Министерство образования и науки Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра телевидения и управления (ТУ)

В.А. Кормилин

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

(1 часть)

Учебно-методическое пособие по организации лабораторных работ

2018

Кормилин В.А.
Вычислительная техника и информационные технологии (1 часть): Учебно-методическое пособие по организации лабораторных работ. Томск: Томский государственный университет систем управления и радиоэлектроники (ТУСУР), 2018. 40 с.

Учебно-методическое пособие предназначено для студентов радиотехнических направлений подготовки ТУСУРа, обучающихся на всех формах обучения и содержит учебный материал и методические указания для организации лабораторных работ входе изучения дисциплины.

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

При разработке комплекса лабораторных работ учитывалась необходимость формирования и оценки достижения общепрофессиональных компетенций, связанных с данной дисциплиной в рабочем учебном плане.

Формируемые знания, умения и навыки связаны со следующими компетенциями:

ОПК-2 – способностью решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением инфокоммуникационных технологий и с учетом основных требований информационной безопасности;

ОПК-3 – способностью владеть основными методами, способами и средствами получения, хранения, переработки информации.

ЛАБОРАТОРНАЯ РАБОТА № 1.

ПРОЦЕДУРА РАЗРАБОТКИ ПРОГРАММ ДЛЯ ОДНОКРИСТАЛЬНЫХ МИКРОКОНТРОЛЛЕРОВ

1 ВВЕДЕНИЕ

Целью работы является начальное изучение этапов разработки, написания, трансляции и отладки программ для однокристальных микроконтроллеров (ОМК), методов обработки данных в микроконтроллерах и способов организации взаимодействия микроконтроллера с объектами управления на примере ОЭВМ MCS-51 (I8051).

2 ОБЩИЕ ПОЛОЖЕНИЯ

2.1 Процедура создания программ для ОМК

Создание и отладка программного обеспечения (ПО) для однокристальных микроконтроллеров считается трудной задачей. Это объясняется закрытым характером процессов обмена информации в ОМК и отсутствием универсальных ЭВМ, построенных на базе микроконтроллеров. Из-за последнего обстоятельства резидентная разработка ПО для ОМК практически не проводится. Поэтому при разработке программ для ОМК используют различные программы – симуляторы и кросс-средства создания, трансляции и отладки программ, работающие на базе универсальных ЭВМ, например IBM PC. Эти ЭВМ имеют несовместимую с ОМК систему команд и поэтому эмулируют работу микроконтроллера.

Наиболее удобно применять специальные пакеты программ, предназначенные для разработки программного обеспечения определенного вида микроконтроллеров, объединенные общей идеологией, общей системой меню и справочной информации. При отсутствии такого пакета используют разрозненные программы, выполняющие необходимые операции.

Необходимый набор программ разработки ПО для микроконтроллеров включает:

- редактор текста, обеспечивающий ввод, редактирование и запись файла программы на языке ассемблера целевого однокристального микроконтроллера;
- кросс – транслятор, преобразующий исходный текст программы в файл объектного кода, содержащего коды команд целевого микроконтроллера, информацию для редактора связей, предупреждения и сообщения об ошибках;

– кросс – редактор связей или компоновщик объектных модулей. Компоновщик связывает несколько объектных модулей в единую программу, добавляет вызываемые из программы библиотечные файлы, подставляет вместо ссылок на объекты их реальные или относительные адреса и формирует исполняемый файл в кодах целевого однокристального контроллера;
– кросс – отладчик или программа – симулятор, эмулирующая работу микроконтроллера, его компонентов: регистров, ячеек памяти, таймеров, портов и т.д.
– справочник по каждой из применяемых программ.

Общая процедура создания и отладки программ для ОМК включает следующие этапы, показанные на рис. 1.

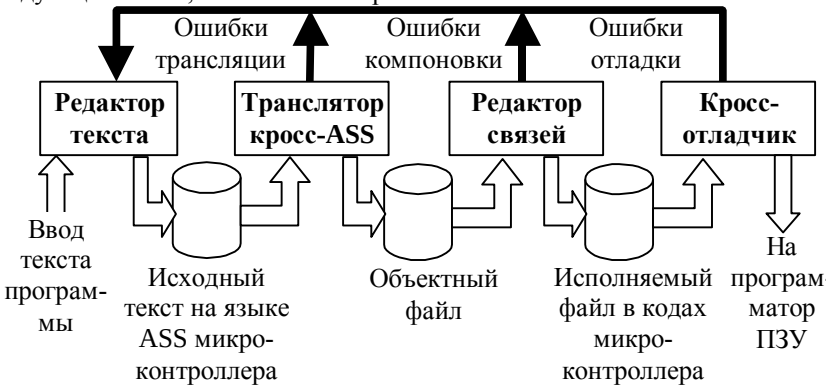

Рисунок 1 – Этапы процедуры создания ПО для ОМК

На каждом из этапов создания ПО могут быть обнаружены различные ошибки, что приводит к необходимости возврата к начальному тексту программы.

2.1.1 Редактор текста

Для создания файла с программой можно использовать любой редактор, позволяющий вводить и записывать в файл текст в виде ASCII (American Standard Code for Information Interchange) символов. Редактор не должен кодировать вводимые символы, записывать служебные комбинации, производить подстановку и переключение фонтов.

Для наших целей наиболее подходят DOS – редакторы текста, например Norton Editor, PE2, NE, ME, Lexicon, редакторы, встроенные в оболочку Norton Commander, FAR и т.д. Можно применить и редакторы системы WINDOWS типа Блокнот или WordPad.

При наборе текста программы следует избегать использования символов табуляции. Они не всегда правильно обрабатываются в трансляторах.

В выбранном текстовом редакторе нужно создать символьный файл программы. При закрытии программы редактора текста нужно не забыть записать созданную программу в виде файла с произвольным именем и расширением *.ASM или *.ASS. Эти расширения являются общепринятыми для файлов на языке ассемблера. Само имя должно быть написано английскими буквами и длина его не должна превышать 8 символов, например, Tablic.ASM (Tablic.ASS).

При использовании программы **«Блокнот»** файл нужно записать в кодировке ANSI. При использовании кодировки Юникод или UTF в файл записываются служебные символы, которые не распознаются транслятором. После вызова в программе «Блокнот» окна сохранения документа «СОХРАНИТЬ КАК…», в поле «ИМЯ ФАЙЛА» необходимо указать английскими буквами желаемое имя с расширением «*.ASM» или «*.ASS». В поле «ТИП ФАЙЛА» желательно указать «ВСЕ ФАЙЛЫ». В этом случае удается сохранить указываемое расширение ASM (ASS). Для файлов с типом «Текстовый» будет добавлено расширение *.TXT, которое придется убирать вручную.

Редактор **WordPad** также позволит создать и записать текст программы. При сохранении тип файла должен быть указан как «Текстовый документ» или «Текстовый документ MS-DOS». У сохраненного файла к предложенному Вами имени и расширению добавится дополнительное расширение *.TXT. С таким дополнительным расширением трансляция программы невозможна. Поэтому после выхода из редактора имя файла придется скорректировать и убрать добавку «.TXT».

При затруднениях в выборе редактора можно использовать несложный в освоении и компактный редактор NE.COM, записанный в рабочем каталоге LANQ. ***Помните, что данный редактор не поддерживает длинные и русские имена файлов и каталогов. Он откажется работать, если в пути его размещения будут русские или длинные имена .***

Для вызова редактора наберите в командной строке DOS команду

>NE

В ответ на приглашение «*Enter file name*» введите имя файла с расширением. Если этот файл уже существует и находится в данном каталоге, он будет открыт. В противном случае будет создан и открыт новый файл с указанным именем.

Ввод текста в редакторе производится обычным образом алфавитно-цифровыми клавишами. Для управления перемещением курсора используются клавиши со стрелками, однако, курсор не будет перемещаться по экрану за пределы занятой символами области.

Вызов команд редактора потребуется только при записи файла, организации поиска в файле, при работе с блоками текста, при работе с двумя ок-

нами. Клавишей [**F1]** можно вызвать текст справки о допустимых командах редактора. Почти все команды начинаются с нажатия одной из функциональных клавиш [**F3]** – [**F7]** и буквы латинского алфавита.

Наиболее важные команды описаны в таблице 1:

Таблица 1

Файловые команды		**Блочные команды**	
F3 E	Запись файла и выход в DOS	F4 S	Установить маркер начала/конца блока
F3 S	Запись файла без выхода	F4 R	Снять маркер блока
F3 Q	Выход без записи файла	F4 L	Пометить всю строку
F3 A	Добавить указанный файл в конец текущего	F4 C	Копировать блок в текущую позицию
F3 X	Включить другое окно и перейти в него	F4 M	Переслать блок в текущую позицию
Команды удаления			
Backspace	Удалить символ слева	DEL	Удалить текущий символ
CTRL U	Вернуть удаленное	F4 D	Стереть выделенный блок

При сохранении файла редактор NE может предложить заменить символы табуляции на пробелы «Should tabs be expanded to spacec? (Y or N)». Остается только согласиться и ввести символ Y.

