

Министерство транспорта Российской Федерации
Федеральное агентство железнодорожного транспорта
ГОУ ВПО «Дальневосточный государственный
университет путей сообщения»

Кафедра «Системы автоматизированного проектирования»

Е.В. Буняева

ОРГАНИЗАЦИЯ ЭВМ И СИСТЕМ

Методическое пособие для выполнения лабораторных работ

Хабаровск
Издательство ДВГУПС
2010

УДК
ББК

Рецензенты:

Буняева Е. В.

Организация ЭВМ и систем: метод. пособие для выполнения лабораторных работ/ Е.В. Буняева – Хабаровск: Изд-во ДВГУПС, 2010. – **51** с.: ил.

Методическое пособие соответствует ГОС ВПО направления подготовки дипломированных специалистов 230100 – «Информатика и вычислительная техника» специальности 230104 – «Системы автоматизированного проектирования».

Пособие содержит описание программной учебной модели ЭВМ, лабораторные работы по исследованию принципов построения и функционирования вычислительных машин. Для каждой работы сформулирована цель, дана краткая теоретическая справка по теме исследования, варианты заданий. Требования к оформлению отчета и контрольные вопросы; к некоторым заданиям приведены примеры выполнения.

Предназначено для студентов 2-го курса дневной формы обучения, изучающих дисциплину «Организация ЭВМ и систем».

УДК
ББК

© ГОУ ВПО «Дальневосточный государственный
университет путей сообщения» (ДВГУПС), 2010

ВВЕДЕНИЕ

Электронные вычислительные машины (ЭВМ) и системы на их основе являются одним из самых сложных произведений научной и инженерной мысли в отношении как аппаратного, так и программного обеспечения. Поэтому понятно то пристальное внимание, которое уделяется изучению ЭВМ в подготовке специалистов и бакалавров по направлению «Информатика и вычислительная техника».

Однако, современные вычислительные машины достаточно сложны для начального этапа изучения архитектуры ЭВМ.

Одним из решений данной проблемы является использование в обучении программной учебной модели ЭВМ (эмулятора). Эмулятор, с одной стороны, достаточно прост, чтобы студент мог освоить базовые понятия архитектуры ЭВМ (система команд, цикл команды, способы адресации, уровни памяти, способы взаимодействия центрального процессора с памятью), с другой – архитектурные особенности эмулятора соответствуют тенденциям развития современных ЭВМ. Программная модель позволяет реализовать доступ к основным элементам ЭВМ, обеспечивая удобство и наглядность.

Методическое пособие рассчитано на выполнение курса лабораторных работ в рамках дисциплины «Организация ЭВМ и систем». Для каждой работы сформулирована цель, дана краткая теоретическая справка по теме лабораторной работы, варианты заданий, требования к содержанию отчета по работе и контрольные вопросы; к некоторым заданиям приведены примеры выполнения.

Курс лабораторных работ разделен на несколько уровней. Лабораторные работы №1 – 4 направлены на ознакомление с архитектурой процессора (ЦП), системой команд, способами адресации и основными приемами программирования на языках низкого уровня. В ходе выполнения лабораторной работы №5 студенты знакомятся с реализацией командного цикла ЦП на уровне микроопераций. Работы №6 и 7 посвящены организации кэш-памяти и эффективности различных алгоритмов замещения строк в заполненной кэш-памяти.

Методическое пособие состоит из двух разделов: в первом разделе приведены базовые понятия организации ЭВМ, дано описание архитектуры программной учебной модели вычислительной машины и изложены принципы работы с ней, второй раздел посвящен циклу лабораторных работ.

1. АРХИТЕКТУРА ПРОГРАММНОЙ УЧЕБНОЙ МОДЕЛИ ЭВМ

1.1. Структура ЭВМ

Под *вычислительной машиной* понимают комплекс программных и технических средств, предназначенный для автоматизации подготовки и решения задач пользователей.

Архитектура вычислительной машины – это логическое построение ЭВМ. Данное понятие включает в себя перечень и формат команд, формы представления данных, механизмы ввода/вывода, способы адресации памяти и т. д.

Понятие структура ВМ включает в себя вопросы физического построения вычислительных средств: состав устройств, число регистров процессора, емкость памяти, наличие блока для обработки чисел в формате с плавающей запятой, тактовая частота центрального процессора и т. д.

В структуре учебной модели ЭВМ, изображенной на рис. 1.1, можно выделить следующие основные устройства: процессор, оперативную память (ОЗУ), сверхоперативную память (регистры общего назначения и кэш-память), устройство ввода (Увв) и устройство вывода (Увыв).

В состав процессора входят центральное устройство управления (УУ) и арифметическое устройство (АУ).

Центральное устройство управления в ЭВМ призвано выполнять следующие функции:

- выборку команд из ОЗУ в последовательности, определяемой естественным порядком выполнения программы (т.е. в порядке возрастания адресов команд в ОЗУ) или командами передачи управления;
- выборку из ОЗУ операндов, задаваемых адресами команды;
- формирование сигналов управления для выполнения операции, предписанной командой;
- определение операции, записанной в команде;
- останов или переход к выполнению следующей команды.

Структурно в составе УУ выделяются следующие устройства:

CR – регистр команды, содержащий код команды;

PC – счетчик команд, содержащий адрес текущей команды;

RB – регистр базового адреса;

SP – указатель стека, содержащий адрес вершины стека;

RA – регистр адреса, содержащий адрес при косвенной адресации.

Арифметическое устройство содержит два регистра: аккумулятор (**Acc**), в котором хранится один из операндов, и регистр операнда (**DR**), хранящий второй операнд.

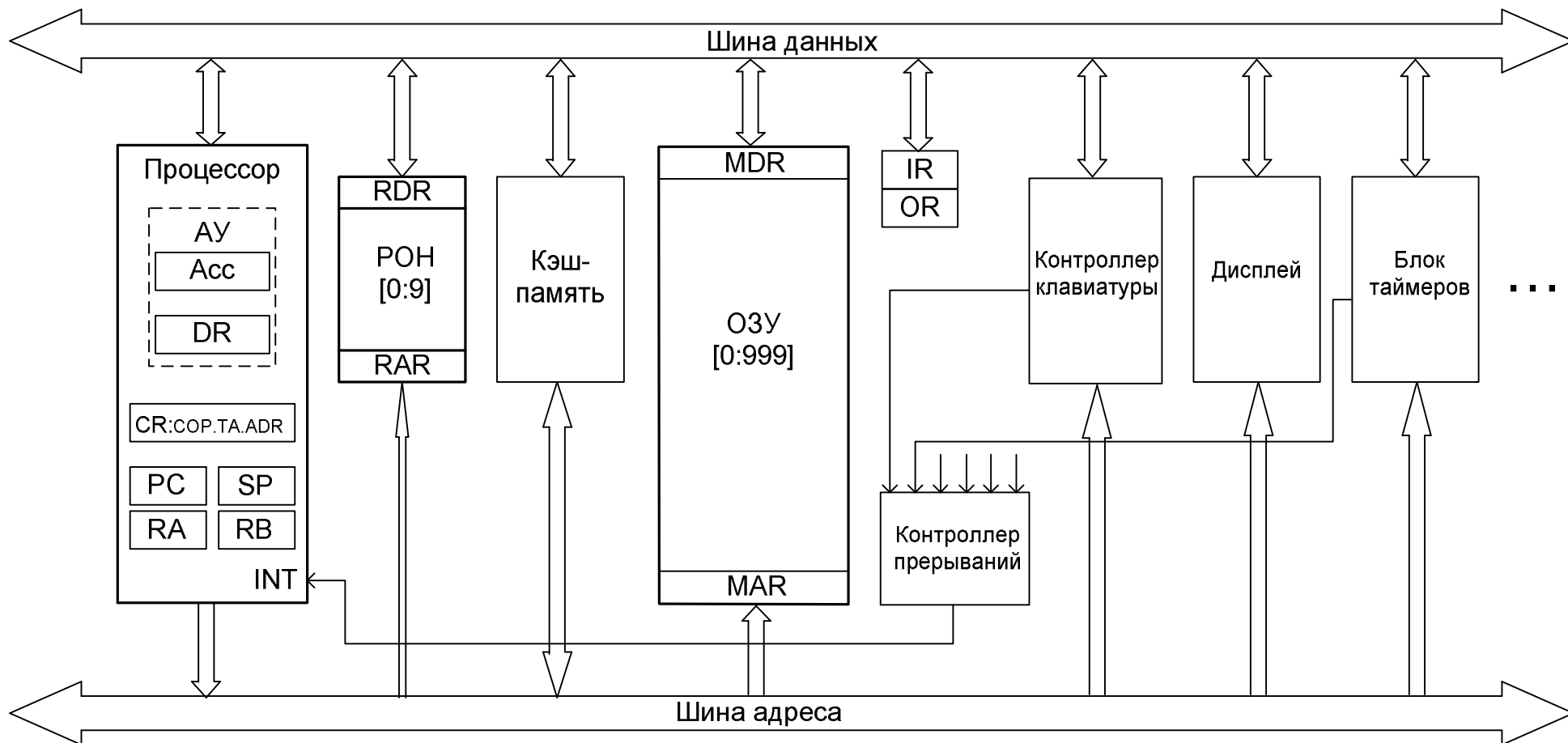


Рис. 1.1. Общая структура учебной ЭВМ

Функции АУ следующие:

- выполнение одной из арифметических операций, определяемой кодом операции (**COP**), над содержимым **Acc** и **DR**. Результат операции всегда помещается в **Acc**.

- формирование сигналов признаков результата: **Z=1**, если результат равен нулю; **S=1**, если результат отрицательный и **OV=1**, если в результате выполнения операции произошло переполнение разрядной сетки. Если эти условия не выполняются, то соответствующие сигналы имеют нулевое значение.

В ячейках *оперативной памяти* хранятся команды и данные. Емкость ОЗУ – 1000 ячеек. По сигналу MW_r выполняется запись содержимого регистра данных (**MDR**) в ячейку памяти с адресом, указанным в регистре адреса (**MAR**).

По сигналу MR_d происходит считывание содержимого ячейки памяти с адресом, указанным в **MAR**, которое передается в **MDR**.

В модели предусмотрена *сверхоперативная память* (СОЗУ) с прямой адресацией, содержащая десять регистров общего назначения (РОН) **R0 – R9**; также может быть подключена модель кэш-памяти. Доступ к РОН может быть осуществлен аналогично доступу к ОЗУ: через регистры адреса (**RAR**) и данных (**RDR**).

В эмулятор ЭВМ включены *внешние устройства* двух типов:

- регистры **IR** и **OR**, которые могут обмениваться данными с **Acc** с помощью безадресных команд $IN (Acc := IR)$ и $OUT (OR := Acc)$;

- набор моделей внешних устройств, которые могут подключаться к системе и взаимодействовать с ней в соответствии с заложенными в модели алгоритмами.

