Министерство образования Республики Беларусь

Международный государственный экологический институт им. А.Д. Сахарова БГУ

Факультет мониторинга окружающей среды

Кафедра экологических информационных систем

Учебно-методическое пособие по курсу

«Аппаратные средства информационных технологий»

Минск

Содержание

Практическая работа № 1. Знакомство с моделью учебной ЭВМ

Цель работы:

1. Знакомство с архитектурой модели учебной ЭВМ.
2. Изучение системы команд модельной ЭВМ, действиями основных классов команд и способов адресации.
3. Освоение программы реализующей модель учебной ЭВМ. Знакомство с интерфейсом пользователя модели ЭВМ, методами ввода и отладки программы.
4. Изучение процесса программирования на модели учебной ЭВМ.

Теоретические сведения:

1. 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.

Структура ЭВМ

Моделируемая ЭВМ включает:
– процессор;
– блок регистров общего назначения;
– оперативную память (ОЗУ);
– сверхоперативную память (СОЗУ) (кэш-память);
– устройства ввода (УВв);
– устройства вывода (УВыв).
Процессор, в свою очередь, состоит из:
– центрального устройства управления (УУ);
– арифметического устройства (АУ);
– блока системных регистров (CR, PC, SP и др.).
Структурная схема ЭВМ показана на рисунке 1.

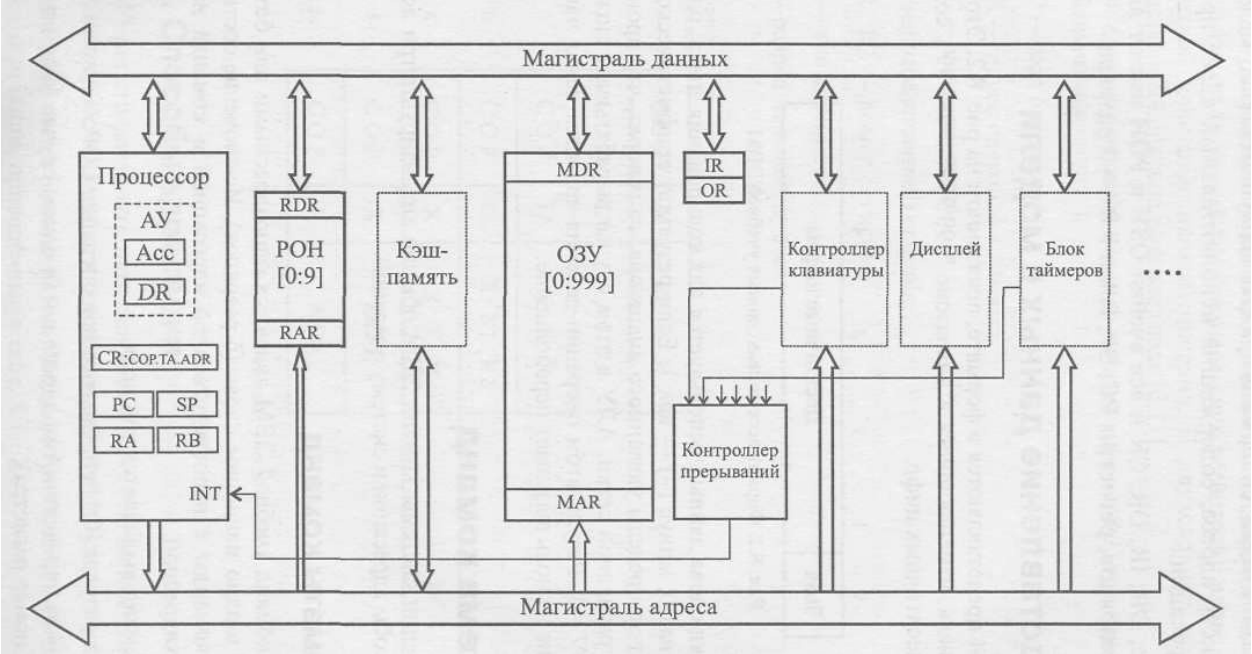

Рис. 1. Общая структура учебной ЭВМ

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

Блок регистров общего назначения (РОН) 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 видимых элементов). Подробнее эти внешние устройства описаны в разделе описания программы моделирующей работу ЭВМ.

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 может подключаться модель кэш-памяти.

В состав УУ ЭВМ входят:

- PC – счетчик адреса команды, содержащий адрес текущей команды;
- CR – регистр команды, содержащий код команды;
- RB – регистр базового адреса, содержащий базовый адрес;
- SP – указатель стека, содержащий адрес верхушки стека;
- RA – регистр адреса, содержащий исполнительный адрес при косвенной адресации.

Регистры данных, такие как : – Acc, DR, IR, OR, CR и все ячейки ОЗУ и РОН имеют длину 6 десятичных разрядов, а регистры команд: – PC, SP, RA и RB – 3 разряда.

Представление данных в модели

Данные в ЭВМ представляются в формате, показанном на рис. 2. Это целые десятичные числа, изменяющиеся в диапазоне "-99 999... +99 999", содержащие знак и 5 десятичных цифр.

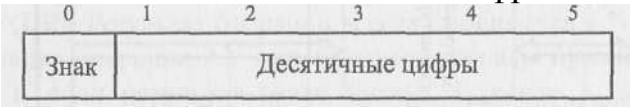

Рис. 2. Формат десятичных данных учебной ЭВМ

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.

Система команд

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

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

Полный список форматов команд показан на рис. 3, где приняты следующие обозначения:

- COP – код операции;
- ADR – адрес операнда в памяти;
- ADC – адрес перехода;
- I – непосредственный операнд;
- R, Rl, R2 – номер регистра; П ТА – тип адресации;
- X – разряд не используется.

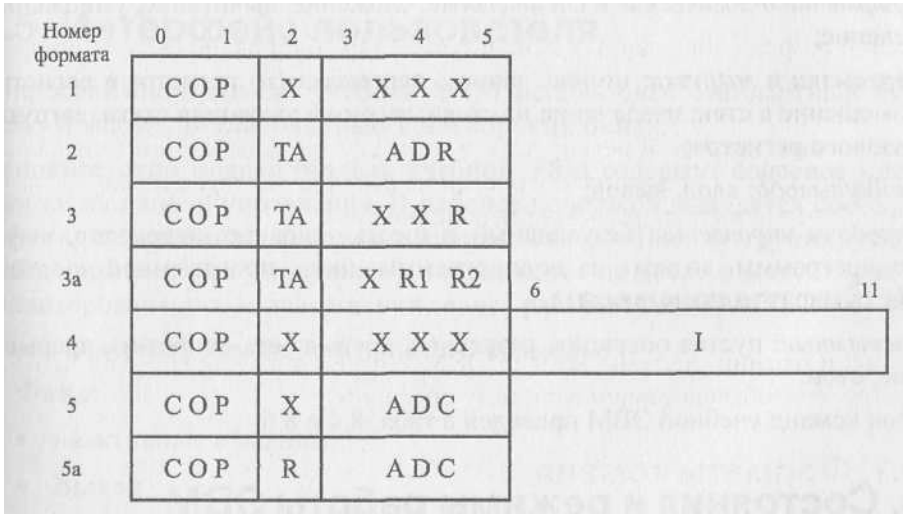

Рис. 3. Форматы команд учебной ЭВМ

Способы адресации

В ЭВМ принято различать пять основных способов адресации:

– прямая,

– косвенная,

– непосредственная,

– относительная,

– безадресная.

Каждый способ адресации имеет разновидности. В модели учебной ЭВМ реализованы семь способов адресации, приведенные в табл. 1.

Таблица 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	РОН(R)[3:5]
5	Индексная с постинкрементом	РОН(R)[3:5], R:=R + 1
6	Индексная с преддекрементом	R:=R-1,РОН(R)[3:5]

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 переход
  - шесть команд условного перехода,
  - вызов подпрограммы,
  - возврат из подпрограммы,
  - цикл,
  - программное прерывание,
  - возврат из прерывания;
- *системные:*
  - пустая операция,
  - разрешить прерывание,

- запретить прерывание,
- стоп.

Список команд учебной ЭВМ приведен в табл. 2 и 3.

Таблица 2. 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		

В таблице 3 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 ← IR
02	OUT	Вывод	OR ← Acc
03	IRET	Возврат из прерывания	FLAGS.PC ← M(SP); INC(SP)
04	WRRB	Загрузка RB	RB ← CR[ADR]
05	WRSP	Загрузка SP	SP ← CR[ADR]
06	PUSH	Поместить в стек	DEC(SP); M(SP) ← R
07	POP	Извлечь из стека	R → M(SP); INC(SP)
08	RET	Возврат	PC → M(SP); INC(SP)
09	HLT	Стоп	Конец командных циклов
10	JMP	Безусловный переход	PC ← CR[ADR]
11	JZ	Переход, если 0	if Acc = 0 then PC ← CR[ADR]

Таблица 3 *(продолжение)*

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

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

КОП	Мнемокод	Название	Действие
38	OUT	Вывод	ВУ(CR[ADR*]) ← Acc
39	Нет		
40	Нет		
41	RDI	Чтение	Acc ← I
42	Нет		
43	ADI	Сложение	Acc ← Acc + I
44	SBI	Вычитание	Acc ← Acc – I
45	MULI	Умножение	Acc ← Acc × I
46	DIVI	Деление	Acc ← Acc/I

собой последовательность команд, записанных в ОЗУ, в автоматическом режиме до команды **HLT** или точки останова. Программа выполняется по командам, начиная с ячейки ОЗУ, на которую указывает PC, причем изменение состояний объектов модели отображается в окнах обозревателей.

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

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

В состоянии **Останов** допускается просмотр и модификация объектов модели: регистров процессора и РОН, ячеек ОЗУ, устройств ввода/вывода. В процессе модификации ячеек ОЗУ и РОН можно вводить данные для программы, в ячейки ОЗУ – программу в кодах. Кроме того, в режиме **Останов** можно менять параметры модели и режимы ее работы, вводить и/или редактировать программу в мнемокодах, ассемблировать мнемокоды, выполнять стандартные операции с файлами.

2. Описание программы CompModel

Модель учебной ЭВМ реализована в виде программы CompModel.exe, которая находиться в подкаталоге Программы, расположенном в том же каталоге где и текст данной работы. 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.

Строка 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 ниже.

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.

2.1. Окна основных обозревателей системы

2.1.1 Окно **Процессор**

Окно **Процессор** (рис. 4) обеспечивает доступ ко всем регистрам и флагам процессора.

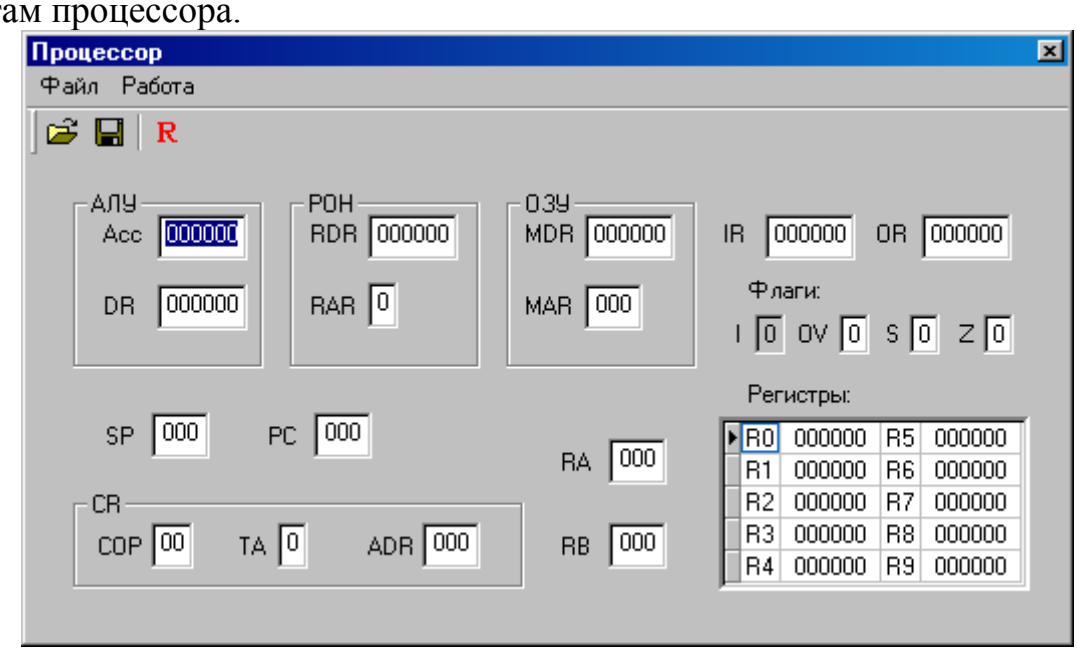

Рис.4. Окно Процессор

- Программно-доступные регистры и флаги:
  - Acc – аккумулятор АЛУ;
  - 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: D COP – код операции;
  - TA – тип адресации;
  - ADR – адрес или непосредственный операнд;
  - Z – флаг нулевого значения Асе;
  - S – флаг отрицательного значения Acc;
  - OV – флаг переполнения.

Регистры Acc, 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.

2.1.2 Окно **Память**

Окно **Память** (рис. 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.

