Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра защиты информации

ЦИФРОВЫЕ И МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА

Методическое пособие
для студентов специальности 1-45 01 03
«Сети телекоммуникаций»
заочной формы обучения

Минск БГУИР 2009

УДК 004.431.4(075.8)
ББК 32.973стд1-018.1я73
Ц75

Авторы:
Л. М. Лыньков, А. Л. Гурский, Н. В. Колбун, В. П. Ширинский

Рецензент:
доцент кафедры электроники БГУИР В. А. Мельников

Ц75 **Цифровые** и микропроцессорные устройства : метод. пособие для студ. спец. 1-45 01 03 «Сети телекоммуникаций» заоч. формы обуч. / Л. М. Лыньков [и др.]. – Минск : БГУИР, 2009. – 34 с.
ISBN 978-985-488-468-4

В пособии представлены примеры сложных случаев проектирования комбинационных и последовательных схем программными методами. Содержатся методические указания, теоретические вопросы и практические задания для выполнения контрольных работ.

УДК 004.431.4(075.8)
ББК 32.973стд1-018.1я73

ISBN 978-985-488-468-4

Содержание

Введение

Практическое использование микропроцессоров немыслимо без понимания их сути. А такое его представление невозможно без его программирования на Ассемблере. Опыт проведения занятий со студентами различных форм обучения и подсказал именно такое содержание пособия.

В пособии иллюстрированы примерами наиболее сложные типичные случаи использования отдельных команд и задач, самостоятельное решение которых для большинства студентов затруднительно.

1. Программирование на Ассемблере микропроцессора I8080

1.1. Программирование на машинном языке

Прикладные программы для первых МП-систем составлялись на машинном языке: все элементы программы (коды операций, адреса, данные) представлялись в двоичной форме или в эквивалентных, но более удобных, восьмеричной и шестнадцатеричной формах. Чтобы показать особенности программирования на машинном языке, рассмотрим короткий фрагмент некоторой прикладной программы.

В области памяти с начальным адресом 00300 находится текст, т. е. некоторая последовательность символов. Каждый символ закодирован восемью битами и размещается в одной ячейке памяти. Необходимо часть текста до первого появления кода точки переслать в другую область памяти с начальным адресом 0400. При обнаружении кода точки 4В (01001011) передача завершается и программа переходит к другим действиям. Фрагмент программы должен начинаться с адреса 0100. Для краткости область-источник назовем ERA1, а область-получатель – ERA2. Специально подчеркнем, что приведенная формулировка оказывается неполной. В частности, в ней отсутствует задание длины области ERA1. Естественно, формулировки реальных прикладных задач должны содержать исчерпывающие ответы на любые вопросы о характере задачи.

Очевидно, рассматриваемый фрагмент представляет собой простую циклическую программу: текущий символ из области ERA1 сравнивается с кодом точки и по результату сравнения либо символ передается ERA2, либо цикл заканчивается.

Чтобы не использовать длинных команд с прямой адресацией, введем два 16-битных указателя памяти, адресующих текущие ячейки областей ERA1 и ERA2. С помощью косвенной адресации через указатели можно обращаться к памяти короткими 1-байтными командами. Примем в качестве указателя текущей ячейки ERA1 регистры (H,L), а области ERA2 – (D, E).

Первый шаг заключается в инициализации указателей: в регистры (H, L) необходимо загрузить начальный адрес 0300, а в регистры (D, E) – адрес 0400. Воспользуемся 3-байтной командой непосредственной загрузки LXI rp,

при выполнении которой второй <B2> и третий <B3> байты загружаются в регистровую пару rp. Команда LXI H имеет код операции 21, а команда LXI D – 11. Напомним, что <B2> содержит младший байт адреса, а <B3> – старший. Поэтому начинаем программу со следующих двух команд:

Адрес	Код операции	<B2>	<B3>
0100	21	00	03
0103	11	00	04

Сравнивать коды символов можно только через аккумулятор, поэтому следующая команда должна передавать символ из ERA1 в аккумулятор. Нужную передачу осуществляет команда MOV A, M с кодом операции 7E:

Адрес	Код операции	<B2>	<B3>
106	7E		

Далее необходимо сравнить код сосчитанного символа с кодом точки. Для этого удобно воспользоваться командой неразрушающего сравнения CPI с непосредственным операндом <B2>, представляющего собой код точки. Команда CPI имеет код операции FE. Записываем следующую команду программы:

Адрес	Код операции	<B2>	<B3>
0107	FE	4B	

Соответствие символа из аккумулятора коду точки фиксируется путем установки флажка Z = 1. В этом случае необходимо осуществить переход к другим действиям. Если Z = 0, следует передать символ из аккумулятора в ERA2 и проанализировать следующий символ текста. Выход из цикла производится с помощью команды условного перехода JZ с кодом операции CA, но пока неизвестен адрес перехода. Поэтому зарезервируем 2 байта памяти:

Адрес	Код операции	<B2>	<B3>
0109	CA	?	?

Следующая команда с адресом 010C должна передавать символ из аккумулятора в область ERA2. Воспользуемся 1-байтной командой STAX D с кодом операции 12:

Адрес	Код операции	<B2>	<B3>
010C	12		

Обработка одного символа закончена и необходимо проверять следующий. Для этого модифицируем указатели 1-байтными командами инкремента INX H (код операции 23) INX D (код операции 13):

Адрес	Код операции	<B2>	<B3>
010D	23		
010E	13		

Повторение цикла осуществляется командой безусловного перехода JMP (код операции C3) к ячейке 0106 с командой MOV A,M:

Адрес	Код операции	<B2>	<B3>
010F	C3	06	01

Теперь известен адрес перехода 0112 в команде JZ:

Адрес	Код операции	<B2>	<B3>
0109	CA	12	01

Рассмотренный пример показывает те трудности, с которыми приходится сталкиваться при программировании на машинном языке:

1) необходимо помнить коды операций многочисленных команд, входящих в систему команд микропроцессора;

2) трудно следить за абсолютными адресами памяти, особенно в длинных программах с большим числом условных переходов;

3) сложно модифицировать составленную программу. Например, введение одной пропущенной команды может потребовать изменения операндов многих команд;

4) трудно разбираться в программе, состоящей только из числовой информации. Для документирования каждую команду следует сопровождать подробным комментарием, так как содержательный смысл (семантика) команд завуалирован в числах. Кроме того, работа исключительно с числами быстро утомляет программиста, что приводит к появлению в программе ошибок.

Однако программирование на машинном языке имеет и некоторые достоинства. Главное из них заключается в том, что программа оказывается наиболее эффективной, естественно, если она составлена опытным программистом. Так как в данном случае программист максимально близок к аппаратным средствам, знает особенности микропроцессора и каждой команды, может выбрать наилучший режим адресации, то программа оказывается короткой и быстродействующей. Программа работы большинства МП-систем материализуется в программном ПЗУ, и ее длина влияет на экономические показатели системы, особенно в условиях массового производства.

1.2. Программирование на языке Ассемблера

Язык Ассемблера, называемый также языком символического кодирования, позволяет в значительной степени устранить отмеченные выше недостатки программирования на машинном языке. Главное его достоинство заключается в том, что язык Ассемблера допускает представление всех элементов программы в символической (буквенно-цифровой) форме, отражающей их содержательный смысл. Преобразование символических наименований в двоичные коды машинного языка возлагается на специальную программу, называемую ассемблирующей программой, или Ассемблером. Программа-ассемблер действует как «сборщик», объединяя элементы программы в конкретные машинные команды.

