

Лабораторный практикум построен в расчете на изучение взаимодействия устройств в структуре ЭВМ с помощью программной модели некоторой абстрактной учебной ЭВМ, которая программируется на языке *ассемблера*.

Часто путь современного программиста начинается со знакомства с языком (языками) высокого уровня и все его общение с компьютером проходит с использованием таких языков.

Во многих случаях знание операторов языка высокого уровня, структуры данных и способов их обработки является достаточным для создания различных полезных приложений. Однако по-настоящему решать проблемы, связанные с управлением различной, особенно нестандартной, аппаратурой (программирование "по железу") невозможно без знания ассемблера [14]. Не случайно практически все компиляторы языков высокого уровня содержат средства связи своих модулей с модулями на ассемблере либо поддерживают выход на ассемблерный уровень программирования.

Однако проводить начальное обучение программированию на низком уровне с рассмотрением механизмов взаимодействия устройств на реальном языке, например x86 на персональной ЭВМ, не всегда удобно. В этом случае между пользователем и аппаратурой ЭВМ присутствует операционная система (ОС), которая существенно ограничивает желания пользователя экспериментировать с аппаратными средствами. Для преодоления этих ограничений необходимо обладать глубокими знаниями как ОС, так и аппаратных средств ЭВМ.

Предлагаемая для использования программная модель учебной ЭВМ отражает все основные особенности систем команд и структур современных простых ЭВМ, включает в себя, помимо процессора и памяти, модели нескольких типичных внешних устройств. Модель позволяет изучить основы программирования на низком уровне, вопросы взаимодействия различных уровней памяти в составе ЭВМ и способы взаимодействия процессора с внешними устройствами.

<http://educomp.runnet.ru/model/>

ГЛАВА 8

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 ЭВМ¹, 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.

8.1. 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 (СК, РС, 5Р и др.). Структурная схема ЭВМ показана на рис. 8.1.

В ячейках ОЗУ хранятся команды и данные. Емкость ОЗУ составляет 1000 ячеек. По сигналу MWt 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, содержащимся 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 (COP), над содержимым аккумулятора (Acc) и регистра операнда (DR). Результат операции всегда помещается в Acc. При завершении 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 (Acc := IR) и OUT (OR := Acc). 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

¹ Программная модель учебной ЭВМ находится на компакт-диске, прилагаемом к книге.

видимых элементов). Подробнее эти внешние устройства описаны в *разд. 8.6*.

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;
- СК — регистр команды, содержащий код команды;
- RB — регистр базового адреса, содержащий базовый адрес;
- SP — указатель стека, содержащий адрес вершины стека;
- RA — регистр адреса, содержащий исполнительный адрес при косвенной адресации.

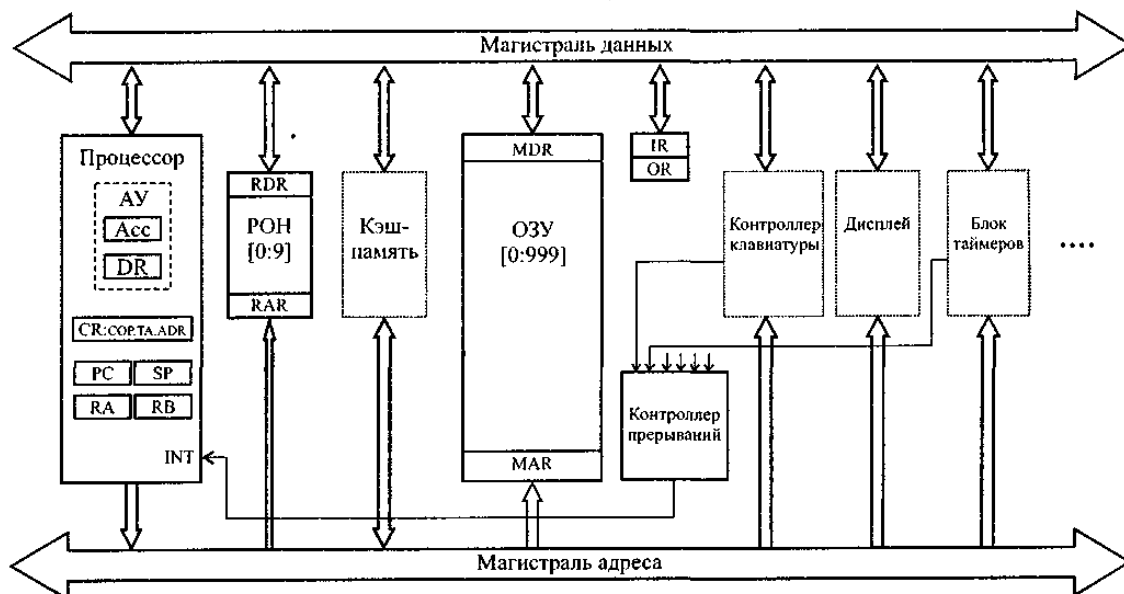


Рис. 8.1. Общая структура учебной ЭВМ

Регистры Акс, ДР, ИР, ОР, CR и все ячейки ОЗУ и РОН имеют длину 6 десятичных разрядов, регистры PC, SP, RA и RB — 3 разряда.

8.2. Представление данных в модели

Данные в ЭВМ представляются в формате, показанном на рис. 8.2. Это целые

десятичные числа, изменяющиеся в диапазоне "-99 999... +99 999", содержащие знак и 5 десятичных цифр.

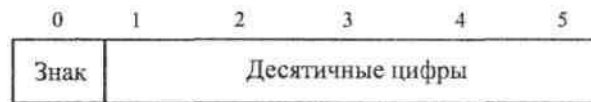


Рис. 8.2. Формат десятичных данных учебной ЭВМ

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.

8.3. Система команд

При рассмотрении системы команд ЭВМ обычно анализируют три аспекта: форматы, способы адресации и систему операций.

8.3.1. 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, являющаяся двухадресной.

В форматах команд выделяется три поля:

- ☐ два старших разряда [0:1] определяют код операции COP;
- ☐ разряд 2 может определять тип адресации (в одном случае (формат 5a) 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. 8.3, где приняты следующие обозначения:

- ☐ COP — код операции;
- ☐ ADR — адрес операнда в памяти;
- ☐ ADC — адрес перехода;
- ☐ I — непосредственный операнд;
- ☐ R, R1, R2 — номер регистра;
- ☐ TA — тип адресации;
- ☐ X — разряд не используется.

Номер формата	0	1	2	3	4	5		
1	COP	X	X	X	X			
2	COP	TA	ADR					
3	COP	TA	X	X	R			
3a	COP	TA	X	R1	R2		6	11
4	COP	X	X	X	X		I	
5	COP	X	ADC					
5a	COP*	R	ADC					

Рис. 8.3. Форматы команд учебной ЭВМ

8.3.2. 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. 8.1.

Таблица 8.1. Адресация в командах учебной ЭВМ

Код ТА	Тип адресации	Исполнительный адрес
0	Прямая (регистровая)	ADR (R)
1	Непосредственная	—
2	Косвенная	ОЗУ(ADR)[3:5]
3	Относительная	ADR + RB
4	Косвенно-регистровая	POH(R)[3:5]
5	Индексная с постинкрементом	POH(R)[3:5], R:= R + 1
6	Индексная с преддекрементом	R:= R – 1, POH(R)[3:5]

8.3.3. 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, стон.

Список команд учебной ЭВМ приведен в табл. 8.4 и 8.6.

8.4. 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 РС, 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 мнемосокодах, ассемблировать мнемосокоды, выполнять стандартные операции с файлами.

8.5. Интерфейс пользователя

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

Основное окно модели **Модель учебной ЭВМ** содержит основное меню и кнопки на панели управления. В рабочее поле окна выводятся сообщения о функционировании системы в целом. Эти сообщения группируются в файле 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 смотрите в разд. 8.6.

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 так же установить параметры файла logfile.txt, формируемого системой и записываемого на диск.

8.5.1. Окна основных обозревателей системы

Окно *Процессор*

Окно **Процессор** (рис. 8.4) обеспечивает доступ ко всем регистрам и флагам процессора.

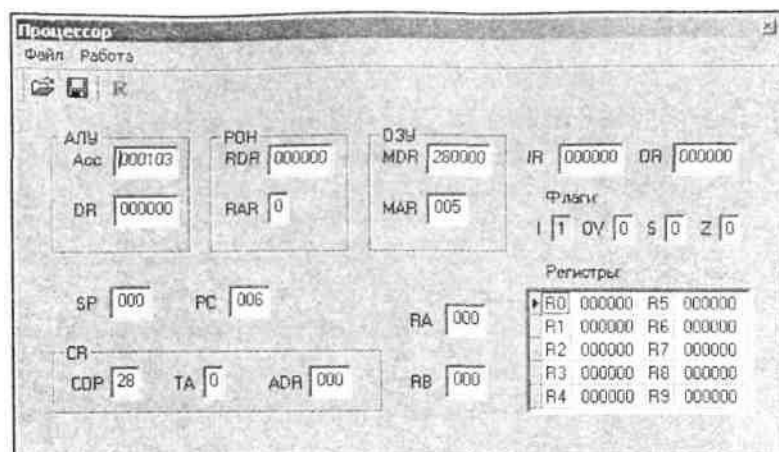


Рис. 8.4. Окно Процессор

- 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:
 - COP — код операции;
 - TA — 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**, **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

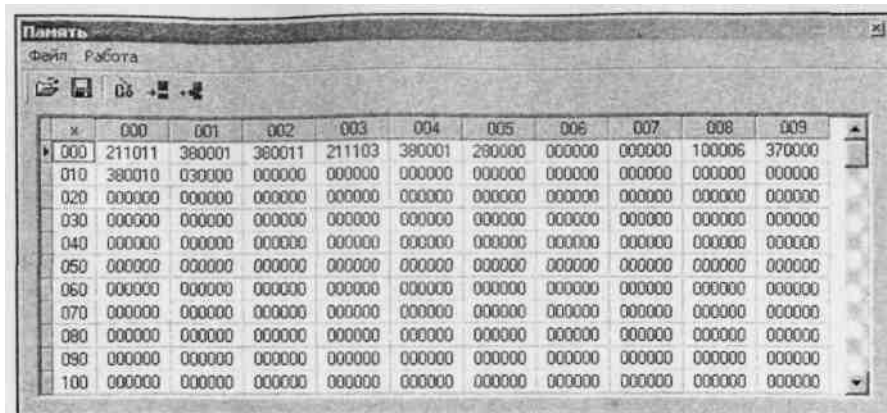
Окно **Память** (рис. 8.5) отражает текущее состояние ячеек ОЗУ. В этом окне допускается редактирование содержимого ячеек, кроме того, предусмотрена возможность выполнения (через меню или с помощью кнопок панели инструментов) пяти команд: **Сохранить**, **Загрузить**, **Перейти к**, **Вставить**, **Убрать**.

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 $[(n + 1): 999]$ перемещается в соседние ячейки с меньшими адресами. Освободившиеся ячейки с адресами 999, 998, ... заполняются нулями.



х	000	001	002	003	004	005	006	007	008	009
000	211011	380001	380011	211103	380001	280000	000000	000000	100006	370000
010	380010	030000	000000	000000	000000	000000	000000	000000	000000	000000
020	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
030	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
040	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
050	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
060	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
070	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
080	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
090	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000
100	000000	000000	000000	000000	000000	000000	000000	000000	000000	000000

Рис. 8.5. Окно Память

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

Окно Текст программы

Окно Текст программы (рис. 8.6) 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.

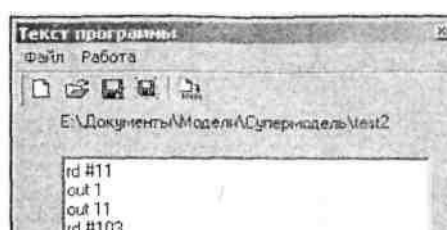


Рис. 8.6. Окно Текст программы

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. 8.7) 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 РС) и точка останова — красная заливка ячейки.

