МАТЕРИАЛ 4. СИСТЕМА МИКРОПРОГРАММИРОВАНИЯ СЕМЕЙСТВА MCS-51 ОТ KEIL ELEKTRONIK GMBH.

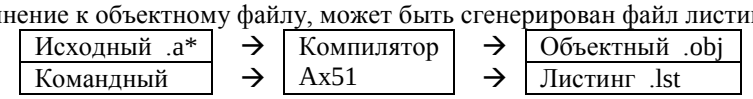
Цель – изучить основные приёмы работы с программными кросс-средствами Keil Elektronik GmbH для разработки ПО микропроцессоров семейства MCS-51 и производных.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА.

Краткая характеристика и пример программы – смотри лекция №1. Важно, что **компилятор двухпроходный**. Стандартные макросредства однообразные. Ассемблеры 8-разрядных МК всегда существенно проще, чем для x86 IA-16 и IA-32 (у них нет поддержки разных типов данных – всё байты. Семейство XA – 16-разрядные, поэтому должна быть поддержка типов с соответствующими директивами и операторами).

Функции компилятора: 1) синтаксический анализ; 2) семантический анализ; 3) генерация объектного кода; 4) поддержка широкой номенклатуры макросредств (включая повторы и условную компиляцию блоков кода); 5) ведение полной базы данных всех символических имён (для поддержки символической отладки).

Файлы, используемые компилятором. Ассемблер при одном запуске обрабатывает один исходный файл. Сгенерированный код помещается в объектный файл, имя которого совпадает с именем исходного файла, а расширение по умолчанию (если не указано явно) -.OBJ. Имя модуля по умолчанию совпадает с именем исходного файла. Имена файлов могут быть указаны любой комбинацией символов верхнего и нижнего регистров, тоже относится и к опциям (тогда как у других компиляторов – IAR A8051 – опции должны указываться строго, как в описании). В дополнение к объектному файлу, может быть сгенерирован файл листинга.



КОМПИЛЯТОР A51 Keil Elektronik GMBH (И РАСШИРЕНИЯ)

Запуск ассемблера с командной строки соответствует следующему синтаксису (директивы не обязательны и могут присутствовать в исходном тексте) (такую строку готовит интегрированная среда разработки):

A51 <sourcefile> [<directives…>]

или: **A51 @commandfile**

AX51 sourcefile [directives…]

A251 sourcefile [directives…]

AX51 @commandfile

A251 @commandfile

Например: H:\KEIL6.1\C51\BIN\A51.EXE .\IZMTEMP.a SET(SMALL) DEBUG XREF EP

Исходный файл по умолчанию имеет расширения **.A51** , **.ASM** (или в общем случае **.A***), **.SRC**. Включаемые файлы имеют расширения **.INC** или **.H**. Опции задаются полным символическим именем или аббревиатурой там, где это допустимо, со своими параметрами в скобках. Опции и имя исходного файла должны разделяться пробелами или символами табуляции. Для продолжения командной строки (например, свыше 128 символов, как в IAR A8051) используется командный файл.

Командный файл в вызове ассемблера используется, когда строка вызова достаточно большая и/или сложная, включающая исходный файл и сопутствующие директивы ассемблера (опции).

Выходные файлы ассемблера:

basename.LST – Файл листинга. Управляется директивами PRINT / NOPRINT, LIST / NOLIST;

basename.OBJ – Файл объектного модуля. Управляется директивами OBJECT / NOOBJECT.

1. ДИРЕКТИВЫ И УКАЗАНИЯ АССЕМБЛЕРУ.

Директивы ассемблера не порождают объектного кода в выходном файле!?

Директивы ассемблера делятся на две категории – просто **директивы** ассемблера и **команды управления** (директивы управления, часто называемые как опции компилятора, чтобы управлять действиями компилятора в процессе ассемблирования).

Директивы ассемблера – указывают ассемблеру, что делать с данными, условными переходами, макроопределениями и повторами, сегментами и другими блоками кода и данных.

Команды управления (директивы управления, controls)– требования к программе ассемблера выполнить определённые действия во время трансляции исходного модуля и что делать с файлами и листингами.

