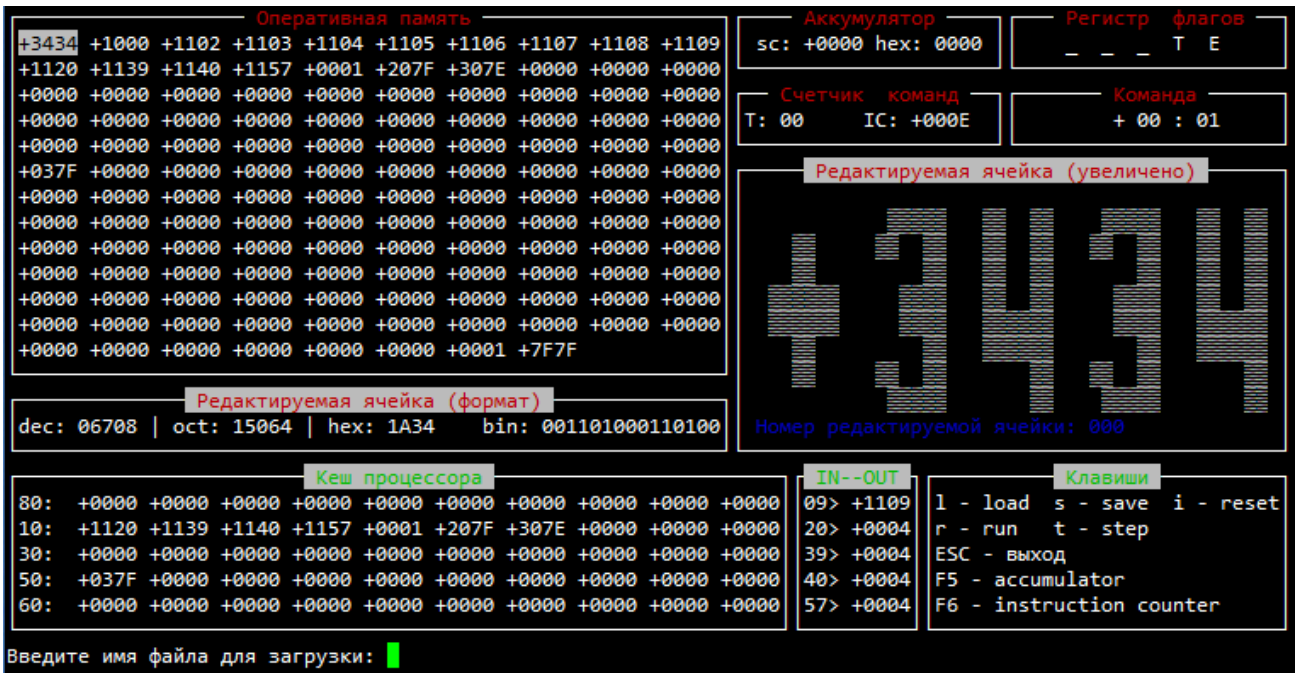


Рисунок 29 – интерфейс консоли управления моделью Simple Computer

Интерфейс разделен на несколько областей:

- “Оперативная память” – содержимое оперативной памяти Simple Computer. В этой области реализован механизм интерактивного редактирования содержимого ячеек памяти. Редактирование доступно только в том случае, если центральный процессор игнорирует тактовые импульсы. Содержимое ячейки памяти выводится и вводится в декодированном виде в соответствии с форматом команд Simple Computer. При выходе за границы блока реализован циклический переход по строкам и столбцам. Редактирование ячейки памяти осуществляется после нажатия на клавишу Enter в том же месте, где выводится значение ячейки. Пользователь вводит значение ячейки в том же виде, как оно выводится на экран (в декодированном), а после того, как пользователь ввел значение оно обратно кодируется и помещается в оперативную память. Во время ввода значение пользователю разрешено вводить только допустимые символы для соответствующего места редактирования (в первом знакоместе только значения + и -, во втором цифры 0-9 и буквы A-E, в третьем цифры 0-9 и буквы A-F и т.п.). Редактирование отменяется при нажатии на клавишу ESC. Завершается нажатием клавиши ENTER;
- «Редактируемая ячейка (формат)», «Редактируемая ячейка (увеличено)» - в этих блоках выводится содержимое текущей ячейки памяти (которая «подсвечена» курсором в блоке «Оперативная память»). В блоке «увеличено» выводится адрес редактируемой ячейки и её значение в увеличенном формате. Значение выводится также в декодированном виде в соответствии с форматом команд Simple Computer. В блоке «формат» не декодированное значение редактируемой ячейки памяти выводится в десятичной системе счисления, в восьмеричной

системе счисления, в шестнадцатеричной системе счисления и в двоичной системе счисления.

- Блок вывода состояния регистров:
  - “Аккумулятор” – выводится значение, находящееся в аккумуляторе в декодированном виде и в виде простого целого числа в шестнадцатеричной системе счисления;
  - “Регистр флагов” – выводятся значения флагов. При этом если флаг не установлен (содержит 0), то выводится символ «_», если флаг установлен, то выводится соответствующая буква: P – переполнение ячейки, “0”(ноль) – ошибка деления на 0, M - ошибка выхода за границы памяти, T – игнорирование тактовых импульсов, E - Получена неверная команда;
  - «Счетчик команд» - выводится содержимое регистров «Счетчик команд» (шестнадцатеричный формат) и «Счетчик тактов простоя процессора» (десятичный формат)
- «Кэш процессора» - выводится содержимое L1 кэша команд и данных. Каждая строка выводится в формате «адрес: строка кеша», где «адрес» — это адрес начала строки кэша, а «строка кэша» — это 10 ячеек памяти, которые сохранены в этой строке. Если строка кеша свободна, то в поле адреса выводится -1 и далее пустая строка.
- «IN—OUT» — это блок выполнения интерактивного ввода вывода, осуществляемому по указанию центрального процессора. Формат каждой строки – «адрес»«тип» «значение», где «адрес» - это адрес ячейки памяти, с которой проводится взаимодействие, «тип» - это условное отображение вида операции: “>” – вывод значения ячейки памяти, «<» - ввод значения ячейки памяти. Содержимое ячейки памяти выводится и вводится в декодированном виде в соответствии с форматом команд SimpleComputer. Ввод значения осуществляется аналогично вводу значения в блоке «Оперативная память». В блоке реализована прокрутка текста (очередная строка снизу поднимает строки вверх, самая верхняя строка удаляется);
- «Команда» - выводится поля команды (см. описание системы команд ниже), расположенной в ячейке памяти, на которую ссылается счетчик команд. Если счетчик команд указан некорректно (будет выход за границы памяти), то в поле «Команда» выводится знак “!” и дефолтное значение “+ FF : FF”;
- «Клавиши» - информационный блок, содержащий подсказку по назначению функциональных клавиш устройства ввода/вывода. Пользователь имеет возможность с помощью клавиш управления курсора выбирать ячейки оперативной памяти и задавать им значения. Нажав клавишу “F5”, пользователь может задать значение аккумулятору, “F6” – регистру «счетчик команд». Сохранить содержимое памяти (в бинарном виде) в файл или загрузить его обратно пользователь может,

нажав на клавиши «l», «s» соответственно (после нажатия внизу окна (в информационном поле) устройства ввода/вывода пользователю предлагается ввести имя файла). Запустить программу на выполнение (установить значение флага «игнорировать такты таймера» в 0) можно с помощью клавиши “r”. В процессе выполнения программы, редактирование памяти и изменение значений регистров недоступно. Чтобы выполнить только текущую команду пользователь может нажать клавишу “t”. Обнулить содержимое памяти и задать регистрам значения «по умолчанию» можно нажав на клавишу “i”. Редактирование значения аккумулятора и счетчика команд осуществляется аналогично вводу значения в блоке «Оперативная память»;

Система команд Simple Computer

Формат команды в Simple Computer следующий (см. рисунок 3): старший разряд содержит знак или признак команды (0 – команда или знак +, 1 – знак минус), разряды с 8 по 14 определяют код операции, младшие 7 разрядов содержат операнд. Коды операций, их назначение и обозначение в Simple Assembler и приведены в таблице 1.

