МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Кафедра «Компьютерная инженерия»

Методические указания к курсовой работе по программированию

для студентов специальностей
6.050102 "Компьютерная инженерия"
дневной и заочной форм обучения

Составители доц. Назаренко В.И., асс. Демеш Н.С., асс. Юсупова К.Б.

Рассмотрено
на заседании кафедры КИ
Протокол № 9
от 21.04.08 г.

Утверждено
на заседании Учебно-издательского совета ДонНТУ
Протокол № 4 от 10.05.08 г.

Донецк – 2008 г.

Учебное издание

УДК 681.3(07)

Методические указания к курсовой работе по программированию (для студентов специальности 6.050102 «Компьютерная инженерия» дневной и заочной форм обучения). Сост. В.И. Назаренко, Н.С. Демеш, К.Б. Юсупова. – Донецк: ДонНТУ, 2008. – 82 с.

Приведены функциональная схема и система команд учебной ЭВМ М1, детально рассмотрена методика программирования на машинном языке. Описаны содержание и этапы выполнения курсовой работы по проектированию эмулятора учебной ЭВМ. Даны рекомендации по разработке различных блоков эмулятора. В качестве примера приведены основные фрагменты эмулятора ЭВМ М1 на языке Турбо Паскаль.

Составители: доц. Назаренко В.И.
асс. Демеш Н.С.
асс. Юсупова К.Б.

Ответственный
за выпуск проф. Святный В.А.

Рецензент доц. Дацун Н.Н.

В В Е Д Е Н И Е

В настоящее время получило широкое распространение использование микропроцессоров в качестве встроенных элементов систем автоматического управления, в том числе как управляющих блоков периферийных узлов вычислительных комплексов. Функции, возлагаемые в такой системе на микропроцессор, полностью определяются программой, хранимой в его памяти. Поскольку программа работы встроенного микропроцессора изменяется сравнительно редко, к параметрам этой программы и к надежности ее функционирования предъявляются особенно жесткие требования. В связи с этим являются обязательными тщательная отладка и всестороннее тестирование программы работы микропроцессора.

Микропроцессоры имеют, как правило, ограниченные возможности ввода-вывода программ и обрабатываемых данных, что затрудняет выполнение работ по отладке программ. Такие работы также затруднены или вообще невозможны при разработке новых микропроцессорных систем. Всё это стимулирует использование универсальных ЭВМ для отладки и тестирования программ микроЭВМ и микропроцессоров, т.е. использование методики эмуляции программ.

Темой курсовой работы является разработка программы-эмулятора для заданной гипотетической ЭВМ, в упрощенном виде воссоздающей структуру реальной машины. Эмуляция выполняется на ПЭВМ в операционной среде MS DOS, рабочим языком программирования является язык Турбо Паскаль.

1. СОДЕРЖАНИЕ КУРСОВОЙ РАБОТЫ

Тематика курсовых работ связана с созданием программ-эмуляторов для учебных ЭВМ со сравнительно простой структурой. Студенту выдается функциональная схема ЭВМ и набор команд, выполняемых данной машиной. После изучения принципа работы ЭВМ и порядка выполнения её команд студент разрабатывает программную модель на языке Турбо Паскаль и проверяет её работу на тестовых примерах. Такими примерами служат небольшие программы в машинных кодах учебной ЭВМ, организующие вычисления по формулам, согласованным с преподавателем.

По окончании выполнения курсовой работы студент оформляет отчет в соответствии с требованиями, изложенными в настоящих методуказаниях, и защищает его.

График выполнения курсовой работы и содержание отчета регламентируются техническим заданием, которое подписывается студентом и утвер-

ждается руководителем курсовой работы.

Основные этапы разработки программы-эмулятора детально описаны ниже на примере гипотетической ЭВМ, которой условно присвоено название М1. Внимательное изучение предлагаемой методики является обязательным для успешного выполнения курсовой работы. При этом следует иметь в виду, что структуры ЭВМ, выдаваемые студентам в техническом задании к курсовой работе, могут существенно отличаться от структуры машины М1, поэтому механический перенос фрагментов эмулятора М1 на другие структуры ЭВМ в общем случае невозможен.

2 УЧЕБНАЯ ВЫЧИСЛИТЕЛЬНАЯ МАШИНА М1

2.1 Архитектура ЭВМ М1

Структурная схема ЭВМ М1 приведена на рис.1. В состав ЭВМ входят следующие узлы:

- оперативная память MEMORY емкостью 256 8-разрядных ячеек, предназначенная для хранения команд и данных, используемых в программе;
- восьмиразрядный счетчик адреса команд SAK, предназначенный для хранения адреса следующей команды;
- восьмиразрядный регистр адреса RA, определяющий адрес ячейки памяти, в которую записывается или из которой читается информация;
- восьмиразрядный буферный регистр слова RS для временного хранения читаемой или записываемой информации;
- шестнадцатиразрядный регистр команд RK, хранящий выполняемую команду;
- сумматор адреса SA, предназначенный для формирования исполнительного адреса операнда;
- шестнадцатиразрядные операционные регистры OR1 и OR2 для временного хранения операндов;
- блок из четырех шестнадцатиразрядных регистров R с номерами 0, 1, 2, 3;
- арифметико-логическое устройство АЛУ, выполняющее операцию, заданную кодом операции команды;
- шестнадцатиразрядный регистр результата RES, предназначенный для временного хранения результата операции, выполняемой АЛУ;
- дешифратор кода операции DC, осуществляющий настройку АЛУ на выполнение очередной команды;
- четырехразрядный регистр признаков RP.

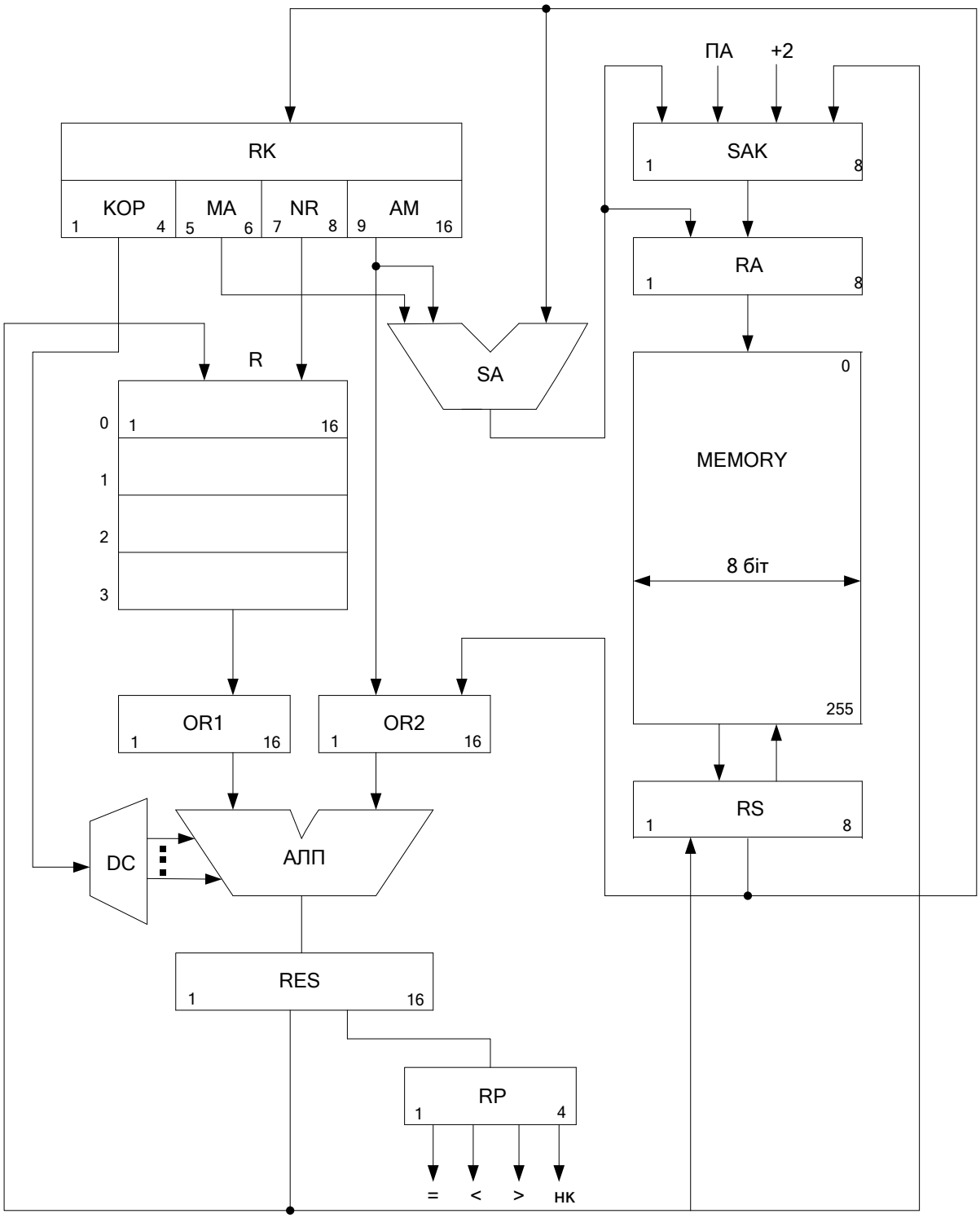


Рис.1. Структурна схема ЕОМ М1

При выполнении арифметической операции первый бит RP устанавливается в «1», если ее результат равен нулю; второй бит устанавливается в «1» при отрицательном, а третий бит – при положительном результате этой операции; четвертый бит устанавливается в «1» в случае некорректности выполнения операции (переполнение регистра результатов RES или попытка деления на нуль).

Для логических операций и для команд сдвига производится лишь анализ результата на равенство нулю: RP[1] = 1, если результат операции равен нулю; RP[2] = 1, если этот результат не равен нулю.

В состав команды входят следующие поля:

- код выполняемой операции KOP (биты 1 – 4);
- модификатор адреса MA (биты 5 и 6);
- номер регистра NR (биты 7 и 8);
- адрес ячейки памяти AM (биты 9 – 16).

Адреса ячеек изменяются от 0 до 255 (от 00000000 до 11111111 в двоичной системе счисления).

Обрабатываемыми данными для арифметических операций являются 16-разрядные целые числа, причем первый разряд определяет знак числа, остальные – двоичные цифры числа. Отрицательные числа представлены в дополнительном коде. В логических операциях также используются 16-разрядные операнды, которые рассматриваются как последовательности битов.

Команды программы и обрабатываемые данные занимают две смежные ячейки памяти, при этом адрес первой ячейки всегда должен быть четным.

В качестве первого операнда в двухместных операциях принимается содержимое регистра R, номер которого определяется значением поля NR регистра команд RK. Формирование второго операнда зависит от типа адресации, определяемой значением модификатора адреса MA:

$MA = 00 \Rightarrow A_{исп} = AM$ – прямая адресация;

$MA = 01 \Rightarrow A_{исп} = MEMORY(AM)$ – косвенная адресация;

$MA = 10 \Rightarrow$ OP2 $= AM$ – непосредственная адресация.

$MA = 11 \Rightarrow A_{исп} = AM + (R_0)$ – индекс-адресация;

Здесь OP2 – условное обозначение второго операнда, которым в данном случае является непосредственно значение поля AM регистра команд RK, а не содержимое ячейки памяти с адресом AM.

В качестве индексного регистра всегда используется нулевой регистр R_0 блока R.

При прямой, косвенной и индекс-адресации $OP2 = MEMORY\ (A_{исп})$, где исполнитеьный адрес $A_{исп}$ определяет адрес левого байта поля длиной два байта. При непосредственной адресации операнды имеют длину один байт, первый операнд размещается в битах 9 – 16 регистра OR1, второй операнд – в битах 9 – 16 регистра OR2. При этом биты 1 – 8 регистров OR1

и OR2 должны быть нулевыми.

2.2 Выполнение команд в ЭВМ М1

Система команд ЭВМ М1, приведенная в табл.1, включает в себя:

- команды выполнения операций с блоком регистров («Загрузка операнда» и «Сохранение операнда»);
- арифметико-логические команды («Сложение», «Вычитание», «Умножение», «Деление», «Сравнение», «Конъюнкция», «Дизъюнкция», «Сложение по модулю 2»);
- команды сдвига («Сдвиг влево логический» и «Сдвиг вправо логический»);
- команды управления порядком выполнения программы («Переход», «Вызов подпрограммы», «Возврат из подпрограммы», «Останов»).

Таблица 1. Система команд ЭВМ М1

Код опер.	Наименование	Признаки				Выполнение команды
		=	<	>	нк	
0000	Останов	–	–	–	–	
0001	Загрузка операнда	+	+	+	–	$OP2 \to R_i$
0010	Сохранение операнда	–	–	–	–	$(R_i) \to A_{исп}$
0011	Сложение	+	+	+	+	$(R_i) + OP2 \to R_i$
0100	Вычитание	+	+	+	+	$(R_i) - OP2 \to R_i$
0101	Умножение	+	+	+	+	$(R_i) * OP2 \to R_i$
0110	Деление	+	+	+	+	$(R_i) * OP2 \to R_i$
0111	Конъюнкция	+	+	–	–	$(R_i) \wedge OP2 \to R_i$
1000	Дизъюнкция	+	+	–	–	$(R_i) \vee OP2 \to R_i$
1001	Сложение по модулю 2	+	+	–	–	$(R_i) \oplus OP2 \to R_i$
1010	Сдвиг влево логический	+	+	–	–	$(R_i) \Leftarrow \quad OP2$
1011	Сдвиг вправо логический	+	+	–	–	$(R_i) \Rightarrow \quad OP2$
1100	Переход	–	–	–	–	См.описание
1101	Сравнение	+	+	+	–	$(R_i) <=> OP2_i$
1110	Обращение к подпрограмме	–	–	–	–	$SAK \to R_i;\ A_{исп} \to SAK$
1111	Возврат из подпрограммы	–	–	–	–	$(R_i) \to SAK$

Круглые скобки в графе «Выполнение команды» означают содержимое регистра или ячейки памяти. Например, запись $A_{исп}$ – это адрес ячейки, а ($A_{исп}$) – содержимое ячейки с адресом $A_{исп}$.

В табл.1 R_i - это i – ый регистр блока R, OP2 – второй операнд, формирование которого зависит от вида адресации.

Команда “Загрузка операнда” пересылает содержимое двух байт оперативной памяти, адрес которых задан значением $A_{исп}$, в регистр R_i, номер которого определяется значением NR. При непосредственной адресации эта команда посылает в биты 9 – 16 регистра R_i (i = 0 .. 3) значение компонента АМ регистра команд RK, при этом биты 1 .. 8 регистра R_i обнуляются. Команда «Сохранение операнда» осуществляет обратную операцию – пересылку данных из регистра R_i в память. При выполнении команды «Загрузка операнда» вырабатывается признак результата, определяющий содержимое соответствующих разрядов регистра RP: RP[1] = 1, если пересылаемое значение равно нулю; RP[2] = 1, если это значение меньше нуля; RP[3] = 1, если оно больше нуля.

В арифметико-логических командах первый операнд выбирается из регистра R_i, а расположение второго операнда определяется полями МА и АМ регистра RK. Признаки завершения операции, которые вырабатываются при выполнении рассматриваемых команд, представлены в табл.1. В частности, признак «нк» (некорректность операции) означает, что в результате выполнения данной операции возникает переполнение регистра результатов Res или же здесь отмечена попытка деления на нуль.

Символ “–” в графе признаков табл.1 определяет ситуацию, которая не может возникнуть в данной операции (например, некорректность при выполнении операции сдвига), или то, что данная операция не изменяет регистр RP (например, команда перехода).

Сущность изменения регистра RP рассмотрим на конкретном примере. Предположим, что в программе выполнена команда сложения и при этом получен положительный результат. Тогда в состояние «1» должен быть установлен лишь третий бит регистра RP, остальные биты должны быть сброшены на нуль.

Примечание. Если в команде, приведенной в табл.1, все графы признаков отмечены символом “–”, то это означает лишь то, что при этом сохраняются предыдущие значения признаков (но не выполняется сброс их на нуль).

Некоторую специфику в данной группе имеет команда сравнения. При ее выполнении вырабатываются только признаки завершения операции: RP[1] = 1, если сравниваемые операнды равны друг другу; RP[2] = 1, если первый операнд меньше второго; RP[3] = 1, если первый операнд больше второго операнда. Содержимое регистра RES при этом не изменяется.

В командах логического сдвига первый операнд OP1 выбирается из регистров R_i, пересылается в регистр OR1, сдвигается влево или вправо, а затем через регистр RES пересылается на тот же регистр R_i. Освобождаемые при выполнении операции разряды в левой или в правой части регистра OR1 заполняются нулями. Количество сдвигов определяется операндом OP2, поступающим на регистр OR2. В зависимости от типа адресации в качестве

операнда OP2 берется исполнительный адрес $A_{исп}$ или значение АМ при непосредственной адресации. Если OP2 > 15, то сдвиг регистра OR1 не производится, но во все его разряды записываются нули.

В команде перехода биты 5 – 8 – это поле маски. Если логическое произведение содержимого маски и содержимого регистра признаков отличается от нуля, то осуществляется переход по адресу $A_{исп}$, т.е. следующей исполняемой командой программы будет команда с адресом $A_{исп}$. Если маска М=1111, то управление передается по адресу $A_{исп}$ вне зависимости от содержимого регистра RP (безусловный переход). Если M = 0000, то выполняется следующая команда (команда перехода в этом случае эквивалентна пустой команде). В команде перехода всегда используется косвенная адресация, т.е. $A_{исп}$ = *MEMORY*(*AM*).

Частные случаи значения маски:

- M = 0000 - пустая команда (нет передачи управления);
- M = 1000 - условный переход по нулю;
- M = 0010 - условный переход по «больше»;
- M = 1100 - условный переход по нулю или по «меньше»;

…….

- M = 1111 - безусловный переход.

Значение M = 0001 определяет переход по признаку некорректности. В этом случае необходимо по адресу АМ перейти на вывод сообщения об аварийном прерывании, указав при этом код операции и адрес команды, вызвавшей прерывание, после чего произвести останов ЭВМ.

По команде «Обращение к подпрограмме» содержимое счетчика адреса команд SAK (адрес следующей команды) временно запоминается в регистре R_i в соответствии с номером NR, а в регистр SAK загружается значение исполнительного адреса $A_{исп}$. Тогда следующей исполняемой командой будет команда с адресом $A_{исп}$ (в данном случае команда входа в подпрограмму).

При выполнении команды возврата из подпрограммы в регистр SAK переписывается содержимое регистра R_i. Эта команда ставится последней в подпрограмме и обеспечивает возврат на продолжение работы основной программы. В данной команде биты 9 – 16 не используются.

В команде останова принимается во внимание лишь код операции. Значения битов 5 – 16 на выполнение команды влияния не оказывают.

В системе команд ЭВМ М1 имеется ограничение на использование непосредственной адресации операндов. Этот вид адресации неприменим для команд 0010, 1100, 1110, 1111 («Запись в память», «Переход», «Вызов подпрограммы», «Возврат из подпрограммы»).

Выполнение машинной программы начинается с команды, адрес которой определяется значением пускового адреса, загруженного в регистр SAK. В дальнейшем содержимое этого регистра указывает адрес каждой очередной выполняемой команды программы. Нормальное завершение работы машинной программы имеет место при достижении команды останова. Если при выполнении какой-либо команды выработан признак некорректности (RP[4] = 1), то это рассматривается как аварийное завершение работы программы; при этом работа программы прерывается, а на экран должно быть выведено сообщение о причине прерывания.