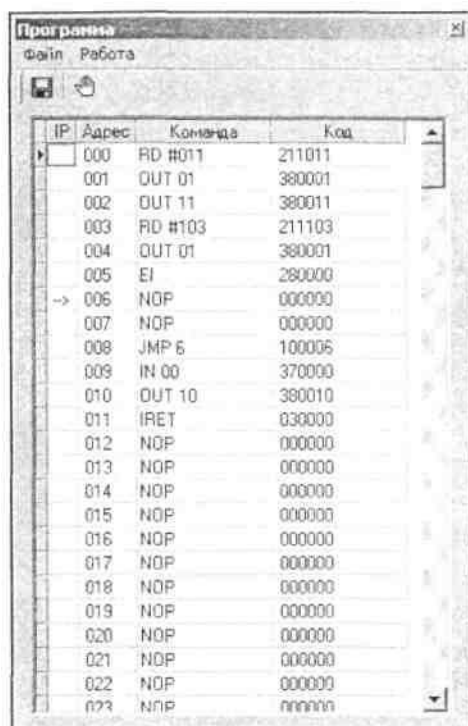


Рис. 8.7. Окно Программа

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 прочитав 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. 8.8) 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".

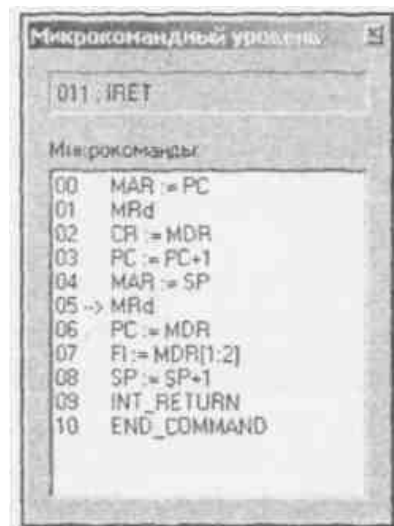


Рис. 8.8. Окно Микрокомандный уровень

Окно Кэш-память

Окно **Кэш-память** используется в режиме с подключенной кэш-памятью. Подробнее смотрите об этом режиме в разд. 8.8.

8.6. 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 пространстве 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`, `OUT 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 создаются 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. 8.2.

Таблица 8.2. Параметры внешних устройств

Внешнее устройство	Базовый адрес	Адреса регистров	Вектор прерывания
Контроллер клавиатуры	0	0, 1, 2	0
Дисплей	10	0, 1, 2, 3	Нет
Блок таймеров	20	0, 1, 2, 3, 4, 5, 6	2
Тоногенератор	30	0, 1	Нет

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 СК* и *состояния SR*, причем обычно регистры CR доступны только по записи, а SR — по чтению.

Регистр CR содержит флаги и поля, определяющие режимы работы ВУ, а 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.

Учебная модель ЭВМ: 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 пространстве 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**, **OUT aa**, где *aa* — двухразрядный десятичный адрес регистра ВУ. Таким образом, общий объем адресного пространства ввода/вывода составляет 100 адресов, причём 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 каждого ВУ начинается с 0. При включении ВУ в систему 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* — одноразрядное 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 таблице

Внешнее устройство	Базовый адрес	Адреса регистров	Вектор прерывания
Контроллер клавиатуры	0	0, 1, 2	0
Дисплей	10	0, 1, 2, 3	Нет
Блок таймеров	20	0, 1, 2, 3, 4, 5, 6	2
Тоногенератор	30	0, 1	Нет

При подключении устройства 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 процессору только по записи, а SR — по чтению.>

Регистр CR содержит флаги и поля, определяющие режимы работы ВУ, а 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.

В описываемой модели 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 (младший) разряд слова рассматривается как код команды для ВУ.

Для подключения ВУ в систему необходимо:

1. В главном окне *Модель учебной ЭВМ* выполнить команду *Внешние устройства/Менеджер ВУ* или нажать кнопку *МВУ*;
2. В открывшемся окне *Подключаемые устройства* выбрать из списка *Устройства* нужное ВУ, при желании изменить предлагаемые по умолчанию вектор прерывания *IRQ* и/или базовый адрес *Base_Adr* и нажать кнопку *Подключить* — при этом название устройства появится в списке *Подключённые устройства*;
3. Повторить п.2 для каждого из подключаемых устройств. Помните, что любое из ВУ в этой модели может быть подключено в систему только в одном экземпляре.
4. Закрыть окно *Подключаемые устройства*.

Если закрыть окно обозревателя ВУ, оно не будет отключено от системы. Для отключения ВУ следует открыть окно *Подключаемые устройства*, выделить его в списке *Подключённые устройства* и нажать кнопку *Отключить*.

8.6.1. Контроллер клавиатуры

Контроллер клавиатуры (рис. 8.9) представляет собой модель внешнего устройства, принимающего ASCII-коды¹ от клавиатуры ПЭВМ.

Символы помещаются последовательно в *буфер символов*, размер которого установлен равным 50 символам, и отображаются в окне обозревателя (рис. 8.10).

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:
 - E — флаг разрешения приема кодов в буфер;
 - I — флаг разрешения прерывания;
 - S — флаг режима посимвольного ввода.

¹Сокр. от American Standard Code for Information Interchange — американский стандартный код обмена информацией.

- SR (адрес 2) — регистр состояния, содержит два флага:
 - Err — флаг ошибки;
 - Rd — флаг готовности.

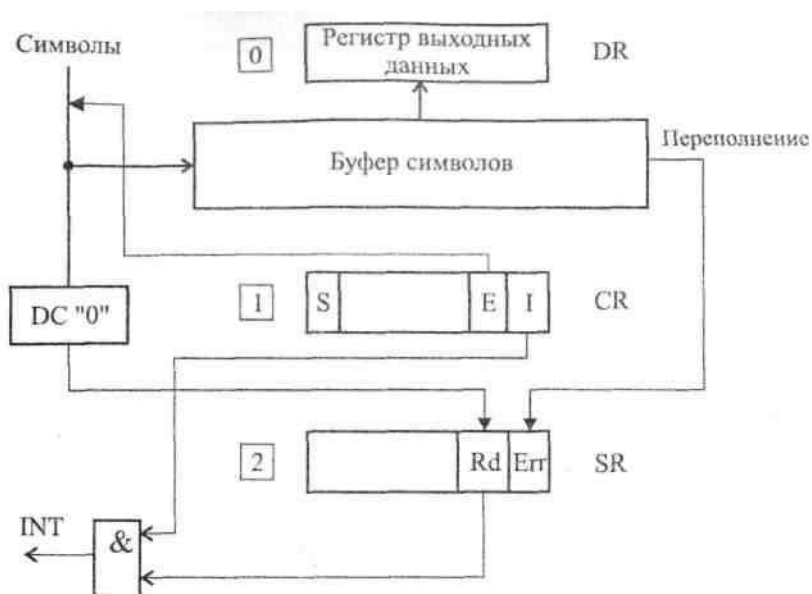


Рис. 8.9. Контроллер клавиатуры

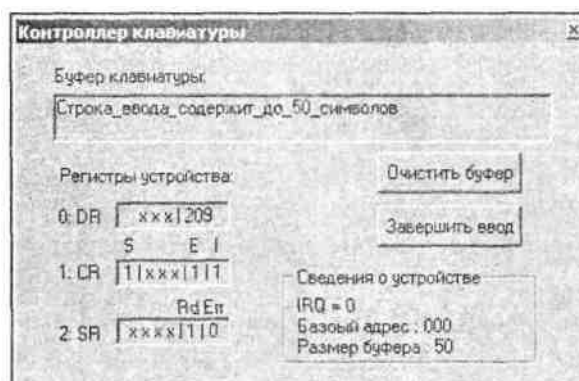


Рис. 8.10. Окно обозревателя контроллера клавиатуры

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 СК* устанавливаются и сбрасываются программно.

Флаг E, будучи установленным, разрешает прием кодов в буфер. При E = 0 контроллер игнорирует нажатие на клавиатуре, прием кодов в буфер не производится. На считывание кодов из буфера флаг E 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 т. н. *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. 8.10). 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:

- xxx101 — очистить буфер (действие команды эквивалентно нажатию кнопки **Очистить буфер**);
- xxx 102 — сбросить флаг Err в регистре SR;
- xxx 103 — установить флаг S в регистре CR;
- xxx 104 — сбросить флаг S в регистре CR.

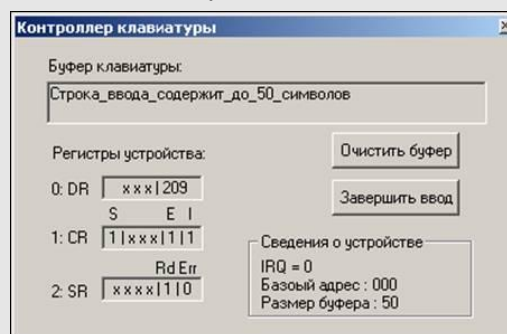
Если по адресу 1 произвести запись числа xxx0nn, то произойдет изменение 4-го и 5-го разрядов регистра CR по следующему правилу:

$$n = \begin{cases} 0 - \text{записать } 0; \\ 1 - \text{записать } 1; \\ 2, \dots, 9 - \text{сохранить разряд без изменения.} \end{cases} \quad (8.1)$$

Учебная модель ЭВМ: контроллер клавиатуры

Контроллер клавиатуры представляет собой модель внешнего устройства, принимающего ASCII-коды от клавиатуры ПЭВМ.

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



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 (устанавливаются и сбрасываются программно):
 - o E — флаг разрешения приема кодов в буфер, при E = 0 контроллер игнорирует нажатие на клавиатуре, прием кодов в буфер не производится. На считывание кодов из буфера флаг E влияния не оказывает;
 - o I — флаг разрешения прерывания, разрешает формирование запроса на прерывание от клавиатуры в момент установки флага готовности Rdy
 - o S — флаг режима посимвольного ввода. При S = 0 флаг готовности Rdy формируется только после нажатия кнопки *Завершить ввод* в окне обозревателя клавиатуры, при S = 1 — после каждого нажатия клавиши;
- SR (адрес 2) — регистр состояния, содержит два флага, устанавливаемые контроллером «аппаратно»:
 - o Err — флаг ошибки устанавливается при вводе в буфер 50-го символа, сбрасывается программно;
 - o Rdy — флаг готовности, устанавливается в зависимости от значения флага S, сбрасывается «аппаратно» после выполнения команды ввода из регистра DR — считывание символа из буфера.

Контроллер клавиатуры выполняет четыре команды при выводе соответствующих кодов по адресу 1:

Десятичное число	Команда
101	очистить буфер (эквивалентно нажатию кнопки <i>Очистить буфер</i>)
102	сбросить флаг Err в регистре SR
103	установить в «1» флаг S в регистре CR
104	сбросить в «0» флаг S в регистре CR

Пример программирования контроллера клавиатуры в режиме посимвольного ввода:

Метка	Команда	Примечание
	RD #10	; установить флаг E в регистре CR
	OUT 1	; - включить клавиатуру;
	OUT 11	; заодно включаем и <u>дисплей</u>
	RD #103	; передаём в контроллер код команды
	OUT 1	; Установить S в «1» (режим посимвольного ввода);
M1:	IN 2	; проверка нажатия – флага готовности Rdy
	JZ M1	; ожидание Rdy = 1;
	IN 0	; считывание введенного символа из буфера в аккумулятор;
	OUT 10	; передача ASCII-кода (например) на символьный дисплей;
	JMP M1	; возврат к ожиданию следующего нажатия.

Внимание! Контроллер клавиатуры с установленным битом E будет реагировать на нажатие клавиш только когда окно обозревателя *Контроллер клавиатуры* активно.

8.6.2. Дисплей

Дисплей (рис. 8.11) 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.

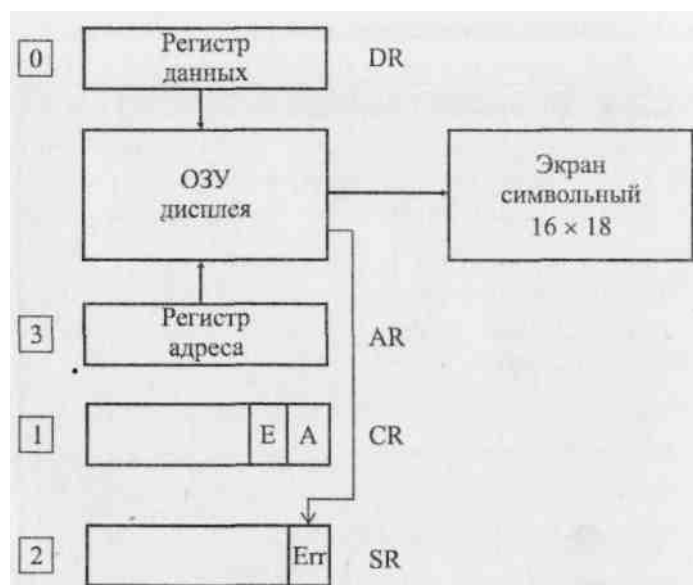


Рис. 8.11. Контроллер дисплея

Через *регистры адреса AR* и *данных DR* по записи и чтению осуществляется доступ к ячейкам видеопамати. При обращении к регистру DR по записи содержимое аккумулятора записывается в DR и в ячейку видеопамати, 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:

- E — флаг разрешения работы дисплея; при $E = 0$ запись в регистры AR и DR блокируется;
- A — флаг автоинкремента адреса; при $A = 1$ 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 $xxx0nn$, при этом изменение 4-го и 5-го разрядов регистра CR произойдет согласно выражению (8.1).