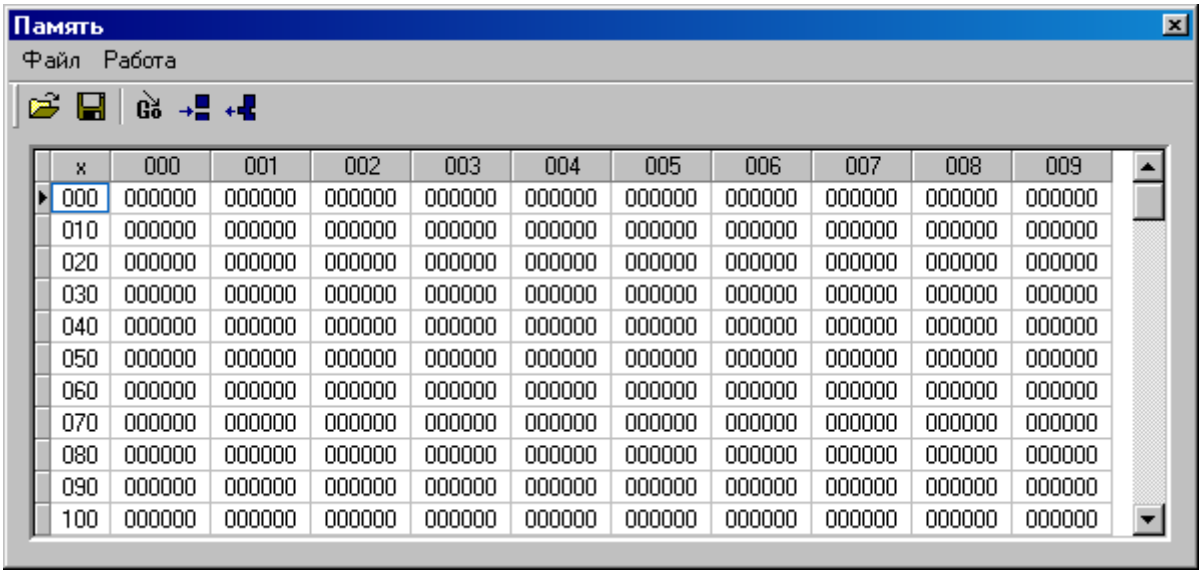

Рис. 5. Окно Память

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.

2.1.3 Окно **Текст программы**

Окно **Текст программы** (рис. 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.

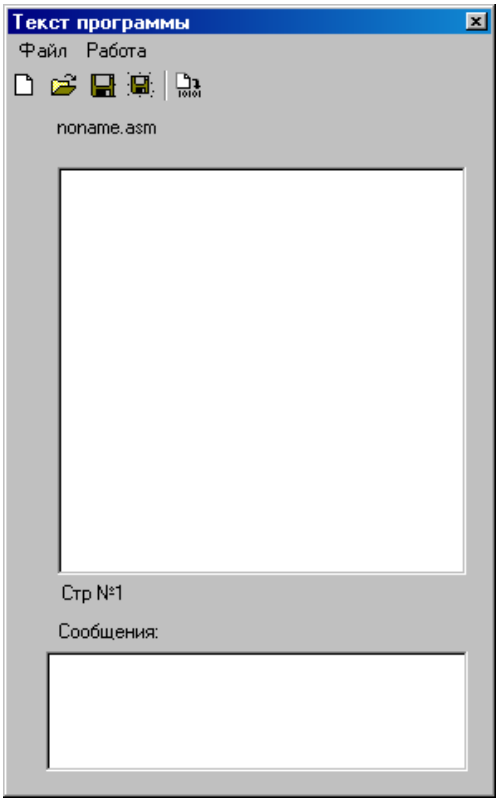

Рис. 6. Окно Текст программы

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.

Правила ввода текстов программ:

1. Набор текста программы производится по стандартным правилам языка ассемблера.
2. Мнемонический код операции отделяется от операндов пробелом.
3. Мнемокод должен быть записан только заглавными буквами.
4. В каждой строке может содержаться метка, одна команда и комментарий.
5. Метка отделяется от команды двоеточием, символы после знака "точка с запятой" до конца строки игнорируются компилятором и могут рассматриваться как комментарии.
6. Строка может начинаться с ; и, следовательно, содержать только комментарии.

2.1.4. Окно **Программа**

Окно **Программа** (рис. 7) отображает таблицу, имеющую 300 строк и 4 столбца. Каждая строка таблицы соответствует дизассемблированной ячейке ОЗУ. Второй столбец содержит адрес ячейки ОЗУ, третий – дизассемблированный мнемокод, четвертый – машинный код команды. В первом столбце может помещаться указатель `- ->` на текущую команду (текущее значение PC) и точка останова – красная заливка ячейки.

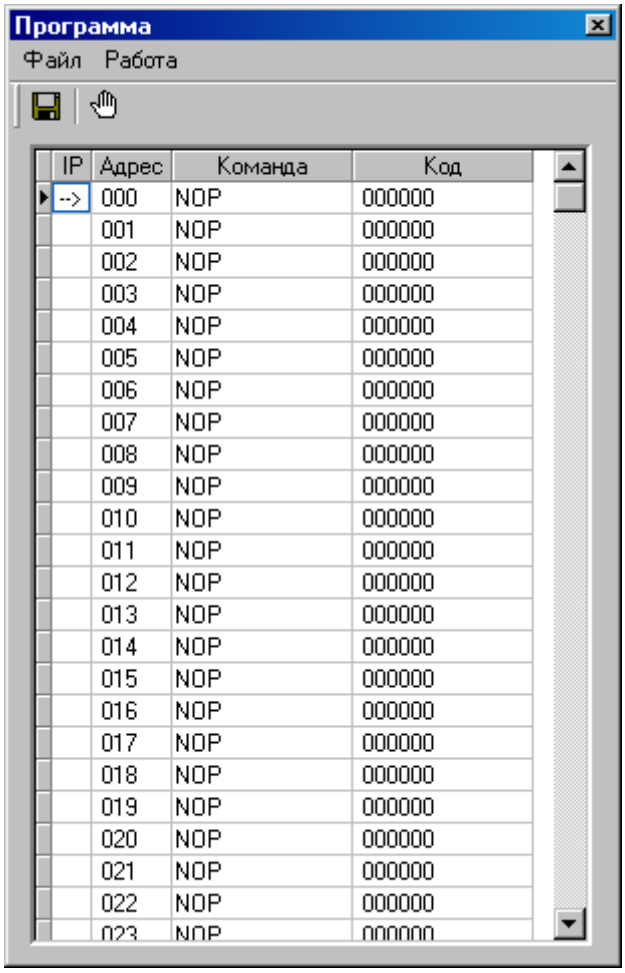

Рис. 7. Окно Программа

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

изначально пишется или редактируется непосредственно в памяти в машинных кодах.

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

2.1.5 Окно **Микрокомандный уровень**

Окно **Микрокомандный уровень** (рис. 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".

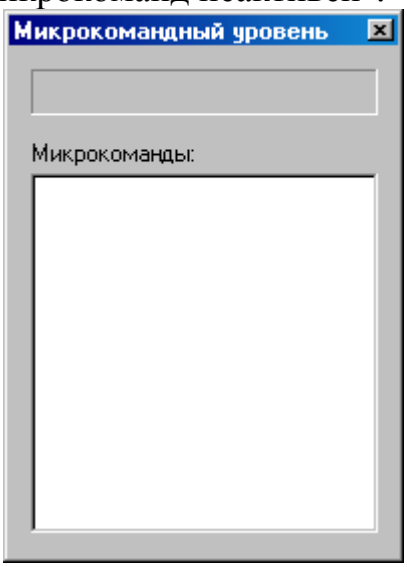

Рис. 8. Окно Микрокомандный уровень

2.2. Вспомогательные таблицы

В данном разделе представлены вспомогательные таблицы (табл. 6–8) для работы с моделью учебной ЭВМ.

Таблица 6. 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

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

Таблица 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					А	Р	а	р
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	щ	ю
F			/	?	O	_	o						П	Я	п	я

Таблица 8. Перевод шестнадцатиричных кодов в десятичные

	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

3. Порядок работы с моделью учебной ЭВМ

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 **IR** и **BR**;
- установить в **PC** стартовый адрес программы;
- перевести модель в режим **Работа.**

Каждое из этих действий выполняется посредством интерфейса модели, описанного в разделе 2.

Ввод программы.

Ввод программы может осуществляться двумя способами:

1-й способ: – в машинных кодах непосредственно в память модели в окне **Память.**

2-й способ: – в мнемокодах в окне **Текст программы** с последующим ассемблированием.

Ввод значений РОН, регистров процессора.

Ввод значений РОН, а также регистров PC, IR, BR др. выполняется в окне **Процессор**, путем непосредственного редактирования соответствующих полей.

Ввод команд и данных в память модели.

Ввод команд и данных в память модели выполняется в окне **Память**.

Практическая работа № 2. Архитектура ЭВМ и система команд

Цель работы:

1. Знакомство с архитектурой модели учебной ЭВМ.
2. Освоение программы реализующей модель учебной ЭВМ.
3. Изучение системы команд модельной ЭВМ.
4. Изучение процесса программирования на модели учебной ЭВМ.

Теоретические сведения:

Классическая архитектура ЭВМ

Считается, что основные идеи построения современных ЭВМ в 1945 г. сформулировал американский математик Дж. фон Нейман, определив их как *принципы программного управления:*

1. Информация кодируется в двоичной форме и разделяется на единицы слова.
2. Разнотипные по смыслу слова различаются по способу использования, но не по способу кодирования.
3. Слова информации размещаются в ячейках памяти и идентифицируются номерами ячеек – адресами слов.
4. Алгоритм представляется в форме последовательности управляющих слов, называемых *командами.* Команда определяет наименование операции и слова информации, участвующие в ней. Алгоритм, записанный в виде последовательности команд, называется *программой.*
5. Выполнение вычислений, предписанных алгоритмом, сводится к последовательному выполнению команд в порядке, однозначно определенном программой.

Поэтому классическую архитектуру современных ЭВМ, представленную на рис. 1, часто называют "архитектурой фон Неймана".

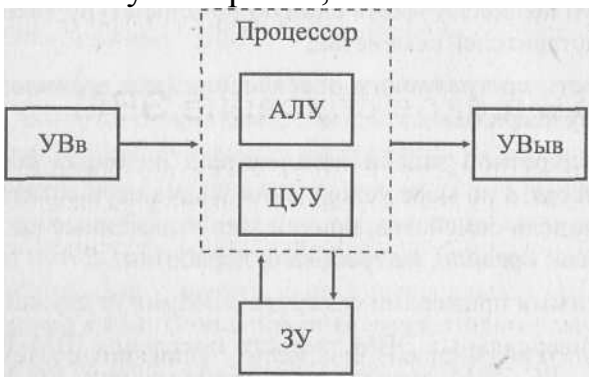

Рис. 1. Классическая архитектура ЭВМ

Программа вычислений (обработки информации) составляется в виде последовательности команд и загружается в память машины – *запоминающее*

устройство (ЗУ). Там же хранятся исходные данные и промежуточные результаты обработки. *Центральное устройство управления* (ЦУУ) последовательно извлекает из памяти команды программы и организует их выполнение. *Арифметико-логическое устройство* (АЛУ) предназначено для реализации операций преобразования информации. Программа и исходные данные вводятся в память машины через *устройства ввода* (УВв), а результаты обработки предъявляются на *устройства вывода* (УВыв).

Характерной особенностью архитектуры фон Неймана является то, что память представляет собой единое адресное пространство, предназначенное для хранения как программ, так и данных.

Такой подход, с одной стороны, обеспечивает большую гибкость организации вычислений – возможность перераспределения памяти между программой и данными, возможность самомодификации программы в процессе ее выполнения. С другой стороны, без принятия специальных мер защиты снижается надежность выполнения программы, что особенно недопустимо в управляющих системах.

Действительно, поскольку и команды программы, и данные кодируются в ЭВМ двоичными числами, теоретически возможно как разрушение программы (при обращении в область программы как к данным), так и попытка "выполнения" области данных как программы (при ошибочных переходах программы в область данных).

Альтернативной фон-неймановской является т. н. *гарвардская архитектура.* ЭВМ, реализованные по этому принципу, имеют два непересекающихся адресных пространства– для программы и для данных, причем программу нельзя разместить в свободной области памяти данных и наоборот. Гарвардская архитектура применяется главным образом в управляющих ЭВМ.

Архитектура ЭВМ– это абстрактное представление ЭВМ, которое отражает ее структурную, схемотехническую и логическую организацию. Понятие архитектуры ЭВМ является комплексным и включает в себя:

- структурную схему ЭВМ;
- средства и способы доступа к элементам структурной схемы;
- организацию и разрядность интерфейсов ЭВМ;
- набор и доступность регистров;
- организацию и способы адресации памяти;
- способы представления и форматы данных ЭВМ;
- набор машинных команд ЭВМ;
- форматы машинных команд;
- обработку нештатных ситуаций (прерываний).

Командный цикл процессора

Командой называется элементарное действие, которое может выполнить процессор без дальнейшей детализации. Последовательность команд, выполнение которых приводит к достижению определенной цели, называется *программой.* Команды программы кодируются двоичными словами и размещаются в памяти ЭВМ. Вся работа ЭВМ состоит в последовательном выполнении команд программы. Действия по выбору из памяти и выполнению одной команды называются *командным циклом.*

В составе любого процессора имеется специальная ячейка, которая хранит адрес выполняемой команды – *счетчик команд* или *программный счетчик.* После выполнения очередной команды его значение увеличивается на единицу (если код одной команды занимает несколько ячеек памяти, то содержимое счетчика команд увеличивается на длину команды). Таким образом осуществляется выполнение последовательности команд. Существуют специальные команды (передачи управления), которые в процессе своего выполнения модифицируют содержимое программного счетчика, обеспечивая переходы по программе. Сама выполняемая команда помещается в *регистр команд* – специальную ячейку процессора.

