ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

**АППАРАТНОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПРОСТЫХ МИКРОПРОЦЕССОРНЫХ СИСТЕМ**

Методические указания

к курсовой работе по разделу

«Электроника и микропроцессорная техника»

Ростов-на-Дону 2006 г.

**1. Цель работы**

Приобретение практических навыков структурных блок-схем и листингов программ на языке «Ассемблер» для простых микропроцессорных систем (МП-систем) управления различными процессами.

**2. Типовые структуры блок-схем алгоритмов обработки данных**

Для успешного выполнения заданий курсовой работы студентам необходимо ознакомиться с набором команд процессора КР580 /1/, а также с требованиями, предъявляемыми к составлению программ на языке «Ассемблер» /2/, /3/, /4/, предназначенных для микропроцессорных систем автоматического управления различными измерительными и технологическими процессами.

Набор команд микропроцессора является основой для разработки структурной схемы машинных команд. За этой схемой следует только написание и кодирование программы. Поэтому структурная схема должна быть настолько детальной, чтобы каждый блок мог быть представлен не более, чем тремя командами.

Любая структурная схема может быть построена путем комбинации нескольких базовых блоков: функциональных (последовательных), цикла (повторения), разветвления (альтернативного решения).

На рис. 1 представлены типовые блоки структурных схем алгоритмов, которые широко используются в программах обработки данных.

Последовательная структура – самая распространенная (рис. 1, а); она означает, что действия должны быть выполнены друг за другом. Показанная на рис. 1, б структура ЕСЛИ–ТО–ИНАЧЕ применяется в тех случаях, когда необходимо реализовать программный переход к одной из двух вычислительных процедур в зависимости от выполнения некоторого проверяемого условия. Структура ЕСЛИ–ТО (рис. 1, в) является упрощение предыдущей и используется в случаях, когда необходимо реализовать одну вычислительную процедуру в зависимости от проверяемого условия. Структура ДЕЛАЙ–ПОКА используется для проверки условия окончания цикла (рис. 1, г). Структура ПОВТОРЯЙ–ДО–ТОГО–КАК (рис. 1, д) аналогична предыдущей, но порядок следования операторов здесь иной: процедура выполняется до проверки условия. Структура ПРОЦЕСС–ПОКА (рис. 1, е) представляет собой объединение двух предыдущих структур. И, наконец, на рис. 1, ж представлена структура ДЕЛАЙ–В–ЗАВИСИМОСТИ–ОТ, с помощью которой осуществляется выбор действия при многозначных решениях и которая используется для замены цепочек структур ЕСЛИ–ТО–ИНАЧЕ.

Все перечисленные блоки в различных комбинациях встречаются в алгоритмах выполнения программ обработки данных.

**3. Программная реализация типовых функций управления**

При проектировании МП-систем управления различными измерительными или технологическими процессами возникает необходимость программирования таких типовых процедур управления, как

– опрос состояния двоичного датчика;

– ожидание события;

– сканирование группы позиционных датчиков;

– формирование временных задержек;

– отыскание минимального или максимального значения вводимого массива параметров;

– операции сортировки и т.п.

Ниже приводятся некоторые способы программного обеспечения типовых функций управления процессами применительно к МП КР580ВМ80.

На рис. 2. показана схема подключения контакта двоичного датчика к входному порту МП-системы. Если контакт S разомкнут, то на входе D5 порта ввода присутствует сигнал логической единицы; если контакт S замкнут, то на D5 – логический нуль. Необходимо в некоторой части правляющей программы МП-системы оросить значение сигнала на вход D5 порта 04 и в зависимости от его значения (0 или 1) передать управление фрагменту программы с меткой, например, LABEL A (если D5=0) или по адресу, отмеченному меткой LABEL В (если D5=1).

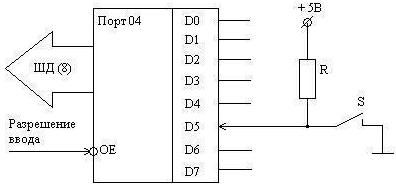


Рис. 2. Схема опроса двоичного кода

На рис. 3, а приведена блок-схема, а на рис. 3, б программа «INPKEY» (ввод ключа), реализующая процедуру опроса двоичного датчика. Символическое имя программы «INPKEY» используется в качестве метки начальной команды этой программы. При программировании с использованием подпрограмм можно обращаться к этой подпрограмме опроса двоичного датчика по команде: CALL, адрес INPKEY.