Рисунок 30 – Формат команды центрального процессора Simple Computer

Таблица 1. Команды центрального процессора Simple Computer

Операция			**Значение**
Обозначение	**Код**		
Операции ввода/вывода			
NOP	00	0x00	Пустая операция. Не выполняется никаких действий
CPUINFO	01	0x01	Требование терминалу вывести в информационном поле вывести информацию об авторе модели Simple Computer в формате «Фамилия Имя Отчество, группа»
Операции ввода/вывода			
READ	10	0x0A	Ввод с терминала в указанную ячейку памяти с контролем переполнения
WRITE	11	0x0B	Вывод на терминал значение указанной ячейки памяти
Операции загрузки/выгрузки в аккумулятор			
LOAD	20	0x14	Загрузка в аккумулятор значения из указанного адреса памяти

STORE	21	0x15	Выгружает значение из аккумулятора по указанному адресу памяти
Арифметические операции			
ADD	30	0x1E	Выполняет сложение слова в аккумуляторе и слова из указанной ячейки памяти (результат в аккумуляторе)
SUB	31	0x1F	Вычитает из слова в аккумуляторе слово из указанной ячейки памяти результат в аккумуляторе)
DIVIDE	32	0x20	Выполняет деление слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе)
MUL	33	0x21	Вычисляет произведение слова в аккумуляторе на слово из указанной ячейки памяти (результат в аккумуляторе)
Операции передачи управления			
JUMP	40	0x28	Переход к указанному адресу памяти
JNEG	41	0x29	Переход к указанному адресу памяти, если в аккумуляторе находится отрицательное число
JZ	42	0x2A	Переход к указанному адресу памяти, если в аккумуляторе находится ноль
HALT	43	0x2B	Останов, выполняется при завершении работы программы
Пользовательские функции			
NOT	51	0x33	Двоичная инверсия слова в аккумуляторе и занесение результата в указанную ячейку памяти
AND	52	0x34	Логическая операция И между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)
OR	53	0x35	Логическая операция ИЛИ между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)
XOR	54	0x36	Логическая операция исключающее ИЛИ между содержимым аккумулятора и словом по указанному адресу (результат в аккумуляторе)
JNS	55	0x37	Переход к указанному адресу памяти, если в аккумуляторе находится положительное число
JC	56	0x38	Переход к указанному адресу памяти, если при сложении произошло переполнение
JNC	57	0x39	Переход к указанному адресу памяти, если при сложении не произошло переполнение
JP	58	0x3A	Переход к указанному адресу памяти, если результат предыдущей операции четный
JNP	59	0x3B	Переход к указанному адресу памяти, если результат предыдущей операции нечетный
CHL	60	0x3C	Логический двоичный сдвиг содержимого указанной ячейки памяти влево (результат в аккумуляторе)
SHR	61	0x3D	Логический двоичный сдвиг содержимого указанной ячейки памяти вправо (результат в аккумуляторе)

RCL	62	0x3E	Циклический двоичный сдвиг содержимого указанной ячейки памяти влево (результат в аккумуляторе)
RCR	63	0x3F	Циклический двоичный сдвиг содержимого указанной ячейки памяти вправо (результат в аккумуляторе)
NEG	64	0x40	Получение дополнительного кода содержимого указанной ячейки памяти (результат в аккумуляторе)
ADDC	65	0x41	Сложение содержимого указанной ячейки памяти с ячейкой памяти, адрес которой находится в аккумуляторе (результат в аккумуляторе)
SUBC	66	0x42	Вычитание из содержимого указанной ячейки памяти содержимого ячейки памяти, адрес которой находится в аккумуляторе (результат в аккумуляторе)
LOGLC	67	0x43	Логический двоичный сдвиг содержимого указанного участка памяти влево на количество разрядов, указанное в аккумуляторе (результат в аккумуляторе)
LOGRC	68	0x44	Логический двоичный сдвиг содержимого указанного участка памяти вправо на количество разрядов, указанное в аккумуляторе (результат в аккумуляторе)
RCCL	69	0x45	Циклический двоичный сдвиг содержимого указанного участка памяти влево на количество разрядов, указанное в аккумуляторе (результат в аккумуляторе)
RCCR	70	0x46	Циклический двоичный сдвиг содержимого указанного участка памяти вправо на количество разрядов, указанное в аккумуляторе (результат в аккумуляторе)
MOVA	71	0x47	Перемещение содержимого указанной ячейки памяти в ячейку, адрес которой указан в аккумуляторе
MOVR	72	0x48	Перемещение содержимого ячейки памяти, адрес которой содержится в аккумуляторе в указанную ячейку памяти.
MOVCA	73	0x49	Перемещение содержимого указанной ячейки памяти в ячейку памяти, адрес которой находится в ячейке памяти, на которую указывает значение аккумулятора
MOVCR	74	0x4A	Перемещение в указанный участок памяти содержимого участка памяти, адрес которого находится в участке памяти, указанном в аккумуляторе
ADDC	75	0x4B	Сложение содержимого указанной ячейки памяти с ячейкой памяти, адрес которой находится в ячейке памяти, указанной в аккумуляторе (результат в аккумуляторе)
SUBC	76	0x4C	Вычитание из содержимого указанной ячейки памяти содержимого ячейки памяти, адрес которой находится в ячейке памяти, указанной в аккумуляторе (результат в аккумуляторе)

Транслятор с языка Simple Assembler

Разработка программ для Simple Computer может осуществляться с использованием низкоуровневого языка Simple Assembler. Для того чтобы программа могла быть обработана Simple Computer необходимо реализовать транслятор, переводящий текст Simple Assembler в бинарный формат, которым может быть считан консолью управления. Пример программы на Simple Assembler:

```
00 READ  09      ; (Ввод A)
01 READ  10      ; (Ввод B)
02 LOAD  09      ; (Загрузка A в аккумулятор)
03 SUB   10      ; (Отнять B)
04 JNEG  07      ; (Переход на 07, если отрицательное)
05 WRITE 09      ; (Вывод A)
06 HALT  00      ; (Останов)
07 WRITE 10      ; (Вывод B)
08 HALT  00      ; (Останов)
09 =     +0000   ; (Переменная A)
10 =     +9999   ; (Переменная B)
```

Программа транслируется по строкам, задающим значение одной ячейки памяти. Каждая строка состоит как минимум из трех полей: адрес ячейки памяти, команда (символьное обозначение), операнд. Четвертым полем может быть указан комментарий, который обязательно должен начинаться с символа точка с запятой. Название команд представлено в таблице 1. Дополнительно используется команда =, которая явно задает значение ячейки памяти в формате вывода его на экран консоли (+XXXX).

Команда запуска транслятора должна иметь вид: sat файл.sa файл.o, где файл.sa – имя файла, в котором содержится программа на Simple Assembler, файл.o – результат трансляции.

Транслятор с языка Simple Basic

Для упрощения программирования пользователю модели Simple Computer должен быть предоставлен транслятор с высокоуровневого языка Simple Basic. Файл, содержащий программу на Simple Basic, преобразуется в файл с кодом Simple Assembler. Затем Simple Assembler-файл транслируется в бинарный формат.