Во время выполнения командного цикла процессор реализует следующую последовательность действий:

1. Извлечение из памяти содержимого ячейки, адрес которой хранится в программном счетчике, и размещение этого кода в регистре команд (чтение команды).
2. Увеличение содержимого программного счетчика на единицу.
3. Формирование адреса операндов.
4. Извлечение операндов из памяти.
5. Выполнение заданной в команде операции.
6. Размещение результата операции в памяти.
7. Переход к п. 1.

Пункты 1, 2 и 7 обязательно выполняются в каждом командном цикле, остальные могут не выполняться в некоторых командах. Если длина кода команды составляет несколько машинных слов, то пп. 1 и 2 повторяются.

Фактически вся работа процессора заключается в циклическом выполнении пунктов 1-–7 командного цикла. При запуске машины в счетчик команд аппаратно помещается фиксированное значение – начальный адрес программы (часто 0 или последний адрес памяти; встречаются и более экзотические способы загрузки начального адреса). В дальнейшем содержимое программного счетчика модифицируется в командном цикле. Прекращение выполнения командных циклов может произойти только при выполнении специальной команды "СТОП".

Система команд процессора

Разнообразие типов данных, форм их представления и действий, которые необходимы для обработки информации и управления ходом

вычислений, порождает необходимость использования различных команд – набора команд. Каждый процессор имеет собственный вполне определенный набор команд, называемый *системой команд процессора.* Система команд должна обладать двумя свойствами – *функциональной полнотой* и *эффективностью.*

Функциональная полнота – это достаточность системы команд для описания любого алгоритма. Требование функциональной полноты не является слишком жестким. Доказано, что свойством функциональной полноты обладает система, включающая всего *три* команды (система Поста):

- *присвоение 0,*
- *присвоение 1,*
- *проверка на 0.*

Однако составление программ в такой системе команд крайне неэффективно.

Эффективность системы команд – степень соответствия системы команд назначению ЭВМ, т. е. классу алгоритмов, для выполнения которых предназначается ЭВМ, а также требованиям к производительности ЭВМ. Очевидно, что реализация развитой системы команд связана с большими затратами оборудования и, следовательно, с высокой стоимостью процессора. В то же время ограниченный набор команд приводит к снижению производительности и повышенным требованиям к памяти для размещения программы. Даже простые и дешевые современные микропроцессоры поддерживают систему команд, содержащую несколько десятков (а с модификациями– сотен) команд.

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

Форматы команд

Под *форматом команды* следует понимать длину команды, количество, размер, положение, назначение и способ кодировки ее полей.

Команды, как и любая информация в ЭВМ, кодируются двоичными словами, которые должны содержать в себе следующие виды информации:

П тип операции, которую следует реализовать в данной команде (КОП); П место в памяти, откуда следует взять первый операнд (А1); П место в памяти, откуда следует взять второй операнд (А2); О место в памяти, куда следует поместить результат (А3).

Каждому из этих видов информации соответствует своя часть двоичного слова – поле, а совокупность полей (их длины, расположение в командном слове, способ кодирования информации) называется форматом команды. В свою очередь, некоторые поля команды могут делиться на подполя. Формат команды, поля которого перечислены выше, называется *трехадресным* (рис. 2, а).

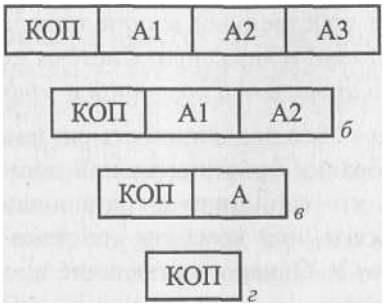

Рис. 2. Форматы команд: *а* – трехадресный; *б* – двухадресный; *в* – одноадресный; *г* – безадресный

Команды трехадресного формата занимают много места в памяти, в то же время далеко не всегда поля адресов используются в командах эффективно. Действительно, наряду с двухместными операциями (сложение, деление, конъюнкция и др.) встречаются и одноместные (инверсия, сдвиг, инкремент и др.), для которых третий адрес не нужен. При выполнении цепочки вычислений часто результат предыдущей операции используется в качестве операнда для следующей. Более того, нередко встречаются команды, для которых операнды не определены (СТОП) или подразумеваются самим кодом операций (DAA, десятичная коррекция аккумулятора).

Поэтому в системах команд реальных ЭВМ трехадресные команды встречаются редко. Чаще используются *двухадресные команды* (рис. 2.1,6), в этом случае в бинарных операциях результат помещается на место одного из операндов.

Для реализации одноадресных форматов (рис. 2.1, б) в процессоре предусматривают специальную ячейку – *аккумулятор*. Первый операнд и результат всегда размещаются в аккумуляторе, а второй операнд адресуется полем А.

Реальная система команд обычно имеет команды нескольких форматов, причем тип формата определяется в поле КОП.

Способы адресации

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

Не следует думать, что во всех случаях в поле адреса команды помещается адрес операнда. Существует пять основных способов адресации операндов в командах.

- *Прямая* – в этом случае в адресном поле располагается адрес операнда. Разновидность – *прямая регистровая* адресация, адресующая не ячейку памяти, а РОН. Поле адреса регистра имеет в команде значительно меньшую длину, чем поле адреса памяти.
- *Непосредственная* – в поле адреса команды располагается не адрес операнда, а сам операнд. Такой способ удобно использовать в командах с константами.

- *Косвенная* – в поле адреса команды располагается адрес ячейки памяти, в которой хранится адрес операнда ("адрес адреса"). Такой способ позволяет оперировать адресами как данными, что облегчает организацию циклов, обработку массивов данных и др. Его основной недостаток– потеря времени на двойное обращение к памяти – сначала за адресом, потом – за операндом. Разновидность – *косвенно-регистровая* адресация, при которой в поле команды размещается адрес РОН, хранящего адрес операнда. Этот способ, помимо преимущества обычной косвенной адресации, позволяет обращаться к большой памяти с помощью коротких команд и не требует двойного обращения к памяти (обращение к регистру занимает гораздо меньше времени, чем к памяти).
- *Относительная* – адрес формируется как сумма двух слагаемых: базы, хранящейся в специальном регистре или в одном из РОН, и смещения, извлекаемого из поля адреса команды. Этот способ позволяет сократить длину команды (смещение может быть укороченным, правда в этом случае не вся память доступна в команде) и/или перемещать адресуемые массивы информации по памяти (изменяя базу). Разновидности – *индексная* и *базово-индексная* адресации. Индексная адресация предполагает наличие индексного регистра вместо базового. При каждом обращении содержимое индексного регистра автоматически модифицируется (обычно увеличивается или уменьшается на 1). Базово-индексная адресация формирует адрес операнда как сумму трех слагаемых: базы, индекса и смещения.
- *Безадресная* – поле адреса в команде отсутствует, а адрес операнда или не имеет смысла для данной команды, или подразумевается по умолчанию. Часто безадресные команды подразумевают действия над содержимым аккумулятора. Характерно, что безадресные команды нельзя применить к другим регистрам или ячейкам памяти.

Одной из разновидностей безадресного обращения является использование т. н. магазинной памяти или *стека.* Обращение к такой памяти напоминает обращение с магазином стрелкового оружия. Имеется фиксированная ячейка, называемая *верхушкой стека.* При чтении слово извлекается из верхушки, а все остальное содержимое "поднимается вверх" подобно патронам в магазине, так что в верхушке оказывается следующее по порядку слово. Одно слово нельзя прочитать из стека дважды. При записи новое слово помещается в верхушку стека, а все остальное содержимое "опускается вниз" на одну позицию. Таким образом, слово, помещенное в стек первым, будет прочитано последним. Говорят, что стек поддерживает дисциплину LIFO – Last In First Out (последний пришел – первый ушел). Реже используется безадресная память типа *очередь* с дисциплиной FIFO – First In First Out (первый пришел – первый ушел).

Система операций

Все операции, выполняемые в командах ЭВМ, принято делить на пять классов.

- *Арифметико-логические* и *специальные* – команды, в которых выполняется собственно преобразование информации. К ним относятся арифметические операции сложение, вычитание, умножение и деление (с фиксированной и плавающей занятой), команды десятичной арифметики, логические операции конъюнкции, дизъюнкции, инверсии и др., сдвиги, преобразование чисел из одной системы счисления в другую и такие экзотические, как извлечение корня, решение системы уравнений и др. Конечно, очень редко встречаются ЭВМ, система команд которых включает все эти команды.
- *Пересылки* и *загрузки* – обеспечивают передачу информации между процессором и памятью или между различными уровнями памяти (СОЗУ <–> ОЗУ). Разновидность – *загрузка регистров и ячеек константами.*
- *Ввода/вывода* – обеспечивают передачу информации между процессором и внешними устройствами. По структуре они очень похожи на команды предыдущего класса. В некоторых ЭВМ принципиально отсутствует различие между ячейками памяти и регистрами внешних устройств (единое адресное пространство) и класс команд ввода/вывода не выделяется, все обмены осуществляются в рамках команд пересылки и загрузки.
- *Передачи управления* – команды, которые изменяют естественный порядок выполнения команд программы. Эти команды меняют содержимое программного счетчика, обеспечивая переходы по программе. Существуют команды безусловной и условной передачи управления. В последнем случае передача управления происходит, если выполняется заданное в коде команды условие, иначе выполняется следующая по порядку команда.

В качестве условий обычно используются признаки результата предыдущей операции, которые хранятся в специальном регистре признаков (флажков). Чаще всего формируются и проверяются признаки нулевого результата, отрицательного результата, наличия переноса из старшего разряда, четности числа единиц в результате и др. Различают три разновидности команд передачи управления:

- переходы;
- вызовы подпрограмм;
- возвраты из подпрограмм.

Команды *переходов* помещают в программный счетчик содержимое своего адресного поля – адрес перехода. При этом старое значение программного счетчика теряется. В микро ЭВМ часто для экономии длины адресного поля команд условных переходов адрес перехода формируется как

сумма текущего значения программного счетчика и относительно короткого знакового смещения, размещаемого в команде. В крайнем случае, в командах условных переходов можно и вовсе обойтись без адресной части – при выполнении условия команда "перепрыгивает" через следующую команду, которой обычно является безусловный переход.

Команда *вызова* подпрограммы работает подобно команде безусловного перехода, но старое значение программного счетчика предварительно сохраняется в специальном регистре или в стеке. Команда возврата передает содержимое верхушки стека или специального регистра в программный счетчик. Команды *вызова* и *возврата* работают "в паре". Подпрограмма, вызываемая командой вызова, должна заканчиваться командой возврата, что обеспечивает по окончании работы подпрограммы передачу управления в точку вызова. Хранение адресов возврата в стеке обеспечивает возможность реализации вложенных подпрограмм.

- *Системные*– команды, выполняющие управление процессом обработки информации и внутренними ресурсами процессора. К таким командам относятся команды управления подсистемой прерывания, команды установки и изменения параметров защиты памяти, команда останова программы и некоторые другие. В простых процессорах класс системных команд немногочисленный, а в сложных мультипрограммных системах предусматривается большое число системных команд.

Задания для выполнения:

1. Ознакомиться с теоретическими сведениями по архитектуре ЭВМ
2. Запустите программу `CompModel.exe`.
3. Записать в ОЗУ "программу", состоящую из пяти команд– номер задания выбрать из табл. 3, в соответствии с номером Вашего варианта. Команды разместить в последовательных ячейках памяти.

Таблица 3. Варианты заданий

№	**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
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	WR 9	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

При необходимости установить начальное значение в устройство ввода IR.

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

4. Выполнить в режиме **Шаг** введенную последовательность команд, фиксируя изменения значений объектов, определенных в п. 4, в таблице следующего формата.

Таблица 2. Результаты выполнения программы

PC	**Acc**	**M(xx)**	**M(нн)**	**PC**	**Acc**	**M(мм)**	**M(оо)**
000							
001							
002							
003							

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

Содержание отчета

2.1.

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. 1).

Таблица 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

2.2. Введите коды табл.1 последовательно в ячейки ОЗУ, начиная с адреса 000.

2.3. Выполняя команды в режиме Шаг, фиксируйте в табл. 2 изменения программно-доступных объектов (в данном случае это Acc, PC и ячейки ОЗУ 020 и 030).

Таблица 2. Содержимое регистров ЭВМ в результате выполнения примера программы

PC	Acc	M(30)	M(20)	**PC**	**Acc**	**M(30)**	M(20)
000	000000	000000	000000	004			000025
001	000020			002			
002		000020		003	000030		
003	000025			004			000030

3. Оформление результатов.

Включите в отчет:

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

Оформите результаты выполнения данной работы в виде отчета (в электронном виде), в соответствии с требованиями.

Задания для выполнения:

1. Изучить теоретические сведения, описывающие модель учебной ЭВМ и ее программную реализацию.

2. Выполнить пример занесения программы и ее выполнения на модели ЭВМ.