Каждое внешнее устройство имеет ряд программно-доступных регистров, может иметь собственное окно видимых элементов. Набор моделей внешних устройств описан далее в разделе 1.6.

Аккумулятор (**Acc**), регистр данных памяти (**DR**), регистры **IR** и **OR**, регистр команд (**CR**), а также все ячейки ОЗУ и РОН имеют длину 6 десятичных разрядов.

Длина же счетчика команд (**PC**), указателя стека (**SP**), регистра адреса (**RA**) и регистра базового адреса (**RB**) – 3 десятичных разряда.

1.2. Представление данных в модели

Данные в учебной ЭВМ представлены в формате целых десятичных чисел, содержащих знаковый разряд и 5 разрядов значащих цифр. Диапазон изменения чисел $-99999 \div +99999$. Формат десятичных данных ЭВМ показан на рис. 1.2.

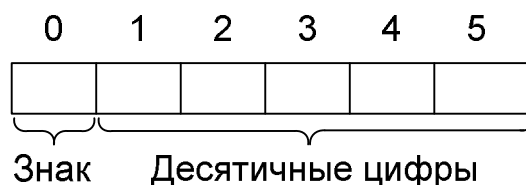


Рис. 1.2. Формат данных учебной ЭВМ

Старший разряд слова данных предназначен для кодирования знака: плюс (+) изображается как 0, минус (-) – как 1. Если результат арифметической операции выходит за рамки указанного диапазона, то происходит переполнение разрядной сетки и АЛУ вырабатывает признак результата **OV=1**. Результатом операции деления является целая часть частного. Деление на ноль вызывает переполнение.

1.3. Система команд

Система команд – совокупность всех команд, которые способна выполнить данная ЭВМ.

Система команд характеризуется тремя аспектами: форматами, способами адресации и системой операций.

1.3.1. Форматы команд

Команда определяет операцию, которую выполняет ЭВМ над данными. Команда содержит в явной или неявной форме информацию о том, где будет помещен результат операции, а также об адресе следующей команды. Код команды состоит из нескольких частей, которые называются полями.

Длина команды, количество, размер, положение, назначение и способ кодировки ее полей называется *форматом команды*.

В общем случае формат команды содержит операционную и адресную части. Операционная часть содержит код операции (например, сложение, умножение, передача данных). Адресная часть состоит из нескольких полей и содержит информацию об адресах операндов, результата операции и следующей команды. Адресная часть, в свою очередь, может состоять из двух полей (типа адресации и адреса операнда). Такой формат команды представлен на рис. 1.3.

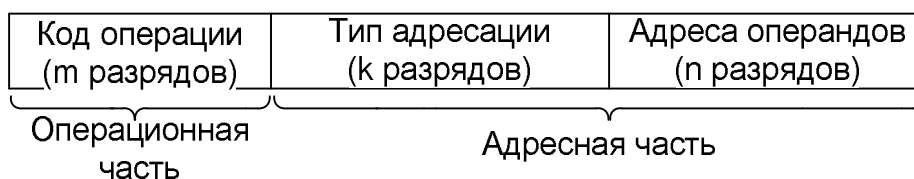


Рис. 1.3. Формат команды

Поле «тип адресации» определяет способ адресации операнда. Биты полей «тип адресации» и «адреса операндов» в совокупности определяют ячейки памяти, в которых хранятся операнды.

В современных ЭВМ выделяют следующие форматы команд: трехадресный, двухадресный, одноадресный и безадресный. Эти форматы показаны на рис. 1.4, в котором приняты следующие обозначения: КОП – код операции, A_i – адрес операнда/результата.

Команды трехадресного формата занимают много места в памяти, но при этом не все поля адресов используются в командах эффективно.

Так, наряду с двухместными операциями (сложение, деление, дизъюнкция и др.) встречаются и одноместные (сдвиг, отрицание и т.д.), для которых третий адрес не нужен. При выполнении цепочки вычислений часто результат предыдущей операции является операндом для следующей. Кроме того, нередко встречаются команды, для которых операнды не определены (СТОП).

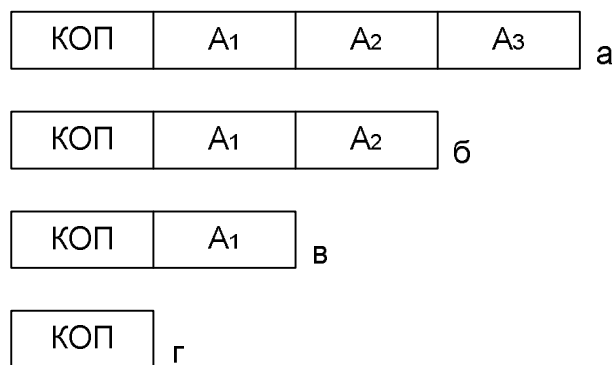


Рис. 1.4. Форматы команд ЭВМ:
а – трехадресный; б – двухадресный;
в – одноадресный; г – безадресный

Поэтому трехадресные команды в системах команд реальных ЭВМ встречаются редко. Чаще используются *двухадресные команды* (рис.1.4,б), в этом случае в двухместных операциях результат помещается на место одного из операндов.

Для реализации одноадресных форматов (рис.1.4,в) в процессоре предусмотрен специальный регистр – *аккумулятор*. Первый операнд и результат всегда размещаются в аккумуляторе, а второй операнд адресуется полем A.

Реальная система команд обычно имеет команды нескольких форматов.

В эмуляторе большинство команд – одноадресные и безадресные, длиной в одно машинное слово (6 разрядов). Исключением являются двухсловные команды с непосредственной адресацией и команда MOV,

являющаяся двухадресной. Форматы команд учебной ЭВМ показаны на рис. 1.5.

В форматах команд выделены три поля:

1. разряды [0:1] определяют код операции COP и являются старшими;
2. разряд 2 может определять тип адресации (в случае формата 5а он определяет номер регистра);
3. 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 разряды [6:11].

На рис. 1.5 приняты следующие условные обозначения:

COP – код операции;

ADR – адрес операнда в памяти;

ADC – адрес перехода;

I – непосредственный операнд;

R, R1, R2 – номер регистрации;

TA – тип адресации;

X – разряд не используется.

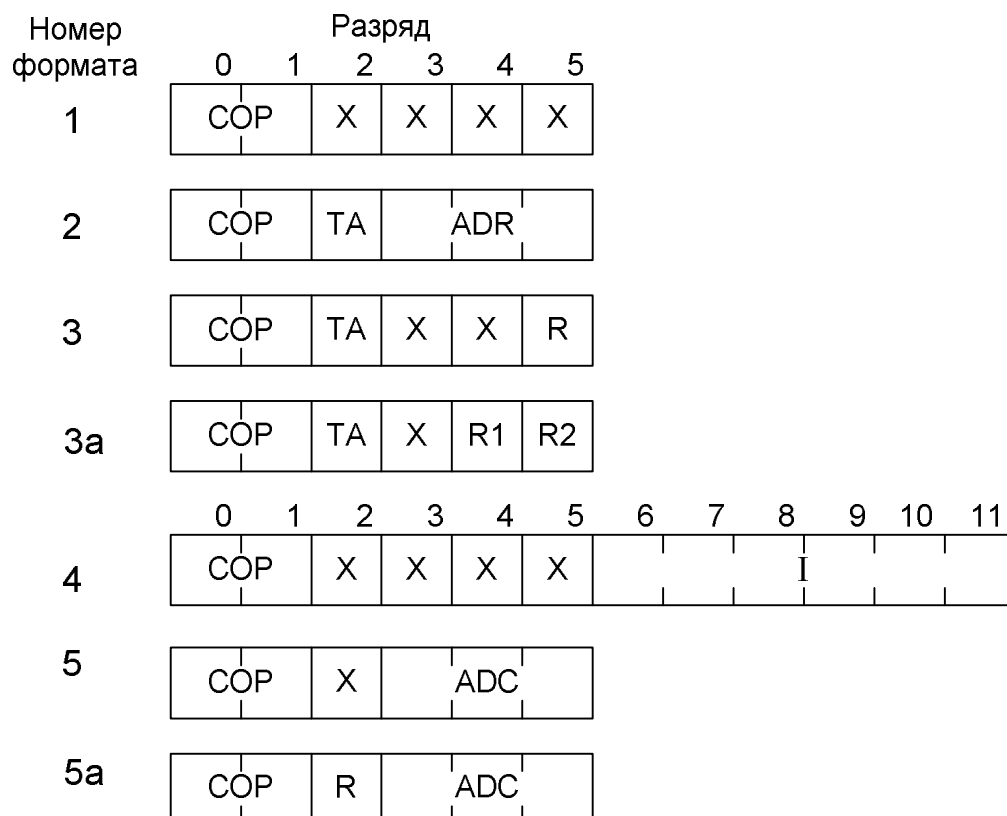


Рис. 1.5. Форматы команд учебной ЭВМ

1.3.2. Способы адресации

Способ адресации определяет, каким образом следует использовать информацию, размещенную в поле адреса команды.

В ЭВМ различают пять основных способов адресации: *прямая, косвенная, непосредственная, относительная и безадресная*. Каждый из перечисленных способов имеет свои разновидности. В модели учебной ЭВМ реализованы семь способов адресации, которые приведены в табл. 1.1.

Таблица 1.1

Адресация в командах учебной ЭВМ

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

Прямая адресация – в адресном поле располагается адрес операнда (номер ячейки памяти). Разновидностью прямой адресации является *прямая регистровая адресация*, при которой адресуется не ячейка памяти, а РОН.

Непосредственная адресация – в поле адреса команды располагается не адрес операнда, а сам операнд.

Относительная адресация – адрес формируется как сумма двух слагаемых: базы, хранящейся в специальном регистре или в одном из РОН, и смещения, извлекаемого из поля адреса команды. Разновидностью относительной адресации является *индексная адресация*. Индексная адресация предполагает наличие индексного регистра вместо базового. При каждом обращении содержимое индексного регистра обычно модифицируется автоматически: увеличивается на 1 (*индексная с постинкрементом*) или уменьшается на 1 (*индексная с прединкрементом*).

Косвенная адресация – в поле адреса команды располагается адрес ячейки памяти, в которой хранится адрес операнда («адрес адреса»). разновидностью косвенной адресации является *косвенно-регистровая адресация*, при которой в поле адреса команды размещается адрес РОН, хранящего адрес операнда.

1.3.3. Система операций

Системой операций называется список операций, непосредственно выполняемый техническими средствами ВМ. Система операций ВМ опре-

деляется областью ее применения, требованиями к стоимости, производительности и точности вычислений.

Все операции, выполняемые в командах ЭВМ принято делить на пять классов. Основные операции всех пяти классов представлены в эмуляторе ЭВМ и описаны в табл. 1.2.