2.1.2 Транслятор КРОСС - АССЕМБЛЕР 8051

Транслятор **X8051.EXE** является кросс – ассемблером 8051 фирмы "2500 AD SORTWARE ", версии 4.02.

Ассемблер 8051 дает возможность пользователю писать программы, которые впоследствии могут быть скомпонованы в перераспределяемый объектный код и связаны с необходимым адресом выполнения с помощью редактора связей фирмы "2500 AD". Список команд ассемблера 8051 приведен в приложении.

Ассемблер может обрабатывать файл любого размера при условии наличия достаточного объема памяти. Все используемые ассемблером буферы запрашиваются по мере необходимости, за исключением буфера ввода исходного кода, буфера вывода объектного кода и буфера распечатки.

Кросс–ассемблер может работать в режиме подсказок и в режиме командной строки.

Для работы в режиме подсказок введите команду

>X8051.EXE

Программа будет выдавать вопросы – подсказки, возможные варианты ответов на которые приведены в скобках. Ввод [**Enter**] задает ответ по умолчанию.

Начальный запрос связан с местом назначения листинга трансляции:

LISTING DESTINATION ? (N, T, P, D, E, L, <CR>=N):,

Аббревиатуры обозначают:

N = не создавать листинг (выбирается по умолчанию)

T = вывод листинга на терминал

P = вывод на принтер

D = запись листинга на диск

E = выдавать только ошибки

L = распечатка вкл/выкл

После этого ассемблер выдает запрос имени исходного файла:

INPUT FILENAME :

При вводе имени исходного файла можно указать расширение, или же ассемблер будет искать расширение "**ASM**".

Затем ассемблер выдает запрос имени выходного объектного файла:

OUTPUT FILENAME :

При вводе только [**Enter**] файл вывода будет иметь имя входного файла с расширением "**OBJ**". Если ответ представлен в виде имени файла без расширения, то файл вывода будет иметь это имя файла с расширением "**OBJ**".

2.1.3 Редактор связей

Использование редактора связей **LINK.EXE** дает возможность пользователю писать программы на языке ассемблера, состоящие из нескольких модулей. Редактор связей решает вопросы относительно внешних ссылок, а также выполняет перераспределение адресов программы.

Редактор связей может генерировать все, наиболее часто используемые форматы файла, устраняя при этом необходимость использования дополнительной утилиты преобразования формата.

В пределах объема памяти размер файла, обрабатываемого редактором связей, не ограничивается. Редактор связей может обрабатывать комбинацию из 256 входных файлов и модулей библиотек.

Редактор связей может активизироваться в режиме подсказок и режиме командной строки. Выходной формат выбирается из директивы в исходном файле или из перечня вариантов связей. Карта загрузки, алфавитный перечень глобальных символов и все ошибки связей могут сохраняться в дисковом файле.

Для запуска редактора связей в режиме подсказок задайте команду

>LINK.EXE

Редактор связей будет реагировать подсказкой, запрашивающей имя файла ввода.

INPUT FILENAME :

В ответ на запрос необходимо ввести имя первого объектного файла. По умолчанию для имени предполагается расширение "**OBJ**".

Далее запрашивается число – смещение для раздела кода программы.

ENTER OFFSET FOR 'CODE' :

Возможны варианты ввода: [**Enter**], 0 или число. В двух первых случаях программа будет размещена в исходных адресах, определенных директивой ORG. В последнем случае все адреса программы будут увеличены на заданное число.

Затем запрашивается имя следующего объектного файла. Если программа состоит из одного ассемблированного файла, необходимо ответить нажатием клавиши [**Enter**], иначе ввести имя следующего файла.

INPUT FILENAME :

После этого редактор связей запрашивает имя выходного файла

OUTPUT FILENAME :

При ответе [**Enter**] имя выходного файла будет совпадать с именем первого объектного модуля. Тип расширения зависит от выбора опции, запрашиваемой в конце диалога.

Следующий запрос связан с именем библиотеки. Редактор связей может выполнять поиск внешних ссылок в 50 библиотеках.

LIBRARY FILENAME :

На запрос имени библиотеки можно просто ответить [**Enter**].

В заключении запрашивается опция, определяющая тип выходного файла программы.

OPTIONS (D,S,A,M,Z,X,H,E,T,1,2,3,<CR>=DEFAULT) :

Назначение предлагаемых опций описано ниже.

Ключ **X** создает выходной выполняемый двоичный файл.

Ключи **D, S, A, M, Z** позволяют создавать варианты файлов, используемых для отладки в различных форматах: MICROTEK, ZAK и других.

Ключи **H, E, T, 1, 2, 3** позволяют создавать различные выходные файлы в шестнадцатеричных форматах фирм INTEL, TEKTRONIX и MOTOROLA.

Наиболее подходящей является опция **X**, формирующая выходной двоичный исполняемый файл с расширением "**TSK**", предназначенный для записи в ПЗУ.

2.1.4 Кросс-транслятор COMP51

Для наших учебных целей вполне подойдет простой и надежный кросс-транслятор COMP51.EXE. При его использовании потребуется в конце программного модуля в отдельной строке указать директиву **END.**

Формат вызова транслятора в командной строке:

>COMP51 input_file[.ass] [/ключи]

Информация в квадратных скобках является необязательной и должна задаваться пользователем по необходимости. При пропуске этой информа-

ции используются значения по умолчанию. Поэтому при запуске транслятора для файла со стандартным расширением *.ASS достаточно указать только имя файла без расширения.

Список ключей кросс-транслятора:

/M:nnn - максимальный размер выходного файла, по умолчанию 4096;

/S:nnn - начальный адрес выходного кода, по умолчанию 0;

/D:out_file[.tsk] - имя выходного файла, по умолчанию - input_file.tsk;

/L:list_file[.lst] - имя файла листинга, по умолчанию не создается;

/C:n - тип микроконтроллера, где n имеет возможные значения:

0 - 8x51 (1830BE51) (по умолчанию)
1 - 8x52
2 - 8xC51
3 - 8xC52
4 - 8xC51Fx

В большинстве случаев нас устраивают значения параметров, которые заданы по умолчанию.

При обнаружении ошибок кросс-транслятор выдает сообщения, указывающие номер прохода, код ошибки, номер строки с ошибкой, тип ошибки, текст ошибочной строки и указатель-стрелку. Рассмотрим несколько примеров процесса трансляции программы, имеющей имя Tablic.asm.

Синтаксическая ошибка в тексте:

```
C:\LANQ>COMP51 Tablic.asm
Проход:1  строк:   11
ОШИБКА 26 в строке 13: Неверная команда или директива
                RED         ; Возврат из подпрограммы
                 ^
```

Опечатка в команде RET не позволила транслятору опознать команду.

Ошибочный операнд:

```
C:\LANQ>COMP51 Tablic.asm
Проход:2  строк:    1,  байт:0
ОШИБКА 19 в строке 3: Значение выражения > 255
                MOV R4,#310
                      ^
```

В этом примере в строке 3 обнаружена попытка занесения в байтовый регистр числа, превышающего максимально допустимое значение.

Вызов несуществующей подпрограммы:

```
C:\LANQ>COMP51 Tablic.asm
Проход:2  строк:    1,  байт:0
ОШИБКА 5 в строке 7: Имя не определено
                LCALL DELAU
                      ^
```

В строке 7 обнаружена ссылка на программу, имя которой не совпадает ни с одной из используемых подпрограмм. В данном примере была допущена описка в имени DELAY. Транслятор не смог сопоставить имя вызываемой подпрограммы с имеющимся набором имен подпрограмм.

Неправильное имя регистра:

```
C:\LANQ>COMP51 Tablic.asm
Проход:2  строк:   11,  байт:20
ОШИБКА 5 в строке 12: Имя не определено
                DJNZ R8, HAP
                     ^
```

В строке 12 использовано имя регистра R8, который не существует.

Результат правильной трансляции без ошибок.

```
C:\LANQ>COMP51 Tablic.asm
Проход:2  строк:   13,  байт:20
```

Транслятор сообщил количество строк кода (13) и длину программы в байтах (20).

Данный кросс-транслятор удачно сочетает простоту и оптимальность набора сервисных функций. Он не поддерживает библиотеки пользовательских подпрограмм. Транслятор содержит мало ключей для настройки и совмещает процесс трансляции с процессом компоновки. Поэтому на выходе транслятора сразу создается двоичный исполняемый файл в кодах микро-ЭВМ. По умолчанию для данного файла задается расширение *.TSK.

2.1.5 Кросс-отладчик 8051

Полноэкранный отладчик-симулятор **FD51.EXE** для программ, написанных на языке ассемблера однокристальных микро-ЭВМ КР1816ВЕ51/ КМ1816ВЕ31/КМ1816ВЕ51, предназначен для логической отладки программ, используемых указанными микро-ЭВМ. Каких-либо аппаратных средств отладчик не поддерживает.