Выполнение любой команды начинается с загрузки содержимого поля памяти, адрес которого задан регистром RK, через буферный регистр RS в регистр команд RK. Для подготовки выборки следующей команды содержимое регистра SAK увеличивается на 2 (каждая команда ЭВМ М1 занимает две смежные ячейки памяти). После этого из регистра RK выделяются поля КОР, МА, NR, АМ. Дешифратор DC анализирует значение кода операции и определяет команду, которая должна выполняться в данный момент (количество входов дешифратора равно количеству разрядов в КОР, количество выходов равно количеству машинных команд).

Примечание.

Если КОП имеет m разрядов, то дещифратор должен иметь 2^m выходов, что однозначно определяет возможное максимальное количество команд данной ЭВМ.

Следующим этапом является формирование исполнительного адреса, если он используется в данной команде. При этом анализируется значение модификатора адреса МА.

Если МА = 00 (прямая адресация), то содержимое поля АМ регистра команд RK передается в регистр адреса RA.

Если МА = 01 (косвенная адресация), то в регистр адреса RA посылается значение АМ+1 (это местонахождение адресной части команды с адресом АМ), по этому адресу выбирается ячейка памяти, ее содержимое поступает в буферный регистр RS, а затем переписывается в регистр RA.

Если МА = 11 (индекс-адресация), значение адреса АМ суммируется в SA с содержимым регистра R_0, результат суммирования передается в регистр RA. По сформированному значению содержимого регистра RA из памяти последовательно выбираются ячейки с адресами $A_{исп}$ и $A_{исп}+1$, содержимое которых затем через регистр RS записывается соответственно в разряды 1 – 8 и 9 – 16 регистра OR2.

При МА = 10 (непосредственная адресация) содержимое поля АМ регистра RK записывается в разряды 9 – 16 регистра OR2. Значение первого операнда из регистра R_i пересылается в регистр OR1. Результат операции формируется в регистре RES и, если это предусмотрено кодом операции, пересылается в тот же регистр R_i.

2.3. Составление программы на машинном языке ЭВМ М1

На первом этапе выполнения курсовой работы необходимо составить три – четыре программы на машинном языке заданной ЭВМ таким образом, чтобы в этих программах были отражены все команды ЭВМ и особенности их выполнения, в частности типы адресации операндов. В последующем эти программы будут использованы как тестовый материал для проверки работы спроектированного эмулятора.

Машинную программу целесообразно сначала записать в условных кодах, что обеспечивает более высокий уровень ее читаемости и уменьшает количество возможных ошибок. При программировании в условных кодах вместо двоичных кодов машинных операций и типов адресации записывают их символические обозначения, вместо двоичного номера регистра – его десятичный номер, вместо реального двоичного адреса операнда – символическое обозначение этого операнда.

Описанная выше методика близка к методике обозначений в языке Ассемблер. Например, команда сложения могла бы иметь следующий вид:

Add I, 2, X

Здесь

- Add – условное обозначение кода операции;
- I – указание на индекс-адресацию;
- 2 – номер регистра для первого операнда;
- X – символический адрес ячейки памяти, в которой содержится переменная X.

Тем не менее указанный способ содержит определенные недостатки, если рассматривать его применительно к разработке эмулятора. Программа на Ассемблере, символически отображающая структуры машинных команд, удобна при ее разработке. Однако дальнейшую работу, в частности преобразование в машинные команды и распределение памяти для команд, переменных и констант, выполняет компилятор. В то же время при разработке эмулятора эту работу приходится выполнять вручную. Поэтому представляется более целесообразным использовать символические обозначения машинных команд, наиболее близкие к структуре этих команд, что облегчает в дальнейшем их окончательное преобразование.

В ЭВМ М1 машинная команда разделяется на 4 поля: код операции, модификатор адреса, номер регистра и адрес операнда.

Код операции целесообразно представлять символически, используя обозначения, которые применяются в языке Ассемблера. В данном случае можно принять:

Halt – останов (код 0000);
Load – загрузка операнда (код 0001);

St – сохранение операнда (код 0010);
Add – сложение (код 0011);
Sub – вычитание (код 0100);
Mul – умножение (код 0101);
Div – деление (код 0110);
And – конъюнкция (код 0111);
Or – дизъюнкция (код 1000);
Xor – сложение по модулю 2 (код 1001);
ShL – сдвиг влево логический (код 1010);
ShR – сдвиг вправо логический (код 1011);
Jump – переход (код 1100);
Cmp – сравнение (код 1101);
Call – вызов подпрограммы (код 1110);
Ret – возврат из подпрограммы (код 1111).

Модификатор адреса имеет лишь 4 значения: 00 (прямая адресация), 01 (косвенная адресация), 10 (непосредственная адресация) и 11 (индекс-адресация). Их нет смысла заменять символическими именами.

Аналогичная ситуация имеет место с номерами регистров, также имеющими 4 значения: 00, 01, 10 и 11.

В команде перехода четырехразрядную маску целесообразно изображать в виде двух групп по два разряда аналогично командам, содержащим модификатор адреса и номер регистра.

Обозначение адреса операнда зависит от вида адресации. При непосредственной адресации значением операнда является 8-разрядное целое неотрицательное число от 0 до 255. Его целесообразно записывать в 10 с/с. При прямой и индекс-адресации нужно указывать адрес операнда. Для этого целесообразно использовать обозначение $< x >$, определяющееадрес поля памяти, где расположена переменная *x*.

Примечание. Если константа в программируемом выражении превышает значение 255 (максиально возможное значение при непосредственной алресации) и равна, например, 310, то соответствующий операнд следует обозначить <310>, что по форме записи будет отличаться от обозначений операнда при непосредственной адресации, т.е. <310> – это не число со значением 310, а адрес ячейки памяти, в которой записано такое число.

Каждая машинная команда имеет конкретный адрес в оперативной памяти, причем этот адрес должен быть четным. Поскольку в командах перехода и вызова подпрограммы используется адрес машинной команды, то слева от символической записи каждой команды будем указывать ее адрес. Тогда приведенная выше команда сложения будет иметь вид

k+10 Add 11 10 < x >,

где *k* – пусковой адрес программы.

Специфику обозначения косвенной адресации рассмотрим на примере команды перехода.

В соответствии с описанием ЭВМ М1 операндом в команде перехода яввляется адрес поля памяти, в котором записан адрес перехода. Предположим, что при реализации разветвляющейся программы необходимо осуществить безусловный переход на машинную команду с адресом k+20. Этот адрес необходимо предварительно записать в какую-либо конкретную ячейку памяти, например, в ячейку k+40, расположенную вне исполняемых команд, во всяком случае после команды останова. Тогда подготовку адреса перехода можно осуществить следующим образом:

```
Load  10  11  k+20   { непосредственная адресация, }
                     { регистр R3 }
St    00  11  k+40   { прямая адресация, регистр R3 }
```

Команда загрузки Load, в которой задана непосредственная адресация, записывает в регистр R_3 значениеадреса перехода, после чего команда сохранения операнда St с прямой адресацией пересылает этот адрес из регистра R_3 в поле памяти с адресом k+40 (здесь имеется в виду, что адрес k+40 находится вне пределов адресов исполняемых команд).

Тогда команда безусловного перехода будет иметь вид

Jump 11 11 k+40, где маска 1111 означает безусловный переход.

Примечание. Здесь уместно напомнить, что записанная выше команда, имея в своем описании косвенную адресацию, выполняет передачу управления не по адресу k+40, а по адресу k+20.

Аналогичным образом подготавливается косвенная адресация и для других машинных команд, но в этом случае модификатор адреса должен иметь значение 01.

Рассмотрим теперь примеры, аналогичные тестам 1, 2 и 3, приведенным в приложении 2 методических указаний.

Пример 1.

$$y = \frac{330\,a + 28\,b}{a - 1} + 8\,a\,b,$$
$$a = 10; \quad b = -4$$

Результат решения должен быть $y = 34$.

```
k+0    Load 00 01  < a >     a → R1
k+2    Sub  10 01    1       a − 1 → R1
k+4    St   00 01    P1      (R1) → P1
k+6    Load 00 01  < a >     a → R1
k+8    Mul  00 01  <330>     330 · a → R1
k+10   St   00 01    P2      (R1) → P2
```

```
k+12   Load 00 01  < b >     b → R1
k+14   Mul  10 01    28      28·b → R1
k+16   Add  00 01    P2      330a + 28b → R1
k+18   Div  00 01    P1      (R1) div (P1) → R1
k+20   St   00 01    P1      (R1) → P1
k+22   Load 00 01  < a >     a → R1
k+24   Mul  00 01  < b >     a·b → R1
k+26   Mul  10 01     8      8·ab → R1
k+28   Add  00 01    P1      (R1) + (P1) → R1
k+30   St   00 01  < y >     (R1) →< y >
k+32   Halt                  останов
k+34      a = 10
k+36      b = -4
k+38      330
k+40      y
k+42      P1
k+44      P2
```

В этой программе имеются 4 константы. Поскольку значения 1, 8, 28 меньше 255, то для их представления в машинных командах используется непосредственная адресация, в то время как для константы 330 отводится отдельная ячейка памяти.

Обозначения R1 – это номер регистра в блоке R, P1 и P2 в поле операнда – это адреса ячеек памяти, предназначенные для временного хранения промежуточных результатов.

Предположим, что в качестве пускового адреса программы задано значение 20, а в качестве исходной системы счисления – 16 с/с. Тогда вначале производится преобразование символьной записи программы в 2 с/с, а затем – в 16 с/с. Фактически такое представление программы – это содержимое входного текстового файла. В первой строке этого файла записывается в 10 с/с пусковой адрес программы, а затем машинные команды, переменные и константы в заданной системе счисления.

Для рассматриваемого примера будем иметь (слева всё, кроме пускового адреса, в 2 с/с, справа – в 16 с/с:

```
      20                    20
0001 00 01 00110110       1136
0100 10 01 00000001       4901
0010 00 01 00111110       213E
0001 00 01 00110110       1136
0101 00 01 00111010       513A
0010 00 01 01000000       2140
0001 00 01 00111000       1138
0101 00 01 00011100       511C
0011 00 01 01000000       3140
0110 00 01 00111110       613E
0010 00 01 00111110       213E
```

```
0001 00 01 00110110       1136
0101 00 01 00111000       5138
0101 10 01 00001000       5908
0011 00 01 00111110       313E
0010 01 01 00111100       253C
0000 00 00 00000000       0000
0000 00 00 00001010       000A    значение a
1111 11 11 11111100       FFFC    значение b
0000 00 01 01001010       014A    значение 330
```

Результат работы программы (k+40) = $33_{10} = 21_{16} = 00100001_2$.

Необходимо обратить внимание, что адреса операндов в этой программе записаны вначале в двоичной, а затем в шестнадцатеричной системе счисления с учетом пускового адреса. Например,

$$<a> = (k+34)_{10} = 54_{10} = 36_{16} = 00110110_2$$

Поля памяти с адресами < y >, P1 и P2 в машинной записи программы не отображены, поскольку их содержимое не вводится извне, а формируется в программе.

Примечание. Если в качестве входной системы счисления задана 8 с/с, то машинное двоичное представление программы разделяется справа налево на триады, после чего каждая триада записывается одной восьмеричной цифрой. Если задана четверичная система счисления, то аналогично производится разделение двоичной записи программы справа налево, но по два разряда, значения которых затем изображаются четверичными цифрами 0, 1, 2 или 3.

Пример 2.

$$y_i = \begin{cases} a \left(x_i - a \right), & \text{если } |x_i| < 6 \\ x_i + 1, & \text{если } x_i \geq 6 \\ a + x_i^3, & \text{если } x_i \leq 6 \end{cases}$$

$i = 1..5; \quad a = 5; \quad x_1 = 3; \quad x_2 = -3; \quad x_3 = 10; \quad x_4 = -6; \quad x_5 = 20;$

Результаты: $y_1 = -10; \quad y_2 = -40; \quad y_3 = 11; \quad y_4 = -211; \quad y_5 = 21$

Блок-схема для примера 2 приведена на рис.2.

На рис.2 звездочкой (x_i^*, y_i^*) обозначены те переменные, адреса которых должны изменяться с шагом 2 в каждом цикле выполнения программы. В начальном состоянии этим переменным назначаются адреса переменных x_1, y_1. Шаг переадресации при индекс-адресации определяется содержимым регистра R_0, значение которого в конце каждого цикла увеличивается на 2.

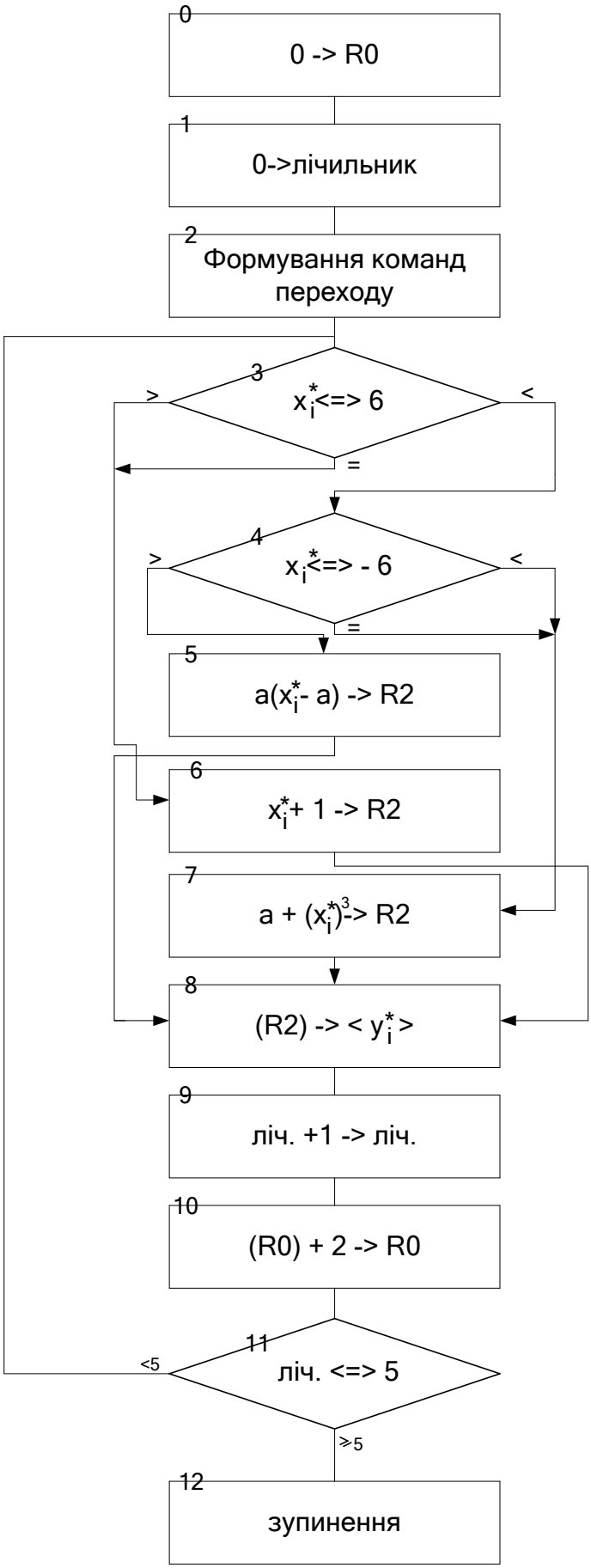


Рис. 2. Блок - схема програми для приклада 2

Блоки на схеме рис.2 изображены как будто в линейном порядке, что визуально совпадает с внешним видом машинной программы, но на самом деле они выполняются не в таком порядке. Блок 3 передает управление на вычислительный блок 6, если $x_i \geq 6$, в противном случае выполняется следующий блок 4, который в свою очередь передает управление на вычислительный блок 7, если $x_i \leq -6$. Если же $x_i > -6$, то выполняется следующий вычислительный блок (блок 8). После каждого вычислительного блока управление передается на блок 8, который записывает полученный результат в ячейку $< y_1^* >$.

Программная реализация примера 2 в условных кодах имеет следующий вид:

```
k+0   Load  00 00     0            0 -> R0
   2   St    00 00   < счетчик >   0 -> < счетчик >
   4   Load  10 01     k+28        подготовка команды
   6   St    00 01     k+100       перехода k+28
   8   Load  10 01     k+32        подготовка команды
k+10   St    00 01     k+102       перехода k+32
  12   Load  10 01     k+38        подготовка команды
  14   St    00 01     k+104       перехода k+38
  16   Load  10 01     k+42        подготовка команды
  18   St    00 01     k+106       перехода k+42
k+20   Load  10 01     k+62        подготовка команды
  22   St    00 01     k+108       перехода k+62
  24   Load  11 10   < x*1 >       xi -> R2
  26   Cmp   10 10     6           (R2) <=> 6?
  28   Jmp   10 10     k+100       УП при xi>= 6 на k+40
k+30   Cmp   00 10   < -6 >        (R2) <=> -6?
  32   Jmp   11 10     k+102       УП при xi<=-6 на k+44
  34   Sub   00 01   < a >         xi - a -> R2
  36   Mul   00 01   < a >         a(xi - a) -> R2
  38   Jmp   11 11     k+104       БП на k+50
k+40   Add   10 10     1           xi + 1 -> R2
  42   Jmp   11 11     k+106       БП на k+50
  44   Mul   11 10   < x*i >       xi * xi -> R2
  46   Mul   11 10   < x*i >       xi * xi^2 -> R2
  48   Add   00 10   < a >         xi^3 + a -> R2
k+50   St    11 10   < y*i >       yi -> < yi >
  52   Add   10 10     2           (R0) + 2 -> R0
  54   Load  00 01  < счетчик >    счетчик-> R1
  56   Add   10 01     1           (R1) + 1 -> R1
  58   St    00 01  < счетчик >    R1 -> < счетчик >
k+60   Cmp   10 00     5           счетчик <=> 5?
    62   Jmp   10 00     k+108     повторение цикла с k+24
  64   Halt                       останов
  66   a = 5
  68   x1 = 3
k+70   x2 = -3
```

```
  72    x3 = 10
  74    x4 = -6
  76    x5 = 20
  78    y1
k+80    y2
  82    y3
  84    y4
  86    y5
  88    счетчик
```

Преобразование двоичной записи машинной программы в заданную систему счисления и формирование входного текстового файла выполняются таким же образом, как и в примере 1.

Пример 3.

В примерах 1 и 2 проверена работа команд Halt, Load, St, Add, Sub, Mul, Div, Cmp, Jump, а также все виды адресации. Остались непроверенными команды And, Or, Xor, Shl, Shr, Call и Ret. В связи с этим в примере 3 будет реализовано обращение к подпрограмме, в работе которой используются логические команды и команды сдвига.

В подпрограмме будет вычисляться следующее выражение:

$$y = (a \ and \ b) \ or \ (a \ shl \ 2) \ xor \ (b \ shr \ 3)$$

В основной программе будет реализовано двукратное обращение к подпрограмме. При первом обращении на место формальных параметров *a* и *b* будут переданы значения переменных *a*1 и *b*1, при втором обращении – значения *a*2 и *b*2. Аналогично выходное значение *y*, получаемое в подпрограмме, будет присвоено соответственно переменным *y*1 и *y*2.

```
k+ 0   Load  00 01   < a1 >
   2   St    00 01   < a >
   4   Load  00 01   < b1 >
   6   St    00 01   < b >
   8   Call  00 03   k+30     { обращение к подпрограмме}
k+10   Load  00 01   < y .
  12   St    00 01   < y1 >
k+14   Load  00 01   < AM >
  16   St    00 01   < a >
  18   Load  00 01   < b2 >
k+20   St    00 01   < b >
  22   Call  00 03   k+30     { обращение к подпрограмме }
  24   Load  00 01   < y .
  26   St    00 01   < y2 >
  28   Halt
k+30   Load  00 01   < a >    { начало подпрограммы }
  32   And   00 01   < b >
  34   St    00 01    P1
  36   Load  00 01   < a >
  38   Shl   10 01     2
k+40   Or    00 01    P1
```

```
  42   St    00 01    P1
  44   Load  00 01   < b >
  46   Shr   10 01     3
  48   Xor   00 01    P1
k+50   St    00 01   < y >
  52   Ret   00 03 00000000   { выход из подпрограммы }
  54       a1 = 3355
  56       b1 - 7767
  58       a2 = 11435
k+60       b2 = 23876
  62       y1
  64       y2
  66       a
  68       b
k+70       y
  72       P1
  74       P2
```

Тестовые примеры фактически предназначены для для проверки корректности работы эмулятора. В связи с этим для каждого из них предварительно должны быть выполнены контрольные расчеты результатов, которые будут получены при безошибочной работе эмулятора.