Таблица 1.2

Система операций учебной ЭВМ

Класс операций	Назначение	Команды, представленные в учебной ЭВМ
арифметико-логические и специальные команды	преобразование информации	сложение, вычитание, умножение, деление
команды пересылки и загрузки	передача информации между процессором и памятью и между разными уровнями памяти (СОЗУ↔ОЗУ)	чтение, запись, пересылка (из регистра в регистр), помещение в стек, извлечение из стека, загрузка указателя стека, загрузка базового регистра
команды ввода/вывода	передача информации между процессором и внешними устройствами	ввод, вывод
команды передачи управления	изменение естественного порядка выполнения команд программы; изменение содержимого счетчика команд с обеспечением переходов по программе	безусловный и шесть условных переходов, вызов подпрограммы, цикл, программное прерывание, возврат из прерывания
системные команды	управление процессом обработки информации и внутренними ресурсами процессора	пустая операция, разрешить прерывание, запретить прерывание

Список команд учебной ЭВМ представлен в приложении в табл. 1 и 3.

1.4.Состояния и режимы работы учебной ЭВМ

Ядром устройства управления ЭВМ является управляющий автомат (УА). Это устройство предназначено для вырабатывания сигналов управления, которые инициируют работу АЛУ, РОН, ОЗУ и УВВ, передачу информации между регистрами устройств ЭВМ и действия над содержимым регистров УУ.

Учебная ЭВМ может находиться в двух состояниях **Останов** и **Работа**.

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

Команда **Пуск** запускает программу на выполнение. Программа представляет собой последовательность команд. Записанных в ОЗУ. Программа выполняется в автоматическом режиме до команды HLT (Стоп) или

точки останова. Программа выполняется по командам, начиная с ячейки ОЗУ, на которую указывает счетчик команд (**РС**), причем изменение состояний объектов модели отображается в окнах компонентов.

В состоянии **Останов** модель учебной ЭВМ переходит в результате действия команды **СТОП** или автоматически в зависимости от выбранного режима работы.

Команда **Шаг** запускает выполнение одной команды либо одной микрокоманды (если установлен Режим микрокоманд) после чего модель переходит в состояние **Останов**.

В состоянии **Останов** пользователь может просмотреть или изменить основные компоненты модели: регистры ЦП и РОН, ячейки ОЗУ, устройства ввода/вывода. В процессе изменения содержимого ОЗУ и РОН можно вводить данные для программы, в ячейки ОЗУ – программу в кодах. В данном состоянии пользователю доступны для изменения параметры модели и режимы ее работы, ввод и/или редактирование программы в мнемокодах, ассемблирование мнемокодов, выполнение основных операций с файлами.

1.5. Интерфейс пользователя и основные компоненты учебной модели ЭВМ

В программной модели учебной ЭВМ реализован стандартный Windows-совместимый интерфейс, состоящий из нескольких окон: основного окна **Модель учебной ЭВМ** и окон компонентов **Процессор**, **Память**, **Текст программы**, **Программа**, **Кэш-память**, **Микрокомандный уровень**.

1.5.1. Окно Модель учебной ЭВМ

Основное окно эмулятора **Модель учебной ЭВМ** содержит основное меню и кнопки на панели управления (рис. 1.6).

В рабочее поле окна выводятся сообщения о функционировании системы в целом. Эти сообщения группируются в файл logfile.txt (по умолчанию), автоматически сохраняющемся на диске, и могут быть проанализированы после завершения работы с моделью.

Меню содержит следующие пункты: **Файл**, **Вид**, **Внешние устройства**, **Работа**.

Пункт меню **Вид** содержит следующие команды: **Показать все**, **Скрыть все**, **Процессор**, **Микрокомандный уровень**, **Память**, **Кэш-память**, **Программа**, **Текст программы**. Эти команды открывают окна соответствующих компонентов, описанных далее.

Пункт меню **Файл** содержит команду **Выход**, позволяющую закончить работу с моделью.

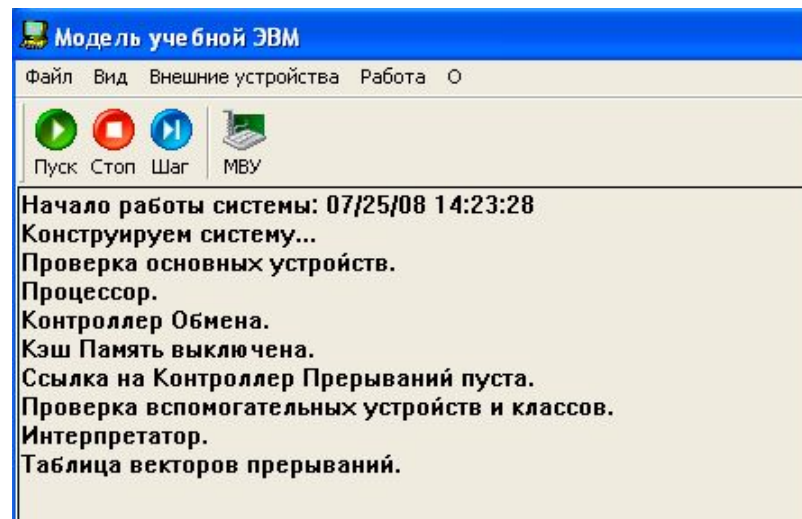


Рис1.6. Основное окно эмулятора

Пункт меню **Работа** содержит команды: **Пуск**, **Стоп**, **Шаг**, **Режим микрокоманд**, **Кэш-память**, **Настройки**. Команда **Пуск** позволяет запустить программу в автоматическом режиме, команда **Шаг** запускает программу в шаговом режиме, а команда **Стоп** позволяет остановить выполнение программы в модели процессора. Эти команды могут выполняться путем нажатия одноименных клавиш на панели управления.

Команда **Режим микрокоманд** включает или выключает микрокомандный режим работы процессора, а команда **Кэш-память** позволяет подключить или отключить модель кэш-памяти.

Команда **Настройки** открывает диалоговое окно **Параметры системы**, позволяющие установить задержку реализации командного цикла (если команда выполняется в автоматическом режиме), а также установить параметры файла logfile.txt.

1.5.2. Компонент Процессор

Окно компонента **Процессор** (рис. 1.7) позволяет получить доступ ко всем регистрам и флагам ЦП.

В модели предусмотрены программно-доступные и системные регистры и флаги.

Программно-доступные регистры и флаги следующие:

Асс – аккумулятор;

РС – счетчик команд;

SP – указатель стека;

RB – регистр базового адреса, содержащий адрес базы при относительной адресации;

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;

RDR – регистр данных блока РОН;

RAR – регистр адреса блока РОН;

MDR – регистр данных ОЗУ;

MAR – регистр адреса ОЗУ;

CR – регистр команд, содержащий три поля:

1. **COP** – поле кода операции;

2. **TA** – поле типа адресации;

3. **ADR** – поле адреса или непосредственного операнда;

OV – флаг переполнения;

S – флаг отрицательного значения **Acc**;

Z – флаг нулевого значения **Acc**.

Регистры **Acc**, **DR**, **CR**, **IR**, **OR** и все ячейки ОЗУ и РОН имеют длину шесть десятичных разрядов, а длина **SP**, **PC**, **RA** и **RB** три десятичных разряда. Компонент **Процессор** отражает текущие значения регистров и флагов ЦП. В состоянии **Останов** все регистры, включая регистры блока РОН, и флаги, кроме флага **I**, доступны для редактирования.

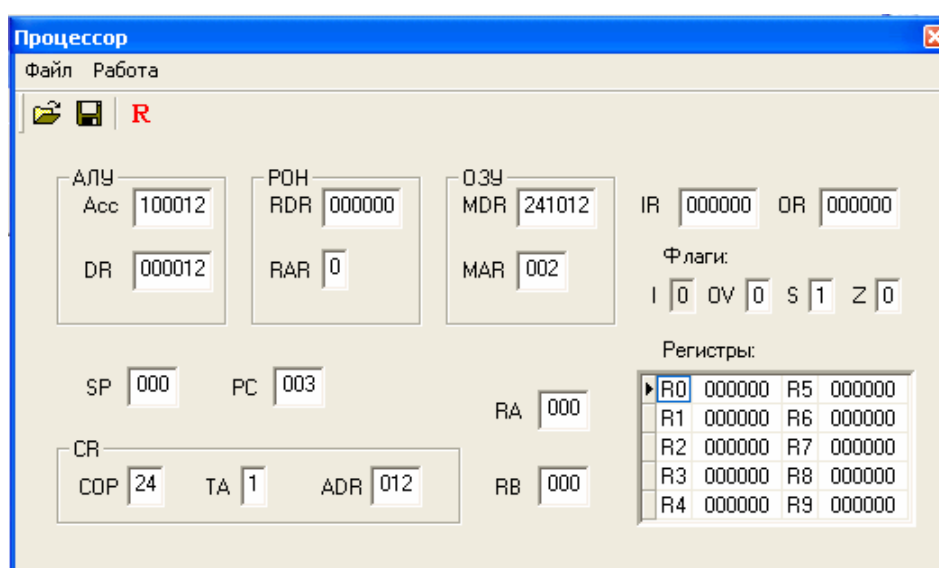


Рис.1.7. Окно компонента Процессор

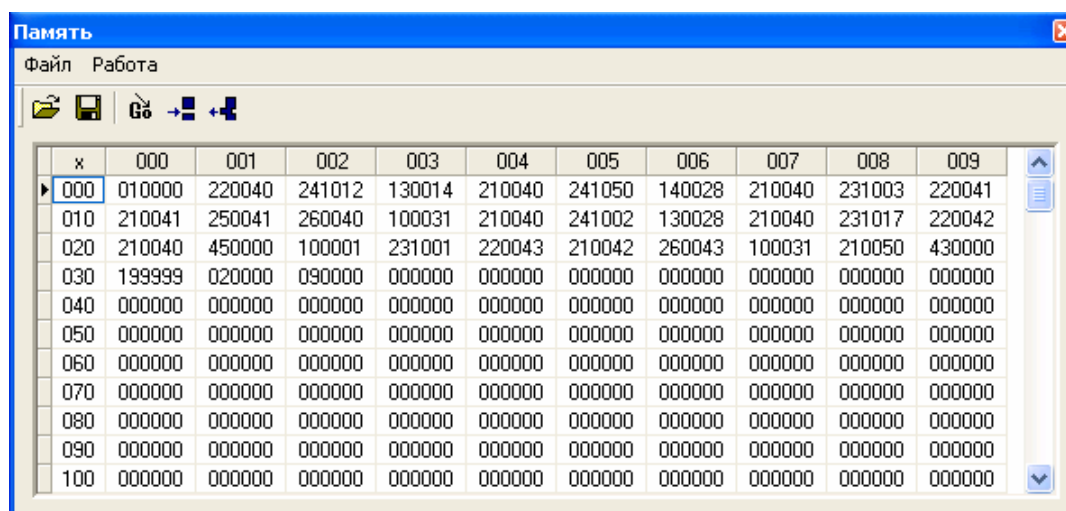
Меню компонента **Процессор** содержит пункты **Файл** и **Работа**. Пункт меню **Файл** содержит команды **Сохранить** и **Загрузить**, которые позволяют сохранить текущие значения регистров и флагов процессора в файле и восстановить состояние ЦП из файла.