Символические наименования (имена), вводимые при программировании на языке Ассемблера, имеют много синонимических названий – идентификаторы, символы, имена, метки, пометки и т. п. Чтобы избежать возможной путаницы, в дальнейшем будем придерживаться следующих определений: символическое наименование адреса называется меткой, символическое наименование кода операции называется мнемоникой или просто кодом. Подчеркнем, что символические наименования отражают семантику программы, например, PARAM – параметр, TABLE – таблица, MASK – маска, NXBIT – следующий бит и т. д. Мнемоники команд также передают их основную функцию, например, ADD – сложение, SUB – вычитание, INR – инкремент и т. п. Манипулировать такими объектами значительно проще и удобнее, чем числовыми данными, а программист допускает меньше ошибок. Длина символических наименований, состоящих из букв и цифр, ограничена 5-6 символами, первым из которых обязательно должна быть буква. В языке Ассемблера для каждого микропроцессора некоторые символические наименования фиксированы и иногда называются ключевыми словами. К ним относятся мнемоники кодов операций, директивы Ассемблера и наименования внутренних регистров. Остальные символические наименования определяет программист, придавая им содержательный смысл, согласующийся с контекстом прикладной программы.

Числовые данные в языке Ассемблера допускается представлять в двоичной, восьмеричной, десятичной и шестнадцатеричной системах счисления.

В качестве алфавита допустимых символов в зарубежных Ассемблерах принят код ASCII (Американский Стандартный Код для обмена информацией), содержащий латинские буквы (в языке Ассемблера – только прописные), цифры, разделители (двоеточие, точка с запятой, запятая) и знаки математических операций.

Ассемблерные программы записываются в виде последовательности команд, называемых также операторами и предложениями Ассемблера; для каждой команды отводится одна строка. Важнейшая особенность языка Ассемблера заключается в том, что каждый его оператор порождает одну машинную команду. Следовательно, программирование на Ассемблере по су-

ществу является более удобным способом программирования на машинном языке с сохранением его положительного качества с точки зрения эффективности прикладных программ. Ассемблер не освобождает программиста от ответственности в организации и оптимальном использовании системных ресурсов и требует полного понимания работы МП-системы.

1.2.1. *Поля ассемблерной строки*

Поле метки. Необязательное символическое наименование в поле метки ассоциируется с 16-битным адресом той ячейки памяти, в которую будет размещен первый байт отмеченной команды. Метки используются в качестве адресов перехода команд передачи управления и освобождают программиста от необходимости оперировать абсолютными адресами памяти.

Метка имеет длину от одного до пяти или шести символов, первым из которых обязательно должна быть буква. Метка не должна содержать знаков пунктуации и пробелов. За последним символом метки следует двоеточие. В качестве меток не допускается использовать ключевые слова.

Примеры допустимых меток:

AGAIN:, A15:, ONE:,C5FA:, Ml:.

Следующие метки недопустимы:

8ABC – начинается не с буквы;

DONE – не заканчивается двоеточием;

SP – использовано ключевое слово.

Поле мнемоники (кода). В поле мнемоники содержится символическое описание выполняемой команды, которое заменяет числовое значение кода операции. Большинство мнемоник представляют собой аббревиатуры предложений, характеризующих основные функции команд, например:

MOV (MOVe) – передать, переслать;

ACI (Add with Carry Immediate) – сложение с переносом непосредственное;

JNZ (Jump if Non Zero) – перейти, если не нуль;

XCHG (eXCHanGe) – обменять.

Некоторые мнемоники явно определяют функцию команды, например, команда PCHL означает передачу содержимого регистров (H, L) в программный счетчик PC, а команда PUSH выполняет загрузку в стек.

Обычно длина поля мнемоники не превышает четырех позиций, а между ним и соседним справа полем операнда необходим минимум один пробел. Запоминать и использовать мнемоники значительно проще и удобнее, чем численные коды операций. Мнемоники кодов операций являются ключевыми словами Ассемблера, и, если содержимое этого поля не входит в множество допустимых мнемоник, Ассемблер выдает сообщение о недействительной команде.

1.2.2. *Поле операнда*

В наиболее сложном поле операнда определяются данные, являющиеся операндом (операндами) команды. Следовательно, содержимое поля операнда должно интерпретироваться в соответствии с функцией команды. В качестве операндов могут фигурировать адреса памяти, внутренние регистры микропроцессора, адреса портов ввода и вывода, числовые и символьные константы. Некоторые команды, оперирующие определенными внутренними регистрами, имеют пустое поле операнда:

```
Метка     Код      Операнд     Комментарий
          CMA                  ; Инвертирование аккумулятора
          RAL                  ; Сдвиг влево через перенос
          XTHL                 ; Обмен (H, L) и верхушки стека
          NOP                  ; Холостая команда
```

Программа-ассемблер должна найти двоичный эквивалент содержимого поля операнда и подставить его (в зависимости от типа команды) в определенные биты кода операции, во второй байт 2-байтных команд или во второй и третий байты 3-байтных команд. Вычисленное двоичное значение содержимого поля операнда выравнивается справа и должно укладываться в диапазон, определяемый содержанием команды. В противном случае фиксируется недействительное поле операнда.

А. Отображение операнда в различных системах счисления

Поле операнда стандартного языка Ассемблера микропроцессора I8080 может содержать информацию следующих типов: числовые и символьные непосредственные данные, внутренние регистры и регистровые пары, адреса памяти. Рассмотрим способы определения информации указанных типов.

Шестнадцатеричные данные. Содержащееся в поле операнда шестнадцатеричное число должно начинаться с цифр 0…9 и завершаться буквой H (Hex). Число, начинающееся с букв A…F, дополняется слева незначащим нулем. Примеры задания шестнадцатеричных чисел:

```
 Метка     Код      Операнд      Комментарий
STORE:     STA      8000H      ; Запоминание в ячейке 8000
  MVI               C, OAAH    ; Загрузка  в  регистр C кода 10101010
           ANI      10H        ; Выделение четвертого бита
COMP:      CPI      290H       ; Недопустимый операнд
```

Десятичные данные. Каждое десятичное число в поле операнда может заканчиваться буквой D (Decimal). Десятичное число в поле операнда заканчивается необязательной буквой D (Decimal). Примеры задания десятичных чисел:

Метка	Код	Операнд	Комментарий
	MVI	B, 15	; Загрузка в регистр B кода 00001111
	ADI	1D	; Инкремент аккумулятора
	ANI	64	; Выделение шестого бита
	IN	32	;Ввод из порта с адресом 00100000

Восьмеричные данные. Каждое восьмеричное число в поле операнда должно заканчиваться буквой O (Octal). Чаще восьмеричную систему идентифицируют буквой Q, чтобы отличить букву O от цифры 0. Примеры восьмеричных чисел:

Метка	Код	Операнд	Комментарий
	ORI	200Q	; Установка старшего бита
	SUI	1Q	; Декремент аккумулятора
	MVI	H,777Q	; Недействительный операнд

Двоичные данные. Двоичное число в поле операнда должно заканчиваться буквой B (Binary). Примеры определения двоичных чисел:

Метка	Код	Операнд	Комментарий
	OUT	1111B	; Вывод в порт с адресом 15
	XRI	10000000B	; Инверсия знакового бита
	ANI	11011111B	; Сброс пятого бита

Б. Символьные константы

Символьные константы. В поле операнда допускается использовать символы внешнего алфавита, заключая их в апострофы. Программа-ассемблер подставляет вместо такого операнда соответствующий двоичный код символа:

Метка	Код	Операнд	Комментарий
	CPI	”.”	; Сравнение с кодом точки
	MVI	D,”T”	; Загрузка кода буквы T

Идентификаторы внутренних регистров. В поле операнда разрешается указывать символические наименования, которые определены в самом Ассемблере и связаны с внутренней архитектурой микропроцессора. В языке Ассемблера микропроцессора I8080 встроены идентификаторы внутренних регистров B, C, D, E, H, L, M, A с соответствующими двоичными значениями от 000 до 111* Примеры использования таких идентификаторов:

```
Метка    Код     Операнд    Комментарий
         MOV     A, E       ; Передача из регистра E в аккумулятор
         ADD     L          ; Прибавление содержимого регистра L
         SUB     M          ; Вычитание содержимого ячейки, адресуемой регистрами (H, L)
```

Вместо идентификаторов внутренних регистров допускается применять их адреса в любой системе счисления. Например, следующие команды эквивалентны: MOV A, B; MOV 7,0; MOV 111B, 0H и т. п.