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 шестиразрядных десятичных чисел (см. форматы команд на рис. 3, коды команд и способов адресации в табл. 2–4).

2.1. Запустите программу `CompModel.exe`.

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. 1).

Таблица 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

2.2. Введите коды табл.1 последовательно в ячейки ОЗУ, начиная с адреса 000.

2.3. Выполняя команды в режиме Шаг, фиксируйте в табл. 2 изменения программно-доступных объектов (в данном случае это Acc, PC и ячейки ОЗУ 020 и 030).

Таблица 2. Содержимое регистров ЭВМ в результате выполнения примера программы

PC	Acc	M(30)	M(20)	**PC**	**Acc**	**M(30)**	M(20)
000	000000	000000	000000	004			000025
001	000020			002			
002		000020		003	000030		
003	000025			004			000030

3. Выполнить задания:

Задание 1. Вычислить значение S:

S = M + D + X

где: M – месяц Вашего рождения;

D – день Вашего рождения;

X – номер Вашего варианта.

Результат сохранить в ячейке по адресу 070.

Задание 2. Вычислить значение A:

A = X x D (X умножить на D);

Результат сохранить в ячейке по адресу 071.

Задание 3. Вычислить значение:

B = X – D;

Задание 4. Вычислить значение:

C = X x 2 + B x 4

Результат сохранить в ячейке по адресу 072.

4. Оформление результатов.

Включите в отчет:

1. Формулировку каждого выполненного примера и задания.
2. Машинные коды команд, соответствующих заданию или примеру.
3. Код программы на языке ассемблер.
3. Результаты выполнения последовательности команд в форме табл. 2.

Оформите результаты выполнения данной работы в виде отчета (в электронном виде), в соответствии с требованиями.

Практическая работа № 3. Команды управления и ветвления

Цель работы:

1. Изучение команд управления ходом вычислений для модельной ЭВМ.
2. Разработка программы реализующей ветвящийся вычислительный процесс на модели учебной ЭВМ.

Теоретические сведения:

Команды управления модельной ЭВМ

К этому классу команд относятся, которые изменяют естественный порядок выполнения команд программы. Эти команды меняют содержимое программного счетчика, обеспечивая переходы по программе. Существуют следующие виды команд управления:

- команды переходов;
- команды вызова подпрограмм (процедур);
- системные команды управления процессом обработки информации и внутренними ресурсами процессора.

1. Команды передачи управления;

Различают два вида переходов в программах:

- безусловный переход;
- условный переход.

В последнем случае передача управления происходит, если выполняется заданное в коде команды условие, иначе выполняется следующая по порядку команда.

В качестве условий обычно используются признаки результата предыдущей операции, которые хранятся в специальном регистре признаков (флажков). Чаще всего формируются и проверяются признаки:

- нулевого результата;
- отрицательного;
- результата;
- наличия переноса из старшего разряда;
- четности числа единиц в результате и др.

Различают три разновидности команд передачи управления:

- переходы;
- вызовы подпрограмм;
- возвраты из подпрограмм.

Команды *переходов* помещают в программный счетчик содержимое своего адресного поля – адрес перехода. При этом старое значение программного счетчика теряется. В микро-ЭВМ часто для экономии длины адресного поля команд условных переходов адрес перехода формируется как

сумма текущего значения программного счетчика и относительно короткого знакового смещения, размещаемого в команде. В крайнем случае, в командах условных переходов можно и вовсе обойтись без адресной части – при выполнении условия команда "перепрыгивает" через следующую команду, которой обычно является безусловный переход.

2. Команда *вызова* подпрограмм работает подобно команде безусловного перехода, но старое значение программного счетчика предварительно сохраняется в специальном регистре или в стеке. Команда возврата передает содержимое верхушки стека или специального регистра в программный счетчик. Команды *вызова* и *возврата* работают "в паре". Подпрограмма, вызываемая командой вызова, должна заканчиваться командой возврата, что обеспечивает по окончании работы подпрограммы передачу управления в точку вызова. Хранение адресов возврата в стеке обеспечивает возможность реализации вложенных подпрограмм.

3. *Системные*– команды, выполняющие управление процессом обработки информации и внутренними ресурсами процессора. К таким командам относятся:

- команды управления подсистемой прерывания;
- команда загрузки PSW – Program Status Word (слово состояния программы);
- команды установки флагов в регистре состояния процессора;
- команды установки и изменения параметров защиты памяти;
- команда останова программы;
- и некоторые другие команды, например команда диагностики работы.

В простых процессорах класс системных команд немногочисленный, а в сложных мультипрограммных системах предусматривается большое число системных команд.

Задания для выполнения:

1. Ознакомиться с теоретическими сведениями о командах управления модельной ЭВМ.

2. Запустите программу `CompModel.exe`.

3. Выполнить пример. В качестве примера рассмотрим программу вычисления функции:

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

Причем *x* вводится с устройства ввода IR, результат *y* выводится на OR. Граф-схема алгоритма решения задачи показана на рис.1.

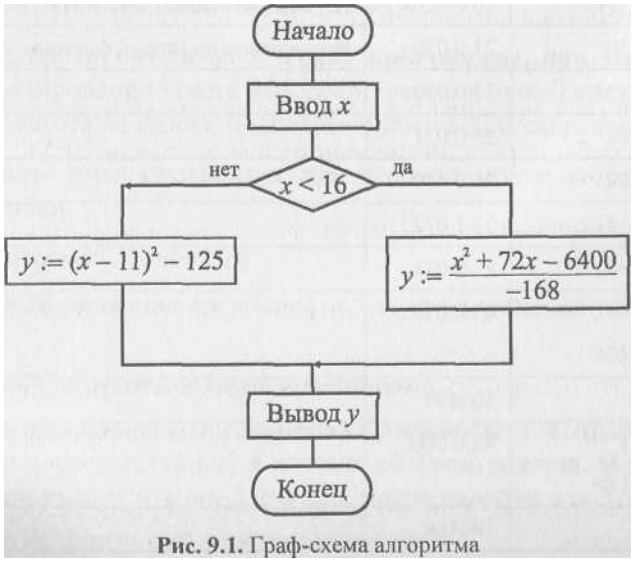

Рис. 9.1. Граф-схема алгоритма

В данном примере 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. 1.

Таблица 1. Пример программы

Адрес	**Команда**		**Примечание**
	Мнемокод	**Код**	
000	IN	010 000	Ввод *x*
001	WR 30	22 0 030	Размещение *x* в ОЗУ(ОЗО)
002	SUB #16	24 1016	Сравнение с границей – *(x* -16)
003	JS 010	130010	Переход по отрицательной разности
004	RD 30	210 030	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	210 030	
014	MUL #72	25 1 072	
015	ADD 31	23 0 031	
016	ADI 106400	43 0 000	

017		106400	
018	DIVI 100168	46 0 000	
019		100168	
020	OUT	02 0 000	Вывод результата
021	HLT	09 0 000	Стоп

4. Разработать программу вычисления и вывода значения функции:

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

4.1 Для вводимого из IR значения аргумента *x.* Функции и допустимые пределы изменения аргумента приведены в табл. 2, варианты заданий – в табл. 3.

4.2. Исходя из допустимых пределов изменения аргумента функций (табл. 2) и значения параметра *a* для своего варианта задания (табл. 3) 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.

4.3. 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.

4.4. Ассемблировать текст программы, при необходимости исправить синтаксические ошибки.

4.5. 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. 4.5г и 4.5*д;* иначе перейти к п. 4.5*е;*

г) записать в PC стартовый адрес программы;

д) наблюдая выполнение программы в режиме Шаг, найти команду, являющуюся причиной ошибки; исправить ее; выполнить пп. 4.5, а – 4.5, в;

е) записать в IR значение аргумента $x < a$ (в области допустимых значений); выполнить пп. 4.5б и 4.5в;

ж) записать в IR недопустимое значение аргумента *x* и выполнить пп. 4.5б и 4.5в.

6. Для выбранного допустимого значения аргумента x наблюдать выполнение отлаженной программы в режиме Шаг и записать в форме табл. 4 содержимое регистров ЭВМ перед выполнением каждой команды.

Таблица 2. Функции

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

Таблица 3. Варианты задания

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

Таблица 4. Результаты выполнения программы

РС	**Асс**	**М(хх)**	**М(нн)**	**РС**	**Асс**	**М(мм)**	**М(оо)**
000							
001							
002							
003							

Содержание отчета

Отчет о практическая 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. 1.
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.

5. Оформление результатов.

Оформите результаты выполнения данной работы в виде отчета (в электронном виде), в соответствии с требованиями.

Практическая работа № 4. Применение косвенной адресации

Цель работы:

1. Изучение косвенной адресации ЭВМ.
2. Разработка циклической программы с переадресацией для модели учебной ЭВМ.

Теоретические сведения:

Косвенная адресация

Косвенная адресация получила широкое применение в современных микропроцессорных системах. Без нее сложно реализовать эффективную работу с массивами памяти, обеспечить циклическое выполнение фрагментов программ, а также обеспечить оперировать адресами как данными (адресная арифметика).

Косвенная адресация – предполагает, что в поле адреса команды располагается адрес ячейки памяти, в которой хранится адрес операнда ("адрес адреса") (рисунок 1 а)). Такой способ позволяет оперировать адресами как данными, что облегчает организацию циклов, обработку массивов данных и др. Его основной недостаток– потеря времени на двойное обращение к памяти – сначала за адресом, потом – за операндом.

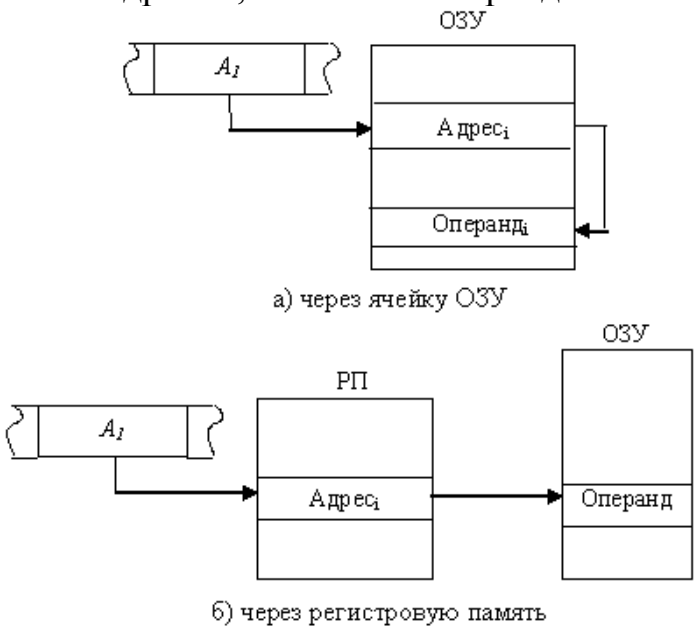

Рис. 1. Косвенная адресация памяти; а) через ячейку ОП, б) через регистр.

Разновидность – косвенно-регистровая адресация, при которой в поле команды размещается адрес РОН, хранящего адрес операнда (рисунок 1 б)). Этот способ, помимо преимущества обычной косвенной адресации, позволяет обращаться к большой памяти с помощью коротких команд и не требует

двойного обращения к памяти (обращение к регистру занимает гораздо меньше времени, чем к памяти).

Реализация косвенной адресации в модельной ЭВМ

Все типы адресации, используемые в учебной ЭВМ приведены в таблице 1.

Таблица 1. **Адресация в командах учебной ЭВМ**

Код ТА	**Тип адресации**	**Исполнительный адрес**
0	Прямая (регистровая)	ADR (R)
1	Непосредственная	–
2	Косвенная	O3Y(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 предцекрементом	R:=R-1,POH(R)[3:5]

Тип адресации, используемый в команде определяется полем команды ТА разряд [2] команды). Косвенной адресации соответствуют коды 2 и 4 (косвенно-регистровая) , содержащиеся в данном поле.

Этот тип адресации может быть использован практически всеми одно- и 2-х адресными командами модельной ЭВМ. При этом разряды [3:5] команды могут определять прямой или косвенный адрес памяти, номер регистра (в команде MOV номера двух регистров), адрес перехода или короткий непосредственный операнд.. При записи программы в мнемокодах для обозначения этого типа адресации используются обозначения приведенные в таблице 2.

Таблица 2. Обозначение способов адресации при использовании мнемокодов.

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

Задания для выполнения:

1. Ознакомиться с теоретическими сведениями о реализации косвенной памяти в модельной ЭВМ.

2. Запустите программу `CompModel.exe`.

3. Выполнить следующий пример.

Изучить алгоритм работы программы вычисления суммы элементов массива чисел C_1, C_2,..., C_n. Исходными данными в этой задаче являются:

- n – количество суммируемых чисел
- $C_1, C_2, .., C_n$ – 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.

Рассмотрим 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, $i \in \{1, 2,..., 10\}$;

– ОЗУ(A_i) – число по адресу A_i,

– S –текущая сумма;

$к$ – счетчик цикла, определяющий число повторений.

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;
- $к$ – по адресу 031;
- S – по адресу 032.

Граф-схема алгоритма (ГСА) программы, показана на рис. 2, текст программы с комментариями приведен в табл. 3.