Команда **Reset**, находящаяся в пункте меню **Работа** позволяет установить все регистры (в том числе блок РОН) в нулевое значение. Пункт меню **Работа** содержит также команду **Reset R0 – R9**, при выполнении которой очищаются только регистры блока РОН.

1.5.3. Компонент Память

Окно компонента **Память** (рис.1.8.) отображает текущее состояние ячеек ОЗУ. Этот компонент допускает возможность редактирования содержимого ячеек и выполнения пяти команд: **Сохранить**, **Загрузить**, **Перейти к**, **Вставить** и **Убрать**.

Команды **Сохранить** и **Загрузить** находятся в пункте меню **Файл** и сохраняют в файле текущее состояние памяти либо восстанавливают это состояние из выбранного файла. Во всех компонентах модели, где они предусмотрены, эти команды работают одинаково и файл в окнах каждого из компонентов записывается по умолчанию с характерным для этого окна расширения.



x	000	001	002	003	004	005	006	007	008	009
000	010000	220040	241012	130014	210040	241050	140028	210040	231003	220041
010	210041	250041	260040	100031	210040	241002	130028	210040	231017	220042
020	210040	450000	100001	231001	220043	210042	260043	100031	210050	430000
030	199999	020000	090000	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.1.8. Окно компонента Память

Команды **Перейти к**, **Вставить** и **Убрать** находятся в пункте меню **Работа**. Команда **Перейти к** открывает окно, позволяющее перейти к заданной ячейке ОЗУ.

Команда **Убрать** открывает диалоговое окно, 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.

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

1.5.4. Компонент Текст программы

Окно компонента **Текст программы** (рис.1.9) содержит поле текстового редактора, в котором можно редактировать текст, загружать в него текстовые файлы и сохранять текст в виде файлов.

Меню данного окна содержит стандартные для всех компонентов модели пункты **Файл** и **Работа**. В пункте меню **Файл** расположены следующие команды: **Новая**, **Загрузить**, **Сохранить**, **Сохранить как** и **Вставить**. Данные команды позволяют выполнять следующие действия:

Новая – открыть новый сеанс редактирования;

Загрузить – открыть диалоговое окно загрузки файла в окно редактора;

Сохранить – сохранить файл под текущим именем;

Сохранить как – открыть диалоговое окно сохранения файла.

Эти команды дублированы кнопками на панели управления.

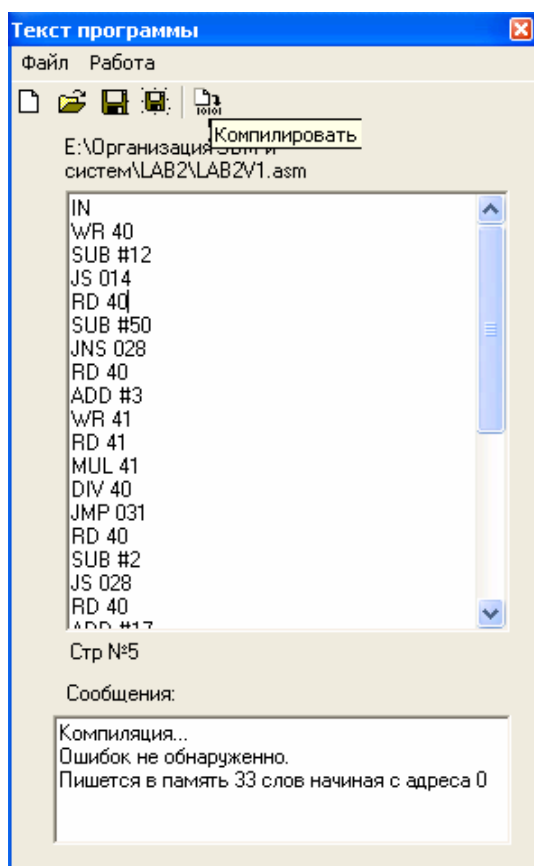


Рис.1.9. Компонент Текст программы

Команда **Вставить** позволяет вставить выбранный файл в позицию курсора.

На панели управления присутствует также кнопка **Компилировать**, которая запускает процедуру ассемблирования текста в поле редактора. Эту же команду можно запустить в пункте меню **Работа**, который содержит также команду **Адрес вставки**. Команда **Адрес вставки** позволяет задать адрес ячейки ОЗУ, начиная с которой команда будет записана в память. По умолчанию этот адрес равен 0.

Ниже области редактирования находится строка состояния, в которую выводится номер строки, определяющий положение курсора.

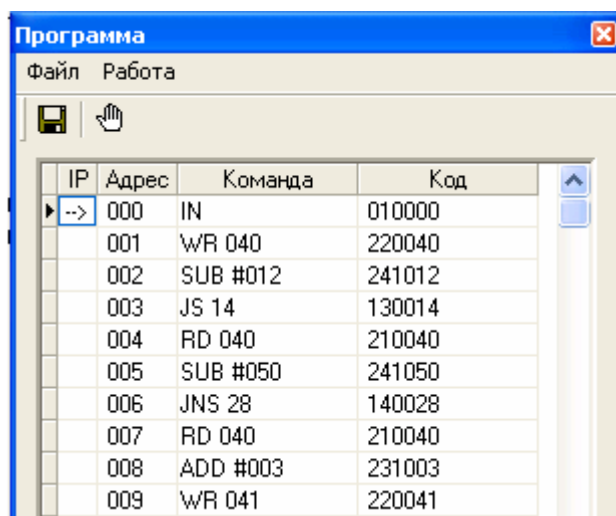
Если в процессе компиляции программы обнаруживаются синтаксические ошибки, то сообщения о них выводятся в окно сообщений и запись в память кода программы не производится.

После исправления ошибок и повторной компиляции выдается сообщение об отсутствии ошибок, о расположении программы в области памяти и о количестве слов, занятых 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 «;» до конца строки игнорируются компилятором и рассматриваются как комментарии. Строка может начинаться с «;» и, следовательно, содержать только комментарии.

1.5.5. Компонент Программа

Окно компонента **Программа** состоит из трех составляющих: стандартного для эмулятора меню, панели управления и таблицы, которая имеет 300 строк и 4 столбца (рис.1.10). Каждая строка соответствует дизассемблированной ячейке памяти. Второй столбец содержит адрес ячейки ОЗУ, третий – дизассемблированный мнемокод, четвертый – машинный код команды. В первом столбце помещается указатель «→» на текущую команду (текущее значение счетчика команд) и точка останова – красная заливка ячейки.



IP	Адрес	Команда	Код
→	000	IN	010000
	001	WR 040	220040
	002	SUB #012	241012
	003	JS 14	130014
	004	RD 040	210040
	005	SUB #050	241050
	006	JNS 28	140028
	007	RD 040	210040
	008	ADD #003	231003
	009	WR 041	220041

Рис.1.10 Окно Программа

Окно компонента **Программа** позволяет наблюдать процесс прохождения программы. Содержимое этого окна нельзя редактировать. Используя пункты меню, можно сохранить содержимое окна в виде текстового файла, выбрать начальный адрес области ОЗУ, которая будет дизассемблироваться (размер области постоянный – 300 ячеек), а также установить или снять точку останова. Установить точку останова можно тремя способами:

командой **Точка останова** из пункта меню **Работа**, одноименной кнопкой на панели управления или двойным щелчком кнопки мыши в первой ячейке соответствующей строки. Прочитать в окне компонента **Программа** ничего нельзя. Сохраненный ранее файл с расширением **.asm** можно загрузить в окно компонента **Текст программы**, ассемблировать его и тогда дизассемблированное значение заданной области памяти автоматически появится в окне компонента **Программа**. Такую процедуру удобно использовать, если программа изначально пишется или редактируется в памяти в машинных кодах.

Начальный адрес области дизассемблирования задается в диалоговом окне, которое вызывается командой **Начальный адрес** и меню **Работа**.

1.5.6. Компонент Микрокомандный уровень

Компонент **Микрокомандный уровень** (рис.1.11) используется только в режиме микрокоманд, который устанавливается командой **Режим микрокоманд** в пункте меню **Работа** основного окна **Модель учебной ЭВМ**. В окне **Микрокомандный уровень** выводится мнемокод выполняемой команды, список микрокоманд, ее реализующих и указатель на текущую выполняемую команду.

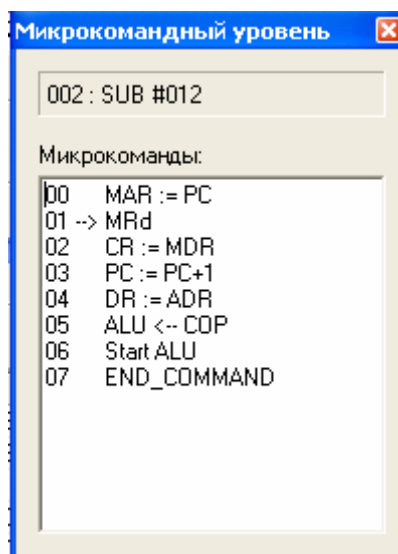


Рис.1.11 Окно Микрокомандный уровень

Чтобы просмотреть процесс выполнения программы на уровне микрокоманд необходимо чтобы модель ЭВМ работала либо в шаговом режиме либо в автоматическом режиме с задержкой командного цикла.

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»*.

1.7. Программная модель кэш-памяти

К модели учебной ЭВМ, описанной в пунктах 1.1 – 1.3, может быть подключена программная модель кэш-памяти (рис.1.12).

Кэш-память содержит N ячеек (в модели N может выбираться из множества $\{4, 8, 16, 32\}$, каждая из которых содержит:

- трехразрядное поле тега (адреса ОЗУ);
- шестиразрядное поле данных;
- три однобитовых признака (флага).

Виды флагов:

1. Z – признак занятости ячейки;
2. U – признак использования;
3. 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 использования ячейки (т.е. любое обращение к ней).

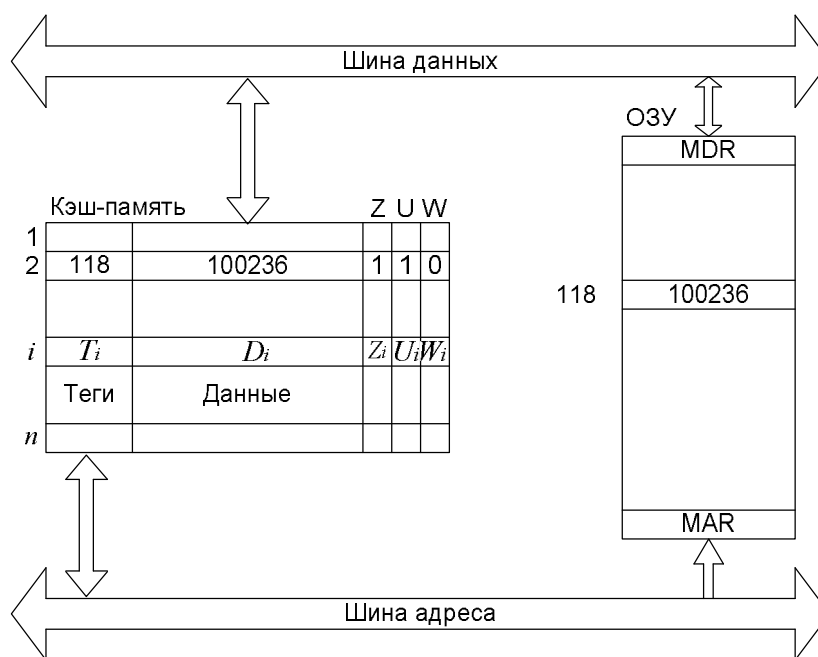


Рис.1.12. Структура модели кэш-памяти

Текущее состояние кэш-памяти отображается на экране в виде таблицы в отдельном окне, причем количество строк этой таблицы соответствует выбранному числу ячеек кэш-памяти. Столбцы таблицы определяют содержимое полей ячеек. Например так, как показано в табл. 1.3.