Отладчик работает на персональных ЭВМ типа IBM PC XT/AT и совместимых с ними ЭВМ, и требует для работы не менее 256 Кбайт оперативной памяти.

Отладчик позволяет:

- загрузить для отладки 16-тиричные файлы, вырабатываемые имеющимися кросс-средствами (транслятором с языка ассемблера), а также файлы чистого двоичного кода, считанные, например, из ПЗУ;
- просмотреть на экране дизассемблированный текст загруженной программы, включая адреса и коды команд, область имитируемого

ОЗУ данных, область внешней памяти, памяти программ, содержимое всех регистров ОЭВМ;

- выполнить загруженную программу по шагам с просмотром результатов после каждого шага и в непрерывном режиме с остановом по точкам прерывания по достижении задаваемых пользователем адресов;
- внести изменения в загруженную программу в мнемонических обозначениях языка ассемблера, а также в машинных кодах;
- внести изменения в содержимое регистров, флагов и памяти в командном режиме и в режиме полноэкранного редактирования;
- вывести на печать или дисковые носители дизассемблированный текст, дампы памяти;
- сохранить содержимое любой области памяти в файле на дисковом носителе;
- загрузить память из дискового файла;
- получить трассировку программы;
- определить время выполнения загруженной программы и ее частей по встроенному счетчику.

Для запуска отладчика необходимо в командной строке набрать команду следующего содержания.

> **FD51.EXE**

Программа выдает входную заставку и через пять секунд включает рабочий режим. При этом на дисплей выводится (рисунок 2) основное окно программы, отображающее возможности многооконного взаимодействия с пользователем в режиме прямого доступа к элементам информации, а также сведения о назначении функциональных клавиш.

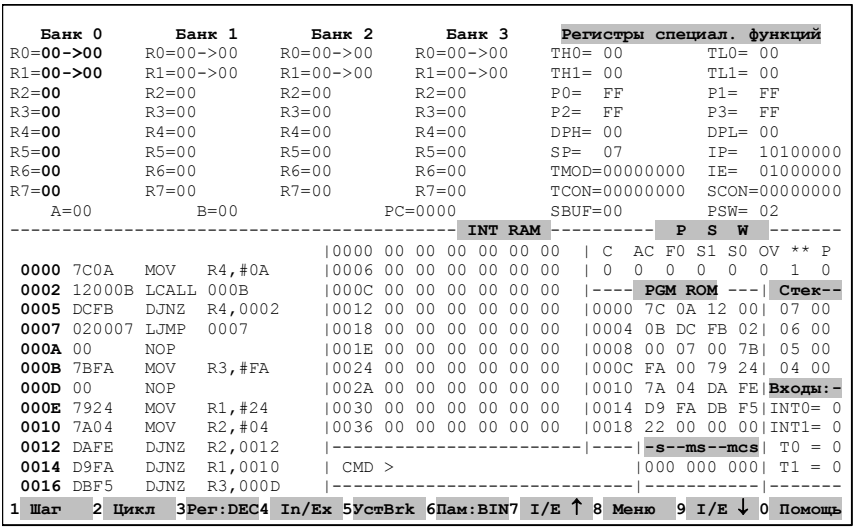

Рисунок 2 – Вид окна программы кросс-отладчика (симулятора)

В верхней части окна программы показаны имена и значения регистров всех 4-х регистровых банков, обозначенных как «**Банк 0**», «**Банк 1**», «**Банк 2**», «**Банк 3**». Активный банк регистров выделен с помощью увеличенной яркости значений регистров. В правой верхней части окна программы также показаны имена и значения всех доступных регистров специальных функций: таймеров/счетчиков, портов ввода/вывода, регистров последовательного порта, регистров прерываний и других.

В нижней половине окна отладчика слева показан дизассемблированный текст загруженной исполняемой программы с указанием адресов и кодов команд. Каждая строка соответствует одной команде и занимает от 1 до 3 адресов.

В центре нижней части экрана отладчика показано окно значений байтов внутренней/внешней памяти данных. Название окна – **INT RAM** или **EXT RAM**. В левой колонке этого окна показаны адреса ячеек памяти с шагом значений по 6 единиц, а справа группами по 6 байтов приведены значения ячеек памяти данных. Внутреннюю и внешнюю память можно отображать только поочередно путем переключения клавишей **F4.**

Под этим окном находится строка ввода команд отладчика **CMD >** и окно счетчика времени выполнения программы **s–ms–mcs**. Время отображается в секундах, миллисекундах и микросекундах. Счетчик настроен на вариант микро-ЭВМ, работающей с тактовой частотой 12 МГц. Для анализа времени выполнения программы на микро-ЭВМ с другой тактовой частотой

полученное значение времени выполнения программы нужно умножить на коэффициент отношения **k**.

$$k = \frac{12}{R},$$

где R – реальная тактовая частота в МГц.

В правой нижней части экрана в окне **PSW** показаны названия и значения битов регистра флагов, содержание байтов памяти программ показано в окне **PRG ROM**, текущее состояние стека приведено в окне **СТЕК**, состояние входов микро-ЭВМ INT0, INT1, T0 и T1 показано в окне **Входы**.

В нижней строке экрана имеется меню функциональных клавиш [**F1**] – [**F10**]. Они настроены на выполнение наиболее употребительных команд. Остальные команды вводятся пользователем в командной строке с клавиатуры при помощи алфавитно-цифровых клавиш. При вводе этих команд можно применять для редактирования клавиши [**Ins**], [**Del**], [**BackSpace**], [**Home**], [**End**], [**Esc**]. Помните, что после начала ввода команды и до нажатия клавиши [**Enter**], функциональные клавиши [**F1] – [F10]** недоступны. Для неверных команд выдается сообщение об ошибке и звуковой сигнал.

Рассмотрим действие команд, активизируемых функциональными клавишами.

F1 - выполнить текущую инструкцию загруженной программы. Текущая инструкция - это инструкция, выделенная в окне дизассемблированного текста светлым прямоугольником. После выполнения на экране можно сразу наблюдать результаты ее выполнения;

F2 - выполнить текущую инструкцию до ее завершения. Эта клавиша позволяет выполнить подпрограмму или цикл как одну инструкцию. Это удобно, так как не нужно просматривать уже отлаженные подпрограммы.

F3 – циклическое переключение представления числовой информации на экране (содержимого регистров и памяти) в десятичную, двоичную или шестнадцатеричную формы;

F4 - переключение большого окна памяти с внутренней (INT RAM) на внешнюю память (EXT RAM) и обратно;

F5 - установка точек прерывания;

F6 - переключение формы представления данных из памяти в окне (INT/EXT RAM) в двоичную или шестнадцатеричную;

F7 - листать окно памяти данных вверх на одну строку.

F8 – вызвать меню команд FD51.

F9 - листать окно памяти данных вниз на одну строку.

F10 - справочная информация.

Для пролистывания в окне INT/EXT RAM памяти данных вверх на один кадр используется клавиша [**Home**], на один кадр вниз – клавиша [**End**].

Просмотр памяти программ в окне PGM ROM можно осуществить с клавишами [**PageUp**] и [**PageDown**].

Остальные команды вводятся в командной строке. Для получения справки по командам можно ввести команду "H" или нажать комбинацию клавиш "Ctrl-H".

Все числовые значения, вводимые в программе, должны иметь шестнадцатеричный формат, при этом не требуется указывать букву "h". Рассмотрим допустимые в отладчике команды, приведенные в таблице 2.

Таблица 2

L <тип памяти><нач. адрес>, <файл. спец.> Загрузить файл в память. <Тип памяти> может быть I, E или P. Соответственно файл загружается во внутреннюю (**Int**), внешнюю (**Ext**) или программную (**Pgm**) память. Пример 1: **L I 01F,C:\PGM\T1** – загрузить двоичный файл C:\PGM\T1 во внутреннюю память данных, начиная с адреса 01F. Пример 2: **L P 0,Tablic.tsk** – загрузить двоичный файл программы Tablic.tsk в программную память с адреса 0.
S <тип памяти><нач. адрес>-<кон. Адрес>,<файл. спец.> Сохранить область памяти в дисковом файле. В качестве <файл. спец.> допускается любая корректная в DOS спецификация файла. Параметры <нач.адрес> - <кон.адрес> указывают соответственно начало и конец сохраняемой области. Сохраненный командой S файл можно потом снова загрузить командой L. Пример: **S P 20-642,C:\PGMLIB\MYFILE** – сохранить в двоичном виде область памяти программ с начального адреса 20h до конечного адреса 642h в файле C:\PGMLIB\MYFILE
PRTD <нач. адрес>,<количество команд>[,<файл. спец.>] Распечатать дизассемблированный текст, начиная с <нач. адреса>. Вывод по умолчанию направляется на принтер.
PRT <тип памяти><нач. адрес>-<кон. адрес>[,<файл. Спец.>] Записать в файл дамп области памяти в шестнадцатеричном формате. Если не указана <файл. спец.>, то дамп выводится на принтер. Пример: **PRT P 20-642,C:\PGMLIB\MYFILE** – сохранить в шестнадцатеричном формате область памяти программ с 20h по 642h адрес в файле C:\PGMLIB\MYFILE
R<номер регистра>=<число> Занести число в указанный регистр текущего банка. Число должно быть байтом. Пример: **R4=FF**
<Имя флага>=<число> Установить или сбросить указанный флаг в **PSW**. Можно использовать следующие имена флагов: **C, AC, F0, S1, S0, OV, P**. Если число=0, то флаг сбрасывается, иначе - устанавливается. Пример: **S1=0**
<Имя регистра>=<число> Занести число в регистр специального назначения. Можно использовать следующие

имена: **A, B, TH0, TH1, TL0, TL1, DPH, DPL, DPTR, SP, IP, IE, TMOD, TCON, SCON, SBUF, PC**. Число для **PC** и **DPTR** может быть двухбайтовой величиной. Для остальных регистров указывается один байт в шестнадцатеричной форме.