Просто **директивы** ассемблера в свою очередь можно разделить на два типа – **собственно директивы** и **псевдоинструкции** ассемблера.

Псевдоинструкции порождают данные в объектный код – это команды резервирования и инициализации памяти под переменные в своём классе памяти.

1.1. Директивы управления.

Директивы управления могут использоваться как в строке вызова компилятора, так и в теле исходного текста программы (порядок следования опций в командной строке не имеет значения). В тексте программы директиве предшествует символ доллара – «**$**», с которого начинается строка. Например:

A51.EXE .\IZMTEMP.A SET(SMALL) DEBUG XREF EP MPL

или в тексте

```
$SET(SMALL) DEBUG XREF EP MPL
```

или врозь по строкам

```
$SET(SMALL)          ; установить переменную SMALL в 1
$DEBUG               ; информация для символьной отладки
$XREF                ; перекрёстные ссылки имён в листинге
$EP                  ; сообщения ошибки на экран
```

$MPL ; макросредства вместо Си

Директивы управления могут быть глобальные и локальные. **Глобальные директивы управления** действуют на весь ассемблируемый файл и, поэтому, могут появляться лишь в первых строках исходного модуля или в строке вызова. Строка вызова имеет приоритет перед директивами в теле исходного модуля. Многие из глобальных директив управления могут быть заданы в диалоговом окне опций ассемблера в интегрированной среде разработки.

Локальные директивы могут встречаться в теле программы несколько раз и в произвольных местах. Действуют с места, где они появились. Не все из них могут быть в командной строке вызова.

Директивы со знаком ‡ применяются единожды в программе.

CASE ‡	**CA** -сокращение. **AX51, A251** только. Разрешает чувствительность к регистру символов. По умолчанию – все строчные буквы. Важна, где нужны прописные буквы.
COND / NOCOND	(**CO / NOCO** -сокращение.) Разрешает / запрещает пропущенные условные блоки выводить в листинге. По умолчанию – пропущенные блоки разрешены.
DATE(*date*) ‡	**DA** -сокращение. Размещает строку даты в заголовок страницы листинга (до 9 букв). По умолчанию – текущая дата.
DEBUG ‡	**DB** -сокращение. Информация о символах для отладчика включается в объектный файл. По умолчанию – информация для отладки запрещена.
EJECT	**EJ** -сокращение. Начать новую страницу в листинге.
ERRORPRINT[(*file*)] ‡	**EP** -сокращение. Задаёт файл для вывода сообщений об ошибках ассемблирования кроме листинга. Если имя файла не задано, то – вывод на консоль.
FIXDRK ‡	**FD** **A251** только. Заменяет INC DRk на ADD DRk для ряда C-Step МП.
GEN ‡	**GE** Генерирует полный листинг макрорасширения. По умолчанию – запрещено.
NOGEN ‡	**NOGE** -сокращение. Запрещает листинг макрорасширения.
INCDIR(*path*)	**ID** -сокращение. Задаёт пути включаемых файлов (перечисляются через ;).
INCLUDE(*file*)	**IC** -сокращение. Включает указанный файл в исходный текст программы, до 9 уровней.
INTR2	**I2** **A251** только. 2 байта адреса в стеке при обслуживании прерывания.
LIST / NOLIST	**LI / NOLI** Разрешает / запрещает вывод листинга. По умолчанию – разрешено.
MOD51 ‡	**M51** **AX51** только. Выбирает набор инструкций 8051. Это по умолчанию.
MOD_MX51 ‡	**MX** **AX51** только. Выбирает набор инструкций Philips 80C51MX.
MOD_CONT ‡	**MC** **AX51** только. Выбирает набор инструкций Dallas 390.
MODSRC ‡	**MS** **A251** только. Выбирает Intel/Temic 251 режим источника. По умолчанию – режим бинарный.
MPL ‡	Разрешает Macro Processing Language. По умолчанию – макросредства «С» разрешены.
NOLINES	**NOLN** Запрещает информацию номеров строк для отладчика.
NOMACRO ‡	Запрещает стандартные макросредства.
NOMOD51	**NOMO** Запрещает распознавать стандартные 8051 SFR-имена.
NOOBJECT	**NOOJ** Не создавать объектный файл.
NOREGISTERBANK	**NORB** Не используется переключение банков регистров.
NOSYMBOLS	**NOSB** Исключает таблицу символов из листинга.
NOSYMLIST	**NOSL** Запрещает определения символов выводить в листинг таблицы символов.
OBJECT[(*file*)]	**OJ** Задаёт имя объектного файла. По умолчанию берётся имя исходного модуля.
PAGELENGTH(n) ‡	**PL** Длина страницы листинга в строках (10-65535).
PAGEWIDTH(n) ‡	**PW** Длина строки листинга в символах (78-132).
PRINT[(*file*)] ‡	**PR** Определяет файл листинга.
NOPRINT ‡	**NOPR** Листинг не выдавать.
REGISTERBANK	**RB** Задаёт используемые банки регистров для редактора связей.
REGUSE	**RU** Определяет используемые регистры для учёта регистровой оптимизации.
RESTORE	**RS** Восстанавливает установки из стека 9 уровней, сохранённых директивой SAVE.
SAVE	**SA** Сохраняет в стеке установки для GEN, LIST и SYMLIST.
SYMBOLS	**SB** Включает таблицу символов в листинг.
SYMLIST	**SL** Разрешает присутствие определений символов в таблице символов.
TITLE(*string*) ‡	**TT** Определяет заголовок страницы в листинге.
XREF ‡	**XR** Создаёт таблицу перекрёстных символов в листинге. По умолчанию – запрещено.