Таблица 1.3

Пример текущего состояния кэш-памяти

	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

Для настройки параметров кэш-памяти следует использовать диалоговое окно **Кэш-память**, которое вызывается 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.

Существует два режима записи в кэш-память: *сквозной* и *обратный*. При сквозной записи в случае кэш-попадания 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 поле данных в соответствующую ячейку ОЗУ.

При кэш-промахе следует поместить в кэш-памяти 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 помощью *алгоритма замещения строк*.

В модели учебной ЭВМ реализованы три алгоритма замещения строк:

- *произвольный выбор*, 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.

Бит использования устанавливается в 1 при любом обращении к ячейке, но, как только все биты U_i установятся в 1, все они тут же сбрасываются в 0, так что в кэше всегда ячейки разбиты на два непересекающихся подмножества по значению бита U . Первое подмножество составляют те ячейки, обращение к которым состоялось относительно недавно (после последнего сброса вектора U), эти ячейки имеют значение $U=1$. Во втором подмножестве находятся ячейки со значением $U=0$, которые являются «кандидатами на удаление» при использовании алгоритма замещения «*бит использования*».

Если в параметрах модели кэш-памяти установлен флаг «с учетом бита записи», то все три алгоритма замещения осуществляют поиск «кандидата на удаление» прежде всего из тех ячеек, признак записи которых не установлен, а при отсутствии таких ячеек – среди всех ячеек кэш-памяти. При снятом флаге «с учетом бита записи» поиск осуществляется по всем ячейкам кэш-памяти без учета значения W .

Оценка эффективности работы системы с кэш-памятью определяется числом кэш-попаданий по отношению к общему числу обращений к памяти. Учитывая разницу в алгоритмах записи, эффективность использования кэш-памяти вычисляется по следующим выражениям:

– для сквозной записи:

$$K = \frac{S_K - S_{KW}}{S_0}, \quad (1.2)$$

– для обратной записи:

$$K = \frac{S_K - S_{KW}^i}{S_0}, \quad (1.3)$$

где:

K – коэффициент эффективности работы кэш-памяти;

S_0 – общее число обращений к кэш-памяти;

S_K – число кэш-попаданий;

S_{KW} – число сквозных записей при кэш-попадании (в режиме сквозной записи);

S_{KW}^i – число обратных записей (в режиме обратной записи).

2. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

2.1. Лабораторная работа №1

Архитектура ЭВМ и система команд

Цель работы – знакомство с интерфейсом модели ЭВМ, методами ввода и отладки программы, действиями основных классов команд и способов адресации.

2.1.1 Общие положения

Все вычисления, предусмотренные алгоритмом решения задачи, должны быть представлены в виде программы. Программа на языке ЭВМ состоит из управляющих слов – команд. Каждая команда предписывает некоторую операцию из набора операций, реализуемых вычислительной машиной. Код каждой команды содержит код операции, тип адресации и адрес. Команды программы хранятся в последовательных ячейках памяти

ВМ и выполняются в естественном порядке, т.е. в порядке их положения в программе. При необходимости, с помощью специальных команд передачи управления, эта последовательность может быть изменена. Решение об изменении порядка выполнения команд программы принимается на основании анализа результатов предшествующих вычислений, либо безусловно.

Для того чтобы получить результат выполнения программы, пользователь должен:

- ввести программу в память ЭВМ;
- определить, если это необходимо, содержимое ячеек ОЗУ и РОН, содержащих исходные данные, а также регистров **IR** и **OR**;
- установить в счетчик команд (**PC**) стартовый адрес программы;
- перевести модель в режим **Работа**.

Каждое из этих действий выполняется с помощью интерфейса эмулятора, описанного в разделе 1. Ввод программы может осуществляться как в машинных кодах непосредственно в память модели, так и в мнемокодах в окно **Текст программы** с последующим ассемблированием.

Для достижения цели, поставленной в начале лабораторной работы, необходимо ввести в память ЭВМ и выполнить в режиме **Шаг** некоторой последовательности команд (определенную вариантом задания) и зафиксировать все изменения на уровне программно-доступных объектов ЭВМ, происходящих при выполнении этих команд.

Команды в память учебной ЭВМ вводятся в виде шестиразрядных десятичных чисел (см. форматы команд на рис. 1.5, коды команд и способы адресации в табл. 1.2 – 1.4).

В данной лабораторной работе ЭВМ программируется в машинных кодах.

2.1.2. Порядок выполнения лабораторной работы

1. Ознакомиться с архитектурой модели учебной ЭВМ (разд. 1).
2. Согласно варианту задания (табл. 2.1) составить таблицу соответствия команд и кодов (табл. 2.2). Записать в ОЗУ «программу», состоящую из пяти команд. 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. 2.3).
6. Если в программе образуется цикл, необходимо посмотреть не более двух повторений каждой команды, входящей в тело цикла.

Таблица 2.1

Варианты задания

№	IR	Команда 1	Команда 2	Команда 3	Команда 4	Команда 5
0	—	RD #20	WR 30	ADD #5	WR @30	JNZ 002
1	000007	IN	MUL #2	WR 10	WR @10	JNS 001
2	—	RD #17	SUB #9	WR 16	WR @16	JNS 001
3	100029	IN	ADD #16	WR 8	WR @8	JS 001
4	—	RD #2	MUL #6	WR 11	WR @11	JNZ 00
5	000016	IN	WR 8	DIV #4	WR @8	JMP 002
6	—	RD #4	WR 11	RD @11	ADD #330	JS 000
7	000000	IN	WR 9	RD @9	SUB #1	JS 001
8	—	RD 4	SUB #8	WR 8	WR @8	JNZ 001
9	100005	IN	ADD #12	WR 10	WR @10	JS 004
10	—	RD 4	ADD #15	WR 13	WR @13	JMP 001
11	000315	IN	SUB #308	WR 11	WR @11	JMP 001
12	—	RD #988	ADD #19	WR 9	WR @9	JNZ 001
13	000017	IN	WR 11	ADD 11	WR @11	JMP 002
14	—	RD #5	MUL #9	WR 10	WR @10	JNZ 001

Примечание: знак «—» в поле IR таблицы означает, что начальное значение регистра **IR** не задано.

1.2.3. Пример выполнения задания

Дана последовательность мнемочкодов (см. вариант 0 в табл.2.1), которую необходимо преобразовать в машинные коды, занести в ОЗУ ЭВМ, выполнить в режиме **Шаг** и зафиксировать изменение состояний программно-доступных объектов ЭВМ.

Для преобразования заданных мнемочкодов в машинные коды воспользуемся табл. 1, 2 приложения и составим таблицу соответствия (табл. 2.2).

Например, Команда 1 имеет вид RD #20. Согласно табл. 1 приложения, которая содержит команды учебной ЭВМ переведем команду RD в машинный код: RD:=21. Далее следует тип адресации #, который согласно табл. 2 приложения в машинных кодах имеет код: #:=1. Затем в команде следует номер ячейки памяти ОЗУ, которую согласно формату данных учебной ЭВМ следует привести к трехзначному числу 020. Следовательно, Команда 1 в машинных кодах имеет вид: 21 1 020. Аналогично переводим в машинные коды остальные четыре команды.

Таблица 2.2

Команды и коды

Последовательность	Значения				
Команды	RD #20	WR 30	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 нашем случае это аккумулятор **Асс**, счетчик команд **РС** и ячейки ОЗУ с номерами 020 и 030) в табл. 2.3.

Таблица 2.3

Содержимое программно-доступных объектов

РС	Асс	М (030)	М (020)	РС	Асс	М (030)	М (020)
000	000000	000000	000000	004	–	–	–
001	000020	–	–	002	–	–	–
002	–	000020	–	003	000030	–	–
003	000025	–	–	004	–	–	000030

2.1.4. Содержание отчета

1. Постановка цели лабораторной работы, формулировка варианта задания.
2. Таблица соответствия команд программы, записанной в задании, и машинных кодов этих команд в форме табл. 2.2.
3. Результаты выполнения последовательности команд в форме табл. 2.3.
4. Выводы по проделанной работе.

2.1.5. Контрольные вопросы

1. Из каких основных элементов 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 отличия между ними?
6. Какие ограничения накладываются на способ представления данных в модели?
7. Какие режимы работы предусмотрены в модели и в чем отличия между ними?
8. Как записать программу в машинных кодах в память модели ЭВМ?
9. Какие способы адресации операндов применяются в командах ЭВМ?
10. Какие команды относятся к классу передачи управления?

2.2. Лабораторная работа №2

Исследование команд передачи управления

Цель работы – разработка и отладка программы, содержащей ветвления; изучение организации команд условной передачи управления.

2.2.1. Общие положения

Концепция фон-неймановской ЭВМ предполагает, что команды программы, как правило, выполняются в порядке их расположения в памяти. Для получения адреса очередной команды достаточно увеличить содержимое счетчика команд на длину выполняемой команды. В то же время, основные преимущества ВМ заключаются именно в возможности изменения хода вычислений в зависимости от возникающих в процессе счета результатов. С этой целью в АСК вычислительной машины включаются команды, позволяющие изменить естественный порядок выполнения команд в программе и передать управление в другую точку программы.

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

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

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

Условный переход происходит только при соблюдении определенного условия, в противном случае выполняется следующая по порядку команда программы.

Условием, на основании которого происходит переход, чаще выступают признаки результата предшествующей арифметической или логической операции. Каждый из признаков фиксируется в своем разряде регистра флагов ЦП. Возможен и иной подход, когда решение о переходе принимается в зависимости от состояния одного из регистров общего назначения, куда предварительно помещается результат операции сравнения. Третий вариант – это объединение операций сравнения и перехода в одной команде.

В системе команд ВМ для каждого признака результата предусматривается своя команда ветвления (иногда – две: переход при наличии признака и переход при его отсутствии).

Большая часть условных переходов связана с проверкой взаимного соотношения двух величин или с равенством (неравенством) некоторой величины нулю.

