**Глава 9**

**Описание архитектуры**

**учебной ЭВМ**

**Современные процессоры и операционные системы — не слишком благоприятная среда для начального этапа изучения архитектуры ЭВМ.**

**Одним из решений этой проблемы может быть создание программных моделей учебных ЭВМ, которые, с одной стороны, достаточно просты, чтобы обучаемый мог освоить базовые понятия архитектуры (система команд, командный цикл, способы адресации, уровни памяти, способы взаимодействия процессора с памятью и внешними устройствами), с другой стороны — архитектурные особенности модели должны соответствовать тенденциям развития современных ЭВМ.**

**Программная модель позволяет реализовать доступ к различным элементам ЭВМ, обеспечивая удобство и наглядность. С другой стороны, модель позволяет игнорировать те особенности работы реальной ЭВМ, которые на данном уровне рассмотрения не являются существенными.**

**Далее приводится описание программной модели учебной ЭВМ 1, предназначенной для начальных этапов изучения архитектуры (в т. ч. на младших курсах вуза и даже в школе). Именно этим объясняется использование в модели десятичной системы счисления для кодирования команд и представления данных.**

**9.1. Структура ЭВМ**

**Моделируемая ЭВМ включает процессор, оперативную (ОЗУ) и сверхоперативную память, устройство ввода (УВв) и устройство вывода (УВыв). Процессор, в свою очередь, состоит из центрального устройства управления (УУ),**

**арифметического устройства (АУ) и системных регистров (CR, PC, SP и др.).**

**Структурная схема ЭВМ показана на рис. 9.1.**

**В ячейках ОЗУ хранятся команды и данные. Емкость ОЗУ составляет 1000 ячеек. По сигналу MWr выполняется запись содержимого регистра данных (MDR) в ячейку памяти с адресом, указанным в регистре адреса (MAR). По сигналу MRd происходит считывание — содержимое ячейки памяти**

**с адресом, находящимся в MAR, передается в MDR.**

**Сверхоперативная память с прямой адресацией содержит десять регистров общего назначения R0—R9. Доступ к ним осуществляется (аналогично доступу к ОЗУ) через регистры RAR и RDR.**

**АУ осуществляет выполнение одной из арифметических операций, определяемой кодом операции (СОР), над содержимым аккумулятора (Асс) и регистра операнда (DR). Результат операции всегда помещается в Асс. При завер-**

**шении выполнения операции АУ вырабатывает сигналы признаков результата: Z (равен 1, если результат равен нулю); S (равен 1, если результат отрицателен); OV (равен 1, если при выполнении операции произошло переполнение разрядной сетки). В случаях, когда эти условия не выполняются,**

**соответствующие сигналы имеют нулевое значение.**

**В модели ЭВМ предусмотрены внешние устройства двух типов. Во-первых, это регистры IR и OR, которые могут обмениваться с аккумулятором с помощью безадресных команд IN (Асс := IR) и OUT (OR := Асс). Во-вторых, это набор моделей внешних устройств, которые могут подключаться к системе и взаимодействовать с ней в соответствии с заложенными в моделях алгоритмами. Каждое внешнее устройство имеет ряд программно-доступных регистров, может иметь собственный обозреватель (окно видимых элементов).**

**Подробнее эти внешние устройства описаны в разд. 9.6.**

**УУ осуществляет выборку команд из ОЗУ в последовательности, определяемой естественным порядком выполнения команд (т. е. в порядке возрастания адресов команд в ОЗУ) или командами передачи управления; выборку из ОЗУ операндов, задаваемых адресами команды; инициирование выполнения**

**операции, предписанной командой; останов или переход к выполнению следующей команды.**

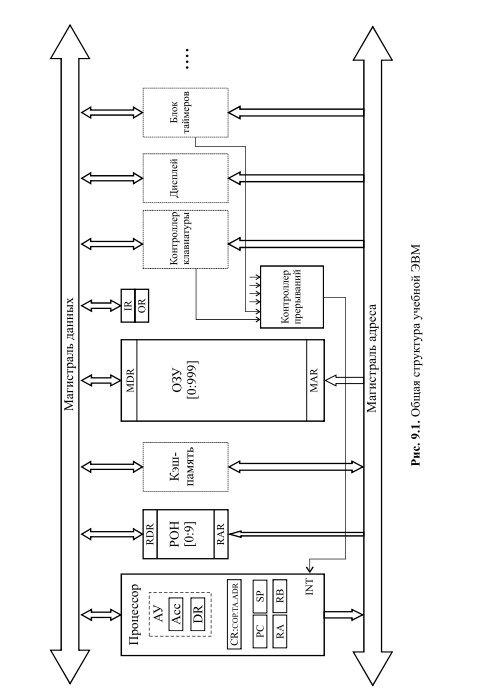
**В качестве сверхоперативной памяти в модель включены регистры общего назначения (РОН), и может подключаться модель кэш-памяти.**

**В состав УУ ЭВМ входят:**

** PC — счетчик адреса команды, содержащий адрес текущей команды;**

** CR — регистр команды, содержащий код команды;**

** RB — регистр базового адреса, содержащий базовый адрес;**

****

** SP — указатель стека, содержащий адрес верхушки стека;**

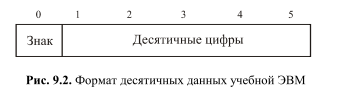
** RA — регистр адреса, содержащий исполнительный адрес при косвенной адресации.**

**Регистры Асc, DR, IR, OR, CR и все ячейки ОЗУ и РОН имеют длину 6 десятичных разрядов, регистры PC, SP, RA и RB — 3 разряда.**

**9.2. Представление данных в модели**

**Данные в ЭВМ представляются в формате, показанном на рис. 9.2. Это целые**

**десятичные числа, изменяющиеся в диапазоне –99 999…+99 999, содержащие знак и 5 десятичных цифр.**

****

**Старший разряд слова данных используется для кодирования знака: плюс (+) изображается как 0, минус (–) — как 1. Если результат арифметической операции выходит за пределы указанного диапазона, то говорят, что произошло**

**переполнение разрядной сетки. АЛУ в этом случае вырабатывает сигнал переполнения OV = 1. Результатом операции деления является целая часть частного. Деление на ноль вызывает переполнение.**

**9.3. Система команд**

**При рассмотрении системы команд ЭВМ обычно анализируют три аспекта:**

**форматы, способы адресации и систему операций.**

**9.3.1. Форматы команд**

**Большинство команд учебной ЭВМ являются одноадресными или безадресными, длиной в одно машинное слово (6 разрядов). Исключение составляют двухсловные команды с непосредственной адресацией и команда MOV , яв-**

**ляющаяся двухадресной.**

**В форматах команд выделяются три поля:**

** два старших разряда [0:1] определяют код операции СОР;**

** разряд 2 может определять тип адресации (в одном случае (формат 5а) он определяет номер регистра);**

** разряды [3:5] могут определять прямой или косвенный адрес памяти, номер регистра (в команде MOV номера двух регистров), адрес перехода или короткий непосредственный операнд. В двухсловных командах непосредственный операнд занимает поле [6:11].**

**Полный список форматов команд показан на рис. 9.3, где приняты следующие обозначения:**

** СОР — код операции;**

** ADR — адрес операнда в памяти;**

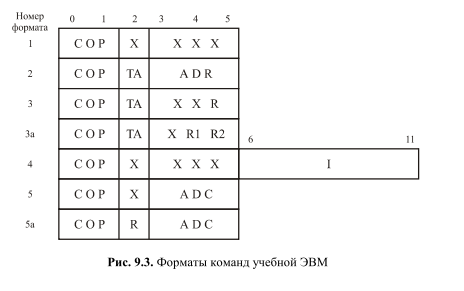
** ADC — адрес перехода;**

** I — непосредственный операнд;**

** R, Rl, R2 — номер регистра;**

** ТА — тип адресации;**

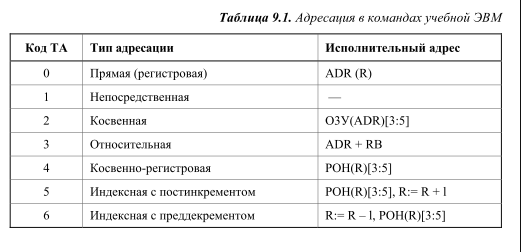
** X — разряд не используется.**

****

**9.3.2. Способы адресации**

**В ЭВМ принято различать пять основных способов адресации: прямая, косвенная, непосредственная, относительная, безадресная.**

**Каждый способ имеет разновидности. В модели учебной ЭВМ реализованы семь способов адресации, приведенные в табл. 9.1.**

****

**9.3.3. Система операций**

**Система команд учебной ЭВМ включает команды следующих классов:**

** арифметико-логические и специальные: сложение, вычитание, умножение, деление;**

** пересылки и загрузки: чтение, запись, пересылка (из регистра в регистр), помещение в стек, извлечение из стека, загрузка указателя стека, загрузка базового регистра;**

** ввода/вывода: ввод, вывод;**

** передачи управления: безусловный и шесть условных переходов, вызов подпрограммы, возврат из подпрограммы, цикл, программное прерывание, возврат из прерывания;**

** системные: пустая операция, разрешить прерывание, запретить прерывание, стоп.**

**Список команд учебной ЭВМ приведен в табл. 9.4 и 9.6.**

**9.4. Состояния и режимы работы ЭВМ**

**Ядром УУ ЭВМ является управляющий автомат (УА), вырабатывающий сигналы управления, которые инициируют работу АЛУ, РОН, ОЗУ и УВВ, передачу информации между регистрами устройств ЭВМ и действия над содержимым регистров УУ.**

**ЭВМ может находиться в одном из двух состояний: Останов и Работа.**

**В состояние Работа ЭВМ переходит по действию команд Пуск или Шаг.**

**Команда Пуск запускает выполнение программы, представляющую собой последовательность команд, записанных в ОЗУ, в автоматическом режиме до**

**команды HLT или точки останова. Программа выполняется по командам, начиная с ячейки ОЗУ, на которую указывает PC, причем изменение состояний**

**объектов модели отображается в окнах обозревателей.**

**В состояние Останов ЭВМ переходит по действию команды Стоп или автоматически в зависимости от установленного режима работы.**

**Команда Шаг, в зависимости от установленного режима работы, запускает выполнение одной команды или одной микрокоманды (если установлен Режим микрокоманд), после чего переходит в состояние Останов.**

**В состоянии Останов допускается просмотр и модификация объектов модели: регистров процессора и РОН, ячеек ОЗУ, устройств ввода/вывода. В процессе модификации ячеек ОЗУ и РОН можно вводить данные для программы, в ячейки ОЗУ — программу в кодах. Кроме того, в режиме Останов**

**можно менять параметры модели и режимы ее работы, вводить и/или редактировать программу в мнемокодах, ассемблировать мнемокоды, выполнять**

**стандартные операции с файлами.**

**9.5. Интерфейс пользователя**

**В программной модели учебной ЭВМ использован стандартный интерфейс Windows, реализованный в нескольких окнах.**

**Основное окно модели Модель учебной ЭВМ содержит главное меню и кнопки на панели управления. В рабочее поле окна выводятся сообщения о функционировании системы в целом. Эти сообщения группируются в фай-**

**ле logfile.txt (по умолчанию), сохраняются на диске и могут быть проанализированы после завершения сеанса работы с моделью.**

**Меню содержит следующие пункты и команды:**

** Файл:**

**• неактивные команды;**

**• Выход;**

** Вид:**

**• Показать все;**

**• Скрыть все;**

**• Процессор;**

**• Микрокомандный уровень;**

**• Память;**

**• Кэш-память;**

**• Программа;**

**• Текст программы;**

** Внешние устройства:**

**• Менеджер ВУ;**

**• окна подключенных ВУ;**

** Работа:**

**• Пуск;**

**• Стоп;**

**• Шаг;**

**• Режим микрокоманд;**

**• Кэш-память;**

**• Настройки.**

**Команды меню Вид открывают окна соответствующих обозревателей, описанные далее. Менеджер внешних устройств позволяет подключать/отключать внешние устройства, предусмотренные в системе. Команда вызова**

**менеджера внешних устройств выполняется при нажатии кнопки на панели инструментов. Подробнее о внешних устройствах и их обозревателях см. в разд. 9.6.**

**Команды меню Работа позволяют запустить программу в автоматическом (команда Пуск) или шаговом (команда Шаг) режиме, остановить выполнение программы в модели процессора (команда Стоп). Эти команды могут**

**выполняться при нажатии соответствующих одноименных кнопок на панели инструментов основного окна.**

**Команда Режим микрокоманд включает/выключает микрокомандный режим работы процессора, а команда Кэш-память подключает/отключает в системе модель этого устройства.**

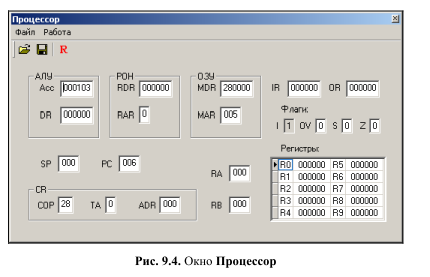
**Команда Настройки открывает диалоговое окно Параметры системы, позволяющее установить задержку реализации командного цикла (при выполнении программы в автоматическом режиме), а также установить параметры**