В языке Simple Basic используются следующие операторы: rem, input, output, goto, if, let, end. Пример программы на Simple Basic:

```
10 REM Это комментарий
20 INPUT A
30 INPUT B
40 LET C = A – B
50 IF C < 0 GOTO 20
60 PRINT C
70 END
```

Каждая строка программы состоит из номера строки, оператора Simple Basic и параметров. Номера строк должны следовать в возрастающем порядке.

Все команды за исключением команды конца программы могут встречаться в программе многократно. Simple Basic должен оперировать с целыми выражениями, включающими операции +, -, *, и /. Приоритет операций аналогичен C. Для того чтобы изменить порядок вычисления, можно использовать скобки.

Транслятор должен распознавания только букв верхнего регистра, то есть все символы в программе на Simple Basic должны быть набраны в верхнем регистре (символ нижнего регистра приведет к ошибке). Имя переменной может состоять только из одной буквы. Simple Basic оперирует только с целыми значениями переменных, в нем отсутствует объявление переменных, а упоминание переменной автоматически вызывает её объявление и присваивает ей нулевое значение. Синтаксис языка не позволяет выполнять операций со строками.

П2.2. Требования к офомрмлению исходного кода

Общее описание проекта Simple Computer

Разрабатываемая модель Simple Computer, транслятор с языка Simple Assembler и транслятор с языка Simple Basic оформляются как один проект. Структура исходного кода проекта представлена на рисунке 31.

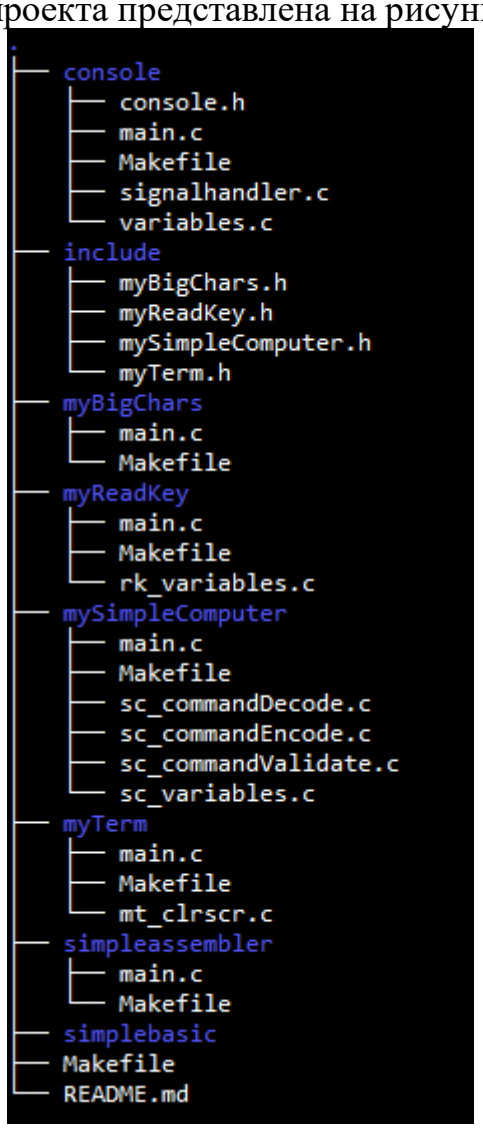


Рисунок 31 – структура проекта Simple Computer

Название каталогов и из функциональное назначение являются строго фиксированными. На рисунке приведены примеры названий файлов, перечень файлов неполный (полный перечень формируется исходя из задания на соответствующую практическую работу или курсовой проект).

Корневой каталог проекта должен содержать файл `README.md`, в котором должна содержаться информация об авторе проекта (Фамилия, Имя, Отчество, группа).

Содержимое каталогов проекта следующее:

- `console` содержит исходный код разрабатываемой консоли;
- `include` заголовочные файлы всех разрабатываемых библиотек;
- `myBigChars`, `myReadKey`, `mySimpleComputer`, `myTerm` – исходные коды соответствующих библиотек и тестов для них;
- `simpleassembler` – исходный код транслятора с языка Simple Assembler;
- `simplebasic` – исходный код транслятора с языка Simple Basic.

Используемые языки программирования и оформление исходного кода

Основная часть модели Simple Computer (содержимое каталогов `console`, `include`, `myBigChars`, `myReadKey`, `mySimpleComputer`, `myTerm`) разрабатывается с использованием языка программирования – Си (стандарт не ниже C11).

Содержимое указанных каталогов должно быть оформлено следующим образом:

- исходный код размещается в нескольких файлах;
- в одном файле исходного кода размещается ТОЛЬКО одна функция или ТОЛЬКО глобальные переменные для одной библиотеки (консоли);
- название каждого файла имеет вид <prefix>_имя содержащейся в нем функции.c, для глобальных переменных используется файл с именем вида <prefix>_variables.c, где <prefix> — это сокращенное название соответствующей библиотеки. Для исходных файлов консоли префикс в имени файла не указывается.
- заголовочные файлы библиотек содержат весь публичный интерфейс библиотеки (функции и доступные пользователю глобальные переменные библиотеки) и должны быть защищены от множественного повторного включения в исходный код. Все заголовочные файла располагаются в каталоге include. Для внутреннего описания общих элементов библиотеки допускается использование служебных заголовочных файлов, которые располагаются в каталоге соответствующей библиотеки (название файла <prefix>.h) и также защищены от повторного включения в исходный код.

- Исходный код должен быть оформлен в соответствии со стандартом оформления GNU (https://www.gnu.org/prep/standards/standards.html).

Трансляторы с языков программирования Simple Assembler и Simple Basic разрабатываются с применением любого языка программирования (на выбор автора). Оформление исходного кода указанных приложений производится в соответствие с общепринятыми требованиями к оформлению исходного кода соответствующего языка программирования.

Автоматизация сборки проекта и контроль версий исходного кода

Весь исходный код должен находиться под управлением системы контроля версий git. В репозитории не должно содержаться ничего лишнего (только исходный код проекта). Для сдачи проекта он должен быть размещен в репозитории git.csc.sibsutis.ru в **ПРИВАТНОМ** проекте с названием simplecomputer.

Основная ветка репозитория (master) должна содержать итоговую версию разрабатываемого программного обеспечения. Для выполнения практических заданий и их защиты в репозитории создаются ветки с названиями pr01, pr02, pr03, pr04, pr05, pr06. Для выполнения курсового проекта в репозитории создаются ветки kp_cache, kp_assembler, kp_basic. После защиты ветки объединяются с основной веткой (без удаления исходных веток и без склеивания коммитов).

Пример графа коммитов проекта приведен на рисунке 32.

