ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Северо-Кавказский филиал
ордена Трудового Красного Знамени федерального государственного бюджетного образовательного учреждения высшего образования
"Московский технический университет связи и информатики"

Методические указания
к лабораторным и практическим занятиям

МИКРОПРОЦЕССОРЫ

Направление подготовки:
09.03.01 Информатика и вычислительная техника
11.03.02 Инфокоммуникационные технологии и системы связи

Ростов-на-Дону
2019

УДК 681.3.06 (076)
ББК 32.07

Чикалов А.Н. Микропроцессоры. Методические указания к лабораторным и практическим занятиям. - Ростов-на-Дону: Северо-Кавказский филиал МТУСИ, 2019. - 76 с.

В пособии изложены методические рекомендации, содержательные материалы и контрольные задания для проведения лабораторных и практических занятий по изучению принципов построения микропроцессоров, исследованию поведения микропроцессора при выполнении основных типов команд, входящих в его архитектуру, освоению приемов программирования. Приведены основные этапы и примеры разработки программного обеспечения. Пособие содержит необходимые справочные материалы.

Методические указания предназначены для студентов, обучающихся по направлениям подготовки 09.03.01 Информатика и вычислительная техника и 11.03.02 Инфокоммуникационные технологии и системы связи, профилей Многоканальные телекоммуникационные системы, Сети связи и системы коммутации, Защищенные системы и сети связи, Системы радиосвязи и радиодоступа, Вычислительные машины, комплексы, системы и сети, Программное обеспечение и интеллектуальные системы.

В разделе 4 использованы материалы, предоставлены Тимофеевым В.И., пример программы задания 1 подготовлен выпускником СКФ МТУСИ Ивановым М.А.

Пособие предназначено для использования при изучении дисциплин Микропроцессорные системы, Вычислительная техника и информационные технологии, а также может быть использовано преподавателями и студентами при изучении родственных дисциплин и в процессе самостоятельной работы.

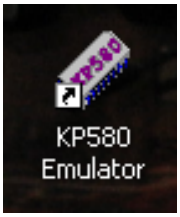
Учебное пособие обсуждено и одобрено на заседании кафедры ИВТ
Протокол №1 от 26.08.2019

Рецензент Зав. кафедрой ИВТ д.т.н. профессор Соколов С.В.

Содержание

ОПИСАНИЕ РАБОТЫ ЭМУЛЯТОРА МИКРОПРОЦЕССОРА

Изучение работы микропроцессоров будет осуществляться на основе эмулятора микропроцессора КР580ВМ80. Это один из первых отечественных микропроцессоров. Уникальность его заключается в том, что он в силу своего исторического положения очень близко повторил структуру и идеологию работы классических вычислительных машин. Он имеет прозрачную структуру, обладает простой и понятной системой команд, основная часть которой входит во все существующие вычислительные средства различных классов. По этой причине именно этот микропроцессор был выбран для начальных занятий по освоению принципов программирования микропроцессоров.



После установки пакета запуск эмулятора осуществляется с помощью ярлыка.

Эмулятор позволяет:

- получить представление о составе и работе элементов структурной схемы микропроцессора;
- изучать принцип и порядок выполнения команд, в том числе в потактовом режиме;
- разрабатывать, отлаживать и выполнять программы на языке ассемблера;
- получить представление о распределении и взаимодействии с оперативной памятью и принципах обмена с внешними устройствами;
- сохранять, загружать и распечатывать данные и программы.

Интерфейс эмулятора выполнен по современным стандартам типовых графических окон, поэтому прост и интуитивно понятен (рис.1). Состав главного меню, как и в типовых программах Microsoft, в основном дублируется горячими клавишами и инструментальными кнопками на схеме микропроцессора. Кроме того, имеется развитая помощь в виде обширной справки, а также в форме всплывающих подсказок. Чтобы узнать назначение инструментальной кнопки достаточно навести на нее курсор.

На структурной схеме микропроцессора (см. рис.1) представлены основные функциональные узлы процессора и связи между ними. Кроме того, показано содержимое оперативного запоминающего устройства (ОЗУ). При необходимости имеется возможность наблюдать содержимое буферов и внешнее проявление данных отдельных внешних устройств.

Среди функциональных элементов схемы особое значение имеют элементы, образующие так называемую программную модель микропроцессора. Это те элементы, которые прямо или косвенно доступны для управления программными средствами. Другими словами, существуют команды, которые изменяют состояние этих элементов или используют их состояние, и программист, следовательно, в состоянии это делать. Остальные элементы структурной схемы, естественно, также работают при выполнении команд процессора,

однако их состояние программистом не контролируется. Эти средства управляются аппаратно.

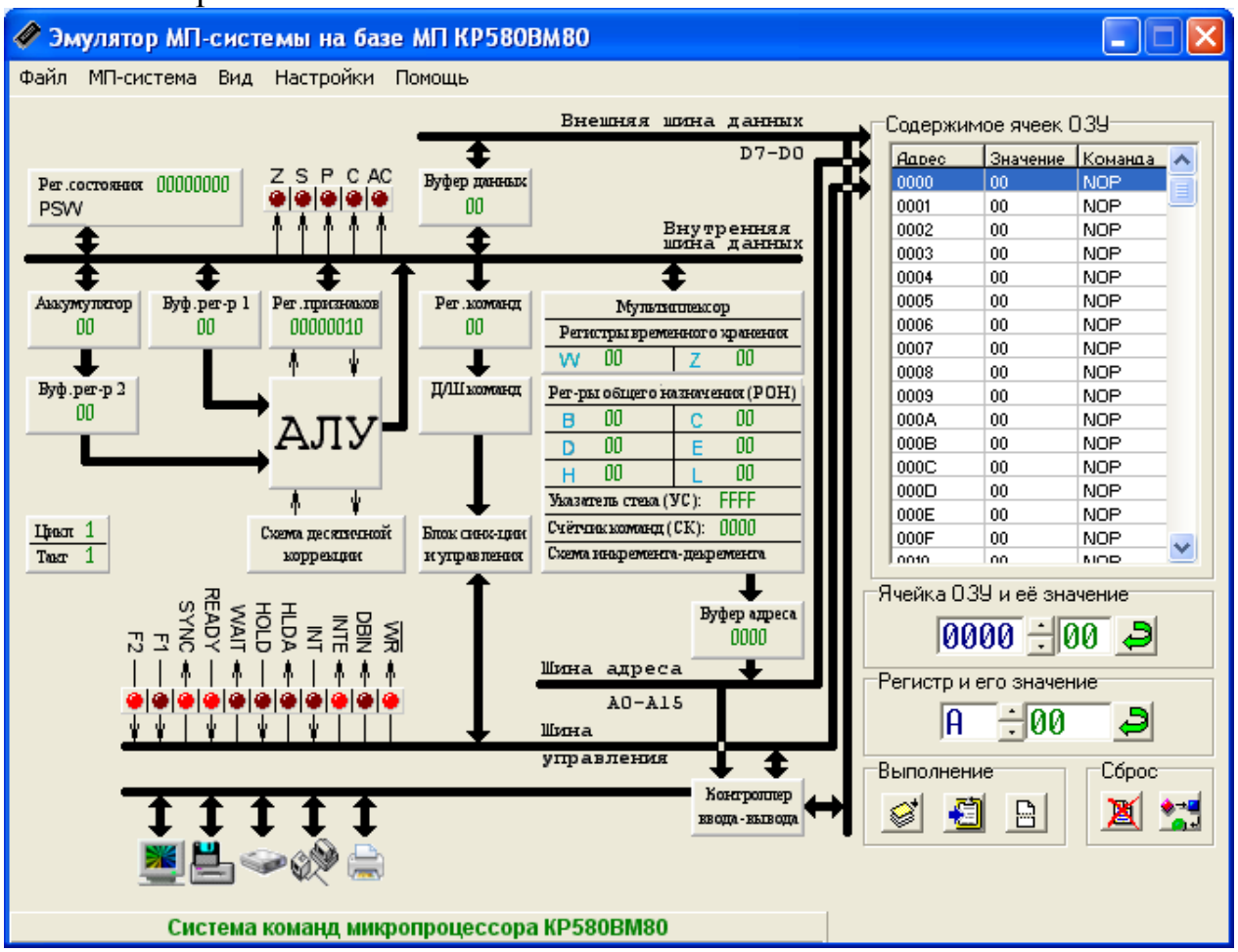
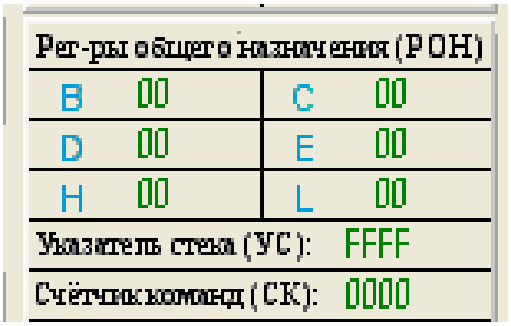


Рис.1. Окно эмулятора МП КР580ВМ80А

К элементам программной модели микропроцессора КР580ВМ580 относятся:

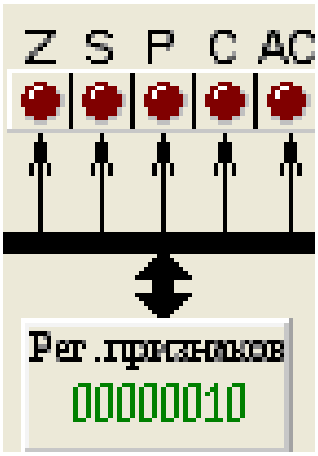
- аккумулятор. Это регистр, в котором находится, как правило, один из операндов выполняемой команды или в который помещается результат этой команды. В этом смысле его назначение уникально, поэтому он имеет такое особое название и в мнемонике команд обозначается латинской буквой "A". Аккумулятор является 8-разрядным;

- регистры общего назначения (РОН). В этом микропроцессоре их шесть. Они именуются и в мнемониках команд обозначаются латинскими буквами: B, C, D, E, H, L. Каждый из них 8-разрядный. Эти регистры могут использоваться как обычные регистры для временного хранения данных, либо каким-либо особым образом, исходя из логики конкретной команды. Задействоваться может как один байт, так и пара регистров сразу. Например, пара регистров H и L, может использоваться для хранения 16-разрядного адреса ячейки ОЗУ при пересылке данных. В этом случае имя этой пары в мнемонике команды обозначается латинской буквой "M";



- указатель стека. В этой модели этот 16-разрядный регистр показывает адрес очередной занятой ячейки ОЗУ при стековых операциях. Заполнение стека осуществляется двумя байтами и адреса при записи изменяются в сторону меньшего значения. По этой причине при отсутствии ограничений начальный адрес указателя стека задается FFFFh. В мнемониках команд этот регистр обозначается "SP". Его значение в процессе выполнения программы никакие аппаратные средства не контролируют, и он может при большом числе операций сохранения в стеке заполнить всю память ОЗУ. Эта проблема должна бать решена программистом;

- счетчик команд. Задача этого 16-разрядного узла – указать устройству управления адрес очередной выполняемой команды. Изменяется его состояние аппаратно и, строго говоря, программист не может командой напрямую задать ему конкретное значение. Однако программист, тем не менее, используя команды передачи управления, косвенно изменяет состояние этого узла. Более того, на языке ассемблера, реализуя определенные цепочки команд, возможно выполнить и чтение и задание значения этому узлу. Однако это скорее уже программные трюки. Но узел включен в состав программной модели;



- регистр признаков (флагов). Как регистр он 8-разрядный, однако, по сути это отдельные биты, устанавливающиеся в зависимости от результатов выполнения конкретной команды. Каждая команда сопровождается своими установками признаков: они могут не изменяться, меняться все либо какой-то отдельный признак. Используются признаки для реализации команд управления вычислительным процессом, поэтому их значению уделяется особое внимание. Программист обязан объективно оценивать состояние этого регистра.

Регистр признаков имеет следующий формат:

	7							0
F:	M	Z	0	AC	0	P	1	C

В МП КР580ВМ80А имеется 5 флагов:

C (Carry)- признак переноса. Устанавливается при наличии переноса (при сложении) или заема (при вычитании) из старшего разряда аккумулятора, иначе сбрасывается. Применяется для реализации на 8-разрядном МП обработки данных длиной в произвольное число байт;

M (Minus) - признак отрицательного результата. Устанавливается, если знаковый бит результата операции (седьмой разряд аккумулятора) равен 1, иначе сбрасывается. Признак используется для организации ветвлений по знаку результата;

Z (Zero) - признак нуля. Устанавливается, если результат операции в аккумуляторе равен нулю, иначе сбрасывается. Признак незаменим для организации циклов и ветвлений;

P (Parity) - признак паритета/четности. Устанавливается, если результат операции в аккумуляторе содержит четное число единиц, иначе сбрасывается;

AC (Auxiliary Carry) - признак половинного переноса. Устанавливается при наличии переноса из третьего разряда аккумулятора в четвертый, иначе сбрасывается.

Остальные биты регистра признаков зафиксированы константами.

Содержимое аккумулятора и регистра флагов называют словом состояния программы PSW (Program Status Word). Для сохранения (извлечения) слова состояния программы в стеке (из стека) эта аббревиатура применяется в мнемонике команды.

Для каждого регистра программной модели МП на структурной схеме имеется индикацию текущего состояния, которое изменяется в процессе выполнения команд. Для ручного изменения состояния регистров используется поле ***Регистр и его значение.*** Адрес регистра выбирается по кругу в окне адреса стрелками, значение кода вводится в окно с клавиатура *шестнадцатеричным кодом* и нажимается в этом поле стрелка ввода..

В возможности эмулятора входит: работа с 5-ю внешними устройствами, такими, как монитор, НГМД, НЖМД, сетевой адаптер и принтер; отладка и выполнение программ в тактовом, командном и сквозном режимах; работа со всем спектром системы команд данного МП; сохранение, загрузка и печать данных и результатов; ручной ввод данных в ОЗУ и РОН. Также, в состав дистрибутива включено подробное руководство пользователя, описание системы команд и файлы-образы ОЗУ эмулятора для примера.

1. ИССЛЕДОВАНИЕ ФУНКЦИОНИРОВАНИЯ МИКРОПРОЦЕССОРА ПРИ ВЫПОЛНЕНИИ КОМАНД ПЕРЕСЫЛКИ

Цель

1. Освоить работу с эмулятором микропроцессора КР580;
2. Исследовать особенности выполнения команд пересылки данных;
3. Совершенствовать навыки анализа, обобщения и систематизации полученных результатов, навыки составления и оформления отчетных материалов, навыки точного и лаконичного представления докладов на вопросы технического характера.

Учебные вопросы

1. Исследовать команды пересылки одного байта MOV, MVI, STAX, LDAX, STA, LDA;
2. Исследовать команды пересылки двух байтов LXI, SHLD, LHLD, SPHL, PUSH, POP;
3. Исследовать команды обмена байтами XCHG, XTHL;
4. Изучить команды ввода-вывода IN, OUT;
5. Разработать программу вывода текста на монитор.

Актуальность занятия

Это самые простейшие и самые часто употребляемые команды. На понимании их строятся все остальные процессы в вычислителях.

Подготовка к занятию

В объеме лекций по структуре и функционированию микропроцессора. Особое внимание уделить вопросам организации памяти, выполнения машинного цикла.

Содержание отчета

1. Название работы;
2. Для каждого задания: название задания и материал в объеме, указанном в задании.

Задания 1.1, 1.2, 1.3. Изучение команд пересылки

Для заданий 1.1-1.3 протестировать выполнение каждой команды из списка учебных вопросов. Описание каждой команды приведено в справке эмулятора.

Все данные по исследованию команд целесообразно оформить в виде таблицы с колонками указанного содержания. Один из возможных вариантов приведен в таблице. При необходимости название, логику работы команды законспектировать.

Мнемоника или примеры	Содержание	Допустимые адреса	Циклы/ Такты	Длина команды, байт	Какие признаки изменяет
MOV A,B	B→R	A, B, C, D, E, H, L, M	1/5-7	1	Не изменяет
MOV A,M	M(HL)→A		5-7	1	

Задания 1.4, 1.5. Изучение команд IN, OUT и написание программы

Для задания 1.4 и 1.5 - изучить принцип выполнения команд IN и OUT в объеме таблицы, предложенной для заданий 1.1-1.3. Изучить протокол обмена с монитором для графического и текстового режимов, руководствуясь справкой эмулятора (раздел Описание программы/Работа с программой/Работа с внешними устройствами/Работа с монитором). Разработать программу вывода сообщения: "число 47"на монитор эмулятора.

В отчете представить:

1. Формат протокола обмена с монитором эмулятора для графического и текстового режимов работы;
2. Исходный текст программы для вывода на монитор эмулятора с комментариями по тексту.
3. Краткие ответы на те контрольные вопросы для самопроверки, которые ещё не нашли своего отражения в отчете.

<u>Вопросы для самопроверки</u>

1. Почему пересылки между A, B, C, D, H, L занимают всего 1 машинный цикл, а с M значительно длиннее?
2. Какие команды занимают 1 байт, какие 2 байта и 3 байта? Почему?
3. Что происходит при выполнение MOV A,A?
4. Каково состояние регистров-источников при окончании пересылки?
5. Сравните время выполнение команд NOP и MOV C,C.
6. Занимается ли ячейка памяти при выполнении PUSH, POP, если при инициализации SPHL HL=0FFFFh? Проверить на эмуляторе.
7. Сколько внешних устройств можно адресовать?
8. Сформируйте точку красного цвета в середине экрана монитора эмулятора.
9. Где располагаются второй и третий байт команды?
10. Какие способы адресации используются при выполнении команд пересылки?
11. Чем отличается текстовый режим работы монитора от графического режима?

2. ИССЛЕДОВАНИЕ ФУНКЦИОНИРОВАНИЯ МИКРОПРОЦЕССОРА ПРИ ВЫПОЛНЕНИИ АРИФМЕТИЧЕСКИХ И ЛОГИЧЕСКИХ КОМАНД

Цель

1. Исследовать особенности выполнения арифметических и логических команд;

2. Получить навыки разработки и отладки элементарных программ на языке Ассемблер;

3. Совершенствовать навыки анализа, обобщения и систематизации полученных результатов, навыки составления и оформления отчетных материалов, навыки точного и лаконичного представления докладов на вопросы технического характера.

Учебные вопросы

1. Исследовать арифметические команды ADD, ADC, SUB, SBB, INR, INX, DAD, CMC, STC, ADI, ACI, SUI, SBI, DCR, DCX;

2. Исследовать логические команды ANA, ORA, XRA, ANI, ORI, XRI, CMA;

3. Исследовать команды сдвига RLC, RRC, RAL, RAR;

4. Разработать программу сложения двух чисел введенных с клавиатуры в буфер.

Актуальность занятия

Команды этой группы являются базовыми для преобразования данных. Совокупность этих или аналогичных команд позволяют выполнить практически преобразования произвольной сложности.

Подготовка к занятию

В объеме лекций по структуре и функционированию микропроцессора. Особое внимание уделить вопросам представления данных в ЭВМ и технологии выполнения арифметических и логических команд.

Содержание отчета

1. Название работы;

2. Для каждого задания: название задания и материал в объеме, указанном в задании.

Задания 2.1, 2.2, 2.3. Изучение арифметических и логических команд

Протестировать выполнение каждой команды из списка учебных вопросов. Описание каждой команды приведено в справке эмулятора.

Все данные по исследованию команд целесообразно оформить в виде таблицы с колонками указанного содержания. Один из возможных вариантов

приведен в таблице. При необходимости название, логику работы команды законспектировать.

Мнемоника или примеры	Содержание	Допустимые адреса	Циклы/ Такты	Длина команды, байт	Какие признаки изменяет
ADD R	A+R →A	A, B, C, D, E, H, L, M	1/4 или 2/7	1	Все признаки

Выполнить на эмуляторе и убедиться в правильности результата при выполнении представленных ниже операций. Следует учесть, что операции в микропроцессоре выполняются в дополнительных кодах. Двоичные коды операндов занести в отчет.

Сложить	1) B+C	B=121	C=52;
	2) B+C	B=121	C=140;
	3) B+C	B=121	C= -52;
Вычесть	1) B-C	B=121	C=52;
	2) B-C	B= -30	C= -8.

Задание 2.4. Разработка программы сложения двух чисел, введенных с клавиатуры в буфер

Написать и отладить программу выполнения сложения 2-х чисел, введенных с клавиатуры, и документирования результатов на экране монитора.