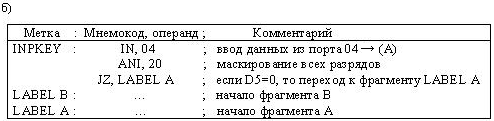


Рис. 3. Блок-схема и листинг программы опроса двоичного датчика

Контроллеры технологических объектов работают в реальном масштабе времени, и, следовательно, их работа определяется событиями, происходящими в объекте управления. Чаще всего события в объекте управления фиксируются с использованием двоичных датчиков; например, путём замыкания или размыкания нулевого переключателя при перемещении исполнительного органа объекта управления.

Если требуется по ходу выполнения управляющей программы приостановить выполнение её команд до тех пор, пока в результате процессов, происходящих в объекте управления, не замкнётся контакт S датчика перемещения, то можно использовать подпрограмму с символическим именем «NUNT» (засада), блок-схема которой приведена на рис. 4, б.

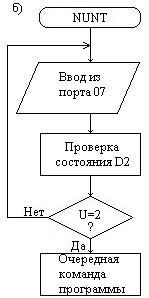


Рис. 4. Схема подключения контакта двоичного датчика к порту ввода МП-системы (а) и блок-схема алгоритма ожидания события

Основная программа МП-системы многократно по команде CALL, адрес NUNT, может вызвать эту подпрограмму. Из блок-схемы алгоритма видно, что программа должна постоянно опрашивать значение сигнала на входе D2 порта 07 до тех пор, пока оно не станет равным нулю (контакт датчика разомкнут), и в этом случае продолжить выполнение основной программы МП-системы. Если переход к циклу ожидания события из основной программы осуществляется по команде CALL, адрес NUNT, то возврат в неё из процедуры NUNT выполняется по команде RET, стоящей в конце подпрограммы.

Аналогичным образом с использованием различных кодов маски в команде ANI можно осуществлять отслеживание множества событий, фиксируемых различными двоичными датчиками, присоединенными к другим входам порта ввода информации.

На рис. 5. показана схема подключения МП-системы к некоторому исполнительному механизму объекта управления через порт вывода информации.

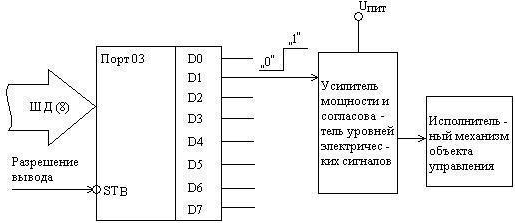


Рис. 5. Схема формирования управляющего сигнала

Предположим, что данный исполнительный механизм работает по принципу «включить–выключить», т.е. может управляться двоичным выходным сигналом МП-системы («1» – включить, «0» – выключить).

Подпрограмма формирования такого управляющего воздействия проста и состоит всего из двух команд. Для включения исполнительного механизма используется подпрограмма «ON»:

ON: MVI A, 02; загрузить в аккумулятор код 000.0010

OUT, 03; выдать управляющий байт в порт 03.

Для выключения исполнительного механизма можно использовать подпрограмму «OFF»:

OFF: XRA A; обнулить аккумулятор

OUT, 03; выдать байт 0000 0000 в порт 03.

(вывести содержимое аккумулятора байт 0000 0000 в порт 03).

В том случае, если к остальным семи выводам выходного порта 03 подсоединяются другие исполнительные механизмы, формируется не двоичное управляющее воздействие, а байт управляющего слова, где каждому разряду ставится в соответствие 0 или 1 в зависимости от того, какие механизмы должны быть выключены или включены.

Программная реализация временной задержки использует метод программных циклов, при котором в какой-либо регистр блока регистров общего назначения (РОН) микропроцессора загружается число, которое при каждом проходе цикла уменьшается на единицу. Так продолжается до тех пор, содержимое регистра-счетчика не станет равным нулю, что интерпретируется программой как момент выхода из цикла. Время задержки при этом определяется числом, загруженным в регистр-счетчик, и временем выполнения команд, образующих цикл. Схема алгоритма такой программы показана на рис. 6.