**файла logfile.txt, формируемого системой и записываемого на диск.**

**9.5.1. Окна основных обозревателей системы**

**Окно Процессор**

**Окно Процессор (рис. 9.4) обеспечивает доступ ко всем регистрам и флагам процессора.**

****

** Программно-доступные регистры и флаги:**

**• Асс — аккумулятор;**

**• PC — счетчик адреса команды, содержащий адрес текущей команды;**

**• SP — указатель стека, содержащий адрес верхушки стека;**

**• RB — регистр базового адреса, содержащий базовый адрес;**

**• RA — регистр адреса, содержащий исполнительный адрес при косвенной адресации;**

**• IR — входной регистр;**

**• OR — выходной регистр;**

**• I — флаг разрешения прерываний.**

** Системные регистры и флаги:**

**• DR — регистр данных АЛУ, содержащий второй операнд;**

**• MDR — регистр данных ОЗУ;**

**• MAR — регистр адреса ОЗУ;**

**• RDR — регистр данных блока РОН;**

**• RAR — регистр адреса блока РОН;**

**• CR — регистр команд, содержащий поля:**

**◊ СОР — код операции;**

**◊ ТА — тип адресации;**

**◊ ADR — адрес или непосредственный операнд;**

**• Z — флаг нулевого значения Асс;**

**• S — флаг отрицательного значения Асс;**

**• OV — флаг переполнения.**

**Регистры Асс, DR, IR, OR, CR и все ячейки ОЗУ и РОН имеют длину 6 десятичных разрядов, регистры PC, SP, RA и RB — 3 разряда.**

**В окне Процессор отражаются текущие значения регистров и флагов, причем в состоянии Останов все регистры, включая регистры блока РОН, и флаги (кроме флага I)**

**доступны для непосредственного редактирования.**

**Элементы управления окна Процессор включают меню и кнопки, вызывающие команды:**

** Сохранить;**

** Загрузить;**

** Reset;**

** Reset R0-R9 (только команда меню Работа).**

**Команды Сохранить и Загрузить позволяют сохранить текущее значение регистров и флагов процессора в файле и восстановить состояние процессора из файла. Команда Reset и кнопка R устанавливают все регистры (в т. ч.**

**блок РОН) в начальное (нулевое) значение. Содержимое ячеек памяти при этом не меняется. Выполняемая лишь из меню Работа команда Reset R0-R9 очищает только регистры блока РОН.**

**Окно Память**

**Окно Память (рис. 9.5) отражает текущее состояние ячеек ОЗУ. В этом окне допускается редактирование содержимого ячеек, кроме того, предусмотрена возможность выполнения (через меню или с помощью кнопок панели инструментов) пяти команд: Сохранить, Загрузить, Перейти к, Вставить, Убрать.**

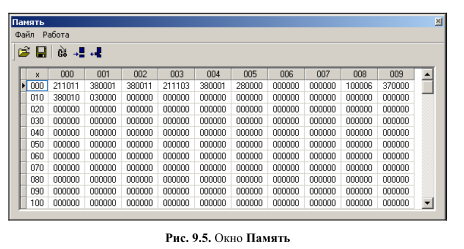
**Команды Сохранить, Загрузить во всех окнах, где они предусмотрены, работают одинаково — сохраняют в файле текущее состояние объекта (в данном случае памяти) и восстанавливают это состояние из выбранного файла,**

**причем файл в каждом окне записывается по умолчанию с характерным для этого окна расширением.**

**Команда Перейти к открывает диалоговое окно, позволяющее перейти на заданную ячейку ОЗУ.**

**Команда Убрать открывает диалог, в котором указывается диапазон ячеек с m по n . Содержимое ячеек в этом диапазоне теряется, а содержимое ячеек [ )1 ( + n : 999] перемещается в соседние ячейки с меньшими адресами.**

**Освободившиеся ячейки с адресами 999, 998, ... заполняются нулями.**

****

**Команда Вставить, позволяющая задать номера ячеек, перемещает содержимое всех ячеек, начиная от m-й на m n − позиций в направлении больших адресов, ячейки заданного диапазона [ m : n ] заполняются нулями, а содер**

**жимое последних ячеек памяти теряется.**

**Окно Текст программы**

**Окно Текст программы (рис. 9.6) содержит стандартное поле текстового редактора, в котором можно редактировать тексты, загружать в него текстовые файлы и сохранять подготовленный текст в виде файла.**

**Команды меню Файл:**

** Новая — открывает новый сеанс редактирования;**

** Загрузить — открывает стандартный диалог загрузки файла в окно редактора;**

** Сохранить — сохраняет файл под текущим именем;**

** Сохранить как — открывает стандартный диалог сохранения файла;**

** Вставить — позволяет вставить выбранный файл в позицию курсора.**

**Все перечисленные команды, кроме последней, дублированы кнопками на панели инструментов окна. На той же панели присутствует еще одна кнопка — Компилировать, которая запускает процедуру ассемблирования текста в поле редактора.**

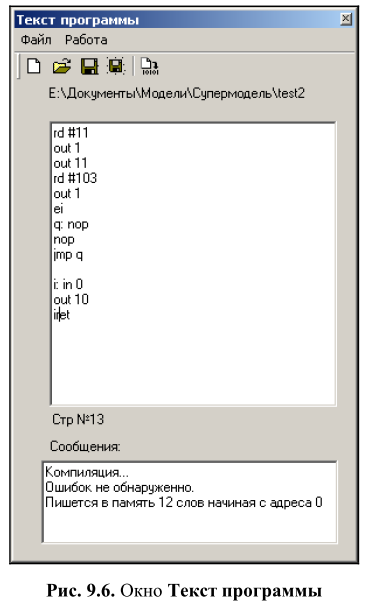
**Ту же процедуру можно запустить из меню Работа. Команда Адрес вставки позволяет задать адрес ячейки ОЗУ, начиная с которой программа будет размещаться в памяти.**

**По умолчанию этот адрес принят равным 0.**

**Ниже области редактирования в строку состояния выводится позиция текущей строки редактора — номер строки, в которой находится курсор.**

**В случае обнаружения синтаксических ошибок в тексте программы диагностические сообщения процесса компиляции выводятся в окно сообщений и**

**запись в память кодов (даже безошибочного начального фрагмента программы) не производится.**

****

**После исправления ошибок и повторной компиляции выдается сообщение об отсутствии ошибок, о расположении и размере области памяти, занятой под ассемблированную программу.**

**Набор текста программы производится по стандартным правилам языка ассемблера. В каждой строке может содержаться метка, одна команда и комментарий. Метка отделяется от команды двоеточием, символы после знака**

**"точка с запятой" до конца строки игнорируются компилятором и могут рассматриваться как комментарии. Строка может начинаться с ; и, следовательно, содержать только комментарии.**

**Окно Программа**

**Окно Программа (рис. 9.7) отображает таблицу, имеющую 300 строк и 4 столбца. Каждая строка таблицы соответствует дизассемблированной ячейке ОЗУ. Второй столбец содержит адрес ячейки ОЗУ, третий — дизассемблированный мнемокод, четвертый — машинный код команды. В первом столбце может помещаться указатель --> на текущую команду (текущее значение PC) и точка останова — красная заливка ячейки.**

**Окно Программа позволяет наблюдать процесс прохождения программы.**

**В этом окне ничего нельзя редактировать. Органы управления окна позволяют сохранить содержимое окна в виде текстового файла, выбрать начальный**

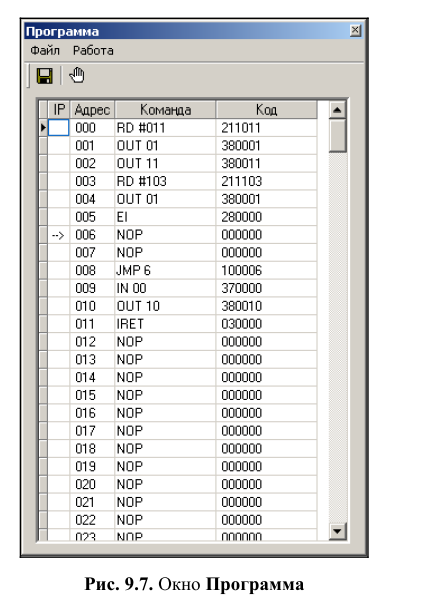
**адрес области ОЗУ, которая будет дизассемблироваться (размер области постоянный — 300 ячеек), а также установить/снять точку останова. Последнее**

**можно проделать тремя способами: командой Точка останова из меню Работа, кнопкой на панели инструментов или двойным щелчком мыши**

**в первой ячейке соответствующей строки. Характерно, что прочитать в это окно ничего нельзя. Сохраненный текстовый asm-файл можно загрузить в окно Текст программы, ассемблировать его и тогда дизассемблированное значение заданной области памяти автоматически появится в окне Программа. Такую процедуру удобно использовать, если программа изначально**

**пишется или редактируется непосредственно в памяти в машинных кодах.**

**Начальный адрес области дизассемблирования задается в диалоге командой Начальный адрес меню Работа.**

****

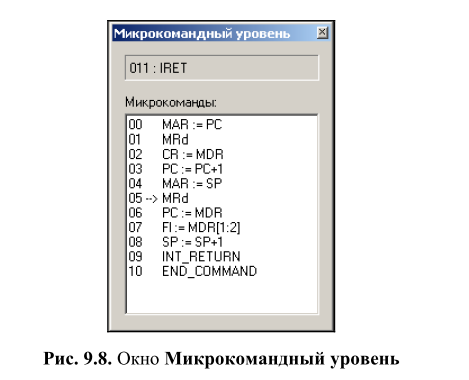
**Окно Микрокомандный уровень**

**Окно Микрокомандный уровень (рис. 9.8) используется только в режиме микрокоманд, который устанавливается командой Режим микрокоманд меню Работа. В это окно выводится мнемокод выполняемой команды, список**

**микрокоманд, ее реализующих, и указатель на текущую выполняемую микрокоманду.**

**Шаговый режим выполнения программы или запуск программы в автоматическом режиме с задержкой командного цикла позволяет наблюдать процесс**

**выполнения программы на уровне микрокоманд.**

****

**Если открыть окно Микрокомандный уровень, не установив режим микрокоманд в меню Работа, то после начала выполнения программы в режиме Шаг (или в автоматическом режиме) в строке сообщений окна будет выдано сообщение "Режим микрокоманд неактивен".**

**Окно Кэш-память**

**Окно Кэш-память используется в режиме с подключенной кэш-памятью.**

**Подробнее об этом режиме см. в разд. 9.8.**

**9.6. Внешние устройства**

**Модели внешних устройств (ВУ), используемые в описываемой системе, реализованы по единому принципу. С точки зрения процессора они представляют собой ряд программно-доступных регистров, лежащих в адресном про-**

**странстве ввода/вывода. Размер регистров ВУ совпадает с размером ячеек памяти и регистров данных процессора — шесть десятичных разрядов.**

**Доступ к регистрам ВУ осуществляется по командам IN aa , О UT aa , где aa — двухразрядный десятичный адрес регистра ВУ. Таким образом, общий объем адресного пространства ввода/вывода составляет 100 адресов. Следует помнить, что адресные пространства памяти и ввода/вывода в этой модели разделены.**

**Разные ВУ содержат различное число программно-доступных регистров, каждому из которых соответствует свой адрес, причем нумерация адресов всех ВУ начинается с 0. При создании ВУ ему ставится в соответствие базовый адрес в пространстве ввода/вывода, и все адреса его регистров становятся смещениями относительно этого базового адреса.**

**Если в системе создается несколько ВУ, то их базовые адреса следует выбирать с учетом величины адресного пространства, занимаемого этими устройствами, исключая наложение адресов.**

**Если ВУ способно формировать запрос на прерывание, то при создании ему ставится в соответствие вектор прерывания — десятичное число. Разным ВУ должны назначаться различные векторы прерываний.**

**Программная модель учебной ЭВМ комплектуется набором внешних устройств, включающим:**

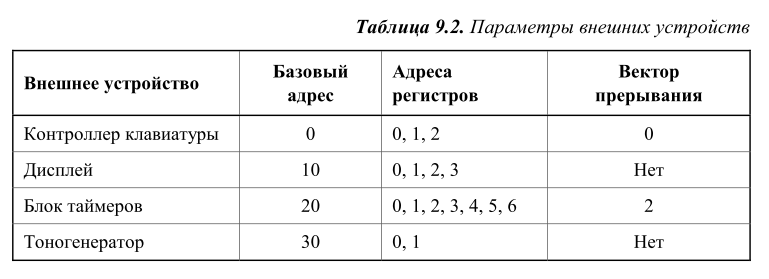
** контроллер клавиатуры;**

** дисплей;**

** блок таймеров;**

** тоногенератор,**

**которым по умолчанию присвоены параметры, перечисленные в табл. 9.2.**

****

**При создании устройств пользователь может изменить назначенные по умолчанию базовый адрес и вектор прерывания.**

**В описываемой версии системы не предусмотрена возможность подключения в систему нескольких одинаковых устройств.**

**Большинство внешних устройств содержит регистры управления CR и состояния SR, причем обычно регистры CR доступны только по записи, a SR — по чтению.**

**Регистр CR содержит флаги и поля, определяющие режимы работы ВУ, a SR — флаги, отражающие текущее состояние ВУ. Флаги SR устанавливаются аппаратно, но сбрасываются программно (или по внешнему сигналу).**

**Поля и флаги CR устанавливаются и сбрасываются программно при записи кода данных в регистр CR или специальными командами.**

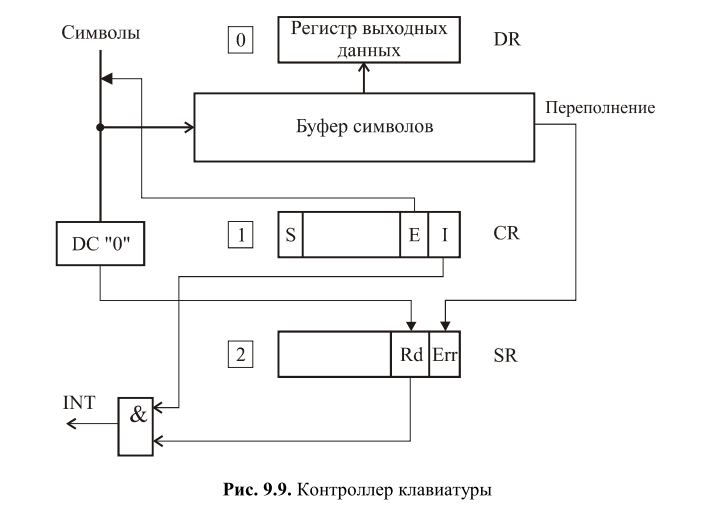
**Контроллер ВУ интерпретирует код, записываемый по адресу CR как команду, если третий разряд этого кода равен 1, или как записываемые в CR данные, если третий разряд равен 0. В случае получения командного слова**