Ниже приводится детальный поэтапный контрольный расчет для примера 3.

Первое обращение

$a1 = 3355_{10} = D1B_{16}$; $b1 = 7767_{10} = 1E57_{16}$

a1 and b1:

0000 1101 0001 1011
0001 1110 0101 0111
0000 1100 0001 0011

a1 shl 2:

0011 0100 0110 1100

b1 shr 3:

0000 0011 1100 1010

(a1 and b1) or (a1 shl 2):

0000 1100 0001 0011
0011 0100 0110 1100
0011 1100 0111 1111

y1 = (a1 and b1) or (a1 shl 2) xor (b1 shr 3):

0011 1100 0111 1111
0000 0011 1100 1010

0011 1111 1000 0101

$y1 = 3FB5_{16} = 16163_{10}$

Второе обращение

$a2 = 11435_{10} = 2CAB_{16}$; $b2 = 20548_{10} = 5044_{16}$

a2 and b2:
0010 1100 1010 1011
0101 0000 0100 0100
0000 0000 0000 0000

a2 shl 2:
0000 0000 0000 0000

b2 shr 3:
0000 1010 0000 1000 100

(a2 and b2) or (a2 shl 2):
0000 0000 0000 0000

y2 = (a2 and b2) or (a2 shl 2) xor (b2 shr 3):
0000 0000 0000 0000
0000 1010 0000 1000
0000 1010 0000 1000

$y2 = A08_{16} = 2568_{10}$

3 РАЗРАБОТКА ЭМУЛЯТОРА

3.1 Программные модели узлов ЭВМ М1

Моделируемыми узлами ЭВМ М1 являются оперативная память, регистры и дешифратор. Им соответствуют в эмуляторе определенные переменные или массивы. Арифметико-логическое устройство АЛУ и сумматор адреса SA явным образом не моделируются: их функции выполняют соответствующие операторы программы.

Каждая ячейка памяти на машинном уровне – это последовательность битовых элементов. Две смежные ячейки памяти ЭВМ М1 могут интерпретироваться в машине как команда, если их содержимое направлено в регистр команд RK, или как обрабатываемые данные, если это содержимое поступает

в блок регистров R или в операционные регистры OR1, OR2. В последнем случае данные могут обрабатываться как числа, если на дешифратор поступил код арифметической команды, или как последовательности бит, если дешифрирована логическая команда.

В общем случае память – это массив битовых строк. Однако язык Турбо Паскаль не имеет в явном виде средств для изображения и обработки таких строк. В связи с этим могут быть предложены два варианта моделирования памяти.

В первом варианте ячейка памяти рассматривается как строка типа **string**[8], каждый байт которой может принимать значение ‘0’ или ‘1’. Тогда модель памяти может быть представлена в следующем виде:

```
Type
   string8 = string[8];
   MemoryAr = array[0..255] of string8;
Var
   Memory : MemoryAr;
```

Во втором варианте ячейка памяти изображается как множество из восьми элементов. Модель памяти для второго варианта:

```
Type
   set8 = set of 1..8;
   MemoryAr = array[0..255] of set8;
Var
   Memory : MemoryAr;
```

Хотя внутренним представлением множества является битовая строка, но на уровне Паскаль-программы множество не может интерпретироваться как строка бит. Множество типа set8 лишь указывает, входят или не входят в его состав элементы 1 .. 8, что косвенным образом определяет значения 1 или 0 соответствующих разрядов ячейки памяти.

Регистры RS, OR1, OR2, Res, RP и блок регистров R моделируются аналогично памяти (**string** или **set**).

Регистр команд RK, включающий в себя поля KOP, MA, NR и AM, целесообразно моделировать записью, компонентами которой являются модели битовых строк.

Модель регистра RK для первого варианта:

```
Type
   string2 = string[2];
   string4 = string[4];
   string8 = string[8];
   RKType = record
     KOP : string4;
     MA,NR : string2;
```

```
      AM : string8
   end;
Var
   RK : RKType;
```

Модель регистра RK для второго варианта:

```
Type
   set2 = set of 1..2;
   set4 = set of 1..4;
   set8 = set of 1..8;
   RKType = record
     KOP : set4;
     MA,NR : set2;
     AM : set8
   end;
Var
   RK : RKType;
```

Рассмотрим теперь вопрос о модели дешифратора. В общем случае дешифратор – это устройство, имеющее n входных и 2^n выходных шин. Его функцией является активизация выходной шины, порядковый номер которой соответствует значению n-разрядного двоичного числа на входе дешифратора. Моделью дешифратора является оператор Case.

Предположим, что код операции RK.KOP (строка или множество) преобразован в программе в значение целочисленной переменной NumKOP типа byte. Тогда модель дешифратора может иметь вид:

```
Case NumKOP of
   0 : ProcHalt;
   1 : Load;
   2 : Store;
   3 : Add;
.................
  15 : FromSubRoutine;
end;
```

Здесь ProcHalt, Load, Store, Add, …, FromSubRoutine – процедуры, реализующие в эмуляторе машинные операции останова, загрузки операнда, сохранения операнда, сложения, …, выхода из подпрограммы.

3.2 Ввод и преобразование исходных данных

Исходными данными для эмулятора является машинная программа ЭВМ М1 и обрабатываемая этой машиной информация.

Машинную программу целесообразно компоновать в том виде, как это

представлено в п.2.3 для ЭВМ М1. В этом случае как машинные команды, так и обрабатываемые этими командами данные располагаются в непрерывной последовательности ячеек памяти и могут быть введены эмулятором из одного текстового файла.

В зависимости от технического задания исходные данные изображаются в четверичной, восьмеричной или шестнадцатеричной системе счисления. Если в задании определена 16 с/с, то входной текстовый файл может иметь, например, следующий вид:

```
 20
1128
5919
4964
212C
112B
6904
312A
612C
212E
0000
FFE2
03EB
```

Здесь в первой строке текстового файла в 10 с/с записано значение пускового адреса машинной программы, в остальных строках – содержимое ячеек памяти в 16 с/с.

Процедура ввода, входящая в состав эмулятора, должна прочесть содержимое входного файла как последовательность строк, преобразовать их в двоичный эквивалент и заполнить значениями 0 и 1 соответствующие элементы модели памяти Memory. Пример программы ввода для эмулятора ЭВМ М1 (информация в текстовом файле записана в 2 с/с, модель памяти – массив множеств) приведен в прил.3.

В общем случае программа ввода может быть представлена следующим образом (в предположении, что содержимое одной строки текстового файла размещается в одной ячейке памяти):

```
Type
   string80 = string[80];
Var
   n,           { кол-во строк машинной программы}
   Adr,            { адрес ячейки памяти }
   SAK : byte;     { счетчик адреса команд }
     S : string80; { строка входного файла }
   InFile : text;  { входной текстовый файл }
Begin
  Reset(InFile);
  Readln(InFile,SAK);
  n:=1; Adr:=SAK;
```

```
While not Eof(InFile) do
  Begin
    Inc(n);
    Readln(InFile,S);
    Удаление в строке S пробелов перед
         первой цифрой
    Преобразование строки S в двоичный
         эквивалент и запись его в элемент
         Memory с индексом Adr
    Inc(Adr);  { подготовка след.адреса }
  End;
Close(InFile);
```

Преобразование строки S входного файла в битовую строку упрощается тем, что четверичная, восьмеричная и шестнадцатеричная системы счисления имеют основанием степень числа 2. Каждой цифре четверичной записи соответствуют две цифры двоичной записи, для восьмеричной записи – три цифры (триада), для шестнадцатеричной – четыре цифры (тетрада).

Предположим, что ячейка ЭВМ имеет длину 16 бит, а ее содержимое изображено в 16 с/с. Тогда соответствие между шестнадцатеричной и двоичной записями может быть представлено схемой, изображенной на рис.3.

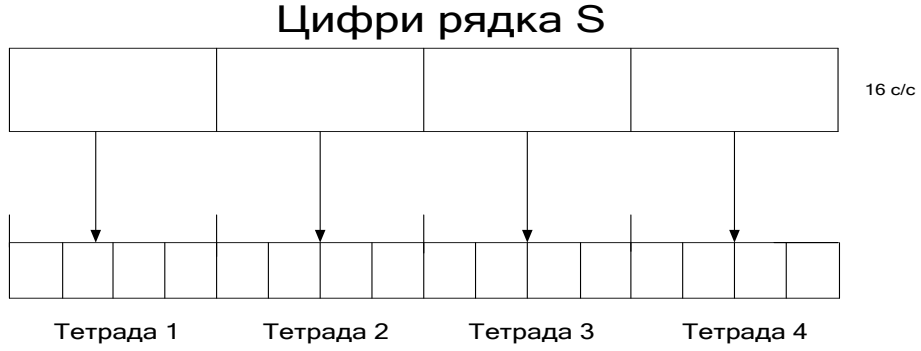


Рис.3. Соответствие между двоичной и шестнадцатеричной записями

Следовательно, программа ввода должна переписать двоичное разложение первой шестнадцатеричной цифры строки S в первую тетраду двоичной записи, второй цифры – во вторую тетраду и т.д. Соответствие между шестнадцатеричной цифрой и ее двоичным эквивалентом в первом варианте модели памяти целесообразно представить в эмуляторе типизированной константой Ekv2.

```
Type
   string4 = string[4];
Const
   Ekv16 = '0123456789ABCDEF';
   Ekv2 : array[1..16] of string4 =
        ('0000','0001','0010','0011','0100','0101',
         '0110','0111','1000','1001','1010','1011',
```

```
'1100','1101','1110','1111');
```

После ввода очередной строки *S* в цикле просматривают символы этой строки, с помощью функции Pos определяют порядковый номер *k* очередного символа и его двоичный эквивалент из типизированного массива Ekv2:

```
Var
   i,k : byte;
    ch : char;
Begin
   For i:=1 to 4 do
     Begin
       ch:=S[i]; k:=Pos(ch,Ekv16);
       Перенос k-го элемента массива Ekv2
       в i-ую тетраду ячейки Memory с
       индексом Adr
     End;
```

При использовании восьмеричной системы счисления для изображения 16-разрядной двоичной строки следует иметь в виду, что поскольку разделение двоичной последовательности на триады производится справа налево, то первая восьмеричная цифра может иметь только значение 0 или 1 и ей соответствует один, а не три бита двоичной строки.

При использовании второго варианта модели памяти преобразование входных строк в двоичный эквивалент имеет аналогичный характер.

3.3. Прямое и обратное преобразования строки или множества в число

Операнды, поступающие в АЛУ из ячеек памяти, представлены в виде строки или в виде множества. При выполнении над ними арифметических операций необходимо предварительно получить численные значения этих операндов, т.е выполнить преобразование строки в число или множества в число. После выполнения арифметической операции ее результат перед записью в память требуется преобразовать в двоичный эквивалент.

Преобразования множества в число и числа в множество представлены в эмуляторе ЭВМ М1 в виде процедур FromSetToNumber и FromNumberToSet (см. прил. 3).

Рассмотрим выполнение указанных преобразований для строк.

В строке *S* символами ‘0’ и ‘1’ изображается двоичное число. Будем считать это число положительным.

Пусть нам требуется преобразовать *n*-разрядную строку *S*, содержащую двоичные цифры числа, в численное значение целой переменной *P*. Учитывая, что первый разряд строки *S* – это знак числа, а значащими являются (*n*-1) цифр, эту строку можно представить в виде

$$S = a_2 2^{n-2} + a_3 2^{n-3} + ... + a_{n-1} 2^0 + a_n,$$

где $a_2, a_3, ..., a_n$ - символы строки S (‘0’ или ‘1’).

Такое представление строки фактически и является схемой ее преобразования в число. Вычисление записанного выше полинома следует проводить по схеме Горнера:

$$P = (...(a_2 \cdot 2 + a_3) \cdot 2 + a_4) \cdot 2 + ... + a_{n-1}) \cdot 2 + a_n,$$

где $a_2, a_3, ..., a_n$ - двоичные цифры 0 или 1 (цифра a_1, определяющая знак числа, равна нулю).

Преобразование числа P в n-разрядную строку S выполняется в обратном порядке по схеме:

$a_2 := P \; div \; 2^{n-2}; \quad P := P \bmod 2^{n-2};$

$a_3 := P \; div \; 2^{n-3}; \quad P := P \bmod 2^{n-3};$

……

$a_{n-1} := P \; div \; 2^1; \quad P := P \bmod 2^1;$

$a_n : p \; div \; 2^0$

Поскольку $2^0 = 1$, то последний оператор эквивалентен следующему:

$a_n := P$

Отрицательные числа представлены в строке S в дополнительном коде. Следовательно, при переводе $S \to P$ требуется в случае необходимости выполнить преобразование дополнительного кода в прямой код, а при преобразовании $P \to S$ – преобразование прямого кода в дополнительный.

Как известно, для получения дополнительного кода отрицательного двоичного числа требуется инверсировать его цифры, а затем добавить единицу к младшему разряду полученного числа. При переходе от дополнительного кода отрицательного числа к его прямому коду из дополнительного кода предварительно вычитается единица. Поскольку арифметические операции над строками невозможны, то эти действия должны производиться по отношению к численным представлениям содержимого преобразуемых строк. Возможные варианты преобразования $строка \to число$ и $число \to строка$ представлены в приведенных ниже процедурах по отношению к 16-разрядным двоичным числам. При рассмотрении указанных процедур следует учитывать, что первый байт строки – это знак двоичного числа.

```
Procedure FromStringToNumber(Operand:string16;
                             Var Number:integer);
{ Преобразование строки в число }
Var   i : byte;
   Cond : boolean;  { признак отрицательного числа }
```

```
Begin
  Cond:=false;
  If Operand[1]='1' then
    Begin
      Cond:=true;
      For i:=1 to 16 do          { инверсирование }
        If Operand[i]='1' then   { цифр отрицатель-}
          Operand[i]:='0'        { ного двоичного }
        Else                     { числа }
          Operand[i]:='1';
    End;
  Number:=0;                    { получение числен- }
  For i:=2 to 16 do             { ного значения по }
    Begin                       { схеме Горнера }
      Number:=2*Number;
      If Operand[i]='1' then
        Inc(Number);
    End;
  If Cond then                { добавление 1, если }
    Begin                     { был преобразован }
      Inc(Number);            { дополнительный код, }
      Number:=-Number         { и изменение знака }
    End;                      { числа }
End { FromStringToNumber };


Procedure FromNumberToString(Number:integer;
                             Var Operand:string16);
{ Преобразование числа в строку }
Var   i : byte;
   k,
   Divisor : integer; { делитель }
   Cond : boolean;    { признак отрицательного числа }
Begin
  Cond:=false;
  If Number<0 then
    Begin               { получение положитель- }
      Cond:=true;             { ного числа и вычи- }
      Number:=-Number;        { тание из него }
      Dec(Number)             { единицы }
    End;

  Divisor:=16384;       { делитель = 2^14   (n=16) }
  Operand:='0';
  For i:=2 to 16 do             { последовательное }
    Begin                       { формирование }
      k:=Number div Divisor;    { двоичных цифр }
      If k>0 then               { числа }
        Operand:=Operand+'1'
      Else
        Operand:=Operand+'0';
      Number:=Number mod Divisor;
```

```
      Divisor:=Divisor shr 1;
    End;
  If Cond then                  { инверсирование }
    For i:=1 to 16 do           { цифр отрицатель- }
      If Operand[i]='1' then    { ного числа  (по- }
        Operand[i]:='0'         { лучение дополни- }
      Else                      { тельного кода) }
        Operand[i]:='1';
End { FromStringToNumber };
```

3.4. Индикация переполнения с фиксированной запятой

Арифметические операции, входящие в систему команд учебной ЭВМ, в эмуляторе выполняются в конечном счете над переменными типа integer или longint: integer, если длина разрядной сетки ЭВМ меньше или равна 16 двоичным разрядам, и longint в противном случае.

При выполнении Паскаль-программы в ПЭВМ переполнение целочисленных переменных не индицируется, т.е. такая ситуация не приводит к прерыванию программы, в том числе при включенной директиве компилятора R. Однако переменная в этом случае получает неправильное значение, что в дальнейшем неизбежно приводит к ошибкам в работе программы.

Рассмотрим следующий фрагмент программы:

```
{R+}
Var  k,l,m,n,s : integer;
     p,q : longint;
Begin
  k:=25000; l:=25000;
  m:=k+l; n:=k*l;
  p:=k+l;
  q:=k; q:=q+l;
  Writeln('m = ',m,'   n = ',n);
  Writeln('p = ',p,'   q = '.q);
  s:=q;
  Writeln('s = ',s);
```

В результате работы программы будет отпечатано:

```
m = -15536   n = -16832
P = -15536   q = 50000
S = -15536
```

В выражении *k+l* оба операнда имеют тип integer, поэтому и значение выражения будет иметь тот же тип. Это объясняет, в частности, почему переменная *p* типа longint получила неправильное значение.

В выражении *q+l* один из операндов имеет тип integer, второй – тип longint. Перед вычислением выражения значение переменной *l* преобразуется

к старшему типу, т.е. типу longint. Вследствие этого переменной *q* присваивается правильное значение, равное 50000.

При вычислении выражений *k+l* и *k*l* директива *R+* не индицирует выход значения выражения за пределы границ, допустимых для типа integer (-32768 .. 32767). Такая индикация (прерывание 201 Range check error) имеет место лишь при выполнении оператора *s := q*, когда производится попытка присвоить переменной s значение вне допустимого диапазона.

В системах команд учебных ЭВМ предусмотрено присваивание регистру признаков RP значения, индицирующего переполнение с фиксированной запятой (ПФЗ). Для того, чтобы эмулятор определил наличие ПФЗ, в процедурах выполнения арифметических операций следует предварительно преобразовать целочисленные операнды к типу real, выполнить соответствующую операцию и сравнить полученный результат с граничными значениями, соответствующими разрядной сетке ЭВМ.

Пусть учебная ЭВМ имеет разрядную сетку, состоящую из 26 двоичных разрядов. Поскольку один разряд при этом отводится для знака числа, то допустимый диапазон изменения целочисленной переменной в этом случае равен

$$-2^{25}..2^{25}-1=-33\,554\,432\ ..33\,554\,431$$

Обозначим значение 33554431 как константу MaxNumber:

```
Const  MaxNumber = 33554431;
```

Тогда индикация ПФЗ в процедуре сложения может быть выполнена следующим образом:

```
Var  Operand1,Operand2,Res : longint;
     a,b,c : real;
Begin
.............................
  a:=Operand1; b:=Operand2;
  c:=a+b;
  If (c<-MaxNumber-1) or (c>MaxNumber) then ...
```

Аналогичные проверки производятся также в процедурах вычитания и умножения.