Пример: **SP=20** или **DPTR=FF00**

P0<номер порта>=<число> Занести число в порт. Номер порта – число от 0 до 3.

Пример: **P02=12**

D<адрес> Установить адрес дизассемблированного текста в окне.

Пример: **D 0240**

<Тип памяти><адрес>[-<кон. адрес>]=<число>

Занести число в память. Если указан <кон. адрес>, то заполняется целая область.

Пример: **I 22=55** или **P 0-40=FF**

<Тип памяти><адрес>.<номер бита>=<число>

Установить или сбросить бит в памяти. <Номер бита> может быть числом от 7 до 0 (старший бит - 7).

Пример: **I 20.6=1**

<Имя регистра>.<номер бита>=<число>

Установить или сбросить бит в регистре специального назначения (**A, B, P0-P3, IP, IE, TMOD, TCON, SCON**).

Пример: **TMOD.3=0**

M <тип памяти><нач. адрес>

Установить начальный адрес памяти в окне.

Примеры: **M I 20** или **M E 0FF** или **M P 0**

G [<нач. адрес>[,<кон. адрес>]]

Выполнить программу с <нач. адреса> до <кон. адреса>. Если <нач. адрес> пропущен, выполнение начинается с текущей команды, которая выделена белым прямоугольником. <Кон. адрес> можно не указывать, если используются точки прерывания. Выполняющуюся программу можно остановить нажатием любой клавиши.

Пример: **G 10, F0** или **G** или **G ,2E**

T ON [,<файл. спец.>] Включить трассировку программы.

Осторожно! Без имени файла поток данных трассировки направляется на принтер.

T OFF Выключить трассировку.

INT<0/1>=<число>

Имитировать высокий или низкий уровень на входах INT0 или INT1

Пример: **INT1=0**

BA=<адрес>

Установить новую «точку отсчета» для дизассемблирования. Эта команда полезна при просмотре таблиц, зашитых в памяти программ, когда при дизассемблировании «назад» неизвестно, откуда вести дизассемблирование.

RSTC Сбросить счетчик времени выполнения программы.

QUIT Выход в DOS.

RST Имитируется сброс процессора.

N И Вы как будто только что запустили FD51.

3 ЛАБОРАТОРНОЕ ЗАДАНИЕ

1. Ознакомьтесь с приведенным выше описанием работы всех программ, используемых при создании ПО для ОЭВМ MCS-51. Введите текст предлагаемой ниже программы. Текст программы должен быть набран с отступом.

2. Выполните трансляцию введенной программы и исправьте возможные ошибки.

3. Обработайте объектный файл в редакторе связей **LINK.EXE** и создайте исполняемый двоичный файл.

4. Загрузите отладчик **FD51.EXE**.

5. Командой L отладчика загрузите исполняемую программу.

6. Выполните программу по шагам.

7. Установите программу на начало. Сбросьте счетчик времени исполнения программы. Установите контрольную точку на программной ловушке. Выполните программу в автоматическом режиме. Определите время работы программы.

8. Последовательно изменяя значения счетчиков на 1 определите степень влияния счетчиков на суммарную задержку.

9. За счет изменения исходного значения счетчиков подберите другую тройку чисел, дающую более близкое к 1 секунде значение задержки.

10. Продемонстрируйте преподавателю работу программы и подготовьте отчет о работе по стандартной форме. В отчете опишите текст программы с комментариями и приведите свои оценки влияния счетчиков на общую задержку и свою тройку чисел с суммарной задержкой.

Программа задержки на 1 секунду.

```
MAIN:          ; PROC          ; Комментарии
               MOV R7,#0H      ; Счетчик секунд
CIKL:          ACALL DEL_01    ; Задержка на 1 секунду
               DJNZ R7, CIKL   ; Сколько секунд осталось?
STOP:          SJMP STOP       ; Программная ловушка

DEL_01:          ;PROC         ; П/программа задержки на секунду
               MOV R1,#50      ; Старший счетчик задан
STAR:          MOV R2,#43      ; Средний счетчик задан
SRED:          MOV R3,#231     ; Младший счетчик задан
MLAD:          DJNZ R3, MLAD   ; Задержка по младшему счетчику
               DJNZ R2, SRED   ; Задержка по среднему счетчику
               DJNZ R1, STAR   ; Задержка по старшему счетчику
               RET
```

4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как можно задавать адрес размещения программы в памяти?
2. Почему в редакторе связей необходимо создавать исполняемый файл в двоичном формате **TSK?**
3. Для какой цели в программе используется конструкция

STOP: SJMP STOP ?

ЛАБОРАТОРНАЯ РАБОТА № 2.

ОБРАБОТКА ВХОДНЫХ ДАННЫХ В ОМК

1 ВВЕДЕНИЕ

Целью работы является дальнейшее изучение этапов создания и методов отладки на примере программы, выполняющей обработку входных данных методом поиска максимального значения в массиве чисел.

2 ОБЩИЕ ПОЛОЖЕНИЯ

2.1 Дополнительные функции отладчика FD51.EXE

2.1.1 Полноэкранное редактирование

Переход в режим полноэкранного редактирования осуществляется нажатием клавиши [**Enter**] без ввода команды. Теперь можно перемещать курсор по экрану с помощью клавиш управления курсором и изменять содержимое регистров, памяти и флагов набором чисел на клавиатуре. Можно изменить также начальный адрес дисассемблированного текста (текущей инструкции) и начальные адреса окон памяти (в первых строках окон). Полноэкранное редактирование можно производить и при десятичном, и при двоичном представлении информации на экране. Во время редактирования остаются доступными все команды, вводимые функциональными клавишами. Чтобы вернуться в командную строку, нажмите [**Enter**] снова. Для быстрого перемещения курсора по экрану можно пользоваться клавишами [**Tab**] и [**Shift**]-[**Tab**].

2.1.2 Режим ассемблера

Для перехода в режим ассемблера (ввода команд отлаживаемой программы в мнемонических обозначениях) нужно в режиме полноэкранного редактирования поместить курсор в поле текущей инструкции загруженной программы. Теперь наберите мнемонику (например, «MOV A,#45») и нажмите [**Enter**]. Если мнемоника верна, то соответствующие ей коды заносятся в память программ, а окно устанавливается на следующий адрес.

При ассемблировании поддерживаются имена регистров специального назначения. При возникновении неоднозначности следует числовые значения предварять нулем. Для выхода из режима ассемблера нажмите клавишу «Q» или уведите курсор из поля текущей инструкции.

2.1.3 Работа с точками прерывания

Меню точек прерывания вызывается клавишей **F5**. Можно установить одновременно 8 точек прерывания. Прерывание (останов) выполняющейся программы происходит при достижении указанного в колонке «**PC**» адреса при выполнении условия «**Counter**» = «**Occur**». «**Counter**» - это счетчик, значение которого определяет, сколько раз программа должна пройти через указанный адрес, чтобы произошел останов. «**Occur**» показывает, сколько раз программа проходила через указанный адрес.

Окончив редактирование, текущие значения точек прерывания можно сохранить на диске, используя клавишу **F2**. У пользователя запрашивается номер набора точек прерывания (0-9). Информация записывается в файл с именем FD51.BRK. Восстановить картинку можно клавишей **F1**, также указав ее номер.

Для возврата в основное меню нажмите **F5.**

Определив точки прерывания, можно запустить программу командой **G** без параметров. При останове программы по прерыванию выдается сообщение с указанием номера точки прерывания.

2.2 Поиск экстремального значения

Одной из задач, решаемых при управлении бытовой, специальной промышленной аппаратурой и аппаратурой электронных систем безопасности, является обработка информации, поступающей в систему с датчиков. Виды датчиков весьма разнообразны, а характер информации, требующей обработки, еще более многообразен.

При обработке данных часто решается задача определения экстремального значения среди набора однотипных величин. Числовые величины поступают потоком в порт микро-ЭВМ с выхода датчика. В учебном примере тяжело моделировать для программы симулятора поток данных. Проще допустить, что данные заранее записаны в виде массива чисел. Для хранения массива можно использовать внешнюю память данных однокристальной микро-ЭВМ. Такое допущение вполне корректно. Например, запись во внешнюю память используют в случае невозможности обработки данных в темпе поступления отсчетов. В этом случае обработка состоит из фазы накопления чисел в памяти данных, и фазы обработки данных, считываемых из ячеек ВПД.