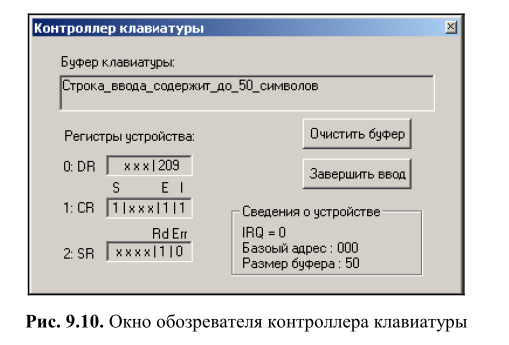
**запись в регистр CR не производится, а пятый разряд слова рассматривается как код операции.**

**9.6.1. Контроллер клавиатуры**

**Контроллер клавиатуры (рис. 9.9) представляет собой модель внешнего устройства, принимающего ASCII-коды 2 (American Standard Code for Information Interchange — американский стандартный код обмена информацией) от клавиатуры ПЭВМ.**

**Символы помещаются последовательно в буфер символов, размер которого установлен равным 50 символам, и отображаются в окне обозревателя (рис. 9.10).**

****

****

**В состав контроллера клавиатуры входят три программно-доступных регистра:**

** DR (адрес 0) — регистр данных;**

** CR (адрес 1) — регистр управления, определяет режимы работы контроллера и содержит следующие флаги:**

**• Е — флаг разрешения приема кодов в буфер;**

**• I — флаг разрешения прерывания;**

**• S — флаг режима посимвольного ввода;**

** SR (адрес 2) — регистр состояния, содержит два флага:**

**• Err — флаг ошибки;**

**• Rd — флаг готовности.**

**Регистр данных DR доступен только для чтения, через него считываются ASCII-коды из буфера, причем порядок чтения кодов из буфера соответствует порядку их записи в буфер — каждое чтение по адресу 0 автоматически**

**перемещает указатель чтения буфера. В каждый момент времени DR содержит код символа по адресу указателя чтения буфера.**

**Флаги регистра управления CR устанавливаются и сбрасываются программно.**

**Флаг Е, будучи установленным, разрешает прием кодов в буфер. При Е = 0 контроллер игнорирует нажатие на клавиатуре, прием кодов в буфер не производится. На считывание кодов из буфера флаг Е влияния не оказывает.**

**Флаг I, будучи установленным, разрешает при определенных условиях формирование контроллером запроса на прерывание. При I = 0 запрос на прерывание не формируется.**

**Флаг S = 1 устанавливает так называемый режим посимвольного ввода, иначе контроллер работает в обычном режиме. Флаг S устанавливается и сбрасывается программно, кроме того, S сбрасывается при нажатии кнопки Очистить**

**буфер в окне Контроллер клавиатуры.**

**Условия формирования запроса на прерывание определяются, с одной стороны, значением флага разрешения прерывания I, с другой — режимом работы**

**контроллера. В режиме посимвольного ввода запрос на прерывание формируется после ввода каждого символа (разумеется, при I = 1), в обычном режиме запрос будет сформирован по окончании набора строки.**

**Завершить набор строки можно, щелкнув по кнопке Завершить ввод в окне Контроллер клавиатуры (см. рис. 9.10). При этом устанавливается флаг готовности Rd (от англ. ready) в регистре состояния SR. Флаг ошибки Err**

**(от англ. error) в том же регистре устанавливается при попытке ввода в буфер 51-го символа. Ввод 51-го и всех последующих символов блокируется.**

**Сброс флага Rd осуществляется автоматически при чтении из регистра DR, флаг Err сбрасывается программно. Кроме того, оба эти флага сбрасываются при нажатии кнопки Очистить буфер в окне Контроллер клавиатуры; одновременно со сбросом флагов производится очистка буфера — весь буфер**

**заполняется кодами 00h, и указатели записи и чтения устанавливаются на начало буфера.**

**Для программного управления контроллером предусмотрен ряд командных слов. Все команды выполняются при записи по адресу регистра управления**

**CR кодов с 1 в третьем разряде.**

**Контроллер клавиатуры интерпретирует следующие командные слова:**

** xxxl0l — очистить буфер (действие команды эквивалентно нажатию кноп-**

**ки Очистить буфер);**

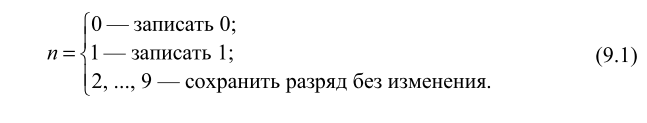
** ххх102 — сбросить флаг Err в регистре SR;**

** ххх10З — установить флаг S в регистре CR;**

** ххх104 — сбросить флаг S в регистре CR.**

**Если по адресу 1 произвести запись числа ххх0nn, то произойдет изменение**

**4-го и 5-го разрядов регистра CR по следующему правилу:**

****

**9.6.2. Дисплей**

**Дисплей (рис. 9.11) представляет собой модель внешнего устройства, реализующую функции символьного дисплея. Дисплей может отображать символы, задаваемые ASCII-кодами, поступающими на его регистр данных. Дисплей включает:**

** видеопамять объемом 128 слов (ОЗУ дисплея);**

** символьный экран размером 8 строк по 16 символов в строке;**

** четыре программно-доступных регистра:**

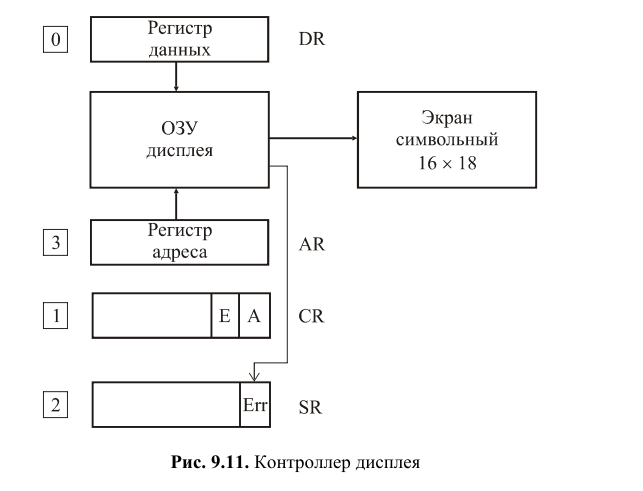
**DR (адрес 0) — регистр данных;**

**CR (адрес 1) — регистр управления;**

**SR (адрес 2) — регистр состояния;**

**AR (адрес 3) — регистр адреса.**

**Через регистры адреса AR и данных DR по записи и чтению осуществляется доступ к ячейкам видеопамяти. При обращении к регистру DR по записи содержимое аккумулятора записывается в DR и в ячейку видеопамяти, адрес которой установлен в регистре AR.**

****

**Регистр управления CR доступен только по записи и содержит в 4-м и 5-м разрядах соответственно два флага:**

** Е — флаг разрешения работы дисплея; при Е = 0 запись в регистры AR и DR блокируется;**

** А — флаг автоинкремента адреса; при А = 1 содержимое AR автоматически увеличивается на 1 после любого обращения к регистру DR — по записи или чтению.**

**Изменить значения этих флагов можно, если записать по адресу CR (по умолчанию — 11) код ххх0nn, при этом изменение 4-го и 5-го разрядов регистра CR произойдет согласно выражению (9.1).**

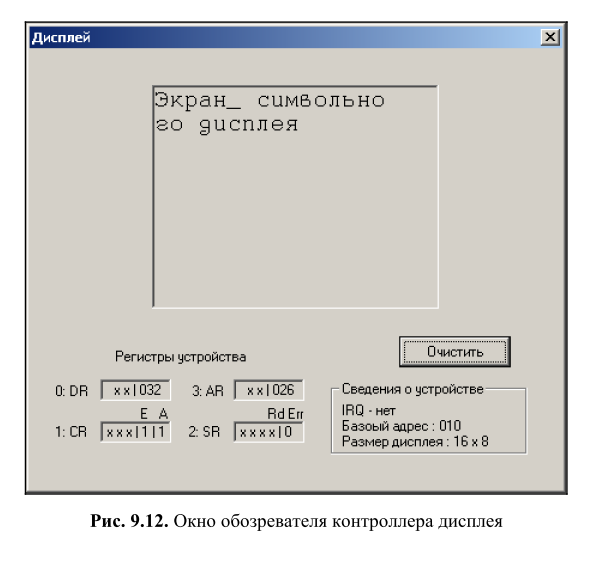
**Для программного управления дисплеем предусмотрены две команды, коды которых должны записываться по адресу регистра CR, причем в третьем разряде командных слов обязательно должна быть 1:**

** xxxl0l — очистить дисплей (действие команды эквивалентно нажатию кнопки Очистить в окне Дисплей), при этом очищается видеопамять (в каждую ячейку записывается код пробела — 032), устанавливается в 000**

**регистр адреса AR и сбрасываются флаги ошибки Err и автоинкремента А;**

** ххх102 — сбросить флаг ошибки Err.**

**Регистр состояния SR доступен только по чтению и содержит единственный флаг (в пятом разряде) ошибки Err. Этот флаг устанавливается аппаратно при попытке записать в регистр адреса число, большее 127, причем как в режиме прямой записи в AR, так и в режиме автоинкремента после обращения по адресу 127. Сбрасывается флаг Err программно или при нажатии кнопки Очистить в окне Дисплей (рис. 9.12).**

****

**9.6.3. Блок таймеров**

**Блок таймеров (рис. 9.13) включает в себя три однотипных канала, каждый из которых содержит:**

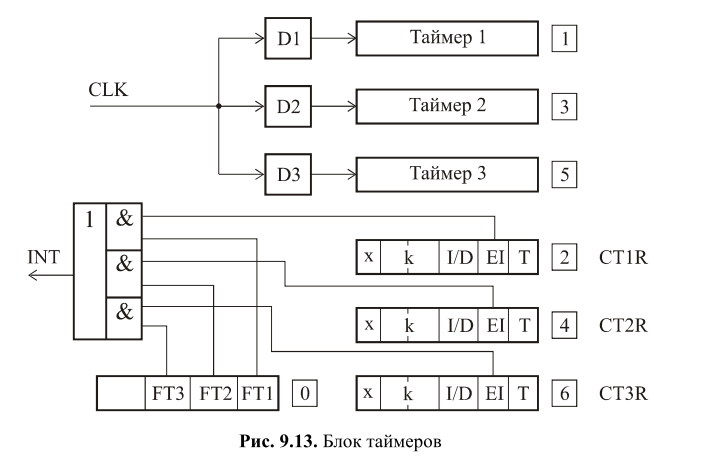
** пятиразрядный десятичный реверсивный счетчик Т, на вход которого поступают метки времени (таймер);**

** программируемый предделитель D;**

** регистр управления таймером CTR;**

** флаг переполнения таймера FT.**

**Регистры таймеров Т доступны по записи и чтению (адреса 1, 3, 5 соответственно для T1, T2, ТЗ). Программа в любой момент может считать текущее содержимое таймера или записать в него новое значение.**

****

**На входы предделителей поступают общие для всех каналов метки времени CLK с периодом 1 мс. Предделители в каждом канале программируются независимо, поэтому таймеры могут работать с различной частотой.**

**Регистры управления CTR доступны по записи и чтению (адреса 2, 4, 6) и содержат следующие поля:**

** Т (разряд 5) — флаг включения таймера;**

** EI (разряд 4) — флаг разрешения формирования запроса на прерывание при переполнении таймера;**

** I/D (разряд 3) — направление счета (инкремент/декремент), при I/D = 0**

**таймер работает на сложение, при I/D = 1 — на вычитание;**

** k (разряды [1:2]) — коэффициент деления предделителя (от 1 до 99).**

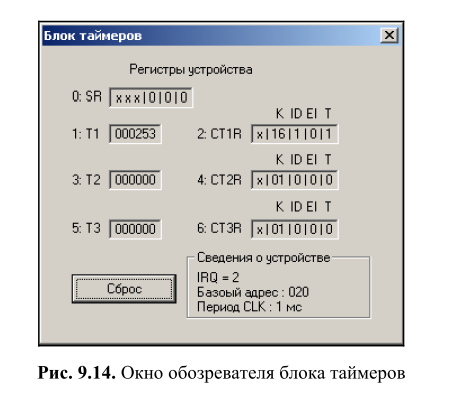
**Флаги переполнения таймеров собраны в один регистр — доступный только по чтению регистр состояния SR, имеющий адрес 0. Разряды регистра (5, 4 и 3 для T1, T2, ТЗ соответственно) устанавливаются в 1 при переполнении соот-**

**ветствующего таймера. Для таймера, работающего на сложение, переполнение наступает при переходе его состояния из 99 999 в 0, для вычитающего**

**таймера — переход из 0 в 99 999.**

**В окне обозревателя (рис. 9.14) предусмотрена кнопка Сброс, нажатие которой сбрасывает в 0 все регистры блока таймеров, кроме CTR, которые устанавливаются в состояние 001000. Таким образом, все три таймера обнуляются,**

**переключаются в режим инкремента, прекращается счет, запрещаются прерывания, сбрасываются флаги переполнения и устанавливаются коэффициенты деления предделителей равными 01.**

****

**Программное управление режимами блока таймеров осуществляется путем записи в регистры CTR соответствующих кодов. Запись по адресу SR числа с 1 в третьем разряде интерпретируется блоком таймеров как команда, причем младшие разряды этого числа определяют код команды:**

** xxxl00 — общий сброс (эквивалентна нажатию кнопки Сброс в окне обозревателя);**

** xxxl0l — сброс флага переполнения таймера FT1;**

** xxxl02 — сброс флага переполнения таймера FT2;**

** xxxl03 — сброс флага переполнения таймера FT3.**

**9.6.4. Тоногенератор**

**Модель этого простого внешнего устройства не имеет собственного обозревателя, содержит всего два регистра, доступных только для записи:**