В процедуре деления ПФЗ не может иметь места, а для индикации деления на нуль должна производиться предварительная проверка целочисленного делителя на равенство его нулю.

3.5. Режимы работы эмулятора

Режимы работы эмулятора определяются естественными требования-

ми, которые возникают при отладке машинной программы.

Является очевидным, что машинную программу гораздо труднее понимать, чем Паскаль-программу; в машинной программе также значительно больше вероятность появления ошибок (неправильный адрес или значение операнда, случайная замена 0 на 1 в машинной команде и т.п.). Поэтому при отладке машинной программы необходимо анализировать состояние функциональных узлов ЭВМ (память и регистры) фактически после выполнения каждой машинной команды.

Приведенные выше соображения требуют включения в состав эмулятора по крайней мере следующих режимов его работы:

1) ввод исходных данных (Input);
2) полное выполнение программы (Run);
3) шаговое выполнение программы (Step);
4) просмотр содержимого памяти (Memory).

Выбор режима работы должен производиться путем передвижения строки-курсора по позициям основного меню.

В режиме Input производится ввод входного текстового файла, формирование значения пускового адреса и заполнение соответствующих ячеек оперативной памяти Memory.

Работа эмулятора может быть проверена путем решения нескольких тестовых примеров (для ЭВМ M1 в п.2.3 приведено три таких теста). Поэтому при проектировании режима Input необходимо предусмотреть ввод по запросу программы имени файла, содержащего требуемый тест.

В режиме Run выполняются все команды машинной программы вплоть до достижения команды останова.

В режиме Step после выполнения каждой очередной машинной команды на экране должно быть отображено состояние всех регистров машины. По отношению к ЭВМ M1 – это регистры SAK, RA, RS, RK, OR1, OR2, Res, RP, блок регистров R. Для регистров SAK и RA достаточно указать в рамке десятичные значения их содержимого, для RP представляется логичным вывести на экран его двоичное значение, для остальных регистров, кроме двоичного изображения их содержимого, целесообразно выводить на экран также соответствующее десятичное значение. Переход к выполнению следующей машинной команды должен осуществляться после нажатия клавиши Enter.

В режиме Memory должен быть организован на экране скроллинг содержимого оперативной памяти ЭВМ (адрес ячейки, двоичное и десятичное значение ее содержимого, а также значение в заданной по варианту системе счисления (4, 8 или 16 с/с). Для улучшения сервиса работы пользователя рекомендуется предусмотреть ввод по запросу эмулятора значений начального и конечного адресов просматриваемой области памяти.

Указанные режимы являются обязательными при проектировании эмулятора любой учебной ЭВМ. Кроме них в техническое задание могут быть включены также дополнительные режимы, например:

1) печать заданной области памяти на принтере (PrintMem);
2) печать на принтере в режиме Step содержимого регистров (PrintStep);
3) перевод числа $10 \to 16$, $10 \to 4$, $10 \to 8$ (Trans10_16, Trans10_4, Trans10_8) ;
4) перевод числа $16 \to 10$, $4 \to 10, 8 \to 10$ (Trans16_10, Trans4_10, Trans8_10);
5) контроль корректности исходных данных (Check) и др.

Остановимся несколько подробнее на режиме Check, в котором должна производиться синтаксическая проверка машинной программы.

Возможные синтаксические ошибки в машинной программе полностью определяются структурой ЭВМ и составом ее системы команд. В частности, по отношению к ЭВМ М1 должны быть выполнены следующие требования:

1) в машинной программе обязательно должна быть команда останова;
2) пусковой адрес должен быть четным;
3) значение пускового адреса должно быть в диапазоне 0 .. 255;
4) адреса операндов должны быть четными;
5) для команды перехода допустима лишь косвенная адресация;
6) для команд St, Call и Ret не должна использоваться непосредственная адресация и др.

При вводе строк исходного текстового файла должны быть предусмотрены проверки, связанные с представлением машинной программы в заданной системе счисления. Такими проверками могут быть:

1) наличие на диске заданного текстового файла;
2) размер входного файла (не является ли он пустым);
3) длина строки текстового файла (количество цифр);
4) отсутствие недопустимых символов в строке (например, символа “5” для 4 с/с или символа “K” для 16 с/с и т.п.).

При обнаружении синтаксических ошибок на экран должно выдаваться сообщение о месте и типе ошибки, после чего работа эмулятора должна прерываться.

3.6. Формирование меню

Наиболее простым способом организации меню является выбор позиции меню с помощью оператора **Case.**

```
Const
  St : array[1..4] of string[32] =
       ('П о з и ц и я  1','П о з и ц и я  2',
        'П о з и ц и я  3','П о з и ц и я  4');
Var  i,Switch : byte;
Begin
  ClrScr;
  For i:=1 to 4 do
    Begin
      GotoXY(2*i+10,20);
      Write(i:2,'  ',St[i]);
    End;
  GotoXY(20,3);
  Write('Укажите номер позиции меню');
  Readln(Switch);
  Case Switch of
    1 : Proc1;
    2 : Proc2;
    3 : Proc3;
    4 : Proc4;
  end;
```

Позиции 1 .. 4 – это наименования режимов работы эмулятора;

Оператор **Case** формирует обращение к одной из процедур Proc1 .. Proc4 в зависимости от введенного значения переменной Switch.

Более совершенным является формирование такого меню, в котором строка-курсор указывает на выбираемую позицию путем ее цветового выделения. При этом клавиши перемещения курсора обеспечивают видимое передвижение по позициям меню, а активизация выбранной позиции производится нажатием заранее определенной управляющей клавиши (например, Enter или Tab). Пример формирования такого меню приведен в тексте эмулятора M1 (прил.3).

При формировании меню в эмуляторе ЭВМ M1 используются процедуры MakeMenu (модуль Emulat), MoveCursor, Frame и InsertColorString (модуль BasUnit).

В процедуре MakeMenu вначале формируется на экране рамка путем обращения к процедуре Frame.

Рамка, которую рисует на экране процедура Frame, может быть окаймлена одинарной (Line = 1), двойной (Line = 2) или широкой (Line = 3) линией. Цвет рамки и фоновый цвет ее внутренней области определяются значением формального параметра Color, которому в данном случае при обращении присваивается значение LightGrayBlack (фон – светлосерый, цвет символа – черный).

После отработки процедуры Frame на экране печатаются строки массива St, определяющие содержание позиций меню. Начальная активизация одной из позиций обеспечивается заданием значений переменным CurrRow (номер текущей строки экрана) и CurrCol (номер текущего столбца экрана),

после чего процедура InsertColorString подсвечивает выделенную позицию цветом BlueYellow (фон – синий, символ – желтый).

Передвижение строки-курсора по позициям меню осуществляется процедурой MoveCursor, с помощью которой можно формировать как вертикальное, так и горизонтальное меню путем задания соответствующих значений ее формальным параметрам. Выход из процедуры MoveCursor производится при нажатии клавиши Enter или клавиши Tab. При этом текущее значение переменной CurrRow указывает положение активизированной позиции меню, после чего с помощью оператора **Case** осуществляется обращение к процедуре, соответствующей этой позиции.

3.7. Организация скроллинга

При вертикальном сдвиге текстового экрана вверх строки 2 .. 25 переписываются в строки 1 .. 24, а в строку 25 вводятся данные из оперативной памяти (содержимое строки типа **string**[80]). Сдвиг вниз осуществляется аналогично.

Вертикальный сдвиг экрана называют также скроллингом (scroll – рулон, свиток).

Пусть для заполнения строки экрана в программе используется переменная BufRow типа ScreenRowArray (см.описание типов в модуле DesUnit, прил.3).

Вертикальный сдвиг вверх:

```
Procedure ScreenToUp;
Var  i : byte;
Begin
  For i:=1 to 24 do
    Screen[i]:=Screen[i+1];
  Screen[25]:=BufRow;
End { ScreenToUp };
```

Вертикальный сдвиг вниз:

```
Procedure ScreenToDown;
Var  i : byte;
Begin
  For i:=25 downto 2 do
    Screen[i]:=Screen[i-1];
  Screen[1]:=BufRow;
End { ScreenToDown };
```

Аналогичным образом можно организовать скроллинг области экрана, ограниченной строками Row1, Row2 и столбцами Col1, Col2.

Для организации скроллинга можно использовать также процедуры In-

sLine и DelLine. Эти процедуры позволяют «прокручивать» часть текстового окна или весь экран вверх и вниз. InsLine вставляет пустую строку на место той, где находится в текущий момент курсор. Все нижние строки, начиная с нее, смещаются вниз на одну строку. Самая нижняя строка уйдет за нижнее поле окна и исчезнет.

Процедура DelLine удаляет строку, в которой находится курсор, подтягивая на ее место все нижестоящие строки. При этом освобождается самая нижняя строка экрана.

Все строки, которые освобождаются при работе процедур InsLine и DelLine, закрашиваются текущим цветом фона.

Пример программы, выполняющей скроллинг текста в окне экрана с помощью процедур InsLine и DelLine, приведен в прил.4.

3.8. Структура эмулятора

Программа эмулятора должна иметь модульную структуру (в противном случае общий объем ее объектного кода не может превышать 64 Кбайт).

Количество модулей и их иерархия в языке Турбо Паскаль не регламентируются. Требуется соблюдать лишь одно требование: в предложениях Uses не должно создаваться перекрестных связей между модулями (например, если в модуле RecUnit стоит фраза "Uses Crt, Dos, TabUnit, VenUnit", а в модуле TabUnit – фраза "Uses Crt, RecUnit, LabUnit").

При выполнении курсовой работы по проектированию эмулятора целесообразно воспользоваться приведенными ниже рекомендациями.

1) Имя модуля в соответствии с требованиями языка Турбо Паскаль должно совпадать с именем файла, который содержит данный модуль. В связи с этим длина имени модуля не может превышать 8 символов. Рекомендуется в качестве последних четырех символов в имени модуля использовать слово Unit (например, DesUnit, BasUnit и т.п.). Это позволяет легко отличать в каталоге файлы с именами модулей.

2) Все глобальные объявления констант, типов и переменных сосредоточить в одном модуле, присвоив ему имя DesUnit (от слова Description – описание).

3) Служебные (базовые) процедуры и функции, используемые в нескольких модулях, сосредоточить в модуле с именем BasUnit. Если при разработке эмулятора используется значительное количество процедур по управлению текстовым режимом экрана, то их целесообразно записать в отдельном модуле (например, с именем ScreUnit; screen – экран).

4) В основную программу "Program Emulat" включить формирование заставки, формирование главного меню, установку в начальное состояние (очистку) узлов эмулируемой ЭВМ и выбор режима работы эмулятора.

5) В одном или в двух модулях разместить процедуры, реализующие режимы работы эмулятора (например, в модуле RegUnit; regime – режим).

6) В заставке указать название программы, учебную группу, фамилию и инициалы студента, название учебной ЭВМ. После вывода на экран заставки должна быть приостановлена работа программы до нажатия клавиши Enter.

7) Все процедуры, непосредственно эмулирующие машинные команды, рекомендуется разместить в отдельном модуле (например, в модуле InsUnit; instruction – команда).

В процессе отладки эмулятора рекомендуется перед именем каждого модуля установить директиву компилятора {$R+}, которая включает проверку корректности присваивания ординальным переменным новых значений, в том числе проверку выхода индекса за границу массива или строки.

Примером использования модулей в программной организации эмулятора является приведенный в приложении 3 эмулятор ЭВМ М1.

3.9. Стиль программирования

Основным принципом в проектировании программ является следующий: программа составляется в первую очередь для человека, а не для машины. Промышленные программные комплексы разрабатывает группа специалистов. В процессе его эксплуатации неизбежно возникает необходимость коррекции и модернизации комплекса, что будет требовать от разработчиков читать и изменять как собственные программы, так и программы их коллег. В связи с этим в текстах разрабатываемых программ обязательно должны содержаться средства, облегчающие понимание программы и улучшающие ее читабельность.

Специальных стандартов на тексты программ не существует. Тем не менее профессиональные программисты в написании текста программы придерживаются определенных правил, направленных на соблюдение приведенного выше принципа программирования.

При написании курсовой работы по проектированию эмулятора обязательным по отношению к тексту программы является выполнение указанных ниже требований.

1) Заголовки модулей и его секций, заголовки программы и каждого ее раздела начинать с первой позиции строки.

2) После заголовка модуля (**`Unit`**) указывать в комментарии основное назначение входящих в модуль процедур и функций.

3) После заголовка процедуры или функции приводить в комментарии краткое описание выполняемой ею работы.

4) Длинные тексты программы разделять комментариями на фрагменты. В комментарии кратко описывать работу фрагмента.

5) После объявления имени переменной в комментарии указывать назначение переменной.

6) Процедуры и функции отделять друг от друга пунктирной линией,

заключенной в комментарий.

7) После слова **End**, завершающего блок процедуры или функции, в комментарии записывать имя данной процедуры или функции.

8) В качестве символов комментария использовать только фигурные скобки.

9) Каждое слово **Begin** начинать с новой строки.

10) Каждое слово **End** записывать с новой строки строго с той позиции, с которой начинается относящееся к нему слово **Begin, Case** или **Record.**

11) Текст программы между словами **Begin** и **End**, **Case** и **End**, **Record** и **End** должен быть сдвинут на один отступ вправо. Отступ равен двум позициям строки.

12) Текст программы после слов **do, then, else** должен начинаться с новой строки, сдвинутой на один отступ вправо.

13) Каждое слово **Else** должно быть расположено под тем словом **If,** к которому оно относится.

14) Разделы описаний, как правило, нужно располагать в том порядке, который предусмотрен стандартным языком Паскаль: **Label, Const, Type, Var**, описание процедур и функций.

15) Имена всех переменных, кроме параметров циклов и индексов элементов массивов и строк, записывать с прописной буквы. Это относится как к простым, так и к составным переменным (массивы, записи, множества и др.).

16) Имена переменных в программе в общем случае не должны отличаться от обозначений, принятых в исходной постановке задачи. Например, было бы недопустимым обозначить в программе эмулятора регистр команд именем Zz, даже снабдив его комментарием, что это имя является обозначение регистра команд.

17) Отдельные части имени, состоящего из нескольких слов, нужно выделять прописными буквами (например, FromSetToNumber).

18) Оператор **Goto** разрешается применять в программе в двух случаях:

а) для принудительного выхода из цикла;

б) для перехода к удаленному фрагменту программы.

Совершенно недопустимо использовать оператор **Goto** для перехода снизу вверх в программе.

Если оператор **Goto** производит переход на конец процедуры или функции, то его целесообразно заменить процедурой Exit. Таким же образом для принудительного выхода из цикла оператор **Goto** можно заменить процедурой Break, а для перехода на конец цикла – процедурой Continue.

19) Длина строки программы не должна превышать 68 символов (для печати текста программы редактором компилятора Паскаль на бумаге формата А4).

20) В программе вместо значений констант использовать, как правило, имена констант.

21) В разделе описания переменных использовать имена типов, а не описания типов. Например, вместо описания

```
Var
   A : array[1..100] of string[40];
```

целесообразно использовать описание

```
Const  Nmax = 100;
Type
    string40 = string[40];
    Ar = array[1..Nmax] of string40;
Var
    A : Ar;
```

Последнее требование связано в основном с тем, что в большой программе любая переменная может быть использована где-либо как фактический параметр при обращениии к процедуре, а при таком обращении всегда требуется совпадение имени типа формального и фактического параметров.

3.10. Описание эмулятора ЭВМ М1

Эмулятор ЭВМ М1 включает в себя основную программу Emulat и четыре модуля: DesUnit, BasUnit, InsUnit и RegUnit.

Модуль DesUnit содержит глобальные описания констант, типов и переменных; модуль BasUnit – служебные процедуры и функции, используемые в других модулях; модуль InsUnit – процедуры реализации машинных команд; модуль RegUnit – процедуры реализации режимов работы эмулятора. Основная программа управляет работой эмулятора.

В начальном фрагменте основной программы производится очистка памяти Memory и регистров эмулируемой ЭВМ. После этого с помощью меню, формируемого процедурой MakeMenu, осуществляется выбор режима работы эмулятора (описание процедур формирования меню приведено в п.3.6).

В эмуляторе ЭВМ М1 предусмотрены следующие режимы работы:

- ввод исходных данных;
- полное выполнение машинной программы;
- шаговое выполнение программы;
- вывод области памяти;
- перевод числа $10 \rightarrow 16$;
- выход из меню.

Ввод исходных данных осуществляется процедурой InputModul. Здесь

из первой строки файла FileModul вводится значение пускового адреса SAK, после чего последовательно вводятся остальные строки файла вплоть до его исчерпания. Содержимое очередной строки файла FileModul, содержащее два машинных слова в двоичной системе счисления, вводится в строковую переменную *S*1, после чего из *S*1 выделяется значащая часть строки, запоминаемая в переменной *S*2. По содержимому строки *S*2 дважды формируется буферная переменная *Slovo* (множество типа set8) и записывается в элементы массива Memory с индексами *k* и *k*+1. Начальное значение индекса *k* равно значению SAK, после ввода каждой очередной строки файла значение переменной k увеличивается на 2.

Режим полного выполнения машинной программы реализуется процедурой Run. Здесь в цикле **`While`** последовательно просматриваются пары ячеек памяти Memory, начиная с адреса SAK. Управление циклом осуществляет булевская переменная CondExec, принимающая значение false при достижении команды останова (код операции 0000).

Каждая пара машинных слов заполняет поля КОР, МА, NR и АМ регистра команд RK. Поля КОР и NR преобразуются в числовое значение переменных Switch и NumReg. После этого по значению переменной Switch с помощью оператора **`Case`** производится обращение к процедуре, которая реализует машинную команду, соответствующую данному коду операции КОР.

В модуле InsUnit, кроме процедур реализации машинных команд, имеются также процедуры формирования исполнительного адреса ExecAddress и формирования содержимого операционного регистра OR2. Указанные процедуры анализируют значение модификатора адреса NR, формируют исполнительный адрес операнда и его значение, записываемое в операционный регистр OR2. При работе процедур реализации машинных команд и формирования машинного адреса выполняется в случае необходимости преобразование множества в число и числа в множество, что осуществляется процедурами FromSetToNumber и FromNumberToSet.

В режиме шагового выполнения программы Step после исполнения каждой машинной команды на экране отображается содержимое регистров ЭВМ М1. Переход к выполнению следующей команды машинной программы производится после нажатия клавиши Enter.

В режиме “Вывод области памяти” осуществляется скроллинг области памяти Memory. Содержимое каждых двух ячеек памяти отображается в 2, 16 и 10 с/с.

В режиме перевода из десятичной в шестнадцатеричную системы счисления осуществляется указанный перевод для числа, задаваемого пользователем с клавиатуры ЭВМ.

Примечание. В прил.3 из методических соображений, а также для сокращения общего объема печатаемого текста приведены лишь основные фрагменты эмулятора ЭВМ М1.

3.11 Рекомендации по разработке эмулятора

Разработку большой программы всегда выполняют путем постепенного наращивания ее “скелета”, последовательно добавляя к ней новые процедуры и модули, расширяющие функции и улучшающие эстетическое оформление проектируемой программы. По отношению к программе-эмулятору рекомендуется выполнять ее разработку в следующей последовательности.

1. Сформировать файл исходных данных, разместив в нем наиболее простой тест (например, $y = a + b$).

2. Создать модули DesUnit, BasUnit, RegUnit и модуль основной программы Emulat.

В модуле DesUnit разместить описания констант, типов и переменных, обеспечивающих работу с оперативной памятью Memory и исходным текстовым файлом.

В модуль BasUnit включить процедуры WaitEnter, PrintString, PrintKeyAndWaitEnter (эти процедуры используются для приостановки решения при отладке программы) и функцию Space (используется при вводе и обработке входного файла).