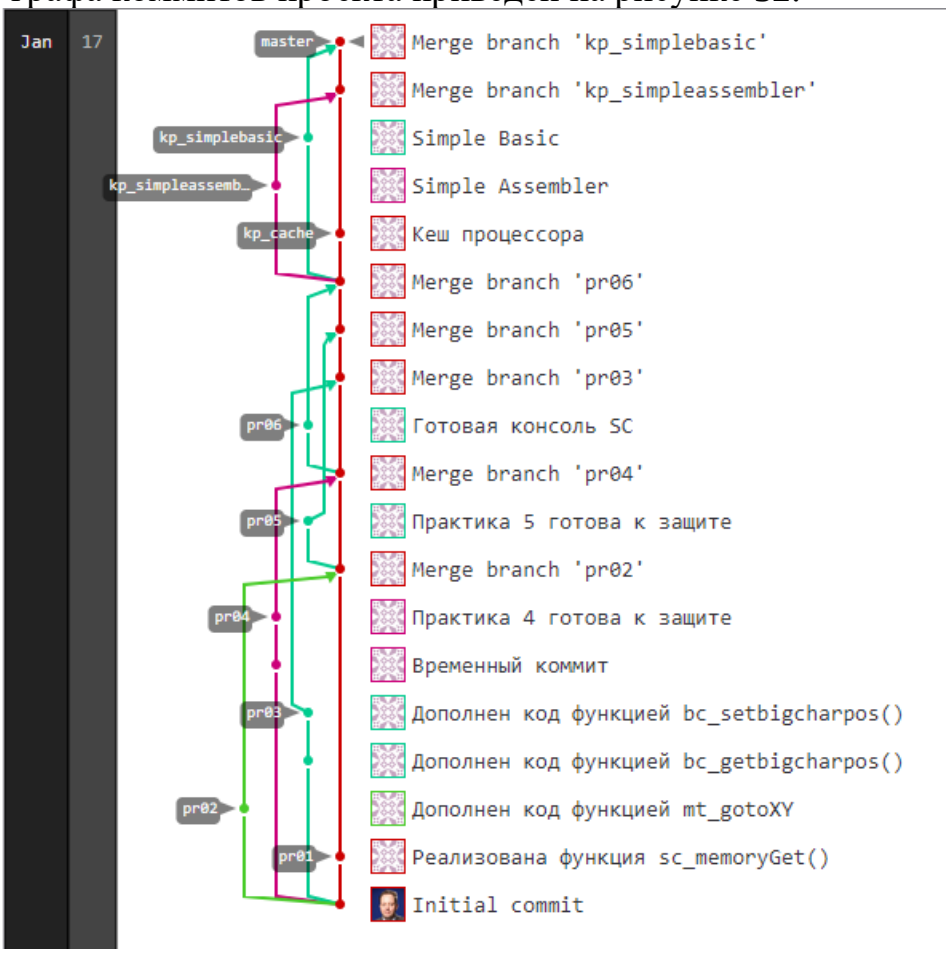


Рисунок 32 – пример графа коммитов проекта

Для сборки проекта должен быть написан Makefile (или множество связанных Makefile-ов), содержащий(ие) все необходимые действия для получения исполняемых файлов. Исполняемые файлы и вспомогательные файлы сборки располагаются в том же месте, где располагается соответствующий исходный код.

Если языки программирования, использованные для разработки трансляторов с языков Simple Assembler и Simple Basic, требуют сбору, то она так же должна быть интегрирована с Make.

Для автоматического контроля исходного кода при его размещении в репозитории git.csc.sibsutis.ru, каждый проект должен быть оснащен системой непрерывной сборки (gitlab CI/CD), в которой должно выполняться два этапа: проверка стиля исходного кода и сборка проекта. **ПРИЛОЖЕНИЕ ИЛИ КАКИЕ-ЛИБО ТЕСТЫ НЕ ДОЛЖНЫ ЗАПУСКАТЬСЯ В ПРОЦЕССЕ СБОРКИ.** Проект считается размещённым в репозитории, если после соответствующего коммита сборка прошла успешно.

Пример файла конфигурации CI/CD (`.gitlab-ci.yml`):

```
image: registry.csc.sibsutis.ru/ci/git-clang-
format:latest

stages:
  - check-format
  - build

check-format:
  stage: check-format
  script:
    - find . -type f -name *.[ch] | xargs clang-format --
style GNU -i --verbose
    - git diff --exit-code

build:
  stage: build
  script:
    - make
```

П2.3. Практическое задание № 1. Подготовка инфраструктуры для разработки проекта.

Цель работы

Понять общую концепцию разрабатываемого программного продукта. Определить список инструментов, необходимых для разработки программного продукта. Подготовить инфраструктуру, необходимую для разработки программного продукта.

Задание на практическое занятие

1. Прочитайте общее задание на практическую часть и курсовое проектирование по дисциплине «Архитектура ЭВМ».
2. Прочитайте требования к оформлению исходного кода разрабатываемого программного обеспечения.
3. В репозитории проектов git.csc.sibsutis.ru создайте проект для вашей дальнейшей работы. Склонируйте получившийся проект к себе на рабочее место.
4. Создайте в проекте структуру каталогов, соответствующую общему заданию на разработку программного продукта. Заполните файл README.md
5. В каталоге console создайте файл test.c и в нем напишите простейшую программу Hello World.
6. Разработайте Makefile (множество Makefile), необходимых для сборки приложения test. При сборке обязательно должны использоваться промежуточные объектные файлы.
7. Подготовьте описание для автоматической сборки проекта.
8. Создайте минимум 1 коммит в нужной ветке. Отправьте этот коммит в репозиторий, созданный в п.3. Убедитесь, что все этапы сборки проекта прошли удачно.
9. Подготовьте отчет о выполнении практического задания.

Контрольные вопросы

1. Какой проект будет разрабатываться в ходе выполнения практических заданий и курсового проектирования по дисциплине «Архитектура ЭВМ»?
2. Какой язык программирования используется для разработки основной части проекта? Какой язык будете использовать для разработки трансляторов?
3. Какова структура каталогов разрабатываемого проекта? Что в каком каталоге будет содержаться?
4. Что такое git? Как им пользоваться? Продемонстрируйте основные этапы использования git для сохранения версий разрабатываемого программного продукта?
5. Что такое репозиторий git.csc.sibsutis.ru? Чем он отличается от git?
6. Как настраивается автоматическая сбора программного обеспечения в git.csc.sibsutis.ru?
7. Что такое make? Зачем нужен Makefile? Какова структура Makefile?
8. Как происходит сборка программного обеспечения с помощью make? Умеет ли make собирать проект так, чтобы не делать шаги сборки, которые ранее были сделаны и не изменились?

П2.4. Практическое задание № 2. Разработка библиотеки mySimpleComputer. Оперативная память, регистр флагов, декодирование операций.

Цель работы

Изучить принципы работы оперативной памяти. Познакомиться с разрядными операциями языка Си. Разработать библиотеку mySimpleComputer, включающую функции по декодированию команд, управлению регистрами и взаимодействию с оперативной памятью.

Задание на лабораторную работу

10. Прочитайте главу 2 учебного пособия. Изучите принципы работы разрядных операций в языке Си: как можно изменить значение указанного разряда целой переменной или получить его значение. Вспомните, как сохранять информацию в файл и считывать её оттуда в бинарном виде.
11. Разработайте часть библиотеки mySimpleComputer, моделирующую работу оперативной памяти и прямого доступа к ней (используется для контроллера оперативной памяти центрального процессора и устройства ввода-вывода):
    a. В качестве «оперативной памяти» используется массив целых чисел, определенный статически в рамках библиотеки (глобальная переменная внутри библиотеки). Размер массива равен 128 элементам. Тип элементов массива – целые числа (`int`). Массив, содержащий «оперативную память» доступен только функциям библиотеки (в пользовательском API он отсутствует);
    b. `int sc_memoryInit (void)` – инициализирует оперативную память Simple Computer, задавая всем её ячейкам нулевые значения.
    c. `int sc_memorySet (int address, int value)` – задает значение указанной ячейки памяти как value. Если адрес выходит за допустимые границы или value не соответствует допустимому диапазону значений, то функция возвращает `-1`, иначе завершается корректно и возвращает `0`;
    d. `int sc_memoryGet (int address, int * value)` – возвращает значение указанной ячейки памяти в value. Если адрес выходит за допустимые границы или передан неверный указатель на value, то функция завершается со статусом `-1`. В случае успешного выполнения функции она завершается со статусом 0.
    e. `int sc_memorySave (char * filename)` – сохраняет содержимое памяти в файл в бинарном виде (используя функцию write или fwrite). Если передан неверный указатель на имя файла или произошла какая-либо ошибка записи данных в файл, то функция завершается со статусом -1. В случае успеха функция завершается со статусом 0;

f. `int sc_memoryLoad (char * filename)` – загружает из указанного файла содержимое оперативной памяти (используя функцию read или fread). Если передан неверный указатель на имя файла или произошла какая-либо ошибка чтения данных из файла, то функция завершается со статусом -1, при этом содержимое «оперативной памяти» никак не изменяется (т.е. оно не должно портиться). В случае успеха функция завершается со статусом 0;