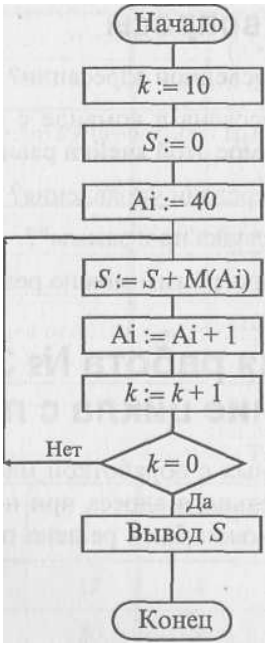

Рис.2. Граф-схема алгоритма

Таблица 3. Текст программы примера

Адрес	**Команда**	**Примечание**
000	RD #40	Загрузка начального адреса массива 040 в ячейку 030
001	WR 30	
002	RD #10	Загрузка параметра цикла *к* = 10 в ячейку 031
003	WR 31	
004	RD #0	Загрузка начального значения суммы *S* = 0 в ячейку 032
005	WR 32	
006	Ml: RD 32	Добавление к текущей сумме очередного элемента массива
007	ADD @30	
008	WR 32	
009	RD 30	Модификация текущего адреса массива (переход к следующему адресу)
010	ADD #1	
011	WR 30	
012	RD 31	Уменьшение счетчика (параметра цикла) на 1
013	SUB #1	
014	WR 31	
015	JNZ Ml	Проверка параметра цикла и переход при *к≠0*
016	RD 32	Вывод результата
017	OUT	
018	HLT	Стоп

4. Разработать программу определения заданной характеристики последовательности чисел C_1, C_2,..., C_n , n <=10.

4.1. Варианты заданий приведены в табл. 4.

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

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

4.4. Загрузить в ОЗУ необходимые константы и исходные данные.

4.5. Отладить программу.

Таблица 4. Варианты задания

Номер варианта	**Характеристика последовательности чисел C_1, C_2,.. … C_n**
1	Определить количество четных чисел в последовательности
2	Определить номер (индекс) минимального числа
3	Вычислить произведение всех чисел последовательности
4	Определить номер номер первого отрицательного числа
5	Определить количество чисел, равных C_1
6	Определить количество отрицательных чисел
7	Определить максимальное отрицательное число
8	Определить номер первого положительного числа
9	Определить минимальное положительное число
10	Определить номер максимального числа
11	Определить количество нечетных чисел
12	Определить количество чисел, меньших C_1
13	Определить разность сумм четных и нечетных элементов массивов
14	Определить 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.

Содержание отчета

Включите в отчет:

1. Формулировку варианта задания.
2. Граф-схему алгоритма решения задачи.
3. Распределение памяти (размещение в ОЗУ переменных, программы и необходимых констант).
4. Программу.
5. Значения исходных данных и результат выполнения программы.

5. Оформление результатов.

Оформите результаты выполнения данной работы в виде отчета (в электронном виде), в соответствии с требованиями.

Практическая работа № 5. Подпрограммы и стек

Цель работы:

1. Изучить особенности работы со стеком на примере модели учебной ЭВМ.
2. Изучить использование стека для передачи параметров при вызове подпрограмм.

Теоретические сведения:

1. Подпрограммы и стек

В программировании часто встречаются ситуации, когда одинаковые действия необходимо выполнять многократно в разных частях программы (например, вычисление функции sin (π);). 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 PC) при вызове и использованием 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 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 In First Out, последним пришел – первым вышел) в отличие от дисциплины типа *очередь*– FIFO (First In First Out, первым пришел – первым вышел).

В обычных ОЗУ нет возможности перемещать слова между ячейками, 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.

Задания для выполнения:

1.Ознакомиться с теоретическими сведениями о реализации косвенной памяти в модельной ЭВМ.

2. Запустите программу `CompModel.exe`.

3. Выполнить следующий пример.

Пример.

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. 5.1 приведен текст основной программы и подпрограммы. Обратите внимание, цикл в подпрограмме организован с помощью команды JRNZ, а модификация текущего адреса – средствами постинкрементной адресации.

Таблица 5.1. Программа примера

Команда	**Примечания**
Основная	
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
CALL M	Вызов подпрограммы

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

4. Разработать 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 практической работе №4 (см. табл. 5.2), причем соответствие между номерами вариантов заданий 3 и 4 устанавливается по табл. 5.3.

Таблица 5.2. Варианты задания практической работы №3

Номер варианта	**Характеристика последовательности чисел $C_1, C_2, .. \ldots C_n$**
1	Определить количество четных чисел в последовательности
2	Определить номер (индекс) минимального числа
3	Вычислить произведение всех чисел последовательности
4	Определить номер первого отрицательного числа
5	Определить количество чисел, равных C_1
6	Определить количество отрицательных чисел
7	Определить максимальное отрицательное число
8	Определить номер первого положительного числа
9	Определить минимальное положительное число

10	Определить номер максимального числа
11	Определить количество нечетных чисел
12	Определить количество чисел, меньших C_1
13	Определить разность сумм четных и нечетных элементов массивов
14	Определить 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.

Таблица 5.3. Соответствие между вариантами заданий по практическим работам №3 и №4.

Номер варианта задания 4	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Номер строки в табл. 5.2	5	7	13	11	9	12	1	10	14	3	6	8	2	4

Содержание отчета

Включите в отчет:

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.

5. Оформление результатов.

Оформите результаты выполнения данной работы в виде отчета (в электронном виде), в соответствии с требованиями.

Практическая работа № 6. Командный цикл процессора

Цель работы:

1. Знакомство с принципами работы устройства управления ЭВМ.
2. Изучения принципа микропрограммного управления.
3. Изучение системы команд модельной ЭВМ.
4. Изучение процесса программирования на модели учебной ЭВМ.

Теоретические сведения:

Устройство управления

Процессор ЭВМ условно можно разделить на два основных блока:
- операционный блок;
- управляющий блок.

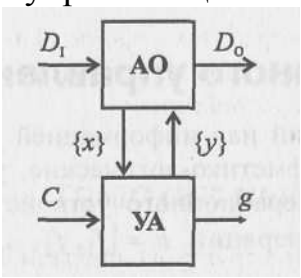

Рис. 6.1. Процессор как композиция операционного и управляющего автоматов.

Для реализации любой команды необходимо на соответствующие управляющие входы операционного устройства процессора подать определенным образом распределенную во времени последовательность управляющих сигналов. Часть процессора, реализующая функции управляющего автомата, предназначенная для выработки этой последовательности, называется устройством управления.

Устройство управления предназначено для выработки управляющих сигналов, под воздействием которых происходит преобразование информации в операционном (арифметико-логическом устройстве), а также операции по записи и чтению информации в/из запоминающего устройства.

Устройства управления делятся на:

- УУ с жесткой, или схемной логикой;
- УУ с программируемой логикой (микропрограммные УУ).

В устройствах управления первого типа для каждой команды, задаваемой кодом операции, строится набор комбинационных схем, которые в нужных тактах вырабатывают необходимые управляющие сигналы (рисунок 6.2).

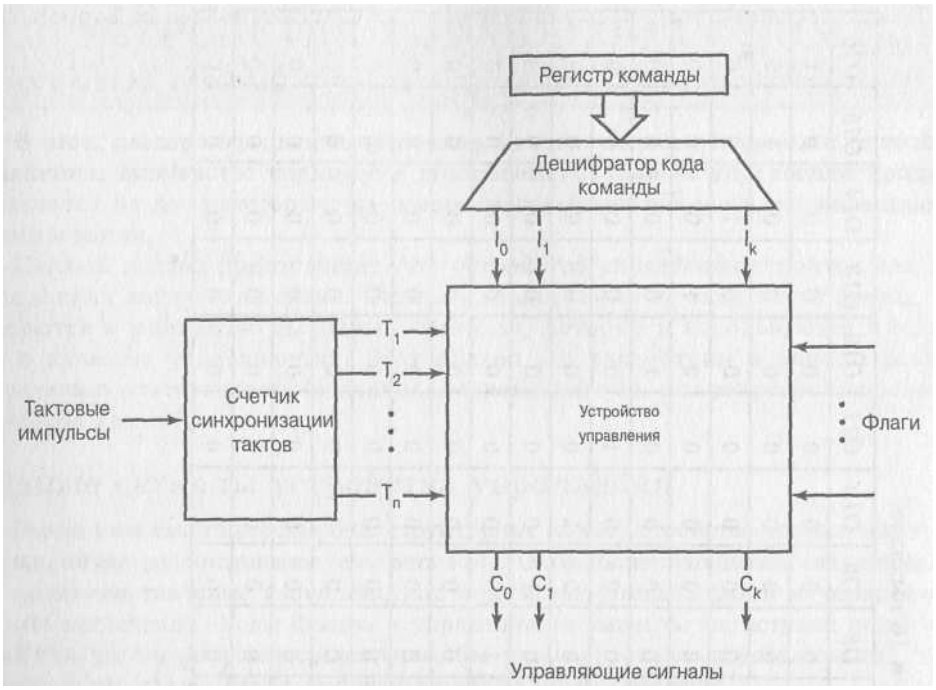

Рис. 6.2. Устройство управления с дешифратором и счетчиком синхронизации тактов

Для примера рассмотрим один из управляющих сигналов – C_5. Предположим, что по этому сигналу данные с внешней магистрали считываются в регистр данных памяти. Значение сигнала C_5 вырабатывается на такте T_2 и определяются в зависимости от двух управляющих сигналов, P и Q, комбинация которых определяет текущую фазу цикла обработки команды:

$PQ = 00$ фаза извлечения

$PQ = 01$ фаза косвенной адресации

$PQ = 10$ фаза выполнения

$PQ = 11$ фаза прерывания

Сигнал C_5 вырабатывается при комбинациях PQ = 00 и 01. Тогда для сигнала C_5 можно записать следующую булеву функцию:

$$C_5 = \neg P \cdot \neg Q \cdot T_2 + \neg P \cdot Q \cdot T_2$$

В соответствии с этой формулой сигнал C_5 будет установлен во втором такте обеих фаз (извлечения и косвенной адресации).

По такому же принципу формируются логические выражения и для остальных управляющих сигналов. В результате будет получено множество булевых выражений, описывающих логику работы устройства управления.

Конечно, для современных процессоров, обладающих сложной системой команд, количество булевых выражений подобного типа должно быть огромным, а сами выражения должны быть очень сложными. Не менее сложной будет и задача построения комбинационной схемы, реализующей эти выражения. Поэтому такой подход применяется только в относительно простых процессорах, вроде Intel 8085. В других же используется более гибкий подход, основанный на идее микропрограммирования

В микропрограммных УУ каждой команде ставится в соответствие совокупность хранимых в специальной памяти слов - микрокоманд. Каждая из микрокоманд содержит информацию о микрооперациях, подлежащих выполнению в данном такте, и указание, какое слово должно быть выбрано из памяти в следующем такте.

Микрокоманды, предназначенные для выполнения некоторой функционально законченной последовательности действий, образуют микропрограмму. Например, микропрограмму образует набор микрокоманд для выполнения команды умножения.

Любое действие, выполняемое в операционном блоке, описывается некоторой микропрограммой и реализуется за один или несколько тактов. Элементарная функциональная операция, выполняемая за один тактовый интервал и приводимая в действие управляющим сигналом, называется микрооперацией

Принцип микропрограммного управления

В микропрограммных УУ каждой команде ставится в соответствие совокупность хранимых в специальной памяти слов - микрокоманд. Каждая из микрокоманд содержит информацию о микрооперациях, подлежащих выполнению в данном такте, и указание, какое слово микропрограммы должно быть выбрано из памяти в следующем такте.

Микрокоманды, предназначенные для выполнения некоторой функционально законченной последовательности действий, образуют микропрограмму. Например, микропрограмму образует набор микрокоманд для выполнения команды умножения.

Любое действие, выполняемое в операционном блоке, описывается некоторой микропрограммой и реализуется за один или несколько тактов. Элементарная функциональная операция, выполняемая за один тактовый интервал и приводимая в действие управляющим сигналом, называется микрооперацией

Структурная схема микропрограммного устройства управления

Микропрограммное устройство управления представлено на рисунке 6.3. Преобразователь адреса микрокоманды преобразует код операции команды, присутствующей в данный момент в регистре команд, в начальный адрес микропрограммы, реализующей данную операцию, а также определяет адрес следующей микрокоманды выполняемой микропрограммы по значению адресной части текущей микрокоманды.

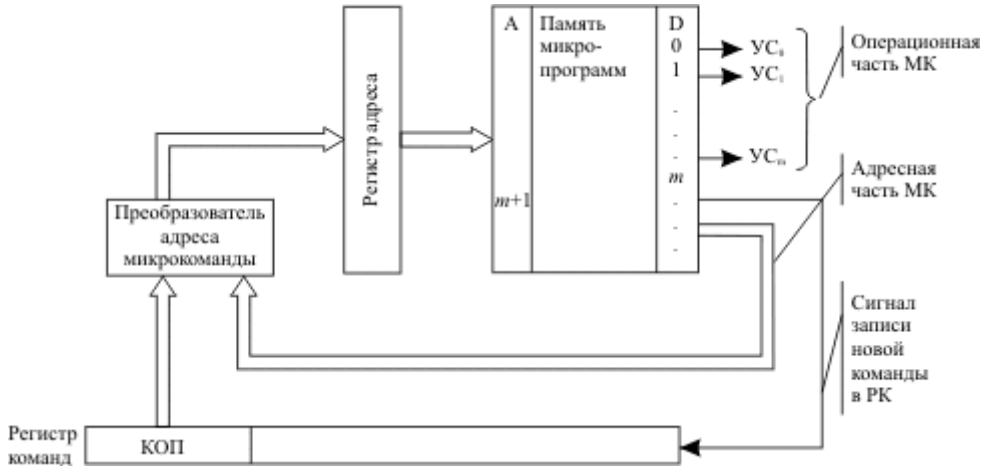

Рис. 6.3. Функциональная схема микропрограммного устройства управления

(УСi – управляющие сигналы, вырабатываемые устройством управления)

Из анализа структуры и принципов работы схемного и микропрограммного устройств управления видно, что УУ первого типа имеют сложную нерегулярную структуру, которая требует специальной разработки для каждой системы команд и должна практически полностью перерабатываться при любых модификациях системы команд. В то же время оно имеет достаточно высокое быстродействие, определяемое быстродействием используемого элементного базиса.