В командах, оперирующих 16-битными значениями, применяются идентификаторы внутренних 16-битных регистров B, D, H, PSW, SP. Примеры таких команд:

```
Метка    Код     Операнд    Комментарий
         LXI     H, 0FF00H  ; Инициализация регистров (H, L)
         INX     SP         ; Инкремент указателя стека
         PUSH    PSW        ; Загрузка в стек содержимого A- и F-регистров
```

Метки. В командах передачи управления можно указывать метки, введенные программистом в поле метки других команд. Метки в поле операнда заменяют абсолютные значения адресов перехода. Пример:

```
Метка    Код     Операнд    Комментарий
         JMP     DONE       ; Переход к метке DONE
   :     ****
         CALL    SWAP       ; Вызов подпрограммы SWAP
         *****
DONE:    RAR
         ****
         *****
SWAP:    MOV     A,M
```

Любой символический адрес, фигурирующий в поле операнда команд передачи управления, должен 1 раз появиться в поле метки некоторого оператора. Если он не появляется ни разу, программа-ассемблер выдает сообщение о неопределенной метке.

Текущее содержимое программного счетчика. В командах передачи управления допускается относительная адресация, т.е. адрес перехода определяется суммой (разностью) текущего содержимого программного счетчика, которое определяется символом #, указываемого в операторе смещения.

Смещение может быть представлено в любой из ранее рассмотренных форм определения числовых значений. Примеры относительной адресации:

Метка	Код	Операнд	Комментарий
GOTO:	JMP	# +20H	; Переход по адресу GOTO+20H («вперед»)
MORE:	JNZ	# - 80	; Адрес перехода = MORE-80 («назад»)

Выражения. Наиболее сложными конструкциями поля операнда являются выражения. Они содержат в качестве аргументов все рассмотренные выше типы данных, которые связываются арифметическими и логическими операторами. Аргументы выражений считываются 15-битными целыми, а значения выражений определяют 16-битные целые без знака. Значение выражения должно соответствовать смыслу операции, определенной в поле мнемоники.

Результатами арифметических операторов сложения (+), вычитания (-), умножения (*) и деления (/) считаются соответственно арифметические суммы, разность, произведение и частное аргументов. Результатом оператора MOD является целый остаток от деления первого (левого) аргумента на второй. Унарный оператор «минус» означает вычитание аргумента из нуля.

Допустимыми логическими операторами являются следующие:

– унарный оператор NOT, инвертирующий каждый бит аргумента;

– операторы AND, OR, XOR, выполняющие соответственно побитные конъюнкцию, дизъюнкцию и сложение по mod 2 аргументов.

Приведем примеры допустимых выражений в поле операнда:

Метка	Код	Операнд	Комментарий
	MVI	B, 30 + 40H/2	; Загрузка числа 62
	SUI	34 MOD 3	; Декремент аккумулятора

Буквенные операторы MOD, NOT и другие должны отделяться от аргументов пробелом. В выражениях иногда допускается использование скобок.

Операторы в выражениях подчиняются стандартному старшинству: умножение, деление, MOD, сдвиги; сложение, вычитание; отрицание, конъюнкция; дизъюнкция, сложение по mod 2.

Введение выражений значительно усложняет программу-Ассемблер, поэтому они при программировании на Ассемблере применяются редко.

1.2.3. *Поле комментариев*

Поле комментария. Поле комментария, начинающееся с некоторого разделителя (точка с запятой или наклонная черта), полностью игнорируется

программой-Ассемблером, поэтому в нем можно помещать любой текст. Содержание этого поля должно пояснять те действия, которые выполняет команда в контексте конкретной прикладной программы. Комментарием может служить целая строка, начинающая с разделителя.

В отношении содержания комментариев следует придерживаться двух рекомендаций. Во-первых, в комментарии акцентируется не общая функция команды, которая определяется мнемоникой, а действие команды в данной программе. Например, для команды INX H бесполезно вводить комментарий «инкремент H-пары». Во-вторых, не следует сокращать объем комментария в ущерб его смыслу. Четкие и содержательные комментарии облегчают понимание и использование программ.

1.2.4. *Директивы Ассемблера*

В операторах ассемблерных программ часто используются мнемоники, отсутствующие в системе команд микропроцессора. Следовательно, они не могут породить команд объектной программы, поэтому их назвают псевдокомандами. Псевдокоманды являются указаниями программе-Ассемблеру о выполнении определенных действий в процессе ассемблирования, поэтому их чаще называют директивами Ассемблера. Директивы определяют порядок ассемблирования, размещают в памяти информацию, присваивают численные значения символическим наименованиям, резервируют память и выполняют другие функции.

Директивы подчиняются стандартному формату операторов Ассемблера, но содержимое их полей имеет некоторые особенности, например, в поле метки директив MACRO, EQU и SET должно обязательно находиться символическое наименование, которое не имеет заключительного двоеточия. В остальных директивах в поле метки может быть необязательная метка, аналогичная меткам машинных команд. Метка директивы относится к ячейке памяти, которая следует сразу же за последней ячейкой предшествующей машинной команды. Операнды директив необязательны.

Директива ORG. Директива ORG (начало) имеет следующий формат:

Метка	Код	Операнд	Комментарий
[метка:]	ORG	<выражение>	; Формат директивы ORG

Значением выражения директивы ORG является допустимый 16-битный адрес, определяющий ячейку памяти, в которую будет загружаться первый байт следующей команды или байт данных. До новой директивы ORG команды и данные размещаются в смежных ячейках памяти. Если в самом начале программы директива ORG отсутствует, то по умолчанию подразумевается наличие директивы ORG с нулевым операндом.

При необходимости в программе может быть несколько директив ORG:

```
Метка     Код     Операнд      Комментарий
          ORG     100H         ; Задает абсолютный адрес 0100H
          LXI     H, AREA1     ; Адрес = 0100H
          LXI     D, AREA2     ; Адрес = 0103H
          MOV     A, M         ; Адрес = 0106H
  W:      ORG     200H         ; Задает абсолютный адрес 0200H
          RAL                  ; Адрес = 0200H
          ANA     D            ; Адрес = 0201H
```

Директива ORG может выполнять функцию резервирования памяти, например, в следующем фрагменте она резервирует 20 байт:

```
 Метка     Код     Операнд     Комментарий
           MOV     A, M
           RAL
           JMP     LOWER
           ORG     # +20
LOWER:     ORA     A
```

Директива END. Формат директивы END выглядит следующим образом:

```
 Метка      Код     Операнд     Комментарий
[метка:]    END                 ; Формат директивы END
```

Эта директива информирует программу-Ассемблер о достижении физического конца входной программы. Разумеется, в каждой программе может быть только одна директива END, находящаяся в последней строке.

Директива EQU. Директива EQU (приравнять, присвоить) прямого присвоения имеет следующий формат:

```
 Метка      Код     Операнд          Комментарий
 <имя>      EQU     <выражение>      ; Формат директивы EQU
```

При выполнении директивы EQU программа-Ассемблер присваивает значение выражения символическому наименованию, находящемуся в поле метки. Когда наименование встречается в поле операнда, программа-Ассемблер подставляет вместо него присвоенное значение:

```
Метка       Код         Операнд      Комментарий
MASK:       EQU         0FH          ;Значение MASK равно 15
            ***
            ANI         MASK
            ***
            CPI         MASK
            ***
            MVI         A, MASK
            ***
```

В командах ANI, CPI, MVI вместо MASK будет фигурировать код 00001111. Если по каким-либо причинам в программе нужно изменить значение MASK, для этого достаточно модифицировать операнд одной директив EQU и выполнить повторное ассемблирование программы.

Заметим, что каждое символическое наименование может появиться в поле метки только одной директивы. При программировании следует сгруппировать все директивы EQU в начале или конце программы.