12. Разработайте часть библиотеки mySimpleComputer, моделирующую регистры Simple Computer:
    a. регистры «Аккумулятор», «Счетчик команд», «Регистр флагов» - целые переменные (глобальные для библиотеки и недоступные напрямую пользователю);
    b. `int sc_regInit (void)` – инициализирует регистр флагов значениями по умолчанию;
    c. `int sc_regSet (int register, int value)` – устанавливает значение указанного регистра флагов. Для номеров регистров флагов должны использоваться маски, задаваемые макросами (#define). Если указан недопустимый регистр, то функция завершается со статусом -1 и значение флага не меняется. Иначе статус завершения – 0. Флаг меняется в соответствии с правилами определения логического значения целой переменной, принятых в языке Си;
    d. `int sc_regGet (int register, int * value)` – возвращает значение указанного флага. Если указан недопустимый регистр и передан неверный указатель на значение, то функция завершается со статусом -1. Иначе статус завершения – 0.
    e. `int sc_accumulatorInit (void)` – инициализирует аккумулятор значением по умолчанию;
    f. `int sc_accumulatorSet (int value)` – устанавливает значение аккумулятора. Если указано недопустимое значение, то функция завершается со статусом -1 и значение аккумулятора не меняется. Иначе статус завершения – 0;
    g. `int sc_accumulatorGet (int * value)` – возвращает значение аккумулятора. Если передан неверный указатель на значение, то функция завершается со статусом -1. Иначе статус завершения – 0.
    h. `int sc_icounterInit (void)` – инициализирует счетчик команд;
    i. `int sc_icounterSet (int value)` – устанавливает значение счетчика команд. Если указано недопустимое значение, то функция завершается со статусом -1 и значение счетчика не меняется. Иначе статус завершения – 0;
    j. `int sc_icounterGet (int * value)` – возвращает значение счетчика команд. Если передан неверный указатель на значение, то функция завершается со статусом -1. Иначе статус завершения – 0.

13. Разработайте часть библиотеки mySimpleComputer, моделирующую часть устройства управления, отвечающую за кодирование и декодирование команды:
    a. `int sc_commandEncode (int sign, int command, int operand, int * value)` – кодирует значение ячейки в соответствии с форматом команды Simple Computer и с использованием в качестве значений полей полученные знак, номер команды и операнд и помещает результат в `value`. Если указаны недопустимые значения для знака, команды или операнда, то функция завершается со статусом -1 и значение value не изменяется. В противном случае – статус завершения 0. Для знака, операнда и команды допустимыми являются все значения, которые соответствуют формату команды Simple Computer;
    b. `int sc_commandDecode (int value, int * sign, int * command, int * operand)` – декодирует значение ячейки памяти как команду Simple Computer. Если декодирование невозможно, то функция завершается со статусом -1 и выходные параметры не меняю своего значения. Иначе статус завершения = 0;
    c. `int sc_commandValidate (int command)` – проверяет значение поля «команда» на корректность. Если значение некорректное, то возвращается -1. Иначе возвращается 0;
14. Оформите разработанные функции как статическую библиотеку. Подготовьте заголовочный файл для неё. Доработайте систему сборки приложения таким образом, чтобы статическая библиотека mySimpleComputer собиралась при изменении любого из файлов с исходным кодом. Собранная библиотека должна располагаться в каталоге mySimpleComputer.
15. Разработайте часть устройства ввода-вывода (все данные, выводимые функциями в рамках данного практического задания, просто выводятся на экран, без формирования интерфейса консоли):
    a. `void printCell (int address)` – выводит на экран содержимое ячейки оперативной памяти по указанному адресу. Формат вывода должен соответствовать заданию (ячейка выводится в декодированном виде);
    b. `void printFlags (void)` – выводит значения флагов. Формат должен соответствовать заданию (выводятся либо _, либо буквы в заданной последовательности);
    c. `void printDecodedCommand (int value)` – выводит переданное значение в десятичной системе счисления, в восьмеричной системе счисления, в шестнадцатиричной системе счисления и в двоичной системе счисления.
    d. `void printAccumulator (void)` – выводит значение аккумулятора;
    e. `void printCounters (void)` – выводит значение счетчика команд.

16. Разработайте тестовую программу pr01, которая должна использовать все созданные выше функции и библиотеку mySimpleComputer и выполнять следующие действия:
    a. инициализировать оперативную память, аккумулятор, счетчик команд и регистр флагов;
    b. установить произвольному количеству произвольных ячеек оперативной памяти произвольные значения. Вывести содержимое оперативной памяти (в декодированном формате по 10 ячеек в строке через пробел);
    c. попробовать задать какой-нибудь ячейке оперативной памяти недопустимое значение и вывести статус завершения соответствующей функции;
    d. установить произвольные значения флагов и вывести содержимое регистра флагов.
    e. попробовать установить некорректное значение флага. Вывести статус завершения функции.
    f. установить значение аккумулятора и вывести его на экран.
    g. попробовать задать аккумулятору недопустимое значение и вывести статус завершения функции;
    h. установить значение счетчика команд и вывести его на экран.
    i. попробовать задать счетчику команд недопустимое значение и вывести статус завершения функции;
    j. декодировать значение произвольной ячейки памяти и значение аккумулятора;
    k. закодировать команду (любую допустимую из системы команд) и вывести полученное значение в разных системах счисления.
17. Итоговый исполняемый файл должен называться pr01 и располагаться в каталоге console.
18. Доработайте Makefile таким образом, чтобы автоматизированная сборка смогла собрать библиотеку и исполняемый файл pr01. Исполняемый файл должен собираться при изменении библиотеки или любого из исходных файлов каталога console. Также необходимо реализовать в Makefile искусственные цели по очистке каталога проекта от временных файлов (цель clean).

Контрольные вопросы

9. Что такое вентиль? Какие значения он может принимать?
10. Сколько вентилей необходимо, чтобы получить логические функции НЕ, ИЛИ-НЕ, И-НЕ, И, ИЛИ?
11. Что такое таблица истинности? Булева функция? Как они связаны между собой?
12. Как получить алгебраическую булеву функцию из таблицы истинности? И наоборот?
13. Каким образом можно синтезировать логическую схему по таблице истинности? По алгебраической формуле?

14. Что такое система счисления? Чем отличается позиционная система счисления от непозиционной?
15. Как получить качественный эквивалент числа в непозиционной системе счисления? В позиционной?
16. Как перевести числа из двоичной системы счисления в десятичную? Восьмеричную? Шестнадцатеричную? И наоборот?
17. Что такое двоично-десятичное число?
18. Как в ЭВМ представляются отрицательные числа и числа с плавающей запятой?
19. Что такое дополнительный код? Зачем он используется?
20. Как перевести десятичное число с плавающей запятой в двоичное?
21. Какие базовые типы данных используются для хранения переменных в языке СИ?
22. Что такое флаг? Зачем он используется? Каким образом можно манипулировать флагами? Что такое маска?

П2.5. Практическое задание № 3. Консоль управления моделью Simple Computer. Текстовая часть.

Цель работы

Изучить принципы работы терминалов ЭВМ в текстовом режиме. Понять, каким образом кодируется текстовая информация и как с помощью неё можно управлять работой терминалов. Разработать библиотеку функций myTerm, включающую базовые функции по управлению текстовым терминалом. Доработать устройство ввода-вывода Simple Computer (вывести на экран текстовую часть консоли).