** FR (адрес 0) — регистр частоты звучания (Гц):**

** LR (адрес 1) — регистр длительности звучания (мс).**

**По умолчанию базовый адрес тоногенератора — 30. Сначала следует записать в FR требуемую частоту тона в герцах, затем в LR — длительность звучания в миллисекундах. Запись числа по адресу регистра LR одновременно**

**является командой на начало звучания.**

**9.7. Подсистема прерываний**

**В модели учебной ЭВМ предусмотрен механизм векторных внешних прерываний. Внешние устройства формируют запросы на прерывания, которые поступают на входы контроллера прерываний. При подключении ВУ, способ-**

**ного формировать запрос на прерывание, ему ставится в соответствие номер входа контроллера прерываний — вектор прерывания, принимающий значение в диапазоне 0—9.**

**Контроллер передает вектор, соответствующий запросу, процессору, который начинает процедуру обслуживания прерывания.**

**Каждому из возможных в системе прерываний должен соответствовать так**

**называемый обработчик прерывания — подпрограмма, вызываемая при возникновении события конкретного прерывания.**

**Механизм прерываний, реализованный в модели учебной ЭВМ, поддерживает таблицу векторов прерываний, которая создается в оперативной памяти моделью операционной системы (если она используется) или непосредственно пользователем.**

**Номер строки таблицы соответствует вектору прерывания, а элемент таблицы — ячейка памяти, в трех младших разрядах которой размещается начальный адрес подпрограммы, обслуживающей прерывание с этим вектором.**

**Таблица прерываний в рассматриваемой модели жестко фиксирована — она занимает ячейки памяти с адресами 100—109. Таким образом, адрес обработчика с вектором 0 должен располагаться в ячейке 100, с вектором 2 — в**

**ячейке 102. При работе с прерываниями не рекомендуется использовать ячейки 100—109 для других целей.**

**Процессор начинает обработку прерывания (если они разрешены), завершив текущую команду. При этом он:**

**1. Получает от контроллера вектор прерывания.**

**2. Формирует и помещает в верхушку стека слово, три младших разряда ([3:5]) которого — текущее значение PC (адрес возврата из прерывания), а разряды [1:2] сохраняют десятичный эквивалент шестнадцатеричной цифры, определяющей значение вектора флагов (I, OV, S, Z).**

**Например, если**

**I = 1, OV = 0, S = 1, Z = 1, то в разряды [1:2] запишется число**

****

**3. Сбрасывает в 0 флаг разрешения прерывания I.**

**4. Извлекает из таблицы векторов прерываний адрес обработчика, соответствующий обслуживаемому вектору, и помещает его в PC, осуществляя тем самым переход на подпрограмму обработчика прерывания.**

**Таким образом, вызов обработчика прерывания, в отличие от вызова подпрограммы, связан с помещением в стек не только адреса возврата, но и текущего значения вектора флагов. Поэтому последней командой подпрограммы**

**обработчика должна быть команда IRET , которая не только возвращает в PC три младшие разряда ячейки — верхушки стека (как RET ), но и восстанавливает те значения флагов, которые были в момент перехода на обработчик**

**прерывания.**

**Не всякое событие, которое может вызвать прерывание, приводит к прерыванию текущей программы. В состав процессора входит программно-доступный флаг I разрешения прерывания. При I = 0 процессор не реагирует**

**на запросы прерываний. После сброса процессора флаг I так же сброшен и все прерывания запрещены. Для того чтобы разрешить прерывания, следует в программе выполнить команду EI (от англ. enable interrupt).**

**Ранее отмечалось, что при переходе на обработчик прерывания флаг I автоматически сбрасывается, в этом случае прервать обслуживание одного прерывания другим прерыванием нельзя. По команде IRET значение флагов вос-**

**станавливается, в т. ч. вновь устанавливается I = 1, следовательно, в основной программе прерывания опять разрешены.**

**Если требуется разрешить другие прерывания в обработчике прерывания, достаточно в нем выполнить команду EI . Контроллер прерываний и процессор на аппаратном уровне блокируют попытки запустить прерывание, если его обработчик начал, но не завершил работу.**

**Таким образом, флаг I разрешает или запрещает все прерывания системы.**

**Если требуется выборочно разрешить некоторое подмножество прерываний, используются программно-доступные флаги разрешения прерываний непо-**

**средственно на внешних устройствах.**

**Как правило, каждое внешнее устройство, которое может вызвать прерывание, содержит в составе своих регистров разряд флага разрешения прерывания (см. формат регистров CR и CTR на рис. 9.9, 9.13), по умолчанию установленный в 0. Если оставить этот флаг в нуле, то внешнему устройству**

**запрещается формировать запрос контроллеру прерываний.**

**Иногда бывает удобно (например, в режиме отладки) иметь возможность вызвать обработчик прерывания непосредственно из программы. Если использовать для этих целей команду CALL , которая помещает в стек только адрес**

**возврата, то команда IRET , размещенная последней в обработчике, может исказить значения флагов (все они будут сброшены в 0, т. к. команда CALL формирует только три младшие разряда ячейки верхушки стека, оставляя ос-**

**тальные разряды в 000).**

**Поэтому в системах команд многих ЭВМ, в т. ч. и нашей модели, имеются команды вызова прерываний — INT n (в нашей модели n ∈ {0, 1, ..., 9}), где n — вектор прерывания. Процессор, выполняя команду INT n , произво-**

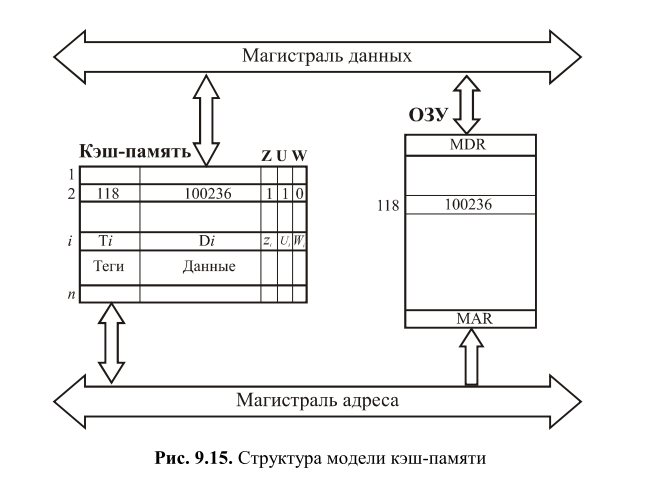
**дит те же действия, что и при обработке прерывания с вектором n .**

**Характерно, что с помощью команды INT n можно вызвать обработчик прерывания даже в том случае, когда флаг разрешения прерывания I сброшен.**

**9.8. Программная модель кэш-памяти**

**К описанной в разд. 9.1 программной модели учебной ЭВМ может быть подключена программная модель кэш-памяти, структура которой в общем виде отображена на рис. 5.2. Конкретная реализация кэш-памяти в описываемой**

**программной модели показана на рис. 9.15.**

****

**Кэш-память содержит N ячеек (в модели N может выбираться из множества {4, 8, 16, 32}), каждая из которых включает трехразрядное поле тега (адреса ОЗУ), шестиразрядное поле данных и три однобитовых признака (флага):**

** Z — признак занятости ячейки;**

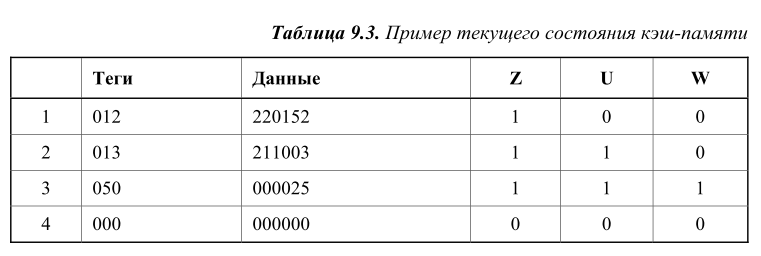
** U — признак использования;**

** W — признак записи в ячейку.**

**Таким образом, каждая ячейка кэш-памяти может дублировать одну любую ячейку ОЗУ, причем отмечается ее занятость (в начале работы модели все ячейки кэш-памяти свободны, 0 Z = ∀ i), факт записи информации в ячейку во время пребывания ее в кэш-памяти, а также использование ячейки (т. е. любое обращение к ней).**

**Текущее состояние кэш-памяти отображается на экране в отдельном окне в форме таблицы, причем количество строк соответствует выбранному числу ячеек кэш. Столбцы таблицы определяют содержимое полей ячеек, напри-**

**мер, так, как показано в табл. 9.3.**

****

**Для настройки параметров кэш-памяти можно воспользоваться диалоговым окном Кэш-память, вызываемым командой Вид | Кэш-память, а затем нажать первую кнопку на панели инструментов открытого окна. После этих действий появится диалоговое окно Параметры кэш-памяти, позволяющее выбрать размер кэш-памяти, способ записи в нее информации и алгоритм**

**замещения ячеек.**

**Напомним, что при сквозной записи при кэш-попадании в процессорных циклах записи осуществляется запись как в ячейку кэш-памяти, так и в ячейку ОЗУ, а при обратной записи — только в ячейку кэш-памяти, причем эта**

**ячейка отмечается битом записи ( 1 : W = i). При очистке ячеек, отмеченных битом записи, необходимо переписать измененное значение ноля данных в соответствующую ячейку ОЗУ.**

**При кэш-промахе следует поместить в кэш-память адресуемую процессором**

**ячейку. При наличии свободных ячеек кэш-памяти требуемое слово помещается в одну из них (в порядке очереди). При отсутствии свободных ячеек следует отыскать ячейку кэш-памяти, содержимое которой можно удалить, записав на его место требуемые данные (команду). Поиск такой ячейки осуществляется с использованием алгоритма замещения строк.**

**В модели реализованы три различных алгоритма замещения строк:**

** случайное замещение, при реализации которого номер ячейки кэш-памяти выбирается случайным образом;**

** очередь, при которой выбор замещаемой ячейки определяется временем пребывания ее в кэш-памяти;**

** бит использования, случайный выбор осуществляется только из тех ячеек, которые имеют нулевое значение флага использования.**

**Напомним, что бит использования устанавливается в 1 при любом обращении к ячейке, однако, как только все биты**

**iU установятся в 1, все они тут же сбрасываются в 0, так что в кэш всегда ячейки разбиты на два непересекаю-**

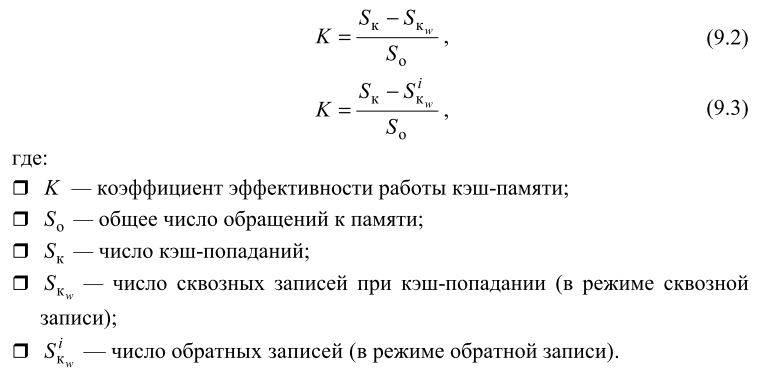
**щихся подмножества по значению бита U — те, обращение к которым состоялось относительно недавно (после последнего сброса вектора U ), имеют**

**значение U = 1, иные — со значением U = 0 являются "кандидатами на удаление" при использовании алгоритма замещения "бит использования".**

**Если в параметрах кэш-памяти установлен флаг "с учетом бита записи", то все три алгоритма замещения осуществляют поиск "кандидата на удаление" прежде всего среди тех ячеек, признак записи которых не установлен, а при**

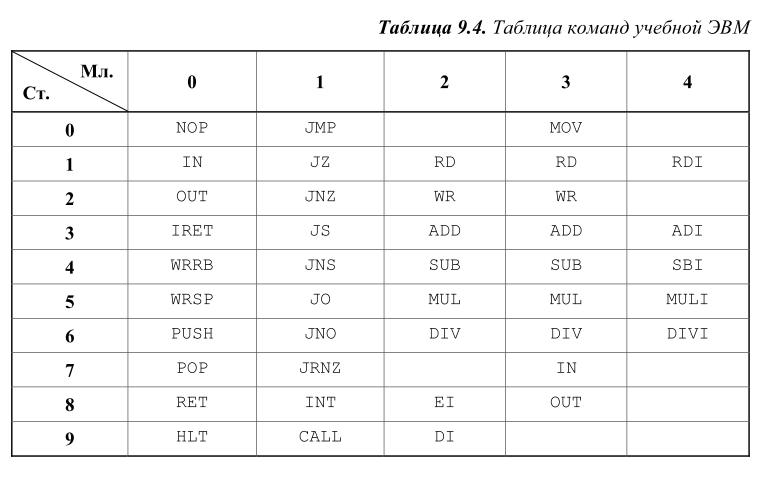
**отсутствии таких ячеек (что крайне маловероятно) — среди всех ячеек кэшпамяти. При снятом флаге "с учетом бита записи" поиск осуществляется по всем ячейкам кэш-памяти без учета значения W.**

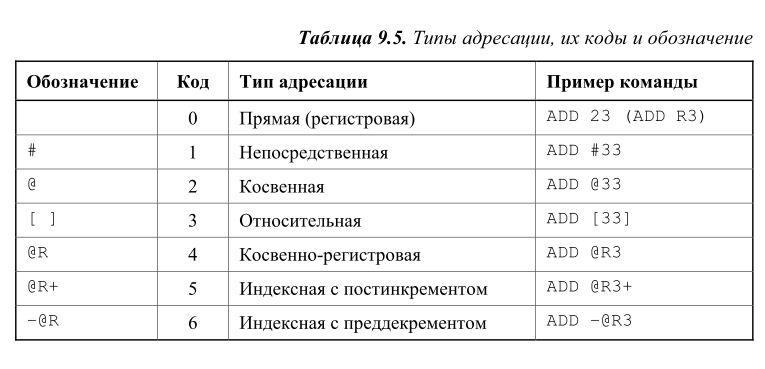
**Оценка эффективности работы системы с кэш-памятью определяется числом кэш-попаданий по отношению к общему числу обращений к памяти. Учитывая разницу в алгоритмах записи в режимах сквозной и обратной записи, эффективность использования кэш-памяти вычисляется по следующим выражениям (соответственно для сквозной и обратной записи):**