Использование регистров – REGUSE – применяется к символическому имени PUBLIC (функции) для комбинирования с программами на Си, чтобы последние могли корректно провести регистровую оптимизацию. После имени в скобках указываются используемые регистры, например:

$REGUSE MYFUNC (ACC, B, R1 – R3, R5 – R7)
$REGUSE PROG1 (DPL, DPH)
$REGUSE PROG2 (ACC, CY, R6, R7)

2. ЯЗЫК АССЕМБЛЕРА MCS-51.

2.1. Программа на языке Ассемблера.

0. Программа – программа на языке Ассемблера состоит из одного или более предложений. Чаще всего пред-

ложение делят на оператор (команду) и комментарий, то или другое по отдельности или вместе (это для всех алгоримических языков).

<предложение 1> Вк Пс	[<оператор 1>] [<комментарий>] Вк Пс
. . . . .	. . . . .
<предложение N> Вк Пс	[<оператор N>] [<комментарий>] Вк Пс

Программа на языке Ассемблера состоит из слов, отделённых друг от друга разделителями (человеческий язык с паузами). Минимальный элемент программы – это лексема. Разделители – это тоже лексемы. Пробельные символы (пробел, табуляция, новая страница, возврат каретки, перевод строки, вертикальная табуляция, конец файла.) обычно не относят к лексемам (или это особый набор лексем).

1. Программа на языке Ассемблера – это конечная последовательность лексем с пробельными символами.

(Так для Си есть особенность: { и } – разделители, после них не требуется «;». Исключение: для структур и объединений всегда ожидается «;».)

Не просто набор слов, а налагаются условия: -1) порядок членов предложения фиксирован; -2) в предложении от 0 до нескольких операторов, т.е. команд; -3) длина предложения ограничена (Си, например, допускает продолжение строк «\»).

2. Программа на языке Ассемблера – это последовательность предложений из ряда лексем, соответствующих заданным 3 условиям и имеющим свободный формат. В одном предложении либо 0, либо 1 оператор (команда). Имена до 31 знака.

[<метка> [**:**]] <команда> [<операнд1>][**,** <операнд2>][**,** <операнд3>][**;** <комментарий>]

Обязательно присутствие команды, если нет – то комментарий или пусто.

3. - / - - - / - - с набором ключевых слов (предопределённых имён) для компилятора: **bit, bseg, code, cseg, data, dseg, db, dbit, ds, . . .** и т.д.; **a, b, pc, sp, dptr, r0, r1 – r7, mov, movc, movx, push, pop, xch, xchd, add, addc, da, subb, inc, dec, . . .** и т.д.; и для Си макросов: **define, undef, if, elif, ifdef, ifndef, else, endif, line, pragma, error, include,** включая **set, reset**.

Смотри лекцию и приложение к лекции №2 с перечнем мнемоник машинных инструкций.