Задание на лабораторную работу

1. Прочитайте главу 4. Изучите страницу `man` для команды `infocmp`, базы `terminfo`, функции `ioctl`.
2. Откройте текстовый терминал и запустите оболочку `bash` (оболочка запускается автоматически). Используя команду `infocmp`, определите (и перепишите их себе) escape-последовательности для терминала, выполняющие следующие действия:
   - очистка экрана и перемещение курсора в левый верхний угол (clear_screen);
   - перемещение курсора в заданную позицию экрана (cursor_address);
   - задание цвета последующих выводимых символов (set_a_background);
   - задание цвета фона последующих выводимых символов (set_a_foreground);
   - скрытие и восстановление курсора (cursor_invisible, cursor_visible);
   - очистка текущей строки (delete_line).

3. Используя оболочку bash, команду echo –e и скрипт[2], проверьте работу полученных последовательностей. Символ escape задается как \033 или \E. Например – echo -e "\033[m". Для проверки сформируйте последовательность escape-команд, выполняющую следующие действия:
   - очищает экран;
   - выводит в пятой строке, начиная с 10 символа Ваше имя красными буквами на черном фоне;
   - в шестой строке, начиная с 8 символа Вашу группу зеленым цветом на белом фоне;
   - перемещает курсор в 10 строку, 1 символ и возвращает настройки цвета в значения «по умолчанию».

   Скрипт должен располагаться в корневом каталоге проекта и называться test_terminal.sh.
4. Разработать функции библиотеки myTerm:
   - `int mt_clrscr (void)`- производит очистку и перемещение курсора в левый верхний угол экрана;
   - `int mt_gotoXY (int, int)` - перемещает курсор в указанную позицию. Первый параметр номер строки, второй - номер столбца;
   - `int mt_getscreensize (int * rows, int * cols)` - определяет размер экрана терминала (количество строк и столбцов). Если определение невозможно, то статус возврата = -1, иначе 0;
   - `int mt_setfgcolor (enum colors)` - устанавливает цвет последующих выводимых символов. В качестве параметра передаётся константа из созданного Вами перечислимого типа `colors`, описывающего цвета терминала;
   - `int mt_setbgcolor (enum colors)` - устанавливает цвет фона последующих выводимых символов. В качестве параметра передаётся константа из созданного Вами перечислимого типа `colors`, описывающего цвета терминала.
   - `int mt_setdefaultcolor (void)` – устанавливает цвета символов и фона в значения по умолчанию;
   - `int mt_setcursorvisible (int value)` – скрывает или показывает курсор;
   - `int mt_delline (void)` – очищает текущую строку

   Все функции возвращают 0 в случае успешного выполнения и -1 в случае ошибки. В качестве терминала используется стандартный поток вывода. Вывод управляющих последовательностей должен быть реализован с использованием низкоуровневого вывода (функция write).

[2] Скрипт – это текстовый файл, содержащий команды оболочки. Запускается на выполнение командой bash имя_файла.

5. Оформите разработанные функции как статическую библиотеку myTerm. Подготовьте заголовочный файл для неё. Доработайте систему сборки приложения таким образом, чтобы статическая библиотека myTerm собиралась при изменении любого из файлов с исходным кодом. Собранная библиотека должна располагаться в каталоге myTerm.
6. Доработайте устройство ввода-вывода:
    - Измените функцию `printCell -> void printCell (int address, enum colors fg, enum colors bg)`. Теперь эта функция выводит значение ячейки памяти с учетом заданных цветов символов и фона. Кроме этого местоположение выводимого значения рассчитывается исходя из заданного адреса ячейки памяти и расположения блока «Оперативная память»;
    - Измените функцию `void printFlags (void)`. Теперь она выводит значения флагов в нужном месте экрана (блок «Регистр флагов»);
    - Измените функцию `void printDecodedCommand (int value)`. Теперь она выводит значения в нужно месте экрана (блок «Редактируемая ячейка (формат)»).
    - Измените функцию `void printAccumulator (void)`. Теперь она выводит значение в нужном месте экрана (блок «Аккумулятор»), значения выводятся в декодированном виде и в шестнадцатеричной системе счисления;
    - Измените функцию `void printCounters (void)`. Теперь она выводит значение счетчика команд в нужном месте экрана (блок «Счетчик команд») и в соответствующем формате;
    - Разработайте функцию `void printTerm (int address, int input)` - которая выводит очередную строку в блок “IN—OUT”, address – это адрес выводимой ячейки памяти, а input – это признак ввода значения или его вывода. Если значение должно быть выведено, то оно выводится. Если значение необходимо ввести, то выводится только адрес ячейки и признак ввода (>). Функция должна реализовать механизм «прокрутки» строк, т.к. выводить текущее значение и максимум три предыдущих;
    - Разработайте функцию `void printCommand (void)` - которая выводит результат декодирования ячейки памяти, адрес которой указан в счетчике команд. Если команда неверная, то перед результатом декодирования должен выводиться знак «!». Значение должно выводиться в соответствующем месте экрана (блок «Команда»);
7. Разработайте приложение console, которое:
    - При запуске приложения проверялось, что поток вывода соответствует терминалу. Если это не так, что приложения принудительно завершается;
    - Проверяется размер экрана терминала. Если размера не хватает для того, чтобы вывести консоль (с учетом псевдографических симво-

лов), то на экран выводится сообщение об ошибке и программа завершается;
- Экран терминала очищается;
- На экран выводятся все текстовые данные консоли (кроме рамок, блока с увеличенным значением текущей редактируемой ячейки памяти, заголовков блоков). Одна ячейка памяти в блоке «Оперативная память» должна быть выведена в инверсном режиме (это «текущая редактируемая ячейка»).
- В блок “IN—OUT” последовательно выводятся 7 значений произвольных ячеек оперативной памяти.

8. Доработайте Makefile таким образом, чтобы автоматизированная сборка дополнительно смогла собрать библиотеку myTerm и исполняемый файл concole. Исполняемый файл должен собираться при изменении библиотек (любой из используемых) или любого из исходных файлов каталога console.

Контрольные вопросы

1. Взаимодействие с устройствами в Linux. Специальные файлы устройств.
2. Функции open, close, read, write.
3. Терминалы. Типы терминалов. Эмуляция терминала. Режимы работы.
4. Управление терминалом. Команды. Низкоуровневое управление.
5. Что такое escape-последовательность?
6. Как определить escape-последовательности для терминала?

П2.6. Практическое задание № 4. Консоль управления моделью Simple Computer. Псевдографика. Шрифты. Таблицы символов. «Большие символы».

Цель работы

Изучить работу текстового терминала с псевдографическими символами. Понять, что такое шрифт и как он используется в терминалах при выводе информации. Разработать библиотеку myBigChars, реализующую функции по работе с псевдографикой и выводу «больших символов» на экран. Доработать консоль управления Simple Computer так, чтобы выводились псевдографические элементы.

Задание на лабораторную работу.

1. Прочитайте главу 4 Изучите страницу `man` для команды `infocmp`, базы `terminfo` (раздел псевдографика).
2. Используя оболочку `bash` и команду `infocmp`, определите escape-последовательности для переключения используемых терминалом кодировочных таблиц (enter_alt_charset_mode и exit_alt_charset_mode) и соответствие символов для вывода псевдографики (acs_chars).
3. Используя оболочку `bash`, команду echo –e и скрипт, проверьте работу полученных последовательностей. Символ escape задается как \033 или

\E. Например - echo -e "\033[m". Для проверки сформируйте последовательность escape-команд, выполняющую следующие действия:

- очищает экран;
- выводит псевдографическую рамку, начиная с 5 символа 10 строки, размером 8 строк на 8 столбцов;
- с помощью псевдографического символа «закрашенный прямоугольник» (ACS_CKBOARD) в рамке выводится большой символ, соответствующий последней цифре дня вашего рождения (например, день рождения 13 января 1991 года, выводится цифра 3).