****

**9.9. Вспомогательные таблицы**

**В данном разделе представлены вспомогательные таблицы (табл. 9.4—9.8) для работы с моделью учебной ЭВМ.**

****

****

**В табл. 9.6 приняты следующие обозначения:**

** DD — данные, формируемые командой в качестве (второго) операнда:**

**прямо или косвенно адресуемая ячейка памяти или трехразрядный непосредственный операнд;**

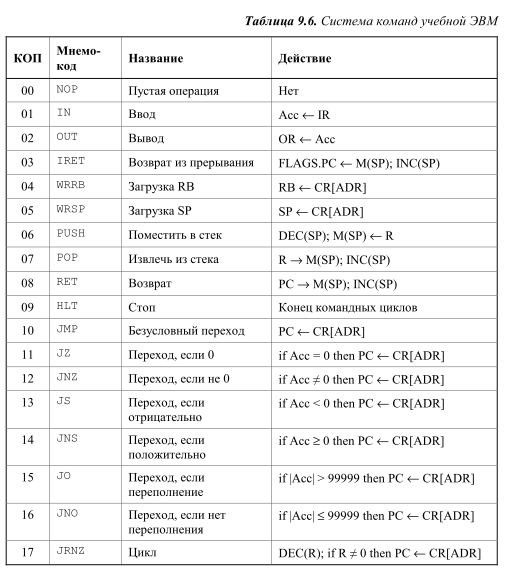
** R\* — содержимое регистра или косвенно адресуемая через регистр ячейка памяти;**

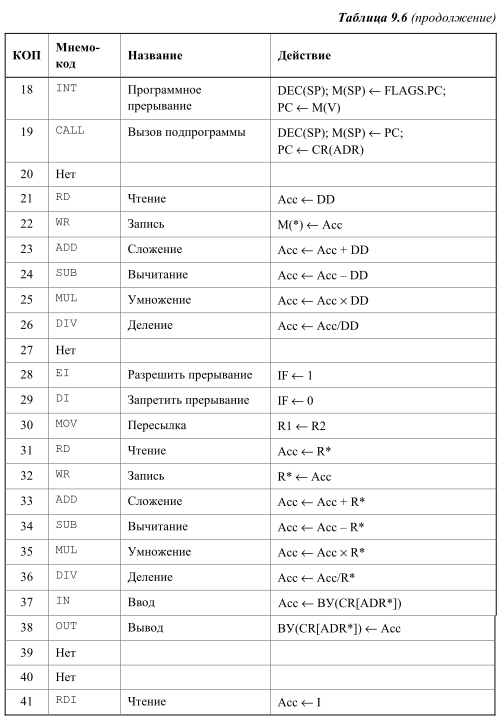
** ADR\* — два младших разряда ADR поля регистра CR;**

** V — адрес памяти, соответствующий вектору прерывания;**

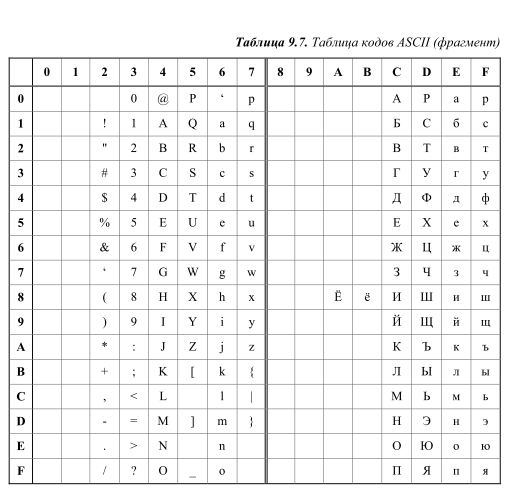
** М(\*) — ячейка памяти, прямо или косвенно адресуемая в команде;**

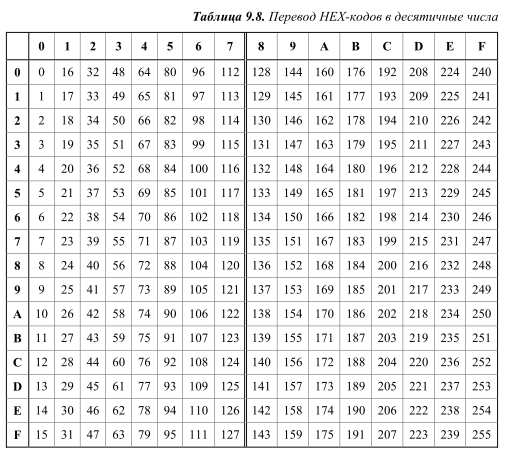
** I — пятиразрядный непосредственный операнд со знаком.**

****

****

****

****

****

**Глава 10**

**Лабораторные работы**

**Цикл лабораторных работ рассчитан на выполнение студентами в рамках изучения курсов "Архитектура ЭВМ" и других, подобных по содержанию (№ 3—10).**

**Работы № 3—10 выполняются на программной модели учебной ЭВМ и взаимодействующих с ней программных моделях ВУ и кэш-памяти, описанных в *главе 9*.**

**Цикл включает работы и задания различного уровня.**

**Лабораторные работы № 3—6 ориентированы на первичное знакомство с архитектурой процессора, системой команд, способами адресации и основными приемами программирования на машинно-ориентированном языке. Лабораторная работа № 7 иллюстрирует реализацию командного цикла процессора на уровне микроопераций. Лабораторная работа № 8 посвящена способам организации связи процессора с внешними устройствами, а в лабораторных работах № 9 и 10 рассматривается организация кэш-памяти и эффективность различных алгоритмов замещения.**

**Описание работы включает постановку задачи, пример выполнения, набор вариантов индивидуальных заданий, порядок выполнения работы, требования к содержанию отчета и контрольные вопросы.**

**10.3. Лабораторная работа № 3.**

**Архитектура ЭВМ и система команд**

**10.3.1. Общие положения**

**Для решения с помощью ЭВМ некоторой задачи должна быть разработана ьпрограмма. Программа на языке ЭВМ представляет собой последовательность команд. Код каждой команды определяет выполняемую операцию, тип адресации и адрес. Выполнение программы, записанной в памяти ЭВМ, осуществляется последовательно по командам в порядке возрастания адресов команд или в порядке, определяемом командами передачи управления.**

**Для того чтобы получить результат выполнения программы, пользователь должен:**

**􀂈ввести программу в память ЭВМ;**

**􀂈определить, если это необходимо, содержимое ячеек ОЗУ и РОН, содержащих исходные данные, а также регистров IR и BR;**

**􀂈установить в PC стартовый адрес программы;**

**􀂈перевести модель в режим Работа.**

**Ввод программы может осуществляться как в машинных**

**кодах непосредственно в память модели, так и в мнемокодах в окно Текст программы с последующим ассемблированием.**

**Цель настоящей лабораторной работы — знакомство с интерфейсом модели ЭВМ, методами ввода и отладки программы, действиями основных классов**

**команд и способов адресации. Для этого необходимо ввести в память ЭВМ и выполнить в режиме Шаг некоторую последовательность команд (определенную вариантом задания) и зафиксировать все изменения на уровне про-**

**граммно-доступных объектов ЭВМ, происходящие при выполнении этих команд.**

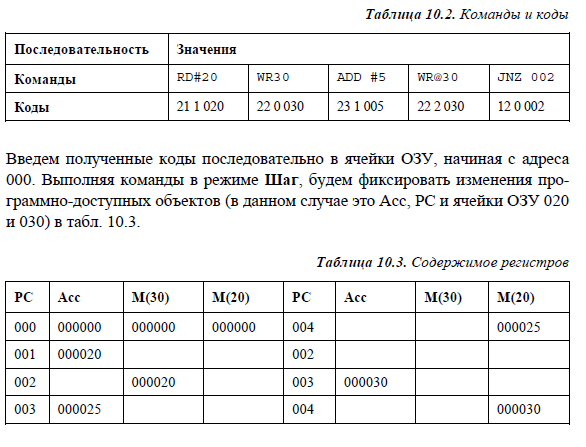
**Команды в память учебной ЭВМ вводятся в виде шестиразрядных десятичных чисел (см. форматы команд на рис. 8.3, коды команд и способов адресации в табл. 8.2—8.4).**

**В данной лабораторной работе будем программировать ЭВМ в машинных кодах.**

**10.3.2. Пример**

**Дана последовательность мнемокодов, которую необходимо преобразовать в машинные коды, занести в ОЗУ ЭВМ, выполнить в режиме Шаг и зафиксировать изменение состояний программно-доступных объектов ЭВМ**

**(табл. 10.2).**

******

**10.3.3. Задание**

**1. Ознакомиться с архитектурой ЭВМ *(см. часть I)*.**

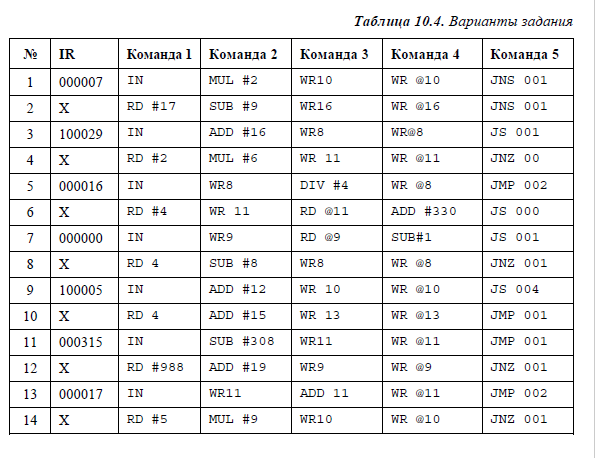
**2. Записать в ОЗУ "программу", состоящую из пяти команд — варианты задания выбрать из табл. 10.4. Команды разместить в последовательных ячейках памяти.**

**3. При необходимости установить начальное значение в устройство ввода IR.**

**4. Определить те программно-доступные объекты ЭВМ, которые будут изменяться при выполнении этих команд.**

**5. Выполнить в режиме Шаг введенную последовательность команд, фиксируя изменения значений объектов, определенных в п. 4, в таблице (см. форму табл. 10.3).**

**6. Если в программе образуется цикл, необходимо просмотреть не более двух повторений каждой команды, входящей в тело цикла.**

******

**10.3.4. Содержание отчета**

1. Формулировка варианта задания.

2. Машинные коды команд, соответствующих варианту задания.

3. Результаты выполнения последовательности команд в форме табл. 10.3.

**10.3.5. Контрольные вопросы**

1. Из каких основных частей состоит ЭВМ и какие из них представлены

в модели?

2. Что такое система команд ЭВМ?

3. Какие классы команд представлены в модели?

4. Какие действия выполняют команды передачи управления?

5. Какие способы адресации использованы в модели ЭВМ? В чем отличие

между ними?

6. Какие ограничения накладываются на способ представления данных

в модели ЭВМ?

7. Какие режимы работы предусмотрены в модели и в чем отличие между

ними?

8. Как записать программу в машинных кодах в память модели ЭВМ?

9. Как просмотреть содержимое регистров процессора и изменить содер-

жимое некоторых регистров?

10. Как просмотреть и, при необходимости, отредактировать содержимоеячейки памяти?

11. Как запустить выполнение программы в режиме приостановки работы

после выполнения каждой команды?

12. Какие способы адресации операндов применяются в командах ЭВМ?

13. Какие команды относятся к классу передачи управления?

**10.4. Лабораторная работа № 4.**

**Программирование разветвляющегося**

**процесса**

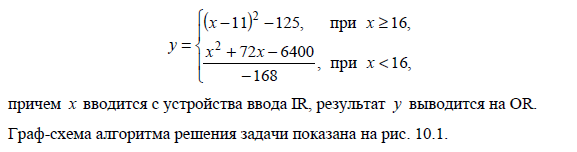
Для реализации алгоритмов, пути в которых зависят от исходных данных,

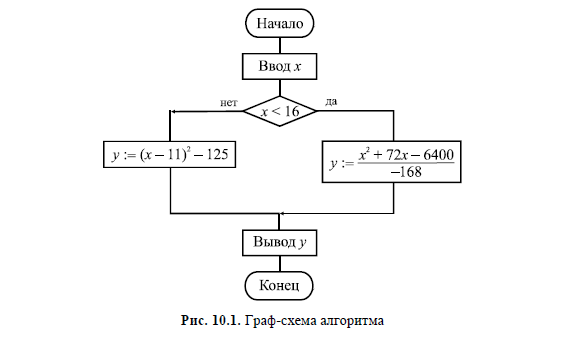
используют команды условной передачи управления.