Задачу поиска максимума (минимума) приходится решать довольно часто: в визуальных устройствах контроля доступа, в видеосистемах охраны, в устройствах считывания штрихового кода, в сканерах, в аппаратуре определения положения объекта, при автоматической настройке аппаратуры (поиск канала, несущей частоты) и т.д.

Рассмотрим задачу точной автоматической настройки радиоприема на сигнал радиостанции. Эта задача формально является задачей определения максимума амплитуд сигналов.

Информация об амплитудах принимаемых радиосигналов поступает в микро-ЭВМ с выхода амплитудного детектора. Входной блок приема радиосигналов при поиске радиостанции получает с управляющей микро-ЭВМ команды для перестройки входного частотного фильтра. Таким способом сканируется весь диапазон радиочастот.

В окрестности несущей частоты передающей станции амплитуда принимаемых радиосигналов плавно нарастает и достигает максимума в момент точной настройки. По мере ухода от несущей частоты амплитуда принимаемых сигналов плавно уменьшается.

График распределения амплитуд сигналов вблизи несущей частоты в идеальном случае имеет гауссоидальную форму, вид которой показан на рисунке 3. Реально такую гладкую форму распределения можно получить только при статистической обработке данных для множества выборок.

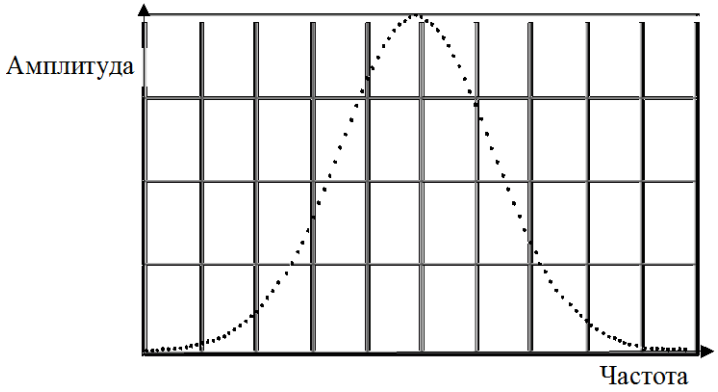

Рисунок 3 – Распределение амплитуд радиосигналов

В реальной форме графика распределения амплитуд кроме глобального максимума будут локальные максимумы и минимумы. Это необходимо учитывать при обработке данных.

Для настройки приемника на частоту станции необходимо определить в частотной области точку с экстремальным уровнем амплитуды сигналов и настроить входной частотный фильтр на это значение. В нашем случае необходимо определить номер ячейки памяти, в которой записан максимальный по значению элемент.

Примем, что каждый элемент массива данных записан в виде одного байта, в массиве задано 256 элементов. Для простоты массив расположен с

нулевого адреса внешней памяти данных однокристальной микро-ЭВМ. Вдали от максимума величина записанных значений близка к нулю.

3 ЛАБОРАТОРНОЕ ЗАДАНИЕ

1. Составьте текст программы, считывающей 256 элементов массива, записанного с 0 адреса во внешней памяти данных, и определяющей адрес максимального по амплитуде элемента.
2. Введите текст программы и запишите ее в файл на диске.
3. Выполните трансляцию введенной программы и исправьте возможные ошибки.
4. Обработайте объектный файл в редакторе связей **LINK.EXE** и создайте исполняемый двоичный файл.
1. Загрузите отладчик **FD51.EXE**.
2. Командой L отладчика загрузите исполняемую программу.
3. Загрузите во внешнюю память (EXT RAM) один из 10 файлов данных, имеющих имя **IMP0.BIN, IMP1.BIN,…, IMP9.BIN**
4. Используя пошаговый прогон, механизм контрольных точек, выполните отладку Вашей программы.
5. Сделайте вычисления в программе для двух соседних файлов данных.
6. Продемонстрируйте преподавателю работу программы и подготовьте отчет о работе по стандартной форме. В отчете опишите текст программы с вашими комментариями и приведите результаты трех вычислений.

4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Можно ли выполнить в Вашей программе поиск минимального элемента?
2. Какой объем внешней памяти данных доступен в однокристальной микроЭВМ К1816ВЕ51?
3. Каким образом можно записать во внешнюю память данных массив видеоданных?
4. Укажите возможные области применения устройства, решающего данную задачу.

ЛАБОРАТОРНАЯ РАБОТА № 3.

ФОРМИРОВАНИЕ СИГНАЛА УПРАВЛЕНИЯ В ОМК

1 ВВЕДЕНИЕ

Целью работы является углубленное изучение этапов создания и методов отладки программ на примере программы, вычисляющей координату с точностью до доли элемента дискретизации сигнала и формирующей сигнал управления.

2 ОБЩИЕ ПОЛОЖЕНИЯ

2.1 Формирование сигнала управления

После обработки данных в ОМК необходимо формировать сигнал управления. Рассмотрим задачу управления частотным фильтром для точной настройки на несущую частоту станции. Грубо координату мы можем определить с помощью программы поиска максимума. Необходимо уточнить значение координаты максимума, до долей элемента дискретизации. Исполнительное устройство, используя полученное разностное значение, компенсирует обнаруженное рассогласование.

В предыдущей работе мы определяли координату частотной настройки фильтра с точностью до элемента дискретизации амплитуды радиосигнала. Во многих задачах эта точность вполне приемлема. При необходимости подстройки с точностью, превышающей размер элемента дискретизации, необходимо повышать точность определения координаты. Точность должна быть повышена до доли элемента дискретизации.

Одним из способов является использование интерполяционных методов, использующих информацию о предполагаемой форме сигнала.

2.2 Интерполяционный метод

Используя информацию о предполагаемой форме сигнала, точность определения координаты центра объекта интерполяционным методом можно повысить до 10% от размера дискретного элемента. Это означает, что координата определяется с погрешность в 0.1 размера элемента. Опишем методику использования интерполяции.

Наиболее просто выполнить интерполяцию по трем точкам. В качестве центральной точки выберем элемент с максимальной амплитудой сигнала. Две другие точки – это левый и правый соседи. Для каждой из трех точек приписаны два признака: адрес расположения элемента и яркость (амплитуда

сигнала). Интерполируя форму сигнала по трем отсчетам яркости, определяем точный адрес положения максимума.

На рисунке 4 показана форма вершины импульса с тремя отсчетами в точках X_0, X_1 и X_2. Координата элемента, имеющего максимальную яркость, обозначена как X_1, а его амплитуда равна A_M. Левый от максимального элемента имеет координату X_0 и амплитуду A_L. Для правого элемента амплитуда равна A_R.

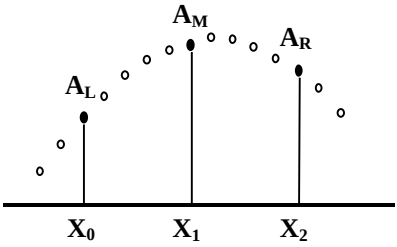

Рисунок 4 – Форма вершины импульса

Алгоритм интерполирования по трем точкам для приведенных обозначений вычисляет координату центра $X_Ц$ по формуле:

$$X_{Ц} = X_0 + 0.5 + \frac{A_M - A_L}{(A_M - A_L) + (A_M - A_R)} \qquad (1)$$

При выполнении вычислений по формуле (1) деление выполняется с использованием подпрограммы дробного деления **DIVDROB**. После работы подпрограммы в регистре B будет записан числитель дроби, знаменатель которой подразумевается и равен 256.

Используйте программу из первой лабораторной работы, которая вычисляет дробное частное от деления меньшего числа на большее. Частное представляется байтом, перед которым стоит значение 0 целых, а значение байта условно необходимо разделить на 256. Самый старший бит имеет вес 2^{-1} и равен 0.5. Самый младший бит имеет вес 2^{-8}=1/256. Таким образом, возможно вычисление дробных величин при целочисленной арифметике.

3 ЛАБОРАТОРНОЕ ЗАДАНИЕ

1. Составьте программу, определяющую точную координату центра объекта. Для поиска максимального элемента используйте программу поиска из предыдущей работы. Программу из первой лабораторной работы используйте для вычисления дробной части результата. Сформируете и запишите в ячейки ВПД 200H и 201H два байта кода сигнала управления, равного точному значению(целая и дробная части), которое необходимо прибавить к

вычисленной координате максимума сигнала для точной подстройки входного фильтра.

2. Введите программу, выполните ее трансляцию и создайте исполняемый двоичный файл.

3. Загрузите отладчик **FD51.EXE**.

4. Командой L отладчика загрузите исполняемую программу.

5. Загрузите во внешнюю память (EXT RAM) один из 10 файлов данных, имеющих имя **IMP0.BIN, IMP1.BIN,…, IMP9.BIN**

6. Выполните отладку Вашей программы и запишите результат вычислений.