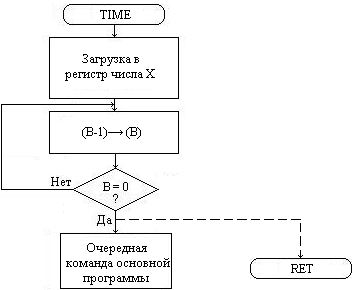


Рис. 6. Блок-схема временной задержки

Программа имеет символическое название «TIME» и, в случае вызова её основной программой по команде CALL, адрес TIME, должна завершаться командой возврата RET.

Предположим, что в МП-системе, использующей тактовую частоту 2 МГц (такт в этом случае составляет 0,5 мкс), необходимо реализовать временную задержку длительностью 250 мкс. Фрагмент программ, реализующей временную задержку, необходимо оформлять в виде подпрограммы, так как предполагается, что основная программа будет обращаться к ней многократно.

Текст программы, отображающей структуру алгоритма, показанного на рис. 6, следующий:

TIME: MVI B, X; загрузка регистра В числом Х

COUNT: DCR B; уменьшение на 1 содержимого

регистра В JNZ, адрес COUNT; повторить цикл, если В≠0

RET; возврат в основную программу, если В=0.

Для получения требуемой временной задержки необходимо определить значение числа Х, загружаемого в регистр В. Определение числа Х выполняется на основе времени выполнения команд, образующих данную подпрограмму. При этом необходимо учитывать, что команды MVI B, X и RET выполняются однократно, а число повторений команд DCR B и JNZ, адрес COUNT равно числу Х, загружаемому в регистр В. Кроме того, обращение к подпрограмме временной задержки осуществляется по команде CALL, адрес TIME, время исполнения которой также необходимо учитывать при подсчете временной задержки. В описании команд МП КР580ИК80 указывается, за сколько тактов основной частоты синхронизации исполняется каждая команда МП. На основе этих данных можно записать:

CALL, TIME – 17 тактов – 8,5 мкс;

MVI B, X – 7 тактов – 3,5 мкс;

DCR B – 5 тактов – 2,5 мкс;

JNZ, адрес COUNT – 10 тактов – 5,0 мкс;

RET – 10 тактов – 5,0 мкс.

Таким образом, однократно исполняемые команды (CALL, MVI, RET) в этой подпрограмме требуют 17 мкс (8,5+3,5+5,0). Следовательно, для получения требуемой задержки в 250 мкс необходимо команды DCR B и JNZ, COUNT столько раз, чтобы время их исполнения составило 233 мкс, т.е. (250–17). Однако время выполнения этой пары команд составляет (2,5+5,0). Поэтому, если принять Х=31, возможно получение временной задержки 232,5 мкс.

Если точность подпрограммной реализации временной задержки длительностью 250 мкс с погрешностью – 0,5 мкс удовлетворяет условиям задачи, то на этом разработка программы заканчивается.

Исходя из приведенного расчета, запишем текст подпрограммы TIME:

TIME: MVI B, 31 ; загрузка в регистр В числа 31

COUNT: DCR B; декремент регистра В

JNZ, адрес COUNT; цикл, если В≠0

RET; возврат в основную программу.

В том случае, если точность представления временного интервала 250 мкс с погрешностью – 0,5 мкс не удовлетворяет разработчика, можно поступить двояко:

– реализовать подпрограмму точной задержки на 50 мкс и пятикратно повторить ее вызов;

– путем внесения в подпрограмму пустых операций NOP и соответствующего изменения набора команд (с целью устранения временного рассогласования 0,5 мкс) обеспечить точную временную выдержку.

Во многих случаях применения МП-систем требуется сформировать длительные временные задержки (секунды, минуты, часы и т.д.). Сделать это при частоте, равной 2МГц с использованием ранее описанного метода невозможно, так как максимальной емкости регистровой пары FFFF не хватит для того, чтобы представить число Х, достаточное для формирования задержки в 1 секунду. Сформировать столь большую для МП задержку можно с использованием метода вложенных циклов (как показано на рис. 7).

С целью получения задержки, равной 1 мин, основная управляющая программа может 60 раз осуществлять вызов подпрограммы ONESEC. Для этого число 60 загружается, например, в регистр В, который выполняет функции декрементного счетчика секунд, и после каждого прогона подпрограммы ONESEC его содержимое уменьшается на 1. Текст программы «ONESEC» приводится ниже.