Директива SET. Директива SET (установить) имеет такой же формат и в общем выполняет такое же действие, что и директива EQU. Однако в отличие от директивы EQU значение символического наименования допускается изменять с помощью новой директивы SET:

```
Метка       Код         Операнд      Комментарий
NAME        SET         15           ; Значение NAME равно 15
            ***
            MVI         B, NAME      ; Загрузка 15 в регистр B
            ***
NAME        SET         1FH          ; Значение NAME становится 31
            ***
            ADI         NAME         ; Прибавление 31
            ***
            CPI         NAME         ; Сравнение с 31
            ***
```

Директивы IF и ENDIF. Директивы условного ассемблирования IF (если) и ENDIF (конец если) применяются в ассемблерных программах следующим образом:

```
Метка       Код         Операнд          Комментарий
[метка:]    IF          <выражение>      ; Директивы условного ассемблирования
            ***
```

```
                Операторы
                  ***
[метка:]        ENDIF
```

В процессе ассемблирования вычисляется значение выражения из поля операнда директивы IF. Если оно равно нулю, операторы между директивами IF и ENDIF игнорируются и не включаются в объектную программу. Когда же значение выражения отличается от нуля, операторы программы ассемблируются так, как будто директив IF и ENDIF нет.

Директива DB. Директива DB (определить байт) относится к группе директив определения, которые применяются для инициализации данных и резервирования памяти. Формат директивы DB имеет следующий вид:

```
 Метка          Код         Операнд        Комментарий
[метка:]        DB          <список>       ;Формат директивы  DB
```

Операнд директивы DB может быть последовательностью выражений, разделенных запятыми и имеющих 8-битные значения, либо цепочкой символов, заключенных в апострофы. При выполнении директивы DB значения выражений или коды символов запоминаются в смежных ячейках (байтах) памяти, начинающихся после ячейки предыдущей команды. Приведем примеры использования директивы DB:

```
 Метка          Код         Операнд        Комментарий
ARRAY:          DB          3,7,15, 31     ; Запоминаются четыре значения
                ***
                DB          'HELLO'        ; Запоминаются пять символов
                ***
COMPL:          DB          – 63           ; Дополнительный код  – 63
```

Директива DW. Директива DW (определить слово – 2 байта) также относится к директивам определения и имеет такой же формат, как и директива DB. Однако здесь списком является последовательность выражений, имеющих 16-битные значения. При выполнении директивы DW вычисляется значение первого выражения и его младшие 8 бит запоминаются по текущему адресу, а старшие 8 бит запоминаются по адресу на единицу больше предыдущего. Затем вычисляется значение второго выражения, процедура запоминания повторяется для следующих ячеек памяти и т. д. Приведем примеры с директивой DW.

```
Метка     Код     Операнд      Комментарий
ADDR:     DW      0FF00H       ; (ADDR) = 00H,
                               ; (ADDR+1) = FFH
DATA:     DW      100H,        ; Инициализируются4 ячейки
                  200H
```

Директива DS. Директива DS (определить память) имеет следующий формат:

```
Метка       Код     Операнд          Комментарий
[метка:]    DS      <выражение>      ; Формат директивы DS
```

Вычисленное значение выражения из поля операнда определяет число ячеек (байт) памяти, резервируемых для запоминания данных. Никакие значения в этих ячейках не запоминаются, в частности, нельзя считать, что эти ячейки содержат нули. Адрес следующего оператора равен сумме адреса оператора, находящегося перед директивой DS и значения выражения директивы DS. Примеры использования директивы DS:

```
Метка     Код     Операнд      Комментарий
ARRAY:    DS      32           ; Резервируются 32 ячейки
TABLE:    DS      64           ; Резервируются 64 ячейки
```

Для улучшения внешнего вида и удобства документирования листинга в Ассемблерах могут применяться следующие директивы:

SPC (пропуск строки), которая означает, что при печати листинга необходимо пропустить одну строку;

PAGE (страница), которая при печати листинга вызывает переход на следующую страницу;

TITLE (заголовок), которая вызывает переход на следующую страницу и печать сверху страницы заголовка программы, введенного программистом.

1.2.5. *Макрокоманды*

При программировании возникают ситуации, когда одно и то же действие, реализуемое с небольшими модификациями и задаваемое группой команд, необходимо выполнять в программе несколько раз. Для сокращения длины входной программы и ускорения программирования в Ассемблерах такую группу команд допускается определить 1 раз как большую команду – макрокоманду – с уникальной мнемоникой, не входящей в систему команд

микропроцессора. После определения макрокоманды в начале программы ее мнемонику можно использовать сколько угодно раз так, как будто она включена в систему команд микропроцессора. Всякий раз программа-Ассемблер заменяет эту мнемонику той последовательностью команд, которая фигурирует в определении макрокоманды. Таким образом, введение макрокоманд придает Ассемблеру некоторые черты языков высокого уровня.

Рассмотрим сначала общие принципы использования макрокоманд на простых примерах. Пусть в прикладной программе несколько раз потребовалось передавать содержимое регистров (D, E) в указатель стека SP. Одной командой осуществить такую передачу невозможно из-за отсутствия соответствующей команды в системе команд микропроцессора. Необходимая передача реализуется тремя командами XCHG, SPHL, XCHG. Они не модифицируют больше ни одного внутреннего регистра микропроцессора и не изменяют значений флажков. Определим приведенную последовательность команд как макрокоманду с мнемоникой SPDE:

```
Метка    Код      Операнд    Комментарий
SPDE     MACRO               ; Макрокоманда  SPDE
         XCHG                ; Макрокоманда  SPDE
         SPHL                ; передает содержимое (D, E)
         ENDM                ; в указатель стека
          ***
         SPDE                ; Передача из (D, E) в SP
          ***
         SPDE                ; Передача из (D, E) в SP
          ***
```

Отметим, что определяемая мнемоника SPDE фигурирует в поле метки, а MACRO и ENDM являются специальными директивами Ассемблера. После определения макрокоманды SPDE ее можно использовать как обычную команду, помещая в поле мнемоники всякий раз, когда возникает необходимость выполнения передачи SP<=(D, E).

Определим далее макрокоманду SHV, которая сдвигает содержимое аккумулятора вправо с записью в освобождающиеся биты нулей, считая, что число сдвигов содержится в регистре D:

```
Метка    Код      Операнд    Комментарий
 SHV     MACRO
LOOP:    RRC                 ; Макрокоманда SHV
         ANI      7FH        ; простого сдвига
         DCR      D          ; аккумулятора вправо
         JNZ      LOOP       ; Константа сдвига в регистре D
         ENDM
```

В следующем фрагменте содержимое ячейки с адресом ITEM сдвигается вправо на 3 бита:

```
Метка     Код      Операнд     Комментарий
          LDA      ITEM        ; Загрузка в аккумулятор
          MVI      D, 3        ; Задание числа сдвигов
          SHV                  ; Сдвиг на 3 бита
          STA      ITEM        ; Запоминание в памяти
```

При выполнении данного фрагмента модифицируются значение флажка C (в команде RRC) и содержимое регистра D, являющегося счетчиком сдвигов. Можно расширить возможности макрокоманды SHV, если в определении ее не указывать ни конкретный регистр-счетчик, ни число сдвигов, а задавать их при каждом обращении к макрокоманде или вызове макрокоманды. Такая возможность достигается путем введения макрокоманд с формальными параметрами. Макрокоманда SHV с двумя параметрами (регистр-счетчик REG, число сдвигов VOL) определяется следующим образом:

```
Метка     Код      Операнд     Комментарий
SHV       MACRO    REG, VOL    ; Пример определения
          MVI      REG, VOL    ; макрокоманды SHV
LOOP:     RRC                  ; с двумя параметрами
          ANI      7FH         ; (аргументами)
          DCR      REG
          JNZ      LOOP
          ENDM
```

Теперь при обращении к SHV в разных местах прикладной программы можно использовать в качестве счетчика любой свободный внутренний регистр и задавать варьируемое число сдвигов. Например, для сдвига содержимого ячейки ITEM на 5 бит с регистром-счетчиком L следует записать:

```
Метка     Код      Операнд     Комментарий
          LDA      ITEM        ; Сдвиг содержимого
          SHV      L,5         ; ячейки с адресом
          STA      ITEM        ; ITEM на пять разрядов вправо
```

Еще один пример обращения к SHV для сдвига содержимого ячейки VALUE на 2 бита с регистром-счетчиком E:

```
Метка       Код       Операнд      Комментарий
            LDA       VALUE      ; Сдвиг содержимого
            SHV       E, 2       ; ячейки с адресом
            STA       VALUE      ; VALUE на 2 разряда влево
```

1.2.6. *Структуры данных*

Одной из важных предпосылок успешного проектирования МП-систем является четкое представление характера данных, которыми оперирует прикладная программа. Создатель языка ПАСКАЛЬ Н. Вирт в своей книге «Алгоритмы + структуры данных = программы» (Prentice Hall, 1976) писал: «...все интуитивно чувствуют, что данные предшествуют алгоритмам: необходимо иметь некоторые объекты, прежде чем оперировать ими».