Одной из форм команд условного перехода являются команды пропуска. В них адрес перехода отсутствует, а при выполнении условия происходит пропуск следующей команды, т.е. предполагается, что отсутствующий в команде адрес следующей команды эквивалентен адресу текущей команды, увеличенному на длину пропускаемой команды. Такой прием позволяет сократить длину команд передачи управления.

2.2.2. Порядок выполнения лабораторной работы

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. 2.4, варианты заданий – в табл. 2.5.

2. Исходя из допустимых пределов изменения аргумента функций (табл. 2.4) и значения параметра a для своего варианта задания (табл. 2.5) 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 199999.

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 адрес останова) в автоматическом режиме. В случае наличия ошибки выполнить пп. 5,г и 5,д; иначе перейти к п. 5,е;

г) записать в **PC** стартовый адрес программы;

д) наблюдая выполнение программы в режиме **Шаг**, найти команду, 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 отлаженной программы в режиме **Шаг** и записать в форме табл. 2.3 содержимое программно-доступных объектов ЭВМ после выполнения каждой команды.

Таблица 2.4

Функции и допустимые пределы изменения аргумента

k	$F_k(x)$	k	$F_k(x)$
1	$\frac{x+17}{x-1}; 2 \leq x \leq 12$	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	$\frac{8100}{x^2}; 1 \leq x \leq 90$

Таблица 2.5

Варианты задания

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

2.2.3. Пример выполнения задания

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

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

причем x вводится с устройства ввода **IR**, результат y выводится на **OR**.

Блок-схема алгоритма решения задачи показана на рис.2.1.

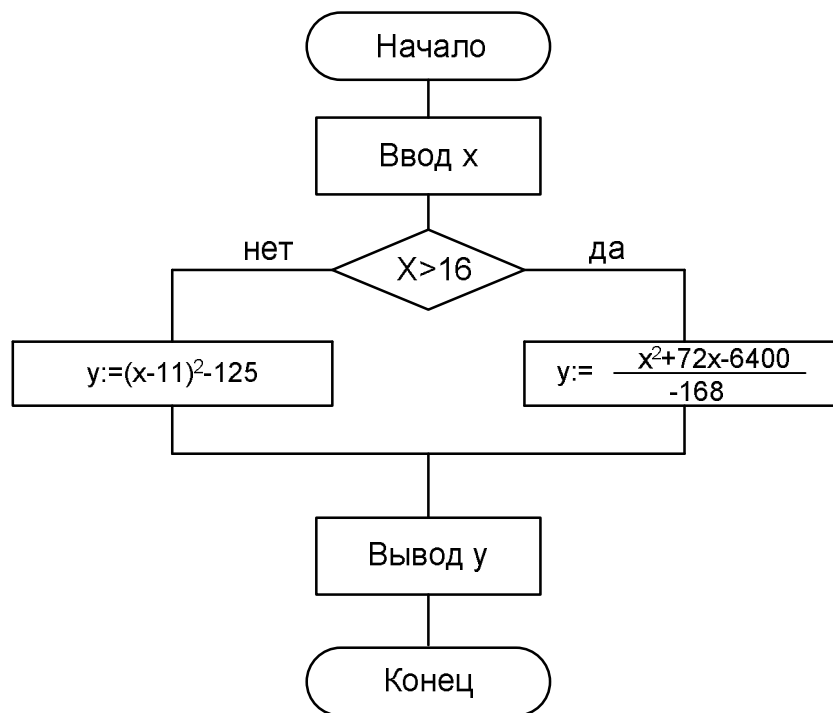


Рис.2.1. Блок-схема алгоритма

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

Оценивая 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. 2.6.

Таблица 2.6

Программа

Адрес	Команда		Комментарии
	Мнемокод	Код	
000	IN	01 0000	Ввод x , т. е. запись x в Асс
001	WR 30	22 0 030	Размещение x в ОЗУ, в ячейке с адресом 30
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	Вычисление по первой формуле. Считывание данных из ячейки с адресом 30
005	SUB #11	24 1 011	$(x - 11)$
006	WR 31	22 0 031	Запись результата предыдущей команды в ячейку с адресом 31
007	MUL 31	25 0 031	$(x - 11)^2$
008	SUB #125	24 1 125	$(x - 11)^2 - 125$

Адрес	Команда		Комментарии
	Мнемокод	Код	
009	JMP 020	10 0 020	Переход на вывод результата
010	RD 30	21 0 030	Вычисление по второй формуле. Считывание данных из ячейки с адресом 30
011	MUL 30	25 0 030	x^2
012	WR 31	22 0 031	Запись результата предыдущей команды в ячейку с адресом 31
013	RD 30	21 0 030	Считывание значения из ячейки с адресом 30
014	MUL #72	25 1 072	$72x$
015	ADD 31	23 0 031	$x^2 + 72x$
016	ADI 106400	43 0 000	$x^2 + 72x + (-6400)$
017		106400	
018	DIVI 100168	46 0 000	$\frac{x^2 + 72x - 6400}{-168}$
019		100168	
020	OUT	02 0 000	Вывод результата
021	HLT	09 0 000	Стоп

Примечание: команды ADI, SUBI, MULI, и DIVI занимают в памяти модели учебной ЭВМ две ячейки.

2.2.4. Содержание отчета

1. Формулировка цели работы и варианта задания.
2. Блок-схема алгоритма решения задачи.
3. Размещение данных в ОЗУ
4. Программа в форме табл. 2.6.
5. Последовательность состояний программно-доступных объектов ЭВМ при выполнении программы в режиме Шаг для одного значения аргумента в форме табл. 2.3.
6. Результаты выполнения программы для нескольких значений аргумента, выбранных самостоятельно.
7. Выводы по проделанной работе.

2.2.5. Контрольные вопросы

1. К какому типу архитектуры ВМ относится программная учебная модель ЭВМ и почему?
2. Какие виды команд условного перехода обычно доминируют в реальных программах?
3. Как работают команды передачи управления?
4. Что входит в понятие «отладка программы»?

5. Какие способы отладки можно реализовать в модели?
6. В чем отличие между командами `MUL 30` и `MUL #72` в программе примера?
7. Почему в ячейке с адресом 016 использована команда `ADI` а не `ADD`?
8. Укажите местонахождение операнда с прямой адресацией?
9. Объясните, как определяется значение операнда с непосредственной адресацией.

2.3. Лабораторная работа №3

Механизм косвенной адресации

Цель работы – разработка и отладка программы, содержащей цикл с переадресацией; изучение механизма косвенной адресации.

2.3.1. Общие положения

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.

Многие команды предполагают чтение операндов из памяти или запись в память. В простейшем случае в адресном поле таких команд явно указывается исполнительный адрес в соответствующей ячейке основной памяти. Однако, часто используется и другой способ указания адреса, когда адрес операнда хранится в какой-то ячейке памяти, а в команде адрес ячейки, содержащий адрес операнда. Подобный прием носит название косвенной адресации. Чтобы прочитать или записать операнд, сначала необходимо извлечь из памяти его адрес и только после этого произвести нужное действие (чтение или запись операнда), т. е. требуется выполнить два обращения к памяти.

Цикл команды, в котором появляется косвенная адресация, претерпевает следующие изменения: содержимое адресного поля в регистре команд используется для обращения к ячейке ОП, в которой храниться адрес операнда, после чего извлеченный из памяти исполнительный адрес операнда помещается в адресное поле регистра команды на место косвенного адреса. Дальнейшее выполнение команды протекает стандартным образом.

2.3.2. Порядок выполнения лабораторной работы

1. Написать программу определения заданной характеристики последовательности чисел C_1, C_2, \dots, C_n . Варианты заданий приведены в табл. 2.7.

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

3. Сохранить набранную программу в виде текстового файла и произвести ассемблирование мнемокодов.

4. Загрузить в ОЗУ необходимые константы и исходные данные.

5. Отладить программу.

Таблица 2.7

Варианты задания

Номер варианта	Характеристика последовательности чисел $C_1, C_2, \dots, C_n; n$
1	Количество четных чисел; $n = 10$
2	Номер минимального числа; $n = 8$
3	Произведение всех чисел; $n = 12$
4	Номер первого отрицательного числа; $n = 11$
5	Количество чисел, равных C_1 ; $n = 6$
6	Количество отрицательных чисел; $n = 16$
7	Максимальное отрицательное число; $n = 7$
8	Номер первого положительного числа; $n = 15$
9	Минимальное положительное число; $n = 14$
10	Номер максимального числа; $n = 10$
11	Количество нечетных чисел; $n = 12$
12	Количество чисел, меньших C_1 ; $n = 8$
13	Разность сумм четных и нечетных элементов массива; $n = 11$
14	Отношение сумм четных и нечетных элементов массива; $n = 6$

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

2.3.3. Пример выполнения задания

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. Обязательно 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, элемента 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 происходит следующим образом. Программа размещена 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. Блок-схема алгоритма программы показана на рис. 2.2, текст программы с комментариями приведен в табл. 2.8.

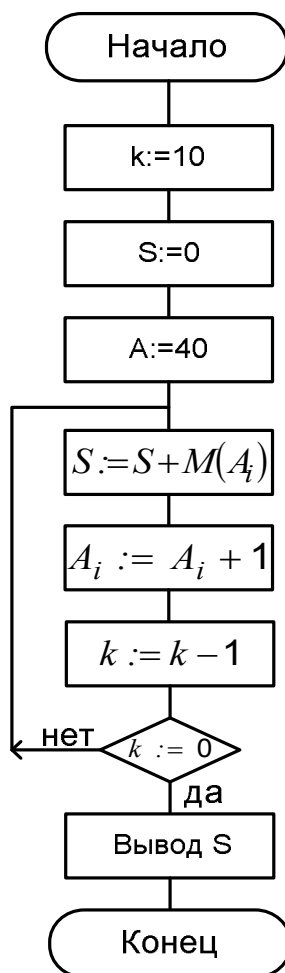


Рис.2.2. Блок-схема алгоритма

Текст программы

Адрес	Команда	Комментарии
000	RD #40	Загрузка начального адреса массива
001	WR 30	в ячейку ОЗУ с адресом 030
002	RD #10	Загрузка параметра цикла $k = 10$ в
003	WR 31	ячейку ОЗУ с адресом 031
004	RD #0	Загрузка начального значения суммы $S = 0$ в
005	WR 32	ячейку ОЗУ с адресом 032
006	M1: RD 32	Добавление к текущей сумме очередного элемента
007	ADD @30	массива
008	WR 32	
009	RD 30	Модификация текущего адреса массива (переход к
010	ADD #1	следующему адресу)
011	WR 30	
012	RD 31	Уменьшение счетчика (параметра цикла)
013	SUB #1	на 1
014	WR 31	
015	JNZ M1	Проверка параметра цикла и переход при $k \neq 0$
016	RD 32	Вывод результата
017	OUT	
018	HLT	Стоп

2.3.4. Содержание отчета

Отчет по лабораторной работе должен содержать следующие разделы:

1. Формулировка цели работы и варианта задания
2. Блок-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
6. Выводы по проделанной работе

2.3.5. Контрольные вопросы

1. Какие функции в вычислительной машине выполняет устройство управления?
2. Чем отличается аккумулятор от других регистров ЦП?
3. Каким образом определяется адрес операнда с косвенной адресацией?
4. Дайте определение понятию «цикл». Какие виды циклических структур вы знаете?
5. Как организовать цикл в программе?
6. Что такое параметр цикла?