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:

- $xxx101$ — 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 Eгг и автоинкремента A;
- $xxx 102$ — сбросить флаг ошибки Eгг.

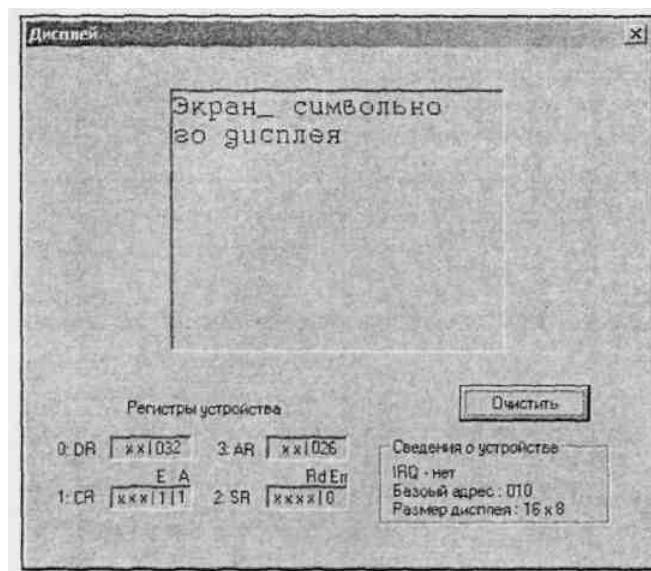
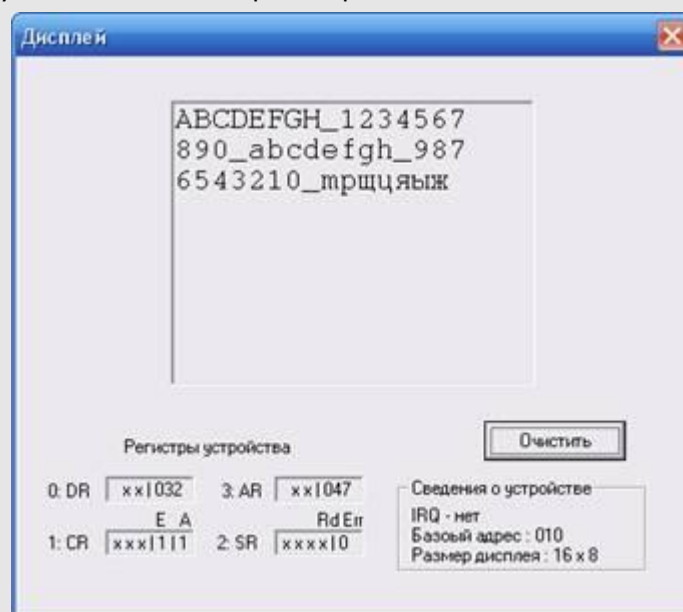


Рис. 8.12. Окно обозревателя контроллера дисплея

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. 8.12).

Учебная модель ЭВМ: контроллер дисплея

Дисплей представляет собой модель внешнего устройства, реализующую функции символического 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:
 - o DR (адрес 0) — регистр данных;
 - o CR (адрес 1) — регистр управления содержит следующие флаги (устанавливаются и сбрасываются программно):
 - E — флаг разрешения работы дисплея; при E = 0 запись в регистры AR и DR блокируется;
 - A — флаг автоинкремента адреса; при A = 1 содержимое AR автоматически увеличивается на 1 после любого обращения к регистру DR — по записи или чтению.
 - o SR (адрес 2) — регистр состояния содержит единственный флаг *Err* — ошибки устанавливается аппаратно при попытке записать в регистр адреса число, большее 127, сбрасывается программно;
 - o AR (адрес 3) — регистр адреса.

Доступ в видеопамять осуществляется через «окно интерфейса» необходимо сначала загрузить в регистр адреса AR номер ячейки видеопамати, тогда обращение к регистру данных будет означать обращение (ввод или вывод) к ячейке видеопамати с указанным адресом.

Контроллер дисплея выполняет две команды при выводе соответствующих кодов по адресу 1:

Десятичное число	Команда
101	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 A;
102	сбросить флаг ошибки Err в регистре SR.

Пример простой программы, выводящей на дисплей (в начало экрана) слово «Май»:

Метка	Команда	Примечание
	RD #11	; включаем дисплей и устанавливаем
	OUT 11	; флаг автоинкремента;
	RD #0	; задаём начальный адрес
	OUT 13	; выводимого слова;
	RD #204	; ввод кода буквы «М»
	OUT 10	; вывод на дисплей
	RD #224	; ввод кода буквы «а»
	OUT 10	; вывод на дисплей
	RD #233	; ввод кода буквы «й»
	OUT 10	; вывод на дисплей
	HLT	

Ещё один пример программы, выставляющей в начало каждой строки её номер:

Метка	Команда	Примечание
	RD #10	; включаем
	OUT 11	; дисплей;
	RD #0	; задаём начальный
	WR R1	; адрес вывода;
	RD #49	; вводим код
	WR R2	; цифры «1»;
	RD #8	; вводим число
	WR R3	; повторений цикла;
M1:	RD R1	; читаем текущий адрес
	OUT 13	; и передаём в регистр адреса дисплея;
	ADD #16	; увеличиваем адрес на 16 — на начало следующей строки;
	WR R1	; сохраняем изменённый адрес;
	RD R2	; читаем код цифры — номер строки;
	OUT 10	; передаём код цифры на дисплей;
	RD @R2+	; увеличиваем содержимое R2 (код цифры) на единицу;
	JRNZ R3, M1	; декремент R3 и переход на начало цикла, если R3 ≠ 0;
	HLT	

8.6.3. Блок таймеров

Блок таймеров (рис. 8.13) включает в себя три одноканальных таймера, каждый

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 Т1, Т2, Т3). Программа в любой момент может считать текущее содержимое таймера или записать в него новое значение.

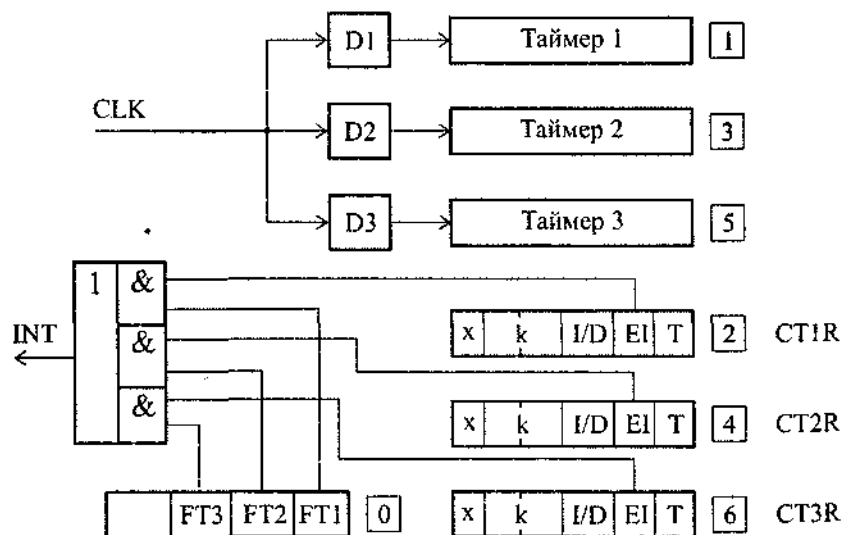


Рис. 8.13. Блок таймеров

На входы предделителей поступает 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 Т1, Т2, Т3 соответственно) устанавливаются в 1 при переполнении соответствующего 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. 8.14) 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.

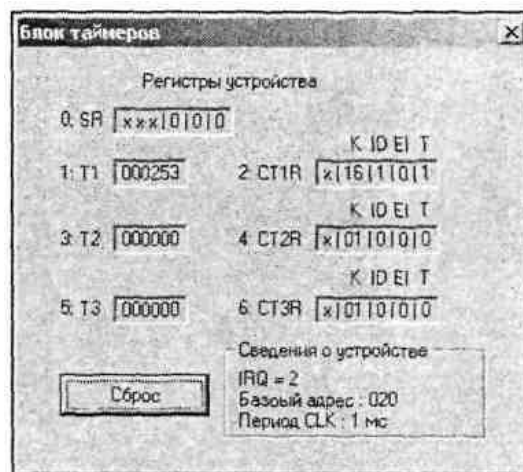


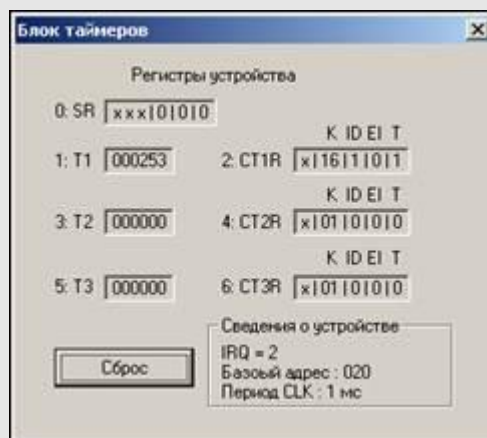
Рис. 8.14. Окно обозревателя блока таймеров

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:

- xxx100 — общий сброс (эквивалентна нажатию кнопки **Сброс** в окне обозревателя);
- xxx101 — сброс флага переполнения таймера FT1;
- xxx102 — сброс флага переполнения таймера FT2;
- xxx103 — сброс флага переполнения таймера FT3.

Учебная модель ЭВМ: блок таймеров

Блок таймеров включает три одноканальных таймеров *T1* (адрес 1), *T2* (адрес 3), *T3* (адрес 5) со своими регистрами управления *CT1R* (адрес 2), *CT2R* (адрес 4), *CT3R* (адрес 6) и общий регистр состояний *SR* (адрес 0).



Каждый таймер может независимо работать в режиме реверсивного счётчика меток времени. Следует отметить, что длительность такта полагалась (весьма условно!) равной 1 мс, но реально она значительно больше и зависит от машины, на которой работает модель. Например, на моей машине такт таймера равен примерно 0,016 с.

Каждый таймер снабжен предделителем с коэффициентом деления от 1 до 99, что позволяет выбирать диапазон измеряемых отрезков времени в широких пределах.

Таймеры могут работать как на сложение, так и на вычитание. Переход 99999 => 0 при сложении или 0 => 99999 при вычитании вызывают установку в «1» соответствующего флага переполнения в регистре SR:

- T1 – в пятом (младшем) разряде регистра SR,
- T2 – в четвёртом разряде,
- T3 – в третьем разряде

и формирование запроса на прерывание, если прерывание от соответствующего таймера разрешено. При этом счёт в таймере не прекращается.

Все шестиразрядные регистры управления таймерами имеют одинаковый формат:

- разряд 0 (старший) – не используется;
- разряды 1 и 2 задают коэффициент K деления предделителя от 01 до 99;
- разряд 3 определяет направление счёта: «0» – инкремент, «1» – декремент;
- разряд 4, будучи установленным в «1», разрешает формирование запроса на прерывание при переполнении этого таймера;
- разряд 5 включает («1») или выключает («0») счёт.

Сброс флагов переполнения таймеров в регистре SR осуществляется только программно по командам:

Десятичное число	Команда
101	сбросить флаг переполнения таймера T1
102	сбросить флаг переполнения таймера T2
103	сбросить флаг переполнения таймера T3

Константы кодов этих команд должны выводиться по адресу 0 блока таймеров (регистра SR).

Пример простой программы: подавать короткий звуковой сигнал каждые 10 сек.

Напомним, что такт таймера может изменяться в зависимости от ЭВМ, на которой работает модель. Если считать, что длительность такта составляет 16 мс, то для отсчёта отрезка времени в 10 сек. потребуется $10000 / 16 = 625$ тактов.