При нажатии клавиш на клавиатуре в буфере клавиатуры формируются коды нажатых символов. Будем считать, что коды введенных символов складываемых одноразрядных чисел сохраняются в ОЗУ в последовательно расположенных ячейках. Адреса ячеек необходимо выбрать самостоятельно и коды ввести вручную. Все остальные преобразования должны выполняться программно.

Необходимо выполнить сложение введенных чисел и отобразить на экране монитора эмулятора запись операции сложения. Сумма чисел ***не должна превышать 9***. Например, введены цифры 1 (код символа 31h) и 7 (код 37h). Результат документирования на мониторе должен быть в виде "1+7=8". Необходимо обратить внимание на то, что число от соответствующего ему кода символа отличается на 30h.

Тестирование программы следует выполнить введением различных комбинаций символов в "буфер клавиатуры" и последующим контролем правильности протокола на экране монитора эмулятора.

Поскольку это первая программа, которую необходимо написать для микропроцессора на языке ассемблера, то следует начинать с хороших привычек и сделать это рационально с точки зрения практики.

Рекомендуется следующий порядок работы с фиксацией на бумаге каждого этапа:

1. Сформулировать словами последовательность этапов алгоритма. Задача этого этапа – продумать в общем виде основную идею алгоритма. Поэтому формулировать лучше в терминах, которые понятным образом реализуемы на этом языке программирования. Эти записи конкретизируют способ решения задачи и значительно сократят восстановление логики после неизбежных перерывов в работе;

2. Оформить алгоритм графически в виде схемы-алгоритма. Данная задача является линейным алгоритмом и этот этап можно опустить. Но для задач более или менее сложных этот этап обязателен. Степень детализации операторов алгоритма определяет сам автор, опираясь на свои возможности и опыт. Схема позволяет более конкретно и ясно увидеть решение задачи, оценить реализуемость алгоритма, облегчает и ускоряет отладку. Операторы схемы нумеруются сверху вниз и справа налево арабскими цифрами. Цифры проставляются с левой стороны в разрыве линии оператора;

3. Написать на языке ассемблера строки каждого оператора схемы алгоритма. Обратить внимание на комментарии к этим фрагментам текста. Подробность их зависит от желания и замысла автора. Применяют запись словами смысла преобразований в блоке, но можно ограничиться сокращенными пометками, если словесное описание алгоритма этапа 1 достаточно подробно. Обязательно надо проставить цифры номеров операторов из схемы алгоритма. Полезна также запись смысла сохраняемых данных в конкретных регистрах процессора.

4. Провести отладку разработанной программы.

Вопросы для самопроверки

1. В каких кодах выполняются арифметические команды? Как формируются эти коды?
2. Как машина различает: числа со знаком или нет? положительное число или отрицательное?
3. Как выполняются арифметические и логические операции с двумя байтами (двухбайтными числами)?
4. Какие виды циклических операций существуют? Чем они различаются?
5. Что такое признаки (флаги) ЦП?
6. Зачем в системе команд операция сдвига через признак С?
7. Разработайте алгоритм и программу сложения двух символов. На монитор вывести: Символ + Символ = Результат.
8. Каков формат выдачи данных на монитор? Разработайте подпрограмму вывода данных на монитор.

3. ИССЛЕДОВАНИЕ ФУНКЦИОНИРОВАНИЯ МИКРОПРОЦЕССОРА ПРИ ВЫПОЛНЕНИИ КОМАНД ПЕРЕДАЧИ УПРАВЛЕНИЯ

Цель

1. Исследовать особенности выполнения команд управления;
2. Научиться разрабатывать и отлаживать элементарные программы на языке Ассемблер;
3. Совершенствовать навыки анализа, обобщения и систематизации полученных результатов, навыки составления и оформления отчетных материалов, навыки точного и лаконичного представления докладов на вопросы технического характера.

Учебные вопросы

1. Исследовать команды передачи управления PCHL, JMP, J-CON;
2. Исследовать команды вызова и возврата из подпрограмм CALL, C-CON, RET, R-CON;
3. Исследовать команды установки признаков CPI, CMP;
4. Разработать подпрограмму вывода сообщения на экран.

Актуальность занятия

Команды этой группы реализуют процессы принятия решения и позволяют управлять вычислительным процессом в зависимости от реальной ситуации. Без этих команд невозможна гибкость, массовость, сложность программ, их оптимизация интеллектуальность.

Подготовка к занятию

В объеме лекций по структуре и функционированию микропроцессора. Особое внимание уделить вопросам представления данных в ЭВМ и технологии и схемотехнике выполнения арифметических и логических команд, принципам организации подпрограмм.

Содержание отчета

1. Название работы;
2. Для каждого задания: название задания и материал в объеме, указанном в задании.

Задания 3.1, 3.2, 3.3. Исследование команд управления

Протестировать выполнение каждой команды из списка учебных вопросов. Описание каждой команды приведено в справке эмулятора.

Все данные целесообразно оформить в виде таблицы с колонками указанного содержания. Один из возможных вариантов приведен в таблице. При необходимости содержание команды законспектировать.

Мнемоника или примеры	Содержание	Допустимые адреса	Циклы/ Такты	Длина команды, байт	Какие признаки изменяет
JMP ADR	ADR → PC	Физические	3/10	3	Не устанавливает

Задание 3.4. Разработка подпрограммы вывода сообщения на экран

Для разработанной подпрограммы представить:

- словесное описание алгоритма;
- схема алгоритма, перечень и назначение переменных с присвоенными адресами;
- текст на Ассемблере;
- файл с работающей программой на эмуляторе.

Для разработки программы следует учесть следующее.

1. Текстовые сообщения в программе кодируются в кодах ASCII (см. приложение А). Это значит, что каждый символ сообщения хранится в виде 8-разрядного кода, который при выводе выдается по адресу внешнего порта монитора (говорят - "выдается в экран"). Драйвер монитора при поддержке аппаратной части монитора и знакогенератора формирует по этому коду совокупность точек на экране (знакоместо в виде прямоугольника заданного размера), которые и образуют символ, соответствующий этому коду. Прямоугольник знакоместа помимо символа, формируемого точками активного цвета, заполняется также точками цвета фона. Таким образом, каждый новый выводимый символ переписывает на экране все точки прямоугольника знакоместа.

2. Каждый символ отправляется по адресу порта экрана и выводится в то место, которое указывается так называемым текущим положением курсора. Курсор может перемещаться от левой верхней координаты экрана слева направо сверху вниз до правой нижней координаты. Далее происходит прокрутка экрана: экран как бы смещается вверх на одну строку, а курсор начинает заполнять последнюю строку экрана. При выводе очередного символа автоматически курсор смещается на одну позицию вправо. После заполнения всей строки курсор перемещается на следующую строку вниз.

3. Управляющие символы для организации текста на экране также вставляются в текстовое сообщение наряду с печатаемыми (выводимыми на экран) символами (см. приложение А). Совокупность допустимых управляющих символов и характер реакции на них определяются используемым

драйвером монитора. Однако имеются коды, которые используются практически всегда. К ним относятся код 0Ah - перевод строки и 0Dh - возврат каретки. Они исторически унаследованы от печатной машинки и телетайпа и соответствовали операциям механического привода этих устройств. По первому из них курсор опускается вертикально вниз на следующую строку, а по второму - перемещается по строке влево до упора. Эти два кода формируются в набираемое сообщение текстового файла при нажатии клавиши ENTER, что отображается на экране как переход к началу новой строки.

4. Любой выводимый на экран текст со своей структурой в контексте программирования называют ***сообщением***, т.е. совокупностью кодов, которые необходимо от начала и до конца выводить по адресу монитора. Часто в контексте формата весь этот же текст называют ***строкой***, хотя в смысле языка строчек там может быть много. Здесь важна процедура машинной обработки: вывести на экран все коды от начала и до конца.

Естественно, что выводимые сообщения должны храниться в памяти. Поэтому для их вывода необходимо определить где находятся эти коды и сколько кодов следует вывести. Задать эти параметры можно тремя способами:

- задать адрес начала размещения кодов и адрес последнего кода;
- задать адрес начала размещения кодов и количество выводимых кодов;
- задать адрес начала размещения кодов и код (признак) окончания сообщения. Последний способ применяется наиболее часто, им и следует воспользоваться для разработки программы. Признаком окончания сообщения принято считать код 0h. В этом случае используют обозначение формата, подчеркивающее принцип кодирования и способ задания сообщения - ASCIIZ. Последний символ подчеркивает присутствие нуля в конце сообщения: Z - (Zero) - нуль. Поэтому алгоритм должен обеспечить вывод всех кодов, начиная с первого, до обнаружения кода 0h, который, естественно, выведен быть не должен.

5. Поскольку разрабатывать необходимо подпрограмму, то для нее следует передавать параметр - выводимую строку. Передача параметров осуществляется в микропроцессорной технике через стек, регистры процессора или области памяти, которые все равно указываются через регистры процессора, которые передают адреса этих областей. Для рассматриваемого формата сообщения в регистрах все коды передать невозможно, поэтому область хранения в памяти выводимых кодов следует передать как адрес начала этой области. Чаще для этого используют регистровую пару HL, в которую перед обращением к подпрограмме заносят адрес начала области выводимых кодов. С учетом этого должен быть построен алгоритм.

Вопросы для самопроверки

1. Как в машине выполняется управляющая конструкция типа "развилка"? Какие данные и как обрабатывает машина при ее реализации?

2. Как выполняется команда RET? Состояние каких регистров при этом изменяется?
3. Зачем организуются подпрограммы?
4. Какова технология разработки программ?
5. Состояние каких регистров изменяется при выполнении команд CPI и CMP?
6. Почему вывод сообщения оформляется в формате подпрограммы?

4. ИССЛЕДОВАНИЕ ФУНКЦИОНИРОВАНИЯ МИКРОПРОЦЕССОРА ПРИ ВЫПОЛНЕНИИ ПРОГРАММ

Цель

1. Выработать навыки разработки программ для управления микропроцессором;

2. Получить представление о выполнении типовых процедур в составе системного программного обеспечения вычислительной техники;

3. Совершенствовать навыки анализа, обобщения и систематизации полученных результатов, навыки составления и оформления отчетных материалов, навыки точного и лаконичного представления докладов на вопросы технического характера.

Учебные вопросы

1. Разработка алгоритма и программы типовой процедуры;
2. Отладка программы на языке Ассемблер.

Актуальность занятия

Микропроцессоры являются универсальным логическим устройством, поведение которого задается используемой программой. Все современные более или менее сложные устройства и системы управляются микропроцессорами. Поэтому просто необходимо получить навыки разработки типовых процедур из состава системного программного обеспечения. Предложенная в качестве задания программа расширит представления о механизмах работы типового интерфейса интерактивной вычислительной системы.

Подготовка к занятию

Требуется повторить методику синтеза программы для ЭВМ, обобщить сведения по изученной системе команд микропроцессора.

Содержание отчета

1. Название работы.
2. Для разработанной подпрограммы:
   - перечень, форматы и назначение переменных;
   - словесное описание алгоритма;
   - схема алгоритма,
   - текст на языке Ассемблер с привязкой к номерам операторов в схеме-алгоритма (с помощью нумерации операторов и комментариев);
   - исполняемый файл с отлаженной программой.

Задание 4.1. Разработка алгоритма и программы типовой процедуры

В качестве примера приведены этапы разработки подпрограммы для пересылки блока данных из исходной области памяти в область назначения.

При выполнении реального задания следует уделить внимание интерфейсу разрабатываемой программы. Экран не должен быть темным, без указания пользователю требуемых манипуляций, сообщениях о процессе выполнения программ и т.д.

Для начала следует уяснить задачу. Проблема в том, что задача может иметь три варианта решения:

- если адрес назначения не лежит внутри исходной области, то пересылать можно просто с начального адреса (рис.4.1 а, б);
- если адрес назначения находится внутри исходной области, то, не повреждая данные, можно их переписывать, начиная только с конечного адреса (рис.4.1.в);
- если объем пересылаемых данных равен нулю, то пересылать ничего не нужно и сразу организуется выход из подпрограммы.

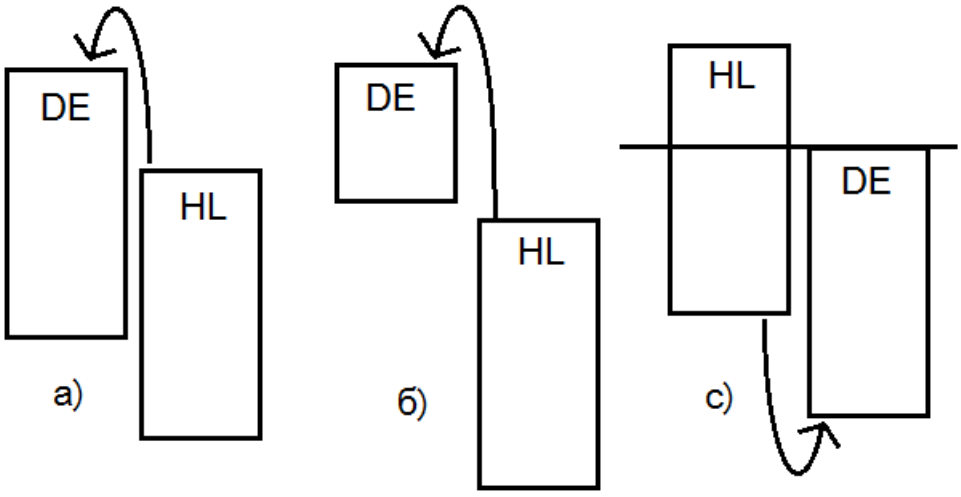


Рис.4.1. Адрес назначения за пределами исходной области данных (а, б), адрес назначения внутри исходной области

1. Перечень, форматы и назначение переменных.

Это первый этап потому, что от него зависит весь алгоритм. Другие исходные данные и их форматы приведут к иному алгоритму для одной и той же задачи.

Для подпрограммы предлагаются следующие исходные данные (см. рис.4.1).

Входные:

- HL - адрес начала исходной области;
- DE - адрес ячейки куда пересылаются данные;
- BC - количество пересылаемых байт.

Выходные: блок перемещен

2. Словесное описание алгоритма.

Этот этап должен дать представление о подходе, положенном в основу алгоритма. Поэтому его следует формулировать в терминах близких к системе команд и с учетом принятого представления входных данных.

1) Если объем пересылаемого блока (BC) равен нулю, то выход из подпрограммы;

2) Если адреса начала и конца блока совпадают (DE=HL), то блоки совпадают, переносить не надо и выход из подпрограммы;

3) Если разность адресов (DE-HL), больше длины блока передаваемых данных (BC), то перемещаем данные с их начала, иначе - к адресам в регистрах DE и HL прибавляем длину передаваемых данных из регистра BC, и данные перемещаем с конца;

4) Перемещая данные, регистр BC уменьшаем на единицу после каждого перемещенного байта и проверяем его содержимое на равенство нулю. Если содержимое регистра BC равняется нулю, то выходим из подпрограммы.

3. Схема алгоритма.

Схема алгоритма уточняет процесс преобразования данных в соответствии со словесным описанием алгоритма. Степень детализации должна быть такова, чтобы по схеме можно было написать исходный текст программы. Если алгоритм сложный, то может быть разработано несколько алгоритмов с различной степенью детализации в итерационном режиме.

В процессе итерационных прогонов алгоритма уточняется динамика изменения состояния регистров, места их временного сохранения и т.д. Все это следует вносить в схему алгоритма, чтобы ускорить и облегчить последующую отладку программы.

Для разрабатываемой задачи алгоритм PODPROGR представлен на рис.4.2.

4. Текст программы.

Текст программы записывается в формате программ ассемблера. Каждый оператор алгоритма получает свой номер в комментариях программы. Приветствуются и смысловые комментарии, поясняющие логику работы программы.

```
PODPROGR:MOV     A,B        ; 1 Проверка BC=0
         ADD     C
         ANA     A          ; установка признаков
         RZ                 ; Выход из подпрограммы, если Z
         MOV     A,E        ; 2 Проверка DE=HL
         CMP     L
         JNZ     M3
         MOV     A,D
         CMP     H
         RZ                 ; Выход из подпрограммы, если Z
```

```
M3:     PUSH    H       ; 3 Вычитание DE-HL
        MOV     A,L
        CMA             ; инверсия аккумулятора
        MOV     L,A     ; получение дополнительного кода HL
        MOV     A,H
        CMA
        MOV     H,A
        INX H
        DAD     D       ; Вычитание DE-HL→HL
        MOV     A,H     ; Сравнение (DE-HL) < BC
        CMP     B
        JNC     M4
        MOV     A,L
        CMP     C
        JNC      M4
        POP     H       ; 7    HL+BC,  DE+BC
        DAD     B
        XCHG
        DAD     B
        XCHG
M8:     MOV     A,M     ; 8 Перемещение данных в направлении
        XCHG            ; снизу вверх
        MOV     M,A
        XCHG
        INX     H
        DCX      D
        DCX     B
        MOV     A,B     ; 9  Проверка BC=0
        ADD     C
        ANA     A
        RZ              ; Выход из подпрограммы, если Z
        JMP      M8
M4:     POP      H      ; 4
M5:     MOV     A,M     ; 5 Перемещение сверху вниз
        XCHG
        MOV     M,A
        XCHG
        INX     H
        INX     D
        DCX     B
        MOV     A,B     ; 6  Проверка BC=0
        ADD     C
        ANA     A
        RZ              ; Выход из подпрограммы, если Z
        JMP     M5
```

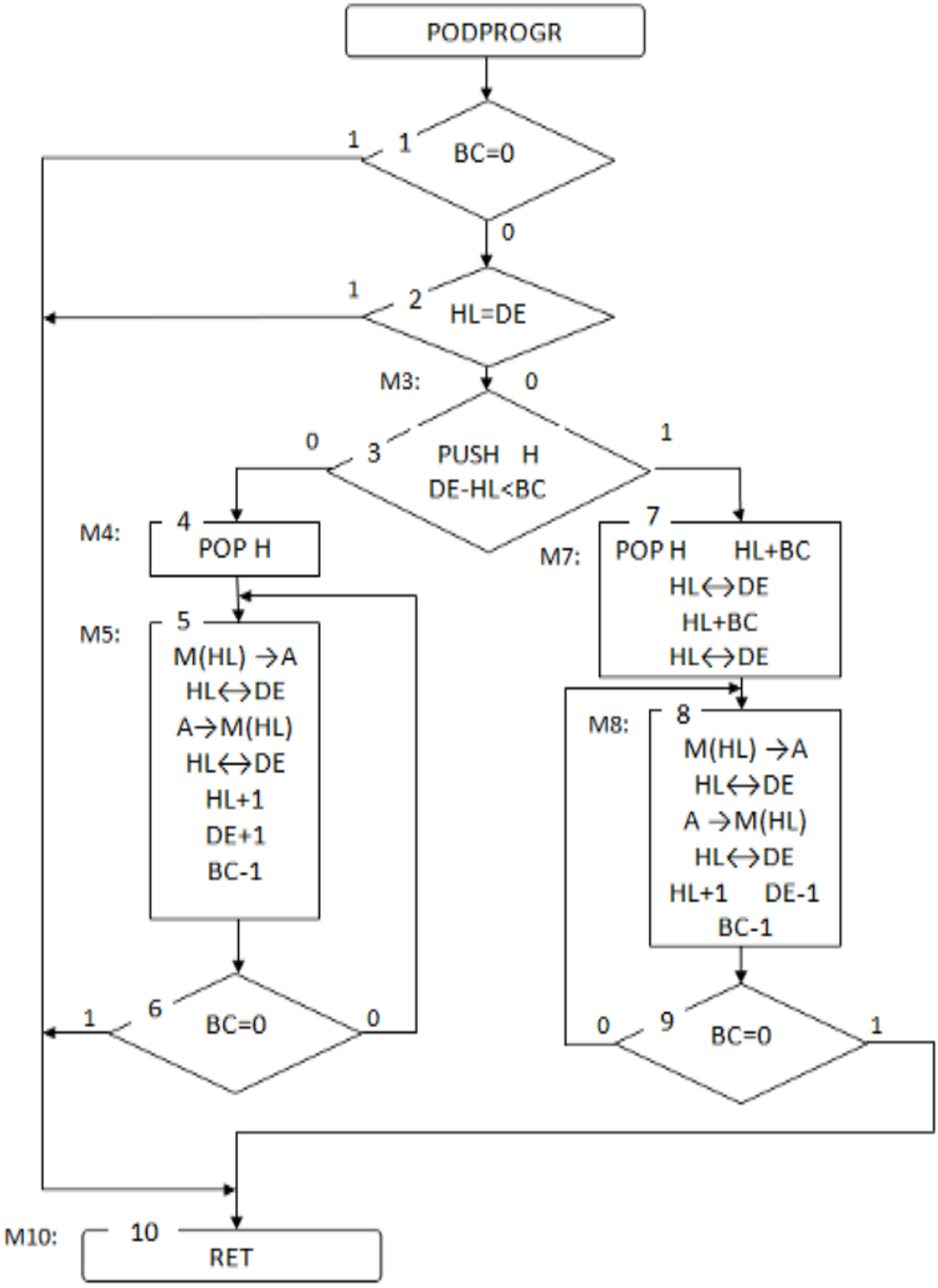


Рис.4.2. Схема алгоритма подпрограммы PODPROGR

Альтернативные задания:

1) Вывод сообщения на экран;
2) Ввод сообщения в формате ASCIIZ в буфер заданной длины;

3) Вывод десяти одинаковых строк с увеличивающимся расстоянием между строками;
4) Вывод десяти одинаковых строк с увеличивающимся расстоянием между строками и изменяющимся цветом;
5) Преобразовать байт в визуальное представление двоичного числа на экране;
6) Преобразовать байт в шестнадцатеричное представление числа на экране;
7) Вставка блока в строку;
8) Поиск подстроки в строке;
9) Текстовый редактор строки;
10) Вывод таблицы формата "Код-цвет" для различных комбинаций физического цвета;
11) Дизассемблирование задачи "Участникам олимпиады" из предоставленного преподавателем файла;
12) Найти минимальный код в заданной строке;
13) Найти сумму чисел массива;
14) Найти номер элемента массива с минимальным значением;
15) Сравнить две строки;
16) Объединить две строки;
17) Преобразование десятичного числа в коде ASCII в двоичное;
18) Преобразование двоичного числа в десятичное в коде ASCII;
19) Преобразование двоичных данных в шестнадцатеричные в коде ASCII.