2.4. Лабораторная работа №4

Подпрограммы и стек

Цель работы – изучение организации программ с использованием подпрограмм, а также принципа работы сверхоперативной памяти (РОН - регистров общего назначения).

2.4.1. Общие положения

В программировании часто встречаются ситуации, когда одинаковые действия необходимо выполнять многократно в разных частях программы (например, вычисление функции $\cos x$). При этом с целью экономии памяти не следует несколько раз повторять одну и ту же последовательность команд – достаточно один раз написать подпрограмму и обеспечить правильный вызов этой подпрограммы и возврат в точку ее 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 счетчика команд – **PC**) при вызове и использовании в конце подпрограммы команды возврата `RET`, которая возвращает сохраненное значение адреса возврата в **PC**.

Для реализации механизма вложенных подпрограмм (возможность вызова подпрограммы из другой подпрограммы и т. д.) адреса возврата следует сохранять в стеке. Стек – особым образом организованная память, доступ к которой осуществляется через единственную ячейку – вершину стека. Работа стека базируется на принципе LIFO (Last In First Out, последним записан – первым считан). При записи слово помещается в вершину стека, предварительно находящиеся в нем слова смещаются на одну позицию, при чтении извлекается содержимое вершины стека (оно при этом из стека исчезает), а все оставшиеся слова смещаются вверх на одну позицию.

В обычных ОЗУ нет возможности перемещать слова между ячейками, поэтому при организации стека перемещается не массив слов относительно неподвижной вершины, а вершина относительно неподвижного массива. Под стек отводится некоторая область ОЗУ, причем адрес вершины хранится в специальном регистре ЦП – указателе стека **SP** (stack pointer).

В стек можно поместить содержимое регистра общего назначения по команде `PUSH` или извлечь содержимое вершины в РОН по команде `POP`. Кроме этого, по команде вызова подпрограммы `CALL`, значение программ-

ного счетчика **PC** (адрес следующей команды) помещается в вершину стека, а по команде **RET** содержимое вершины стека извлекается в **PC**. При каждом обращении к стеку указатель **SP** автоматически модифицируется.

В большинстве ЭВМ стек «растет» в сторону меньших адресов, поэтому перед каждой записью содержимое **SP** уменьшается на единицу, а после каждого извлечения содержимое **SP** увеличивается на 1. Таким образом, **SP** всегда указывает на вершину стека.

В данной лабораторной работе при организации циклов используется сверхоперативная память – РОН. В реальных ЭВМ доступ к РОН занимает значительно меньше времени, чем к ОЗУ. Также 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.

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 прямой и косвенной адресации в регистровых командах используется постинкрементная и преддекрементная (табл. 1.6). К регистровым также относится команда функции цикла **JRNZ, R, M**. По этой команде содержимое указанного в команде регистра уменьшается на единицу, и если в результате вычитания содержимое регистра на равно нулю, то управление передается на метку **M**. Эту команду следует ставить в конце тела цикла, метку **M** – в первой команде тела цикла, а в регистр **R** помещать число повторений цикла.

2.4.2. Порядок выполнения лабораторной работы

1. Составить 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 (табл. 2.7), а соответствие между номерами вариантов заданий 3 и 4 устанавливается по табл. 2.9.

2. Загрузить в ОЗУ необходимые константы и исходные данные.

3. Выполнить программу в режиме **Работа** и при необходимости отладить ее.

Таблица 2.9

Соответствие между номерами заданий

Номер варианта задания 4	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Номер варианта в табл. 2.7	5	7	13	11	9	12	1	10	14	3	6	8	2	4

2.4.3. Пример выполнения задания

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

Программа будет состоять из основной части и подпрограммы, т. к. в программе трижды необходимо выполнить поиск максимального элемента массива.

Основная часть программы 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 РОН **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. регистр **R3** используется для хранения текущего максимума, а **R4** – для временного хранения текущего элемента. Подпрограмма возвращает результат через аккумулятор. Цикл в программе реализован с помощью команды JRNZ, а модификация текущего адреса – средствами постинкрементной адресации.

Ниже (табл. 2.10), приведен текст основной программы и подпрограммы для конкретной задачи. Первый массив расположен, начиная с адреса 085, и имеет длину 14 элементов, второй массив – 100 и 4, третий – 110 и 9.

Таблица 2.10

Текст программы

Команда	Комментарии
Основная часть программы	
RD #85 WR R1 RD #14 WR R2	Загрузка параметров первого массива
CALL M	Вызов подпрограммы
WR R6	Сохранение результата
RD #100 WR R1	Загрузка параметров второго массива
RD #4 WR R2	
CALL M	Вызов подпрограммы
WR R7	Сохранение результата
RD #110 WR R1 RD #9 WR R2	Загрузка параметров третьего массива

Команда	Комментарии
CALL M	Вызов подпрограммы
ADD R7 ADD R6 DIV #3	Вычисление среднего арифметического
OUT	Вывод результата
Подпрограмма	
HLT	Стоп
M: RD @R1 WR R3	Загрузка первого элемента массива в регистр R3
L2: RD @R1+ WR R4	Чтение элемента и модификация адреса
SUB R3 JS L1 MOV R3, R4	Сравнение и замена, если R3<R4
L1: JRNZ R2, L2	Цикл
RD R3	Чтение результата в Асс
RET	Возврат из подпрограммы

2.4.4. Содержание отчета

1. Формулировка цели работы и варианта задания.
2. Блок-схема алгоритма основной программы.
3. Блок-схема алгоритма подпрограммы.
4. Распределение памяти (размещение в ОЗУ переменных, программы и необходимых констант).
5. Тексты программы и подпрограммы в форме табл. 2.10.
6. Значение исходных данных и результат выполнения программы.
7. Выводы по проделанной работе.

2.4.5. Контрольные вопросы

1. Дайте определение стековой памяти?
2. Каково назначение стековой памяти?
3. Какие типы стековой памяти вы знаете?
4. Объясните назначение регистра **SP**?
5. Какие операции выполняет процессор по командам CALL, RET, PUSH, POP, PUSH A, POP A?
6. Как работает команда MOV R3, R4 в программе примера?

2.5. Лабораторная работа №5

Цикл команды

Цель работы – изучение организации цикла команды, а также реализации его действия на микрокомандном уровне.

2.5.1. Общие положения

Программа в фон-неймановской ЭВМ реализуется центральным процессором посредством последовательного исполнения образующих эту программу команд. Действия, требуемые для выборки (извлечения из основной памяти) и выполнения команды, называют *циклом команды*. В общем случае цикл команды включает в себя несколько этапов:

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

Перечисленные этапы выполнения команды образуют *стандартный цикл команды*. Однако, не все из этапов присутствуют при выполнении любой команды (зависит от типа команды), тем не менее этапы выборки, формирования адреса следующей команды, декодирования и исполнения операции имеют место всегда.

В свою очередь, каждая команда 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 компоненте **Микрокомандный уровень** (рис. 1.11).

2.5.2. Порядок выполнения работы

1. Выполнить последовательность команд по варианту задания из лабораторной работы №1 (см. табл. 2.1) в режиме **Шаг**.

2. Зарегистрировать изменения состояния процессора и памяти в форме табл. 2.11, в которой приведены состояния учебной ЭВМ при выполнении фрагмента примера к лабораторной работе №1.

3. Запишите последовательность микрокоманд для следующих команд эмулятора:

- | | |
|---------------|-----------------|
| – ADD R3; | – JMP M; |
| – ADD @R3; | – CALL M; |
| – ADD @R3+; | – RET: PUSH R3; |
| – ADD -@R3; | – MOV R4, R2; |
| – JRNZ R3, M; | – POP R5. |

2.5.3. Содержание отчета

1. Формулировка цели работы и варианта задания.
2. Состояния процессора и памяти модели ЭВМ в форме табл. 2.11.
3. Последовательность микрокоманд для команд модели п.3 задания в форме таблицы:

Команда	Последовательность микрокоманд

4. Выводы по проделанной работе.

2.5.4. Контрольные вопросы

1. Какие микрокоманды связаны с изменением состояния аккумулятора?
2. Какие регистры процессора участвуют в реализации этапа выборки команд?
3. Укажите назначение регистра команд; аккумулятора; блока РОН?
4. Из чего состоит командный цикл?
5. Назовите возможные типы машинных циклов.
6. Какие действия выполняет ЦП при поступлении запроса прерывания в режиме прерывания?
7. Какие из этапов цикла команды являются обязательными для всех команд?
8. Местоположение какого из этапов цикла команды в общей их последовательности в принципе может быть изменено?

2.6. Лабораторная работа №6

Алгоритмы замещения строк кэш-памяти

Цель работы – проверка работы различных алгоритмов замещения при различных режимах записи.

2.6.1. Общие положения

В разд. 1.7 данного методического пособия описаны алгоритмы замещения строк в памяти, реализованные в модели учебной ЭВМ. Рассмотрим подробнее алгоритмы согласования содержимого кэш-памяти и основной памяти (ОП).

В процессе вычислений ЦП может не только считывать имеющуюся информацию, но и записывать новую, обновляя тем самым содержимое кэш-памяти. С другой стороны, многие устройства ввода/вывода умеют напрямую обмениваться информацией с ОП. В обоих случаях возникает ситуация, когда содержимое строки КЭШа и соответствующего блока ОП перестает совпадать. В результате на связанное с ОП устройство вывода

может быть выдана «устаревшая» информация, т.к. все изменения в ней, сделанные процессором, фиксируются только в кэш-памяти, а процессор будет использовать старое содержимое кэш-памяти вместо новых данных, загруженных в ОП из устройства ввода.

Для разрешения ситуации, когда процессор выполняет операцию записи, в системах с кэш-памятью предусмотрены методы обновления основной памяти, которые можно разбить на две группы: *метод сквозной записи* и *метод обратной записи*.

По методу сквозной записи прежде всего обновляется слово, хранящееся в основной памяти. Если в кэш-памяти существует копия этого слова, то она также обновляется. Если в кэше отсутствует нужная копия, то либо из основной памяти в кэш-память переносится блок, содержащий обновленное слово (*сквозная запись с отображением*), либо этого не делается (*сквозная запись без отображения*). Главное достоинство метода сквозной записи состоит в том, что когда строка в кэш-памяти назначается для хранения другого блока, то удаляемый блок можно не возвращать в основную память, т.к. его копия там уже есть. Недостаток метода – отсутствие эффекта от использования кэш-памяти (сокращение времени доступа) в отношении к операциям записи.