Метка	Команда	Примечание
M0:	RD #625 ; загрузка константы, соответствующей задержке в 10 сек. OUT 21 ; в регистр таймера T1; RDI 1101 ; запуск таймера с коэффициентом деления $K=1$ OUT 22 ; на вычитание, прерывание запрещено;	
M1:	IN 20 ; анализ регистра состояния блока таймеров JZ M1 ; и возврат, если флаг переполнения не установлен;	
	RDI 1000 ; останов OUT 22 ; таймера; RD #101 ; сброс OUT 20 ; флага переполнения FT1; RD #200 ; задаём частоту звучания 200 Гц OUT 30 ; в регистр частоты тоногенератора; RD #300 ; задаём длительность звучания – 300 мс OUT 31 ; и включает звук; JMP M0 ; повторяем бесконечный цикл.	

Внимание. Перед запуском этого примера следует подключить к системе не только блок таймеров, но и тоногенератор.

8.6.4. Тоногенератор

Тоногенератор предназначен для вывода тона одной заданной частоты и заданной длительности на встроенный динамик ПЭВМ.

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

- FR – регистр частоты (адрес 0), в который записывается частота тона звучания

в герцах;

- LR – регистр длительности звучания (адрес 1), в который задаётся время звучания в миллисекундах. Команда вывода числа по этому адресу одновременно с заданием длительности звучания является командой на начало звучания.

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.

По умолчанию базовый адрес тоногенератора – 30. Использование тоногенератора рассмотрено в примере на странице Блок таймеров.

8.7. Подсистема прерываний

В модели учебной ЭВМ предусмотрен механизм векторных внешних прерываний. Внешние устройства формируют запросы на прерывания, которые ступают на входы *контроллера прерываний*. При подключении ВУ, способного 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 т.н. *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 — 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.

36201120. Формирует и помещает в верхушку стека слово, три младших разряда ([3:5]) которого — текущее значение РС (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 $11_{10} = 1011_2$.

36201121. Сбрасывает в 0 флаг разрешения прерывания I.

36201122. Извлекает из таблицы векторов прерываний адрес обработчика, соответствующий обслуживаемому вектору, и помещает его в РС, 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 РС 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*).

Выше 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 восстанавливается, 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 непосредственно 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. 8.9, 8.13), 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 остальные разряды в 000).

Поэтому в системах команд многих ЭВМ, в т. ч. и нашей модели, имеют команды вызова прерываний — $INT\ n$ (в нашей модели $n \in \{0, 1, \dots, 9\}$), где n — вектор прерывания. Процессор, выполняя команду, производит 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.

8.8. Программная модель кэш-памяти

К описанной в разд. 8.1 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. 8.15.

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, $\forall Z_i = 0$), факт записи информации в ячейку во время пребывания ее в кэш-памяти, а также использование ячейки (т. е. любое обращение к ней).

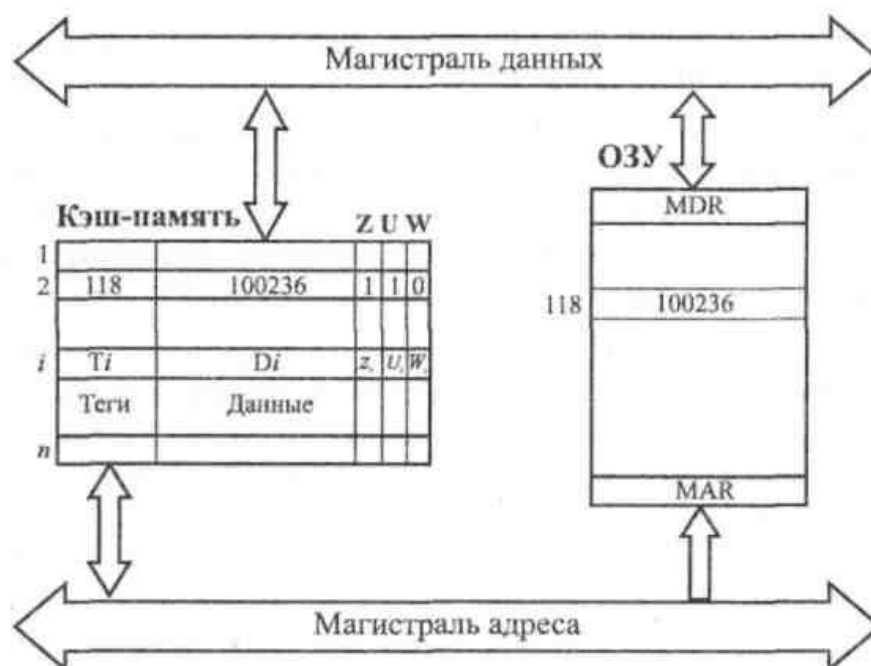


Рис. 8.15. Структура модели кэш-памяти

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, например, так, как показано в табл. 8.3.

	Теги	Данные	Z	U	W
1	012	220152	1	0	0
2	013	211003	1	1	0
3	050	000025	1	1	1
4	000	000000	0	0	0

Таблица 8.3. Пример текущего состояния кэш-памяти

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 ($W_i = 1$). При очистке ячеек, отмеченных битом записи, необходимо переписать измененное значение поля данных 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 U_i установятся в 1, все они тут сбрасываются в 0, так что в кэш всегда ячейки разбиты на два непересекающихся подмножества по значению бита U — те, обращение к которым стоялось 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 кэш-памяти. При снятом флаге "*с учетом бита записи*" поиск осуществляется 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):

$$K = \frac{S_K - S_{K_w}}{S_o}, \quad (8.2)$$

$$K = \frac{S_K - S_{K_w}^i}{S_o}, \quad (8.3)$$

где:

- K — коэффициент эффективности работы кэш-памяти;
- S_o — общее число обращений к памяти;
- S_K — число кэш-попаданий;
- S_{K_w} — число сквозных записей при кэш-попадании (в режиме сквозной записи);
- $S_{K_w}^i$ — число обратных записей (в режиме обратной записи).

8.9. Вспомогательные таблицы

В данном разделе представлены вспомогательные таблицы (табл. 8.4—8.8) для работы с моделью учебной ЭВМ.

Таблица 8.4. Таблица команд учебной ЭВМ

Мл. \ Ст.	0	1	2	3	4
0	NOP	JMP		MOV	.
1	IN	JZ	RD	RD	RDI
2	OUT	JNZ	WR	WR	
3	IRET	JS	ADD	ADD	ADI
4	WRRB	JNS	SUB	SUB	SBI
5	WRSP	JO	MUL	MUL	MULI
6	PUSH	JNO	DIV	DIV	DIVI
7	POP	JRNZ		IN	
8	RET	INT	EI	OUT	
9	HLT	CALL	DI		

Таблица 8.5. Типы адресации, их коды и обозначение

Обозначение	Код	Тип адресации	Пример команды
	0	Прямая (регистровая)	ADD 23 (ADD R3)
#	1	Непосредственная	ADD #33
@	2	Косвенная	ADD @33

Таблица 8.5 (окончание)

Обозначение	Код	Тип адресации	Пример команды
[]	3	Относительная	ADD [33]
@R	4	Косвенно-регистровая	ADD @R3
@R+	5	Индексная с постинкрементом	ADD @R3+
-@R	6	Индексная с преддекрементом	ADD -@R3

В табл. 8.6 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;
- $M(*)$ — ячейка памяти, прямо или косвенно адресуемая в команде;
- I — пятиразрядный непосредственный операнд со знаком.

Таблица 8.6. Система команд учебной ЭВМ

КОП	Мнемокод	Название	Действие
00	NOP	Пустая операция	Нет
01	IN	Ввод	$Acc \leftarrow IR$
02	OUT	Вывод	$OR \leftarrow Acc$
03	IRET	Возврат из прерывания	$FLAGS.PC \leftarrow M(SP); INC(SP)$
04	WRRB	Загрузка RB	$RB \leftarrow CR[ADR]$
05	WRSP	Загрузка SP	$SP \leftarrow CR[ADR]$
06	PUSH	Поместить в стек	$DEC(SP); M(SP) \leftarrow R$
07	POP	Извлечь из стека	$R \rightarrow M(SP); INC(SP)$
08	RET	Возврат	$PC \rightarrow M(SP); INC(SP)$
09	HLT	Стоп	Конец командных циклов
10	JMP	Безусловный переход	$PC \leftarrow CR[ADR]$
11	JZ	Переход, если 0	if $Acc = 0$ then $PC \leftarrow CR[ADR]$

КОП	Мнемокод	Название	Действие
12	JNZ	Переход, если не 0	if Acc \neq 0 then PC \leftarrow CR[ADR]
13	JS	Переход, если отрицательно	if Acc < 0 then PC \leftarrow CR[ADR]
14	JNS	Переход, если положительно	if Acc \geq 0 then PC \leftarrow CR[ADR]
15	JO	Переход, если переполнение	if Acc > 99999 then PC \leftarrow CR[ADR]
16	JNO	Переход, если нет переполнения	if Acc \leq 99999 then PC \leftarrow CR[ADR]
17	JRNZ	Цикл	DEC(R); if R > 0 then PC \leftarrow CR[ADR]
18	INT	Программное прерывание	DEC(SP); M(SP) \leftarrow FLAGS.PC; PC \leftarrow M(V)
19	CALL	Вызов подпрограммы	DEC(SP); M(SP) \leftarrow PC; PC \leftarrow CR(ADR)
20	Her		
21	RD	Чтение	Acc \leftarrow DD
22	WR	Запись	M(*) \leftarrow Acc
23	ADD	Сложение	Acc \leftarrow Acc + DD
24	SUB	Вычитание	Acc \leftarrow Acc - DD
25	MUL	Умножение	Acc \leftarrow Acc \times DD
26	DIV	Деление	Acc \leftarrow Acc/DD
27	Her		
28	EI	Разрешить прерывание	IF \leftarrow 1
29	DI	Запретить прерывание	IF \leftarrow 0
30	MOV	Пересылка	R1 \leftarrow R2
31	RD	Чтение	Acc \leftarrow R*
32	WR	Запись	R* \leftarrow Acc
33	ADD	Сложение	Acc \leftarrow Acc + R*
34	SUB	Вычитание	Acc \leftarrow Acc - R*
35	MUL	Умножение	Acc \leftarrow Acc \times R*
36	DIV	Деление	Acc \leftarrow Acc/R*
37	IN	Ввод	Acc \leftarrow BY(CR[ADR*])

Таблица 8.6. (продолжение)

Таблица 8.6 (окончание)

КОП	Мнемокод	Название	Действие
38	OUT	Вывод	$BY(CR[ADR*]) \leftarrow Acc$
39	Нет		
40	Нет		
41	RDI	Чтение	$Acc \leftarrow I$
42	Нет		
43	ADI	Сложение	$Acc \leftarrow Acc + I$
44	SBI	Вычитание	$Acc \leftarrow Acc - I$
45	MULI	Умножение	$Acc \leftarrow Acc \times I$
46	DIVI	Деление	$Acc \leftarrow Acc / I$

Таблица 8.7. Таблица кодов ASCII (фрагмент)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	.	p					A	P	a	p
1			!	1	A	Q	a	q					Б	С	б	с
2			”	2	B	R	b	r					В	Т	в	т
3			#	3	C	S	c	s					Г	У	г	у
4			\$	4	D	T	d	t					Д	Ф	д	ф
5			%	5	E	U	e	u					Е	Х	е	х
6			&	6	F	V	f	v					Ж	Ц	ж	ц
7			‘	7	G	W	g	w					З	Ч	з	ч
8			(8	H	X	h	x					И	Ш	и	ш
9)	9	I	Y	i	y					Й	Щ	й	щ
A			*	:	J	Z	j	z					К	Ъ	к	ъ
B			+	;	K	[k	{					Л	Ы	л	ы
C			·	<	L		l						М	Ь	м	ь
D			-	=	M]	m	}					Н	Э	н	э
E			·	>	N		n						Щ	Ю	ш	ю
F			/	?	O	_	o						П	Я	п	я

Таблица 8.8. Перевод HEX-кодов в десятичные числа

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
1	1	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241
2	2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242
3	3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243
4	4	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244
5	5	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245
6	6	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246
7	7	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247
8	8	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248
9	9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249
A	10	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250
B	11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251
C	12	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252
D	13	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253
E	14	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254
F	15	31	47	63	79	95	111	127	143	159	175	191	207	223	239	255

ГЛАВА 9

Лабораторные работы

Цикл лабораторных работ рассчитан на выполнение студентами в рамках курса "Архитектура ЭВМ" и других, подобных по содержанию.