Разработка задачи осуществляется в порядке, который предложен в примере.

Исходный текст программы следует набирать в текстовом редакторе Блокнот с именем test.txt и последующим изменением расширения на .asm. Команды программы пользователя должны размещаться с адреса 2000h. Файл должен предусматривать зацикливание программы. В противном случае возможен выход из программы или зависание ЭВМ. Файл исходного текста программы должен быть перенесен в папку разработчика.

Программа должна иметь определенную структуру. Пример такой программы приведен ниже для тестового примера, который можно запустить из рабочей папки для проведения занятия.

```
                        ORG     2000H
; тестовый  вариант для  КР580
;
BASE        EQU  120h
INIT        EQU  BASE           ;120H инициализация пакета
KEY         EQU  BASE+3         ;123H опрос состояния клавиатуры
CONOUT      EQU  BASE+9         ;129H вывод символа на экран
CONIN       EQU  BASE+18        ;132H ввод символа с клавиатуры
```

```
SPIC    EQU  BASE+24       ;138H вывод сообщения на экран
INKEY   EQU  BASE+27       ;13BH опрос без ожидания
COLOR   EQU  BASE+54       ;156H установка цвета отображения
FON     EQU  BASE+57       ;159H установка фона экрана
BORD    EQU  BASE+60       ;15CH установка цвета бордюра
SCOLOR  EQU  BASE+105      ;189H установка таблицы цветов
SPLAN   EQU  BASE+108      ;18CH уст. экранных плоскостей
LINE    EQU  141H
PLOT    EQU  13EH
DUMP    EQU  135H
MASC    EQU  16EH
TIME    EQU  16BH
PUT     EQU  14DH
;
BEGIN:                     ; Блок инициализации
        LXI     SP,52FFH   ; НЕ ИЗМЕНЯТЬ !
        CALL    INIT
        MVI     A,0FH
        CALL    SPLAN
        LXI     D,TABCOL
        CALL    SCOLOR
        MVI     A,40H
        CALL    FON
        MVI     A,7
        CALL    COLOR
        MVI     A,0D0H
        CALL    BORD

;Здесь можно вводить команды программы пользователя

; вывод сообщения и символа в качестве примера
        LXI     H,TXT11
        CALL    SPIC
 MI1:   CALL    CONIN
        MOV     C,A
        CALL    CONOUT
        LXI     H,NAZHAT
        CALL    SPIC
        JMP     MI1

;Конец программы пользователя. Далее - подпрограммы,
; таблица цветов, текстовые сообщения, переменные, буферы

;WZW:   CALL    132H       Вариант завершения программы для
;       JMP     WZW        невыхода из эмулятора
```

```
; Управляющая последовательность 1BH,'Y',21H,23H позволяет
;установить курсор по координатам 21H,23H
TXT11:      DB    1BH,'Y',21H,23H
            DB    'SKF MTUSI','    ВЫВОД СООБЩЕНИЯ',0Ah,0Dh,0
;Коды 0AH,0DH позволяют перевести курсор на новую строку:
;они эквивалентны нажатию клавиши ENTER
NAZHAT:  DB    '     NAZHATA KLAVISHA',0AH,0DH,0
;
;
TABCOL:  DB    64,128,16,208,6,134,22,54        ;Таблица цветов
         DB    0,197,34,192,2,152,82,173        ;Для инициализации
END                        ;Обязательная директива конца исходного
                           ; текста
```

Знакогенератор при выполнении программы может отображать символы неправильно. Поэтому выводимые символы целесообразно кодировать кодами латинского алфавита.

Результаты выполнения приведенного примера соответствуют рис.4.9.

Задание 4.2. Отладка программы на языке ассемблера

Для трансляции и отладки программы для 8-разрядного процессора КР580 (а также микропроцессоров 8080, 8085) необходимы следующие программные средства:

- эмулятор операционной системы CP/M (файл cpm.exe);
- компилятор для 8-разрядных микропроцессоров MAC (файл mac.com);
- символьный отладчик SID для 8-разрядного процессора (файл sid.com);
- драйверы устройств 8-разрядной вычислительной машины Вектор 06Ц (файл drid.com);
- файл исходного текста программы с обязательным именем test.asm;
- эмулятор вычислительной машины Вектор 06Ц (файл Vector.exe);
- командный файл компоновки (файл com.tis).

Все файлы необходимо иметь в одной папке, размещенной на рабочем столе. Кроме того, необходимо установить программу DOSBox 0.74 и вынести ее ярлык на рабочий стол.

Отладка предполагает реализацию нескольких этапов.

1. Выполнение компиляции

Для выполнения компиляции файла исходного текста ***test.asm*** необходимо под управлением эмулятора CP/M запустить компилятор MAC. Для

этого следует запустить командный файл ***mac.bat***. Его содержимое представлено на рис.4.3.

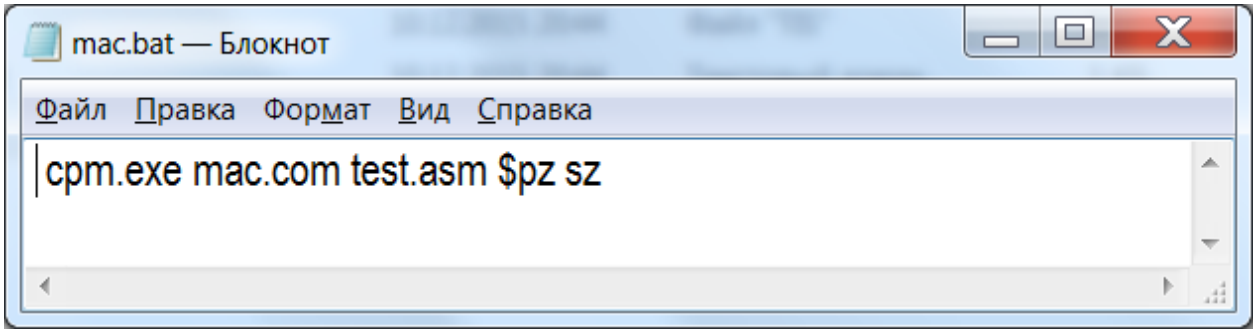


Рис.4.3. Содержание командного файла для выполнения компиляции

Строка командного файла предписывает запустить ассемблер mac.com под управлением эмулятора cpm.exe. В этой строке предусмотрено фиксированное имя файла исходного текста - test.asm и его изменять недопустимо. Объект для компиляции должен быть указан в командном файле явно. Компилятор создает в процессе работы файлы с расширением .HEX, .PRN и .SYM.

В командном файле параметры ассемблирования отделены знаком "$". Назначение параметров:

- ***pz*** - определяет запрет на создание файла печати .PRN. На этапе отладки этот параметр целесообразно не исключать, т.к. в файле .PRN можно увидеть наличие синтаксических ошибок, их расшифровку, области распределения памяти и т.д. Структура файла представлена при рассмотрении сообщений об ошибках;

- ***sz*** - определяет запрет на создание файла символьных имен (таблицы символов) с расширением .SYM. В этом файле указываются все символьные имена и присвоенные им абсолютные адреса. Это файл действительно не нужен и будет засорять папку основных документов;

- ***$1*** – такая директива может присутствовать в параметрах трансляции. Она предписывает параллельно с трансляцией вывод на экран протокола трансляции, который аналогичен содержимому файла .PRN.

Кроме этих файлов ассемблер создаст результирующий файл с шестнадцатеричным (объектным) кодом (с расширением HEX). Это файл машинных команд в стандартном формате фирмы Intel. Все файлы создаются на том же диске, на котором размещен ассемблер MAC, и имеют имя файла исходного текста - "test".

Анализируя файл test.PRN, необходимо выявить причины появления ошибок, устранить их и повторить компиляцию. Такие итерации следует проводить до устранения всех ошибок. Отсутствие сообщений об ошибках говорит о том, что в тексте компилятор не обнаруживает синтаксических ошибок. Это не значит, что ошибки логические отсутствуют. Они могут быть обнаружены и исправлены только на этапе отладки.

Перед запуском командного файла лучше удалить старые файлы печати .PRN и HEX-файл. Будущие новые файлы будут иметь те же имена с новым содержанием.

Сообщения об ошибках в исходной программе

Если в процессе ассемблирования программа-ассемблер обнаружит в исходном тексте ошибки, то сообщения об этих ошибках могут выдаваться на экран по мере их обнаружения до завершения процесса ассемблирования. Сообщения об ошибках содержат специальные коды, по которым можно определить тип ошибки. Эти же коды выдаются в соответствующих строках распечатки в файле с расширением ".PRN".

Программа ассемблера в этом файле выдает сообщения об ошибках в следующем формате:

U 0100 210000SORT: LXI H, CONST ;загрузить константу в HL

где:

- символ "U" обозначает код ошибки. Он всегда распечатывается в первой позиции строки, содержащей ошибку. В данном случае символ сообщает, что в строке есть символическое имя, которое не объявлено. Это касается имени CONST, т.к. других имен в этой строке нет. При отсутствии ошибки – символа нет;
- 0100 - четырехзначный шестнадцатеричный адрес команды, в котором размещается код команды, соответствующий этой строке;
- 214601 - реальный шестнадцатеричный код команды, который получен в результате ассемблирования. Нули, как в данном случае, говорят об ошибке, команда полностью не сформирована, транслятор не смог вместо имени подставить его значение. Количество байт в этом числе соответствует длине команды;
- далее размещается запись этой строки в исходном тексте по правилам записи языка ассемблер: метка, мнемоника команды, операнды, комментарии.

Программа-транслятор может выдавать следующие типовые сообщения о синтаксических ошибках в исходном тексте:

D — Ошибка в данных. Эта ошибка возникает в тех случаях, когда выражение в исходной строке не соответствует назначенной ему области данных. Обычно это означает, что выражение слишком длинное и его необходимо по возможности сократить;

E — Ошибка в выражении. Это сообщение означает, что выражение в исходной строке сформировано неправильно или что значение этого выражения слишком длинное и не может быть записано в указанную область данных;

I — Ошибка в символическом имени. Эта ошибка возникает при неправильном использовании символического имени. Наиболее распространенной причиной такой ошибки является спецификация метки более чем в одной исходной строке;

N — Средство не реализовано. Эта ошибка возникает в тех случаях, когда программа ассемблера обнаруживает в исходном тексте директиву, которую она опознает, но не может обработать. Наиболее вероятной причиной такой ошибки является то, что в исходной программе используются директивы другого ассемблера:

O — Ошибка переполнения. Эта ошибка возникает в тех случаях, когда программа ассемблера обнаруживает в исходном тексте строку, содержащую слишком сложное выражение, которое нельзя обработать. Для того чтобы разрешить эту проблему, разбейте выражение на более мелкие части, или же уменьшите число операторов в данном выражении;

P — Ошибка фазы. Эта ошибка возникает в тех случаях, когда значение символического имени в исходной программе изменяется некорректным образом. Эта ошибка может также возникнуть, если в процессе ассемблирования программа обнаружит повторное определение символического имени;

R— Ошибка в имени регистра. Это сообщение означает, что в исходной строке указано имя регистра, которое не соответствует мнемоническому. Если, например, задано выражение *"POP A"*, то в результате появится сообщение об ошибке в имени регистра, поскольку в языке ассемблера для микропроцессора Intel 8080 это выражение является недопустимым;

S— Ошибочный символ. Это сообщение означает, что в поле комментариев исходной строки использован недопустимый символ (например, символ "!");

U — Неопределенное символическое имя. Эта ошибка возникает в тех случаях, когда в выражении обнаружено символическое имя с неопределенным значением. Это чаще всего получается, если имя не объявлено в тексте программы, или при объявлении были допущены ошибки, которые, по сути, привели к объявлению другого имени;

V— Ошибка в значении. Эта ошибка имеет место в тех случаях, когда программа обнаруживает выражение или операнд, которые указаны неправильно. Обычно это происходит в результате отсутствие запятой или других необходимых специальных символов.

2. Выполнение компоновки

Выполнение компоновки файла после ассемблирования пакетом MAC выполняется не совсем традиционным способом. Файл машинных команд создается неперемещаемым. Он может работать только в адресах ОЗУ, начиная с адреса 2000H. Поэтому этот процесс лучше доверить отлаженному командному файлу ***comp.bat*** (рис.4.4).

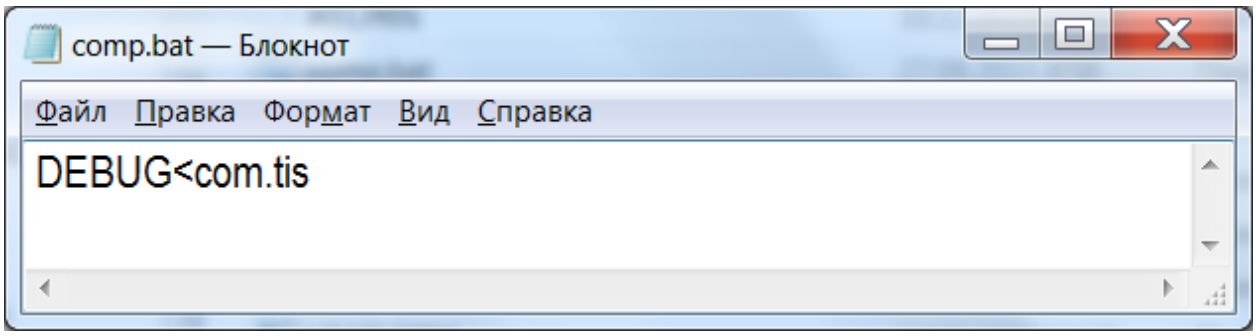


Рис.4.4. Основной файл компоновки программы

Файл ***comp.bat*** предусматривает выполнение программы отладчика DEBUG, используя в качестве источника ввода файл ***com.tis***. Значок "<" носит название оператора перенаправления команд. Он заменяет местоположение потока команд. В данном случае вместо типового источника (клавиатуры) используются данные из файла. На каждый запрос ввода программы DEBUG считывается одна строка из текста программы ***com.tis*** (рис.4.5).

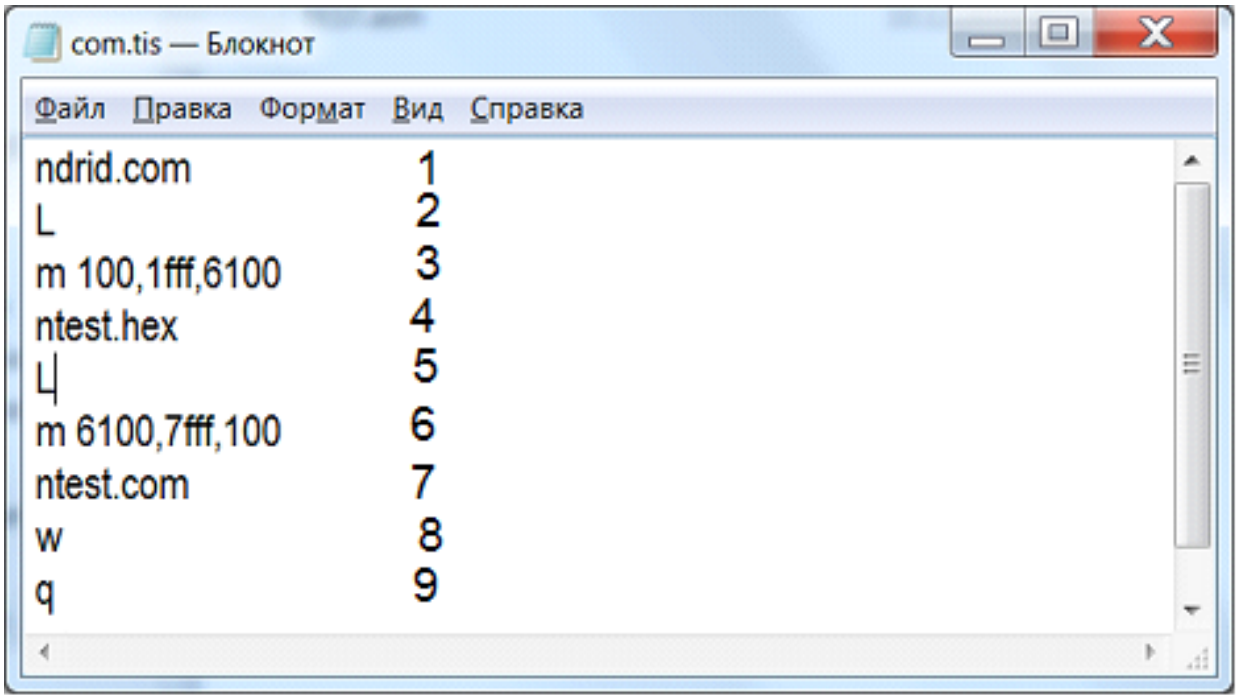


Рис.4.5. Командный файл компоновки неперемещаемых файлов и драйверов для ПЭВМ Вектор 06Ц

Параметры в файле ***com.tis***, отрабатываемые отладчиком DEBUG, имеют следующее назначение:

- ***N*** - задание имени файла для команд загрузки L и записи W. В данном примере в первой строке задается имя файла драйверов ПЭВМ ВЕКТОР 06Ц: ***drid.com***;

- L - загрузка файла в ОЗУ с именем, определенным командой ***N***, по указанному адресу или по адресу 100h. Драйверы (файл ***drid.com***) уже откомпилированы именно с адреса 100h и занимают место до адреса 1FFFh;

- ***M*** - перемещение блока памяти по указанному адресу. В третьей строке память, начиная с адреса 100h до 1FFFh, перемещается в область памяти, начиная с адреса 6100h. Эта операция временно сохраняет драйверы в свободной памяти, чтобы не повредить их при загрузке исполняемой программы;

- ***W*** - запись в файл, заданный предшествующей командой ***N***, содержимого памяти начиная с указанного адреса или с адреса 100h;

- ***Q*** - завершение сеанса работы с отладчиком.

Таким образом, после временного сохранения драйверов (строка файла №3) задается имя откомпилированного файла test.hex (строка файла №4), и он загружается в ОЗУ с адреса 100h (строка 5). Следует учесть, что файл test.com откомпилирован с директивой ORG 2000h. Это значит, что машинные команды этого файла после загрузки будут начинаться в памяти с адреса 2000h, а данные с адреса 100h до 1FFFh будут заполнены "мусором".

В строке №6 коды драйверов в полном объеме возвращаются на прежнее место, начиная с адреса 100h до адреса 1FFFh. Таким образом, после выполнения строки №6 командного файла com.tis в ОЗУ разместились коды драйверов (адреса 100h - 1FFFh) и коды пользовательской программы (адреса, начиная с 2000h до конца адресов заданных файлом test.hex).