Устройство управления, реализованное по микропрограммному принципу, может легко настраиваться на возможные изменения в операционной части ЭВМ. При этом настройка во многом сводится лишь к замене микропрограммной памяти. Однако УУ этого типа обладают худшими временными показателями по сравнению с устройствами управления на жесткой логике.

Режим микрокоманд модели ЭВМ

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** (рисунок 6.1).

Окно **Микрокомандный уровень** 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".

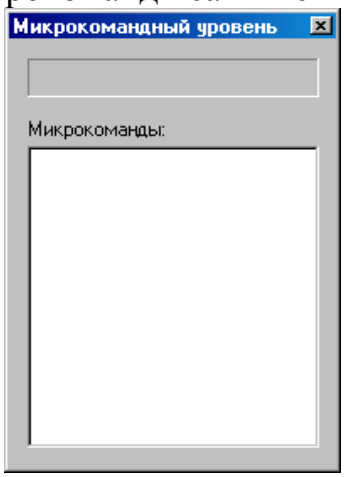

Рис. 6.1. Окно Микрокомандный уровень

Задания для выполнения:

1. Изучить теоретические сведения.

2. Выполнить снова последовательность команд по варианту задания практической работы 2 (таблица 6.1), но в режиме **Шаг.** Зарегистрировать изменения состояния процессора и памяти в форме таблицы 6.2, в которой приведены состояния ЭВМ при выполнении примера 1 (фрагмент).

Таблица 6.1 Варианты заданий

№	**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
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 @013	JMP 001
11	000315	IN	SUB #308	WR11	WR @011	JMP 001
12	X	RD #988	ADD #19	WR9	WR @9	JNZ 001
13	000017	IN	WR11	ADD 11	WR @011	JMP 002
14	X	RD #5	MUL #9	WR10	WR @10	JNZ 001

Таблица 6.2 Результат выполнения задания

Что изменится	**Мнемокод**	**Микрокоманда**	**ОЗУ**		**CR**				АУ	**Ячейки**	
			MAR	**MDR**	**COP**	TA	**ADR**	**Ace**	**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		Ace := 000.ADR									
	WR 30	MAR := PC						000020			
		MRd	001								
		CR := MDR		220030							
		PC := PC + 1			22	0	030				
002		MAR := ADR									
		MDR := Ace	030								
		MWr		000020							
	ADD #5	MAR := PC									000020
		MRd	002								
		CR := MDR		231005							
		PC := PC + 1			23	1	005				
003		DR := 000.ADR									
		F_{Ay} := ALI							000005		
	WR @30	MAR := PC						000025			

3. Оформление результатов.

Оформите результаты выполнения данной работы в виде отчета (в электронном виде), в соответствии с требованиями.

В отчет поместите результаты выполнения задания в виде таблицы 6.2

Практическая работа № 7. Программирование внешних устройств

Цель работы:

1. Знакомство с внешними устройствами модели учебной ЭВМ.
2. Изучение способов организации взаимодействия процессора и внешних устройств (ВУ) в составе ЭВМ.

Теоретические сведения:

1. Структура модели ЭВМ

Моделируемая ЭВМ включает:
- процессор;
- блок регистров общего назначения;
- оперативную память (ОЗУ);
- сверхоперативную память (СОЗУ) (кэш-память);
- устройства ввода (УВв);
- устройства вывода (УВыв).

Процессор, в свою очередь, состоит из:
- центрального устройства управления (УУ);
- арифметического устройства (АУ);
- блока системных регистров (CR, PC, SP и др.).

Структурная схема ЭВМ показана на рисунке 1.

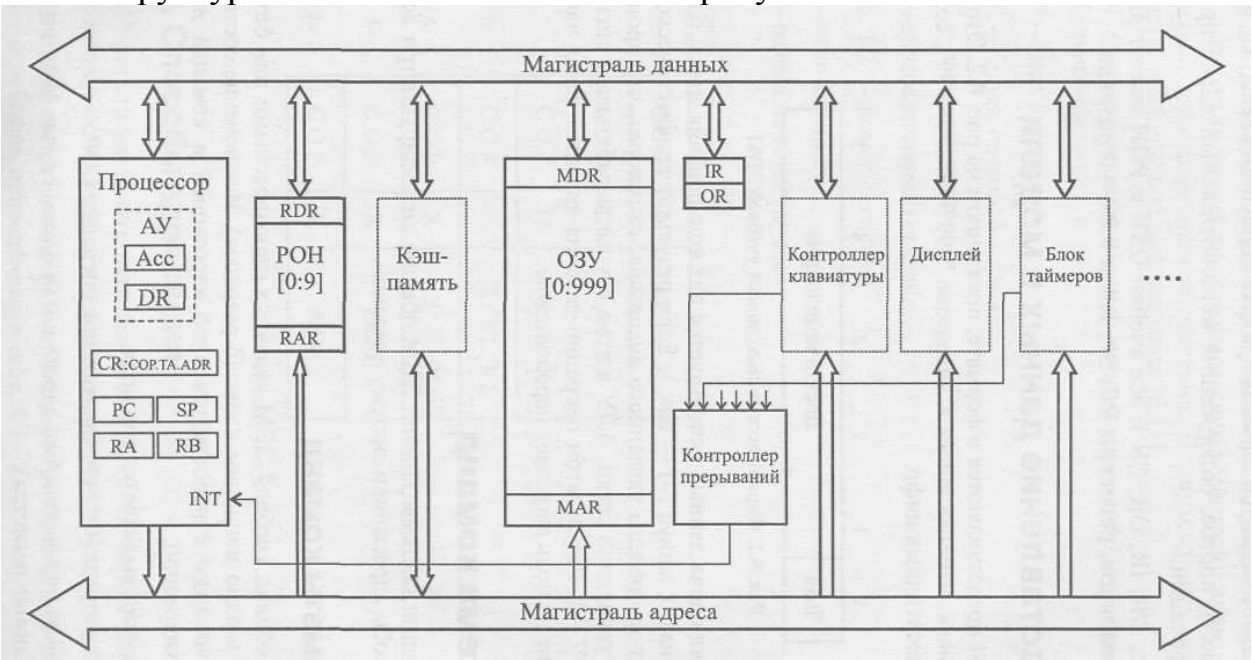

Рис. 1. Общая структура учебной ЭВМ

Представление данных в модели

Данные в ЭВМ представляются в формате, показанном на рис. 2. Это целые десятичные числа, изменяющиеся в диапазоне "-99 999... +99 999", содержащие знак и 5 десятичных цифр.

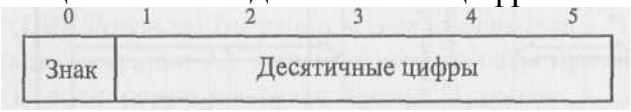

Рис. 2. Формат десятичных данных учебной ЭВМ

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.

2. Работа с внешними устройствами в программе CompModel

Модель учебной ЭВМ реализована в виде программы CompModel.exe, которая находиться в подкаталоге Программы, расположенном в том же каталоге где и текст данной лабораторной работы. В программной модели учебной ЭВМ использован стандартный интерфейс Windows, реализованный в нескольких окнах.

2.1. Внешние устройства модели ЭВМ

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. 1.

Таблица 1. 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* CR 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.

2.1.1. Контроллер клавиатуры

Контроллер клавиатуры (рис. 3) 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 (рис. 4).

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

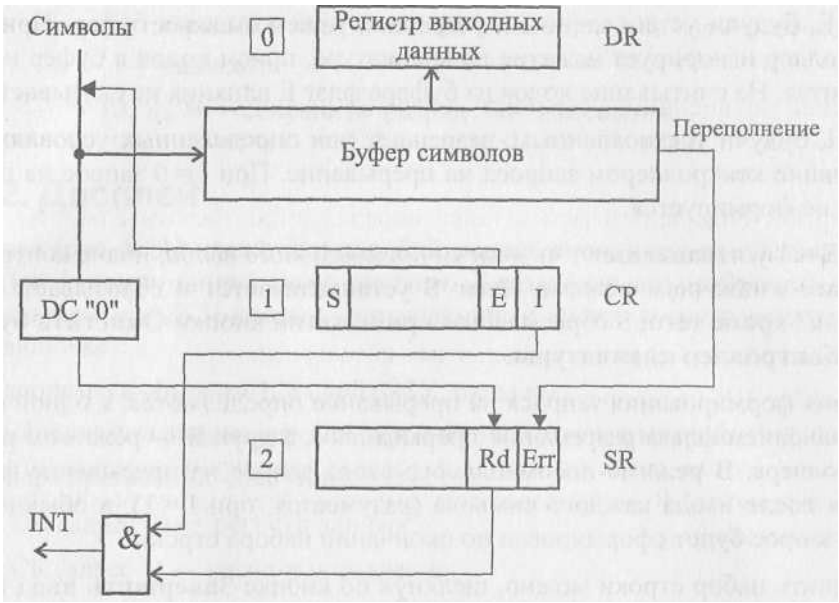

Рис. 3. Контроллер клавиатуры

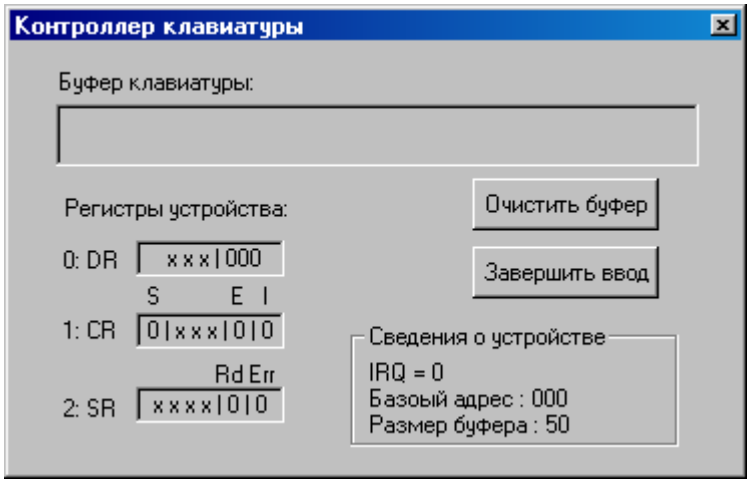

Рис. 4. Окно обозревателя контроллера клавиатуры

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 CR* устанавливаются и сбрасываются программно.

Флаг E, будучи установленным, разрешает прием кодов в буфер. При E = O 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 1 = 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:

- `xxxl0l` – очистить буфер (действие команды эквивалентно нажатию кнопки **Очистить буфер);**
- `xxx 102` – сбросить флаг Err в регистре SR;
- `xxx103` – установить флаг S в регистре CR;
- `xxx 104` – сбросить флаг S в регистре CR.

Если по адресу 1 произвести запись числа xxx0nnc, то произойдет изменение 4-го и 5-го разрядов регистра CR по следующему правилу (1):

$$n = \begin{cases} 0 — \text{записать } 0; \\ 1 — \text{записать } 1; \\ 2, \ldots, 9 — \text{сохранить разряд без изменения.} \end{cases}$$

2.1.2. Дисплей

Дисплей (рис. 5) 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.

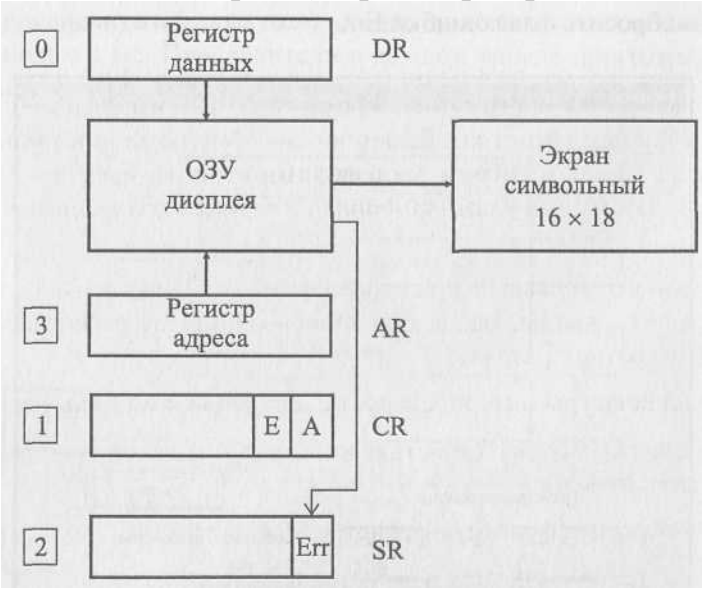

Рис. 5. Контроллер дисплея

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

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

- 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 xxx○nn, при этом изменение 4-го и 5-го разрядов регистра CR произойдет согласно выражению (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:

• `xxxlol` – 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;

• `xxx102` – сбросить флаг ошибки Err.