В модуль RegUnit записать процедуру ввода исходных данных, их преобразования и занесения в ячейки памяти Memory.

Начальные функции основной программы Emulat:

- установка ячеек памяти Memory в нулевое состояние;
- обращение к процедуре ввода исходных данных.

Для отладки процедуры ввода исходных данных и проверки корректности заполнения памяти Memory можно использовать процедуру, аналогичную процедуре WriteMemory (модуль BasUnit эмулятора ЭВМ М1).

3. Добавить в модуль DesUnit объявления регистров ЭВМ, а в начальный фрагмент основной программы – операторы их обнуления. В модуль BasUnit включить процедуры прямого и обратного преобразования строки (множества) в число, проверить работу этих процедур.

4. Добавить в основную программу простое меню (см.п.3.5), заполнив в нем две позиции – “Ввод исходных данных” и “Полное выполнение программы”, в остальных позициях меню поставить пустые операторы.

5. Сформировать модуль InsUnit, включив в него процедуры реализации команд, необходимых для выполнения простого теста.

Разработать процедуру Run, осуществляющую перебор команд машинной программы и обращение к процедурам их реализации; включить эти процедуры в состав модуля RegUnit.

Віполнить отладку разработанных процедур. Для вывода на экран содержимого регистров ЭВМ можно использовать процедуру, аналогичную процедуре WriteRegisters (модуль BasUnit эмулятора ЭВМ М1).

Разработать и отладить процедуры реализации всех оставшихся машинных команд.

6. Заменить в основной программе простое меню более совершенным

типом меню.

7. Разработать и отладить процедуры шагового выполнения программы (с отображением на экране содержимого регистров ЭВМ) и скроллинга содержимого памяти Memory.

8. Модернизировать процедуру ввода исходных данных, включив в нее формирование по запросу имени входного файла.

9. Разработать и отладить процедуру формирования заставки эмулятора.

10. Разработать и отладить процедуры реализации дополнительных режимов работы эмулятора.

11. Произвести тестирование работы эмулятора.

3.12 Содержание пояснительной записки

Пояснительная записка к курсовой работе должна содержать в себе следующие разделы.

1. Введение. Во введении следует определить понятие эмуляции, описать области применения эмуляторов, указать цель выполнения курсовой работы.

2. Описание учебной ЭВМ. Изобразить структурную схему ЭВМ, описать архитектуру машины, привести систему команд и описать последовательность их выполнения.

3. Программирование в кодах учебной ЭВМ. Привести используемые в дальнейшем мнемонические коды операций и других элементов машинной команды, для каждого теста написать программу в условных кодах (с комментариями), программу на машинном языке и содержимое исходного файла в заданной системе счисления, кратко описать программную реализацию тестов.

4. Описание эмулятора. Описать структуру, перечень и назначение программных модулей, режимы работы эмулятора, программную реализацию каждого режима, привести количественные характеристики эмулятора.

5. Заключение. В заключении должны быть кратко подведены итоги работы: что именно выполнено в курсовой работе, какой язык программирования и операционная система использованы при разработке программного продукта, основные характеристики программы (количество модулей, режимы работы, объем исходного и объектного кодов программы, результаты тестирования и пр.).

6. Список использованных источников.

7. Приложение 1. Текст эмулятора.

8. Приложение 2. Результаты тестирования.

Л И Т Е Р А Т У Р А

1. Поляков Д.Б., Круглов И.Ю. Программирование в среде Турбо Паскаль. – М.: Изд-во МАИ, 1992. – 576 с.

2. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс. – М.: “Нолидж”, 1997. – 616 с.

3. Фаронов В.В. Турбо Паскаль 7.0. Практика программирования. – М.: “Нолидж”, 1997. – 432 с.

4. Немнюгин С.А. Turbo Pascal. – СПб: “Питер”, 2001. – 496 с.

5. Немнюгин С.А. Turbo Pascal: практикум. – СПб: “Питер”, 2001. – 256 с.

6. Назаренко В.И. Основы программирования на языке Турбо Паскаль (на дискете). – Донецк, ДонНТУ, 2005. – 395 с.

7. Богумирский Б. Эффективная работа на IBM PC. – СПб: “Питер”, 1995. – 688 с.

Приложение 1

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

НА КУРСОВОЕ ПРОЕКТИРОВАНИЕ

"УТВЕРЖДАЮ:"

Руководитель курсовой работы

_____

"____" _____ 200 г.

Т Е Х Н И Ч Е С К О Е З А Д А Н И Е

Студенту_____гр._____ факультета ВТИ

1. Срок защиты курсовой работы _____
2. Цель работы: выполнить проектирование, отладку и тестирование программы-эмулятора для эаданной учебной ЭВМ на языке Турбо Паскаль.
3. Вариант № _____
4. Тип вычислительной машины _____
5. Система счисления _____
6. Программная модель узлов ЭВМ (string, set) _____
7. Тестовые примеры:

   Тест 1

   Тест 2

   Тест 3

8. Дополнительный режим эмулятора:
9. Содержание пояснительной записки:

   Введение.

   1. Учебная вычислительная машина _____ .
      1.1. Архитектура ЭВМ.
      1.2. Форматы представления информации и команд.
      1.3. Система команд.
      1.4. Принцип работы ЭВМ.
   2. Программирование в кодах учебной ЭВМ.
      2.1. Программирование формул.
      2.2. Циклическая программа с разветвлениями.
      2.3. Выполнение логических команд с использованием подпрограмм.
   3. Программная модель учебной ЭВМ.
      3.1. Назначение и область применения.

3.2. Технические характеристики программы.

3.3. Описание программы.

Заключение.

Приложение. Текст программы-эмулятора.

10. График выполнения курсовой работы.

1-я неделя: выдача задания, изучение системы команд, составление тестовых примеров в условных кодах и на машинном языке.

2-я неделя: разработка логической структуры эмулятора, выбор форматов представления входной, промежуточной и выходной информации, формирование входных текстовых файлов.

3-я неделя: разработка основной программы и процедур ввода исходных данных.

4-я неделя: отладка основной программы и процедур ввода исходных данных; разработка процедур выполнения машинных команд.

5-я неделя: отладка процедур выполнения машинных команд; разработка процедур реализации основных режимов работы эмулятора.

6-я неделя: отладка процедур реализации основных режимов работы эмулятора.

7-я неделя: разработка и отладка процедур реализации дополнительных режимов работы эмулятора.

8-я неделя: тестирование программы эмулятора.

9-я неделя: оформление пояснительной записки.

10-я неделя: представление курсовой работы на кафедру и ее защита.

Подпись студента _____

ВАРИАНТЫ ЗАДАНИЙ

1. Тест 1

Содержанием теста 1 является вычисление арифметического выражения. Значение результата приводится в каждом варианте для контроля правильности решения, получаемого при эмуляции машинной программы.

Примечание 1. Если в системе команд учебной ЭВМ имеются команды арифметического сдвига, то их рекомендуется применять для замены умножения или деления на коэффициенты, являющиеся степенью числа 2. Применять с этой целью команды логического сдвига можно лишь по отношению к положительным операндам; для отрицательных операндов в этом случае будут получены неправильные результаты.

Примечание 2. В примерах теста 1 результатом деления является целая часть получаемого частного, что соответствует в Паскале операции **`div`**.

1.1. $y = \dfrac{18a - 486b - 25}{8a + b}$; $\quad a = 10; \quad b = -2; \quad y = 14$

1.2. $y = \dfrac{320\,a + 25b}{a - 1} + 6\,a\,b$; $\quad a = 8; \quad b = -5; \quad y = 107$

1.3. $y = \dfrac{10\,a\,b - 333(a + b)}{16a - 4b}$; $\quad a = 5; \quad b = -8; \quad y = 5$

1.4. $y = \dfrac{328a^2 - 6b^2 - 25}{32a + 10b} + 1$; $\quad a = 4; \quad b = -8; \quad y = 101$

1.5. $y = \dfrac{526(a - 1) + 16b}{10a - 12b} - 25$; $\quad a = 11; \quad b = -10; \quad y = -3$

1.6. $y = \dfrac{18(a^2 + 1) - 288b}{22a + 8b} - 60$; $\quad a = 9; \quad b = -10; \quad y = -24$

1.7. $y = \dfrac{64(a + b^2) - 1}{2(a + b)} - 371$; $\quad a = 14; \quad b = -4; \quad y = -276$

1.8. $y = \dfrac{3a^2 + 128(a+b)}{a - b^2} - 1000$; $a = 20;\quad b = -10;\quad y = -1031$

1.9. $y = \dfrac{222(a-b) + 8a^2}{16a + b} - 500;$ $a = 10;\quad b = -10;\quad y = -466$

1.10. $y = \dfrac{777(a - 2b) - 8b^2}{32a - 2b} - 100;$ $a = 10;\quad b = -5;\quad y = -54$

1.11. $y = \dfrac{64(a - b^2) - 321b}{8a + 10} - 300;$ $a = 10;\quad b = -20;\quad y = -506$

1.12. $y = \dfrac{26(a-b)^2 - 128(a+b)}{16a + 2} + 10;$ $a = 10;\quad b = -20;\quad y = 162$

1.13. $y = \dfrac{321(a-b) + 32b}{2(a+b)^2}$; $a = 20;\quad b = -10;\quad y = 46$

1.14. $y = \dfrac{64a - 291(b^2 - a)}{21a + 1} + 12;$ $a = 20;\quad b = -10;\quad y = -40$

1.15. $y = \dfrac{135(a^2 + b^2) - 32(a+b)}{2(a-b)^2} - 100$; $a = 10;\quad b = -10;\quad y = -67$

2. Т е с т 2

Содержанием теста 2 является организация циклической программы с разветвлениями.

2.1.
$$y_i = \begin{cases} 2x_i + 1, & \text{если } x_i > 10 \\ 2x_i, & \text{если } x_i = 10 \\ 2x_i - 1, & \text{если } x_i < 10 \end{cases}$$

$i = 1..3; \quad x_1 = 50; \quad x_2 = 10; \quad x_3 = -50$
$y_1 = 101; \quad y_2 = 20; \quad y_3 = -101$

2.2.
$$y_i = \begin{cases} 1 - 2x_i, & \text{если } x_i > 2 \\ 1, & \text{если } x_i = 2 \\ 1 - x_i, & \text{если } x_i < 2 \end{cases}$$

$i = 1..3; \quad x_1 = 75; \quad x_2 = 2; \quad x_3 = -75$
$y_1 = -149; \quad y_2 = 1; \quad y_3 = 76$

2.3.
$$y_i = \begin{cases} a(1 + x_i), & \text{если } x_i > a \\ a, & \text{если } x_i = a \\ 10a / x_i, & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 50; \quad x_2 = -20; \quad x_3 = -50$
$a = -20; \quad y_1 = -1020; \quad y_2 = -20; \quad y_3 = 4$

2.4.
$$y_i = \begin{cases} (10 + a)\ x_i, & \text{если } x_i > a \\ 10 + a, & \text{если } x_i = a \\ (10 + a) / x_i, & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 50; \quad x_2 = 30; \quad x_3 = -10$
$a = 30; \quad y_1 = 2000; \quad y_2 = 40; \quad y_3 = -4$

2.5.
$$y_i = \begin{cases} (5a - 1)/(x_i + 1), & \text{если } x_i > a \\ 5a - 1, & \text{если } x_i = a \\ (a - 1)(x_i + 1), & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 90; \quad x_2 = 50; \quad x_3 = -20$
$a = 50; \quad y_1 = 2; \quad y_2 = 249; \quad y_3 = -931$

2.6.
$$y_i = \begin{cases} (x_i + a)/(x_i - a), & \text{если } x_i > a \\ a, & \text{если } x_i = a \\ (x_i + a)\ (x_i - a), & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 100; \quad x_2 = 60; \quad x_3 = -30$

$a = 60; \quad y_1 = 4; \quad y_2 = 60; \quad y_3 = -2700$

2.7.

$$y_i = \begin{cases} (3a + x_i)\,(a - x_i), & \text{если } x_i > a \\ x_i, & \text{если } x_i = a \\ (3a - x_i)\,/\,(a + x_i), & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = -1; \quad x_2 = 10; \quad x_3 = 30$

$a = 10; \quad y_1 = 3; \quad y_2 = 10; \quad y_3 = -1200$

2.8.

$$y_i = \begin{cases} (x_i - a)\,x_i, & \text{если } x_i > a \\ x_i, & \text{если } x_i = a \\ x_i - a, & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = -100; \quad x_2 = 100; \quad x_3 = 120$

$a = 100; \quad y_1 = -200; \quad y_2 = 100; \quad y_3 = 2400$

2.9.

$$y_i = \begin{cases} a(x_i + a), & \text{если } x_i > a \\ x_i + a, & \text{если } x_i = a \\ x_i, & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 40; \quad x_2 = 20; \quad x_3 = -20$

$a = 20; \quad y_1 = 1200; \quad y_2 = 40; \quad y_3 = -20$

2.10.

$$y_i = \begin{cases} (1 + a)\,(1 + x_i), & \text{если } x_i > a \\ 1 + a, & \text{если } x_i = a \\ 1 + x_i, & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 100; \quad x_2 = 50; \quad x_3 = -100$

$a = 50; \quad y_1 = 5151; \quad y_2 = 51; \quad y_3 = -99$

2.11.

$$y_i = \begin{cases} a\,(a - x_i), & \text{если } x_i > a \\ 0, & \text{если } x_i = a \\ -a\,x_i, & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 40; \quad x_2 = 20; \quad x_3 = -20$

$a = 20; \quad y_1 = -400; \quad y_2 = 0; \quad y_3 = 400$

2.12.
$$y_i = \begin{cases} x_i\ (x_i - a), & \text{если } x_i > a \\ 1, & \text{если } x_i = a \\ x_i - a, & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 100; \quad x_2 = 30; \quad x_3 = -100$
$a = 30; \quad y_1 = 7000; \quad y_2 = 1; \quad y_3 = -130$

2.13.
$$y_i = \begin{cases} 1 + a\,x_i, & \text{если } x_i > a \\ 1 + a, & \text{если } x_i = a \\ (1 + x_i)\ /\ (a - 1), & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 100; \quad x_2 = 40; \quad x_3 = -100$
$a = 40; \quad y_1 = 4001; \quad y_2 = 41; \quad y_3 = -2$

2.14.
$$y_i = \begin{cases} (x_i - 1)\ (a + 1), & \text{если } x_i > a \\ a + 1, & \text{если } x_i = a \\ (x_i + 1)\ /\ (a - 1), & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 80; \quad x_2 = 25; \quad x_3 = -80$
$a = 25; \quad y_1 = 2054; \quad y_2 = 26; \quad y_3 = -3$

2.15.
$$y_i = \begin{cases} a\ x_i + 1, & \text{если } x_i > a \\ 1, & \text{если } x_i = a \\ a\ x_i - 1, & \text{если } x_i < a \end{cases}$$

$i = 1..3; \quad x_1 = 100; \quad x_2 = 70; \quad x_3 = -70$
$a = 70; \quad y_1 = 7001; \quad y_2 = 1; \quad y_3 = -4901$

3. Тест 3

В тесте 3 необходимо осуществить проверку выполнения логических команд, команд логического сдвига и способа организации подпрограмм с использованием соответствующих команд.

Учебные ЭВМ имеют различный набор логических команд и команд сдвига, а также различную длину слова памяти. В связи с этим не представляется возможным составить тесты, общие для всех учебных ЭВМ. Тест 3 студент составляет самостоятельно и согласовывает его с преподавателем при утверждении технического задания. Обязательным для теста 3 является выполнение следующих условий:

- в состав машинной программы входит одна подпрограмма;
- в машинной программе дважды производится обращение к подпрограмме, но

для разных имен (адресов) входных и выходных переменных;

- подпрограмма реализует вычисление выражения, в состав которого входят все логические операции и операции логического сдвига, включенные в состав системы команд учебной ЭВМ.

Ниже приводится пример теста для ЭВМ В2, в систему команд которой входят команды логического умножения (**and**), логического сложения (**or**), суммы по модулю 2 (**xor**), отрицания (**not**) и сдвига логического вправо (**shr**). Слово памяти ЭВМ В2 имеет длину 32 бита.

```
y = (a shr 3) and b or (not c) xor (d shr 2);
z = (m shr 3) and n or (not p) xor (q shr 2);
a = AB08FC97;  b = 12EC38F4;  c = 563F2ECD;
d = 4950F036;
m = ABCDEF01;  n = 12345678;  p = FEA97643;
q = 345ABC01.
```

Результаты решения:

```
y = ABB4E5BF; z = 1C6122FC.
```

Рассмотрим порядок получения результата при вычислении значения y(a,b,c,d).

```
a = 1010 1011 0000 1000 1111 1100 1001 0111 = AB08FC97
a shr 3 = 0001 0000 0110 0000 0001 1000 1001 0000 = =10601890
c = 0101 0110 0011 1111 0010 1110 1100 1101 = 563F2ECD not c =
1010 1001 1100 0000 1101 0001 0011 0010 = =A9C0D132
d = 0100 1001 0101 0000 1111 0000 0011 0110 = 4950F036
d shr 2 = 0001 0010 0101 0100 0011 1100 0000 1101 = =12543C0D
                        (a shr 3) and b:
   0001 0000 0110 0000 0001 1000 1001 0000 = 10601890
                       and
   0001 0010 1110 1100 0011 1000 1111 0100 = 12EC38F4
   ──────────────────────────────────────────────────
   0001 0000 0110 0000 0001 1000 1001 0000 = 10601890
                  (a shr 3) and b or (not c):
   0001 0000 0110 0000 0001 1000 1001 0000 = 10601890
                              or
   1010 1001 1100 0000 1101 0001 0011 0010 = A9C0D132
   ──────────────────────────────────────────────────
   1011 1001 1110 0000 1101 1001 1011 0010 = B9E0D9B2
      (a shr 3) and b or (not c) xor (d shr 2):
   1011 1001 1110 0000 1101 1001 1011 0010 = B9E0D9B2
                              xor
   0001 0010 0101 0100 0011 1100 0000 1101 = 12543C0D
   ──────────────────────────────────────────────────
   1010 1011 1011 0100 1110 0101 1011 1111 = ABB4F5BF
```

4. ДОПОЛНИТЕЛЬНЫЕ РЕЖИМЫ ЭМУЛЯТОРА

Основными режимами работы эмулятора являются:

- ввод исходных данных;
- полное выполнение программы;
- шаговое выполнение программы;
- просмотр содержимого памяти.

Ниже приводятся дополнительные режимы, реализуемые в курсовой работе по заданию преподавателя. Во всех режимах должен осуществляться контроль корректности ввода с клавиатуры (отсутствие недопустимых символов, задание значения адреса в пределах емкости памяти Memory и т.п.). При обнаружении ошибки ввода программа должна выдать на экран сообщение об ошибке и предложить пользователю повторить ввод данных.

4.1. Поиск слова в памяти.

По запросу программы пользователь вводит с клавиатуры содержимое искомого слова в соответствующей системе счисления (4, 8 или 16 с/с) и адрес ячейки памяти, начиная с которой должен осуществляться поиск. Если такое слово найдено в памяти Memory, программа должна вывести на экран его адрес, в противном случае - сообщение об отсутствии этого слова.

4.2. Ввод слова в память.

По запросу программы пользователь вводит с клавиатуры адрес и значение слова памяти Memory. Программа должна выдать на экран старое значение слова и сделать запрос пользователю о подтверждении замены слова. Замена должна производиться при положительном ответе пользователя.

4.3. Выполнение фрагментов теста.