В строке №7 задается имя файла test.com и с этим именем осуществляется запись всех данных в файл, начиная с адреса 100h (строка №8). Таким образом, имя файла test.com сохранится в папке, но под этим именем будет сохранен новый файл машинных команд, объединяющий в себе драйверы устройств и пользовательскую программу. Это вполне самостоятельная программа, готовая к выполнению с адреса 100h. Далее осуществляется выход из отладчика (строка №9).

Структура программы после компоновки представлена на рис.4.6.

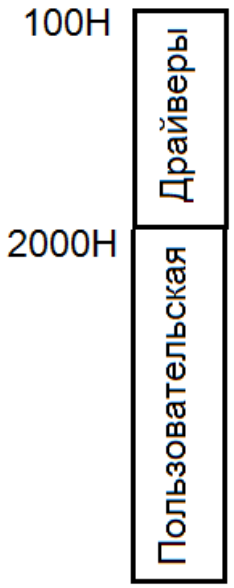


Рис.4.6. Структура программы после компоновки

3. Запуск программы на выполнение

Для запуска полученного файла ***test.com*** необходимо иметь на Рабочем столе ярлык программы ***DOSBox 0.74***. Программу ***test.com*** перетаскиванием следует перенести на ярлык DOSBox 0.74 и отпустить. При этом появится окно, представленное на рис.4.7. Имя диска ***C:*** будет ассоциировано с папкой разработчика, в которой хранится запускаемый файл ***test.com***.

Для контроля ассоциирования папки с именем диска "***C:"*** и осуществления безошибочного ввода командной строки можно выполнить команду вывода на экран содержания каталога ***DIR***. При этом должно отобразиться содержание папки разработчика (рис.4.8). Если это получилось правильно, следует набрать строку запуска программы ***test.com*** в среде эмулятора ПЭВМ ВЕКТОР 06Ц:

vector.exe test.com

После нажатия ENTER можно увидеть результаты работы разработанной программы test.com на экране монитора (рис.4.9). Программа должна реагировать на команды клавиатуры в соответствии с логикой разработанного алгоритма.

Для выхода из программы необходимо нажать ***ESC*** и в появившемся окне после приглашения "C:\>" набрать EXIT и нажать ENTER. Эмулятор завершить свою работу.

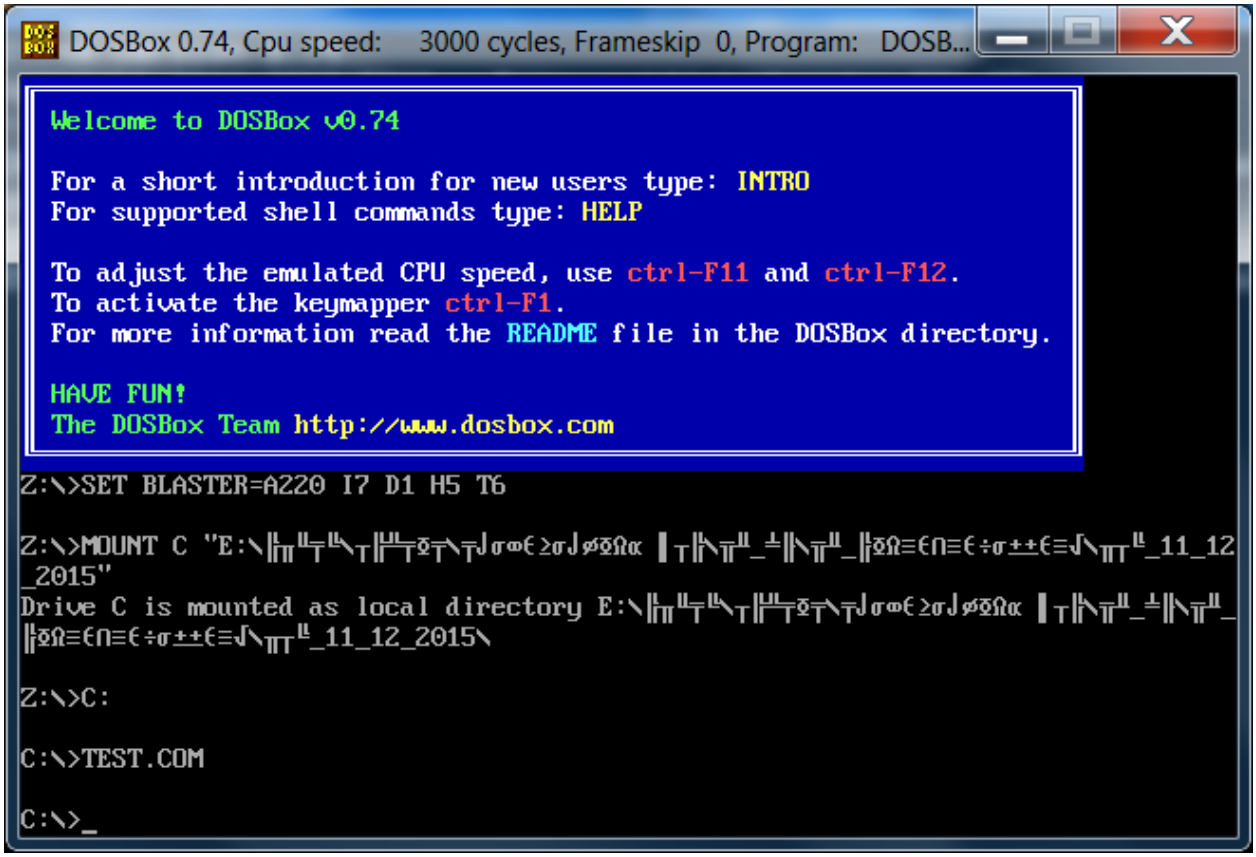


Рис.4.7. Окно перехода к командной строке программы DOSBox 0.74

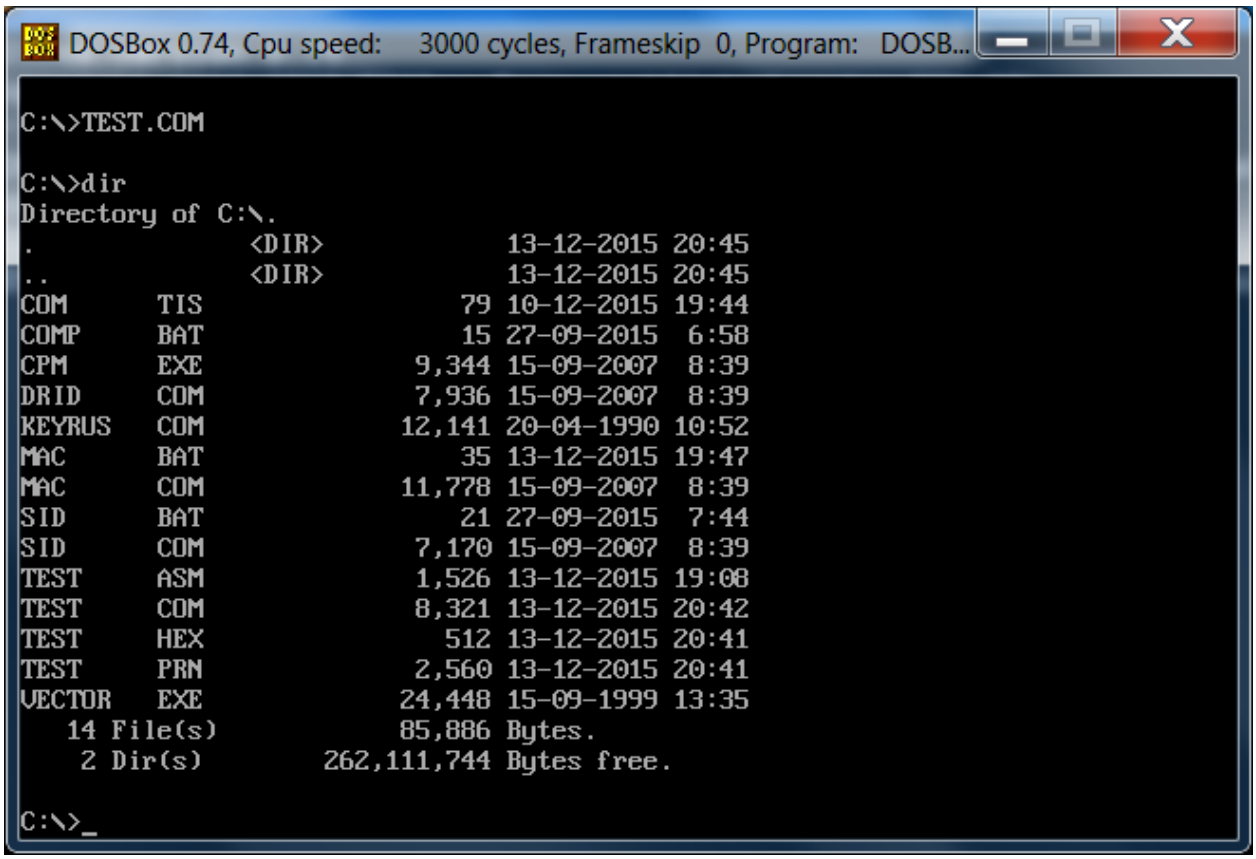


Рис.4.8. Содержимое папки разработчика и подготовка команды на запуск программы ***test.com*** на исполнение

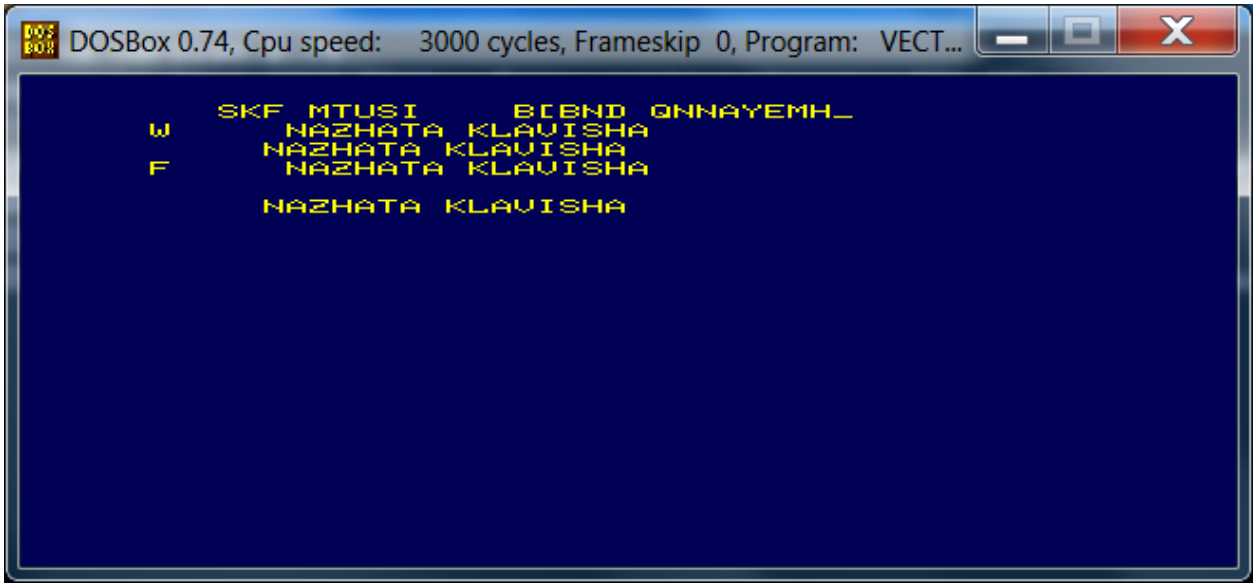


Рис.4.9. Отображение работы заготовки для программирования на экране эмулятора

4. Отладка программы

Отладка осуществляется в целях выявления и устранения логических ошибок в программе. По сути, необходимо проверить, что программа правильно выполняет все ветви алгоритма. Для этого подбираются тестовые входные данные таким образом, чтобы они вынуждали пройти по проверяемой ветви алгоритма и контролируется факт реального прохождения по этой ветви. Если проверяемые условия сложные, типа больше или равно, то проверке подлежит каждое из условий в отдельности.

Средства отладки должны предоставить стандартный набор возможностей для реализации этого процесса:

- пошаговое выполнение программы;
- просмотр и назначение значений переменных и регистров процессора;
- выполнение программы до точек останова;
- дизассемблирование программы;
- просмотр ОЗУ и изменение кодов программы и т.д.

В данной работе используется символьный отладчик SID для языка ассемблера микропроцессоров 8080 и 8085. Для загрузки отладчика следует выполнить командный файл ***sid.bat***. При этом происходит запуск отладчика без загрузки отлаживаемого файла. Появляется специальный промпт "#", что свидетельствует об ожидании ввода с клавиатуры.

Выход из программы обеспечивается нажатием ***Ctrl+C***.

Отладчик SID выполняет следующие команды:

Команда **I**

- ***I*** – (от Insert – вставить или Identify - отождествлять) - задание имени файла для команд загрузки ***R*** и записи ***W***. После этой команды имя файла вставляется в специальный буфер блока управления файлами (FCB) и может быть использовано в любое время.

Формат команды: ***ITest.com***, где Test.com – задаваемое имя. Файл должен располагаться в той же папке, что и запускаемые программы;

Команда **R**

- ***R*** – (от Read – читать) загрузка файла в ОЗУ с именем, определенным командой ***I***, по указанному адресу или по адресу 100h. Без команды ***I*** эту команду использовать недопустимо.

Форматы команды:

R – загрузка файла в ОЗУ, начиная с первой ячейки области транзитных команд 100h;

R2000 – загрузка программы с адреса 2000h. Здесь 2000 - шестнадцатеричное число, являющееся начальным адресом для загрузки файла. Программа пользователя без подключенных драйверов для рассматриваемого

примера оттранслирована именно с этого адреса. При вводе ошибочного адреса в команде R программа будет размещена не в своей области ОЗУ и в машинных командах, имеющих адресные составляющие, ссылки будут неправильные. После работы компоновщика законченная версия программы должна размещаться с адреса 100h, но начиная с этого адреса до 1FFFh размещены драйверы устройств, а коды пользователя все равно начинаются с адреса 2000h.

После загрузки в окне отобразятся данные о состоянии памяти (рис.4.10). На рисунке приведены команда задания имени и команда загрузки. Под заголовком "NEXT" записан адрес следующей свободной ячейки после последней ячейки под загруженную программу. Иными словами, это первый свободный адрес. Число под заголовком "PC" (Program Counter – счетчик команд) представляет собой текущее, установленное в программе SID, значение счетчика команд. С этого места можно выполнять программу. По этим двум данным можно определить размер файла text.com.

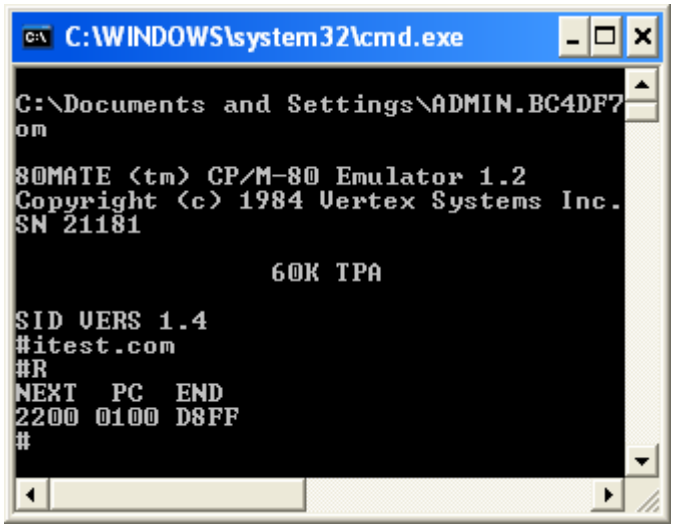


Рис.4.10. Параметры загрузки в отладчике файла Test.com

Число под заголовком "END" представляет собой последний свободный адрес в транзитной области ОЗУ.

Команда **D**

- D – (от Display – вывести) – выдача содержимого некоторой области памяти в шестнадцатеричном и ASCII форматах на экран (рис.4.11). Левая колонка представляет собой начальные адреса каждой строки в шестнадцатеричном формате. Строка (средняя часть таблицы) образована 16 шестнадцатеричными числами, которые представляют собой значения ячеек, расположенных по соответствующему адресу. Например, в первой строке сразу после адреса "2000" стоит число "31", которое представляет собой значение, хранящееся по адресу 2000h. Следующее число в первой строке – "FF", соответствует значению ячейки, расположенной по адресу 2001h и так до последнего числа в строке с адресом 200Fh. После 16 адресов вторая строка начинается с адреса 2010h. Правая часть таблицы представляет собой последовательность алфавитно-цифровых символов, соответствующих каждому коду каждой строки. Если код не попадает в диапазон рисуемых символов, то он представлен в этом поле точкой ".". Если в программе данные вводились содержательными последовательностями символов, то в этом поле таблицы они хорошо читаются. Например, "SKF MTUSI", начиная с адреса 203Ah (см. рис.4.8).

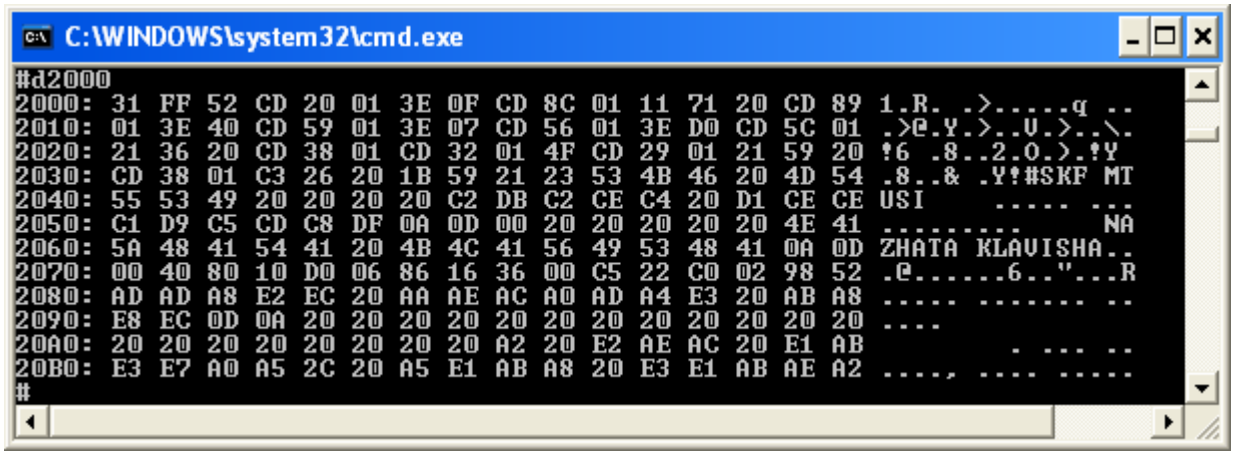


Рис.4.11. Формат вывода данных по команде отладчика D

Если открыть файл Test.prn, то легко увидеть, что коды в показанных адресах (см. рис.4.8) полностью совпадают с теми кодами и адресами, которые зафиксированы в файле печати.

Форматы команды D:

- ***D*** – выдается следующие за счетчиком команд 12 строк. Такой формат предполагает выдачу, что называется, с текущего положения счетчика команд. Повторное использование этого формате приведет к выдаче следующих 12 строк после последнего выведенного адреса;

- ***D2000*** – выдается 12 строк, начиная с указанного шестнадцатеричного адреса (в данном примере это 2000). Этот случай показан на рис.4.11;

- ***D2000,2010*** – отображаются данные только от начального адреса (2000) до конечного (2010);

- ***D,1000*** – отображаются данные от текущего значения счетчика команд до конечного адреса (1000).

Команда **L**

- ***L*** – (от List –печать) – производит дизассемблирование хранящихся в памяти кодов и вывод операторов языка ассемблера на экран. Кроме того, выводится начальный адрес размещения каждой команды. Очень важно, естественно, чтобы адрес, с которого начинается дизассемблирование, был обязательно адресом размещения кода операции какой-либо команды. В противном случае все остальное не будет соответствовать действительности.