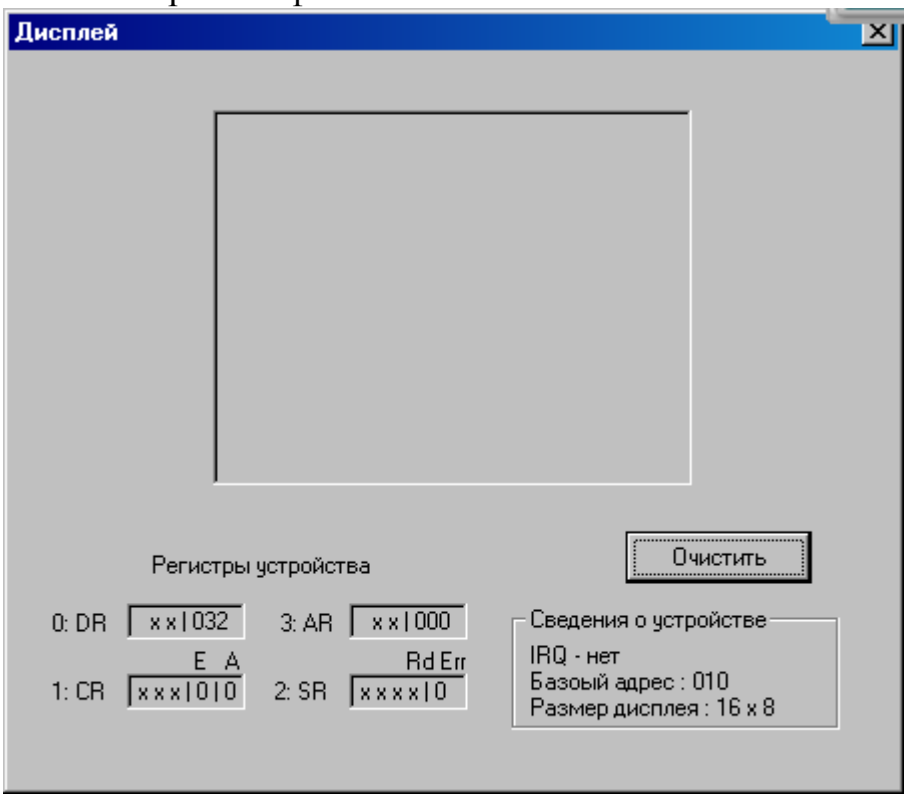

Рис. 6. Окно обозревателя контроллера дисплея

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. 6).

2.1.3. Блок таймеров

Блок таймеров (рис. 7) включает в себя три однотипных канала, каждый из которых содержит:

• пятиразрядный десятичный реверсивный счетчик T, 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 Tl, Т2, ТЗ). Программа в любой момент может считать текущее содержимое таймера или записать в него новое значение.

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.

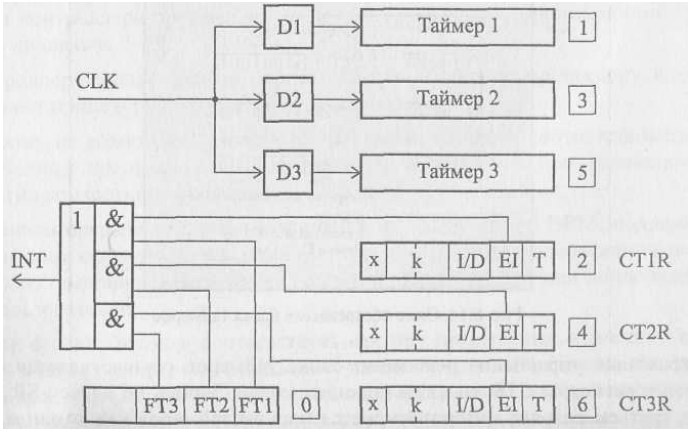

Рис. 7. Блок таймеров

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 = О таймер работает на сложение, при I/D =1 – на вычитание;
- *к* (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 Tl, Т2, ТЗ 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) 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.

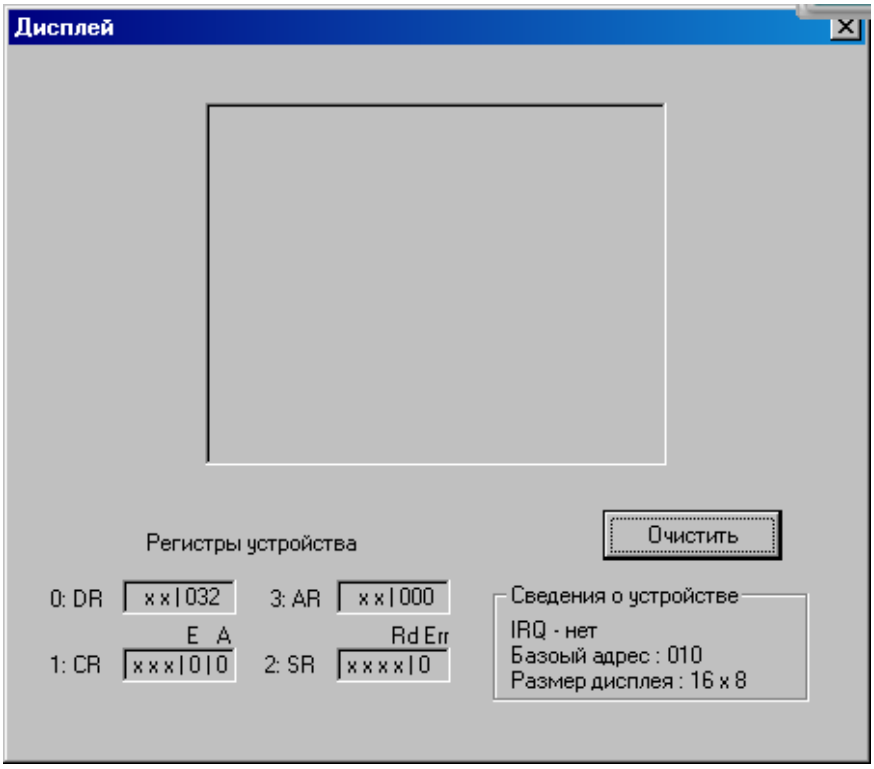

Рис. 8. Окно обозревателя блока таймеров

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);
- xxxl0l – сброс флага переполнения таймера FT1;
- xxx102 – сброс флага переполнения таймера FT2;
- xxx103 – сброс флага переполнения таймера FT3.

2.1.4. Тоногенератор

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

- FR (адрес 0) – регистр частоты звучания (Гц):
- LR (адрес 1) – регистр длительности звучания (мс).

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

2.2. Подсистема прерываний

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

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– она занимает ячейки памяти с адресами 100–109. Таким образом, адрес обработчика с вектором 0 должен располагаться в ячейке 100, с вектором 2 – в ячейке 102. При работе с прерываниями не рекомендуется использовать ячейки 100–109 для других целей.

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

1. Получает от контроллера вектор прерывания.

2. Формирует и помещает в верхушку стека слово, три младших разряда ([3:5]) которого – текущее значение PC (адрес возврата из прерывания), а разряды [1:2] сохраняют десятичный эквивалент шестнадцатеричной цифры, определяющей значение вектора флагов (I, OV, S, Z). Например, если `I=1, OV = ○, S = 1, Z– 1`, то в разряды [1:2] запишется число $11_{10} = 1011_2$.

3. Сбрасывает в 0 флаг разрешения прерывания I.

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

Таким образом, вызов обработчика прерывания, в отличие от вызова подпрограммы, связан с помещением в стек не только адреса возврата, но и текущего значения вектора флагов. Поэтому последней командой подпрограммы обработчика должна быть команда `IRET`, которая не только возвращает в PC три младшие разряда ячейки – верхушки стека (как `RET`), НО И восстанавливает те значения флагов, которые были в момент перехода на обработчик прерывания.

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

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

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. 9, 13), по умолчанию установленный в 0. Если оставить этот флаг в нуле, то внешнему устройству запрещается формировать запрос контроллеру прерываний.

Иногда бывает удобно (например, в режиме отладки) иметь возможность вызвать обработчик прерывания непосредственно из программы. Если использовать для этих целей команду CALL, которая помещает в стек только адрес возврата, то команда IRET, размещенная последней в обработчике, может исказить значения флагов (все они будут сброшены в 0, т. к. команда CALL формирует только три младшие разряда ячейки верхушки стека, оставляя остальные разряды в 000).

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

3. Порядок работы с внешними устройствами модели учебной 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.

4. Вспомогательные таблицы

В данном разделе представлены вспомогательные таблицы (табл. 2–4) для работы с моделью учебной ЭВМ.

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

Таблица 3. Таблица кодов ASCII (фрагмент)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0				0	@	P	‘	p					А	Р	а	р
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	щ	ю
F			/	?	O	_	o						П	Я	п	я

Таблица 4. Перевод шестнадцатиричных кодов в десятичные

	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

Задания для выполнения:

1. Изучить теоретические сведения, посвященные работе внешних устройств модели учебной ЭВМ.

2. Выполнить задание, указанное в таблице 5. Выбор задания выполняется по номеру Вашего индивидуального задания.

Свой вариант задания (табл. 1) 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":

```
M: NOP
   NOP
   JMP M
```

Таблица 5. Варианты задания

№ 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 нецифр 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
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 15с

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 OR. (Учитывая инерционность модели, нажатия не следует производить слишком быстро.)

2.1. Запустить программную модель учебной ЭВМ и подключить к ней определенные в задании внешние устройства (меню **Внешние устройства** > **Менеджер ВУ).**

2.2. Написать и отладить программу, предусмотренную заданием, с использованием программного анализа флагов готовности ВУ.

В отчет поместить тексты программ и скриншоты их работы.

Практическая работа № 8. Знакомство с организацией кэш-памяти учебной ЭВМ

Цель работы:

1. Знакомство с организацией кэш-памяти в учебной ЭВМ.
2. Познакомиться с работой различных алгоритмов замещения строк кэш-памяти.
3. Изучение системы команд модельной ЭВМ.
4. Изучение процесса программирования на модели учебной ЭВМ.

Теоретические сведения:

1. Концепция многоуровневой памяти

Известно, что память ЭВМ предназначена для хранения программ и данных, причем эффективность работы ЭВМ во многом определяется характеристиками ее памяти. Во все времена к памяти предъявлялись три основных требования: большой *объем,* высокое *быстродействие* и низкая (умеренная) *стоимость.*

Все перечисленные выше требования к памяти являются взаимно-противоречивыми, поэтому пока невозможно реализовать один тип ЗУ, отвечающий всем названным требованиям. В современных ЭВМ организуют комплекс разнотипных ЗУ, взаимодействующих между собой и обеспечивающих приемлемые характеристики памяти ЭВМ для каждого конкретного применения.

В основе большинства ЭВМ лежит трехуровневая организация памяти: сверхоперативная (СОП) – оперативная (ОП) – внешняя (ВП). СОП и ОП могут непосредственно взаимодействовать с процессором, ВП взаимодействует только с ОП.

СОП обладает максимальным быстродействием (равным процессорному), небольшим объемом (10^1–10^5 байтов) и располагается, как правило, на кристалле процессорной БИС. Для обращения к СОП не требуются магистральные (машинные) циклы. В СОП размещаются наиболее часто используемые на данном участке программы данные, а иногда – и фрагменты программы.

Быстродействие ОП может быть ниже процессорного (не более чем на порядок), а объем составляет 10^6–10^9 байтов. В ОП располагаются подлежащие выполнению программы и обрабатываемые данные. Связь между процессором и ОП осуществляется по системному или специализированному интерфейсу и требует для своего осуществления машинных циклов.

Информация, находящаяся в ВП, не может быть непосредственно использована процессором. Для использования программ и данных, расположенных в ВП, их необходимо предварительно переписать в ОП.

Процесс обмена информацией между ВЗУ и ОЗУ осуществляется средствами специального канала или (реже) – непосредственно под управлением процессора. Объем ВЗУ практически неограничен, а быстродействие на 3–6 порядков ниже процессорного.

Схематически взаимодействие между процессором и уровнями памяти представлено на рис. 8.1.

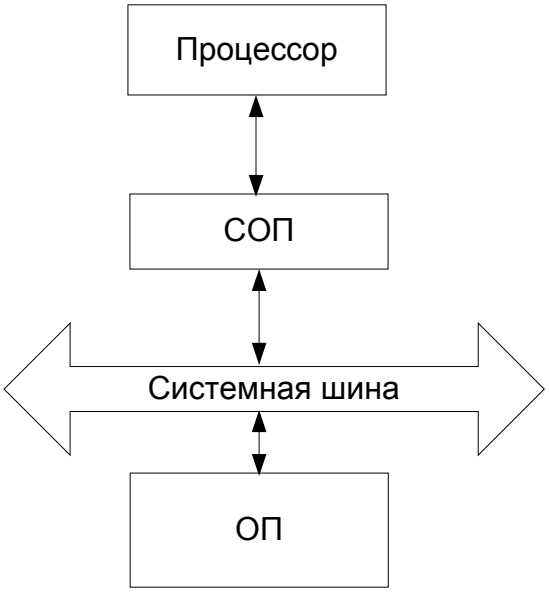

Рис. 8.1. Взаимодействие ЗУ различных уровней в составе ЭВМ

1.1 Виды организации СОП

При организации памяти современных ЭВМ особое внимание уделяется принципам организации СОП и способам обмена информацией между СОП и ОП.