Скрипт должен располагаться в корневом каталоге проекта и называться test_terminal2.sh.

4. Прочитайте информацию про таблицу символов UTF-8 (читать тут). Определите сколько байт будет использоваться для кодирования символов английского алфавита (большие маленькие буквы), цифр, русского алфавита (большие маленькие буквы).
5. Разработать следующие функции библиотеки myBigChars:
   - `int bc_strlen (char * str)` – подсчитывает количество символов в UTF-8 строке. Если декодирование какого-либо символа невозможно или передан некорректный указатель, то функция возвращает 0.
   - `int bc_printA (char * str)` - выводит строку символов с использованием дополнительной кодировочной таблицы;
   - `int bc_box(int x1, int y1, int x2, int y2, enum colors box_fg, enum colors box_bg, char *header, enum colors header_fg, enum colors header_bg)` – выводит на экран псевдографическую рамку, в которой левый верхний угол располагается в строке `x1` и столбце `y1`, а её ширина и высота равна y2 столбцов и `x2` строк. Цвет псевдографических символов и их фон указан в параметрах `box_fg`, `box_bg`. Если передан корректный указатель на `header` и полученная строка декодируется из UTF-8, то в верхней строке рамки посередине выводится строка заголовка с указанным цветом символов на указанном цвете фона;
   - `int bc_setbigcharpos (int * big, int x, int y, int value)` – манипулирует элементом шрифта и устанавливает в нем значение знакоместа "большого символа" в строке `x` и столбце `y` в значение `value`;
   - `int bc_getbigcharpos(int * big, int x, int y, int *value)` - возвращает значение позиции в "большом символе" в строке `x` и столбце `y`;
   - `int bc_printbigchar (int [2], int x, int y, enum color, enum color)` - выводит на экран "большой символ" размером восемь строк на восемь столбцов, левый верхний угол ко-

торого располагается в строке x и столбце y. Третий и четвёртый параметры определяют цвет и фон выводимых символов. "Символ" выводится исходя из значений массива целых чисел следующим образом. В первой строке выводится 8 младших бит первого числа, во второй следующие 8, в третьей и 4 следующие. В 5 строке выводятся 8 младших бит второго числа и т.д. При этом если значение бита = 0, то выводится символ "пробел", иначе - символ, закрашивающий знакоместо (ACS_CKBOARD);

- `int bc_bigcharwrite (int fd, int * big, int count)` - записывает заданное число "больших символов" в файл. Формат записи определяется пользователем. Символы записываются в бинарном виде;
- `int bc_bigcharread (int fd, int * big, int need_count, int * count)` считывает из файла заданное количество "больших символов". Третий параметр указывает адрес переменной, в которую помещается количество считанных символов или 0, в случае ошибки.

Все функции возвращают 0 в случае успешного выполнения и -1 в случае ошибки. В качестве терминала используется стандартный поток вывода.

6. Оформите разработанные функции как статическую библиотеку myBigChars. Подготовьте заголовочный файл для неё. Доработайте систему сборки приложения таким образом, чтобы статическая библиотека myBigChars собиралась при изменении любого из файлов с исходным кодом. Собранная библиотека должна располагаться в каталоге myBigChars.
7. Разработайте программу `font.c`, которая должна располагаться в каталоге `console`. Функциональное назначение программы – сгенерировать шрифт «больших символов». Считаем, что кодировочная таблица этого шрифта, следующая:

Код	Символ		Код	Символ		Код	Символ
0	0		6	6		12	C
1	1		7	7		13	D
2	2		8	8		14	E
3	3		9	9		15	F
4	4		10	A		16	+
5	5		11	B		17	1

8. Доработайте устройство ввода-вывода следующим образом:
   - При запуске приложения должен считываться файл со шрифтом больших символов. Имя файла задается как параметр командной строки, указываемый при запуске консоли. Если параметр не задан, то используется имя файла шрифта – font.bin. Если файл шрифта не найден или чтение шрифта невозможно, то приложения завершается с ошибкой и выводом на экран соответствующего сообщения;

- Доработайте интерфейс так, чтобы были выведены все рамки и заголовки. Вид итогового интерфейса должен полностью соответствовать заданию.
- Разработайте функцию void printBigCell (void) – которая выводит в нужном месте экрана увеличенное значение текущей редактируемой ячейки;

9. Доработайте Makefile таким образом, чтобы автоматизированная сборка дополнительно смогла собрать все разработанные библиотеки и исполняемые файлы. Исполняемые файлы должен собираться при изменении библиотек (любой из используемых) или любого из исходных файлов каталога console.

Контрольные вопросы

1. Что такое шрифт? Как он используется при выводе символов на экран?
2. Зачем используется кодировочная таблица символов? Какие таблицы Вы знаете?
3. Почему символы, рисующие рамку в текстовом режиме, называются «псевдографическими»?

П2.7. Практическое задание № 5. Консоль управления моделью Simple Computer. Клавиатура. Обработка нажатия клавиш. Неканонический режим работы терминала

Цель работы

Изучить устройство клавиатуры и принципы обработки нажатия клавиш в текстовом терминале. Создать «распознаватель» нажатой клавиши по формируемой последовательности символов. Разработать библиотеку myReadkey. Доработать интерфейс консоли управления Simple Computer так, чтобы можно было изменять значения ячеек памяти и регистров.

Задание на лабораторную работу

1. Прочитайте главу 4. Изучите страницу `man` для команд `infocmp` и `read`, базы `terminfo`.
2. Используя оболочку `bash` и команду `read`, определите последовательности, формируемые нажатием на буквенно-цифровые (a-z A-Z 0-9 + -), функциональные клавиши (F5, F6, ESC) и клавиши управления курсором (← ↑ → ↓ ENTER). Используя команду `infocmp`, убедитесь, что получены правильные последовательности символов, генерируемые функциональными клавишами «F5» и «F6».
3. Разработайте следующие функции библиотеки myReadkey:
   - `int rk_readkey (enum keys *)` - анализирующую последовательность символов (возвращаемых функцией `read` при чтении с терминала) и возвращающую первую клавишу, которую нажал пользователь. В качестве параметра в функцию передаётся адрес

переменной, в которую возвращается номер нажатой (`enum keys` – перечисление распознаваемых клавиш);

- `int rk_mytermsave (void)` - сохраняет текущие параметры терминала;
- `int rk_mytermrestore (void)` - восстанавливает сохранённые параметры терминала.
- `int rk_mytermregime (int regime, int vtime, int vmin, int echo, int sigint)` - переключает терминала между режимами. Для неканонического режима используются значения второго и последующего параметров.
- `int rk_readvalue (int *value, int timeout)` – обеспечивает ввод с клавиатуры значения в соответствии с форматом команд и ограничением на вводимые символы.

4. Оформите разработанные функции как статическую библиотеку myReadkey. Подготовьте заголовочный файл для неё. Доработайте систему сборки приложения таким образом, чтобы статическая библиотека myReadkey собиралась при изменении любого из файлов с исходным кодом. Собранная библиотека должна располагаться в каталоге myReadkey.
5. Доработайте устройство ввода-вывода следующим образом. После запуска программы она переходит в интерактивный режим, в котором реализовано следующее:
   - перемещение курсора по области редактирования оперативной памяти. Перемещение должно быть циклическим (переход за границу строки или столбца приводи к перемещению курсора в другой конец строки или столбца). При перемещении курсора должны изменяться значения блоков «Редактируемая ячейка (увеличено)» и «Редактируемая ячейка (формат)».
   - при нажатии на клавишу ENTER программа должна перейти в режим редактирования текущей ячейки. Редактирование производится в режиме «InPlace», т.е. в том же месте, где выводится значение ячейки.
   - При нажатии на ESC программа завершается.
   - Должны быть реализованы функционалы клавиш F5, F6. Редактирование соответствующего регистра также реализуется в режиме InPlace.
   - Должны быть реализованы функционалы клавиш l, s, i.