Режим является дополнительным по отношению к режиму пошагового выполнения программы и позволяет после каждого нажатия клавиши Enter выполнять группу команд вместо одной команды. На каждом этапе работы в данном режиме пользователь указывает по запросу программы адрес команды, до которой должен быть выполнен фрагмент машинной программы. При появлении в счетчике адреса команды SAK заданного адреса программа эмулятора должна выполнить текущую команду, выдать на экран содержимое регистров ЭВМ и повторить запрос о значении нового адреса останова.

4.4. Дешифрация команды программы.

По адресу памяти Memory, задаваемому пользователем, выполняется чтение команды и ее дешифрация на экране: код операции, модификатор адреса (если он имеется), маска (для команд перехода), адреса операндов, значения операндов и т.п.

4.5. Дешифрация заданной команды.

Аналогично п.4.4, но вместо адреса существующей команды пользователь набирает с клавиатуры содержимое произвольной команды.

4.6. Копирование блока памяти.

Пользователь указывает начальный Adr1 и конечный Adr2 адреса блока памяти Memory, а также адрес Adr3, начиная с которого нужно скопировать заданный блок (в ячейки Adr3, Adr3+1, Adr3+2, …).

При выполнении этого режима программа эмулятора вначале сдвигает вниз содержимое k = Adr2-Adr1+1 ячеек памяти, начиная с адреса Adr3, а затем записывает в освободившееся место копию содержимого ячеек с адресами от Adr1 до Adr2. При этом содержимое последних k ячеек памяти Memory выходит за ее нижнюю границу и

исчезает.

4.7. Перемещение блока памяти.

Как и в режиме 4.6, пользователь задает значения адресов Adr1, Adr2 и Adr3. При этом фрагмент Adr1 .. Adr2 должен быть перенесен (не скопирован !) на участок, начиная с адреса Adr3. При этом предполагается, что Adr3 > Adr2.

В отличие от предыдущего режима, здесь количество информативных ячеек не изменяется.

Реализацию данного режима рекомендуется выполнять в следующем порядке.

а) Содержимое ячеек Adr1 .. Adr2 (их количество равно d1 = Adr2 –Adr1+1) скопировать в буферный массив.

б) Содержимое ячеек Adr2+1 .. Adr3-1 (их количество равно d2 = Adr3 – Adr2 – 1) сдвинуть вверх в область с начальным адресом Adr1.

в) Содержимое буферного массива переписать в область памяти, начиная с адреса Adr1+d2.

4.8. Определение времени эмуляции.

С помощью процедуры GetTime программа должна определить время эмуляции теста. Для этого программа организует многократное выполнение теста (10 или более раз) и определяет среднее время работы по его эмуляции. Количество повторений теста задается пользователем с клавиатуры.

4.9. Контроль исходного теста.

Программа производит ввод машинной программы из входного файла и выполняет контроль корректности текста программы (наличие недопустимых символов, длина строки, значение пускового адреса и т.п.). При обнаружении ошибки должно быть выдано сообщение о типе и местонахождении этой ошибки.

4.10. Калькулятор в 10 с/с.

Пользователь набирает с клавиатуры два операнда; после нажатия операционной клавиши ("+", "–", "*" или "/") выполняется соответствующая операция, ее результат выдается на экран. Операнды и результат – целые числа, не превышающие по модулю значения константы MaxLongInt. При вводе с клавиатуры должен быть предусмотрен контроль корректности вводимых данных. Если результат превышает значение MaxLongInt, на экран должно быть выведено соответствующее сообщение.

4.11. Калькулятор в 4 с/с.

Аналогично п.4.10.

4.12. Калькулятор в 8 с/с.

Аналогично п.4.10.

4.13. Калькулятор в 16 с/с.

Аналогично п.4.10.

4.14. Перевод 10 $\rightarrow$ 4.

Пользователь набирает с клавиатуры произвольное целое число в десятичной системе счисления, на экран выдается его эквивалент в четверичной системе счисления. Отрицательные числа в 4 с/с должны быть представлены в дополнительном коде. Исходные числа по модулю не должны превышать значения константы MaxLongInt. При вводе с клавиатуры должен быть предусмотрен контроль корректности вводимых данных.

4.15. Перевод 10 $\rightarrow$ 8.

Аналогично п.4.14.

4.16. Перевод 10 $\rightarrow$ 16.

Аналогично п.4.14.

4.17. Перевод 4 $a_1 = P \cdot a_i \cdot 2^{n-1}$ 10.

Пользователь набирает с клавиатуры произвольное целое число в четверичной системе счисления, на экран выдается его эквивалент в десятичной системе. Для

отрицательных чисел используется дополнительный код. Исходные числа по модулю не должны превышать значения, соответствующего константе MaxLongInt. При вводе с клавиатуры должен быть предусмотрен контроль корректности вводимых данных.

4.18. Перевод 8 $\rightarrow$ 10.

Аналогично п.4.17.

4.19. Перевод 16 $\rightarrow$ 10.

Аналогично п.4.17.

4.20. Контроль исходного файла. По запросу программы пользователь указывает внешнее имя исходного текстового файла. Введенное с клавиатуры значение подвергается в программе синтаксическому контролю (наличие недопустимых символов, длина имени и его расширения и др.). При обнаружении ошибки выдается соответствующее сообщение с требованием повторить ввод имени файла. При выполнении процедуры Reset проверяется наличие ошибок открытия файла (отсутствие файла на диске, неправильный путь к файлу, неготовность дисковода и др.), при обнаружении ошибки на экран выдается сообщение об ошибке. При нормальном открытии файла программа определяет степень его заполнения (количество строк в текстовом файле и его размер в байтах).

Указание. Для определения размера текстового файла использовать процедуру FindFirst, входящую в стандартный модуль Dos.

4.21. Формирование строки статуса. Строка статуса определяет назначение управляющих клавишей при работе программы. Для ее размещения обычно отводят последнюю, двадцать пятую, строку экрана. Информация, размещаемая в строке статуса, зависит от режима работы программы. Во всех режимах в левой части строки статуса должна быть надпись "F10 - выход", определяющая выход из программы. В пошаговом режиме и в режиме просмотра памяти дополнительно разместить надпись "Esc - рестарт", которая должна означать возврат в меню, позицией которого является данный режим. В пошаговом режиме должна быть также надпись "Enter - работа", определяющая переход к выполнению следующей команды машинной программы. В режиме просмотра памяти в правой части строки статуса разместить надпись "Down,PgDn - просмотр", если в окне просмотра находится начальная часть массива Memory, "Up,PgUp - просмотр", если в этом окне находится конечная часть массива Memory, "Up, PgUp/Down,PgDn - просмотр" - для промежуточной части данного массива. Реакция на нажатие управляющих клавишей осуществляется в программе при приостановке ее работы (выполнение очередной команды в пошаговом режиме, "перелистывание" содержимого памяти в режиме просмотра и т.п.).

4.22. Формирование меню исходных файлов. При старте эмулятора должен быть автоматически выполнен просмотр текущего каталога и сформирован список имен файлов, имеющих расширение dat. Выдача меню исходных файлов на экран должна производиться в режиме "Ввод исходного файла". При этом необходимо обеспечить передвижение строки-курсора по меню и чтение файла, соответствующего выбранной позиции меню.

4.23. Формирование справочной информации. В данном случае речь идет о реализации Help-режима, определяющего выдачу справочной информации при нажатии клавиши F1. Такая информация должна выдаваться в следующих случаях:

- информация общего характера (назначение программы, перечень и краткая характеристика режимов работы и т.п.) - при выдаче на экран заставки программы;
- подробная информация о том режиме или подрежиме работы, позицию которого в меню указывает строка-курсор.

Для выдачи справочной информации целесообразно создавать окно просмотра, занимающее часть экрана на фоне других визуальных изображений (например, окно 12 x 46). Для размещения справочной информации используется типизированный файл с

компонентами типа **string**[n], где *n* определяется длиной строки окна просмотра. В программе в зависимости от положения строки-курсора из типизированного файла выбирается та часть, которая относится к данному режиму (в программе указываются номера начального и конечного компонентов файла для каждого режима). Исходная справочная информация размещается в текстовом файле, строки которого по длине не превышают значения *n*, после чего отдельная сервисная программа читает текстовый файл и формирует соответствующий типизированный файл.

4.24. Создание и просмотр файла трассировки. Рассматривается как подрежим, используемый в режиме полного выполнения программы. При его включении после выполнения каждой команды машинной программы в файл заносится информация о состоянии регистров машины. Просмотр файла организуется в эмуляторе аналогично режиму просмотра содержимого памяти Memory. Файл трассировки целесообразно создавать в виде типизированного файла, компонентами которого являются записи.

4.25. Статистика обращения к памяти. Рассматривается как подрежим, используемый в режимах Run и Step. После окончания решения задачи на экран должна быть выведена следующая информация:

- общее количество операций чтения из памяти Memory;
- адреса ячеек чтения с указанием количества обращений к этим ячейкам; адреса должны быть сгруппированы в порядке убывания количеств обращений к соответствующим ячейкам;
- аналогично для операций записи в память.

4.26. Статистика выполнения команд машинной программы. Рассматривается как подрежим, используемый в режимах Run и Step. После окончания решения задачи на экран должна быть выведена информация о количествах выполнения каждой машинной команды учебной ЭВМ с группировкой по убыванию этих количеств.

4.27. Используя процедуры GetDate и GetTime, вывести на экран информацию о текущей дате и текущем времени в приведенном ниже виде.

Первая строка: “15 января 2005 года”.

Вторая строка: “10 часов 15 минут 32 секунды”.

При этом в зависимости от значений числительных нужно формировать соответствующий падеж существительного. Например, 10 минут, но 21 минута, 22 минуты.

4.28. Калькулятор длинных чисел в 4 с/с.

В отличие от п.4.11, исходные числа и результат могут значительно превосходить значение константы MaxLongInt (например, иметь 40 разрядов, в то время как MaxLongInt в 4 с/с имеет 16 разрядов). По согласованию с преподавателем здесь можно ограничиться операциями сложения и вычитания).

4.29. Калькулятор длинных чисел в 8 с/с.

В отличие от п.4.12, исходные числа и результат могут значительно превосходить значение константы MaxLongInt (например, иметь 40 разрядов, в то время как MaxLongInt в 8 с/с имеет 11 разрядов). По согласованию с преподавателем здесь можно ограничиться операциями сложения и вычитания).

4.30. Калькулятор длинных чисел в 10 с/с.

В отличие от п.4.13, исходные числа и результат могут значительно превосходить значение константы MaxLongInt (например, иметь 40 разрядов, в то время как MaxLongInt в 10 с/с имеет 10 разрядов). По согласованию с преподавателем здесь можно ограничиться операциями сложения и вычитания).

31. Калькулятор тригонометрических функций sin(x) и cos(x). При этом аргумент должен быть задан в градусах и минутах в диапазоне от -1000° до 1000°.

Указания.

1) Заданное значение аргумента привести к диапазону $0 \,..\, 2\pi$ радиан.
2) Вычисление функций производить по их разложению в ряд Маклорена.

3) Значения функций печатать в виде целой и дробной частей, разделенных точкой. Для дробной части отвести 6 позиций.