Важно: предложение ассемблера = команда ассемблера и им может быть либо машинная инструкция, либо директива; в противном случае – пустая строка или комментарий.

Самая короткая программа состоит из директивы **END**. После END может стоять всё, что угодно.

2.2. Элементы языка Ассемблера.

Элементами языка являются: – множества символов; константы; идентификаторы; ключевые слова; комментарии; другие лексемы. (можно более систематичнее изложить)

Множества символов.

В программах на Ассемблере – **множество символов ассемблера** и **множество представимых символов**. Множество символов ассемблера содержит латинские буквы, цифры и знаки пунктуации, которые имеют определенное значение для ассемблера.

Множество символов ассемблера является подмножеством множества представимых символов. Множество представимых символов состоит из всех букв, цифр и символов, которые пользователь может представить графически как отдельный символ. Программа на Ассемблере может содержать только символы из множества символов ассемблера, за исключением строковых литералов, символьных констант и комментариев, где может быть использован любой представимый символ.

Буквы и цифры

Множество символов ассемблера включает большие и малые буквы из английского алфавита и 10 десятичных арабских цифр. A – Z, a – z, 0 – 9.

Пробельные символы

Ассемблер игнорирует пробельные символы, если они не используются как разделители или как компоненты константы-символа или строковых литералов.

Знаки пунктуации и специальные символы

Символ	Наименование	Символ	Наименование
,	Запятая	!	Восклицательный знак
.	Точка	|	+ Вертикальная черта
;	Точка с запятой	/	Наклонная черта вправо
:	Двоеточие	\	+ Наклонная черта влево
?	+ Знак вопроса	~	+ Тильда
'	Одиночная кавычка	_	Подчеркивание
“	+ Двойная кавычка	#	Знак номера
(	Левая круглая скобка	%	Знак процента
)	Правая круглая скобка	&	Амперсанд
{	+ Левая фигурная скобка	^	Caret
}	+ Правая фигурная скобка	-	Знак минус
<	Левая угловая скобка	=	Знак равно
>	Правая угловая скобка	+	Знак плюс
[	+ Левая квадратная скобка	$	Знак доллара
]	+ Правая квадратная скобка	@	Коммерческий знак

Операции

Операции - это специальные комбинации символов, специфицирующие действия по преобразованию различных величин. Ассемблер интерпретирует каждую из этих комбинаций как самостоятельную единицу, называемую лексемой (token). (Значительно меньше, чем в Си. Операции во многих случаях задаются ещё и символическими именами (ключевыми словами)).

Операция	Наименование
+	Сложение
-	Вычитание, арифметическое отрицание
*	Умножение
/	Деление
<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно
=	Равно
<>	Не равно

Константы.

Константа – это число, символ или строка символов и т.п.. Используются в программе как **неизменяемые величины**. В Ассемблере А51 различают три типа констант: целые константы, константы-символы и строчные литералы (строки). Константы с плавающей точкой требуют специальной поддержки.

Константы бывают явные и неявные. **Неявные константы** задаются псевдонимами (символическими именами) с помощью директив **EQU** и **SET. Явные константы** присутствуют в тексте в своём собственном представлении.

Целые константы

Целая константа – это десятичное, двоичное, восьмеричное или шестнадцатеричное число, которое представляет целую величину. Используется суффиксная нотация обозначения чисел. В пределах числа могут быть выделены поля для более ясного его представления. Символом поля является знак «**$**», который не может быть первым или последним. Компилятор его игнорирует; так 10000000 и 10$000$000 – идентичны.

Двоичная, восьмиричная, десятичная и **шестнадцатиричная** константы имеют следующий формат представления:

Основание	Суффикс	Знаки	Примеры
Шестнадцатиричное	H, h	0 – 9, A – F, a – f	0x1234 0x99 1234H 0A0F0h 0FFh
Десятичное	D, d	0 – 9	1234 65590d 20d 123
Восьмиричное	O, o, Q, q	0 – 7	177o 25q 123o 177777q
Двоичное	B, b	0 и 1	10011111b 101010101b

Шестнадцатиричная константа имеет один из следующих форматов представления языка Си:

0x<hdigits>
0X<hdigits>,

где <hdigits> одна или более шестнадцатеричных цифр.