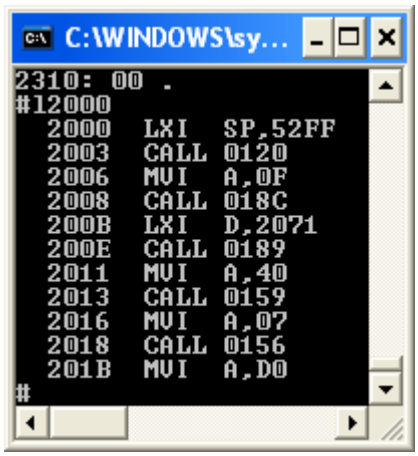


Рис.4.12. Дизассемблирование по команде L, начиная с адреса 2000h

Формат команды:

- ***L*** – выводит дизассемблированный код очередных 11 команд, начиная с текущего адреса счетчика команд;

- ***L2000*** - выводит дизассемблированный код очередных 11 команд, начиная от заданного шестнадцатеричного адреса (2000). На рис.4.12

легко заметить, что команды начинаются с адреса 2000 и совпадают с кодами файла test.prn;

- ***L2000,201F*** – дизассемблирование и вывод команд с начального шестнадцатеричного адреса (2000) до конечного шестнадцатеричного адреса (201F). Если конечный адрес диапазона не включает код последней команды целиком, то она дизассемблирована не будет.

Команда **M**

- ***M*** – (от Move – переслать) – производит пересылку указанного блока данных ОЗУ по указанному адресу. Данные перемещаются побайтно. Если блок, куда пересылаются данные, расположен после пересылаемого, то перезапись производится, начиная с конечных адресов областей памяти. Если же блок, куда пересылаются данные, расположен до пересылаемого, то перезапись производится, начиная с младших адресов областей памяти. Реально выполняется копирование и "старые данные" не исчезают. Однако при определенных перекрытиях границ областей, естественно, будет иметь место повреждение части данных по адресам исходного блока.

Формат команды:

M100,200,300

где: 100 – шестнадцатеричный адрес начала блока;

200 – шестнадцатеричный адрес конца блока;

300 – шестнадцатеричный адрес начала нового размещения. Размер блока вычисляется автоматически как разность конца и начала исходного блока.

Команда **S**

- ***S*** – (от Set – установить) – производит вывод на экран содержимого последовательно расположенных ячеек в шестнадцатеричном формате с возможностью записи нового значения также в шестнадцатеричном формате. На экране, кроме того, показывается адрес размещения выведенного байта. Если изменять значение ячейки не нужно, то нажимается ENTER, после чего в ново строке выводится содержимое следующей ячейки. Если же необходимо значение ячейки изменить, то вводится новое шестнадцатеричное значение и нажимается ENTER. Выход из режима осуществляется вводом точки "." или вводом знака, который будет воспринят ошибочным для режима ввода данных.

Формат команды:

- ***S2000*** – вывод данных в байтном формате (по одному адресу), начиная с начального шестнадцатеричного адреса (2000). В этом режиме, помимо ввода новых шестнадцатеричных чисел, можно вводить символы ASCII, если перед ними ставить двойные кавычки (");

- ***SW2000*** – вывод данных в двухбайтном (словном) формате (по два адреса сразу), начиная с начального шестнадцатеричного адреса (2000). К этому формату следует относиться крайне внимательно, потому что вывод

осуществляется двух адресов сразу, начиная с указанного адреса, при этом данные размещаются сначала из старшего адреса, а затем из младшего. Это соответствует правилам размещения двухбайтных данных в памяти при трансляции (байты как бы переворачиваются). В случае ошибки в указании начального адреса данных, отображение не будет соответствовать реальности.

Команда **A**

- ***A*** – (от Assemble –ассемблировать) – вставляет команды ассемблера в программу, начиная с заданного адреса. Однако в этой команде вводить можно числа только в шестнадцатеричном формате (четырьмя или двумя знаками в зависимости от двухбайтного или однобайтного числа соответственно) без учета формальных правил записи в файле исходного текста на ассемблере. Нельзя использовать символические имена и присваивать им значения. В случае неправильного ввода команды будет выдан символ вопроса "?". В этом случае надо повторить ввод по тому же адресу.

Формат команды:

- ***A2000*** – начать ассемблирование с начального шестнадцатеричного адреса (2000). Пример таких действий представлен на рис.4.13. После ввода команды с адресом 2000 предложено с новой строки ввести мнемонику команды. В этом примере повторены команды файла test.prn. Следующий адрес уже предложен 2003, т.к. введенная команда является трехбайтной. Выход из режима осуществляется нажатием ENTER на новой строке, не вводя символов.

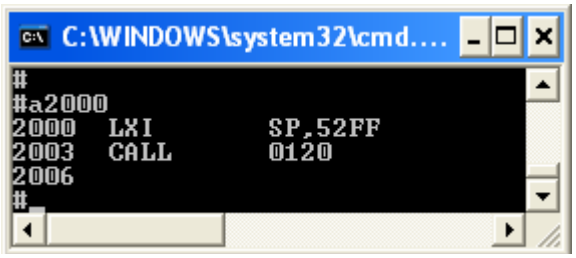


Рис.4.13. Пример использования команды отладчика A

Команда **X**

- ***X*** – отображает состояние ЦП для отладочного режима. Команда позволяет наблюдать текущие значения всех регистров и изменять состояние одного или нескольких регистров общего назначения.

Формат команды:

- ***X*** – выводит текущее состояние всех регистров процессора, т.е. его состояние полностью. Этому режиму соответствует первая длинная строка на рис.4.14.

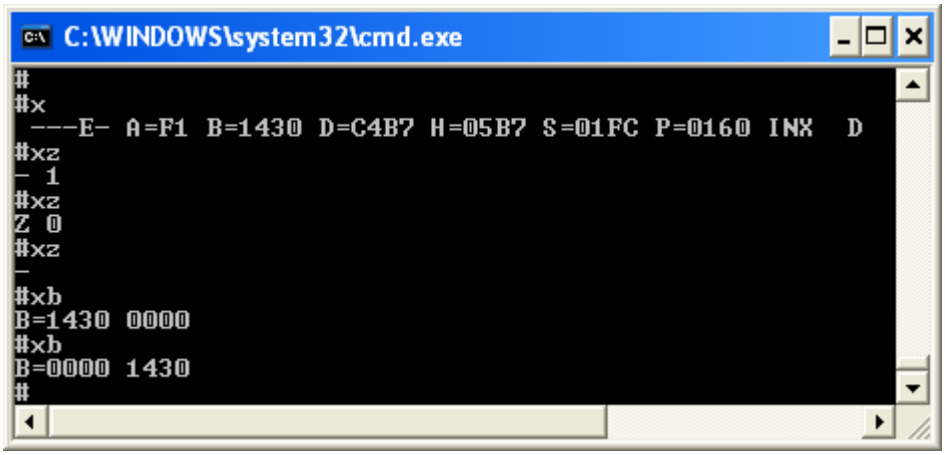


Рис.4.14. Отображение результатов выполнения команды отладчика X различных форматов

Штрихи и отдельные буквы в начале сроки отображают состояния флагов. Каждое знакоместо закреплено за своим флагом. Если выведен штрих, то флаг сброшен, если же буква – то ей соответствующий флаг установлен, т.е. равен 1. Слева направо расположены флаги: C – переноса, Z – нуля, M – минуса, E – четности, I – внутреннего переноса.

Далее в шестнадцатеричном формате расположены коды, находящиеся в регистрах:

A – аккумуляторе;
B – регистровой паре BC;
D – регистровой паре DE;
H – регистровой паре HL;
S – указателе стека SP;
P – счетчике команд PC.
Замыкает строку дизассемблированная мнемоника команды;

X"флаг" – позволяет наблюдать конкретный флаг и при необходимости его изменить. Вместо обозначения "флаг" может быть установлена одна из букв, соответствующих обозначению конкретного флага: C, Z, M, E, I. На рис.4.11 показан пример наблюдения флага Z: его исходное значение равно 0, затем он изменяется на 1, что подтверждается при очередном просмотре, а затем опять изменяется на 0. Если флаг не подлежит изменению, то следует просто нажать ENTER;

X"регистр" - позволяет аналогично предыдущему формату наблюдать и изменять значение регистров процессора. Отличие заключается в том, что вместо параметра "регистр" в этом формате должен быть установлен допустимый символ имени регистра: A, B, D, H, S или P. Такой пример показан на рис.4.11: значение регистров BC заменено на 0000, а затем вновь на первоначальное значение. Следует помнить, что аккумулятор является однобайтным, а остальные отображаемые имена регистров – двухбайтными. Если регистр не подлежит изменению, то следует просто нажать ENTER.

Команда **T**

- T – (от Trace – трассировать) – выполняет команды машины, размещенные в памяти, с отображение состояния регистров процессора.

Форматы команды:

-T – выполняет одну машинную команду (пошаговое выполнение программы), расположенную по адресу, который соответствует текущему счетчику команд. Счетчик команд PC можно установить командой "X". В отображаемой строке состояние регистров соответствует времени до выполнения команды, которая представлена мнемоникой в этой строке. Состояние счетчика команд PC в строке является адресом этой команды. Это легко проследить на рис.4.15. Легко видно, что вторя строка регистров, как и первая отображает неизменное состояние регистровой пары HL. Во второй строке указана команда инкремента HL, после выполнения которой состояние регистровой пары HL реально изменилось. После второй строки отобразился адрес со звездочкой "*" - *0162: это адрес следующей команды. Адрес изменился на 1, потому что команда "INX H" занимает 1 байт (1 адрес). Аналогично в третьей строке выполнилась команда "JMP 0158", расположенная по адресу 0162. После ее выполнения адрес следующей команды обозначен как *0158.

- Tn – выполняет сразу n (число шестнадцатеричное) команд, начиная с текущего адреса с отображением состояния процессора для каждой команды. На экране подряд будут отображены n строк. Для выхода из режима достаточно нажать любую клавишу в любой момент вывода.

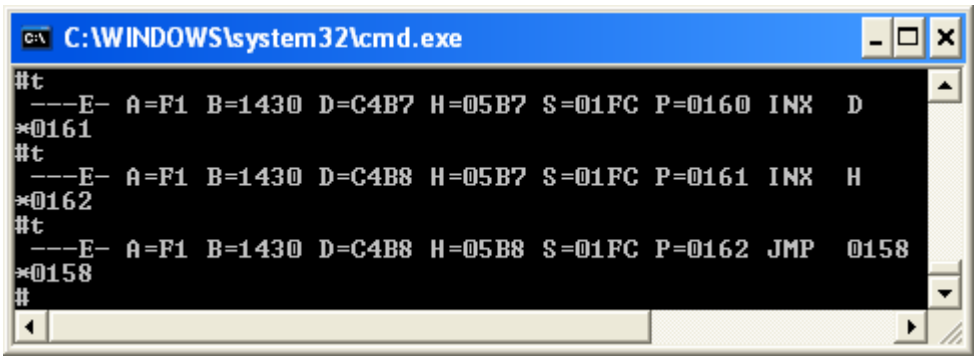


Рис.4.15. Пошаговое выполнение программы с помощью команды отладчика T

Использование приведенных команд позволяет реализовать все типовые режимы работы отладчика. При использовании команд необходимо внимательно контролировать выполняемые операции, четко понимать в каких адресах памяти выполняются операции, отдавать себе отчет о последствиях выполняемых изменений кодов. Очень удобно при использовании отладчика опираться на файл печати (расширение .prn).

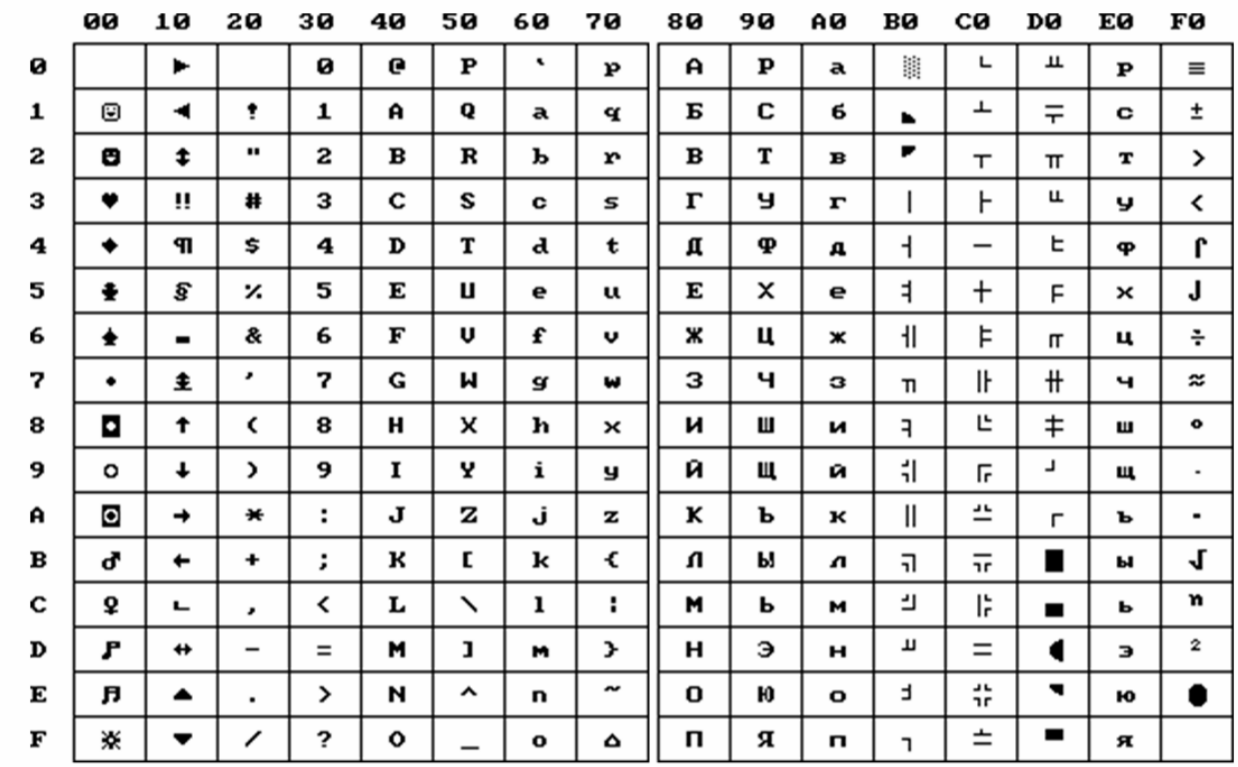
Вопросы для самопроверки

1. В каком формате вводятся данные с клавиатуры?
2. Какие коды имеют числовые символы?
3. Как из кода символа получить число, ему соответствующее?
4. Как из одноразрядного числа получить код, ему соответствующий?
5. Сформулируйте алгоритм преобразования кода символов многоразрядного числа в двоичное число?
6. Каковы задачи компоновщика?
7. Какова структура файла типа .COM?
8. Какие данные хранятся в файле типа .PRN?
9. Что находится в файле типа .SYM?
10. Каковы задачи компилятора?
11. Что такое "символическое имя"? Как оно записывается в исходном тексте и чему соответствует в исполняемом файле?
12. Каковы требования к текстовому редактору системы программирования?
13. Какие задачи выполняет символьный отладчик?
14. Что такое "точка останова"?
15. Какова методика отладки программы?
16. Можно ли отладить программу без символьного отладчика?
17. В файле с каким расширением предполагается настройка адресов? В чем она заключается?
18. Как программно различить коды символов букв от символов чисел?

ПРИЛОЖЕНИЕ А

Таблица ASCII символов

ASCII *– American Standard Code for Information Interchange*, что в переводе с английского означает "американский стандартный код для обмена информацией". Таблица ASCII представляет собой перечень букв русского и латинского алфавита, различных знаков и управляющих символов.



Каждый символ в таблице стоит на пересечении вертикали (шестнадцатеричная цифра старшего полубайта с нулем в позиции младшего полубайта) и горизонтали (шестнадцатеричная цифра младшего полубайта). Значение кода на пересечении получается сложением этих кодов (или присоединением к цифре старшего полубайта справа цифры младшего полубайта). Например, заглавный латинский символ "A" имеет код, образуемый верхней координатой – 40h и левой (горизонтальной) координатой – 1. Таким образом, код символа "A" - 41h. Этот код еще называют по отношению к этой таблице ***номером символа***.

Реакция машины на тот или иной код определяется тем программным пакетом, с которым работают в настоящий момент. Выдача в адрес стандартного устройства вывода (монитора) управляющих кодов, например, может сопровождаться их отработкой в соответствии с предназначением или отображением на мониторе их индивидуального значка (см. таблицу), либо отображением всех управляющих кодов единым значком пустого прямоугольника, точкой, пустым символом или другим образом. Отдельные управляю-

щие коды почти всегда отрабатываются для сохранения элементарных возможностей управления монитором: 07h, 0Ah, 0Dh, но это не общее правило.

Следует отметить, что даже коды буквенных символов отрабатываются привычным образом только в текстовых редакторах – по ним рисуются на мониторе соответствующие коду символы. Однако, например, для графических пакетов реакция может быть иной и соответствовать контексту выполняемых в пакете операций. В этом случае клавиши приобретают смысл, который принято связывать с понятием "горячие клавиши".

ПРИЛОЖЕНИЕ Б

Управляющие символы

Поскольку ASCII изначально предназначался для обмена информацией (по телетайпу), в нём, кроме информационных символов, используются символы-команды для управления связью (коды от 00h до 1Fh). Это обычный набор спецсигналов, применявшийся и в других докомпьютерных средствах обмена сообщениями (азбука Морзе, семафорная азбука), дополненный с учётом специфики устройства. Расшифровка этих кодов следующая:

Dec	Hex	Char	Cmd
0	00		NUL
1	01	☺	SOH
2	02	☻	STX
3	03	♥	ETX
4	04	♦	EOT
5	05	♣	ENQ
6	06	♠	ACK
7	07	•	BEL
8	08	◘	BS
9	09	○	TAB
10	0A	◙	LF
11	0B	♂	VT
12	0C	♀	FF
13	0D	♪	CR
14	0E	♫	SO
15	0F	☼	SI
16	10	►	DLE
17	11	◄	DC1
18	12	↕	DC2
19	13	‼	DC3
20	14	¶	DC4
21	15	§	NAK
22	16	▬	SYN
23	17	↨	ETB
24	18	↑	CAN
25	19	↓	EM
26	1A	→	SUB
27	1B	←	ESC
28	1C	∟	FS
29	1D	↔	GS
30	1E	▲	RS
31	1F	▼	US

- **NUL**, 00h— Null, пустой. Всегда игнорировался. На перфолентах 1 представлялась дырочкой, 0 — отсутствием дырочки. Поэтому пустые части перфоленты до начала и после конца сообщения состояли из таких символов. Сейчас используется во многих языках программирования как конец строки. (Строка понимается как последовательность символов.) В некоторых операционных системах NUL — последний символ любого текстового файла;
- **SOH**, 01h — Start Of Heading, начало заголовка;
- **STX,** 02h — Start of Text, начало текста. Текстом называлась часть сообщения, предназначенная для печати. Адрес, контрольная сумма и т. д. входили или в заголовок, или в часть сообщения после текста;
- **ETX,** 03h — End of Text, конец текста. Здесь телетайп прекращал печатать. Использование символа Ctrl-C, имеющего код 03h, для прекращения работы чего-то (обычно программы), восходит ещё к тем временам;
- **EOT,** 04h— End of Transmission, конец передачи. В системе UNIX Ctrl-D, имеющий тот же код, означает конец файла при вводе с клавиатуры;
- **ENQ**, 05h— Enquire. Прошу подтверждения;
- **ACK**, 06h — Acknowledgement. Подтверждаю;
- **BEL,** 07h — Bell, звонок, звуковой сигнал. Сейчас тоже используется. В языках программирования C и C++ обозначается \a;
- **BS,** 08h — Backspace, возврат на один символ. Сейчас стирает предыдущий символ;

- **TAB,** 09h — Tabulation. Обозначался также HT — Horizontal Tabulation, горизонтальная табуляция. Во многих языках программирования обозначается \t;
- **LF,** 0Ah — Line Feed, перевод строки. Сейчас в конце каждой строчки текстового файла ставится либо этот символ, либо CR, либо и тот и другой (CR, затем LF), в зависимости от операционной системы. Во многих языках программирования обозначается \n и при выводе текста приводит к переводу строки;
- **VT**, 0Bh — Vertical Tab, вертикальная табуляция;
- **FF**, 0Ch — Form Feed, новая страница;
- **CR**, 0Dh — Carriage Return, возврат каретки. Во многих языках программирования этот символ, обозначаемый \r, можно использовать для возврата в начало строчки без перевода строки. В некоторых операционных системах этот же символ, обозначаемый Ctrl-M, ставится в конце каждой строчки текстового файла перед LF;
- **SO**, 0Eh — Shift Out, измени цвет ленты (использовался для двуцветных лент; цвет менялся обычно на красный). В дальнейшем обозначал начало использования национальной кодировки;
- **SI,** 0Fh — Shift In, обратно к Shift Out;
- **DLE**, 10h — Data Link Escape, следующие символы имеют специальный смысл;
- **DC1**, 11h — Device Control 1, 1-й символ управления устройством — включить устройство чтения перфоленты;
- **DC2**, 12h — Device Control 2, 2-й символ управления устройством — включить перфоратор;
- **DC3**, 13h— Device Control 3, 3-й символ управления устройством — выключить устройство чтения перфоленты;
- **DC4**, 14h — Device Control 4, 4-й символ управления устройством — выключить перфоратор;
- **NAK**, 15h — Negative Acknowledgment, не подтверждаю. Обратно к Acknowledgment;
- **SYN**, 16h — Synchronization. Этот символ передавался, когда для синхронизации было необходимо что-нибудь передать;
- **ETB**, 17h — End of Text Block, конец текстового блока. Иногда текст по техническим причинам разбивался на блоки;
- **CAN**, 18h — Cancel, отмена (того, что было передано ранее);
- **EM**, 19h — End of Medium, кончилась перфолента и т. д.;
- **SUB**, 1Ah — Substitute, подставить. Ставится на месте символа, значение которого было потеряно или испорчено при передаче. Сейчас Ctrl-Z используется как конец файла при вводе с клавиатуры в системах DOS и Windows. У этой функции нет никакой очевидной связи с символом SUB;
- **ESC**, 1Bh — Escape. Следующие символы — что-то специальное;
- **FS**, 1Ch — File Separator, разделитель файлов;
- **GS**, 1Dh — Group Separator, разделитель групп;

- **RS**, 1Eh — Record Separator, разделитель записей;
- **US**, 1Fh — Unit Separator, разделитель юнитов. То есть поддерживалось 4 уровня структуризации данных: сообщение могло состоять из файлов, файлы из групп, группы из записей, записи из юнитов;
- **DEL**, 7Fh — Delete, стереть последний символ. Символом DEL, состоящим в двоичном коде из всех единиц, можно было забить любой символ. Устройства и программы игнорировали DEL так же, как NUL. Код этого символа происходит из первых текстовых процессоров с памятью на перфоленте: в них удаление символа происходило забиванием его кода дырочками (обозначавшими логические единицы.