Под структурами данных понимают наборы некоторым образом организованных данных. Характер организации может быть простым, например, когда элемент в массиве определяется своим номером (индексом), или сложным, например, когда элемент двусвязного списка вместе с собственно данными должен содержать указатели преемника и предшественника. Программисту необходимо знать наиболее распространенные структуры данных, способы хранения их в памяти МП-систем и эффективного использования в прикладных программах. Правильный выбор структуры данных позволяет уменьшить объем памяти и увеличить производительность МП-системы. Пока в МП-системах применяются простые структуры данных, но расширение возможностей микропроцессоров позволяет использовать все более сложные структуры данных. В данном подразделе кратко рассматриваются основные структуры данных и приводятся программы некоторых операций с ними.

Массивы. Наиболее простой и распространенной структурой данных является одномерный массив. Он представляет собой набор элементов данных (записей) одинаковой длины, который размещается в области смежных ячеек памяти с начальным (базовым) адресом BASE. Число элементов в массиве называется его длиной. Положение любого элемента в массиве характеризуется его порядковым номером, называемым индексом или смещением. Адрес элемента равен сумме базового адреса BASE и индекса IND, умноженного на длину элемента в байтах. На практике целесообразно применять массивы, длина элементов которых равна степени двух. Тогда при вычислении адреса элемента операция умножения заменяется сдвигами. В наиболее простых массивах длина элементов составляет 1 байт. Формирование и обработка массивов осуществляются циклическими программами, состоящими из трех частей: инициализации, обращения к текущему элементу, перехода к следующему элементу и проверки условия окончания цикла. В программах используются два важных регистра:

регистр, хранящий адрес текущего элемента и называемый указателем POINTER (сокращенно **PTR);**

регистр-счетчик, фиксирующий окончание цикла после обработки последнего элемента массива.

В микропроцессоре I8080 для операций с массивам удобно применять косвенную адресацию, где функции указателя выполняют регистры (H, L). Приведем простую программу поиска максимального числа в массиве 8-битных целых чисел. Длина массива хранится в ячейке с адресом LENGT, а в качестве счетчика используется регистр B.

```
Метка     Код     Операнд     Комментарий
          LDA     LENGT       ; Инициализация счетчика
          MOV     B, A
          LXI     H, BASE     ; Инициализация указателя
NEWMX:    MOV     A, M
NEXT:     DCR     B           ; Проверка окончания цикла
          JZ      DONE
          INX     H           ; Переход к следующему элементу
          CMP     M           ; Сравнение его с максимумом
          JC      NEWMX       ; Новый максимум
JMP NEXT                      ; Следующий элемент меньше
DONE:     ***                 ; Максимум в аккумуляторе
```

Сначала первый элемент массива принимается в качестве максимального, а затем каждый следующий элемент сравнивается с ним. Если текущий элемент больше ранее найденного максимума, он замещает его в аккумуляторе.

Когда длина элементов больше одного байта, необходимо считывать из массива соответствующее число байт и правильно модифицировать указатель. Следующая программа вычисляет в регистрах (D, E) сумму 16-битных элементов массива, представляющих собой целые без знака.

```
Метка     Код     Операнд     Комментарий
          LDA     LENGT       ; Инициализация счетчика
          MOV     B, A
          LXI     D, 0        ; Начальное значение суммы
ADDW:     MOV     A, M        ; Суммирование
          ADD     E           ; младшего
          MOV     E, A        ; байта
          INX     H           ; Указатель на старший байт
          MOV     A, M        ; Суммирование
          ADC     D           ; старшего байта
          MOV     D, A        ; (с учетом переноса)
```

```
        INX     H       ; Следующий элемент
        DCR     B       ; Проверка окончания цикла
        JNZ     ADDW
DONE:   ***             ; Сумма в регистрах (D, E)
```

Отметим, что устанавливаемый при сложении младших байт флажок переноса C не модифицируется командами INX и MOV и учитывается при сложении старших байт с помощью команды ADC.

Иногда при обработке массивов указатель и счетчик приходится хранить в ячейках памяти, например с адресами PTR и COUNT. В такой ситуации обращение к элементу и модификация указателя выполняются последовательностью команд LHLD PTR; MOV A, M; INX H; SHLD PTR. Декремент счетчика осуществляется через аккумулятор: LDA COUNT; DCR A; STA COUNT.

Более общую структуру данных представляет двумерный массив. Каждый элемент двумерного массива характеризуется двумя индексами: номером строки I и номером столбца J. Однако в памяти двумерный массив хранится в линейно-упорядоченных смежных ячейках, что несколько усложняет обращение к текущему элементу. В массиве из 1-байтных элементов общий индекс IND, который суммируется с базовым адресом, зависит от способа размещения массива:

IND = I *Nr+J – если массив хранится по строкам;

IND = I+J * Nc– если массив хранится по столбцам.

Здесь N_r и N_c – число строк и столбцов в массиве (нумерация строк и столбцов начинается с нуля).

Когда длина элементов массива превышает 1 байт, значение IND следует умножить на длину.

Обработка двумерных массивов осуществляется программами, содержащими два цикла. Во внешнем цикле производится инкремент I, а во внутреннем (вложенном) – инкремент J.

При необходимости можно организовать массивы большей размерности. Вычисление общего индекса IND несколько усложняется, так как в памяти многомерные массивы по-прежнему хранятся линейно. Программы обработки многомерных массивов имеют соответствующее число вложенных циклов.

Очереди. Очередь представляет собой структуру данных, в которой элементы можно исключать только с одного конца (начало очереди), а включать только с другого (конец очереди). Главная особенность очереди заключается в том, что она сохраняет порядок элементов неизменным. Число элементов в очереди называется ее длиной.

Обычно очередь иллюстрируют вереницей людей, ожидающих обслуживания. В машинной реализации очереди удобно сохранять элементы неподвижными, а ввести два указателя: начала и конца очереди. Первый адре-

сует элемент, подлежащий исключению из очереди, а второй – последний элемент в очереди или, что иногда более удобно, ячейку сразу за последним элементом.

Наиболее часто в МП-системах очереди используются при вводе и выводе символьных данных. Для очереди выделяется некоторая область смежных ячеек памяти, число которых определяется максимально возможной длиной очереди. При организации очереди необходимо учитывать два особых случая. Попытка включить элемент в очередь, все ячейки которой уже заняты, называется переполнением, а попытка исключения элемента из пустой очереди называется антипереполнением. Обнаружение особых случаев значительно упрощается, если область очереди размещается на одной странице памяти, т.е. старшие 8 бит адресов всех ячеек очереди постоянны и сравнивать адреса можно по младшим 8 бит.