Цикл включает работы различного уровня. Лабораторные работы № 1—4 ориентированы на первичное знакомство с архитектурой процессора, системой команд, способами адресации и основными приемами программирования на машинно-ориентированном языке. Лабораторная работа № 5 иллюстрирует реализацию командного цикла процессора на уровне микроопераций. Лабораторная работа № 6 посвящена способам организации связи процессора с внешними устройствами, а в лабораторных работах № 7 и 8 рассматривается организация кэш-памяти и эффективность различных алгоритмов замещения.

Все работы выполняются на программной модели учебной ЭВМ и взаимодействующих с ней в программных моделях ВУ и кэш-памяти, описанных в главе 8.

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

9.1. Лабораторная работа № 1.

Архитектура ЭВМ и система команд

9.1.1. Общие положения

Для решения с помощью ЭВМ некоторой задачи должна быть разработана программа. 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 1К. и ВК;
- ☐ установить в РС стартовый адрес программы;
- ☐ перевести модель в режим **Работа**.

Каждое из этих действий выполняется посредством интерфейса модели, описанного в *главе 8*. 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 программно-доступных объектов ЭВМ, происходящие при выполнении этих команд.

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

В настоящей лабораторной работе будем программировать ЭВМ в машинных кодах.

9.1.2. Пример 1

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. 9.1).

Таблица 9.1. Команды и коды

Последовательность	Значения				
Команды	RD#20	WR30	ADD #5	WR#30	JNZ 002
Коды	21 1 020	22 0 030	23 1 005	22 2 030	12 0 002

Введем полученные коды последовательно в ячейки ОЗУ, начиная с адреса 000. Выполняя команды в режиме **Шаг**, будем фиксировать изменения программно-доступных объектов (в данном случае это Асс, РС и ячейки ОЗУ 020 и 030) в табл. 9.2.

Таблица 9.2. Содержимое регистров

РС	Асс	М(30)	М(20)	РС	Асс	М(30)	М(20)
000	000000	000000	000000	004			000025
001	000020			002			
002		000020		003	000030		
003	000025			004			000030

9.1.3. Задание 1

1. Ознакомиться с архитектурой ЭВМ (см. часть I).
2. Записать в ОЗУ "программу", состоящую из пяти команд— варианты задания выбрать из табл. 9.3. 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. 9.2).
6. Если в программе образуется цикл, необходимо просмотреть не более двух повторений каждой команды, входящей в тело цикла.

Таблица 9.3. Варианты задания 1

№	IR	Команда 1	Команда 2	Команда 3	Команда 4	Команда 5
1	000007	IN	MUL #2	WR10	WR @10	JNS 001
2	X	RD #17	SUB #9	WR16	WR @16	JNS 001
3	100029	IN	ADD #16	WR8	WR@8	JS 001
4	X	RD #2	MUL #6	WR 11	WR @11	JNZ 00
5	000016	IN	WR8	DIV #4	WR @8	JMP 002
6	X	RD #4	WR 11	RD @11	ADD #330	JS 000

Таблица 9.3. (окончание)

№	IR	Команда 1	Команда 2	Команда 3	Команда 4	Команда 5
7	000000	IN	WR9	RD @9	SUB#1	JS 001
8	X	RD 4	SUB #8	WR8	WR @8	JNZ 001
9	100005	IN	ADD #12	WR 10	WR @10	JS 004
10	X	RD 4	ADD #15	WR 13	WR @13	JMP 001
11	000315	IN	SUB #308	WR11	WR @11	JMP 001
12	X	RD #988	ADD #19	WR9	WR @9	JNZ 001
13	000017	IN	WR11	ADD 11	WR @11	JMP 002
14	X	RD #5	MUL #9	WR10	WR @10	JNZ 001

9.1.4. Содержание отчета

1. Формулировка варианта задания.
2. Машинные коды команд, соответствующих варианту задания.
3. Результаты выполнения последовательности команд в форме табл. 9.2.

9.1.5. 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 содержимое некоторых регистров?
10. Как просмотреть и, при необходимости, отредактировать содержимое ячейки 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?

9.2. Лабораторная работа № 2.

Программирование разветвляющегося процесса

Для реализации алгоритмов, пути в которых зависят от исходных данных, используют команды условной передачи управления.

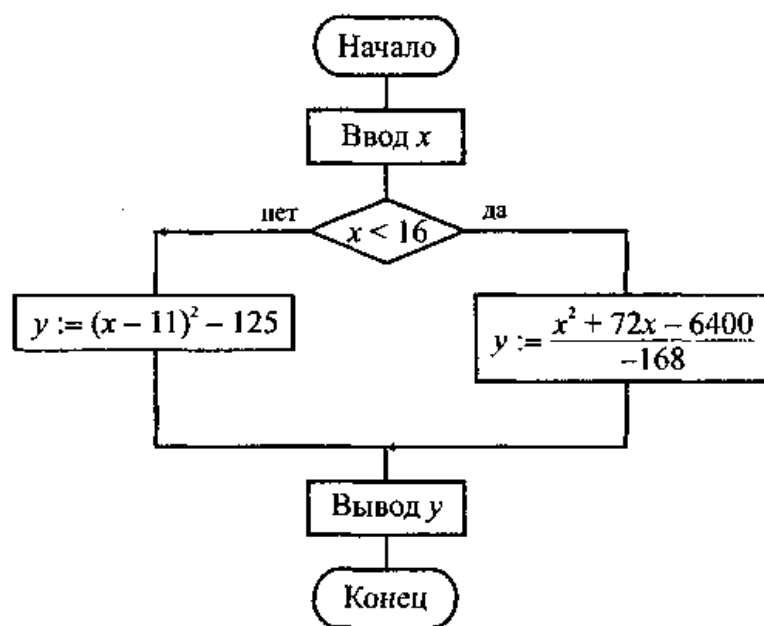
9.2.1. Пример 2

В качестве примера (несколько упрощенного по сравнению с заданиями лабораторной работы № 2) рассмотрим программу вычисления функции

причем x вводится с устройства ввода IR, результат y выводится на OR. Граф-схема алгоритма решения задачи показана на рис. 9.1.

$$y = \begin{cases} (x-11)^2 - 125, & \text{при } x \geq 16, \\ \frac{x^2 + 72x - 6400}{-168}, & \text{при } x < 16, \end{cases}$$

Рис. 9.1. Граф-схема алгоритма



В данной лабораторной работе используются двухсловные команды с непосредственной адресацией, позволяющие оперировать отрицательными числами и числами по модулю, превышающие 999, в качестве непосредственного операнда.

Оценив размер программы примерно в 20—25 команд, отведем для области данных ячейки ОЗУ, начиная с адреса 030. Составленная программа с комментариями представлена в виде табл. 9.4.

Адрес	Команда		Примечание
	Мнемокод	Код	
000	IN	01 0 000	Ввод x
001	WR 30	22 0 030	Размещение x в ОЗУ(ОЗО)
002	SUB #16	24 1 016	Сравнение с границей — $(x - 16)$
003	JS 010	13 0 010	Переход по отрицательной разности
004	RD 30	21 0 030	Вычисления по первой формуле
005	SUB #11	24 1 011	
006	WR 31	22 0 031	
007	MUL 31	25 0 031	
008	SUB #125	24 1 125	
009	JMP 020	10 0 020	Переход на вывод результата
010	RD 30	21 0 030	Вычисления по второй формуле
011	MUL 30	25 0 030	
012	WR 31	22 0 031	
013	RD 30	21 0 030	
014	MUL #72	25 1 072	
015	ADD 31	23 0 031	
016	ADI 106400	43 0 000	
017		106400	
018	DIVI 100160	46 0 000	

Таблица 9.4 (окончание)

Адрес	Команда		Примечание
	Мнемокод	Код	
020	OUT	02 0 000	Вывод результата
021	HLT	09 0 000	Стоп

9.2.2. Задание 2

1. Разработать программу вычисления и вывода значения функции:

$$y = \begin{cases} F_i(x), & \text{при } x \geq a, \\ F_j(x), & \text{при } x < a, \end{cases}$$

для вводимого из IR значения аргумента x . Функции и допустимые пределы изменения аргумента приведены в табл. 9.5, варианты заданий — в табл. 9.6.

2. Исходя из допустимых пределов изменения аргумента функций (табл. 9.5) и значения параметра a для своего варианта задания (табл. 9.6) выделить на числовой оси Ox области, в которых функция y вычисляется по представленной в п. 1 формуле, и недопустимые значения аргумента. На недопустимых значениях аргумента программа должна выдавать на OR максимальное 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 синтаксические 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 результата и адреса останова) в автоматическом режиме. В случае наличия ошибки выполнить пп. 5, *з* и 5, *д*; иначе перейти к п. 5, *е*;
- г) записать в РС стартовый адрес программы;
- д) наблюдая выполнение программы в режиме **Шаг**, найти команду, являющуюся причиной ошибки; исправить ее; выполнить пп. 5, *а* — 5, *в*;
- е) записать в IR значение аргумента $x < a$ (в области допустимых значений); 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 выполнение отложенной программы в режиме **Шаг** и записать в форме табл. 9.2 содержимое регистров ЭВМ перед выполнением каждой команды.

Таблица 9.5. Функции

k	$F_k(x)$	k	$F_k(x)$
1	$\frac{x+17}{1-x}; 2 \leq x \leq 12$	5	$\frac{(x+2)^2}{15}; 50 \leq x \leq 75$
2	$\frac{(x+3)^2}{x}; 1 \leq x \leq 50$	6	$\frac{2x^2+7}{x}; 1 \leq x \leq 30$
3	$\frac{1000}{x+10}; -50 \leq x \leq -15$	7	$\frac{x^2+2x}{10}; -50 \leq x \leq 50$
4	$(x+3)^3; -20 \leq x \leq 20$	8	$\frac{8100}{x^2}; 1 \leq x \leq 90$

Номер варианта	i	j	a	Номер варианта	i	j	a
1	2	1	12	8	8	6	30
2	4	3	-20	9	2	6	25
3	8	4	15	10	5	7	50
4	6	1	12	11	2	4	18
5	5	2	50	12	8	1	12
6	7	3	15	13	7	6	25
7	6	2	11	14	1	4	5

9.2.3. 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. 9.4.

5. Последовательность состояний регистров ЭВМ при выполнении программы в режиме **Шаг** для одного значения аргумента.
6. Результаты выполнения программы для нескольких значений аргумента, выбранных самостоятельно.

9.2.4. 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?

9.3. Лабораторная работа № 3.

Программирование цикла с переадресацией

При решении задач, связанных с обработкой массивов, возникает необходимость изменения исполнительного адреса при повторном выполнении некоторых команд. Эта задача может быть решена путем использования косвенной адресации.

9.3.1. Пример 3

Разработать программу вычисления суммы элементов массива чисел C_1, C_2, \dots, C_n . Исходными данными в этой задаче являются: n — количество суммируемых чисел и C_1, C_2, \dots, C_n — массив суммируемых чисел. Заметим, что должно выполняться условие $n > 1$, т. к. алгоритм предусматривает, по крайней мере, одно суммирование. Кроме того, предполагается, что суммируемые числа записаны в ОЗУ подряд, т. е. в ячейки памяти с последовательными адресами. Результатом является сумма S .

Составим программу для вычисления суммы со следующими конкретными параметрами: число элементов массива — 10, элементы массива расположены в ячейках ОЗУ по адресам 040, 041, 042, ..., 049. Используемые для решения задачи промежуточные переменные имеют следующий смысл: A_i — адрес числа C_i , $i \in \{1, 2, \dots, 10\}$; $ОЗУ(A_i)$ — число по адресу A_i , 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; промежуточные переменные: A_i — в ячейке ОЗУ с адресом 030, k — по адресу 031, S — по адресу 032. ГСА программы показана на рис. 9.2, текст программы с комментариями приведен в табл. 9.7.