Ввод ASCII символов с клавиатуры

ASCII коды, для которых отсутствуют клавиши, можно вводить прямо с клавиатуры. Это часто требуется при разработке управляющих файлов. Все, что для этого нужно - это расширенная клавиатура. На клавиатурах настольных компьютеров это цифровая панель справа. На ноутбуках, как правило, нужно воспользоваться клавишей Fn (или Fn вместе с NumLk).

Для ввода кода нажмите ***Alt,*** затем введите **десятичный** ASCII код символа на расширенной цифровой панели и отпустите ***Alt***. Например, 1 с нажатым ***Alt*** введет смайлик ☺, 3 с нажатым ***Alt*** введет сердечко ♥ и т.д. (см. таблицу управляющих кодов в таблице ASCII).

ПРИЛОЖЕНИЕ В

Справка по ассемблеру микропроцессора КР580

ДИРЕКТИВЫ АССЕМБЛЕРА

Директива **EQU**

Назначение. Задать значение символическому имени. Ассемблер присваивает имени значение выражения и использует его при ассемблировании везде, где встречается имя. Позволяет повысить читабельность программы и эффективно изменяет значения выражения в случае необходимости. Выражения не должны совпадать с мнемокодами команд языка и его директивами.

Формат. Имя EQU выражение ;приравнять имя

Пример.

```
Nachalo  EQU  100h      ;Имя Nachalo будет соответствоать числу
                        ;100h
```

Директива **ORG**

Назначение. Установить начальное значение счетчика команд. С этого адреса начнется ассемблирование последующих команд и, следовательно, должно быть выполнено размещение машинных команд в памяти для выполнения.

Формат. ORG выражение ;выражение = число, символическое имя
;или выражение

Пример.

```
ORG  100h       ;адрес первой команду будет 100h
ORG  Nachalo    ;адрес первой команды будет равен числовому
                ;значению имени Nachalo
ORG  4+7        ;адрес первой команды будет равен результату
                ;выражения, т.е.7
```

Директива **DB**

Назначение. Задать байт данных (Define Byte - побайтное описание данных). Восьмибитовые значения, соответствующие списку данных, будут последовательно размещены в памяти, начиная с байта, адресованного "именем". При отсутствии "имени" они будут размещены за теми ячей-

ками памяти, которые занимают предыдущие команды или данные в тексте программы. "Список" может быть образован числами, именами и выражениями, которые соответствуют восьмибитным значениям данных, либо знаками ASCII, заключенными в апострофы. Элементы списка разделяются запятыми.

Формат. Имя: DB список ;задать байт данных

Пример.

SPISOK1: DB 5h,'TEXT',65,'Я',4+7

Этот пример потребует 8 ячеек ОЗУ. За ними будут размещены следующие инструкции программы.

Директива **DS**

Назначение. Зарезервировать область памяти (Define Storage - определить область). Память резервируется в байтах, в объеме, соответствующем выражению. "Выражением" является число, имя или выражение. Выделяемая область никак не инициализируется, и о ее содержании должен позаботиться программист. За выделенными ячейками будут размещены следующие инструкции программы.

Формат. Имя: DS Выражение ;задать объем памяти

Пример.

LABEL: DS 20 ;Резервируется 20 ячеек памяти (20 байт)

Директива **DW**

Назначение. Задать слово данных (Define Word - определить слово). Двухбайтные элементы (слова), соответствующие списку данных, будут последовательно размещены в памяти, начиная с байта, адресованного "именем". При отсутствии "имени" они будут размещены за теми ячейками памяти, которые занимают предыдущие команды в тексте программы. "Список" может быть образован числами, именами и выражениями, которые соответствуют двухбайтным значениям данных. Элементы списка разделяются запятыми.

Формат. Имя: DW список ;задать слово данных

Пример.

SPISOK2: DW 27A7h,0A5h,65

Этот пример потребует 6 ячеек ОЗУ. За ними будут размещены следующие инструкции программы. Запоминание двухбайтных слов осуществляется как всегда: сначала размещается младший байт, затем - старший, т.е. перевернутым образом.

Директива **END**

Назначение. Конец исходной записи. Директива сообщает ассемблеру об окончании одной физической записи исходной программы и следует закончить генерацию кодов. Программа может состоять их нескольких записей, каждая из которых должна заканчиваться этой директивой. Указание имени не обязательно.

Формат. Имя: END ;Конец записи

Пример.

```
ZAPIC1:  END
```

КОМАНДЫ АССЕМБЛЕРА

Мнемоника команды	Циклы(такты)	Байты	Влияние на признаки

Команда **ACI D8** 2(7) 2 байта На все признаки

Назначение. Сложение с константой и переносом (Add Immediate with carry). Байт данных, заданный непосредственно вторым байтом команды складывается с аккумулятором и флагом переноса CR. Результат сохраняется в аккумуляторе. Сложение осуществляется в дополнительных кодах.

Формат. ACI D8

Пример. Пусть D8=42h, A=14h, CR=1

```
ACI  42h   ; выполнение команды обеспечит сложение
           ;  A= 14h=  00010100b
           ;  D8=42h= 01000010b
           ;  CR=1h=        1b
           ;  A= 57h=  01010111b. Флаг нуля и переноса сбросится
           ; Z=0, CR=0
```

Команда **ADC R** 1-2(4-7) 1 байт На все признаки

Назначение. Сложение с переносом (Add with Carry). Байт в указанном регистре складывается с аккумулятором и значением флага переноса CR. Результат сохраняется в аккумуляторе. Используется для сложения многобайтных слов. Сначала складываются младшие байты с помощью команды ADD, а затем последующие байты с учетом переноса CR с помощью команды ADC.

Формат. ADC R ; R=A, B, C, D, E, H, L, M(HL)
Пример. Пусть B=25h, A=0ABh, CR=1
ADC B ; выполнение команды обеспечит сложение
; B= 25h= 00100101b
; A=0ABh=10101011b
; CR=1=.....1b
; A=0D1h=11010001b. Флаг нуля и переноса сбросятся
; Z=0 и CR=0

Команда **ADD R** 1-2(4-7) 1 байт На все признаки

Назначение. Сложение (Add - сложить или Addition - сложение). Байт в указанном регистре складывается с аккумулятором. Результат сохраняется в аккумуляторе. Сложение осуществляется в дополнительных кодах.
Формат. ADD R ; R=A, B, C, D, E, H, L, M(HL)
Пример. Пусть B=65h, A=0ABh,
ADD B ; выполнение команды обеспечит сложение
; B= 65h= 01100101b
; A=0ABh=10101011b
; A=10h= 00010000b. Флаг нуля сбросится Z=0, а флаг
; переноса установится CR=1

Команда **ADI D8** 2(7) 2 байта На все признаки

Назначение. Сложение с константой (Add Immediate). Байт данных, заданный непосредственно вторым байтом команды складывается с аккумулятором. Результат сохраняется в аккумуляторе. Сложение осуществляется в дополнительных кодах.
Формат. ADI D8
Пример. Пусть D8=42h, A=14h,
ADI 42h ; выполнение команды обеспечит сложение
; A= 14h= 00010100b
; D8=42h= 01000010b
; A= 56h= 01010110b. Флаг нуля и переноса сбросится
; Z=0, CR=0

Команда **ANA R** 1-2(4-7) 1 байт На все признаки

Назначение. Логическое умножение (And). Байт в указанном регистре поразрядно умножается логически с аккумулятором. Результат сохра-

няется в аккумуляторе. Команда часто используется для обнуления группы битов

Формат. ANA R ; R=A, B, C, D, E, H, L, M(HL)

Пример. Пусть B=25h, A=0ABh, тогда

```
ANA  B   ;выполнение команды обеспечит поразрядное логическое И
         ;  B= 25h=  00100101b
         ;  A=0ABh=10101011b
         ;  A=.21h = 00100001b
         ; Флаг нуля и переноса сбросятся Z=0 и CR=0
```

Команда	**ANI D8**	2(7)	2 байт	На все признаки

Назначение. Логическое умножение на константу (And Immediate). Байт, находящийся во втором байте команды поразрядно умножается логически на аккумулятор. Результат сохраняется в аккумуляторе. Команда часто используется для обнуления группы битов.

Формат. ANI D8

Пример. Пусть D8=25h, A=0ABh. Тогда

```
ANI  25h ;выполнение команды обеспечит поразрядное логическое И
         ;  D8=25h= 00100101b
         ;  A=0ABh=10101011b
         ;  A=.21h = 00100001b
         ; Флаг нуля и переноса сбросятся Z=0 и CR=0
```

Команда	**CALL D16**	5(17)	3 байта	Не изменяет

Назначение. Безусловный вызов (Call) подпрограммы. Сохранить адрес следующей команды за командой CALL в стеке и передать управление команде, чей адрес находится во втором и третьем байтах команды CALL (занести адрес, находящийся во втором и третьем байте команды, в счетчик команд PC).

Формат. CALL D16

Пример. Пусть D16=2AF1h=PODPR, PC=2742h, SP=0FFFFh. Тогда

```
CALL  2AF1h   ; после выполнения команды PC= 2AF1h
              ; SP=0FFFDh, в стеке сохранится число 2745h
CALL  PODPR   ; изменения аналогичные предыдущему примеру
```

Команда	**C-CON D16**	3-5(11-17)	3 байта	Не изменяет

Назначение. Условный вызов (Condition Call) подпрограммы. Если условие истинно, то действия определяются выполнение команды CALL.

В противном случае продолжается выполнение программы (к текущему содержимому счетчика команд прибавляется длина команды CALL - 3 байта). Возможные условия перехода аналогичны таблице для команды J-CON (приложение Е).

Формат. C-CON D16

Пример. CZ 234AH ;Вызов подпрограммы по адресу 234AH
;при наличии признака Z.

Команда **CMA** 1(4) 1 байт Не изменяет

Назначение. Инверсия аккумулятора (Complement Accumulator). Каждый бит аккумулятора инвертируется.

Формат. CMA

Пример. Пусть A=2Ah=00111010b, тогда
CMA ;после выполнения команды A=0C5h=11000101b

Команда **CMC** 1(4) 1 байт На CR

Назначение. Инверсия флага переноса (Complement Carry). Если. CR=0, то установить CR=1. Если CR=1, то сбросить CR=0.

Формат. CMC

Пример. Пусть CR=1, тогда
CMC ;после выполнения команды CR=0

Команда **CMP R** 1-2(4-7) 1 байт На все признаки

Назначение. Сравнение (Compare) с регистром. Байт в указанном регистре сравнивается с содержимым аккумулятора. Сравнение производится вычитанием регистра из аккумулятора с установкой всех признаков. Вычитание производится по правилам операции вычитания SUB. Но содержимое аккумулятора и регистра не изменяются.

Формат. CMP R ; R=A, B, C, D, E, H, L, M(HL)

Пример. Пусть B=25h, A=6Bh, тогда
CMP B ; выполнение команды обеспечит следующие действия:
; Доп(B)= 11011011b=0DBh
; A=6Bh= 01101011b
; A=46h= 01000110b. Флаг нуля и переноса сбросятся
; Следовательно B<A

Другие сочетания признаков дают другие результаты сравнения:
- при Z=1 сравниваемый регистр и аккумулятор равны;

- при Z=0 и CR=1 - регистр больше аккумулятора;

- при CR=0 и неучете признака Z можно говорить о том, что аккумулятор больше или равен регистру.

На основании анализа флагов, полученных после операции сравнения (или полученных в результате других операций) осуществляется принятие решения в условных командах.

Команда **CPI D8** 2(7) 2 байта На все признаки

Назначение. Сравнение с константой (Compare Immediate). Константа, находящийся непосредственно во втором байте команды, сравнивается с содержимым аккумулятора. Сравнение производится вычитанием константы из аккумулятора с установкой всех признаков. Вычитание производится по правилам операции вычитания SUB. Но содержимое аккумулятора не изменяются.

Формат. CPI D8

Пример. Пусть D8=25h, A=6Bh, тогда

```
CPI  25h  ; выполнение команды обеспечит следующие действия:
          ; Доп(D8)=  11011011b=0DBh
          ; A=6Bh=    01101011b
          ;  A=46h=   01000110b. Флаг нуля и переноса сбросятся
          ; Следовательно 25h<
```

Другие сочетания признаков дают другие результаты сравнения, см. команду CMP R. Команда используется для установки признаков командах условного перехода.

Команда **DAD YZ** 3(10) 1 байт Только CR

Назначение. Сложение пары регистров с HL (Add register pair to H and L). Указанная пара регистров как единое шестнадцатеричное число складывается с парой HL. Результат сохраняется в паре HL.

Формат. DAD YZ ; YZ= BC, DE, HL, SP

Пример. Пусть DE=339Fh, HL=0A17Bh

```
DAD  D    ; выполнение команды обеспечит сложение
          ;  DE=  339Fh
          ;  HL=0A17Bh
          ;  HL=0D51Ah
          ; флаги нуля Z=0 и переноса CR=0
```

Команда **DCR R** 1-3(5-10) 1 байт Кроме CR

Назначение. Декремент содержимого регистра или ячейки памяти (Decrement).
Формат. DCR R ;R=A, B, C, D, E, H, L, M(HL)
Пример. Пусть (C)=89h, тогда
DCR C ;после выполнения команды C=88h
DCR M ;после выполнения команды содержимое ячейки памяти,
;адрес которой находится в регистровой паре HL будет
;уменьшено на 1

Команда	**DCX YZ**	1(5)	1 байт	Не изменяет

Назначение. Декремент пары регистров (Decrement register pair). Содержимое пары указанных регистров уменьшается на единицу..
Формат. DCX YZ ; YZ=BC, DE, HL, SP
Пример. Пусть DE=2AFFh, тогда
DCX D ;после выполнения команды DE=2AFEh

Команда	**DI**	1(4)	1 байт	Не изменяет

Назначение. Запрет прерываний (Disable Interrupt). Команда устанавливает в 0 триггер прерываний, запрещая тем самым прерывания процессора.
Формат. DI

Команда	**EI**	1(4)	1 байт	Не изменяет

Назначение. Разрешение прерываний (Enable Interrupt). Команда устанавливает в 1 триггер прерываний, разрешая тем самым прерывания процессора.
Формат. EI

Команда	**HLT**	1(7)	1 байт	Не изменяет

Назначение. Останов (Halt). Содержимое счетчика команд PC увеличивается на единицу. Процессор останавливается. Вывести его из этого состояния возможно только аппаратным сигналом прерывания.
Формат. HLT

Команда **IN N** 3(10) 2 байта Не изменяет

Назначение. Ввод данных с внешнего порта (Input). Один байт данных вводится в аккумулятор из внешнего порта с номером, указанным во втором байте команды.

Формат. IN N ; N=0 - 255

Пример. Пусть N=23, данные порта с номером 23 равны 39h. Тогда

```
IN   23   ; после выполнения команды A=39h
```

Команда **INR R** 1-3(5-10) 1 байт Кроме CR

Назначение. Инкремент содержимого регистра или ячейки памяти (Increment).

Формат. DCR R ;R=A, B, C, D, E, H, L, M(HL)

Пример. Пусть (C)=89h, тогда

```
INR  C    ;после выполнения команды C=8Ah
INR  M    ;после выполнения команды содержимое ячейки памяти,
          ;адрес которой находится в регистровой паре HL будет
          ;увеличено на 1
```

Команда **INX YZ** 1(5) 1 байт Не изменяет

Назначение. Инкремент пары регистров (Increment register pair). Содержимое пары указанных регистров увеличивается на единицу..

Формат. INX YZ ; YZ=BC, DE, HL, SP

Пример. Пусть DE=2AFFh, тогда

```
INX  D    ;после выполнения команды DE=2B00h
```

Команда **JMP D16** 3(10) 3 байта Не изменяет

Назначение. Безусловный переход (Jump) по адресу. Передать управление команде, чей адрес находится во втором и третьем байтах команды. Выполняется команда путем переноса второго байта команды в младший байт счетчика команд, а третьего байта команды - в старший байт счетчика команд.

Формат. JMP D16

Пример. Пусть D16=2AF1h. Тогда

```
JMP  2AF1h    ; после выполнения команды PC= 2AF1h
              ; Это означает, что следующая команда
              ; будет извлекаться по адресу ОЗУ 2AF1h
```

Команда **J-CON D16** 3(10) 3 байта Не изменяет

Назначение. Условный переход (Conditional Jump) по адресу. Если условие истинно, то адрес перехода по программе определяется содержимым второго и третьего байта команды (этот адрес заносится в счетчик команд PC). В противном случае продолжается выполнение программы со следующего за этой командой адреса (к счетчику команд PC прибавляется длина команды - 3 байта).

Формат. J-CON D16 ; CON - признак, либо его отрицание.

Пример. Пусть D16=2AF1h=PEREHOD, признак Z=0, PC=1247h. Тогда

```
JZ   2AF1h      ; после выполнения команды PC= 124Ah
JZ   PEREHOD    ; после выполнения команды PC= 124Ah
JNZ  2AF1h      ; после выполнения команды PC= 2AF1h
```

Команды условной передачи управления МП КР580ВА580 сведены в таблицу (см. приложение Е). В таблице аббревиатура ADR обозначает двухбайтный адрес перехода (D16).

Команда **LDA D16** 4(13) 3 байт Не изменяет

Назначение. Загрузка в аккумулятора из память по прямому адресу (Load Accumulator direct). Содержимое ячейки ОЗУ, адрес которой задан вторым и третьим байтом команды, переслать в аккумулятор.

Формат. LDA D16

Пример. Пусть D16=025ABh, M(025ABh)=27h, тогда

```
LDA  025ABh   ;после выполнения команды содержимое ячейки ОЗУ
              ; с адресом 25ABh запишется в аккумулятор:  A= 27h
```

Команда **LDAX YZ** 2(7) 1 байт Не изменяет

Назначение. Загрузка из памяти в аккумулятор (Load Accumulator indirect). Из ячейки памяти, адрес которой находится в заданной регистровой паре (BC или DE) данные переписываются в аккумулятор. Команда использует только две пары регистров. Третья пара (HL) используется командой MOV.