Наибольшее распространение получили следующие три типа организации СОП:

СОП с прямым отображением (direct mapped). В этом случае каждый блок основной памяти имеет только одно фиксированное место в СОП, на котором данные из этого блока могут появиться в кэш-памяти (рисунок 8.2).

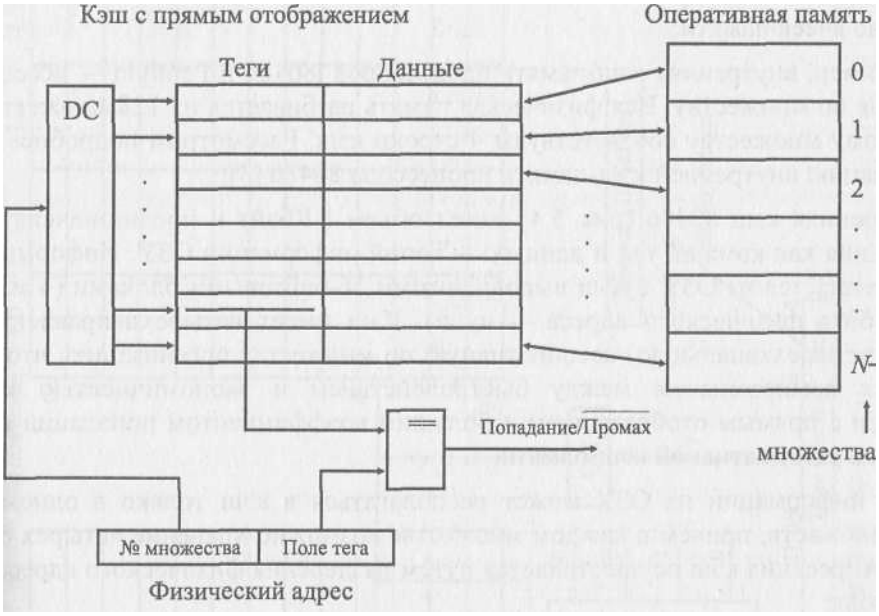

Рис. 8.2. СОП с прямым доступом

Ассоциативная СОП (fully associative). Блок основной памяти может отображаться на любую строку кэш-памяти (рисунок 8.3).

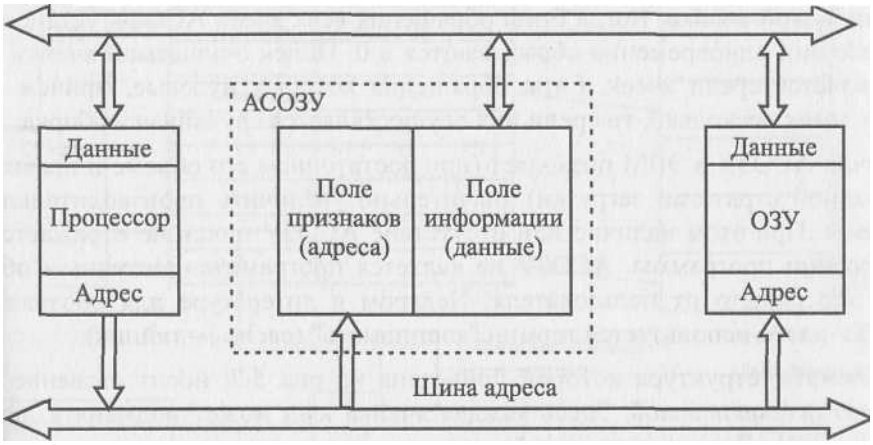

Рис. 8.3. СОЗУ с ассоциативным доступом

Принцип ассоциативного доступа состоит в следующем. Накопитель ассоциативного запоминающего устройства (АЗУ) разбит на два поля – информационное и признаков.

Структура информационного поля накопителя соответствует структуре обычного ОЗУ, а запоминающий элемент поля признаков, помимо функции записи, чтения и хранения бита, обеспечивает сравнение хранимой информации с поступающей и выдачу признака равенства.

Признаки равенства всех элементов одной ячейки поля признаков объединяются по "И" и устанавливают в 1 индикатор совпадения ИС, если информация, хранимая в поле признака ячейки, совпадает с информацией, подаваемой в качестве признака на вход *P* накопителя.

Во второй фазе обращения (при чтении) на выход данных *D* последовательно поступает содержимое информационных полей тех ячеек, индикаторы совпадения которых установлены в 1 (если таковые найдутся).

Множественно-ассоциативная СОП (set associative). Блок основной памяти может располагаться на ограниченном множестве мест (строк) в кэш-памяти. Обычно множество представляет собой группу из двух или большего числа блоков в кэше. Множество определяется младшими разрядами адреса блока памяти (индексом).

Связь между кэш-памятью и основной памятью

У каждого блока в кэш-памяти имеется адресный тег, указывающий, какой блок в основной памяти данный блок кэш-памяти представляет. Эти теги обычно одновременно сравниваются с выработанным процессором адресом блока памяти.

Кроме того, необходим способ определения того, что блок кэш-памяти содержит достоверную или пригодную для использования информацию. Наиболее общим способом решения этой проблемы является добавление к тегу так называемого бита достоверности (valid bit).

Адресация множественно-ассоциативной кэш-памяти осуществляется путем деления адреса, поступающего из процессора, на три части:

- поле смещения используется для выбора байта внутри блока кэш-памяти,
- поле индекса определяет номер множества,
- поле тега используется для сравнения.

Если общий размер кэш-памяти зафиксировать, то увеличение степени ассоциативности приводит к увеличению количества блоков в множестве, при этом уменьшается размер индекса и увеличивается размер тега.

2. Алгоритмы замещения

2.1 Процесс замещения при промахе

Промах – это событие которое наступает в случае обращения к памяти по чтению, при котором требуемые данные отсутствуют в СОП.

При возникновении промаха, контроллер кэш-памяти должен выбрать подлежащий замещению блок.

В случае кэша с прямым отображением на попадание проверяется только один блок и только этот блок может быть замещен.

При полностью ассоциативной или множественно-ассоциативной организации кэш-памяти имеются несколько блоков, из которых надо выбрать кандидата в случае промаха. Как правило для замещения блоков применяются две основных стратегии:

- Случайная - заменяется случайно выбранная строка;
- FIFO – First in, first out – заменяется строка, записанная в кэш раньше остальных;

- LRU – Least recently used – заменяется строка, к которой дольше всего не было обращений;
- LFU – Least frequently used – заменяется строка, с наименьшей частотой обращения.

3. Запись в кэш-память

Организация кэш-памяти может отличаться и стратегией выполнения записи. Когда выполняется запись в кэш-память имеются две базовые возможности:

- сквозная запись (write through, store through) - информация записывается в два места: в блок кэш-памяти и в блок более низкого уровня памяти.
- запись с обратным копированием (write back, copy back, store in) - информация записывается только в блок кэш-памяти. Модифицированный блок кэш-памяти записывается в основную память только когда он замещается.

Для сокращения частоты копирования блоков при замещении обычно с каждым блоком кэш-памяти связывается так называемый бит модификации (dirty bit). Этот бит состояния показывает был ли модифицирован блок, находящийся в кэш-памяти. Если он не модифицировался, то обратное копирование отменяется, поскольку более низкий уровень памяти (например ОП) содержит ту же самую информацию, что и кэш-память.

4. Программная модель кэш-памяти учебной ЭВМ

К описанной программной модели учебной ЭВМ может быть подключена программная модель кэш-памяти, структура которой в общем виде отображена на рис. 8.3, а конкретная реализация кэш-памяти в описываемой программной модели показана на рис. 8.4.

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

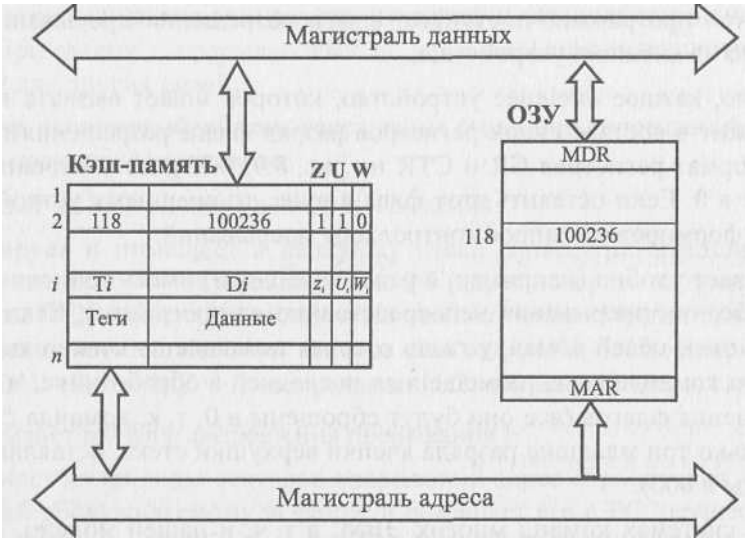

Рис. 8.4. Структура модели кэш-памяти

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. 8.1.

Таблица 8.1. 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 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:= 1). При очистке ячеек, отмеченных битом записи, необходимо переписать измененное значение ноля данных в соответствующую ячейку ОЗУ.

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

В модели реализованы три различных алгоритма замещения строк:

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

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

• *очередь (LRU),* при которой выбор замещаемой ячейки определяется временем пребывания ее в кэш-памяти/

Напомним, что бит использования устанавливается в 1 при любом обращении к ячейке, однако, как только все биты **U** установятся в 1, все они тут же сбрасываются в 0, так что в кэш всегда ячейки разбиты на два непересекающихся подмножества по значению бита U – те, обращение к которым состоялось относительно недавно *(после* последнего сброса вектора U) имеют значение U = 1, иные – со значением U = 0 являются "кандидатами на удаление" при использовании алгоритма замещения *"бит использования".*

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

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

$$K = \frac{S_к - S_{к_w}}{S_o},$$

$$K = \frac{S_к - S^i_{к_w}}{S_o},$$

где:

- K – коэффициент эффективности работы кэш-памяти;
- S_o – общее число обращений к памяти;
- S_K – число кэш-попаданий;
- S_K – число сквозных записей при кэш-попадании (в режиме сквозной записи);
- S^i_K – число обратных записей (в режиме обратной записи).

Задание для выполнения:

1. Изучить теоретические сведения посвященные организации кэш-памяти в модели учебной ЭВМ.

2. Выполнить задание, представленное некоторой короткой "программой" (табл. 8.2), которую необходимо выполнить с подключенной кэш-памятью (размером 4 и 8 ячеек) в шаговом режиме для следующих двух вариантов алгоритмов замещения указанных в таблице 8.3). Номер задания определяется номером варианта.

Таблица 8.2. Варианты задания

№ вари-анта	Номера команд программы						
	1	**2**	**3**	**4**	**5**	**6**	**7**
1	RD #12	WR 10	WR @10	ADD 12	WR RO	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	WR9	ADD #3	WR ©9	CALL 006	POPR4
7	RD #8	WR R2	WR @R2+	PUSH R2	POP R3	WR -@R3	CALL 003
8	RD #13	WR 14	WR @14	WR @13	ADD 13	CALL 006	RET
9	RD #42	SUB #54	WR16	WR @16	WRR1	ADD @R1+	PUSH Rl
10	RD #10	WR R5	ADD R5	WR R6	CALL 005	PUSH R6	RET
11	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, 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.

* Напомним, что программа определяется как последовательность команд, выполнение которых позволяет получить некоторый результат.

Таблица 8.3. 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

Где:

СЗ – случайное замещение запись;

О – очередь;

БИ – бит использования.

2.1. Ввести в модель учебной ЭВМ текст своего варианта программы (см. табл. 8.2), ассемблировать его и сохранить на диске в виде txt-файла.

2.2. Установить параметры кэш-памяти размером 4 ячейки, выбрать режим записи и алгоритм замещения в соответствии с первой строкой своего варианта из табл. 8.3.

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

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

2.5. Для кэш-памяти размером 8 ячеек установить параметры в соответствии со второй строкой своего варианта из табл. 8.4 и выполнить программу в шаговом режиме еще раз, фиксируя последовательность номеров замещаемых ячеек кэш-памяти.

3. Содержание отчета

Отчет о лабораторной работе должен содержать следующие разделы:

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).

4. Оформление результатов.

Оформите результаты выполнения данной работы в виде отчета (в электронном виде), в соответствии с требованиями.

Требования к оформлению отчета по практической работе

Результаты выполнения работы должны быть оформлены в виде отчета, который содержит:

а) Титульный лист;

*Вид титульного листа показан ниже. Строками содержащими знак "–" отмечены границы титульного листа, их не нужно включать в титульный лист.

Наименование дисциплины

Отчет
по практической работе №XX

"Название практической работы"

Выполнил: *Ф.И.О. студент*
Группа: *номер группы*
Курс:*номер курса*

Проверил:
должность и Ф.И.О. преподавателя

Минск 20XX

б) Отчеты по каждому индивидуальному заданию в соответствии с требованиями данного задания.

ЛИТЕРАТУРА

А.П. Жмакин. Архитектура ЭВМ – БХВ-Петербург, 2006. – 315 стр.