**10.4.1. Пример**

В качестве примера (несколько упрощенного по сравнению с заданиями ла-

бораторной работы № 2) рассмотрим программу вычисления функции





В данной лабораторной работе используются двухсловные команды с непо-

средственной адресацией, позволяющие оперировать отрицательными чис-

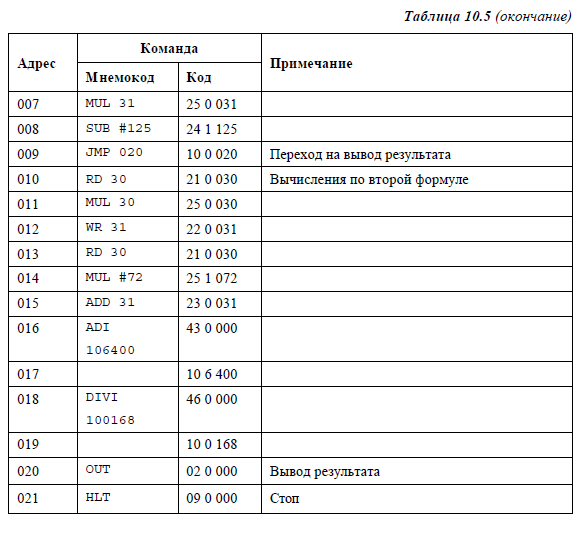
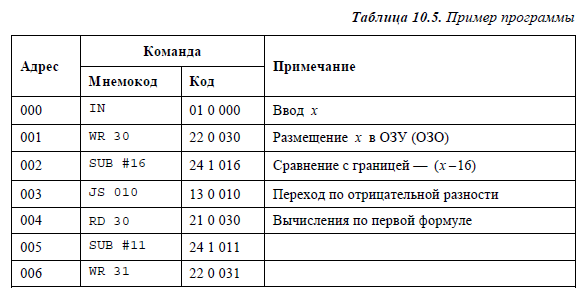
лами и числами по модулю, превышающие 999, в качестве непосредственно-

го операнда.

Оценив размер программы примерно в 20—25 команд, отведем для области

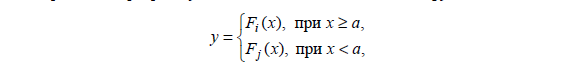
данных ячейки ОЗУ, начиная с адреса 030. Составленная программа с ком-

ментариями представлена в виде табл. 10.5.

******

**10.4.2. Задание**

1. Разработать программу вычисления и вывода значения функции:

для вводимого из IR значения аргумента *x* . Функции и допустимые пре-

делы изменения аргумента приведены в табл. 10.6, варианты заданий —

в табл. 10.7.

2. Исходя из допустимых пределов изменения аргумента функций (табл. 10.6)

и значения параметра *a* для своего варианта задания (табл. 10.7) выделить

на числовой оси *Ox* области, в которых функция *y* вычисляется по пред-

ставленной в п. 1 формуле, и недопустимые значения аргумента. На недо-

пустимых значениях аргумента программа должна выдавать на OR мак-

симальное по модулю отрицательное число: 199 999.

3. Ввести текст программы в окно **Текст программы**, при этом возможен

набор и редактирование текста непосредственно в окне **Текст программы**

или загрузка текста из файла, подготовленного в другом редакторе.

4. Ассемблировать текст программы, при необходимости исправить синтак-

сические ошибки.

5. Отладить программу. Для этого:

а) записать в IR значение аргумента *x* *a* (в области допустимых значений);

б) записать в PC стартовый адрес программы;

в) проверить правильность выполнения программы (т. е. правильность ре-

зультата и адреса останова) в автоматическом режиме. В случае нали-

чия ошибки выполнить пп. 5, *г* и 5, *д*; иначе перейти к п. 5, *е*;

г) записать в PC стартовый адрес программы;

д) наблюдая выполнение программы в режиме **Шаг**, найти команду, яв-

ляющуюся причиной ошибки; исправить ее; выполнить пп. 5, *а* — 5, *в*;

е) записать в IR значение аргумента *x* *a* (в области допустимых значе-

ний); выполнить пп. 5, *б* и 5, *в*;

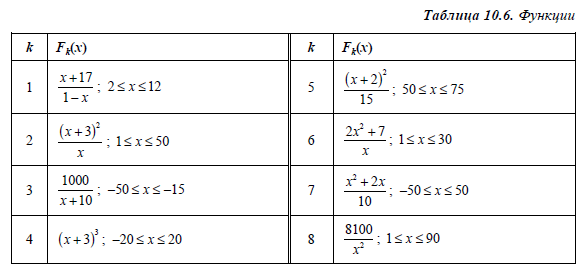
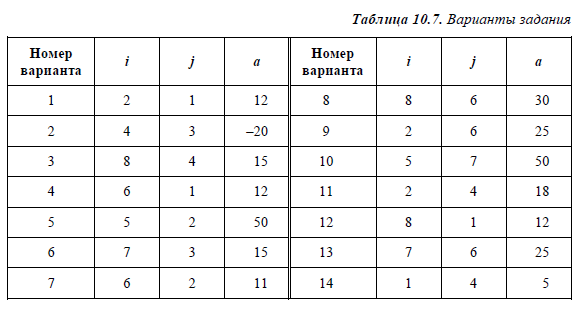
ж) записать в IR недопустимое значение аргумента *x* и выполнить пп. 5, *б*

и 5, *в*.

6. Для выбранного допустимого значения аргумента *x* наблюдать выполне-

ние отлаженной программы в режиме **Шаг** и записать в форме табл. 10.3

содержимое регистров ЭВМ перед выполнением каждой команды.

**********10.4.3. Содержание отчета**

Отчет о лабораторной работе должен содержать следующие разделы:

1. Формулировка варианта задания.

2. Граф-схема алгоритма решения задачи.

3. Размещение данных в ОЗУ.

4. Программа в форме табл. 9.5.

5. Последовательность состояний регистров ЭВМ при выполнении програм-

мы в режиме **Шаг** для одного значения аргумента.

6. Результаты выполнения программы для нескольких значений аргумента,

выбранных самостоятельно.

**10.4.4. Контрольные вопросы**

1. Как работает механизм косвенной адресации?

2. Какая ячейка будет адресована в команде с косвенной адресацией через

ячейку 043, если содержимое этой ячейки равно 102 347?

3. Как работают команды передачи управления?

4. Что входит в понятие "отладка программы"?

5. Какие способы отладки программы можно реализовать в модели?

**10.5. Лабораторная работа № 5.**

**Программирование цикла с переадресацией**

При решении задач, связанных с обработкой массивов, возникает необходи-

мость изменения исполнительного адреса при повторном выполнении неко-

торых команд. Эта задача может быть решена путем использования косвен-

ной адресации.

**10.5.1. Пример**

Разработать программу вычисления суммы элементов массива чисел *C*1 ,

*C*2 , ..., *Cn* . Исходными данными в этой задаче являются: *n —* количество

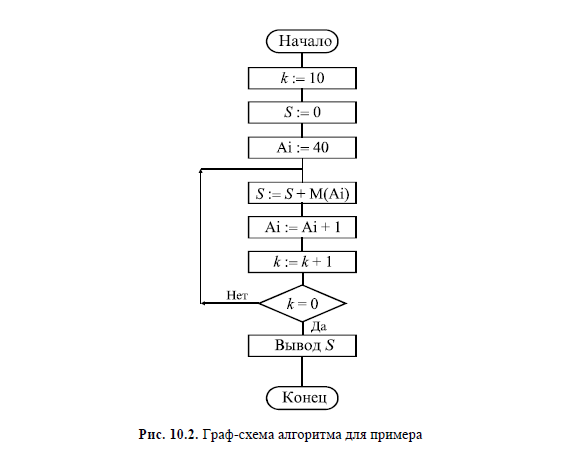
суммируемых чисел и *C*1 , *C*2 , ..., *Cn* — массив суммируемых чисел. Заме-

тим, что должно выполняться условие *n* 1, т. к. алгоритм предусматривает,

по крайней мере, одно суммирование. Кроме того, предполагается, что сум-

мируемые числа записаны в ОЗУ подряд, т. е. в ячейки памяти с последова-

тельными адресами. Результатом является сумма *S* .

**Составим программу для вычисления суммы со следующими конкретными

параметрами: число элементов массива — 10, элементы массива расположе-

ны в ячейках ОЗУ по адресам 040, 041, 042, ..., 049. Используемые для реше-

ния задачи промежуточные переменные имеют следующий смысл: *Ai* — ад-

рес числа *Ci* , *i*∈1, 2, ...,10; ОЗУ( *Ai* ) — число по адресу *Ai* , *S* — текущая

сумма; *k* — счетчик цикла, определяющий число повторений тела цикла.

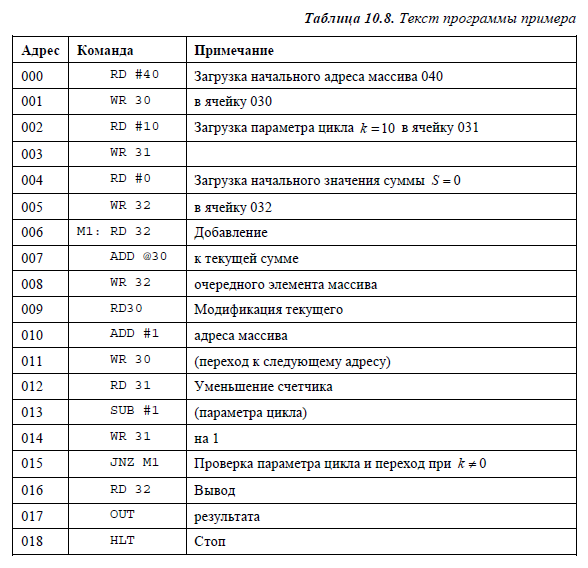
Распределение памяти таково. Программу разместим в ячейках ОЗУ, начиная

с адреса 000, примерная оценка объема программы — 20 команд; промежу-

точные переменные: *Ai* — в ячейке ОЗУ с адресом 030, *k —* по адресу 031,

*S —* по адресу 032. ГСА программы показана на рис. 10.2, текст программы

с комментариями приведен в табл. 10.8.

****10.5.2. Задание**

1. Написать программу определения заданной характеристики после-

довательности чисел *C*1 , *C*2 , ..., *Cn* . Варианты заданий приведены

в табл. 10.9.

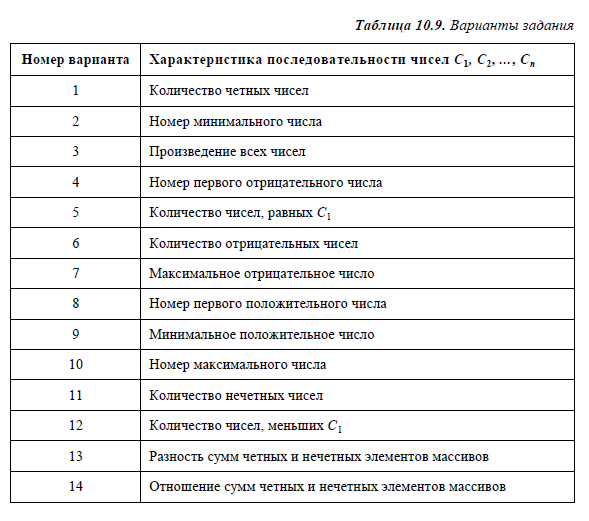
2. Записать программу в мнемокодах, введя ее в поле окна **Текст программы**.

3. Сохранить набранную программу в виде текстового файла и произвести

ассемблирование мнемокодов.

4. Загрузить в ОЗУ необходимые константы и исходные данные.

5. Отладить программу.

****Примечание.** Под четными (нечетными) элементами массивов понимаются элементы масси-

вов, имеющие четные (нечетные) индексы. Четные числа — элементы массивов, делящиеся

без остатка на 2.

*Часть III. Лабораторный 306 практикум и курсовое проектирование*

**10.5.3. Содержание отчета**

1. Формулировка варианта задания.

2. Граф-схема алгоритма решения задачи.

3. Распределение памяти (размещение в ОЗУ переменных, программы и не-

обходимых констант).

4. Программа.

5. Значения исходных данных и результата выполнения программы.

**10.5.4. Контрольные вопросы**

1. Как организовать цикл в программе?

2. Что такое параметр цикла?

3. Как поведет себя программа, приведенная в табл. 10.8, если в ней будет

отсутствовать команда WR 31 по адресу 014?

4. Как поведет себя программа, приведенная в табл. 10.8, если метка Ml бу-

дет поставлена по адресу 005? 007?

**10.6. Лабораторная работа № 6.**

**Подпрограммы и стек**

В программировании часто встречаются ситуации, когда одинаковые дейст-

вия необходимо выполнять многократно в разных частях программы (напри-

мер, вычисление функции sin *x* ). При этом с целью экономии памяти не сле-

дует многократно повторять одну и ту же последовательность команд —

достаточно один раз написать так называемую *подпрограмму* (в терминах

языков высокого уровня — процедуру) и обеспечить правильный вызов этой

подпрограммы и возврат в точку вызова по завершению подпрограммы.

Для *вызова* подпрограммы \_\_\_\_\_\_\_\_\_\_необходимо указать ее начальный адрес в памяти

и передать (если необходимо) параметры — те исходные данные, с которыми

будут выполняться предусмотренные в подпрограмме действия. Адрес под-

программы указывается в команде вызова CALL, а параметры могут переда-

ваться через определенные ячейки памяти, регистры или стек.

*Возврат* в точку вызова обеспечивается сохранением адреса текущей коман-

ды (содержимого регистра PC) при вызове и использованием в конце подпро-

граммы команды возврата RET, которая возвращает сохраненное значение

адреса возврата в PC.

Для реализации механизма вложенных подпрограмм (возможность вызова

подпрограммы из другой подпрограммы и т. д.) адреса возврата целесообраз-

но сохранять в стеке. *Стек* ("магазин") — особым образом организованная

безадресная память, доступ к которой осуществляется через единственную

ячейку, называемую *верхушкой стека*. При записи слово помещается в вер-

хушку стека, предварительно все находящиеся в нем слова смещаются вниз на

одну позицию; при чтении извлекается содержимое верхушки стека (оно при

этом из стека исчезает), а все оставшиеся слова смещаются вверх на одну пози-

цию. Такой механизм напоминает действие магазина стрелкового оружия (от-

сюда и второе название). В программировании называют такую дисциплину

обслуживания LIFO (Last In First Out, последним пришел — первым вышел)

в отличие от дисциплины типа *очередь* — FIFO (First In First Out, первым

пришел — первым вышел).

В обычных ОЗУ нет возможности перемещать слова между ячейками, поэто-

му при организации стека перемещается не массив слов относительно непод-

вижной верхушки, а верхушка относительно неподвижного массива. Под

стек отводится некоторая область ОЗУ, причем адрес верхушки хранится

в специальном регистре процессора — указателе стека SP.