7. Сделайте вычисления в программе для двух соседних файлов данных.

8. Продемонстрируйте преподавателю работу программы и подготовьте отчет о работе по стандартной форме. В отчете опишите текст программы с вашими комментариями и приведите результаты трех вычислений.

4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чему равна дробная величина, равная в двоичной форме величине 0.01010010B?

2. Какой объем внутренней памяти данных доступен в однокристальной микроЭВМ К1816ВЕ51?

3. Можно ли с помощью программы дробного деления вычислять целую часть частного и как?

4. Укажите возможные области применения устройства, решающего данную задачу.

ЛАБОРАТОРНАЯ РАБОТА № 4.

ЭЛЕКТРОННЫЙ КОДОВЫЙ ЗАМОК

1 ВВЕДЕНИЕ

Целью работы является освоение методов организации взаимодействия с объектами управления, углубленное изучение этапов создания и методов отладки программ на примере программы, управляющей работой цифрового электронного кодового замка.

2 ОБЩИЕ ПОЛОЖЕНИЯ

2.1 Структура кодового замка

Однокристальный микроконтроллер может использоваться для управления различными устройствами. Рассмотрим построение на базе ОЭВМ MCS-51 цифрового электронного кодового замка. Для построения устройства кроме ОЭВМ необходимы некоторые дополнительные блоки: клавиатура для ввода кодового слова, усилитель цифрового сигнала для открывания механизма замка, схема управления сигналом тревоги, срабатывающая при многократном ошибочном наборе кода.

На рисунке 5 показана предполагаемая схема подключения указанных устройств к разрядам порта P1 ОЭВМ.

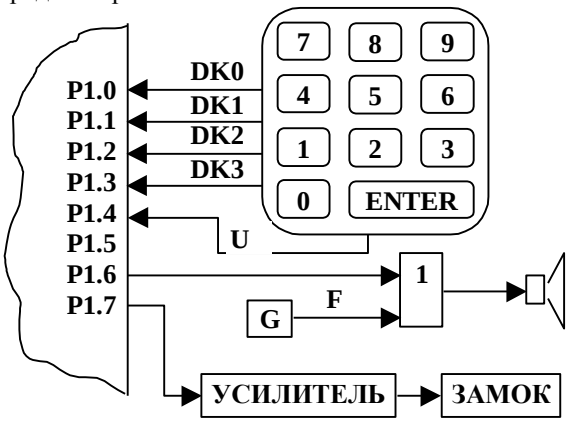

Рисунок 5 – Структура электронного замка

К младшим разрядам P1.0 – P1.3 подключен цифровой выход клавиатуры DK0–DK3, на котором формируется код введенной цифры. Для синхронизации процесса ввода цифр на выводе клавиатуры U, подключенном к разряду P1.4, формируется импульс готовности текущей цифры.

Для управления открыванием замка используется сигнал разряда P1.7. Сигнал открывания проходит через усилитель и включает соленоид ригеля замка, который втягивает защелку и открывает замок. Для включения сигнала тревоги разряд P1.6 подключен к входу логического элемента ИЛИ, на другой вход которого поступает частота F с генератора G. Изменяя сигнал на входе логического элемента, можно включать и выключать сигнал тревоги.

Поясним процесс ввода цифр. Диаграмма сигналов, приведенная на рисунке 6, показывает формирование цифры 3. Для синхронизации ввода код цифры сопровождается импульсом U. Длительность импульсных сигналов равна 1–2 миллисекундам.

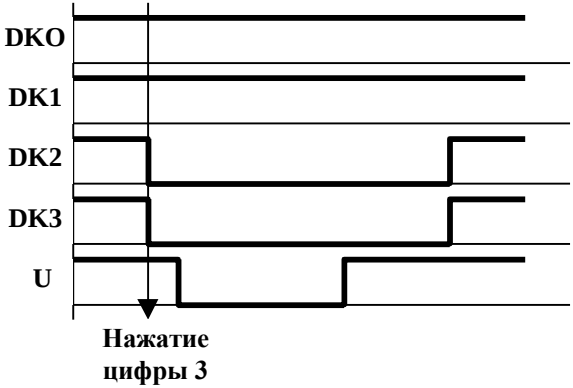

Рисунок 6 – Диаграмма сигналов на выводах клавиатуры при вводе цифры 3.

2.2 Алгоритм работы кодового замка

Для работы электронного замка необходим эталонный код. Эталонный код хранится во внешней памяти данных, в двух последовательных ячейках с адресами 100H (младшие 2 цифры) и 101H (старшие 2 цифры). Задание эталона осуществляется с помощью 16 разрядного DIP переключателя.

На рисунке 7 показан возможный вариант реализации формирователя эталонного кода. Комбинация включенных и выключенных DIP переключателей при чтении адресов 100H и 101H из ВПД формирует на выходе шинных формирователей BF 2 байта эталонного кода по две цифры.

Для открывания замка нужно набрать код минимум из 4-х цифр и нажать клавишу <ENTER>. При ошибке в наборе до нажатия <ENTER> можно набирать любое число цифр. При нажатии <ENTER> проверяются только четыре последних цифры.

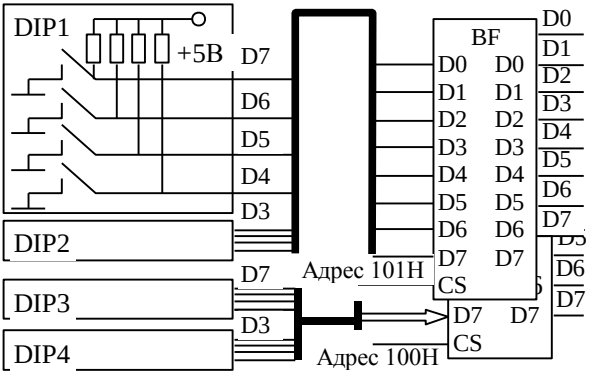

Рисунок 7 – Схема формирования эталонного кода

Коды цифр, формируемых на выходах DK0 – DK3 клавиатуры, соответствуют их двоичным 4-х битовым эквивалентам. Код <ENTER> равен 0Fh.

При пятикратном неправильном вводе кода на несколько секунд включается тревожный звонок.

3 ЛАБОРАТОРНОЕ ЗАДАНИЕ

1. Составьте текст программы, управляющей работой кодового замка. Запуск программы должен быть синхронизован по импульсу управления U.
2. Введите программу, выполните ее трансляцию и создайте исполняемый двоичный файл.
3. Загрузите в отладчик **FD51.EXE** исполняемую программу.
4. Выполните отладку Вашей программы.
5. Продемонстрируйте преподавателю работу программы и подготовьте отчет о работе по стандартной форме. В отчете опишите текст программы с вашими комментариями.

4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как можно повысить защищенность устройства от «взлома»?
2. Предложите способ задания эталонного кода без использования аппаратных переключателей.
3. Как можно организовать работу замка без использования синхросигнала U?

ЛАБОРАТОРНАЯ РАБОТА № 5.

ФИЛЬТРАЦИЯ ДАННЫХ

1 ВВЕДЕНИЕ

Целью работы является углубленное изучение методов обработки информации в микроконтроллерах, этапов создания и методов отладки программ на примере программы, выполняющей цифровую фильтрацию поступающего потока данных.

2 ОБЩИЕ ПОЛОЖЕНИЯ

2.1 Цифровая фильтрация данных

Информация, поступающая с датчиков, может быть представлена как в цифровой, так и аналоговой форме. Однокристальный микроконтроллер может выполнять различные виды обработки информации, даже фильтрацию аналоговых сигналов.

Каждый непрерывный сигнал, изменяющийся во времени, имеет свой спектр частот. Любая частота или полоса частот может быть усилена, ослаблена, исключена или выделена фильтрацией. При фильтрации изменяется спектр частот сигнала.

Цифровым фильтром называется цифровая система для изменения частотного спектра дискретных сигналов. Для обработки аналоговых сигналов с помощью цифрового фильтра используется схема, показанная на рисунке 8.

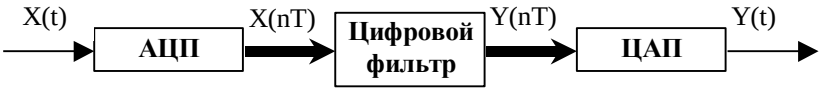

Рисунок 8 – Схема цифрового фильтра

Рассмотрим простейший пример цифровой реализации фильтра нижних частот, показанного на рисунке 9.