Контрольные вопросы

1. Режимы работы терминала. Как настроить терминал для работы в неканоническом режиме?
2. Работа с терминалом в Linux. Структура termios.

П2.8. Практическое задание № 6. Подсистема прерываний ЭВМ. Сигналы и их обработка.

Цель работы

Изучить принципы работы подсистемы прерываний ЭВМ. Понять, как обрабатываются сигналы в Linux. Реализовать обработчик прерываний в модели Simple Computer. Доработать модель Simple Computer, создав обработчик прерываний от внешних устройств «системный таймер» и «кнопка».

Задание на практическое занятие

1. Прочитайте главу 5. Изучите страницу `man` для функций `signal`, `setitimer`.
2. Для реализации модели разработайте две функции:
   a. `void CU (void)` – реализует алгоритм работы одного такта устройства управления. Ввод и вывод (интерактивное взаимодействие с пользователем) должно осуществляться в блоке “IN—OUT”;
   b. `int ALU (int command, int operand)` – реализует алгоритм работы одного такта арифметико-логического устройства.

   Функции должны реализовывать все команды Simple Computer с кодами меньше 0x50 и две функции из блока «Пользовательские функции» (коды соответствующих команд определяются преподавателем).
3. Разработайте функцию `void IRC (int signum)` – которая реализует алгоритм работы контроллера прерываний. Считаем, что от генератора импульсов будет поступать сингал SIGALRM, от Reset – сигнал SIGUSR1;
4. Доработайте консоль Simple Computer. Создайте обработчик сигналов от генератора тактовых импульсов и от кнопки Reset. При нажатии на клавишу r консоль переходит из интерактивного режима в режим работы модели Simple Computer и пока модель работает никакие интерактивные клавиши не обрабатываются. Если пользователь в интерактивном режиме нажимает клавишу s, то контроллер прерываний обрабатывает это нажатие как поступление одного сигнала от генератора тактовых импульсов без проверки состояния флага T.

По итогам выполнения практического задания должна получить полноценно работающая модель Simple Computer (за исключением блока «кэш»).

Для итоговой сдачи проекта Вам необходимо разработать небольшую программу для этой модели, которую можно интерактивно ввести в оперативную память (или считать из файла) и продемонстрировать работу модели.

Контрольные вопросы

1. Что такое прерывание? Что такое сигнал? Чем они отличаются друг от друга? Какую информацию несут в себе прерывание и сигнал?

2. Как происходит обработка сигнала в программах, работающих под управлением ОС Linux?
3. Каким образом настраивается таймер? Как программа «узнаёт» о срабатывании таймера?
4. Каким образом пользовательская программа может узнать об изменении размера окна виртуального терминала?

П2.9. Расчетно-графическое задание

В рамках курсовой работы необходимо:

- Разработать транслятор с языка Simple Basic. Итог работы транслятора – бинарный файл с образом оперативной памяти Simple Computer, который можно загрузить в модель и выполнить;
- Доработать модель Simple Computer – реализовать алгоритм работы блока «L1-кэш команд и данных» и модифицировать работу контроллера оперативной памяти и обработчика прерываний таким образом, чтобы учитывался простой процессора при прямом доступе к оперативной памяти;
- Разработать транслятор с языка Simple Basic. Итог работы транслятора – текстовый файл с программой на языке Simple Basic.

Транслятор с языка Simple Assembler

Разработка программ для Simple Computer может осуществляться с использованием низкоуровневого языка Simple Assembler. Для того чтобы программа могла быть обработана Simple Computer необходимо реализовать транслятор, переводящий текст Simple Assembler в бинарный формат, которым может быть считан консолью управления. Пример программы на Simple Assembler:

```
00 READ  09      ; (Ввод A)
01 READ  10      ; (Ввод B)
02 LOAD  09      ; (Загрузка A в аккумулятор)
03 SUB   10      ; (Отнять B)
04 JNEG  07      ; (Переход на 07, если отрицательное)
05 WRITE 09      ; (Вывод A)
06 HALT  00      ; (Останов)
07 WRITE 10      ; (Вывод B)
08 HALT  00      ; (Останов)
09 =     +0000   ; (Переменная A)
10 =     +9999   ; (Переменная B)
```

Программа транслируется по строкам, задающим значение одной ячейки памяти. Каждая строка состоит как минимум из трех полей: адрес ячейки памяти, команда (символьное обозначение), операнд. Четвертым полем может быть указан комментарий, который обязательно должен начинаться с символа точка

с запятой. Название команд представлено в таблице 1. Дополнительно используется команда =, которая явно задает значение ячейки памяти в формате вывода его на экран консоли (+XXXX).

Команда запуска транслятора должна иметь вид: sat файл.sa файл.o, где файл.sa – имя файла, в котором содержится программа на Simple Assembler, файл.o – результат трансляции.

Транслятор с языка Simple Basic

Для упрощения программирования пользователю модели Simple Computer должен быть предоставлен транслятор с высокоуровневого языка Simple Basic. Файл, содержащий программу на Simple Basic, преобразуется в файл с кодом Simple Assembler. Затем Simple Assembler-файл транслируется в бинарный формат.

В языке Simple Basic используются следующие операторы: rem, input, output, goto, if, let, end. Пример программы на Simple Basic:

```
10  REM Это комментарий
20  INPUT A
30  INPUT B
40  LET C = A – B
50  IF C < 0 GOTO 20
60  PRINT C
70  END
```

Каждая строка программы состоит из номера строки, оператора Simple Basic и параметров. Номера строк должны следовать в возрастающем порядке. Все команды за исключением команды конца программы могут встречаться в программе многократно. Simple Basic должен оперировать с целыми выражениями, включающими операции +, -, *, и /. Приоритет операций аналогичен C. Для того чтобы изменить порядок вычисления, можно использовать скобки.

Транслятор должен распознавания только букв верхнего регистра, то есть все символы в программе на Simple Basic должны быть набраны в верхнем регистре (символ нижнего регистра приведет к ошибке). Имя переменной может состоять только из одной буквы. Simple Basic оперирует только с целыми значениями переменных, в нем отсутствует объявление переменных, а упоминание переменной автоматически вызывает её объявление и присваивает ей нулевое значение. Синтаксис языка не позволяет выполнять операций со строками.

Оформление отчета по РГЗ

Отчет о курсовой работе представляется в виде пояснительной записки (ПЗ), к которой прилагается диск с разработанным программным обеспечением. В пояснительную записку должны входить:

- титульный лист;
- полный текст задания к курсовой работе;
- реферат (объем ПЗ, количество таблиц, рисунков, схем, программ, приложений, краткая характеристика и результаты работы);
- содержание:

· постановка задачи исследования;
· блок-схемы используемых алгоритмов;
· программная реализация;
· результаты проведенного исследования;
· выводы;

- список использованной литературы;
- подпись, дата.

Пояснительная записка должна быть оформлена на листах формата А4, имеющих поля. Все листы следует сброшюровать и пронумеровать.

Сергей Николаевич Мамойленко
Юрий Сергеевич Майданов

Архитектура ЭВМ

Учебное пособие

Редактор:
Корректор:

Подписано в печать 12.04.2024г.
Формат бумаги 60 х 84/16, отпечатано на ризографе, шрифт № 10,
изд. л.6,6, заказ № 30, тираж – 100 экз., СибГУТИ.
630102, г. Новосибирск, ул. Кирова, д. 86