В стек можно поместить содержимое регистра общего назначения по коман-

де PUSH или извлечь содержимое верхушки в регистр общего назначения по

команде POP. Кроме того, по команде вызова подпрограммы CALL значение

программного счетчика PC (адрес следующей команды) помещается в вер-

хушку стека, а по команде RET содержимое верхушки стека извлекается в PC.

При каждом обращении в стек указатель SP автоматически модифицируется.

В большинстве ЭВМ стек "растет" в сторону меньших адресов, поэтому пе-

ред каждой записью содержимое SP уменьшается на 1, а после каждого из-

влечения содержимое SP увеличивается на 1. Таким образом, SP всегда ука-

зывает на верхушку стека.

Цель настоящей лабораторной работы — изучение организации программ

с использованием подпрограмм. Кроме того, в процессе организации циклов

мы будем использовать новые возможности системы команд модели ЭВМ,

которые позволяют работать с новым классом памяти — сверхоперативной

(регистры общего назначения — РОН). В реальных ЭВМ доступ в РОН за-

нимает значительно меньшее время, чем в ОЗУ; кроме того, команды обра-

щения с регистрами короче команд обращения к памяти. Поэтому в РОН

размещаются наиболее часто используемые в программе данные, промежу-

точные результаты, счетчики циклов, косвенные адреса и т. п.

В системе команд учебной ЭВМ для работы с РОН используются специаль-

ные команды, мнемоники которых совпадают с мнемониками соответствую-

щих команд для работы с ОЗУ, но в адресной части содержат символы реги-

стров R0—R9.

Кроме обычных способов адресации (прямой и косвенной) в регистровых

командах используются два новых — постинкрементная и преддекрементная

(см. табл. 9.5). Кроме того, к регистровым относится команда организации

цикла JRNZ R,М. По этой команде содержимое указанного в команде регистра

уменьшается на 1, и если в результате вычитания содержимого регистра не

равно 0, то управление передается на метку М. Эту команду следует ставить

в конце тела цикла, метку М — в первой команде тела цикла, а в регистр R

помещать число повторений цикла.

**10.6.1. Пример**

Даны три массива чисел. Требуется вычислить среднее арифметическое их

максимальных элементов. Каждый массив задается двумя параметрами: ад-

ресом первого элемента и длиной.

Очевидно, в программе трижды необходимо выполнить поиск максимально-

го элемента массива, поэтому следует написать соответствующую подпро-

грамму.

Параметры в подпрограмму будем передавать через регистры: R1 — началь-

ный адрес массива, R2 — длина массива.

Рассмотрим конкретную реализацию этой задачи. Пусть первый массив на-

чинается с адреса 085 и имеет длину 14 элементов, второй — 100 и 4, тре-

тий — 110 и 9. Программа будет состоять из основной части и подпрограммы.

Основная программа задает параметры подпрограмме, вызывает ее и сохра-

няет результаты работы подпрограммы в рабочих ячейках. Затем осуществ-

ляет вычисление среднего арифметического и выводит результат на устрой-

ство вывода. В качестве рабочих ячеек используются регистры общего

назначения R6 и R7 — для хранения максимальных элементов массивов.

Подпрограмма получает параметры через регистры R1 (начальный адрес

массива) и R2 (длина массива). Эти регистры используются подпрограммой

в качестве регистра текущего адреса и счетчика цикла соответственно. Кроме

того, R3 используется для хранения текущего максимума, а R4 — для вре-

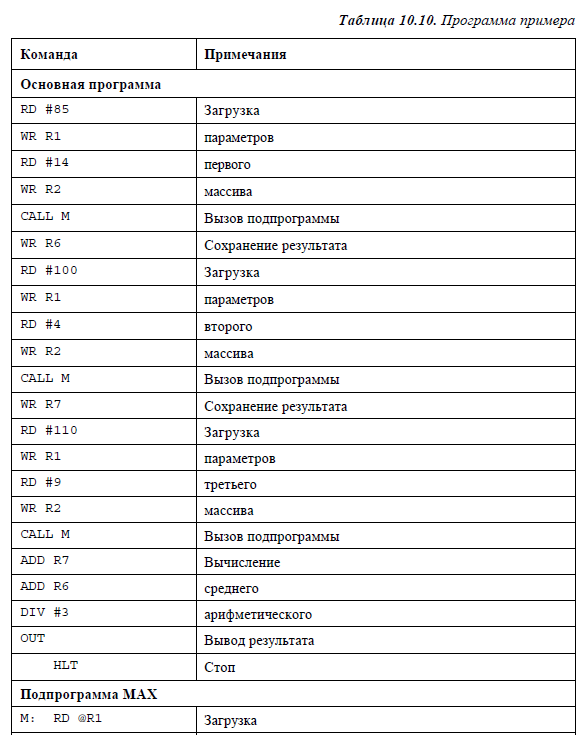
менного хранения текущего элемента. Подпрограмма возвращает результат

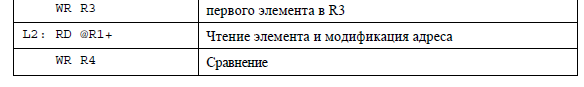
через аккумулятор. В табл. 10.10 приведен текст основной программы и под-

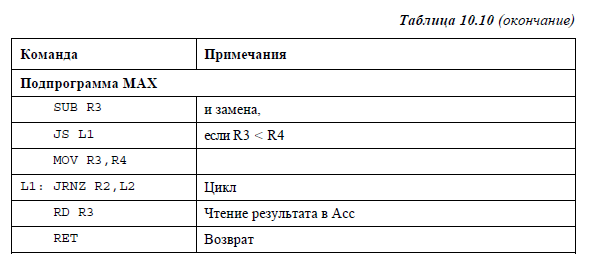
программы. Обратите внимание, цикл в подпрограмме организован с помо-

щью команды JRNZ, а модификация текущего адреса — средствами пост-

инкрементной адресации.

******

**

****10.6.2. Задание**

Составить и отладить программу учебной ЭВМ для решения следующей за-

дачи. Три массива в памяти заданы начальными адресами и длинами. Вычис-

лить и вывести на устройство вывода среднее арифметическое параметров

этих массивов. Параметры определяются заданием к предыдущей лаборатор-

ной работе (см. табл. 10.9), причем соответствие между номерами вариантов

заданий к лабораторным работам № 5 и 6 устанавливается по табл. 10.11.

****10.6.3. Содержание отчета**

1. Формулировка варианта задания.

2. Граф-схема алгоритма основной программы.

3. Граф-схема алгоритма подпрограммы.

4. Распределение памяти (размещение в ОЗУ переменных, программы и не-

обходимых констант).

5. Тексты программы и подпрограммы.

6. Значения исходных данных и результата выполнения программы.

**10.6.4. Контрольные вопросы**

1. Как работает команда MOV R3, R7?

2. Какие действия выполняет процессор при реализации команды CALL?

3. Как поведет себя программа примера из *разд. 10.6.1*, если в ней вместо

команд CALL M использовать команды JMP M?

4. После начальной установки процессора (сигнал **Сброс**) указатель стека SP

устанавливается в 000. По какому адресу будет производиться запись

в стек первый раз, если не загружать SP командой WRSP?

5. Как, используя механизмы постинкрементной и преддекрементной адре-

сации, организовать дополнительный стек в произвольной области памя-

ти, не связанный с SP?

**10.7. Лабораторная работа № 7.**

**Командный цикл процессора**

Реализация программы в ЭВМ сводится к последовательному выполнению

команд. Каждая команда, в свою очередь, выполняется как последователь-

ность микрокоманд, реализующих элементарные действия над операцион-

ными элементами процессора.

В программной модели учебной ЭВМ предусмотрен **Режим микрокоманд**,

в котором действие командного цикла реализуется и отображается на уровне

микрокоманд. Список микрокоманд текущей команды выводится в специ-

альном окне **Микрокомандный уровень** (см. рис. 9.8).

**10.7.1. Задание 1**

Выполнить снова последовательность команд по варианту задания из

*разд. 10.3.3* (см. табл. 10.4), но в режиме **Шаг**. Зарегистрировать изменения

состояния процессора и памяти в форме табл. 10.12, в которой приведены

состояния ЭВМ при выполнении примера из *разд. 10.3.2* (фрагмент).

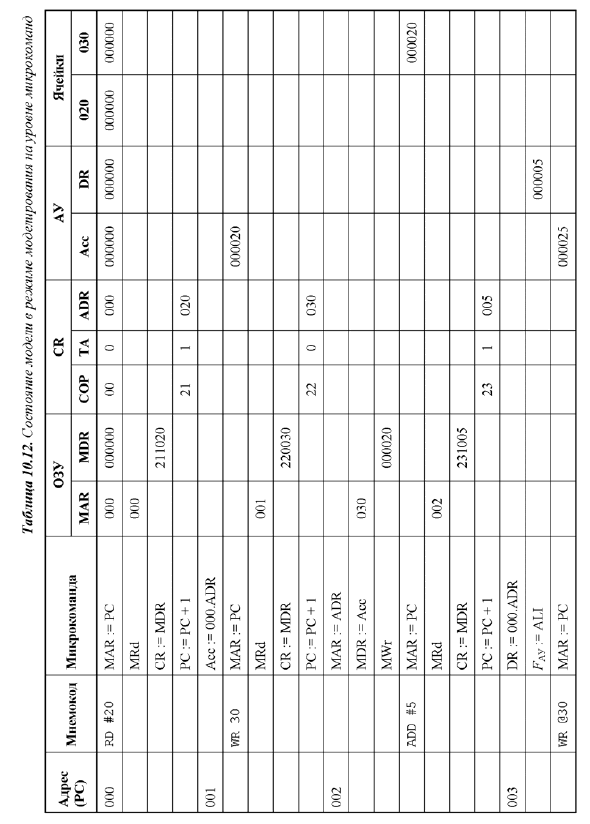
**10.7.2. Задание 2**

Записать последовательность микрокоманд для следующих команд модели

учебной ЭВМ:

􀂈ADD R3

􀂈ADD @R3

**

􀂈ADD @R3+

􀂈ADD -@R3

􀂈JRNZ R3,M

􀂈MOV R4,R2

􀂈JMP M

􀂈CALL M

􀂈RET: PUSH R3

􀂈POP R5

**10.7.3. Контрольные вопросы**

1. Какие микрокоманды связаны с изменением состояния аккумулятора?

2. Какие действия выполняются в модели по микрокоманде MRd? RWr?

3. Попробуйте составить микропрограмму (последовательность микроко-

манд, реализующих команду) для не существующей в данной модели ко-

манды "умножение модулей чисел".

4. Что изменится в работе процессора, если в каждой микропрограмме мик-

рокоманду увеличения программного счетчика PC := РС + 1 переместить

в самый конец микропрограммы?

**10.8. Лабораторная работа № 8.**

**Программирование внешних устройств**

Целью этой лабораторной работы является изучение способов организации

взаимодействия процессора и внешних устройств (ВУ) в составе ЭВМ.

Ранее отмечалось, что связь процессора и ВУ может осуществляться в син-

хронном или асинхронном режиме. *Синхронный режим* используется для

ВУ, всегда готовых к обмену. В нашей модели такими ВУ являются дисплей

и тоногенератор — процессор может обращаться к этим ВУ, не анализируя

их состояние (правда дисплей блокирует прием данных после ввода 128 сим-

волов, формируя флаг ошибки).

*Асинхронный обмен* предполагает анализ процессором состояния ВУ, кото-

рое определяет готовность ВУ выдать или принять данные или факт осуще-

ствления некоторого события, контролируемого системой. К таким устройст-

вам в нашей модели можно отнести клавиатуру и блок таймеров.

Анализ состояния ВУ может осуществляться процессором двумя способами:

􀂈в программно-управляемом режиме;

􀂈в режиме прерывания.

В первом случае предполагается программное обращение процессора к реги-

стру состояния ВУ с последующим анализом значения соответствующего

разряда слова состояния. Такое обращение следует предусмотреть в про-

грамме с некоторой периодичностью, независимо от фактического наступле-

ния контролируемого события (например, нажатие клавиши).

Во втором случае при возникновении контролируемого события ВУ форми-

рует процессору запрос на прерывание программы, по которому процессор и

осуществляет связь с ВУ.