ONESEC: MVI B, FF; счетчик внешних циклов

L1: MVI C, FB; счетчик внутренних циклов

L2: NOP; точная подгонка времени

внутреннего цикла

NOP;

NOP; время

NOP;

DCR C; декремент счетчика внутренних циклов

JNZ, адрес L2; возврат во внутренний цикл, если С≠0

DCR В; декремент счетчика внешних циклов

JNZ, адрес L1; возврат во внешний цикл, если В≠0

RET

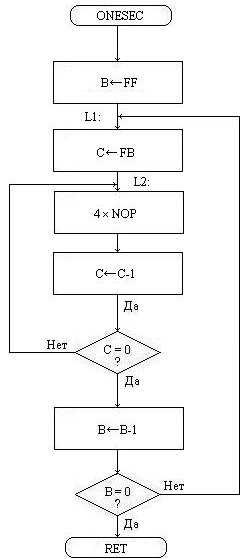


Рис. 7. Алгоритм задержки на 1 секунду

Блок-схема типовой процедуры сбора и формирования в ОЗУ МП-системы массива данных от одного источника показана на рис. 8. Источником вводимых данных является порт ввода с символическим адресом NN, 8100 – начальный адрес массива данных, регистр С используется в качестве счетчика данных и регистровая пара HL используется командами с косвенно-регистровой адресацией в качестве указателя данных; ETX – знак-терминатор «конец массива».

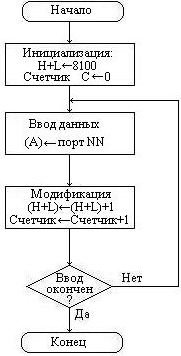


Рис. 8. Типовая процедура сбора данных

Программа имеет вид:

LXI H, 8100; запись начального адреса

8100 → (H+L)

MVI C, C, 00; счетчик = 0

SAVE: IN, NN; ввод данных из порта в А

MOV M, A; перенос данных А → ячейку ОЗУ,

адрес которой в (H+L)

INX H; (H+L) = (H+L+1)

INR C; счетчик = счетчик+1

SUI, ETX; проверка терминатора

JNZ, адрес SAVE; продолжение сбора, если не 0

DONE

Если число слов данных известно и хранится в ячейке с адресом 81N0, то программа сбора данных будет иметь вид:

LXI H, 8100;

LDA, 81N0; пересылка содержимого 81N0 → (A)

MOV C, A; счетчик = длина массива: А → (С)

SAVE: IN, NN;

MOV M, A;

INX H;

DCR C; счетчик = счетчик-1

JNZ, адрес SAVE; продолжение сбора, если не 0

DONE

Рассмотрим несколько примеров обработки массива данных.

LDA COUNT; загрузка содержимого М 8200

(COUNT) → (A)

MOV B, A; загрузка счетчика: (В) ← (А)

LXI H, 8100; запоминание в (H+L) адреса начала

массива данных

SUB A; сброс аккумулятора: (А) ← 0

ADDN: ADD M ; прибавление элемента М+(А) → (A)

INX H; переход к следующему адресу

(H+L) ← (H+L+1)

DCR B; декремент счетчика: (В) ← (В-1)

JNZ, адрес ADDN; организация цикла, если не 0

DONE

LDA COUNT; загрузка содержимого

М (COUNT) → (A)

MOV B, A; организация счета в регистре В:

(В) ← (А)

LXI H;

NEWMX: MOV A, M; загрузка нового максимума

NEXTE: DCR B; декремент счетчика: (В) ← (В-1)

JNZ, адрес DONE; проверка окончания цикла: если 0,

то скачок на адрес метки DONE

INX H; (H+L) ← (H+L+1)

CMP M; сравнение с максимумом

JС, адрес NEWMX;

JМР, адрес NEXTE;

DONE

**Литература**

1. Г.И. Пухальский. Программирование микропроцессорных систем. Учебное пособие для Вузов – М. Политехника, 2002.

2. В.С. Ямпольский. Основы автоматики и электронно-вычислительной техники. – М.: Просвещение, 1991.

3. Л.Н. Ананченко, И.Е. Рогов. Составление алгоритмов и программ на языке «Ассемблер» для управления технологическими процессами: Метод. указания – Ростов-на-Дону: ДГТУ, 1993 г.

4. Л.Н. Ананченко. Набор команд микропроцессора КР580ИК80: Метод. указания – Ростов-на-Дону, РИСХМ, 1991.