Для исключения из очереди необходимо считать элемент, адресуемый указателем начала очереди, а затем произвести инкремент этого указателя (предполагается, что очередь «растет вниз», в область больших адресов). Включаемый в очередь элемент записывается в ячейку, адресуемую указателем конца очереди (удобнее, если этот указатель адресует первую свободную ячейку), после чего производится инкремент данного указателя. Разумеется, неограниченно увеличивать оба указателя нельзя. Чтобы очередь находилась в пределах нижней границы BQ и верхней границы TQ, применяется круговая (кольцевая) организация. Смысл ее заключается в том, что как только любой из указателей после инкремента достигает значения BQ+1, он модифицируется на TQ. Другими словами, ячейки TQ и BQ становятся в некотором роде как бы смежными.

В приводимых ниже подпрограммах включения в очередь INPUT и исключения из очереди OUTPT предполагается, что очередь размещается в пределах одной страницы XX. Указатели начала и конца очереди хранятся в ячейках памяти с адресами FQ и EQ соответственно (каждый из указателей хранится в двух смежных ячейках). Если после выполнения операции фиксируется особый случай, подпрограммы возвращают установленный флажок Z. После этого операцию, аналогичную предыдущей, выполнять нельзя:

```
Метка    Код      Операнд     Комментарий
                              ; Включаемый элемент находится в
                              аккумуляторе
INPUT:   LHLD     EQ          ; Указатель конца очереди в (H, L)
         MOV      M, A        ; Включение в очередь
         INX      H           ; Инкремент указателя конца
         MVI      A, BQ+1     ; Проверка нижней границы
         CMP      L           ; Достигнута нижняя граница ?
         JNZ      NON         ; Нет, модифицировать не нужно
         MVI      L, TQ       ; Да, модификация указателя
```

```
NOT:    LDA     FQ      ; Проверка переполнения
        SUB     L       ; очереди
        SHLD    EQ      ; Запоминание указателя конца
        RET
                        ; Исключаемый элемент возвращается в аккумулятор
OUTPT:  LHLD    FQ      ; Указатель начала очереди в (H, L)
        MOV     B, M    ; Считывание элемента
        INX     H       ; Инкремент  указателя начала
        MVI     A, BQ+1 ; Проверка нижней границы
        CMP     L       ; Достигнута нижняя граница ?
        JNZ     NON     ; Нет, модифицировать не нужно
        MVI     L, TQ   ; Да, модификация указателя
NON:    LDA     EQ      ; Проверка антипереполнения
        SUB     L       ; очереди
        SHLD    FQ      ; Запоминание указателя начала
        MOV     A, B    ; Передача элемента в аккумулятор
        RET
```

Стек. Стек представляет собой, как и очередь, специальную разновидность одномерного массива. Осуществлять загрузку элементов данных в стек и извлечение их из стека можно только с одного конца, называемого верхушкой (верхом) стека. Таким образом, в любой момент времени из стека можно считать только элемент, находящийся в его верхушке и представляющий собой последние загруженные в стек данные. Ячейку, выполняющую роль верхушки стека, адресует указатель стека SP.

В микропроцессоре I8080 имеется аппаратный указатель стека и все команды, связанные с загрузкой данных в стек и извлечением их из стека, сопровождаются автоматической модификацией SP. До использования стека необходимо инициализировать SP командой LXI SP на максимальный адрес области оперативной памяти, выделенной для стека.

В операциях со стеком могут возникнуть те же особые случаи, что и при использовании очереди: попытка загрузить данные в ячейку, находящуюся за минимальным адресом области стека, называется переполнением, а попытка извлечения данных из ячейки, адрес которой больше максимального адреса области стека, называется антипереполнением. В микропроцессоре I8080 не предусмотрено специальных средств контроля границ стека; ответственность за это возлагается на программиста. Следует помнить, что команды вызовов и возвратов используют стек автоматически и что в каждой операции загрузки и извлечения данных участвуют 16-битные слова.

При программировании иногда удобно использовать то обстоятельство, что извлечение данных из стека является неразрушающим. Пусть, например, командой POP H произведены извлечение данных из стека и передача их в

регистры (H, L). Если произвести декремент SP двумя командами DCX SP, то те же самые данные можно затем передать в любую регистровую пару командой POP rp.

Доступ к SP производится через регистры (H, L).Когда требуется запомнить содержимое SP в двух ячейках памяти, адрес первой из которых есть STPTR, выполняется следующая последовательность команд: LXI H, 0; DAD SP; SHLD STPTR. Загрузка SP из указанных ячеек осуществляется двумя командами LHLD STPTR; SPHL.

При необходимости несложно реализовать программный стек со своим указателем. Если, например, адрес верхушки стека хранится в двух ячейках памяти, адрес первой из которых есть STACK, то операция PSHST загрузки в стек содержимого аккумулятора и операция POPST извлечения данных из стека и передача их в аккумулятор реализуются следующими программами:

```
Метка     Код      Операнд     Комментарий
                               ; Включение в стек содержимого
                               аккумулятора
PSHST:    LHLD     STACK       ; Загрузка указателя стека в (H, L)
          DCX      H           ; Декремент указателя
          MOV      M, A        ; Загрузка в стек
          SHLD     STACK       ; Запоминание указателя стека
                               ; Исключение из стека
                               в аккумулятор
POPST:    LHLD     STACK       ; Загрузка указателя
          MOV      A, M        ; Извлечение из стека
          INX      H           ; Инкремент указателя стека
          SHLD     STACK       ; Запоминание указателя стека
```

Стек с двумя концами называется полкой. Один из концов считается верхом, а другой низом полки. Данные можно включать в полку и исключать из полки с любого конца по принципу LIFO. Следовательно, при организации полки необходимо ввести два указателя и предусмотреть выполнение четырех операций. Очевидно, операции загрузки и извлечения для верха полки аналогичны соответствующим операциям обычного стека: загрузка сопровождается автоматическим декрементом, а извлечение – инкрементом указателя верха полки. Для загрузки в низ полки следует производить инкремент, а для извлечения – декремент указателя низа полки. В микропроцессоре I8080 полку можно организовать только программно.

1.2.7. *Команды управления микропроцессором*

Однобайтные команды данной, обычно немногочисленной, группы применяются для задания режима работы микропроцессора.

Команда останова HLT вызывает прекращение выполнения программы и переводит микропроцессор в состояние останова.

Команда разрешения прерываний EI устанавливает внутренний триггер INTE разрешения прерываний в такое состояние (INTE = 1), когда микропроцессор реагирует на запросы прерываний, поступающие от периферийных устройств, инициирующих обмен данными.

Команда запрещения (маскирования) прерываний DI устанавливает вышеупомянутый триггер в такое состояние (INTE = 0), когда микропроцессор не воспринимает запросов прерываний от периферийных устройств.

Наконец, своеобразная холостая команда NOP не производит никаких действий, кроме инкремента программного счетчика для перехода к следующей команде. Команда NOP обычно используется в так называемых программных циклах задержки, в которых микропроцессор генерирует сигналы программируемой длительности.

2. Проектирование микропроцессорных устройств

2.1. Разделение проблемы на части и алгоритмизация

Большинство современных применений МП-систем связано с решением довольно сложных проблем, которые в недалеком будущем должны еще более усложняться. Поэтому целесообразно разделить общую проблему на простые и управляемые части. Программная реализация каждой из частей называется блоком (модулем). Блоки могут быть сложными, например, арифметические операции в формате с плавающей точкой, или простыми, например, преобразование формата данных. Сложные блоки разделяются на меньшие субблоки до такого уровня, чтобы разработка алгоритмов работы каждого из них стала достаточно простой или почти очевидной.

Основные блоки содержат управляющий блок (основная программа), блоки интерфейса с периферийными устройствами, блоки реакций на прерывания и блоки собственно программных функций, например, поиска данных, сортировки, математических преобразований и т. п. Особое внимание уделяется организации ввода–вывода с учетом особенностей команд ввода–вывода, реакций на прерывания, способов идентификации прерывающих устройств. Интерфейсы с быстродействующими периферийными устройствами организуются по способу прямого доступа к памяти.