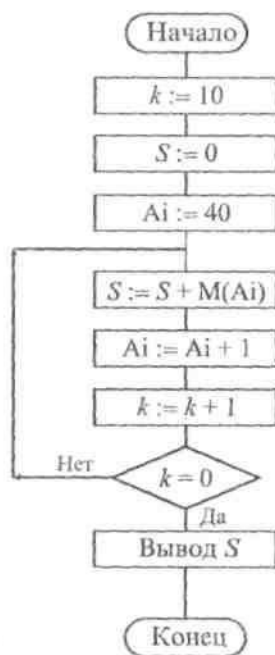


Рис. 9.2. Граф-схема алгоритма для примера 3

Таблица 9.7. Текст программы примера 3

Адрес	Команда	Примечание
000	RD #40	Загрузка начального адреса массива 040
001	WR 30	в ячейку 030

Таблица 9.7 (окончание)

Адрес	Команда	Примечание
002	RD #10	Загрузка параметра цикла $k = 10$ в ячейку 031
003	WR 31	
004	RD #0	Загрузка начального значения суммы $S = 0$
005	WR 32	в ячейку 032
006	M1: RD 32	Добавление
007	ADD @30	к текущей сумме
008	WR 32	очередного элемента массива
009	RD 30	Модификация текущего

9.3.2. Задание 3

1. Написать программу определения заданной характеристики последовательности чисел C_1, C_2, \dots, C_n . Варианты заданий приведены в табл. 9.8.
2. Записать программу в мнемосодах, введя ее в поле окна **Текст программы**.
3. Сохранить набранную программу в виде текстового файла и произвести ассемблирование мнемосохов.
4. Загрузить в ОЗУ необходимые константы и исходные данные.
5. Отладить программу.

Номер варианта	Характеристика последовательности чисел C_1, C_2, \dots, C_n
1	Количество четных чисел
2	Номер минимального числа
3	Произведение всех чисел
4	Номер первого отрицательного числа
5	Количество чисел, равных C_1
6	Количество отрицательных чисел
7	Максимальное отрицательное число
8	Номер первого положительного числа
9	Минимальное положительное число
10	Номер максимального числа
11	Количество нечетных чисел
12	Количество чисел, меньших C_1
13	Разность сумм четных и нечетных элементов массивов
14	Отношение сумм четных и нечетных элементов массивов

Примечание. Под четными (нечетными) элементами массивов понимаются элементы массивов, имеющие четные (нечетные) индексы. Четные числа — элементы массивов, делящиеся без остатка на 2.

9.3.3. Содержание отчета

1. Формулировка варианта задания.
2. Граф-схема алгоритма решения задачи.
3. Распределение памяти (размещение в ОЗУ переменных, программы и необходимых констант).
4. Программа.
5. Значения исходных данных и результата выполнения программы.

9.3.4. Контрольные вопросы

1. Как организовать цикл в программе?
2. Что такое параметр цикла?
3. Как поведет себя программа, приведенная в табл. 9.7, если в ней будет отсутствовать команда WR 31 по адресу 014?
4. Как поведет себя программа, приведенная в табл. 9.7, если метка M1 будет поставлена по адресу 005? 007?

9.4. Лабораторная работа № 4.

Подпрограммы и стек

В программировании часто встречаются ситуации, когда одинаковые действия необходимо выполнять многократно в разных частях программы (например, вычисление функции $\sin x$). При этом с целью экономии памяти не следует 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 подпрограммы указывается в команде вызова CALL, а параметры могут передаваться через определенные ячейки памяти, регистры или стек.

Возврат в точку вызова обеспечивается сохранением адреса текущей команды (содержимого регистра РС) при вызове и использованием в конце подпрограммы команды возврата RET, которая возвращает сохраненное значение адреса возврата в РС.

Для реализации механизма вложенных подпрограмм (возможность вызова подпрограммы из другой подпрограммы и т. д.) адреса возврата целесообразно 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 верхушку 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 позицию. Такой механизм напоминает действие магазина стрелкового оружия (отсюда и второе название). В программировании называют такую дисциплину обслуживания LIFO (Last, 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, поэтому при организации стека перемещается не массив слов относительно неподвижной 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 команде 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 верхушку 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 перед каждой записью содержимое SP уменьшается на 1, а после каждого извлечения содержимое SP увеличивается на 1. Таким образом, SP всегда указывает 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 занимает значительно меньшее время, чем в ОЗУ; кроме того, команды обращения с регистрами короче команд обращения к памяти. Поэтому в РОН размещаются наиболее часто используемые в программе данные, промежуточные результаты, счетчики циклов, косвенные адреса и т. п.

В системе команд учебной ЭВМ для работы с РОН используются специальные команды, мнемоники которых совпадают с мнемониками соответствующих команд для работы с ОЗУ, но в адресной части содержат символы регистров R0—R9.

Кроме обычных способов адресации (прямой и косвенной) в регистровых командах используются два новых — постинкрементная и преддекрементная (см. табл. 8.5). Кроме того, к регистровым относится команда организации цикла JRNZ R,M. 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.

9.4.1. Пример 4

Даны три массива чисел. Требуется вычислить среднее арифметическое их максимальных элементов. Каждый массив задается двумя параметрами: адресом первого элемента и длиной.

Очевидно, в программе трижды необходимо выполнить поиск максимального

элемента массива, поэтому следует написать соответствующую подпрограмму.

Параметры в подпрограмму будем передавать через регистры: R1 — начальный адрес массива, R2 — длина массива.

Рассмотрим конкретную реализацию этой задачи. Пусть первый массив начинается с адреса 085 и имеет длину 14 элементов, второй — 100 и 4, третий — 110 и 9. Программа будет состоять из основной части и подпрограммы. Основная программа задает параметры подпрограмме, вызывает ее и сохраняет результаты работы подпрограммы в рабочих ячейках. Затем осуществляет вычисление среднего арифметического и выводит результат на устройство 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 временного хранения текущего элемента. Подпрограмма возвращает результат через аккумулятор. В табл. 9.9 приведен текст основной программы и подпрограммы. Обратите внимание, цикл в подпрограмме организован с помощью команды JRBZ, а модификация текущего адреса — средствами постинкрементной адресации.

Таблица 9.9. Программа примера 4

Команда *	Примечания
Основная программа	
RD #85	Загрузка
WR R1	параметров
RD #14	первого
WR R2	массива
CALL M	Вызов подпрограммы
WR R6	Сохранение результата
RD #100	Загрузка
WR R1	параметров
RD #4	второго
WR R2	массива

Таблица 9.9 (окончание)

Команда	Примечания
CALL M	Вызов подпрограммы
WR R7	Сохранение результата
RD #110	Загрузка
WR R1	параметров
RD #9	третьего
WR R2	массива
CALL M	Вызов подпрограммы
ADD R7	Вычисление
ADD R6	среднего
DIV #3	арифметического
OUT	Вывод результата
Подпрограмма MAX	
HLT	Стоп
M: RD @R1	Загрузка
WR P3	первого элемента в R3
L2: RD @R1+	Чтение элемента и модификация адреса
WR R4	Сравнение
SUB R3	и замена,
JS L1	если $R3 < R4$
MOV R3, R4	
L1: JRNZ R2, L2	Цикл
RD R3	Чтение результата в Acc
RET	Возврат

9.4.2. Задание 4

Составить и отладить программу учебной ЭВМ для решения следующей задачи. Три массива в памяти заданы начальными адресами и длинами. Вычислить и вывести на устройство вывода среднее арифметическое параметров этих массивов. Параметры определяются заданием к предыдущей лабораторной работе (см. табл. 9.8), причем соответствие между номерами вариантов заданий 3 и 4 устанавливается по табл. 9.10.

Таблица 9.10. Соответствие между номерами заданий

Номер варианта задания 4	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Номер строки в табл. 9.9	5	7	13	11	9	12	1	10	14	3	6	8	2	4

9.4.3. 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 необходимых констант).
5. Тексты программы и подпрограммы.
6. Значения исходных данных и результата выполнения программы.

9.4.4. Контрольные вопросы

1. Как работает команда MOV R3, R7?
2. Какие действия выполняет процессор при реализации команды CALL?
3. Как поведет себя программа примера 4, 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 адресации, организовать дополнительный стек в произвольной области памяти, не связанный с SP?

9.5. Лабораторная работа № 5. Командный цикл процессора

Реализация программы в ЭВМ сводится к последовательному выполнению команд. Каждая команда, в свою очередь, выполняется как последовательность микрокоманд, реализующих элементарные действия над операционными 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 специальном окне **Микрокомандный уровень** (см. рис. 8.8).

9.5.1. Задание 5.1

Выполнить снова последовательность команд по варианту задания 1 (см. табл. 9.3), но в режиме **Шаг**. Зарегистрировать изменения состояния процессора и памяти в форме табл. 9.11, в которой приведены состояния ЭВМ при выполнении примера 1 (фрагмент).

9.5.2. Задание 5.2

Записать последовательность микрокоманд для следующих команд модели учебной ЭВМ:

9.5.3. 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 микрокоманд, реализующих команду) для несуществующей команды "умножение модулей чисел".
4. Что изменится в работе процессора, если в каждой микропрограмме микрокоманду увеличения программного счетчика $PC := PC + 1$ переместить в самый конец микропрограммы?

- ☐ ADD R3
- ☐ ADD @R3
- ☐ ADD @R3+
- ☐ ADD -@R3
- ☐ JRNZ R3,M
- ☐ MOV R4,R2
- ☐ JMP M
- ☐ CALL M
- ☐ RET: PUSH R3
- ☐ POP R5

Таблица 9.11. Состояние модели в режиме моделирования на уровне микрокоманд

Аспект (PC)	Микрокоманда	Микрокоманда	ОЗУ		СР			АУ		Ячейки	
			MAR	MDR	COP	TA	ADR	Acc	DR	020	030
000	RD #20	MAR := PC	000	000000	00	0	000	000000	000000	000000	000000
		MRd	000								
		CR := MDR		211020							
		PC := PC + 1			21	1	020				
001		Acc := 000.ADR									
	WR 30	MAR := PC						000020			
		MRd	001								
		CR := MDR		220030							
		PC := PC + 1			22	0	030				
002		MAR := ADR									
		MDR := Acc	030								
		MWr		000020							
	ADD #5	MAR := PC									000020
		MRd	002								
		CR := MDR		231005							
		PC := PC + 1			23	1	005				
003		DR := 000.ADR									
		F _{AV} := ALL							000005		
	WR @30	MAR := PC						000025			

9.6. Лабораторная работа № 6.

Программирование внешних устройств

Целью этой лабораторной работы является изучение способов организации взаимодействия процессора и внешних устройств (ВУ) в составе ЭВМ.

Выше отмечалось, что связь процессора и ВУ может осуществляться в синхронном 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 символов, формируя флаг ошибки).

Асинхронный обмен предполагает анализ процессором состояния ВУ, которое определяет готовность ВУ выдать или принять данные или факт осуществления некоторого события, контролируемого системой. К таким устройствам 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 регистру состояния ВУ с последующим анализом значения соответствующего разряда слова состояния. Такое обращение следует предусмотреть в программе с некоторой периодичностью, независимо от фактического наступления контролируемого события (например, нажатие клавиши).

Во втором случае при возникновении контролируемого события ВУ формирует процессору запрос на прерывание программы, по которому процессор и осуществляет связь с ВУ.

9.6.1. Задание 6

Свой вариант задания (табл. 9.12) требуется выполнить двумя способами — сначала в режиме программного контроля, далее модифицировать программу таким образом, чтобы события обрабатывались в режиме прерывания программы. Поскольку "фоновая" (основная) задача для этого случая в заданиях отсутствует, роль ее может сыграть "пустой цикл":

```
М: NOP
    NOP
    JMP M
```

Таблица 9.12. Варианты задания 6

№ варианта	Задание	Используемые ВУ	Пояснения
1	Ввод пятиразрядных чисел в ячейки ОЗУ	Клавиатура	Программа должна обеспечивать ввод последовательности ASCII-кодов десятичных цифр (не длиннее пяти), перекодировку в "8421", упаковку в десятичное число (первый введенный символ — старшая цифра) и размещение в ячейке ОЗУ. ASCII-коды не-цифр игнорировать.
2	Программа ввода символов с клавиатуры с выводом на дисплей	Клавиатура, дисплей, таймер	Очистка буфера клавиатуры после ввода 50 символов или каждые 10 с
3	Вывод на дисплей трех текстов, хранящихся в памяти, с задержкой	Дисплей, таймер	Первый текст выводится сразу при запуске программы, второй — через 15 с, третий — через 20 с после второго
4	Вывод на дисплей одного из трех текстовых сообщений, в зависимости от нажатой клавиши	Клавиатура, дисплей	<1> — вывод на дисплей первого текстового сообщения, <2> — второго, <3> — третьего, остальные символы — нет реакции
5	Выбирать из потока ASCII-кодов только цифры и выводить их на дисплей	Клавиатура, дисплей, тоногенератор	Вывод каждой цифры сопровождается коротким звуковым сигналом
6	Выводить на дисплей каждый введенный с клавиатуры символ, причем цифру выводить "в трех экземплярах"	Клавиатура, дисплей, тоногенератор	Вывод каждой цифры сопровождается троекратным звуковым сигналом
7	Селективный ввод символов с клавиатуры	Клавиатура, дисплей	Все русские буквы, встречающиеся в строке ввода — в верхнюю часть экрана дисплея (строки 1—4), все цифры — в нижнюю часть экрана (строки 5—8), остальные символы не выводить.
8	Вывод содержимого заданного участка памяти на дисплей посимвольно с заданным промежутком времени между выводами символов	Дисплей, таймер	Остаток от деления на 256 трех младших разрядов ячейки памяти рассматривается как ASCII-код символа. Начальный адрес памяти, длина массива вывода и промежуток времени — параметры подпрограммы
9	Программа ввода символов с клавиатуры с выводом на дисплей	Клавиатура, дисплей	Очистка буфера клавиатуры после ввода 35 символов

Таблица 9.12 (окончание)

№ варианта	Задание	Используемые ВУ	Пояснения
10	Выводить на дисплей каждый введенный с клавиатуры символ, причем заглавную русскую букву выводить "в двух экземплярах"	Клавиатура, дисплей, таймер	Очистка буфера клавиатуры после ввода 48 символов, очистка экрана каждые 15 с
11	Вывод на дисплей содержимого группы ячеек памяти в числовой форме (адрес и длина группы — параметры подпрограммы)	Дисплей, таймер	Содержимое ячейки распаковывается (с учетом знака), каждая цифра преобразуется в соответствующий ASCII-код и выдается на дисплей. При переходе к выводу содержимого очередной ячейки формируется задержка 10 с
12	Определить промежуток времени между двумя последовательными нажатиями клавиш	Клавиатура, таймер	Результат выдается на ОР. (Учитывая инерционность модели, нажатия не следует производить слишком быстро.)