Согласно методу обратной записи, слово заносится только в кэш-память. Если соответствующей строки в кэш-памяти нет, то нужный блок сначала пересылается из ОП, после чего запись все равно выполняется исключительно в кэш-память. При замещении строки ее необходимо предварительно переслать в соответствующее место основной памяти. Для метода обратной записи, в отличие от алгоритма сквозной записи, характерны две пересылки между основной и кэш-памятью при каждом чтении из ОП. В среднем, обратная запись на 10% эффективнее сквозной записи, но для ее реализации требуются и повышенные аппаратные затраты.

Существует ситуация, когда в основную память из устройства ввода, минуя процессор заносится информация, и неверной становится копия, хранимая в кэш-памяти. Предотвратить эту несогласованность позволяют два приема. В первом случае система строится так, чтобы ввод любой информации в ОП автоматически сопровождался соответствующими изменениями в кэш-памяти. Для второго подхода «прямой» доступ к основной памяти допускается только через кэш-память.

2.6.2. Задание к лабораторной работе

В качестве задания предлагается некоторая короткая «программа» (табл. 2.13), которую необходимо выполнить с подключенной кэш-памятью (размером 4 и 8 ячеек) в шаговом режиме для следующих двух вариантов алгоритмов замещения (табл. 2.14).

Таблица 2.13

Варианты задания

№ варианта	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	WR R2	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	WR 9	ADD #3	WR @9	CALL 006	POP R4
7	RD #6	CALL 006	WR 11	WR R2	PUSH R2	RET	JMP 002
8	RD #8	WR R2	WR @R2+	PUSH R2	POP R3	WR -@R3	CALL 003
9	RD #13	WR 14	WR @14	WR @13	ADD 13	CALL 006	RET
10	RD #42	SUB #54	WR 16	WR @16	WR R1	ADD @R1+	PUSH R1
11	RD #10	WR R5	ADD R5	WR R6	CALL 005	PUSH R6	RET
12	JMP 006	RD #76	WR 14	WR R2	PUSH R2	RET	CALL 001

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

Таблица 2.14

Пояснения к вариантам задания

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

2.6.3. Порядок выполнения работы

1. Ввести в модель учебной ЭВМ текст своего варианта «программы» (табл. 2.13), ассемблировать его и сохранить на диске в виде txt-файла.

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

3. В пункте меню **Вид** выбрать команду **Кэш-память**, открыв тем самым модуль **Кэш-память**. Открыть диалоговое окно **Параметры кэш-памяти**, и установить размер – 4 ячейки, далее выбрать режим записи и алгоритм замещения в соответствии с первой строкой своего варианта из табл. 2.14.

4. В шаговом режиме выполнить программу, фиксируя после каждого шага состояние кэш-памяти.

5. Для одной из команд записи (WR) перейти в режим **Такт** и отметить, в каких микрокомандах происходит изменение кэш-памяти.

6. Для кэш-памяти размером 8 ячеек установить параметры в соответствии со второй строкой своего варианта из табл. 2.14 и выполнить программу в шаговом режиме еще раз, фиксируя последовательность номеров замещаемых ячеек кэш-памяти.

2.6.4. Содержание отчета

1. Формулировка цели работы и варианта 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).

5. Выводы по проделанной работе.

2.6.5. Контрольные вопросы

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

2. Какому требованию должен отвечать «идеальный» алгоритм замещения содержимого кэш-памяти?

3. В какую ячейку кэш-памяти будет помещаться очередное слово, если свободные ячейки отсутствуют?

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

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

6. Какие факторы влияют на выбор емкости кэш-памяти и размера блока?

7. В чем состоит принцип временной локальности?

8. В чем состоит принцип пространственной локальности?

9. Сравните два вида кэш-памяти: с прямым отображением и полностью ассоциативную.

2.7. Лабораторная работа №7

Эффективность работы системы с кэш-памятью

Цель работы – изучение влияния параметров кэш-памяти и выбранного алгоритма замещения на эффективность работы системы.

2.7.1. Общие положения

В данной лабораторной работе эффективность работы системы 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 по выражениям (1.2) и (1.3) 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. Также напрямую на эффективность работы кэш-памяти влияет ее размер.

2.7.2. Порядок выполнения работы

В ходе выполнения лабораторной работы необходимо исследовать эффективность работы кэш-памяти при выполнении двух разнотипных программ, написанных и отлаженных при выполнении работ №2 и 4. Для этого следует выполнить пункты 1 – 7.

1. Загрузить в модель учебной ЭВМ отлаженную программу из лабораторной работы №2.

2. Открыв диалоговое окно **Параметры кэш-памяти**, установить следующие параметры: размер – 4, режим записи – сквозная, алгоритм замещения – случайное, без учета бита записи (W).

3. 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. 2.15, помеченную звездочкой.

4. Выключить кэш-память модели (**Работа** → **Кэш-память**) и изменить один из ее параметров – установить флаг с учетом бита записи (в окне **Параметры кэш-памяти**).

5. Повторить п.3, поместив значение полученного коэффициента эффективности в следующую справа ячейку табл. 2.15.

6. Последовательно меняя параметры кэш-памяти, повторить пп.2 – 4, заполняя все ячейки табл. 2.15.

7. Повторить все действия, описанные в пп. 1 – 6 для программы из лабораторной работы № 4, заполняя вторую таблицу по форме табл. 2.15.

2.8.3. Содержание отчета

1. Формулировка цели работы, тексты программ своего варианта из лабораторных работ №2 и 4.
2. Две таблицы по форме табл. 2.15 с результатами моделирования программ из лабораторных работ №2 и 4 при различных режимах работы кэш-памяти.
3. Выводы, объясняющие полученные результаты.

Таблица 2.15

Результаты эксперимента

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

2.8.4. 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 полностью помещается программа)?
3. Как скажется на эффективности алгоритмов замещения учет значения бита записи W 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?
5. Какие проблемы порождает включение в иерархию запоминающих устройств кэш-памяти?
6. Как зависит эффективность работы ЭВМ от размера кэш-памяти?
7. Сравните полностью ассоциативную кэш-память с множественно-ассоциативной кэш-памятью.
8. Чем обусловлено разнообразие методов отображения основной памяти на кэш-память?

ПРИЛОЖЕНИЕ

В приложении представлены вспомогательные таблицы (табл.1 – 3) для работы с моделью учебной ЭВМ.

Таблица 1

Таблица команд учебной ЭВМ

Младший разряд	Старший разряд				
	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 2

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
[]	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. 2 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 3

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]
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(R); M(SP) \leftarrow FLAGS.PC; PC \leftarrow M(V)
19	CALL	Вызов подпрограммы	DEC(SP); M(SP) \leftarrow PC; PC \leftarrow CR(ADR)
20	Нет		
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	Нет		
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 [*]])

Окончание табл. 3

КОП	Мнемокод	Название	Действие
38	OUT	Вывод	$BY(CR[ADR']) \leftarrow Acc$
39	Нет		
40	Нет		
41	RDI	Чтение	$Acc \leftarrow 1$
42	Нет		
43	ADI	Сложение	$Acc \leftarrow Acc + 1$
44	SUBI	Вычитание	$Acc \leftarrow Acc - 1$
45	MULI	Умножение	$Acc \leftarrow Acc \times 1$
46	DIVI	Деление	$Acc \leftarrow Acc / 1$

ЗАКЛЮЧЕНИЕ

Дисциплина «Организация ЭВМ и систем» играет большую роль в подготовке специалистов и бакалавров по направлению «Информатика и вычислительная техника». Находясь в цикле общепрофессиональных дисциплин для всех специальностей данного направления, она закладывает фундамент для дальнейшего изучения современных и перспективных ЭВМ и их элементной базы, микропроцессоров и микроконтроллеров, а также периферийных устройств, их интерфейсов и каналов связи с центральными устройствами.

Лабораторный практикум является неотъемлемой частью учебного процесса, т. к. предоставляет студенту возможность на реальном примере ознакомиться с основными принципами построения и функционирования вычислительных машин, а также процессов, протекающих в отдельных ее элементах.

В настоящем методическом пособии рассмотрены ключевые вопросы организации ЭВМ. В подборе материала, включенного в первую часть пособия, была учтена необходимость в более полном объеме представить архитектуру и структуру вычислительной машины.

Каждая представленная лабораторная работа содержит достаточно подробное теоретическое описание исследуемых процессов, что в полной мере позволит студентам использовать его при подготовке к лабораторным работам.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Жмакин, А.П. Архитектура ЭВМ [Текст] / А. П. Жмакин; – СПб.: БХВ - Петербург, 2006. – 320 с.
2. Цилькер, Б. Я. Организация ЭВМ и систем [Текст]: учеб. для вузов / Б.Я. Цилькер, С.А. Орлов. – СПб.: Питер, 2006. – 667 с.
3. Таненбаум, Э. С. Архитектура компьютера [Текст]: / Э. С. Таненбаум. – 2-е изд. – СПб.: Питер, 2003. – 704 с.
4. Хамахер, К. Организация ЭВМ [Текст]: учеб. / К. Хамахер, З. Вранешич, С. Заки. – 5-е изд. – СПб.: Питер, 2003. – 848 с.
5. Схемотехника электронных систем. Микропроцессоры и микроконтроллеры [Текст] / Бойко В.И. [и др.]. – СПб.: БХВ - Петербург, 2004 – 464 с.
6. Кузин, А.В. Архитектура ЭВМ и вычислительных систем [Текст]: учебник / А. В. Кузин, С.А. Пескова. – М.: ФОРУМ: ИНФРА - М, 2006. – 352 с.
7. Крейгон, Х. Архитектура компьютеров и ее реализация [Текст]: пер. с англ. / Х. Крейгон. – М.: Мир, 2004. – 416 с.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	1
1. АРХИТЕКТУРА ПРОГРАММНОЙ УЧЕБНОЙ МОДЕЛИ ЭВМ.....	4
1.1. Структура ЭВМ.....	4
1.2. Представление данных в модели	6
1.3. Система команд.....	7
1.4. Состояния и режимы работы учебной ЭВМ	11
1.5. Интерфейс пользователя и основные компоненты учебной модели ЭВМ...	12
1.7. Программная модель кэш-памяти.....	19
2. ЛАБОРАТОРНЫЙ ПРАКТИКУМ	21
2.1. Лабораторная работа №1	21
Архитектура ЭВМ и система команд	21
2.2. Лабораторная работа №2	24
Исследование команд передачи управления.....	24
2.3. Лабораторная работа №3	30
Механизм косвенной адресации	30
2.4. Лабораторная работа №4	34
Подпрограммы и стек	34
2.5. Лабораторная работа №5	37
Цикл команды	37
2.6. Лабораторная работа №6	40
Алгоритмы замещения строк кэш-памяти	40
2.7. Лабораторная работа №7	43
Эффективность работы системы с кэш-памятью.....	43
ПРИЛОЖЕНИЕ	46
ЗАКЛЮЧЕНИЕ	49
БИБЛИОГРАФИЧЕСКИЙ СПИСОК	50