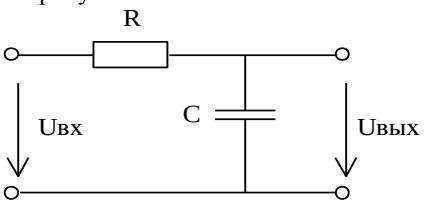

Рисунок 9 – Фильтр нижних частот

Уравнение фильтра имеет вид

$$U_{BX} = U_{BЫX} + RC\left(\frac{dU_{BЫX}}{dt}\right) \qquad (2)$$

Рассмотрим значения входного и выходного сигналов в дискретные моменты времени nΔt, n=0, 1,…. Заменяя производную конечной разностью

$$\frac{dU_{BЫX}}{dt} \approx \frac{U_{BЫX}[n\Delta t] - U_{BЫX}[(n-1)\Delta t]}{\Delta t},$$

и обозначая nΔt как n, получаем:

$$U_{BЫX}[n] = \frac{1}{1+RC/\Delta t}U_{BX}[n] + \frac{RC/\Delta t}{1+RC/\Delta t}U_{BЫX}[n-1] =$$
$$= k_1 U_{BX}[n] + k_2 U_{BЫX}[n-1] \qquad (3)$$

При условии Δt<<RC, несложно по формуле (3) реализовать цифровой фильтр в виде программы для МП.

Обозначим $Y_n=U_{ВЫХ}[n]$, $Y_{n-1}=U_{ВЫХ}[n-1]$, $X_n=U_{ВХ}[n]$ и зададим значения коэффициентов $k_1=0.375$ и $k_2=0.625$.

$$Y_n = 0.375 \cdot X_n + 0.625 \cdot Y_{n-1} = \qquad (4)$$
$$= 0.375 \cdot X_n + (0.625 \cdot Y_{n-1} - Y_{n-1}) + Y_{n-1} =$$
$$= 0.375 \cdot X_n - 0.375 \cdot Y_{n-1} + Y_{n-1} =$$
$$= 0.375 \cdot (X_n - Y_{n-1}) + Y_{n-1} \qquad (5)$$

Формулы (4) и (5) математически эквивалентны, но требуют разного объема вычислений в вычислителе. Дробные значения чисел можно получить при сдвиге их вправо с учетом знака. Число после сдвига вправо равно 0.5 от исходного значения. Несложно получить и коэффициенты 0.625 и 0.375.

Массив данных X_n размером 100 байтов должен располагаться во внешней памяти данных ОЭВМ, начиная с 0 ячейки. Фильтрованный массив Y_n нужно записывать с 200 ячейки памяти (EXT RAM).

3 ЛАБОРАТОРНОЕ ЗАДАНИЕ

1. Составьте текст программы, выполняющей цифровую фильтрацию массива данных X_n размером 100 байтов и создающей выходной массив Y_n, с 200 ячейки памяти.
2. Введите программу, выполните ее трансляцию и создайте исполняемый двоичный файл.
3. Загрузите в отладчик **FD51.EXE** исполняемую программу.

4. Создайте средствами отладчика входной массив данных X_n размером 100 байтов, запишите его с адреса 0. Для проверки работы фильтра наиболее удобно использовать тестовые сигналы в виде короткой положительной и отрицательной δ – функции и ступенчатой функции произвольного размаха, изменяющейся с положительным и отрицательным знаком.

5. Выполните отладку Вашей программы.

6. Продемонстрируйте преподавателю работу программы и подготовьте отчет о работе по стандартной форме. В отчете опишите текст программы с вашими комментариями. Нарисуйте в отчете графики входного сигнала и фильтрованного сигнала.

4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. На что влияют значения коэффициентов K_1 и K_2 в формуле (3)?
2. Какой характер выходного сигнала будет для плавно нарастающего входного сигнала?
3. Сравните особенности реализации фильтра по формуле (4) и (5).
4. Определите области применения данной разработки.

ПРИЛОЖЕНИЕ А

(справочное)

СПИСОК КОМАНД ОЭВМ MCS-51

Мнемокод	Название	Байт	Цикл	Операция
Группа команд передачи данных				
MOV @Ri,#d	Пересылка в РПД константы (i=0,1)	2	1	**((Ri))←#d**
MOV @Ri,A	Пересылка в РПД байта из аккумулятора	1	1	**((Ri))←(A)**
MOV @Ri,ad	Пересылка в РПД байта с прям.адреса ad	2	2	**((Ri))←(ad)**
MOV A,#d	Загрузка в аккумулятор константы	2	1	**(A)←#d**
MOV A,@Ri	Пересылка в аккумулятор из РПД (i=0,1)	1	1	**(A)←((Ri))**
MOV A,ad	Пересылка в аккумулятор байта с адреса ad	2	1	**(A)←(ad)**
MOV A,Rn	Пересылка в аккумулятор байта из регистра (n=0÷7)	1	1	**(A)←(Rn)**
MOV ad,#d	Пересылка константы по прямом.адресу	3	2	**(ad)←#d**
MOV ad,@Ri	Пересылка по прямому адресу байта из РПД (i=0,1)	2	2	**(ad)←((Ri))**
MOV ad,A	Пересылка аккумулятора по прямому адресу	2	1	**(ad)←(A)**
MOV ad,Rn	Пересылка регистра по прямому адресу	2	2	**(ad)←(Rn)**
MOV add,ads	Пересылка прямоадресуемого байта по прямому адресу	3	2	**(add)←(ads)**
MOV DPTR,#d16	Загрузка указателя данных словом	3	2	**(DPTR)←#d16**
MOV Rn,#d	Загрузка в регистр (n=0÷7) константы	2	1	**(Rn)←#d**
MOV Rn,A	Пересылка в регистр (n=0÷7) байта из аккумулятора	1	1	**(Rn)←(A)**
MOV Rn,ad	Пересылка в регистр с прямого адреса	2	2	**(Rn)←(ad)**
MOVC A,@A+DPTR	Пересылка в аккумулятор байта из ПП	1	2	**(A)←((A)+(DPTR))**

Мнемокод	Название	Байт	Цикл	Операция
MOVC A,@A+PC	Пересылка в аккумулятор байта из ПП	1	2	**(A)←((A)+(PC))**
MOVX @DPTR,A	Пересылка в расшир. ВПД из аккумулятора	1	2	**((DPTR))←(A)**
MOVX A,@DPTR	Пересылка в аккумулятор из расшир. ВПД	1	2	**(A)←((DPTR))**
MOVX @Ri,A	Пересылка в ВПД из аккумулятора	1	2	**((Ri))←(A)**
MOVX A,@Ri	Пересылка в аккумулятор байта из ВПД	1	2	**(A)←((Ri))**
POP ad	Извлечь из стека	2	2	**(ad)←((SP))** **(SP)←(SP)-1**
PUSH ad	Загрузить в стек	2	2	**(SP)←(SP)+1** **((SP))←(ad)**
XCH A,@Ri	Обмен аккумулятора с байтом из РПД	1	1	**(A)↔((Ri))**
XCH A,ad	Обмен аккумулятора и байта с адреса ad	2	1	**(A)↔(ad)**
XCH A,Rn	Обмен аккумулятора с регистром	1	1	**(A)↔(Rn)**
XCHD A,@Ri	Обменять младш. тетраду аккумулятора с мл. тетрадой байта РПД	1	1	**$(A_{0-3})\leftrightarrow((Ri)_{0-3})$**
SWAP A	Обменять тетрады в аккумуляторе	1	1	**$(A_{0-3})\leftrightarrow(A_{4-7})$**
Группа арифметических операций				
ADD A,#d	Сложить аккумулятор с константой	2	1	**(A)←(A)+#d**
ADD A,@Ri	Сложить аккумулятор с байтом из РПД	1	1	**(A)←(A)+((Ri))**
ADD A,ad	Сложить аккумулятор с байтом по адресу ad	2	1	**(A)←(A)+(ad)**
ADD A,Rn	Сложить аккумулятор с регистром	1	1	**(A)←(A)+(Rn)**
ADDC A,#d	Сложить аккумулятор с константой и переносом	2	1	**(A)←(A)+#d+(C)**
ADDC A,@Ri	Сложить аккумулятор с байтом из РПД (i=0,1) и переносом	1	1	**(A)←(A)+((Ri))+(C)**
ADDC A,ad	Сложить аккумулятор с байтом по адресу ad и переносом	2	1	**(A)←(A)+(ad)+(C)**
ADDC A,Rn	Сложить аккумулятор с регистром (n=0÷7) и переносом	1	1	**(A)←(A)+(Rn)+(C)**