Формат. LDAX YZ ; YZ=BC, DE

Пример. Пусть B=25h, C=0ABh, в памяти с адресом 25ABh хранится байт 2Ah

```
LDAX   B      ;после выполнения команды байт из ячейка ОЗУ с
               ;адресом 25ABh пересылается в аккумулятор: A=2Ah
```

Команда **LHLD D16** 5(16) 3 байт Не изменяет

Назначение. Запись из памяти в пару HL (Load H and L Direct) по прямому адресу. Содержимое из ячейки ОЗУ, адрес которой задан вторым и третьим байтом команды передать в регистр L. Из ячейки с последующим адресом содержимое передать в регистр H (байты из ячеек памяти записываются наоборот: сначала в младший регистр L, затем - в старший регистр H).

Формат. LHLD D16 ; только для HL

Пример. Пусть D16=025ABh, M(025ABh)=3Ah, M(025ACh)=27h, тогда

```
LHLD  025ABh  ;после выполнения команды L=3Ah, H=27h
```

Команда **LXI YZ,D16** 3(10) 3 байта Не изменяет

Назначение. Загрузка пары регистров константой (Load register pair Immediate). Константа находится непосредственно во втором и третьем байте самой команды. В соответствии с IBM-овским подходом старший и младшие байты шестнадцатиразрядной константы в памяти хранятся в обратном порядке. Содержимое третьего байта команды загружается в старший регистр указанной пары, содержимое второго байта команды загружается в младший регистр указанной пары.

Формат. LXI YZ,D16 ; YZ=BC, DE, HL, SP

Пример. Пусть D16=2AF1h. Тогда

```
LXI  B,2AF1h    ; после выполнения команды BC= 2AF1h
```

Команда **MOV R1,R** 1-2(5-7) 1 байт Не изменяет

Назначение. Пересылка байта (Move). Байт из регистра R пересылается в регистр R1. Состояние регистра-источника не изменяется. Количество циклов и тактов увеличивается при выполнении пересылки с памятью M(HL).

Формат. MOV R1,R ; R=A, B, C, D, E, H, L, M(HL)

Пример. Пусть A=2Ah, H=25h, L=0ABh тогда

```
MOV   B,A       ;после выполнения команды A=2Ah, B=2Ah
MOV   M,A       ; после выполнения команды A=2Ah, ячейка памяти
                ; ОЗУс адресом, равным содержимому регистровой
                ; пары HL=25ABh приобретает значение 2Ah
MOV   B,B       ;после выполнения команды ничего не изменяется
```

Команда **MVI R,D8** 2-3(7-10) 2 байта Не изменяет

Назначение. Непосредственная загрузка (Move Immediate) однобайтной константы. Байт данных, заданный непосредственно в самой команде вторым байтом загружается в указанный регистр или ячейку памяти. При загрузке ячейки памяти адрес ячейки ОЗУ определяется содержимым регистровой пары HL, при этом увеличивается число циклов и тактов.

Формат. MVI R,D8 ; R=A, B, C, D, E, H, L, M(HL)

Пример. Пусть D8=2Ah, HL=2547h. Тогда

```
MVI  B,2Ah     ;после выполнения команды B=2Ah
MVI  M, 2Ah    ; после выполнения команды M(2547h)=2Ah
```

Команда **NOP** 1(4) 1 байт Не изменяет

Назначение. Команда ничего не выполняет (No Operations), но реализуется как любая другая команда: выбирается из памяти (код 00), расшифровывается, осуществляется переход к следующей команде. На все это расходуется время, поэтому результат этой команды - задержка.

Формат. NOP

Команда **ORA R** 1-2(4-7) 1 байт На все признаки

Назначение. Логическое сложение (OR). Байт в указанном регистре поразрядно складывается логически с аккумулятором. Результат сохраняется в аккумуляторе. Команда часто используется для установки группы битов в 1.

Формат. ORA R ; R=A, B, C, D, E, H, L, M(HL)

Пример. Пусть B=25h, A=0ABh, тогда

```
ORA  B    ;выполнение обеспечит поразрядное логическое ИЛИ
          ;  B= 25h=  00100101b
          ;  A=0ABh=10101011b
          ;  A=0AFh =10101111b
          ; Флаг нуля и переноса сбросятся Z=0 и CR=0
```

Команда **ORI D8** 2(7) 2 байт На все признаки

Назначение. Логическое ИЛИ непосредственное (OR Immediate). Константа, размещенная во втором байте команды поразрядно складывается логически с аккумулятором. Результат сохраняется в аккуму-

ляторе. Команда часто используется для установки группы битов в 1.

Формат. ORI D8

Пример. Пусть D8=25h, A=0ABh, тогда

```
ORI  25h  ;выполнение обеспечит поразрядное логическое ИЛИ
          ;  D8=25h= 00100101b
          ;  A=0ABh=10101011b
          ;  A=0AFh =10101111b
          ; Флаг нуля и переноса сбросятся Z=0 и CR=0
```

Команда **OUT N** 3(10) 2 байта Не изменяет

Назначение. Вывод данных во внешний порт (Output). Один байт данных из аккумулятора выводится во внешний порт с номером, указанным во втором байте команды.

Формат. OUT N ; N=0 - 255

Пример. Пусть N=23, F=39h. Тогда

```
OUT 23    ; после выполнения команды порт с номером 23 равен 39h
```

Команда **PCHL** 1(5) 1 байта Не изменяет

Назначение. Безусловный переход с косвенной адресацией через регистровую пару HL (Jump H and L indirect - move H and L to PC). Содержимое регистровой пары HL заносится в счетчик команд PC. Это эквивалентно переходу по адресу, хранящемуся в HL, т.к. очередная команда будет извлекаться из адреса, соответствующего новому значению счетчика команд PC после выполнения этой команды.

Формат. PCHL

Пример. Пусть HL=2AF1h. Тогда

```
PCHL      ; после выполнения команды PC= 2AF1h
```

Команда **POP YZ** 3(10) 1 байт Для PSW

Назначение. Чтение данных из стека (Pop - хлопнуть, выстрелить). Два байта из ОЗУ по адресу, находящемуся в регистре указателе стека SP записывается в указанную пару регистров. Байт по адресу SP загружается в младший байт указанной пары, а байт по адресу на единицу большему SP загружается в старший байт указанной пары. Содержимое SP увеличивается на два. Если в качестве пары указан PSW, то содержимое M(SP) загружается в регистр

флагов, а M(SP+1) - в аккумулятор. Состояние регистра флагов при этом, естественно, изменяется. Состояние считываемых ячеек ОЗУ - не изменяется.

Формат. POP YZ ; YZ=BC, DE, HL, PSW

Пример. Пусть SP=0F52Ch, M(0F52Ch)=2Ah, M(0F53Ch) = 0FFh. Тогда

```
PUSH    PSW     ; после выполнения команды Флаги=0FFh, а
                ; A=0FFh,   SP=0F52Eh
```

Команда **PUSH YZ** 3(11) 1 байт Не изменяет

Назначение. Запись данных в стек (Push - толкать). Содержимое указанной пары регистров записывается в два байта в ОЗУ по адресу, находящемуся в регистре указателе стека SP. Содержимое первого байта пары записывается по адресу на единицу меньшему, чем SP, а содержимое второго байта пары - на два меньшему SP. Содержимое SP уменьшается на два. Если в качестве пары указан PSW, то первым записывается аккумулятор, а вторым - байт состояния. Состояние указанной регистровой пары не изменяется.

Формат. PUSH YZ ; YZ=BC, DE, HL, PSW

Пример. Пусть HL=2AF1h, SP=0F52Ch. Тогда

```
PUSH        H       ;после выполнения команды M(0F52Bh)=2Ah, а
                    ; M(0F52Ah)=0F1h
                    ; SP=0F52h
```

Команда **RAL** 1(4) 1 байт На CR

Назначение. Сдвиг влево через перенос CR (Rotate Left through Carry). Содержимое аккумулятора циклически сдвигается на один бит влево, включая признак переноса CR: бит переноса сдвигается на место младшего бита аккумулятора, а старший бит аккумулятора - на место переноса.

Формат. RAL

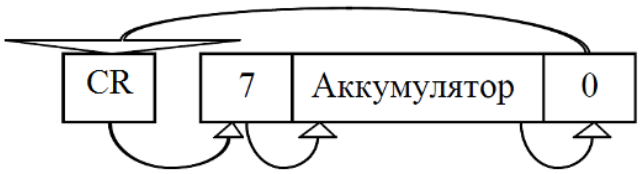
Пример. Пусть A=25h=00100101b, CR=1 тогда

```
RAL         ;после выполнения A=4Bh=01001011b, CR=0
```

Команда **RAR** 1(4) 1 байт На CR

Назначение. Сдвиг вправо через перенос CR (Rotate Right through Carry). Содержимое аккумулятора циклически сдвигается на один бит

вправо, включая перенос CR: младший бит передается в бит флага переноса CR, а бит переноса на место старшего бита.



Формат. RAR

Пример. Пусть A=25h=00100101b, CR=0 тогда

```
RAR          ;после выполнения A=12h=00010010b, CR=1
```

Команда **RET** 3(10) 1 байт Не изменяет

Назначение. Возврат (Return) из подпрограммы. Содержимое стека заносится в счетчик команд PC. Если там хранится адрес следующей команды за командой вызова подпрограммы, то запись этого адреса в счетчик команд PC и обеспечивает возврат к этой команде. Технически содержимое ячейки памяти, адрес которой определяется содержимым регистра SP, заносится на место младших восьми разрядов в счетчик команд PC. Содержимое ячейки памяти, адрес которой на единицу больше исходного содержимого регистра SP, заносится на место старших восьми разрядов счетчика команд. Содержимое регистра указателя стека SP увеличивается на 2.

Формат. RET

Пример. Пусть SP=0FF02h, M(0FF02h)=28h, M(0FF03h)=0ABh. Тогда

```
RET          ; после выполнения команды PC= 0AB28h
             ; SP=0FF04h
```

Команда **R-CON** 1-3(5-11) 1 байт Не изменяет

Назначение. Условный возврат (Condition Return) из подпрограммы. Если условие истинно, то действия определяются выполнением команды RET. В противном случае продолжается выполнение программы со следующего за этой командой адреса (к счетчику команд PC прибавляется длина команды - 1 байт). Состав условий определен таблицей в команде J-CON.

Формат. R-CON

Команда **RLC** 1(4) 1 байт На CR

Назначение. Циклический сдвиг влево (Rotate Left). Содержимое аккумулятора сдвигается на один бит влево, старший бит передается на место младшего бита. Флаг переноса CR приобретает значение старшего бита.

Формат. RLC

Пример. Пусть A=25h=00100101b, CR=1 тогда

```
RLC          ;после выполнения A=4Ah=01001010b, CR=0
```

Команда **RRC** 1(4) 1 байт На CR

Назначение. Циклический сдвиг вправо (Rotate Right). Содержимое аккумулятора сдвигается на один бит вправо, младший бит передается на место старшего бита. Флаг переноса CR приобретает значение младшего бита.

Формат. RRC

Пример. Пусть A=25h=00100101b, CR=0 тогда

```
RRC          ;после выполнения A=92h=10010010b, CR=1
```

Команда **RST X** 3(11) 1 байт Не изменяет

Назначение. Перезапуск (Restart) - переход к фиксированной подпрограмме. Адрес следующей команды записывается в стек. Управление передается команде, адрес которой определяется как фиксированный адрес ADR, вычисляемый умножением X (X=0, 1, 2, 3, 4, 5, 6, 7) на 8. Эта команда используется для организации работы обработчиков прерываний. Для этого эти 8 адресов должны иметь соответствующие команды управления.

Формат. RST X ; X=0, 1, 2, 3, 4, 5, 6, 7

Пример. Пусть SP=0FF02h, X=2, PC=1267h. Тогда

```
RST  2       ; после выполнения команды PC= 0010h,  SP=0FF04h
             ; В стеке будет записано число  1266h
```

Команда **SBB R** 1-2(4-7) 1 байт На все признаки

Назначение. Вычитание с заемом (Subtract with borrow). Из аккумулятора вычитается байт из указанного регистра и значение признака переноса CR (технически это осуществляется как сложение указан-

ного регистра с признаком переноса CR и вычитание получившегося результата из аккумулятора). Результат сохраняется в аккумуляторе. Операция выполняется точно так же как вычитание и используется для вычитания многобайтных чисел аналогично команде ADC.

Формат. SBB R ; R=A, B, C, D, E, H, L, M(HL)

Пример. Пусть B=2, A=4, CR=1. Тогда

```
SBB  B   ; выполнение команды обеспечит следующие действия:
         ;1) сложение признака с байтом из регистра:
         ;      02h+CR=03h
         ;2) получение дополнительного кода:
         ; Доп. код = 11111101b
         ;3) сложение с аккумулятором
         ; A= 4h= 00000100b
         ;код=    11111101b
         ; A=01h=00000001b
         ; Флаг нуля и переноса сбросятся
               ;  Z=0 и CR=
```

Команда **SBI D8** 2(7) 2 байта На все признаки

Назначение. Вычитание с константой и заемом (Subtract Immediate with borrow). Байт данных, заданный непосредственно вторым байтом команды складывается с переносом CR и вычитаются из аккумулятора. Сложение осуществляется в дополнительных кодах. Для выполнения операции предварительно сумма байта и переноса преобразуется в дополнительный код и на сумматоре складывается с содержимым аккумулятора. Результат сохраняется в аккумуляторе.

Формат. SBI D8

Пример. Пусть D8=25h, A=6Bh, CR=1. Тогда

```
SBI  25h  ; выполнение команды обеспечит следующие действия:
          ;Получение суммы байта D8 и переноса CR
          ; D8= 25h=  00100101b
          ; CR= 1h=           1b
          ; Sum= 26h=00100110b
          ;Получение дополнительного кода байта Sum
          ;(инвертирование Sum и прибавление единицы)
          ;  Sum= 26h=  00100110b
          ; NOT(Sum)=   11011001b
          ; +                1b
          ; Доп(Sum)=    11011010b=0DAh
          ;Далее сложение с аккумулятором
          ;  A=6Bh=     01101011b
```

```
;  Доп(Sum)=11011010b
;  A=45h=     01000101b. Флаг нуля и переноса сбросятся
; Z=0 и CR=0
```

Команда **SHLD D16** 5(16) 3 байт Не изменяет

Назначение. Запись в память пары HL (Store H and L Direct) по прямому адресу. Содержимое регистра L передать в ячейку ОЗУ, адрес которой задан вторым и третьим байтом команды. Содержимое H передать в ячейку с последующим адресом (байты из HL записываются наоборот: сначала младший, затем - старший).

Формат. SHLD D16 ; только для HL

Пример. Пусть HL=3A27h, D16=025ABh, тогда

```
SHLD   025ABh  ;после выполнения команды ячейка ОЗУ с
               ;адресом 25ABh станет равна 27h
               ;с адресом 25ACh станет равна 3Ah
```

Команда **SPHL** 1(5) 1 байт Не изменяет

Назначение. Пересылка пары регистров HL в стек (Move HL to SP). Данные, содержащиеся в паре регистров HL пересылаются в регистр указателя стека SP.

Формат. SPHL

Пример. Пусть HL=2AF1h, SP=0F52Ch. Тогда

```
SPHL       ; после выполнения команды HL= 2AF1h,
           ; SP=2AF1h
```

Команда **STA D16** 4(13) 3 байт Не изменяет

Назначение. Загрузка из аккумулятора в память по прямому адресу (Store Accumulator direct). Содержимое аккумулятора передать в ячейку ОЗУ, адрес которой задан вторым и третьим байтом команды.

Формат. STA D16

Пример. Пусть A=27h, D16=025ABh, тогда

```
STA  025ABh    ;после выполнения команды ячейка ОЗУ с
               ;адресом 25ABh станет равна 27h
```

Команда **STAX YZ** 2(7) 1 байт Не изменяет

Назначение. Запись в память содержимого аккумулятора (Store Accumulator indirect). Содержимое аккумулятора пересылается в ячейку памяти, адрес которой находится в заданной регистровой паре (BC или DE). Состояние аккумулятора не изменяется. Команда использует только две пары регистров. Третья пара (HL) используется командой MOV.

Формат. STAX YZ ; YZ=BC, DE

Пример. Пусть A=2Ah, B=25h, C=0ABh тогда

```
STAX    B    ;после выполнения команды ячейка ОЗУ с адресом
             ;25ABh приобретает значение 2Ah
```

Команда	**STC**	1(4)	1 байт	На CR

Назначение. Установка флага переноса CR=1 (Set Carry).

Формат. STC

Пример.

```
STC     ;после выполнения команды CR=1 при любом начальном
        ; значении
```

Команда	**SUB R**	1-3(4-7)	1 байт	На все признаки

Назначение. Вычитание (Subtract). Байт в указанном регистре вычитается из аккумулятора. Результат сохраняется в аккумуляторе. Для выполнения операции предварительно байт в указанном регистре преобразуется в дополнительный код и на сумматоре складывается с содержимым аккумулятора. Содержимое регистра не изменяется.

Формат. SUB R ; R=A, B, C, D, E, H, L, M(HL)

Пример. Пусть B=25h, A=6Bh, тогда

```
SUB  B  ; выполнение команды обеспечит следующие действия:
        ;Получение дополнительного кода регистра B
        ;(инвертирование регистра и прибавление единицы)
        ;   B= 25h=  00100101b
        ; NOT(B)=   11011010b
        ;  +                1b
        ; Доп(B)=   11011011b=0DBh
        ;Далее сложение с аккумулятором
        ;   A=6Bh=01101011b
        ;   Доп(B)=11011011b
        ;   A=46h= 01000110b. Флаг нуля и переноса сбросятся
        ; Z=0 и CR=0
```

Команда **SUI D8** 2(7) 2 байта На все признаки

Назначение. Вычитание с константой (Subtract Immediate). Байт данных, заданный непосредственно вторым байтом команды вычитается из аккумулятора. Сложение осуществляется в дополнительных кодах. Для выполнения операции предварительно байт преобразуется в дополнительный код и на сумматоре складывается с содержимым аккумулятора. Результат сохраняется в аккумуляторе.

Формат. ACI D8

Пример. Пусть D8=25h, A=6Bh, тогда

```
SUI 25h   ; выполнение команды обеспечит следующие действия:
          ;Получение дополнительного кода байта D8
          ;(инвертирование байта и прибавление единицы)
          ;  D8= 25h=  00100101b
          ; NOT(D8)=   11011010b
          ; +                1b
          ; Доп(D8)=    11011011b=0DBh
          ;Далее сложение с аккумулятором
          ;  A=6Bh=  01101011b
          ;  Доп(D8)=11011011b
          ;  A=46h=    01000110b. Флаг нуля и переноса сбросятся
          ; Z=0 и CR=0
```

Команда **XCHG** 1(4) 1 байт Не изменяет

Назначение. Обмен регистровых пар HL и DE (eXCHanGe H and L with D and E). Пара регистров H и L обмениваются содержимым с парой регистров D и E.

Формат. XCHG

Пример. Пусть HL=2AF1h, DE=0F52Ch. Тогда

```
XCHG          ; после выполнения команды DE=2AF1h, HL=0F52Ch
```

Команда **XRA R** 1-2(4-7) 1 байт На все признаки

Назначение. Сложение по модулю 2 (исключающее ИЛИ, неравнозначность Exclusive OR). Байт в указанном регистре поразрядно складывается по модулю 2 (1 получается при разных битах) с аккумулятором. Результат сохраняется в аккумуляторе. Операция часто используется для определения изменившихся битов.

Формат. XRA R ; R=A, B, C, D, E, H, L, M(HL)

Пример. Пусть B=25h, A=0ABh, тогда

```
XRA  B    ;выполнение команды обеспечит сложение по модулю 2:
          ;  B= 25h=  00100101b
          ;  A=0ABh=10101011b
          ;  A=.8Eh = 10001110b
          ; Флаг нуля и переноса сбросятся Z=0 и CR=0
```

Команда **XRI D8** 2(7) 2 байт На все признаки

Назначение. Сложение по модулю 2 с константой (исключающее ИЛИ, неравнозначность - Exclusive OR Immediate). Байт, находящийся во втором байте команды поразрядно складывается по модулю 2 (1 получается при разных битах) с аккумулятором. Результат сохраняется в аккумуляторе. Операция часто используется для определения изменившихся битов.