ТЕКСТ ЭМУЛЯТОРА ЭВМ М1

```
UNIT DesUnit;
{ Глобальные описания констант,типов и переменных }

Interface

Uses Crt;

Const
   MaxAddress = 255;  { максимальный адрес оперативной памяти }
   MaxNumber = 32767.0; { макс.значение 16-разрядного двоично-}
                        { го числа для индикации переполнения }
   Enter = 13;          { код клавиши Enter }
   PressKey = 'Нажмите клавишу "В В О Д" ';

   BlackLightGray = 16*Black+LightGray;  { константы цветовых }
   BlackLightGreen = 16*Black+LightGreen;         { атрибутов }
   BlueLightGray = 16*Blue+LightGray;
   BlueLightRed = 16*Blue+LightRed;
   BlueYellow = 16*Blue+Yellow;
   BlueWhite = 16*Blue+White;
   GreenBlack = 16*Green+Black+128;
   CyanBlack = 16*Cyan+Black;
   RedWhite = 16*Red+White;
   MagentaLightGreen = 16*Magenta+LightGreen;
   MagentaLightCyan = 16*Magenta+LightCyan;
   MagentaYellow = 16*Magenta+Yellow;
   MagentaWhite = 16*Magenta+White;
   BrownBlack = 16*Brown+Black;
   BrownDarkGray = 16*Brown+DarkGray;
   BrownLightCyan = 16*Brown+LightCyan;
   BrownWhite = 16*Brown+White;
   LightGrayBlack = 16*LightGray+Black
Type
   set2  = set of 1..2;      { описания типов }
   set4  = set of 1..4;      { для памяти и регистров }
   set8  = set of 1..8;
   set16 = set of 1..16;
   RKType = record           { тип регистра команд }
     KOP : set4;             { код операции }
     MA,                     { модификатор адреса
     NR : set2;              { номер регистра }
     AM : set8;              { адрес операнда }
   end;
   RegType = array[0..3] of set16;  { тип блока регистров }
   MemoryType = array[0..MaxAddress] of set8;{ тип памяти }

   String2 = string[2];
```

```
   String4 = string[4];
   String8 = string[8];
   String12 = string[12];
   String16 = string[16];
   String80 = string[80];

   ScreenColRange = 1..80; { тип номера столбца экрана }
   ScreenRowRange = 1..25; { тип номера строки экрана }
   ScreenChar = record     { тип элемента видеопамяти }
     Symbol : char;        { символ }
     Attrib : byte;        { атрибут }
   end;
   ScreenRowArray = array[ScreenColrange] of ScreenChar;
                   { тип строки экрана }
   ScreenArray = array[ScreenRowRange] of ScreenRowArray;
                   { тип видеопамяти }
   ScreenPointer = ^ScreenArray;

Const                       { информационные сообщения }
   Mes0  : string[78] = ' Esc \ выход '+
           ' \Enter \ выполнить';
   Mes1  : string[78] = 'Esc \ выход \'+Chr(24)+Chr(25)+
           ' \ указатель'+' \Enter \ выполнить';
   MesReg : string[78] = ' Работа режима закончена. '+
           'Нажмите клавишу Enter ';
   MesStep : string[78] = 'Esc \ выход '+
             ' \Enter \ выполнение команды';
Var
   Aisp,                  { исполнительный адрес }
   SAK,                   { счетчик адреса команд }
   SAK1,                  { буферная переменная }
   RA,                    { регистр адреса }
   NumReg                 { номер регистра в блоке R }
      : byte;
   RP : set4;             { регистр признаков }
   Slovo,                 { буферная переменная }
   RS : set8;             { регистр слова }
   RK : RKType;           { регистр команд }
   OR1, OR2,              { операционные регистры }
   Res : set16;           { регистр результата }
   Number1, Number2       { численные значения операндов }
      : integer;
   CondExec : boolean;    { признак выполнения маш.программы }
   Reg : RegType;         { блок регистров }
   Memory : MemoryType;   { оперативная память }

   CurrRow,CurrCol,       { текущие строка и колонка экрана }
   Key: byte;             { ординальный номер символа }
   RowAr : ScreenRowArray;   { строка видеопамяти }
   Screen : ScreenArray
            absolute $B800:$0000; { видеопамять }
   BufScreen1,BufScreen2, { указатели буферных массивов }
   BufScreen3             { для сохранения видеопамяти }
```

```
      : ScreenPointer;

   FileModul                    { файл исходных данных }
      : text;

Implementation

End.
```

```
{$R+}
UNIT BasUnit;
{ Базовые процедуры и функции }

Interface

Uses Crt,Dos,DesUnit,Printer;

Procedure WaitEnter;
Function  Space(S:String80; k:byte):byte;
Function  NotSpace(S:String80; k:byte):byte;
Function  GetKey : byte;
Procedure SetCursor(Cursor:word);
Procedure MoveFromScreen(Var BufScreen:ScreenPointer;
                         Row1,Row2:byte);
Procedure MoveToScreen(Var BufScreen:ScreenPointer;
                       Row1,Row2:byte);
Procedure InsertColorString(Row,Col,Len,Color : byte);
Procedure ClrEolXY(Col:ScreenColRange; Row:ScreenRowRange;
                   Color:byte);
Procedure ClearScreen(x1,y1,x2,y2,Attrib:word);
Procedure WriteColor(S:string; Color:byte);
Procedure WriteXY(S:string; Col:ScreenColRange;
                  Row:ScreenRowRange; Color:byte);
Function  FillString(Len:byte; ch:char) : string;
Function  CenterString(S:string; Len:byte):string;
Procedure PrintError(Error:string);
Function  GetNumber(Prompt:string; Low,High:real; m,n:byte;
                   Var Result:boolean) : real;
Procedure Frame(x1,y1,x2,y2,Line,Color : byte);
Procedure MoveCursor(UpRow,DownRow,StepRow,LeftCol,RightCol,
                     StepCol,Len,NewColor,OldColor:byte);
Procedure WriteLastString(S:string; BlueYellow,
                          LightGrayBlack:byte);
Procedure FromSetToNumber(Operand:set16;
                          Var Number:integer);
Procedure FromNumberToSet(Number:integer;
                          Var Operand:set16);
Procedure StartPosMemory;
Procedure StartPosRegisters;
Procedure WriteMemory(Adr1,Adr2:byte);
```

```
Procedure WriteRegisters;

Implementation

{ ------------------------------------------------------ }

Procedure WaitEnter;
{ Задержка выполнения программы до тех пор, }
{ пока не будет нажата клавиша Enter )
Var ch : char;
Begin
  Repeat
    ch:=ReadKey;
  Until ord(ch) = Enter;
End { WaitEnter };
{ ------------------------------------------------------ }

Function  Space(S:String80; k:byte):byte;
{ Поиск ближайшего пробела, начиная с позиции k }
{ строки S; Space = 0, если пробел не обнаружен }
Var  i : byte;
Begin
  Space:=0;
  For i:=k to length(S) do
    If S[i]=' ' then
      Begin
        Space:=i; Exit
      End;
End { Space };
{ ------------------------------------------------------ }

Function  NotSpace(S:String80; k:byte):byte;
{ Поиск ближайшего непробела, начиная с позиции k }
{ строки S; NotSpace=0, если непробел не обнаружен }
Var  i : byte;
Begin
  NotSpace:=0;
  For i:=k to length(S) do
    If S[i]<>' ' then
      Begin
        NotSpace:=i; Exit
      End;
End { NotSpace };
{ ------------------------------------------------------ }

Function GetKey : byte;
[ Чтение символа клавиатуры }
Var  ch : char;
Begin
  ch := ReadKey;
  If ord(ch) = 0 then       { Расширенный код символа }
    GetKey := ord(ReadKey)
  Else
```

```
    GetKey := ord(ch);          { Нормальный код символа }
End { GetKey };
{ ------------------------------------------------------ }

Procedure SetCursor(Cursor:word);
{ Установка формы курсора (Cursor = $2000 - отключение }
{ курсора, Cursor = $0607 - нормальный курсор) }
Var  Reg : Registers;
Begin
  With Reg do
    Begin
      AH:=1; BH:=0;
      CH:=Hi(Cursor); CL:=Lo(Cursor);
    End;
  Intr($10,Reg);
End { SetCursor };
{ ------------------------------------------------------ }

Procedure MoveFromScreen(Var BufScreen:ScreenPointer;
                         Row1,Row2:byte);
{ Сохранение области экрана, ограниченной строками Row1,Row2 }
Var  Len : word;
Begin
  New(BufScreen);
  Len:=(Row2-Row1+1)*80 shl 1;
  Move(Screen[Row1],BufScreen^,Len);
End { MoveFromScreen };
{ ------------------------------------------------------ }

Procedure MoveToScreen(Var BufScreen:ScreenPointer;
                       Row1,Row2:byte);
{ Восстановление области экрана, ограниченной }
{ строками Row1,Row2 }
Var  Len : word;
Begin
  Len:=(Row2-Row1+1)*80 shl 1;
  Move(BufScreen^,Screen[Row1],Len);
  Dispose(BufScreen);
End { MoveFromScreen };
{ ------------------------------------------------------ }

Procedure InsertColorString(Row,Col,Len,Color : byte);
{ Выделение строки Row длиной Len цветом Color с позиции Col }
{ (изменение атрибутов знакомест строки экрана) }
Var  j : byte;
Begin
  For j:=Col to Col+Len do
    Screen[Row,j].Attrib := Color;
End { InstColorString };
{ ------------------------------------------------------ }

Procedure ClrEolXY(Col:ScreenColRange; Row:ScreenRowRange;
                   Color:byte);
```

```
{ Очистка до конца текущей строки экрана в заданном }
{ цвете (начиная с позиции  Col, Row }
Begin
  GotoXY(Col, Row);
  TextAttr := Color; ClrEOL;
End { ClrEolXY };
{ ------------------------------------------------------- }

Procedure ClearScreen(x1,y1,x2,y2,Attrib:word);
{ Очистка области экрана, ограниченной координатами }
{ x1,y1 ( левый верхний угол) и x2,y2 (правый нижний }
{ угол) фоновым цветом Attrib }
Var  Reg : Registers;
Begin
  If (x1>x2) or (y1>y2) then  { Неправильные значения }
    Exit;
  With Reg do
    Begin
      AX := $0600;         { Очистка экрана с помощью BIOS }
      BH := Attrib;
      CH := Pred(y1); CL := Pred(x1);
      DH := Pred(y2); DL := Pred(x2);
      Intr($10, Reg);
    End;
End { ClearScreen };
{ ------------------------------------------------------- }

Procedure WriteColor(S : string; Color : byte);
{ Вывод строки на экран в заданном цвете }
Begin
  TextAttr:=Color; Write(S);
End { WriteColor ;}
{ ------------------------------------------------------- }

Procedure WriteXY(S:string; Col:ScreenColRange;
                  Row:ScreenRowRange; Color:byte);
{ Вывод строки S на экран с позиции Col,Row в цвете Color }
Begin
  GotoXY(Col,Row);
  TextAttr:=Color; Write(S);
End { WriteXY };
{ ------------------------------------------------------- }

Function FillString(Len:byte; ch:char) : string;
{ Заполнение Len байт строки символом ch }
Var  S : string;
Begin
  S[0]:=chr(Len);
  FillChar(S[1],Len,ch); FillString := S;
End { FillString };
{ ------------------------------------------------------- }

Function CenterString(S:string; Len:byte):string;
```

```
{ Установка строки S по центру поля длиной Len байт }
Var l : byte;
   S1 : string;
Begin
  l:=length(S) div 2; S1:=FillString(Len,' ');
  Insert(S,S1,(Len div 2)-l+1);
  CenterString:=S1;
End { CenterString };
{ ------------------------------------------------------ }

Procedure PrintError(Error : string);
{ Вывод на экран в строку 25 сообщения об ошибке }
Var  S : string;
Begin
  MoveFromScreen(BufScreen3,25,25);
  ClrEOLXY(1,25,RedWhite);
  S:='  '+Error + '. Нажмите Escape'; Write(S);
  Sound(220); Delay(300); NoSound;
  Repeat
    Key := GetKey;
  Until Key = 27;
  MoveToScreen(BufScreen3,25,25);
End { PrintError };
{ ------------------------------------------------------ }

Function GetNumber(Prompt:string; Low,High:real; m,n:byte;
                   Var Result:boolean) : real;
{ Ввод с клавиатуры вещественного числа.  Строка Prompt – за- }
{ прос ввода; Low,Hight - допустимый диапазон вводимого числа;}
{ m,n -  формат  печати числа  в сообщении  об  ошибке ввода; }
{ Result - признак результата. При ошибке ввода (неправильный }
{ формат или неправильное значение) в строке 25 печатается со-}
{ общение "Число должно быть в пределах от Low до Hight". Со- }
{ общение сопровождается звук. сигналом. Запрос ввода повто-  }
{ ряется до тех пор, пока не будет введено правильное значение}
Var  S,S1,S2 : string[12];
     Error : integer;
     L : real;
Begin
  TextAttr:=MagentaWhite;
  Repeat
    WriteXY(Prompt + ': ',1,24,MagentaWhite);
    ClrEol; SetCursor(7);
    Readln(S); SetCursor($2000);
    Val(S,L,Error);
    Result := (Error = 0) and (L >= Low) and (L <= High);
    Str(Low:n:n, S1); Str(High:n:n, S2);
    If not Result then
      PrintError('Число должно быть в пределах от ' + S1 +
                 ' до ' + S2)
  Until Result;
  ClrEolXY(1,24,MagentaWhite); GetNumber := L;
```

```
End { GetNumber };
{ ------------------------------------------------------ }

Procedure Frame(x1,y1,x2,y2,Line,Color:byte);
{ Вычерчивание прямоугольной рамки с заданными координатами }
{ одинарной (Line=1), двойной (Line=2) или широкой (Line=3) }
{ линией; цвет и фон - Color; при Line=0 рамка не вычерчи-  }
{ вается, но прямоугольная область  закрашивается  фоновым  }
{ цветом Color }
Var  ChGr,          { символ горизонтальной линии рамки }
   ChVr,            { символ вертикальной линии рамки }
   LfUp,            { левый верхний угол рамки }
   LfDw,            { левый нижний угол }
   RtUp,            { правый верхний угол }
   RtDw : char;     { правый нижний угол }
   i : byte;
Begin
  Case Line of
    0 : Begin
          ChGr:=#32; ChVr:=#32; LfUp:=#32;
          LfDw:=#32; RtUp:=#32; RtDw:=#32
        End;
    1 : Begin
          ChGr:=#196; ChVr:=#179; LfUp:=#218;
          LfDw:=#192; RtUp:=#191; RtDw:=#217
        End;
    2 : Begin
          ChGr:=#205; ChVr:=#186; LfUp:=#201;
          LfDw:=#200; RtUp:=#187; RtDw:=#188
        End;
    3 : Begin
          ChGr:=#176; ChVr:=#176; LfUp:=#176;
          LfDw:=#176; RtUp:=#176; RtDw:=#176
        End;
  End;
  WriteXY(LfUp,x1,y1,Color); Write(FillString(x2-x1,ChGr));
  WriteXY(RtUp,x2,y1,Color);
  For i:=Succ(y1) to Pred(y2) do
    Begin
      WriteXY(ChVr,x1,i,Color); WriteXY(ChVr,x2,i,Color)
    End;
  WriteXY(LfDw,x1,y2,Color); Write(FillString(x2-x1,ChGr));
  WriteXY(RtDw,x2,y2,Color);
  ClearScreen(x1+1,y1+1,x2-1,y2-1,Color);
End { Frame };
{ ------------------------------------------------------ }

Procedure MoveCursor(UpRow,DownRow,StepRow,LeftCol,RightCol,
                     StepCol,Len,NewColor,OldColor:byte);
{ Перемещение курсора-строки по экрану: }
{   UpRow,DownRow - верхняя и нижняя границы перемещения }
{                   курсора; }
```

```
{   StepRow - шаг перемещения курсора по вертикали; }
{   LeftCol, RightCol - левая и нижняя границы перемещения; }
{   StepCol - шаг перемещения по горизонтали; }
{   Len - длина строки-курсора; }
{   NewColor, OldColor - новые и старые цветовые атрибуты }
{                        строки-курсора и остального поля }
Var  Cond : boolean;
Begin
  Repeat
    Key:=GetKey;  Cond:=false;
    Case Key of
     13,27 : Cond := true; Enter,Escape
        72 : If CurrRow >= UpRow+StepRow then    { Up }
               Begin
                 InsertColorString(CurrRow,CurrCol,Len,
                                   OldColor);
                 CurrRow := CurrRow-StepRow;
               End;
        80 : If CurrRow <= DownRow-StepRow then  { Down }
               Begin
                 InsertColorString(CurrRow,CurrCol,Len,
                                   OldColor);
                 CurrRow := CurrRow+StepRow;
               End;
        77 : If CurrCol+StepCol <= RightCol then { Right }
               Begin
                 InsertColorString(CurrRow,CurrCol,Len,
                                   OldColor);
                 CurrCol := CurrCol+StepCol;
               End;
        75 : If CurrCol-StepCol >= LeftCol  then  { Left }
               Begin
                 InsertColorString(CurrRow,CurrCol,Len,
                                   OldColor);
                 CurrCol := CurrCol-StepCol;
               End;
    end;
    InsertColorString(CurrRow,CurrCol,Len,OldColor);
  Until Cond;
End { MoveCursor };
{ ------------------------------------------------------- }

Procedure WriteLastString(S:string; BlueYellow,
                          LightGrayBlack:byte);
{ Печать сообщения в последней строке экрана. Текст сообщения }
{ разделен на фрагменты символами "\". Нечетные фрагменты вы- }
{ деляются цветом BlueYellow, четные - цветом LightGrayBlack  }
Var  k : byte;
    St : string80;
Begin
  ClrEolXY(1,25,LightGrayBlack);
  S:=S+'\'; GotoXY(1,25);
```

```
  While S<>'' do
    Begin
      WriteColor('.',LightGrayBlack);
      k:=Pos('\',S);
      WriteColor(Copy(S,1,pred(k),BlueYellow);
      Delete(S,1,k);
      k:=Pos('\',S);
      If S[1]='\' then
        St:='-'
      Else
        St:='-'+Copy(S,1,pred(k));
      WriteColor(St,LightGrayBlack);
      Delete(S,1,k);
    End;
End  WriteLastString ;
{ ------------------------------------------------------ }

Procedure FromSetToNumber(Operand:set16; Var Number:integer);
{ Преобразование множества в число }
Var   i : byte;
   Cond : boolean;
Begin
  Cond:=false;
  If 1 in Operand then
    Begin
      Cond:=true;
      For i:=1 to 16 do          { инверсирование цифр }
        If i in Operand then     { отрицательного двоичного }
          Operand:=Operand-[i]   { числа }
        Else
          Operand:=Operand+[i];
    End;
  Number:=0;
  For i:=2 to 16 do              { получение численного }
    Begin                        { значения по схеме }
      Number:=2*Number;          { Горнера }
      If i in Operand then
        Number:=Number+1;
    End;
  If Cond then                   { добавление единицы, если }
    Begin                        { был преобразован дополни-}
      Inc(Number);               { тельный код, и изменение }
      Number:=-Number            { знака числа }
    End;
End { FromSetToNumber };
{ ------------------------------------------------------ }

Procedure FromNumberToSet(Number:integer; Var Operand:set16);
{ Преобразование числа в множество }
Var  i : byte;
    k,Divisor : integer;
    Cond : boolean;
```

```
Begin
  Cond:=false;
  If Number<0 then
    Begin
      Cond:=true;               { получение положительного }
      Number:=-Number;          { числа и вычитание из него }
      Dec(Number);              { единицы }
    End;
  Divisor:=16384; Operand:=[]; { делитель равен 2**14 }
  For i:=2 to 16 do
    Begin                       { последовательное формиро- }
      k:=Number div Divisor;    { вание двоичных цифр числа
      If k>0 then
        Operand:=Operand + [i];
      Number:=Number mod Divisor;
      Divisor:=Divisor shr 1;
    End;
  If Cond then
    For i:=1 to 16 do           { инверсирование цифр отри- }
      If i in Operand then      { цательного числа (получе- }
        Operand:=Operand - [i] { ние дополнительного кода) }
      Else
        Operand:=Operand + [i];
End { FromNumberToSet };
{ ------------------------------------------------------ }

Procedure StartPosMemory;
{ Установка памяти в начальное состояние }
Var  i : byte;
Begin
  For i:=0 to MaxAddress do
   Memory[i]:=[];
End { StartPosMemory };
{ ------------------------------------------------------ }

Procedure StartPosRegisters;
{ Установка регистров в начальное состояние }
Var  i : byte;
Begin
  SAK:=0; RA:=0; Aisp:=0;
  RK.KOP:=[]; RK.MA:=[]; RK.NR:=[]; RK.AM:=[];
  RS :=[]; Res:=[]; OP1:=[]; OP2:=[]; RP:=[];
  For i:=0 to 3 do
   Reg[i]:=[];
End { StartPosRegisters };
{ ------------------------------------------------------ }

Procedure WriteMemory(Adr1,Adr2:byte);
{ Вывод на экран области памяти Memory, ограниченной }
{ адресами Adr1 и Adr2 (процедура используется при   }
{ отладке эмулятора) }
Var  i,j,k : byte;
      S1 : string80;
```

```
Begin
  ClrScr; i:=Adr1;
  While i<= Adr2 do
    Begin
      S1:=''; Slovo:=Memory[i];
      For j:=1 to 8 do
        If j in Slovo then
          S1:=S1+'1'
        Else
          S1:=S1+'0';
      S1:=S1+' '; Slovo:=Memory[i+1];
      For j:=1 to 8 do
        If j in Slovo then
          S1:=S1+'1'
        Else
          S1:=S1+'0';
      Inc(i,2); Writeln(S1);
    End;
End { WriteMemory };
{ ------------------------------------------------------ }

Procedure WriteRegisters;
{ Вывод содержимого регистров (процедура используется }
{ при отладке эмулятора) }
Var  i,j : byte;
Begin
  Write('SAK=',SAK); Write('   NumReg=',NumReg);
  Writeln('    Aisp=',Aisp);
  Write('RK.KOP=');
  For i:=1 to 4 do
    If i in RK.KOP then
      Write('1')
    Else
      Write('0');
  Write('   '); Write('RK.MA=');
  For i:=1 to 2 do
    If i in RK.MA then
      Write('1')
    Else
      Write('0');
  Write('   '); Write('RK.NR=');
  For i:=1 to 2 do
    If i in RK.NR then
      Write('1')
    Else
      Write('0');
  Write('   '); Write('RK.AM=');
  For i:=1 to 8 do
    If i in RK.AM then
      Write('1')
    Else
      Write('0');
```

```
  Writeln; Write('RS= ');
  For i:=1 to 8 do
    If i in RS then
      Write('1')
    Else
      Write('0');
  Writeln;
  For i:=0 to 3 do
    Begin
      Write('Reg[',i,']= ');
      For j:=1 to 16 do
        If j in Reg[i] then
          Write('1')
        Else
          Write('0');
      Writeln;
    End;
  Write('OP1= ');
  For i:=1 to 16 do
    If i in OP1 then
      Write('1')
    Else
      Write('0');
  FromSetToNumber(OP1,Number1);
  Writeln('    ',Number1); Write('OP2= ');
  For i:=1 to 16 do
    If i in OP2 then
      Write('1')
    Else
      Write('0');
  FromSetToNumber(OP2,Number1);
  Writeln('    ',Number1); Write('Res= ');
  For i:=1 to 16 do
    If i in Res then
      Write('1')
    Else
      Write('0');
  FromSetToNumber(Res,Number1);
  Writeln('    ',Number1); Write('RP= ');
  For i:=1 to 4 do
    If i in RP then
      Write('1')
    Else
      Write('0');
End { WriteRegisters };
{ ------------------------------------------------------- }

End.
```

```
$R+
UNIT InsUnit;
{ Процедуры выполнения машинных команд }

Interface

Uses Crt,Dos,DesUnit,BasUnit,Printer;

Procedure Load;
Procedure Store;
Procedure Add;
Procedure Subtract;
Procedure Multiple;
Procedure Divide;
Procedure Conjunct;
Procedure Disjunct;
Procedure Exclus;
Procedure ShiftLeft;
Procedure ShiftRight;
Procedure BranchCond;
Procedure Compare;
Procedure ToSubroutine;
Procedure FromSubroutine;

Implementation

 ------------------------------------------------------------

Procedure ExecAddress;
{ Формирование исполнительного адреса }
Var    i : byte;
   BufSet : set16;
Begin
  BufSet:=[];           { пересылка АМ в 16-разрядное }
  For i:=1 to 8 do      { множество BufSet и преобра- }
    If i in RK.AM then  { зование АМ в число Number1  }
      BufSet:=BufSet+[i+8];
  FromSetToNumber(BufSet,Number1);
  If RK.MA=[] then      { прямая адресация }
    Aisp:=Number1
  Else
    If RK.MA=[2] then   { косвенная адресация }
      Begin
        RS:=Memory[Number1+1];
        BufSet:=[];
        For i:=1 to 8 do
          If i in RS then
            BufSet:=BufSet+[i+8];
        FromSetToNumber(BufSet,Number1);
        Aisp:=Number1;
      End
    Else
      If RK.MA=[1,2] then  { индекс-адресация }
```

```
        Begin
          FromSetToNumber(Reg[0],Number2);
          Aisp:=Number1+Number2
        End;
End { ExecAddress };
 ------------------------------------------------------------

Procedure MakeOR2;
{ Формирование содержимого регистра OP2 }
Var  i : byte;
Begin
  If RK.MA=[1] then              { непосредственная адресация: }
    Begin                        { пересылка АМ в OR2 }
      OR2:=[];
      For i:=1 to 8 do
        If i in RK.AM then
          OR2:=OR2 + [i+8]
    End
  Else                           { другие типы адресации: }
    Begin                        { пересылка двух смежных }
      ExecAddress; RA:=Aisp;     { ячеек памяти в OR2 }
      RS:=Memory[RA];
      OR2:=[]; OR2:=OR2 + RS;
      RA:=RA+1; RS:=Memory[RA];
      For i:=1 to 8 do
        If i in RS then
          OR2:=OR2 + [i+8];
    End;
End { MakeOR2 };
 ------------------------------------------------------------

Procedure Load;
{ Реализация команды загрузки  0001 }
Var  i : byte;
Begin
  MakeOr2;
  Res:=OR2; Reg[NumReg]:=Res;
  RP:=[];
  If Or2=[] then
    RP:=RP+[1]
  Else
    If 1 in OR2 then
      RP:=RP+[2]
    Else
      RP:=RP+[3];
End { Load };
 ------------------------------------------------------------

Procedure Store;
{ Реализация команды записи в память  0010 }
Var  i : byte;
Begin
  ExecAddress; RA:=Aisp;
```

```
  RS:=Reg[NumReg]; Memory[RA]:=RS;
  RA:=RA+1; RS:=[];
  For i:=9 to 16 do
    If i in Reg[NumReg] then
      RS:=RS+[i-8];
  Memory[RA]:=RS;
End { Store };
 ------------------------------------------------------------
Procedure Add;
{ Реализация команды сложения  0011 }
Var  x,y,z : real;
     S1,S2 : string80;
Begin
  MakeOR2; OR1:=Reg[NumReg];
  FromSetToNumber(OR1,Number1);
  FromSetToNumber(OR2,Number2);
  x:=Number1; y:=Number2; z:=x+y;
  Number1:=Number1+Number2;
  If (z<-MaxNumber-1) or (z>MaxNumber) then
    Begin
      RP:=[4];
      Number1:=SAK-2; Str(Number1,S1);
      S2:=' Переполнение с фикс.зпт Адрес = '+S1;
      PrintError(S2);
    End
  Else
    If Number1=0 then
      RP:=[1]
    Else
      If Number1<0 then
        RP:=[2]
      Else
        RP:=[3];
  FromNumberToSet(Number1,Res);
  Reg[NumReg]:=Res;
End { Add };
 ------------------------------------------------------------

Procedure Subtract;
{ Реализация команды вычитания  0100 }
Begin
End { Subtract };
 ------------------------------------------------------------

Procedure Multiple;
{ Реализация команды умножения  0101 }
Begin
End { Multiple };
 ------------------------------------------------------------

Procedure Divide;
{ Реализация команды деления  0110 }
Begin
```

```
End { Divide };
 ------------------------------------------------------------

Procedure Conjunct;
{ Реализация команды конъюнкции  0111 }
Begin
  MakeOPR;
  OR1:=Reg[NumReg]; Res:=OR1*OR2;
  If Res=[] then
    RP:=[1]
  Else
    RP:=[2];
  Reg[NumReg]:=Res;
End { Conjunct };
 ------------------------------------------------------------

Procedure Disjunct;
{ Реализация команды дизъюнкции  1000 }
Begin
End { Disjunct };
 ------------------------------------------------------------

Procedure Exclus;
{ Реализация ком-ды исключ. ИЛИ (сложения по модулю 2) 1001 }
Begin
End { Exclus };
 ------------------------------------------------------------

Procedure ShiftLeft;
{ Реализация команды сдвига влево  1010 }
Var  i : byte;
   BufSet : set16;
Begin
  If RK.MA=[1] then
    Begin
      OR1:=[];
      For i:=1 to 8 do
        If i in RK.AM then
          OR1:=OR1+[i+8];
      FromSetToNumber(OR1,Number1);
      Aisp:=Number1;
    End
  Else
    ExecAddress;
  OR1:=Reg[NumReg]; BufSet:=[];
  For i:=Aisp+1 to 16 do
    If i in OR1 then
      BufSet:=BufSet+[i-Aisp];
  OR1:=BufSet; Res:=OR1;
  If Res=[] then
    RP:=[1]
  Else
    RP:=[2];
```

```
  Reg[NumReg]:=Res;
End { ShiftLeft };
 ------------------------------------------------------------

Procedure ShiftRight;
{ Реализация команды сдвига вправо  1011 }
Begin
End { ShiftRight };
 ------------------------------------------------------------

Procedure BranchCond;
{ Реализация команды перехода  1100 }
Var    i : byte;
   Maska : set4;
Begin
  Maska:=[]; Maska:=Maska+RK.Ma;
  For i:=1 to 2 do
    If i in RK.NR then
      Maska:=Maska+[i+2];
  RK.MA:=[2]; ExecAddress;
  If Maska=[1..4] then
    SAK:=Aisp
  Else
    Begin
      Maska:=Maska*RP;
      If Maska<>[] then
        SAK:=Aisp;
    End;
End { BranchCond };
 ------------------------------------------------------------

Procedure Compare;
{ Реализация команды сравнения  1101 }
Begin
End { Compare };
 ------------------------------------------------------------

Procedure ToSubroutine;
{ Реализация команды перехода к подпрограмме  1110 }
Begin
  Number1:=SAK;
  FromNumberToSet(Number1,Res);
  Reg[NumReg]:=Res;
  ExecAddress; SAK:=Aisp;
End { ToSubroutine };
 ------------------------------------------------------------

Procedure FromSubroutine;
{ Реализация команды выхода из подпрограммы  1111 }
Begin
  OP1:=Reg[NumReg];
  FromSetToNumber(OP1,Number1);
  SAK:=Number1;
```

```
End { FromSubroutine };
 ------------------------------------------------------------

End.


{$R+}
UNIT RegUnit;
{ Процедуры реализации режимов работы эмулятора }

Interface

Uses Crt,Dos,DesUnit,BasUnit,InsUnit,Printer;

Procedure InputModul;
Procedure Run;
Procedure Step;
Procedure WriteStore;
Procedure From10To16;

Implementation
 ------------------------------------------------------------

Procedure FileName(Var Sn:string12);
{ Формирование имени файла: просмотр текущего каталога; выбор }
{ имен всех файлов  с расширением dat;  формирование  меню  с }
{ именами файлов; выбор позиции в меню;  запись  в строку вы- }
{ бранного имени файла }
Begin
End { FileName };
 ------------------------------------------------------------

Procedure InputModul;
{ Ввод исходного модуля машинной программы }
Const   Dig = '0123456789ABCDEF';
     Bin : array[1..16] of string4 =
     ('0000','0001','0010','0011','0100','0101','0110','0111',
      '1000','1001','1010','1011','1100','1101','1110','1111');
Var i,k,k1,k2 : byte;
    ch : char;
    Sn : string12;
    DirName,S1,S2 : string80;
Begin
  FileName(Sn);
  If Key=27 then
    Begin
      Key:=0; Exit
    End;
  GetDir(0,DirName); S1:=DirName+'\'+Sn;
  Assign(FileModul,S1); Reset(FileModul);
  Readln(FileModul,SAK); k:=SAK;
  While not eof(FileModul) do
    Begin
```

```
      Readln(FileModul,S1);
      k1:=NotSpace(S1,1); k2:=Space(S1,k1+1);
      If k2=0 then
        k2:=length(S1)+1;
      S2:=Copy(S1,k1,k2-k1);
      S1:='';
      For i:=1 to 4 do
        Begin
          ch:=S2[i]; k1:=Pos(ch,Dig);
          S1:=S1+Bin[k1];
        End;
      Slovo:=[];
      For i:=1 to 8 do
        If S1[i]='1' then
          Slovo:=Slovo+[i];
      Memory[k]:=Slovo;
      Inc(k); Slovo:=[];
      For i:=9 to 16 do
        If S1[i]='1' then
          Slovo:=Slovo+[i-8];
      Memory[k]:=Slovo; Inc(k);
    End;
  SAK1:=SAK;
  Close(FileModul);
  WriteXY(MesReg,1,25,BlueWhite); WaitEnter;
End { InputModul };
 ------------------------------------------------------------

Procedure Run;
{ Непрерывный режим выполнения машинной программы }
Var  i,Switch : byte;
Begin
  CondExec:=true;
  While CondExec do
    Begin
      RK.KOP:=[]; RK.MA:=[]; RK.NR:=[]; RK.AM:=[];
      RS:=Memory[SAK]; RK.KOP:=RK.KOP+RS;
      If RK.KOP<>[] then
        Begin
          For i:=5 to 6 do
            If i in RS then
              RK.MA:=RK.MA+[i-4];
          For i:=7 to 8 do
            If i in RS then
              RK.NR:=RK.NR+[i-6];
          RS:=Memory[SAK+1]; RK.AM:=RS;
          NumReg:=0;              { перевод значения номера }
          If 1 in RK.NR then      { регистра RK.NR в числовую }
            Inc(NumReg,2);        { форму }
          If 2 in RK.NR then
            Inc(NumReg);
          Inc(SAK,2);
```

```
          Switch:=0;              { перевод значения кода }
          For i:=1 to 4 do        { операций в числовую форму }
            Begin
              Switch:=2*Switch;
              If i in RK.KOP then
                Switch:=Switch+1;
            End;
          Case Switch of
            1 : Load;
            2 : Store;
            3 : Add;
            4 : Subtract;
            5 : Multiple;
            6 : Divide;
            7 : Conjunct;
            8 : Disjunct;
            9 : Exclus;
           10 : ShiftLeft;
           11 : ShiftRight;
           12 : BranchCond;
           13 : Compare;
           14 : ToSubroutine;
           15 : FromSubroutine;
         end;
       End
      Else
        CondExec:=false
    End;
  WriteXY(MesReg,1,25,BlueWhite); WaitEnter;
End { Run };
 ------------------------------------------------------------

Procedure Step;
{ Шаговый режим выполнения машинной программы }
Begin
End { Step };
 ------------------------------------------------------------

Procedure WriteStore;
{ Вывод на экран содержимого оперативной памяти Memory }
Begin
End { WriteStore };
 ------------------------------------------------------------

Procedure From10To16;
{ Перевод целого положительного числа из  10 с/с  в  16 с/с }
Begin
End { From10To16 };
------------------------------------------------------------ }

End.
```

```
{$R+}
Program Emulat;
{ Основная программа эмулятора }

Uses Crt,Dos,DesUnit,BasUnit,RegUnit,InsUnit;

{ ------------------------------------------------------------ }

Procedure Title;
{ Формирование заставки }
Begin
End { Title };
{ ------------------------------------------------------------ }

Procedure MakeMenu;
{ Формирование основного меню }
Const
   LeftBorder = 18;      { левая граница рамки }
   UpBorder = 6;         { верхняя граница рамки }
   MenuCount = 6;        { кол-во позиций меню }
   LenSt = 32;           { длина строки позиции меню }
   St : array[1..MenuCount] of string[LenSt] =
       ('Ввод исходных данных',
        'Полное выполнение программы',
        'Шаговое выполнение программы',
        'Вывод области памяти',
        'Преобразование  10 --> 16',
        'Выход из меню');
Var  i, Switch : byte;
Begin
  TextBackGround(Cyan); ClrScr; { очистка экрана }
  SetCursor($2000);          { отключение курсора }
               { теневая рамка }
  Frame(LeftBorder,UpBorder,LeftBorder+LenSt+6,
        UpBorder+2*MenuCount+2,3,Blue);
                { основная рамка }
  Frame(LeftBorder+1,UpBorder+1,LeftBorder+LenSt+7,
        UpBorder+2*MenuCount+3,2,LightGrayBlack);
  WriteXY(' Р Е Ж И М   Р А Б О Т Ы ',LeftBorder+9,
          UpBorder+1,LightGrayBlack);
  For i:=1 to MenuCount do
    WriteXY(St[i],LeftBorder+5,2*i+UpBorder+1,
            LightGrayBlack);
  S:=FillString(80,' ');
  WriteXY(S,1,24,MagentaWhite);
  WriteXY(S,1,25,BlueLightGray);
  TextAttr:=CyanBlack;
  GotoXY(1,1); InsLine;
  WriteLastString(Mes1,BlueBlueLightRed,
                  BlueLightGray);
  Repeat
    MoveFromScreen(BufScreen1,1,25); { сохранение экрана }
    SetCursor($2000);                { отключение курсора }
```

```
    CurrRow:=9; CurrCol:=21; { нач.значение позиции экрана }
          { цветовое выделение начальной позиции }
    InsertColorString(CurrRow,CurrCol,34,BlueYellow);
          { управление передвижением строки-курсора }
    MoveCursor(9,19,2,21,21,0,34,BlueYellow,LightGrayBlack);
    If Key=27 then
      Begin
        TextBackGround(Cyan); ClrScr; Exit;
      End;
    Switch:=(CurrRow-UpBorder-1) div 2;
    Case Switch of
      1 : InputModul;     { Ввод исходного файла }
      2 : Run;            { Полное выполнение программы }
      3 : Step;           { Шаговое выполнение программы }
      4 : WriteStore;     { Вывод содержимого памяти }
      5 : From10To16;     { Преобразование  10 →16 }
      6 : Begin           { Выход из меню }
            TextBackGround(Cyan); ClrScr; Exit;
          End;
    end;
    MoveToScreen(BufScreen1,1,25); {восстановление }
                                   { экрана }
  Until false;
  SetCursor($0607);               { включение курсора }
End { MakeMenu };
{ ------------------------------------------------------ }
Begin

{ Формирование заставки }
  Title;
  If Key=27 then
    Exit
  Else
    Key:=0;

{ Очистка памяти и регистров }
  StartPosMemory;
  StartPosRegisters;

{ Формирование меню и выбор режима работы программы }
  MakeMenu;

End.
```