Мнемокод	Название	Байт	Цикл	Операция
DA A	Десятичная коррекция аккумулятора	1	1	
SUBB A,#d	Вычесть из аккумулятора константу и заем	2	1	**(A)←(A)-#d-(C)**
SUBB A,@Ri	Вычесть из аккумулятора байт из РПД и заем	1	1	**(A)←(A)-((Ri))-(C)**
SUBB A,ad	Вычесть из аккумулятора байт с адреса ad и заем	2	1	**(A)←(A)-(ad)-(C)**
SUBB A,Rn	Вычесть из аккумулятора регистр и заем	1	1	**(A)←(A)-(Rn)-(C)**
INC @Ri	Инкремент байта в РПД	1	1	**((Ri))←((Ri))+1**
INC A	Инкремент аккумулятора	1	1	**(A)←(A)+1**
INC ad	Инкремент байта по адр. ad	2	1	**(ad)←(ad)+1**
INC DPTR	Инкремент указателя данных	1	2	**(DPTR)←(DPTR)+1**
INC Rn	Инкремент регистра	1	1	**(Rn)←(Rn)+1**
DEC @Ri	Декремент байта в РПД	1	1	**((Ri))←((Ri))-1**
DEC A	Декремент аккумулятора	1	1	**(A)←(A)-1**
DEC ad	Декремент байта по адресу ad	2	1	**(ad)←(ad)-1**
DEC Rn	Декремент регистра	1	1	**(Rn)←(Rn)-1**
MUL AB	Умножение аккумулятора на регистр B	1	4	**(B)(A)←(A)·(B)**
DIV AB	Деление аккумулятора на регистр B	1	4	**(A).(B)←(A)/(B)**
Группа логических операций				
ANL A,#d	Логическое И аккумулятора и константы	2	1	**(A)←(A)∧#d**
ANL A,@Ri	Логическое И аккумулятора с байтом из РПД (i=0,1)	1	1	**(A)←(A)∧((Ri))**
ANL A,ad	Логическое И аккумулятора с байтом по адресу ad	2	1	**(A)←(A)∧(ad)**
ANL A,Rn	Логическое И аккумулятора с регистром (n=0÷7)	1	1	**(A)←(A)∧(Rn)**
ANL ad,#d	Логическое И байта по адресу ad с константой	3	2	**(ad)←(ad)∧#d**
ANL ad,A	Логическое И байта по адресу ad с аккумулятором	2	1	**(ad)←(ad)∧(A)**

Мнемокод	Название	байт	цикл	Операция
ORL A,#d	Логическое ИЛИ аккумулятора с константой	2	1	**(A)←(A)∨#d**
ORL A,@Ri	Логическое ИЛИ аккумулятора с байтом из РПД (i=0,1)	1	1	**(A)←(A)∨((Ri))**
ORL A,ad	Логическое ИЛИ аккумулятора с байтом по адресу ad	2	1	**(A)←(A)∨(ad)**
ORL A,Rn	Логическое ИЛИ аккумулятора с регистром (n=0÷7)	1	1	**(A)←(A)∨(Rn)**
ORL ad,#d	Логическое ИЛИ байта по адресу ad с константой	3	2	**(ad)←(ad)∨#d**
ORL ad,A	Логическое ИЛИ байта по адресу ad с аккумулятором	2	1	**(ad)←(ad)∨(A)**
XRL A,#d	Исключ. ИЛИ аккумулятора с константой	2	1	**(A)←(A)⊕#d**
XRL A,@Ri	Исключающее ИЛИ аккумулятора с байтом из РПД (i=0,1)	**1**	**1**	**(A)←(A)⊕((Ri))**
XRL A,ad	Исключающее ИЛИ аккумулятора с байтом по адресу ad	2	1	**(A)←(A)⊕(ad)**
XRL A,Rn	Исключающее ИЛИ аккумулятора с регистром (n=0÷7)	1	1	**(A)←(A)⊕(Rn)**
XRL ad,#d	Исключающее ИЛИ байта по адресу ad с константой	3	2	**(ad)←(ad)⊕#d**
XRL ad,A	Исключающее ИЛИ байта по адресу ad с аккумулятором	2	1	**(ad)←(ad)⊕(A)**
CLR A	Сброс аккумулятора	1	1	**(A)←0**
CPL A	Инверсия аккумулятора	1	1	**(A)←($\overline{A}$)**
RL A	Сдвиг аккумулятора влево по циклу	1	1	**(A_{n+1})←(A_n), n=0÷6 (A_0)←(A_7)**
RR A	Сдвиг аккумулятора вправо по циклу	1	1	**(A_n)←(A_{n+1}), n=0÷6 (A_7)←(A_0)**
RLC A	Сдвиг аккумулятора влево через перенос	1	1	**(A_{n+1})←(A_n), n=0÷6 (A_0)←(C) (C)←(A7)**
RRC A	Сдвиг аккумулятора вправо через перенос	1	1	**(A_n)←(A_{n+1}), n=0÷6 (A_7)←(C) (C)←(A_0)**
Группа битовых операций				
MOV bit,C	Пересылка разряда переноса в бит	2	2	**(b)←(C)**

Мнемокод	Название	Байт	Цикл	Операция
MOV C,bit	Пересылка бита в разряд переноса	2	1	**(C)←(b)**
ANL C,bit	Логическое И бита и переноса	2	2	**(C)←(C)∧(b)**
ANL C,/bit	Логическое И инверсии бита и переноса	2	2	**(C)←(C)∧($\overline{b}$)**
ORL C,bit	Логическое ИЛИ бита и переноса	2	2	**(C)←(C)∨(b)**
ORL C,/bit	Логическое ИЛИ инверсии бита и переноса	2	2	**(C)←(C)∨($\overline{b}$)**
CLR bit	Сброс бита	2	1	**(b)←0**
CLR C	Сброс переноса	1	1	**(C)←0**
SETB bit	Установка бита	2	1	**(b)←1**
SETB C	Установка переноса	1	1	**(C)←1**
CPL bit	Инверсия бита	2	1	**(b)←($\overline{b}$)**
CPL C	Инверсия переноса	1	1	**(C)←($\overline{C}$)**
Группа команд передачи управления				
LCALL ad16	Длинный вызов программы	3	2	**(PC)←(PC)+3, (SP)←(SP)+2, (SP-1)←(PC_{0-7}), (SP)←(PC_{8-15}), (PC)←ad16**
ACALL ad11	Абсолютный вызов подпрограммы в странице 2 Кбайта	2	2	**(PC)←(PC)+2, (SP)←(SP)+2, (SP-1)←(PC_{0-7}), (SP)←(PC_{8-15}), (PC_{0-10})←ad11**
LJMP ad16	Длинный переход	3	2	**(PC)←ad16**
AJMP ad11	Абсолютный переход в странице размером 2 Кбайт	2	2	**(PC_{0-10})←ad11**
SJMP rel	Короткий относительный переход	2	2	**(PC)←(PC)+2+rel**
CJNE A,ad,rel	Сравнение аккумулятора с байтом по адресу ad и переход, если не равно	3	2	**(PC)←(PC)+3, если (A)≠(ad), то (PC)←(PC)+rel**
CJNE @Ri,#d,rel	Сравнение байта из РПД с константой и переход, если не равно	3	2	**(PC)←(PC)+3, если ((Ri))≠#d, то (PC)←(PC)+rel**

Мнемокод	Название	Байт	Цикл	Операция
CJNE A,#d,rel	Сравнение аккумулятора с константой и переход, если не равно	3	2	**(PC)←(PC)+3, если (A)≠#d, то (PC)←(PC)+rel**
CJNE Rn,#d,rel	Сравнение регистра с константой и переход, если не равно	3	2	**(PC)←(PC)+3, если (Rn)≠#d, то (PC)←(PC)+rel**
DJNZ ad,rel	Декремент байта по адресу ad и переход, если не нуль	3	2	**(PC)←(PC)+2, (ad)←(ad)-1, если (ad)≠0, то (PC)←(PC)+rel**
DJNZ Rn,rel	Декремент регистра и переход, если не нуль	2	2	**(PC)←(PC)+2, (Rn)←(Rn)-1, если (Rn)≠0, то (PC)←(PC)+rel**
JB bit,rel	Переход, если бит равен единице	3	2	**(PC)←(PC)+3, если (b)=1, то (PC)←(PC)+rel**
JBC bit,rel	Переход, если бит установлен, с последующим сбросом бита	3	2	**(PC)←(PC)+3, если (b)=1, то (b)←0, (PC)←(PC)+rel**
JC rel	Переход, если перенос равен единице	2	2	**(PC)←(PC)+2, если (C)=1, то (PC)←(PC)+rel**
JMP @A+DPTR	Косвенный относительный переход	1	2	**(PC)←(A)+(DPTR)**
JNB bit,rel	Переход, если бит равен нулю	3	2	**(PC)←(PC)+3, если (b)=0, то (PC)←(PC)+rel**
JNC rel	Переход, если перенос равен нулю	2	2	**(PC)←(PC)+2, если (C)=0, то (PC)←(PC)+rel**
JNZ rel	Переход, если аккумулятор не равен нулю	2	2	**(PC)←(PC)+2, если (A)≠0, то (PC)←(PC)+rel**
JZ rel	Переход, если аккумулятор равен нулю	2	2	**(PC)←(PC)+2, если (A)=0, то (PC)←(PC)+rel**

Мнемокод	Название	Байт	Цикл	Операция
RET	Возврат из подпрограммы	1	2	**$(PC_{8-15})\leftarrow((SP))$** **$(PC_{0-7})\leftarrow((SP)-1)$** **$(SP)\leftarrow(SP)-2$**
RETI	Возврат из подпрограммы обработки прерывания	1	2	**$(PC_{8-15})\leftarrow((SP))$** **$(PC_{0-7})\leftarrow((SP)-1)$** **$(SP)\leftarrow(SP)-2$**
NOP	Нет операции	1	1	**$(PC)\leftarrow(PC)+1$**