Формат. XRI D8

Пример. Пусть D8=25h, A=0ABh. Тогда

```
XRI  25h  ;выполнение команды обеспечит сложение по модулю 2:
          ;  D8=25h= 00100101b
          ;  A=0ABh=10101011b
          ;  A=.8Eh = 10001110b
          ; Флаг нуля и переноса сбросятся Z=0 и CR=0
```

Команда **XTHL** 5(18) 1 байт Не изменяет

Назначение. Обмен стека с регистрами HL (Exchange stack top with H and L). Содержимое регистра L обменивается с содержимым ячейки ОЗУ, адрес которой находится в SP. Содержимое регистра H обменивается с содержимым ячейки ОЗУ, чей адрес на единицу больше адреса указателя стека SP. Содержимое указателя SP не изменяется.

Формат. XTHL

Пример. Пусть HL=2AF1h, SP=0F52Ch, M(0F52Ch)=00, M(0F52Dh)=0FFh. Тогда

```
XTHL      ; после выполнения команды HL= 00FFh,
          ; M(0F52Ch)= 0F1h,  M(0F52Dh)=2Ah
```

ПРИЛОЖЕНИЕ Г

Описание основных драйверов устройств

BORD 015Ch

Назначение. Подпрограмма устанавливает цвет бордюра экрана. Код цвета запоминается в ОЗУ, а установка цвета производится в процессе прерываний при смене кадра. Выдача цвета осуществляется аппаратно, не используя экранное ОЗУ.

Входные: A - номер цвета бордюра из таблицы цветов (0-15)

COLOR 0156h

Назначение. Подпрограмма устанавливает цвет отображения и подкладки при выводе символа на экран в текстовом режиме, а также цвет точки при установке графического курсора. Цвет определяется номером физического цвета (0-15) в таблице цветов. Коды физических цветов устанавливаются подпрограммой SCOLOR.

Входные: A (младший полубайт) - номер цвета отображения (0-F)h

A (старший полубайт) - номер цвета подкладки (0 - рисование подкладки цветом фона)

CONIN 0132h

Назначение. Подпрограмма ожидает нажатия клавиши с клавиатуры и вводит ее код в буфер клавиатуры с последующей записью в аккумулятор. Если буфер уже содержит коды введенных символов, то будет переписан символ, введенный туда первым. При этом этот символ удаляется из буфера.

Выходные: A - код нажатой клавиши

CONOUT 0129h

Назначение. Подпрограмма осуществляет вывод символа на экран по месту установки курсора

Входные: C - код выводимого символа

FON 0159h

Назначение. Подпрограмма устанавливает цвет фона экрана. Следует иметь в виду, что подпрограмма только запоминает в памяти физический цвет экрана, а установка цвета экрана производится по прерыванию путем занесения кода физического цвета по номеру 0 таблицы цветов

Входные: A - код физического цвета фона (0-255)

INIT 0120h

Назначение. Подпрограмма инициализирует работу драйверов устройств, записывая по адресу ОЗУ 38H код команды JMP и адрес перехода на подпрограмму обработки внутренних прерываний. Подпрограмма формирует также таблицу для вычисления адресов элементов графических изображений (512 байт). Должна выполняться в начале программы пользователя

INKEY 013Bh

Назначение. Подпрограмма вводит код символа клавиши, которая нажата на момент обращения к подпрограмме (без ожидания ввода).

Выходные: A - код нажатой клавиши или 0FFH, если при обращении к подпрограмме ни одна из клавиш не была нажата (буфер клавиатуры пуст)

KEY 0123h

Назначение. Подпрограмма опрашивает состояние клавиатуры, производя проверку буфера клавиатуры на наличие в нем введенных символов.

Выходные: A - код состояния буфера клавиатуры:

- 00H - нет символа в буфере;
- 0FFH - есть символ в буфере либо производилось нажатие клавиши УС

LINE 0141h

Назначение. Подпрограмма в зависимости от заданного режима рисует линии, вычерчивает прямоугольники или закрашенные прямоугольники, а также подготавливает вывод символов увеличенного размера, начиная с точки, установленной подпрограммой PLOT.

Входные: B - координаты по оси Y конца линии или противоположного угла прямоугольника, или кратность увеличения размера символа;

C - аналогичные параметры по оси X;

A - режим:

0 - линии;

1 - прямоугольник;

2 - закрашенный прямоугольник;

3 - подготовка вывода символов увеличенного размера. Вывод осуществляется подпрограммами CONOUT или SPIC. Для перехода к стандартному размеру символов и текстовому курсору необходимо вывести код 00h

PLOT 013Eh

Назначение. Подпрограмма производит в зависимости от режима гашение или высвечивание точки на экране по месту текущего положения графического курсора или установку графического курсора без каких-либо действий над ним.

Входные: A - режим:

0 - гашение (стирание);

1 - высвечивание (рисование цветом отображения);

2 - установка графического курсора;

B - координата точки по оси Y (0-255);

C - координата точки по оси X (0-255).

Графические координаты не зависят от текущего положения текстового курсора. Начало координат в левом нижнем углу.

SPIC 0138h

Назначение. Подпрограмма осуществляет вывод на экран сообщения (цепочки символов) в формате ASCIIZ (цепочка заканчивается кодом 00h). Начало сообщения на экране определяется текущим положением курсора.

Входные: HL - адрес в ОЗУ выводимого сообщения

SCOLOR 0189h

Назначение. Подпрограмма устанавливает физические цвета. Это осуществляется путем переноса созданной пользователем 16-байтовой таблицы в специальный буфер. Его содержимое в момент прерывания переносится в порт таблицы цветов. Шестнадцатибайтовая таблица должна содержать коды физических цветов (из 256 цветов палитры). Порядковый номер цвета в этой таблице является кодом математического цвета (0-15) и определяется относительным смещением от начала таблицы (начиная с нуля). Значение физического цвета в первом байте таблицы, например, соответствует номеру цвета 0, во втором байте - номеру цвета 1 и т.д. Коды физических цветов формируются в соответствии с таблицей:

Первичный цвет	Синий (B)	Зеленый (G)	Красный (R)
Вес разряда в цвете	1/2 1/4	1/2 1/4 1/8	1/2 1/4 1/8
Код физическ. цвета	128 64	32 16 8	4 2 1
	БАЙТ таблицы цветов		

Например, желтый цвет средней яркости получается из 1/2 красного и 1/2 зеленого цветов. Тогда код этого цвета равен 4+32=36. Типовой вариант таблицы цветов, определяющей соответствие номеров цветов физическим цветам:

Номер цвета	Код физ. (дес.)	Код физ. (hex)	Код физич. (двоичный)	Цвет
0	64	40	01000000	Темно-синий
1	128	80	10000000	Синий
2	16	10	00010000	Зеленый
3	208	DO	11010000	Голубой
4	6	6	00000110	Красный
5	134	86	10000110	Малиновый
6	22	16	00010110	Кирпичный
7	54	36	00110110	Желтый
8	0	0	00000000	Черный
9	197	C5	11000101	Фиолетовый
10	34	22	00100010	Ярко-зеленый
11	192	CO	11000000	Ярко- синий
12	2	2	00000010	Темно-красный
13	152	98	10011000	Бирюзовый
14	82	52	01010010	Серый
15	173	AD	10101101	Белый

Математический цвет с номером 0 всегда является цветом фона.

Входные: DE - адрес таблицы в ОЗУ, содержащей 16 байт физических цветов

ПРИЛОЖЕНИЕ Д

СИСТЕМА КОМАНД МП КР580ВМ80А

ОДНОБАЙТОВЫЕ ПЕРЕСЫЛКИ					
MOV	R1,R	5/7	R	→	R1
MVI	R,D8	7	D8	→	R
STAX	YZ*	7	A	→	M(YZ)
LDAX	YZ*	7	M(YZ)	→	A
STA	ADR	13	A	→	M(ADR)
LDA	ADR	13	M(ADR)	→	A
ДВУХБАЙТОВЫЕ ПЕРЕСЫЛКИ					
LXI	YZ**,D16	10	D16	→	YZ
LHLD	ADR	16	M(ADR+1,ADR)	→	HL
SHLD	ADR	16	HL	→	M(ADR+1,ADR)
PUSH	YZ	11	YZ	→	M(SP-1,SP-2) SP-2→SP
POP	YZ	10	M(SP,SP+1)	→	YZ SP+2→SP
SPHL		5	HL	→	SP
ОБМЕН БАЙТАМИ					
XCHG		4	HL	↔	DE
XTHL		18	HL	↔	M(SP+1,SP)
КОМАНДЫ ВВОДА И ВЫВОДА					
IN	N	10	(N)	→	A
OUT	N	10	A	→	(N)

УСТАНОВКА ПРИЗНАКОВ		
CPI' D8	7	A-D8
CMP' R	4/7	A-R
16-БИТОВЫЕ ОПЕРАЦИИ		
DAD" YZ**	10	HL+YZ → HL
СПЕЦИАЛЬНЫЕ КОМАНДЫ		
EI	4	Разрешение прерывания
DI	4	Запрещение прерывания
HLT	7	Останов
NOP	4	Холостая операция

КОМАНДЫ СДВИГА Акк		
RLC"	4	Сдвиг влево
RAL"	4	Влево через C (CR)
RRC"	4	Сдвиг вправо
RAR"	4	Вправо через C (CR)

PSW = A + F

АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОПРЕРАЦИИ С ОДНИМ ОПЕРАНДОМ

CMC"		4	NOT CR	→	CR
STC"		4	1	→	CR
CMA		4	NOT A	→	A
DAA'		4	Десятичная коррекция		
INR"'	R	5/10	R+1	→	R
DCR"'	R	5/10	R-1	→	R
INX	YZ**	5	YZ+1	→	YZ
DCX	YZ**	5	YZ-1	→	YZ

АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОПРЕРАЦИИ С ДВУМЯ ОПЕРАНДАМИ

ADD'	R	4/7	A+R	→	A	ADI'	D8	7	A+D8	→	A
ADC'	R	4/7	A+R+CR	→	A	ACI'	D8	7	A+D8+CR	→	A
SUB'	R	4/7	A-R	→	A	SUI'	D8	7	A-D8	→	A
SBB'	R	4/7	A-R-CR	→	A	SBI'	D8	7	A-D8-CR	→	A
ANA'	R	4/7	A&R	→	A	ANI'	D8	7	A&D8	→	A
ORA'	R	4/7	AVR	→	A	ORI'	D8	7	AVD8	→	A
XRA'	R	4/7	A(m2)R	→	A	XRI'	D8	7	A(m2)D8	→	A

КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ

PCHL		5	HL	→	PC
JMP	ADR	10	ADR	→	PC
J-CON	ADR	10	ADR	→	PC

РЕГИСТР ФЛАГОВ F

M Z 0 AC 0 P 1 CR

M- знак AC- перенос из половины байта

Z- нуль P- четность CR- перенос

КОМАНДЫ ВЫЗОВА И ВОЗВРАТА ИЗ ПОДПРОГРАММ

CALL	ADR	17	PC+3	→	M(SP-1,SP-2),	ADR→PC		SP-2	→	SP
C-CON	ADR	11(17)	PC+3	→	M(SP-1,SP-2),	ADR→PC		SP-2	→	SP
RST	X	11	PC	→	M(SP-1,SP-2),	ADD→PC		SP-2	→	SP
					X=0, 1, … 7 и соответствующие					
					ADD= 0h, 8h, 10h, 18h, 20h, 28h, 30h, 38h					
RET		10	M(SP,SP+1)	→	PC		SP+2→SP			
R-CON		5(11)	SP+2	→	SP		M(SP,SP+1)→PC			

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ:

' - команда оказывает воздействие на все признаки; " - на признак C; '" - на все кроме C

R,R1 - содержимое регистров A,B,C,D,E,H,L или M(HL). Большее число - такты при работе с M(HL)

YZ - содержимое пары BC, DE, HL или PSW; YZ* = BC или DE; YZ** = BC, DE, HL или SP

SP - содержимое указателя стека перед выполнением команды (последней занятой ячейки)

D8 - 8-разрядный операнд, D16 - 16-разрядный операнд

(N) - содержимое порта ввода или вывода с номером (адресом) N (N = 0 … 255)

ADR - 16-разрядный адрес в трехбайтовой команде

M( ) - содержимое ячейки памяти с адресом указанным в скобках

-CON - часть мнемоники команды, определяющая условие передачи управления, вызова и возврата из подпрограммы (заменяется на NZ, Z, NC, C, PO, PE, P, или M). В скобках указано число тактов при выполнении условия передачи управления.

PC - счетчик команд, PSW - аккумулятор и регистр флагов. CR - бит флага переноса "C"

ПРИЛОЖЕНИЕ Е

Команды условной передачи управления МП КР580ВМ80А

Условие	Команда, устанавливающая флаг	Команда передачи управления
Любой бит Аккумулятора =0	ANI D8 (1 в соотв. Разряде)	JZ ADR
Любой бит Аккумулятора =1	ANI D8 (1 в соотв. Разряде)	JNZ ADR
Бит 7 Аккумулятора =0	RAL, RLC или ADD A	JNC ADR
Бит 7 Аккумулятора =1	RAL, RLC или ADD A	JC ADR
Бит 6 Аккумулятора =0	ADD A	JP ADR
Бит 6 Аккумулятора =1	ADD A	JM ADR
Бит 0 Аккумулятора =0	RAR или RRC	JNC ADR
Бит 0 Аккумулятора =1	RAR или RRC	JC ADR
Все биты Аккумулятора =0	ANA A или ORA A	JZ ADR
Содержимое Аккумулятора <>0	ANA A или ORA A	JNZ ADR
Содержимое Аккумулятора положительно (D7=0)	ANA A или ORA A	JP ADR
Содержимое Аккумулятора отрицательно (D7=1)	ANA A или ORA A	JM ADR
Содержимое Аккумулятора =D8	CPI D8	JZ ADR
Содержимое Аккумулятора <>D8	CPI D8	JNZ ADR
Содержимое Аккумулятора >=D8	CPI D8	JNC ADR
Содержимое Аккумулятора <D8	CPI D8	JC ADR
Содержимое Аккумулятора =R	CMP R	JZ ADR
Содержимое Аккумулятора <>R	CMP R	JNZ ADR
Содержимое Аккумулятора >=R	CMP R	JNC ADR
Содержимое Аккумулятора <R	CMP R	JC ADR

ПРИЛОЖЕНИЕ Ж

Коды команд МП КР580ВМ80А

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	NOP	LXI B,&	STAX B	INX B	INR B	DCR B	MVI B,#	RLC		DAD B	LDAX B	DCX B	INR C	DCR C	MVI C,#	RRC	0
1		LXI D,&	STAX D	INX D	INR D	DCR D	MVI D,#	RAL		DAD D	LDAX D	DCX D	INR E	DCR E	MVI E,#	RAR	1
2		LXI H,&	SHLD *	INX H	INR H	DCR H	MVI H,#	DAA		DAD H	LHLD *	DCX H	INR L	DCR L	MVI L,#	CMA	2
3		LXI SP,&	STA *	INX SP	INR M	DCR M	MVI M,#	STC		DAD SP	LDA *	DCX SP	INR A	DCR A	MVI A,#	CMC	3
4	MOV B,B	MOV B,C	MOV B,D	MOV B,E	MOV B,H	MOV B,L	MOV B,M	MOV B,A	MOV C,B	MOV C,C	MOV C,D	MOV C,E	MOV C,H	MOV C,L	MOV C,M	MOV C,A	4
5	MOV D,B	MOV D,C	MOV D,D	MOV D,E	MOV D,H	MOV D,L	MOV D,M	MOV D,A	MOV E,B	MOV E,C	MOV E,D	MOV E,E	MOV E,H	MOV E,L	MOV E,M	MOV E,A	5
6	MOV H,B	MOV H,C	MOV H,D	MOV H,E	MOV H,H	MOV H,L	MOV H,M	MOV H,A	MOV L,B	MOV L,C	MOV L,D	MOV L,E	MOV L,H	MOV L,L	MOV L,M	MOV L,A	6
7	MOV M,B	MOV M,C	MOV M,D	MOV M,E	MOV M,H	MOV M,L	HLT	MOV M,A	MOV A,B	MOV A,C	MOV A,D	MOV A,E	MOV A,H	MOV A,L	MOV A,M	MOV A,A	7
8	ADD B	ADD C	ADD D	ADD E	ADD H	ADD L	ADD M	ADD A	ADC B	ADC C	ADC D	ADC E	ADC H	ADC L	ADC M	ADC A	8
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB M	SUB A	SBB B	SBB C	SBB D	SBB E	SBB H	SBB L	SBB M	SBB A	9
A	ANA B	ANA C	ANA D	ANA E	ANA H	ANA L	ANA M	ANA A	XRA B	XRA C	XRA D	XRA E	XRA H	XRA L	XRA M	XRA A	A
B	ORA B	ORA C	ORA D	ORA E	ORA H	ORA L	ORA M	ORA A	CMP B	CMP C	CMP D	CMP E	CMP H	CMP L	CMP M	CMP A	B
C	RNZ	POP B	JNZ *	JMP *	CNZ *	PUSH B	ADI #	RST 0	RZ	RET	JZ *		CZ *	CALL *	ACI #	RST 1	C
D	RNC	POP D	JNC *	OUT N	CNC *	PUSH D	SUI #	RST 2	RC		JC *	IN N	CC *		SBI #	RST 3	D
E	RPO	POP H	JPO *	XTHL	CPC *	PUSH H	ANI #	RST 4	RPE	PCHL	JPE *	XCHG	CPE *		XRI #	RST 5	E
F	RP	POP PSW	JP *	DI	CP *	PUSH PSW	ORI #	RST 6	RM	SPHL	JM *	EI	CM *		CPI #	RST 7	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

N - однобайтовый номер порта ввода/вывода
& - двухбайтовый операнд - D16
* - двухбайтовый адрес - ADR
# - однобайтовый операнд - D8

Например:
команда LHLD * имеет код 2Ah и занимает 3 байта:
1-й байт - код операции;
2 и 3-й байты - адрес ADR

Список литературы

1. Потемкин И.С. Функциональные узлы цифровой автоматики. - М.: Энергоатомиздат, 1988. - 320с.
2. Хоровиц П., Хилл У. Искусство схемотехники. В 3 т. – М.: Мир, 1993.
3. Алексеенко А.Г., Галицын А.А., Иванников А.Д. Проектирование радиоэлектронной аппаратуры на микропроцессорах: Программирование, типовые решения, методы отладки. - М.: Радио и связь, 1984. - 272с.
4. Левенталь Л., Сэйвилл У. Программирование на языке ассемблера для микропроцессоров 8080 и 8085.-М.: Радио и связь, 1987. - 448с.
5. Угрюмов Е.П. Цифровая схемотехника. - СПб.: БХВ-Петербург, 2004. -528с.
6. Бойко В.И. и др. Схемотехника электронных систем. Микропроцессоры и микроконтроллеры. - СПб.: БХВ-Петербург, 2004. - 464с.
7. Микушин А. и др. Цифровые устройства и микропроцессоры. Учебное пособие. - Санкт-Петербург: «БХВ-Петербург», 2010. - 832с.
8. Белов А.В. Самоучитель разработчика устройств на микроконтроллерах AVR. - СПб.: Наука и техника, 2008. - 544с.
9. Белов А.В. Разработка устройств на микроконтроллерах AVR. - СПб.: Наука и техника, 2013. - 528с.
10. Евстифеев А.В. Микроконтроллеры AVR семейства Tiny. Руководство пользователя. - М.: Издательский дом "Додэка-XXI", 2007. - 432с.
11. Евстифеев А.В. Микроконтроллеры AVR семейства Mega. Руководство пользователя. - М.: Издательский дом "Додэка-XXI", 2007. - 592с.
12. Ревич Ю.В. Практическое программирование микроконтроллеров AtmelAVR на языке ассемблера. - СПб.: БХВ-Петербург, 2011. - 352с.
13. Болдырихин Н.В., Манин А.А. Техника микропроцессорных систем в коммутации. Учебное пособие. - Ростов н/Дону: СКФ МТУСИ, 2010. -135 с.
14. Основы микропроцессорной техники http://window.edu.ru/resource/737/74737
15. Ресурсы Internet по электронике и микропроцессорной технике http://newit.gsu.by/resources/mp/inet.htm
16. Микропроцессорная лаборатория http://mculab.ru/index.php
17. Списки литературы по теме "Микропроцессоры" http://spilit.info/mikro/mikroprocessor.htm
18. Сайт информационной безопасности http://security-corp.org/hard/1058-rossiyskie-mikroprocessory.html
19. Архив журнала CHIP http://ichip.ru
20. Форум программистов и системных администраторов http://Cyberguru.ru
21. Библиотека электронных схем http://chipdip.ru
22. Библиотека литературы по микроконтроллерам http://mirmk.ru