Большое значение имеет правильная организация данных. Следует задать и описать форматы входных и выходных величин, промежуточных и окончательных результатов, возможные варианты упаковки данных, выбрать организацию данных в памяти. Целесообразно каким-либо образом упорядочить данные в виде таблиц, массивов, списков и т. п. Рациональная организация данных может сократить длину прикладной программы и/или уменьшить время ее выполнения.

Далее находятся приемлемые компромиссные решения между аппаратными и программными средствами. Например, умножение можно реализовать подпрограммой, которая выполняется сравнительно медленно, но занимает всего несколько байт программной памяти. Кроме того, для умножения можно применить специализированную БИС, которая формирует произведение сомножителей гораздо быстрее подпрограммы. По возможности в разрабатываемую программу следует включать уже готовые и отлаженные подпрограммы.

После выделения функциональных блоков разрабатываются алгоритмы их работы в виде четкой последовательности шагов. В принципе, алгоритмы составляются безотносительно к конкретному языку и микропроцессору, но уже здесь желательна ориентация на конкретный микропроцессор. Логику алгоритмов удобно представлять графическими блок-схемами такого уровня, при котором отдельные их элементы соответствуют нескольким машинным командам. При разработке алгоритмов большое значение имеют опыт и квалификация программиста, и поэтому можно дать только общие рекомендации, которые в основном, сводятся к следующему:

– подробно рассмотреть функциональное назначение блока;

– проанализировать источники данных, их форматы и возможности регуляризации;

– рассмотреть необходимость предварительной обработки данных, например, масштабирование, переход к относительным единицам, упаковка и т. п.;

– алгоритмизировать те преобразования данных, которые должен выполнять блок, с максимальным использованием готовых подпрограмм;

– определить форматы выходных данных и рассмотреть сопряжение блока с получателями данных.

Разработка алгоритмов занимает при проектировании прикладных программ значительную часть времени. Она представляет собой итеративную процедуру, когда для достижения необходимого результата приходится делать несколько пробных попыток. Значительное внимание приходится уделять тем временным ограничениям, которые характерны для большинства прикладных программ. Во многих случаях производительность МП-системы можно существенно повысить за счет увеличения объема памяти. Например, вычисление многих функций традиционно реализуется путем разложения их в ряд с достаточным для требуемой точности числом членов. Другой более быстродействующий способ заключается в применении таблиц, хранимых в памяти.

Следует тщательно спланировать программу и обратить самое серьезное внимание на ее документирование. Обычно программа начинается с заголовка-комментария, в котором сжато описываются основные характеристики программы: имя, дата составления, версия, требования к памяти и особенности ВВ. После заголовка следуют директивы определения непосредст-

венных данных, назначения адресов портам ВВ, директивы резервирования памяти, таблицы переходов и подпрограммы ВВ. Далее следует основная программа, представляющая собой последовательность инициализаций и вызовов подпрограмм. Подпрограммы размещаются с учетом их уровней вложения. Завершается программа таблицами данных, если они есть. Для наглядного представления размещения элементов программы рекомендуется построить так называемую карту памяти.

При составлении карты памяти целесообразно назначать области в пределах страниц, что упрощает дешифрирование и манипуляции адресами.

Проверка и отладка. После кодирования следуют тесно связанные друг с другом этапы проверки и отладки программы, заключающиеся в локализации ошибок и удалении их из программы. Практика показывает, что на отладку программы часто уходит больше времени, чем на ее проектирование и кодирование.

Проверка начинается с тщательного просмотра программы с последующим переходом к отладке подпрограмм низшего уровня, которые не вызывают других подпрограмм. Следует убедиться, что кодирование соответствует логике блок-схемы, что данные правильны, а циклы правильно инициализируются и оканчиваются. Необходимо разработать контрольные примеры (тесты) для проверки функционирования каждого модуля. В тестах предусматриваются наихудшие случаи и предельные значения параметров. Закончив отладку простейших подпрограмм, можно переходить к подпрограммам следующего уровня и так далее до основной программы. Большое внимание следует уделить передаче подпрограммам параметров.

Документирование. Каждый этап реализации прикладных программ должен сопровождаться тщательным и полным документированием. Комплект документации обычно включает: рабочее описание программы, карту памяти, организацию ввода–вывода, способы передачи параметров подпрограммам, листинг программы с комментариями, блок-схему, тестовые примеры и результаты.

3. Методические указания по выполнению контрольных работ

3.1. Содержание контрольной работы

Контрольная работа №2

Включает три теоретических вопроса по программе 2-го семестра курса ЦИМПУ и одну задачу из этого же курса по проектированию цифрового устройства, реализуемого, в отличие от контрольной работы №1, на микропроцессорной платформе программным методом в соответствии с вариантом, задаваемым в таблице.

В задачу входят следующие элементы:

– ввод исходных данных;

– запоминание данных;

– предварительная обработка;

– построение математических моделей проектируемых устройств;

– программирование этих моделей на Ассемблере;

– формирование заданных форматов полученной информации;

– организация вывода их из микропроцессорной системы.

Генерация исходных данных (Xi) предполагается по схеме двоичного датчика.

Аргумент Xi представляется одним битом, записываемым в нулевой разряд аккумулятора с помощью команды IN.

Запоминание данных представляет собой формирование одномерных массивов в оперативной памяти микропроцессорной системы.

Предварительная обработка – это вычисление инверсных значений Xi и их запоминание.

Построение математических моделей проектируемых устройств – вычисление их логистических функций [18].

Программирование этих моделей на Ассемблере заключается в вычислении дизьюнктивных нормальных форм с помощью операций логического сложения ORA и логического умножения ANA.

Формирование заданных форматов полученной информации производится в том случае, когда требуется вывод информации в параллельном формате. Этот формат формируется из значений Yi с помощью операций циклического сдвига и логического суммирования.

Таблица

Вариант	Теоретические вопросы	Практические задания
1	2	3
1	1, 16, 31	1
2	2, 17, 32	2
3	3, 18, 33	3
4	4, 19, 34	4
5	5, 20, 35	5
6	6, 21, 36	6
7	7, 22, 37	7
8	8, 23, 38	8
9	9, 24, 39	9
10	10, 25, 40	10
11	11, 26, 41	11
12	12, 27, 42	12
13	13, 28, 43	13
14	14, 29, 44	14
15	15, 30, 45	15
16	10, 29, 37	3
17	6, 28, 36	6
18	7, 27, 35	9

Окончание таблицы

1	2	3
19	8, 26, 34	12
20	9, 25, 33	15
21	10, 24, 32	1
22	11, 23, 31	4
23	12, 22, 30	7
24	13, 21, 38	10
25	14, 20, 39	13
26	15, 19, 40	2
27	5, 18, 41	5
28	6, 17, 42	8
29	7, 16, 43	11
30	8, 15, 44	14

3.2. Теоретические вопросы для контрольной работы

1. Микропроцессоры (МП). Основные понятия и определения.
2. Основные характеристики микропроцессоров.
3. Типы микропроцессоров.
4. Сигнальные микропроцессоры.
5. Представление данных в микропроцессорных системах .
6. Двоичная система счисления.
7. Восьмеричная система счисления.
8. Шестнадцатеричная система счисления.
9. Перевод данных из двоичной системы счисления в шестнадцатеричную.
10. Перевод данных из шестнадцатеричной системы в двоичную.
11. Основные составные элементы микропроцессоров и микропроцессорных систем.
12. Структура типового микропроцессора.
13. Арифметико-логическое устройство (АЛУ) микропроцессора.
14. Устройство управления (УУ) микропроцессора.
15. Регистры общего назначения (РОН) микропроцессора
16. Аккумулятор.
17. Регистровые пары в микропроцессоре I8080.
18. Структура типовой микропроцессорной системы.
19. Процессор.
20. Оперативное запоминающее устройство (ОЗУ) в микропроцессорной системе.
21. Устройства ввода-вывода (УВВ) в микропроцессорной системе.
22. Системная магистраль в микропроцессорной системе.

23. Память микропроцессорной системы .

24. Единицы измерения информации.

25. Регистровая память.

26. Стековая память.

27. Запоминающее устройство (ЗУ) с произвольной выборкой и последовательным доступом.