**10.8.1. Задание**

Свой вариант задания (табл. 10.13) требуется выполнить двумя способами —

сначала в режиме программного контроля, далее модифицировать программу

таким образом, чтобы события обрабатывались в режиме прерывания про-

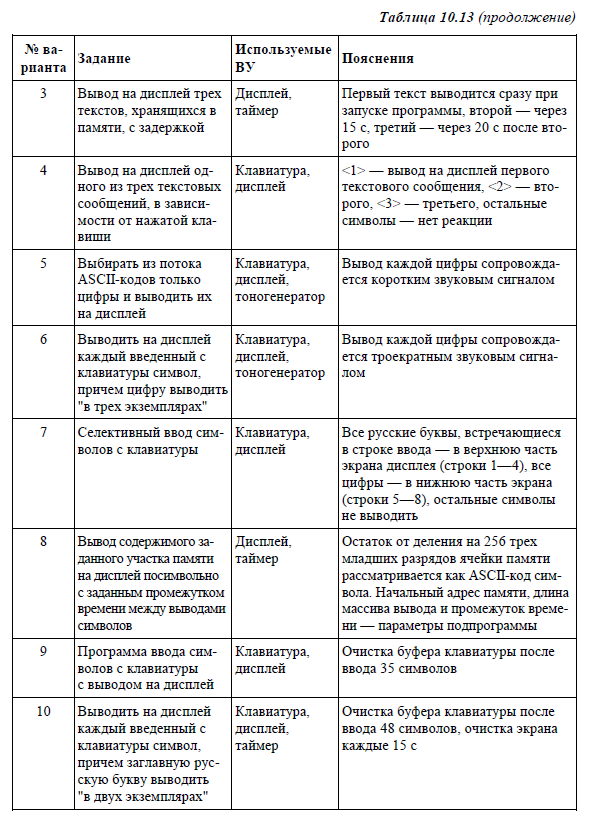
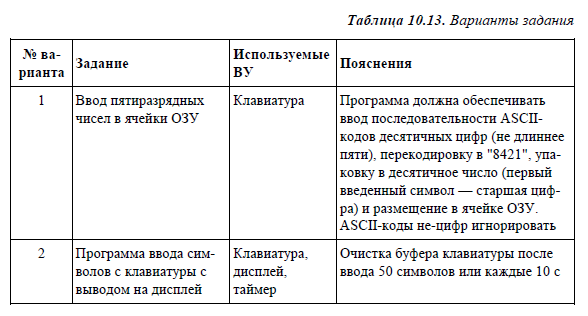
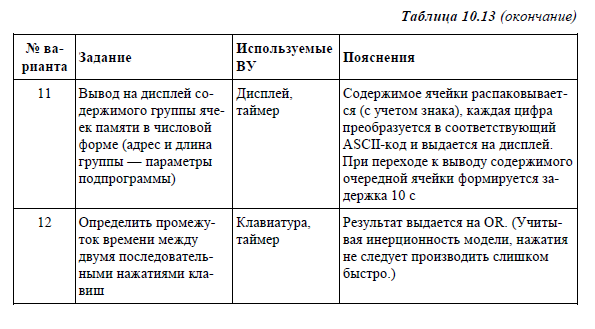
граммы. Поскольку "фоновая" (основная) задача для этого случая в заданиях

отсутствует, роль ее может сыграть "пустой цикл":

M: NOP

NOP

JMP M

********

**10.8.2. Задания повышенной сложности**

1. Разработать программу-тест на скорость ввода символов с клавиатуры. По

звуковому сигналу включается клавиатура и таймер на *T* секунд. Можно

начинать ввод символов, причем каждый символ отображается на дис-

плее, ведется подсчет количества введенных символов (после каждых 50

дается команда на очистку буфера клавиатуры, после 128 — очищается

дисплей). Переполнение таймера выключает клавиатуру и включает сиг-

нал завершения ввода (можно тон этого сигнала сопоставить с количест-

вом введенных символов). Параметр *T* вводится из IR. Результат *S* —

средняя скорость ввода (символ/с) выдается на OR. Учитывая, что модель

учебной ЭВМ оперирует только целыми числами, можно выдавать ре-

зультат в формате *S* 60 символов/мин.

2. Разработать программу-тест на степень запоминания текста. Три различ-

ных варианта текста выводятся последовательно на дисплей на *T*1 секунд

с промежутками *T*2 секунд. Далее эти тексты (то, что запомнилось) вво-

дятся с клавиатуры (в режиме ввода строки) и программно сравниваются

с исходными текстами. Выдается количество (процент) ошибок.

3. Разработать программу-калькулятор. Осуществлять ввод из буфера кла-

виатуры последовательности цифр, упаковку (см. вариант 1 в табл. 10.13).

Разделители — знаки бинарных арифметических операций и =. Результат

переводится в ASCII-коды и выводится на дисплей.

**10.8.3. Порядок выполнения работы**

1. Запустить программную модель учебной ЭВМ и подключить к ней опре-

деленные в задании внешние устройства (меню **Внешние устройства |**

**Менеджер ВУ**).

2. Написать и отладить программу, предусмотренную заданием, с использо-

ванием программного анализа флагов готовности ВУ. Продемонстриро-

вать работающую программу преподавателю.

3. Изменить отлаженную в п. 2 программу таким образом, чтобы процессор

реагировал на готовность ВУ с помощью подсистемы прерывания. Про-

демонстрировать работу измененной программы преподавателю.

**10.8.4. Содержание отчета**

1. Текст программы с программным анализом флагов готовности ВУ.

2. Текст программы с обработчиком прерывания.

**10.8.5. Контрольные вопросы**

1. При каких условиях устанавливается и сбрасывается флаг готовности кла-

виатуры Rd?

2. Возможно ли в блоке таймеров организовать работу всех трех таймеров

с разной тактовой частотой?

3. Как при получении запроса на прерывание от блока таймеров определить

номер таймера, достигшего состояния 99 999 (00 000)?

4. Какой текст окажется на экране дисплея, если после нажатия в окне обо-

зревателя дисплея кнопки **Очистить** и загрузки по адресу CR (11) кон-

станты #10 вывести по адресу DR (10) последовательно пять ASCII-кодов

русских букв А, Б, В, Г, Д?

5. В какой области памяти модели ЭВМ могут располагаться\_\_программы —

обработчики прерываний?

6. Какие изменения в работе отлаженной вами второй программы произой-

дут, если завершить обработчик прерываний командой RET, а не IRET?

**10.9. Лабораторная работа № 9.**

**Принципы работы кэш-памяти**

В *разд. 8.8* описаны некоторые алгоритмы замещения строк кэш-памяти.

Цель настоящей лабораторной работы — проверить работу различных алго-

ритмов замещения при различных режимах записи.

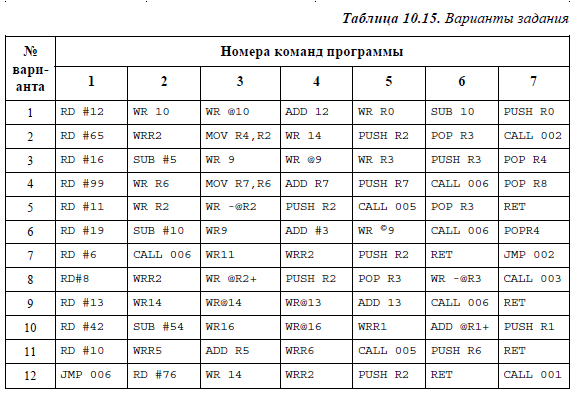
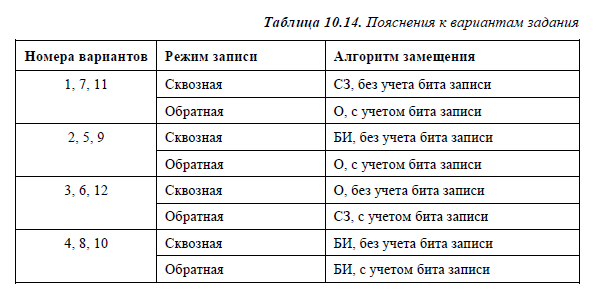
**10.9.1. Задание**

В качестве задания предлагается некоторая короткая "программа" (табл. 10.15),

которую необходимо выполнить с подключенной кэш-памятью (размером 4

и 8 ячеек) в шаговом режиме для следующих двух вариантов алгоритмов за-

мещения (табл. 10.14).

******

Не следует рассматривать заданную последовательность команд как фраг-

мент программы1. Некоторые конструкции, например, последовательность

команд PUSH R6, RET в общем случае не возвращает программу в точку вызо-

ва подпрограммы. Такие группы команд введены в задание для того, чтобы

обратить внимание студентов на особенности функционирования стека.

**10.9.2. Порядок выполнения работы**

1. Ввести в модель учебной ЭВМ текст своего варианта программы (см.

табл. 10.15), ассемблировать его и сохранить на диске в виде txt-файла.

2. Установить параметры кэш-памяти размером 4 ячейки, выбрать режим

записи и алгоритм замещения в соответствии с первой строкой своего ва-

рианта из табл. 10.14.

3. В шаговом режиме выполнить программу, фиксируя после каждого шага

состояние кэш-памяти.

4. Для одной из команд записи (WR) перейти в режим **Такт** и отметить,

в каких микрокомандах происходит изменение кэш-памяти.

5. Для кэш-памяти размером 8 ячеек установить параметры в соответствии

со второй строкой своего варианта из табл. 10.14 и выполнить программу

в шаговом режиме еще раз, фиксируя последовательность номеров заме-

щаемых ячеек кэш-памяти.

**10.9.3. Содержание отчета**

1. Вариант задания — текст программы и режимы кэш-памяти.

2. Последовательность состояний кэш-памяти размером 4 ячейки при одно-

кратном выполнении программы (команды 1—7).

3. Последовательность микрокоманд при выполнении команды WR с отмет-

кой тех микрокоманд, в которых возможна модификация кэш-памяти.

4. Для варианта кэш-памяти размером 8 ячеек — последовательность номе-

ров замещаемых ячеек кэш-памяти для второго варианта параметров кэш-

памяти при двукратном выполнении программы (команды 1—7).

**10.9.4. Контрольные вопросы**

1. В чем смысл включения кэш-памяти в состав ЭВМ?

2. Как работает кэш-память в режиме обратной записи? Сквозной записи?

1 Напомним, что программа определяется как последовательность команд, выполне-

ние которых позволит получить некий *результат*.

3. Как зависит эффективность работы ЭВМ от размера кэш-памяти?

4. В какую ячейку кэш-памяти будет помещаться очередное слово, если сво-

бодные ячейки отсутствуют?

5. Какие алгоритмы замещения ячеек кэш-памяти вам известны?

**10.10. Лабораторная работа № 10.**

**Алгоритмы замещения строк кэш-памяти**

Цель работы — изучение влияния параметров кэш-памяти и выбранного ал-

горитма замещения на эффективность работы системы. Эффективность

в данном случае оценивается числом кэш-попаданий по отношению к обще-

му числу обращений к памяти. Учитывая разницу в алгоритмах в режимах

*сквозной* и *обратной записи*, эффективность использования кэш-памяти вы-

числяется выражениям (9.2) и (9.3) соответственно для сквозной и обратной

записи.

Очевидно, эффективность работы системы с кэш-памятью будет зависеть не

только от параметров кэш-памяти и выбранного алгоритма замещения, но

и от класса решаемой задачи. Так, линейные программы должны хорошо ра-

ботать с алгоритмами замещения типа *очередь*, а программы с большим чис-

лом условных переходов, зависящих от случайных входных данных, могут

давать неплохие результаты с алгоритмами *случайного замещения*. Можно

предположить, что программы, имеющие большое число повторяющихся

участков (часто вызываемых подпрограмм и/или циклов) при прочих равных

условиях обеспечат более высокую эффективность применения кэш-памяти,

чем линейные программы. И, разумеется, на эффективность напрямую дол-

жен влиять размер кэш-памяти.

Для проверки высказанных выше предположений выполняется настоящая

лабораторная работа.

**10.10.1. Задание**

В данной лабораторной работе все варианты задания одинаковы: исследо-

вать эффективность работы кэш-памяти при выполнении двух разнотипных

программ, написанных и отлаженных вами при выполнении лабораторных

работ № 4 и 6.

**10.10.2. Порядок выполнения работы**

1. Загрузить в модель учебной ЭВМ отлаженную программу из лаборатор-

ной работы № 4.

2. В меню **Работа** установить режим **Кэш**-**память**.

3. В меню **Вид** выбрать команду **Кэш-память**, открыв тем самым окно **Кэш**-

**память**, в нем нажать первую слева кнопку на панели инструментов, от-

крыв диалоговое окно **Параметры кэш-памяти**, и установить следующие

параметры кэш-памяти: размер — 4, режим записи — сквозная, алгоритм

замещения — случайное, без учета бита записи (W).

4. Запустить программу в автоматическом режиме; по окончании работы

просмотреть результаты работы кэш-памяти в окне **Кэш**-**память**, вычис-

лить значение коэффициента эффективности *K* и записать в ячейку

табл. 10.16, помеченную звездочкой.

5. Выключить кэш-память модели (**Работа | Кэш-память**) и изменить один

из ее параметров — установить флаг с учетом бита записи (в окне **Пара-**

**метры кэш-памяти**).

6. Повторить п. 4, поместив значение полученного коэффициента эффек-

тивности в следующую справа ячейку табл. 10.16.

7. Последовательно меняя параметры кэш-памяти, повторить пп. 3—5, за-

полняя все ячейки табл. 10.16.\_\_

***Совет***

При очередном запуске программы не забывайте устанавливать процессор

модели в начальное состояние, нажимая кнопку **R** в окне **Процессор**!

8. Повторить все действия, описанные в пп. 1—7 для программы из лабора-

торной работы № 6, заполняя вторую таблицу по форме табл. 10.16.

**10.10.3. Содержание отчета**

1. Две таблицы по форме табл. 10.16 с результатами моделирования про-

грамм из лабораторных работ № 4 и 6 при разных режимах работы кэш-

памяти.

2. Выводы, объясняющие полученные результаты.

**10.10.4. Контрольные вопросы**

1. Как работает алгоритм замещения *очередь* при установленном флажке

**С учетом бита записи** в диалоговом окне **Параметры кэш-памяти**?

2. Какой алгоритм замещения будет наиболее эффективным в случае приме-

нения кэш-памяти большого объема (в кэш-память целиком помещается

программа)?

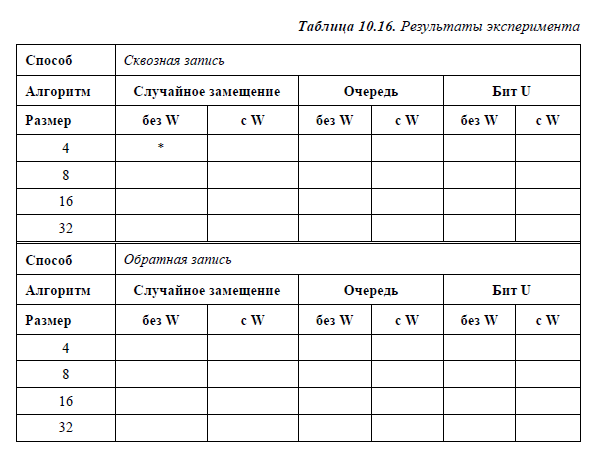
3. Как скажется на эффективности алгоритмов замещения учет значения би-

та записи W при работе кэш-памяти в режиме обратной записи? Сквозной

записи?

4. Для каких целей в структуру ячейки кэш-памяти включен бит использова-

ния? Как устанавливается и сбрасывается этот бит?

******