Шестнадцатеричная цифра может быть цифрой от 0 до 9 или буквой (большой или малой) от A до F. В представлении константы допускается "смесь" больших и малых букв. Запись ведущего нуля и следующего за ним символа x или X необходима.

В нотации Ассемблера шестнадцатиричная константа требует ведущий нуль перед буквой.

Целые константы всегда специфицируют положительные величины и имеют тип int (16 разрядов). Если требуются отрицательные величины, то необходимо сформировать константное выражение со знаком минус и следующей за ним константы.

Константа-символ

Константа-символ – это буква, цифра или знак пунктуации, заключенные в одиночные кавычки. Допускается до двух символов в константе. Величина константы-символа равна значению представляющего ее кода символа (или символов). Константа-символ имеет следующую форму представления:

'<char>' или **'<char1> <char2>'**,

где <char> может быть любым символом из всего множества представимых символов, включая любой графический символ, исключая одиночную кавычку (') и символ новой строки. Константы-символы имеют тип int (16 разрядов), когда встречаются в выражении.

Чтобы использовать одиночную кавычку в качестве константы-символа, необходимо вставить её дважды.

'A' -вычисляется к 0041h
'AB' -вычисляется к 4142h
'a' -вычисляется к 0061h
'ab' -вычисляется к 6162h
'' -пустой символ вычисляется к 0000h
'abc' -ошибочная ситуация ERROR

```
LETTER_A    EQU     'A'
TEST:       MOV @R0, #'F'
            SUBB A, #'0'
```

Строковые литералы

Строковый литерал – это последовательность букв, цифр и символов, заключенная в одиночные кавычки. Строковый литерал рассматривается как массив символов, каждый элемент которого представляет отдельный

символ. Обычно используется в комбинациях с директивой **DB**. Строковый литерал имеет следующую форму представления:

'<characters>' ,

где <characters> - это нуль или более символов из множества представимых символов, исключая одиночную кавычку (') и символ новой строки. Например:

```
EOLMSG:   DB      'Конец передачи', 00h
MSGTXT:   DB      'ISN"T A QUOTE REQUIRED HERE?'
```

Идентификаторы

Ассемблер, как правило, работает с адресами данных (исключение – некоторые непосредственные данные). В языке эти адреса заменяются символическими именами для упрощения восприятия и осмысленности.

Идентификаторы - это имена переменных, функций и меток, используемых в программе. Идентификатор создается объявлением соответствующей ему переменной или функции. После этого его можно использовать в последующих операторах программы. Идентификатор - это последовательность из одной или более латинских букв, цифр или подчерков (_) и знака вопроса (?), которая не начинается с цифры. Допускается любое число символов в идентификаторе, вплоть до 31 символа, которые распознаёт Ассемблер.

```
NUMBER_FIVE     EQU   5
TRUE_FLAG       SET   1
FALSE_FLAG      SET   0
```

Ключевые слова

Ключевые слова – это предопределенные идентификаторы (мнемоники инструкций и директив, имена переменных), которые имеют специальное значение для Ассемблера. Их можно использовать только так, как они определены. Имена объектов программы не могут совпадать с названиями ключевых слов. Символ доллара, например, обозначает программный счётчик (**$**).

Мнемоники машинных инструкций приведены в приложении к лекции №2.

Комментарии

Комментарий – это последовательность символов, которая воспринимается компилятором как отдельный пробельный символ или, другими словами, игнорируется. Комментарий имеет следующую форму представления:

;<characters>

где <characters> может быть любой комбинацией символов из множества представимых символов с предшествующей (;). Это означает, что комментарии не могут занимать более одной строки.

Лексемы

Операции, константы, идентификаторы и ключевые слова, описанные в этом разделе, являются примерами лексем. Знаки пунктуации, (такие как квадратные скобки ([]) для семейства XA), угловые скобки (<>), круглые скобки и запятые, также являются лексемами. Границы лексем определяются пробельными символами и другими лексемами, такими как операции и знаки пунктуации.

2.3. Метки Ассемблера.