28. Организация и функционирование памяти в микропроцессорной системе.

29. Программное обеспечение микропроцессорной системы.

30. Операционная система.

31. Организация ввода-вывода в микропроцессорной системе.

32. Система прерываний в микропроцессорной системе.

33. Система команд микропроцессора.

34. Форматы команд микропроцессора.

35. Режимы адресации микропроцессора.

36. Типы команд.

37. Ассемблер микропроцессора Intel 8080.

38. Структура команды микропроцессора Intel 8080.

39. Команды и директивы Ассемблера микропроцессора Intel 8080.

40. Структура программы.

41. Программирование операций обмена с внешними устройствами микропроцессорных систем.

42. Особенности применения микропроцессоров в системах телекоммуникаций.

43. Сопряжение микропроцессорных систем с каналами связи.

44. Понятие о протоколах обмена данными.

45. Повышение помехоустойчивости каналов передачи данных.

3.3. Практические задания для контрольной работы

1. Разработать программу на Ассемблере, реализующую логическую функцию 4-разрядного дешифратора с инверсными выходами.

2. Разработать программу на Ассемблере, реализующую логическую функцию демультиплексора 1-16.

3. Разработать программу на Ассемблере, реализующую логическую функцию шифратора, преобразующего унитарный инверсный код 0-7 в прямой двоичный код.

4. Разработать программу на Ассемблере, реализующую логическую функцию, выполняемую с помощью дешифратора КР1533ИД7 и элементов И-НЕ.

5. Разработать программу на Ассемблере, реализующую логическую функцию приоритетного шифратора с инверсными входами и выходами 16-4 на двух ИМС КР555ИВ1 и логических элементах той же серии.

6. Разработать программу на Ассемблере, реализующую логическую функцию мультиплексора 16-1, построенную на базе ИМС КР1533КП2.

7. Разработать программу на Ассемблере, реализующую логическую функцию, реализуемую на базе мультиплексора КР1533КП2 и JK-триггера КР1533ТВ9.

8. Разработать программу на Ассемблере, реализующую логическую функцию на одном мультиплексоре КР1533КП2.

9. Разработать программу на Ассемблере, реализующую логическую функцию, выполняемую 4-разрядным сумматором К555ИМ6.

10. Разработать программу на Ассемблере, реализующую логическую функцию, выполненную на 4-разрядном сумматоре К555ИМ6 одно и двух-разрядного суммирования.

11. Разработать программу на Ассемблере, реализующую логическую функцию, выполненную на основе КР1533ТВ9 и логических элементах той же серии и генерирующую код 7-2-4-11.

12. Разработать программу на Ассемблере, реализующую логическую функцию, обычно выполняемую на основе двух ИМС КР1533ТВ15 и логических элементах той же серии и построить генератор чисел 7-1-3-12.

13. Разработать программу на Ассемблере, реализующую логическую функцию, выполняемую на КР1533ТМ2 и ИМС той же серии 3-разрядный счетчик, который при управляющем сигнале M = 1 работает как счетчик в коде Грея, а при M = 0 – как двоичный суммирующий счетчик.

14. Разработать программу на Ассемблере, реализующую логическую функцию асинхронного счетчика на основе КР1533ТМ2 и логических элементов той же серии.

15. Разработать программу на Ассемблере, реализующую логическую функцию, выполняемую регистром сдвига на базе ИМС КР1554ТМ8 и генерирующего двоичные последовательности 1-0-1-1-0.

Литература

1. Пирогов, В. Ю. Ассемблер для Windows / В. Ю. Пирогов. – СПб. : БХВ-Петербург, 2005. – 862 с.

2. Пильщиков, В. Н. Программирование на языке Ассемблера IBM PC / В. Н. Пильщиков. – М. : Диалог-МИФИ, 1994 . – 288 с.

3. Крупник, А. Б. Изучаем Ассемблер / А. Б. Крупник. – М. : Питер, 2005. – 249 с.

4. Юров, В. И. Assembler. Практикум / В. И. Юров. – М. : Питер, 2006 . – 398 с.

5. Бурдаев, О. В. Ассемблер в задачах защиты информации / О. В. Бурдаев, М. А. Иванов, И. И. Тетерин. – М. : Кудиц-образ, 2004. – 538 с.

6. Гурский, А. Л. Проектирование микропроцессорных устройств : метод. пособие к лаб. работам по курсу «Цифровые и микропроцессорные устройства» для студ. спец. 1-45 01 03 «Сети телекоммуникаций» / А. Л. Гурский, С. М. Лапшин. – Минск : БГУИР, 2003. – 48 с.

7. Токхайм, Р. Микропроцессоры. Курс и упражнения / Р. Токхайм : пер. с англ.; под ред. В. Н. Грасевича. – М. : Энергоатомиздат, 1988. – 336 с.

8. Каган, Б. М. Основы проектирования микропроцессорных устройств автоматики / Б. М. Каган, В. В. Сташин. – М. : Энергоатомиздат, 1987. – 304 с.

9. Шпота, С. Д. Изучение архитектуры и структуры однокристального микропроцессора (на базе микросхемы КР580ИК80) : лаб. практикум по курсу «Цифровые устройства и устройства цифровой техники» / С. Д. Шпота, Н. А. Ломако – Минск : БГУИР, 1998. – 66 с.

10. Лапшин, С. М. Проектирование микропроцессорных систем передачи и обработки информации : метод. пособ. по курс. и диплом. проектированию по курсам «Устройства цифровой техники» и «Микропроцессоры в системах телекоммуникаций» для студ. спец. «Телекоммуникационные системы» / С. М. Лапшин. – Минск. : БГУИР, 1997. – 51 с.

11. Точчи, Р. Д. Цифровые системы: Теория и практика. – 8-е изд. / Р. Д. Точчи, Н. С. Уидмер – М. : Вильямс, 2004. – 1024 с.

12. Угрюмов, Е. П. Цифровая схемотехника: учеб. пособие. – изд. 2-е, перераб. и доп. / Е. П. Угрюмов. – СПб. : БХВ – Петербург, 2004. – 782 с.

13. Опадчий, Ю. Ф. Аналоговая и цифровая электроника / Ю. Ф. Опадчий, О. П. Глудкин, А. И. Гуров. – М. : Горячая линия – Телеком, 2005. – 767 с.

14. Справочник по интегральным микросхемам / Б. В. Тарабрин [и др.]; под ред. Б. В. Тарабрина – М. : Энергия, 1980. – 816 с.

15. Интегральные микросхемы: справочник / Б. В. Тарабрин [и др.]; под ред. Б. В. Тарабрина. – М. : Энергоатомиздат, 1985. – 528 с.

16. Аналоговые и цифровые интегральные микросхемы: справ. пособие / С. В. Якубовский [и др.]; под ред. С. В. Якубовского. – М. : Радио и связь, 1985. – 432 с.

17. Угрюмов, Е. П. Проектирование узлов и элементов ЭВМ / Е. П. Угрюмов. – М. : Высш. шк., 1987.

18. Лыньков, Л. М. Разработка логической схемы цифрового устройства : метод. пособие к курс. проектированию по дисц. «Цифровые и микропроцессорные устройства средств измерений» для студ. спец. 1-54 01 01-02 «Метрология, стандартизация и сертификация» днев. формы обуч. / Л. М. Лыньков, В. П. Ширинский. – Минск : БГУИР, 2007. – 48 с.

Св. план 2009, поз. 105

Учебное издание

Лыньков Леонид Михайлович
Гурский Александр Леонидович
Колбун Наталья Викторовна
Ширинский Валерий Павлович

ЦИФРОВЫЕ И МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА

Методическое пособие
для студентов специальности 1-45 01 03
«Сети телекоммуникаций»
заочной формы обучения

Редактор Н. В. Гриневич
Корректор Е. Н. Батурчик

Подписано в печать 16.11.2009. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 2,21.
Уч.-изд. л. 2,0. Тираж 100 экз. Заказ 285.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6