Приложение 4

ПРОГРАММА ОРГАНИЗАЦИИ СКРОЛЛИНГА

В программе Scrolling при нажатии клавишей Up или Down производится "прокручивание" вверх или вниз текста, записанного в массиве строк St. Для организации скроллинга применяются процедуры InsLine и DelLine. Текст в массиве St предварительно заполняется символами, выбираемыми случайным образом из таблицы ASCII, для чего используется генератор случайных чисел Random.

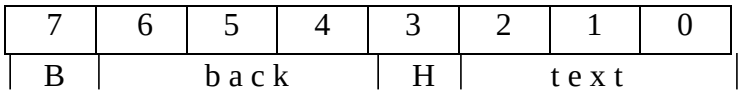
```
Program Scrolling;
{ Скроллинг текста в окне экрана }
Uses Crt,Dos;
Const
    NumStringMax = 100;       { макс.размер массива строк }
    LeftBorder = 17;  { абсцисса левого верхнего угла окна }
    UpBorder = 8;     { ордината левого верхнего угла окна }
    WindowLength = 43;        { длина окна }
    WindowHeight = 12;        { высота окна }
Type
    WindowString = string[WindowLength-1];
    StringAr = array[1..NumStringMax] of WindowString;
Var i,j,k,
    Key,                { код клавиши }
    NumString : byte;   { номер элемента массива St,который }
    Cond  : boolean;    { расположен в первой строке окна }
    St : StringAr;      { массив строк }
{ --------------------------- }
Procedure SetCursor(Cursor:word);
{ Управление курсором }
Var  Reg : Registers;
Begin
  With Reg do
    Begin
      AH:=1; BH:=0;
      CH:=Hi(Cursor); CL:=Lo(Cursor);
    End;
  Intr($10,Reg);
End { SetCursor };
{ ------------------------------- ]
Function GetKey : byte;
{ Чтение символа клавиатуры }
Var  ch : char;
Begin
  ch := ReadKey;
  If ord(ch) = 0 then           { Расширенный код символа }
    GetKey := ord(ReadKey)
  Else
    GetKey := ord(ch);          { Нормальный код символа }
End { GetKey };
------------------------------------------------------
Begin
```

```
  TextBackGround(Black);
  ClrScr;

{ Формирование массива строк }
  Randomize;
  For i:=1 to NumStringMax do
    Begin
      St[i]:=' ';
      For j:=1 to WindowLength-1 do
        Begin
          k:=Random(200)+32;
          St[i]:=St[i]+chr(k);
        End;
    End;

{ Формирование окна }
  Window(LeftBorder,UpBorder,LeftBorder+WindowLength-1,
         UpBorder+WindowHeight-1);
  TextAttr:=Blue + Brown shl 4;
  ClrScr;
  SetCursor($2000);                { Отключение курсора ]

{ Начальное заполнение окна текстом из массива St }
  For i:=1 to WindowHeight do
    Begin
      GotoXY(1,i); Write(St[i]);
    End;
  NumString:=1;

{ Управление скроллингом текста }
  Repeat
    Key:=GetKey;  Cond:=false;
    Case Key of
      13,27 : Cond := true; Enter,Escape
         72 : If NumString <= NumStringMax - WindowHeight then
                Begin                                  { Up }
                  GotoXY(1,1); DelLine;
                  GotoXY(1,WindowHeight);
                  Write(St[NumString+WindowHeight]);
                   Inc(NumString);
                End;
         80 : If NumString > 1 then                    { Down }
                Begin
                  GotoXY(1,1); InsLine;
                  Write(St[NumString-1]);
                  Dec(NumString);
                End;
    end;
  Until Cond;
  SetCursor($0607);                      { Включение курсора }
End.
```

Приложение 5

ЦВЕТОВЫЕ АТРИБУТЫ

Структура байта атрибутов:



Здесь *back* - цвет фона, *text* - цвет символа. Каждый разряд в разделах *back* и *text* - это один из основных цветов монитора: красный, зеленый или синий. Смешивание основных цветов дает 8 цветовых комбинацій. *H* - разряд интенсивности цвета. С учетом интенсивности получаем 16 цветов для отображения символа (темные и светлые). Для фона возможны только темные цвета. Разряд *B* управляет мерцанием символа: если *B* = 1, символ мигает с частотой около 2 гц.

К видеопамяти можно обращаться из программы с помощью адреса. Для этого в ПЭОМ зарезервированы адреса с сегментной частью от $A000 до $DFFF, то-есть 256 Кбайт.

В стандартном модуле Crt для текстового режима экрана определены константы для всех 16 значений цветового атрибута в соответствии с приведенной ниже таблицей.

Бит H	Бит цвета кр.	зел.	син.	Десятичное значение	Имя константы	Цвет
0	0	0	0	0	Black	Черный
0	0	0	1	1	Blue	Синий
0	0	1	0	2	Green	Зеленый
0	0	1	1	3	Cyan	Голубой
0	1	0	0	4	Red	Красный
0	1	0	1	5	Magenta	Фиолетовый
0	1	1	0	6	Brown	Коричневый
0	1	1	1	7	LightGray	Светло-серый
1	0	0	0	8	DarkGray	Темно-серый
1	0	0	1	9	LightBlue	Ярко-синий
1	0	1	0	10	LightGreen	Ярко-зеленый
1	0	1	1	11	LightCyan	Ярко-голубой
1	1	0	0	12	LightRed	Розовый
1	1	0	1	13	LightMagenta	Малиновый
1	1	1	0	14	Yellow	Желтый
1	1	1	1	15	White	Белый
				128	Blink	Мерцание символа

Приложение 6

ТАБЛИЦА ASCII

При формировании в программе рамок, таблиц и т.п. зачастую используются символы псевдографики. Коды этих символов содержатся в ASCII-таблицах, приводимых в справочной литературе.

При отсутствии справочных материалов для определения ASCII-кода любого символа можно использовать приведенную ниже программу ASCII.

```
Program ASCII;
{ Символы таблицы ASCII }
Uses Crt;
Const
   SpecialChars: set of char =
                 [#7,#8,#10,#13];
     ch : char = #0;
Var
   i,k : byte;
   S : string;
{ ------------------------ }
Procedure WaitEnter;
Var  ch : char;
Begin
  Repeat
    ch:=ReadKey;
   Until ord(ch)=13;
End { WaitEnter };
{ ------------------------ }
Procedure Symbols;
Var  i : byte;
Begin
  For i:=1 to 10 do
    Begin
      If ch in SpecialChars then
        Write('   ')
      Else
        Write(ch,' ');
      If ch=#255 then
        Begin
          WaitEnter; Halt
        End;
      ch:=succ(ch);
    End;
End { Symbols };
{ ------------------------ }
Begin
  ClrScr; k:=0;
  For i:=1 to 13 do
    Begin
      Str(k:3,S); Write(S,' ');
```

```
      Symbols; Inc(k,10);
      Str(k:3,S); Write('',S,'  ');
      Symbols; Inc(k,10);
      Writeln; Writeln;
    End;
End.
```

Приложение 7

РАСШИРЕННЫЕ КОДЫ КЛАВИШЕЙ

При формировании различного вида меню, строк статуса и т.п. используются специальные клавиши (функциональные, перемещения курсора, сочетания клавиши Alt или Ctrl с другими клавишами и т.д.). Кодирование таких клавишей осуществляется расширенным ASCII-кодом, который представляется двухбайтными последовательностями. Первый байт является служебным и равен нулю, второй байт - информационный. С помощью расширенных ASCII-кодов кодируются сигналы следующих клавишей и их комбинаций:

- функциональных (F1..F10);
- удаления и вставки (Del и Ins);
- перемещения курсора (стрелки, Home, End, PgDn, PgUp, BackSpace);
- Alt + (другая клавиша);
- Ctrl + (другая клавиша);
- Shift+Tab.

Таблицы расширенных кодов клавишей приведены в справочной литературе. При отсутствии справочных материалов для определения кода любой клавиши или комбинации клавишей можно использовать приведенную ниже программу KeyTest.

```
Program KeyTest;
{ Коды клавишей и их комбинаций }
Uses Crt;
Var ch1,ch2 : char;
Begin
  ClrScr;
  Repeat
    ch1:=ReadKey;
    Writeln('ch1= ',ch1,' ord(ch1)= ',ord(ch1));
    If ord(ch1)=0 then
      Begin
        ch2:=ReadKey;
        Writeln('ch2= ',ch2,' ord(ch2)= ',
                ord(ch2));
      End;
  Until ord(ch1)=13;
End.
```

С О Д Е Р Ж А Н И Е