9.6.2. 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 дисплее, ведется подсчет количества введенных символов (после каждых 50 дается команда на очистку буфера клавиатуры, после 128 — очищается дисплей). Переполнение таймера выключает клавиатуру и включает сигнал завершения ввода (можно тон этого сигнала сопоставить с количеством введенных символов). Параметр T вводится из IR. Результат S — средняя скорость ввода (символ/с) выдается на ОР. Учитывая, что модель учебной ЭВМ оперирует только целыми числами, можно выдавать результат в формате $S \times 60$ символов/мин.
2. Разработать программу-тест на степень запоминания текста. Три различных варианта текста выводятся последовательно на дисплей на T_1 секунд с промежутками T_2 секунд. Далее эти тексты (то, что запомнилось) вводятся с клавиатуры (в режиме ввода строки) и программно сравниваются с исходными текстами. Выдается количество (процент) ошибок.
3. Разработать программу-калькулятор. Осуществлять ввод из буфера клавиатуры последовательности цифр, упаковку (см. задание 1 в табл. 9.12).

Разделители — знаки бинарных арифметических операций и $=$. Результат переводится в ASCII-коды и выводится на дисплей.

9.6.3. Порядок выполнения работы

1. Запустить программную модель учебной ЭВМ и подключить к ней определенные в задании внешние устройства (меню **Внешние устройства | Менеджер ВУ**).
2. Написать и отладить программу, предусмотренную заданием, с использованием

программного анализа флагов готовности ВУ. Продemonстрировать работающую программу преподавателю.

3. Изменить отлаженную в п. 2 программу таким образом, чтобы процессор реагировал на готовность ВУ с помощью подсистемы прерывания. Продemonстрировать работу измененной программы преподавателю.

9.6.4. Содержание отчета

1. Текст программы с программным анализом флагов готовности ВУ.
2. Текст программы с обработчиком прерывания.

9.6.5. Контрольные вопросы

1. При каких условиях устанавливается и сбрасывается флаг готовности клавиатуры 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 обозревателя дисплея кнопки **Очистить** и загрузки по адресу CR (11) константы #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 произойдут, если завершить обработчик прерываний командой RET, а не IRET?

9.7. Лабораторная работа № 7.

Принципы работы кэш-памяти

В разд. 8.8 данной книги описаны некоторые алгоритмы замещения строк кэш-памяти. Цель настоящей лабораторной работы — проверить работу различных алгоритмов замещения при различных режимах записи.

9.7.1. Задание 7

В качестве задания предлагается некоторая короткая "программа" (табл. 9.14), которую необходимо выполнить с подключенной кэш-памятью (размером 4 и 8 ячеек) в шаговом режиме для следующих двух вариантов алгоритмов замещения (табл. 9.13).

Таблица 9.13. Пояснения к вариантам задания 7

Номера вариантов	Режим записи	Алгоритм замещения
1, 7, 11	Сквозная	СЗ, без учета бита записи
	Обратная	О, с учетом бита записи
2, 5, 9	Сквозная	БИ, без учета бита записи
	Обратная	О, с учетом бита записи
3, 6, 12	Сквозная	О, без учета бита записи
	Обратная	СЗ, с учетом бита записи
4, 8, 10	Сквозная	БИ, без учета бита записи
	Обратная	БИ, с учетом бита записи

Таблица 9.14. Варианты задания 7

№ вари- анта	Номера команд программы						
	1	2	3	4	5	6	7
1	RD #12	WR 10	WR @10	ADD 12	WR R0	SUB 10	PUSH R0
2	RD #65	WRR2	MOV R4,R2	WR 14	PUSH R2	POP R3	CALL 002
3	RD #16	SUB #5	WR 9	WR @9	WR R3	PUSH R3	POP R4
4	RD #99	WR R6	MOV R7,R6	ADD R7	PUSH R7	CALL 006	POP R8
5	RD #11	WR R2	WR -@R2	PUSH R2	CALL 005	POP R3	RET
6	RD #19	SUB #10	WR9	ADD #3	WR @9	CALL 006	POP R4

Таблица 9.14 (окончание)

№ вари- анта	Номера команд программы						
	1	2	3	4	5	6	7
7	RD #6	CALL 006	WR11	WRR2	PUSH R2	RET	JMP 002
8	RD#8	WRR2	WR @R2+	PUSH R2	POP R3	WR -@R3	CALL 003
9	RD #13	WR14	WR@14	WR@13	ADD 13	CALL 006	RET
10	RD #42	SUB #54	WR16	WR@16	WRR1	ADD @R1+	PUSH R1
11	RD #10	WRR5	ADD R5	WRR6	CALL 005	PUSH R6	RET
12	JMP 006	RD #76	WR 14	WRR2	PUSH R2	RET	CALL 001

Не следует рассматривать заданную последовательность команд как фрагмент программы¹. Некоторые конструкции, например, последовательность команд PUSH R6, RET в общем случае не возвращает программу в точку вызова подпрограммы. Такие группы команд введены в задание для того, чтобы обратить внимание студентов на особенности функционирования стека.

9.7.2. Порядок выполнения работы

1. Ввести в модель учебной ЭВМ текст своего варианта программы (см. табл. 9.14), 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 варианта из табл. 9.13.
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. 9.13 и выполнить программу в шаговом режиме еще раз, фиксируя последовательность номеров замещаемых ячеек кэш-памяти.

¹ Напомним, что программа определяется как последовательность команд, выполнение которых позволит получить некий результат.

9.7.3. Содержание отчета

1. Вариант задания — текст программы и режимы кэш-памяти.
2. Последовательность состояний кэш-памяти размером 4 ячейки при однократном выполнении программы (команды 1—7).
3. Последовательность микрокоманд при выполнении команды WR с отметкой тех микрокоманд, в которых возможна модификация кэш-памяти.
4. Для варианта кэш-памяти размером 8 ячеек — последовательность номеров замещаемых ячеек кэш-памяти для второго варианта параметров кэш-памяти при двукратном выполнении программы (команды 1—7).

9.7.4. Контрольные вопросы

1. В чем смысл включения кэш-памяти в состав ЭВМ?
2. Как работает кэш-память в режиме обратной записи? Сквозной записи?
3. Как зависит эффективность работы ЭВМ от размера кэш-памяти?
4. В какую ячейку кэш-памяти будет помещаться очередное слово, если свободные ячейки отсутствуют?
5. Какие алгоритмы замещения ячеек кэш-памяти вам известны?

9.8. Лабораторная работа № 8.

Алгоритмы замещения строк кэш-памяти

Цель работы — изучение влияния параметров кэш-памяти и выбранного алгоритма замещения на эффективность работы системы. Эффективность в данном случае оценивается числом кэш-попаданий по отношению к общему числу обращений к памяти. Учитывая разницу в алгоритмах в режимах сквозной и обратной записи, эффективность использования кэш-памяти вычисляется выражениям (8.2) и (8.3) 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 работать с алгоритмами замещения типа очередь, а программы с большим числом 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 должен влиять размер кэш-памяти.

Для проверки высказанных выше предположений выполняется настоящая лабораторная работа.

9.8.1. Задание 8

В данной лабораторной работе все варианты задания одинаковы: исследовать эффективность работы кэш-памяти при выполнении двух разнотипных программ, написанных и отлаженных вами при выполнении лабораторных работ № 2 и 4.

9.8.2. Порядок выполнения работы

1. Загрузить в модель учебной ЭВМ отлаженную программу из лабораторной работы № 2.
2. В меню **Работа** установить режим **Кэш-память**.
3. В меню **Вид** выбрать команду **Кэш-память**, открыв тем самым окно **Кэшпамять**, в нем нажать первую слева кнопку на панели инструментов, открыв 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**, вычислить значение коэффициента эффективности K и записать в ячейку табл. 9.15, помеченную звездочкой.
5. Выключить кэш-память модели (**Работа | Кэш-память**) и изменить один из ее параметров — установить флаг с учетом бита записи (в окне **Параметры кэш-памяти**).
6. Повторить п. 4, поместив значение полученного коэффициента эффективности в следующую справа ячейку табл. 9.15.
7. Последовательно менять параметры кэш-памяти, повторить пп. 3—5, заполняя все ячейки табл. 9.15.

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 лабораторной работы № 4, заполняя вторую таблицу по форме табл. 9.15.

9.8.3. Содержание отчета

1. Две таблицы по форме табл. 9.15 с результатами моделирования программ из лабораторных работ № 2 и 4 при разных режимах работы кэш-памяти.
2. Выводы, объясняющие полученные результаты.

9.8.4. 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 применения кэш-памяти большого объема (в кэш-память целиком помещается программа)?
3. Как скажется на эффективности алгоритмов замещения учет значения бита записи XV при работе кэш-памяти в режиме обратной записи? Сквозной записи?
4. Для каких целей в структуру ячейки кэш-памяти включен бит использования. Как устанавливается и сбрасывается этот бит?

Таблица 9.15. Результаты эксперимента

Способ	Сквозная запись					
Алгоритм	Случайное замещение		Очередь		Бит U	
Размер	без W	с W	без W	с W	без W	с W
4	*					
8						
16						
32						
Способ	Обратная запись					
Алгоритм	Случайное замещение		Очередь		Бит U	
Размер	без W	с W	без W	с W	без W	с W
4						
8						
16						
32						

ГЛАВА 10

Курсовая работа

10.1. Цель и содержание работы

Целью курсовой работы является:

- обобщение, закрепление и углубление знаний по дисциплинам, связанным с проектированием средств ВТ;
- формирование навыков разработки и оформления текстовой и графической технической документации;
- развитие навыков устных сообщений по содержанию работы.

Содержанием курсовой работы является *разработка арифметико-логического устройства* (АЛУ), реализующего заданный набор операций с учетом ограничений на код выполнения операций и способ построения управляющего автомата.

10.2. Задания

Задания на курсовую работу включают в себя некоторый набор исходных данных и ограничений для проектирования АЛУ. Все варианты задания сведены в табл. 10.1. Строка таблицы представляет один вариант задания, причем номер варианта определяется номером группы (1—2) и порядковым номером студента по списку группы (1—25).

Разрабатываемое АЛУ должно выполнять одну арифметическую и одну поразрядную бинарную логическую операцию, причем на способ выполнения арифметической операции заданием накладываются некоторые ограничения. Варианты операций обозначаются в табл. 10.1 следующим образом:

- \pm — алгебраическое сложение/вычитание;
- \times — умножение обыкновенное;

Таблица 10.1. Варианты курсовых заданий

№	Операции	Код ВО	Флаги	Тип УА	№	Операции	Код ВО	Флаги	Тип УА
1-1	$\pm, \&$	ПК	OV, Z	2	2-1	$\times 2, \oplus$	ПК	OV, P	4
1-2	\times, \vee	ПК	OV, P	3	2-2	\times, \oplus	ПК	OV, C	1
1-3	$\div 1, \oplus$	ПК	OV, Z	4	2-3	$\pm, \&$	ОК	OV, Z	5
1-4	$\times 2, \equiv$	ПК	OV, C	5	2-4	$\div 2, \equiv$	ПК	OV, P	6
1-5	$\div 2, \&$	ПК	OV, Z	6	2-5	$\pm, \&$	ПК	OV, Z	4
1-6	\times, \vee	ПК	OV, P	1	2-6	$\div 1, \vee$	ПК	OV, P	3
1-7	\pm, \equiv	ОК	OV, C	2	2-7	$\pm, \&$	ДК	OV, Z	2
1-8	$\times 2, \oplus$	ПК	OV, P	3	2-8	$\times 2, \equiv$	ПК	OV, P	5
1-9	$\div 1, \&$	ПК	OV, Z	4	2-9	$\div 2, \&$	ПК	OV, Z	4
1-10	$\times 2, \vee$	ПК	OV, C	5	2-10	$\times 2, \vee$	ПК	OV, P	6
1-11	\pm, \equiv	ДК	OV, Z	6	2-11	$\pm, \&$	ОК	OV, Z	1
1-12	\times, \vee	ПК	OV, P	5	2-12	$\div 1, \vee$	ПК	OV, Z	2
1-13	\pm, \oplus	ОК	OV, C	4	2-13	$\pm, \&$	ДК	OV, C	3
1-14	$\div 2, \vee$	ПК	OV, P	6	2-14	$\times 2, \oplus$	ПК	OV, Z	4
1-15	$\pm, \&$	ДК	OV, Z	3	2-15	$\div 1, \equiv$	ПК	OV, P	3
1-16	\times, \vee	ПК	OV, C	2	2-16	$\div 2, \vee$	ПК	OV, Z	2
1-17	\pm, \equiv	ПК	OV, Z	1	2-17	$\pm, \&$	ОК	OV, C	1
1-18	$\times 2, \oplus$	ПК	OV, P	1	2-18	\times, \oplus	ПК	OV, C	6
1-19	$\pm, \&$	ОК	OV, C	2	2-19	$\div 1, \&$	ПК	OV, Z	5
1-20	$\div 2, \vee$	ПК	OV, P	3	2-20	$\times 2, \vee$	ПК	OV, P	1
1-21	$\div 1, \&$	ПК	OV, Z	4	2-21	\pm, \equiv	ОК	OV, Z	2
1-22	\times, \equiv	ПК	OV, C	5	2-22	\times, \oplus	ПК	OV, Z	6
1-23	$\pm, \&$	ДК	OV, Z	6	2-23	$\pm, \&$	ДК	OV, P	5
1-24	$\times 2, \vee$	ПК	OV, P	3	2-24	$\times 2, \vee$	ПК	OV, C	4
1-25	$\times 1, \equiv$	ПК	OV, C	5	2-25	$\div 2, \equiv$	ПК	OV, Z	3

- $\times 2$ — умножение ускоренное (с анализом двух разрядов множителя);
- $\div 1$ — деление с восстановлением остатка;
- $\div 2$ — деление без восстановления остатка;
- \vee — дизъюнкция;
- $\&$ — конъюнкция;
- \oplus — неравнозначность;
- \equiv — эквивалентность.

Для всех вариантов заданий исходные данные (операнды) поступают в формате 16-разрядных двоичных чисел с фиксированной запятой, представленных в прямом коде $[a_0a_1\dots a_{15}]_d$, $[b_0b_1\dots b_{15}]_d$ причем нулевой разряд является знаковым и запятая фиксирована после знакового разряда. Таким образом, в арифметических операциях участвуют правильные дроби со своими знаками (в логических операциях, естественно, положение запятой и знак игнорируются, операции выполняются над 16-разрядными двоичными векторами). Соответственно, результат операции должен быть представлен в той же форме: $[c_0c_1\dots c_{15}]_d$.

В задании вводится ограничение на код выполнения операции (столбец **Код ВО** в табл. 10.1). Если код ВО отличается от прямого — обратный (ОК) или дополнительный (ДК), то при выполнении *арифметической операции* следует перевести операнды в заданный код, выполнить в нем операцию, а результат вновь перевести в прямой код. Логические операции, естественно, выполняются без всякого преобразования.

Результатом выполнения операции в АЛУ должно быть не только значение суммы (произведения, конъюнкции и др.) но и *признаки результата* (флаги). Каждый вариант задания предполагает формирования двух различных флагов (заданных в столбце **Флаги** табл. 10.1) из приведенного ниже множества.

- Z — признак нулевого результата;
- P — признак четности числа единиц в результате;
- C — признак переноса (заема) из старшего разряда;
- OV — признак арифметического переполнения.

В столбце **Тип УА** задан номер типа управляющего автомата, который необходимо использовать при проектировании заданного АЛУ. Список типов УА приведен ниже.

- 1 — "жесткая логика", автомат Мура;
- 2 — "жесткая логика", автомат Мили;
- 3 — программируемая логика, единый формат микрокоманды, принудительная адресация;
- 4 — программируемая логика, единый формат микрокоманды, естественная адресация;
- 5 — программируемая логика, различные форматы для операционных микрокоманд и микрокоманд перехода, естественная адресация;
- 6 — программируемая логика, различные форматы для операционных микрокоманд и микрокоманд перехода, принудительная адресация.

В задании не определены ограничения на базис логических, операционных элементов и элементов памяти. Поэтому при разработке структурных и функциональных схем можно использовать любые стандартные логические и операционные элементы.

10.3. Этапы выполнения работы

В главе 4 настоящего пособия подробно рассматривается процесс проектирования цифрового устройства. При проектировании его удобно представить в виде композиции

операционного и управляющего автоматов (см. разд. 4.2). Тогда процесс проектирования устройства сводится к процедурам последовательного проектирования операционного и управляющего автоматов. Здесь можно выделить следующие этапы:

1. Разработка алгоритмов выполняемых операций. На этом этапе следует определить список входных, выходных и внутренних переменных и выбрать коды выполняемых операций. Поскольку все задания предполагают реализацию одной/двух арифметических и одной логической операций, целесообразно представить все алгоритмы в форме объединенной ГСА.
2. Разработка структуры операционного автомата— определение состава элементов и связей между ними. Разработка структуры нестандартных элементов. Результатом работы на этом этапе должна стать структурная (функциональная) схема операционного автомата, а также функциональные схемы всех использованных в ОА нестандартных элементов.
3. Определение списка микроопераций и логических условий. Необходимо сопоставить каждому оператору из ГСА микрокоманду или группу микрокоманд, обеспечивающих реализацию этого оператора на разработанной ранее структуре. На этом этапе возможно расширение набора элементов и/или связей структуры, если без такого расширения не удастся реализовать все операторы ГСА. Кроме того, необходимо определить, где будут формироваться значения логических переменных, которые анализируются в логических вершинах ГСА и при необходимости предусмотреть специальные элементы структуры для формирования этих значений. Результат работы на этом этапе — списки микроопераций и логических условий ОА.
4. Разработка микропрограммы выполнения заданных операций на выбранной структуре ОА. В простейшем случае можно сохранить топологию графа алгоритма и просто заменить операторы во всех операторных вершинах на соответствующие микрооперации, а условия, которые анализируются в условных вершинах — на соответствующие логические условия из списка, полученного на предыдущем этапе. Однако при переходе от ГСА к микропрограмме следует всегда стремиться к уменьшению числа (операторных) вершин, что, в свою очередь, приведет к упрощению схемы управляющего автомата. Достигнуть этого можно, например, совмещением двух или более операторных вершин ГСА в одну вершину микропрограммы, если смысл реализуемого алгоритма и разработанная ранее структура операционного автомата позволяют выполнить эти действия одновременно. Разработанная на этом этапе микропрограмма является исходной для проектирования управляющего автомата.

На этом заканчивается процесс разработки операционного автомата.

Этапы разработки управляющего автомата различны в зависимости от его типа. Для разработки микропрограммного автомата с *"жесткой" логикой* следует:

1. Осуществить разметку микропрограммы. Эта процедура устанавливает соответствие между вершинами микропрограммы и состояниями автомата. В разд. 4.4.1 настоящего пособия описано, как осуществлять разметку микропрограммы для проектирования *автомата Мура* и *автомата Мили*.
2. Построить граф автомата. Граф автомата строят по размеченной микропрограмме, причем вершины графа соответствуют состояниям автомата, а ребра — переходам, на этом этапе можно не показывать на графе функцию переходов.
3. Выбрать тип элемента памяти, закодировать состояния автомата.
4. Составить автоматную таблицу переходов. Пример построения такой таблицы

для случая использования в качестве элементов памяти D-триггеров приведен в *разд. 4.4.1*. Аналогичный формат имеет таблица при использовании Т-триггеров. Если в качестве элемента памяти автомата выбран RS-триггер, в каждом разряде необходимо сформировать две функции возбуждения — для R- и S-входов.

5. Определить функции возбуждения для переключения элементов памяти. Автоматная таблица может рассматриваться как таблица истинности, задающая функции возбуждения для входов элементов памяти автомата. Все функции возбуждения в общем случае зависят от значений элементов памяти T_j и значений логических условий X_j . При необходимости можно для каждой из функций построить карту Карно и записать ее минимальное выражение. Иногда проще бывает предварительно дешифровать состояния автомата и записать функции возбуждения в зависимости от текущего состояния автомата и слова логических условий.
6. Определить функции выходов, формирующие значения микроопераций. Для автомата Мура функция выходов в каждом такте дискретного времени зависит только от текущего состояния автомата и значение выхода определяется содержимым операторной вершины микропрограммы, соответствующей этому состоянию автомата. В автомате Мили выходное слово соответствует содержимому той операторной вершины микропрограммы, через которую осуществляется переход из текущего состояния автомата в следующее. Поэтому функция выходов автомата Мили, как и его функция переходов, зависит от текущего состояния автомата и слова логических условий.
7. Построить функциональную схему УА. Получив выражения для функций возбуждения и выходов, можно построить функциональную схему управляющего автомата с использованием выбранных элементов памяти и стандартного базиса логических и операционных элементов.

Для разработки микропрограммного автомата с программируемой логикой следует:

1. Разбить множество микроопераций на подмножества попарно-несовместимых микроопераций (этот пункт не выполняется, если выбран "вертикальный" или "горизонтальный" способ кодирования поля микроопераций).
2. Определить формат микрокоманды (микрокоманд).
3. Разработать функциональную схему управляющего автомата.
4. Заполнить таблицу программирования ПЗУ микрокоманд.

Проектирование управляющего автомата с программируемой логикой с различными способами адресации микрокоманд и кодирования микроопераций подробно описаны в *разд. 4.4.2*.

10.4. Содержание пояснительной записки

Пояснительная записка к курсовой работе должна включать следующую информацию (для вариантов с управляющими автоматами с "жесткой" логикой):

1. Титульный лист.
2. Задание на проектирование АЛУ.
 36201216. Форматы входных, выходных и внутренних переменных, с которыми оперирует АЛУ.
 36201217. ГСА выполняемых операций и объединенную ГСА.
 36201218. Структурную схему операционного автомата АЛУ.
 36201219. Функциональные схемы "нестандартных" элементов ОА. При

необходимости привести процедуры синтеза операционных элементов (например, карты Карно для минимизации булевых функций).

7. Список микроопераций, реализуемых в ОА.

8. Список логических условий, формируемых в ОА.

36201264. Микропрограмму выполняемых в АЛУ операций в терминах микроопераций и логических условий с разметкой состояний для проектирования управляющего автомата.

10. Граф автомата.

11. Таблицу кодирования внутренних состояний автомата.

36201360. Описание выбранного элемента памяти (триггера) и его таблица функционирования.

36201361. Автоматную таблицу переходов.

36201362. Выражения для функций возбуждения элементов памяти (при необходимости — процедуру минимизации).

15. Выражения для функций выходов управляющего автомата.

16. Функциональную схему управляющего автомата.

17. Заключение.

18. Библиографический список.

Для вариантов с *управляющими автоматами с программируемой логикой* вместо информации, приведенной в пп. 9—16, необходимо включить:

1. Микропрограмму выполняемых в АЛУ операций в терминах микроопераций и логических условий.

2. Формат или форматы микрокоманд с указанием размеров и назначения полей.

3. Разбиение множества микроопераций ОА на подмножества несовместимых микроопераций.

4. Функциональную схему управляющего автомата.

5. Таблицу программирования ПЗУ микрокоманд.