Метка определяет место (адрес) указанной позиции в сегменте кода или данных. Метка задаётся идентификатором с последующим символом двоеточия (**:**). Может задавать переменную (в сегменте данных).

```
LABEL1:
LABEL2:   DS      2                       ; две метки одного места
NUMBER:   DB      27, 33, 'STRING', 0     ; метка сообщения
COPY:     MOV R6, #12H                    ; метка в программе
```

Идентификатор метки определяется только один раз в программе (для однозначности). Метка не может быть переопределена на другой адрес с тем же символическим именем.

2.4. Операнды директив и машинных инструкций.

В машинных инструкциях может присутствовать **до трёх операндов**, разделённых запятой (**,**). Запись операндов – самое сложное в языке Ассемблера. Операнды задаются с помощью выражений. Константные выражения присущи для директив. В машинных инструкциях используются как константные выражения, так и адресные (операнд в памяти) (абсолютные и перемещаемые) в соответствии с режимами адресации.

Выделяется **6 типов** операндов.

Тип операнда	Описание
1. Непосредственные данные	Выражение, используемое как числовая величина. Выражению предшествует символ номера (**#**).
2. Прямой адрес бита	Выражение, оцениваемое в результате как адрес бита. Память класса BIT (от 0 до 7F) и бит-адресуемые SFR (от 80 до FF). Для семейства расширений 80C51 – это память класса EBIT
3. Адрес программы	Выражение, оцениваемое в результате как адрес в сегменте кода. Абсолютное или перемещаемое выражение в командах перехода. Относительный переход – SJMP, CJNE, DJNZ, JB, JNB, JBC, JC, JNC, JZ, JNZ. Переход в блоке – AJMP, ACALL. Далёкий переход – LJMP, LCALL. Для семейства расширений 80C51 – расширенный переход – EJMP, ECALL.
4. Прямой адрес данных	Выражение, оцениваемое в результате как адрес в сегменте данных. Память класса DATA и SFR.

5. Косвенный адрес	Выражение, специального формата с однозначным вариантом написания. Остальные классы памяти – IDATA, CODE, XDATA. Для семейства расширений 80C51 – EDATA, CONST, HDATA, HCONST.
6. Специальный символ	Имена регистров.

Специальные символы операндов Ассемблера.

Основной набор:

A	Представляет аккумулятор семейства MCS-51.
DPTR	Представляет 16-разрядный указатель данных в памяти класса XDATA или CODE.
PC	Представляет 16-разрядный программный счётчик.
C	Представляет флаг переноса в слове состояния программы.
AB	Представляет 16-разрядную регистровую пару A и B для умножения и деления.
R0 – R7	8-разрядные регистры общего назначения семейства MCS-51.
AR0 – AR7	Абсолютные адреса регистров общего назначения семейства MCS-51. Учитываются с директивами USING и NOAREGS.

Набор для расширений семейства MCS-51:

PR0, PR1	Универсальные указатели для Philips 80C51MX с 16 Мбайт. PR0 состоит из регистров R1, R2 и R3. PR1 – R5, R6 и R7.
EPTR	Дополнительный указатель данных внешней памяти Philips 80C51MX
R8 – R15	Дополнительные 8–разрядные регистры общего назначения семейства MCS-251.
WR0 – WR30	16 штук. 16–разрядные регистры общего назначения семейства MCS-251. WR0 - WR14 перекрывают R0 - R15.
DR0 – DR28, DR56, DR60	10 штук. 32- разрядные регистры общего назначения семейства MCS-251. DR0 - DR28 перекрывают WR0 - WR30.

Признак непосредственной адресации – знак номера (**#**).

Признак косвенной адресации – коммерческий знак номера (**@**).

Прямая адресация задаётся адресным выражением – абсолютным или перемещаемым. Например:

```
MyVal           EQU     50H             ; псевдоним числа
EXTRN CODE (My_1Func)
                CSEG AT 3
                JMP Intr_Ex0            ; к обработчику внешнего прерывания.
IntrProgr SEGMENT CODE                  ; сегмент с обработчиками прерываний.
                RSEG IntrProgr
Intr_Ex0:       JB UFlag,Labflag_OK
                INC my_var
Labflag_OK:     CPL UFlag
                RETI
                . . . . . .
SegMyBits SEGMENT BIT                   ; сегмент класса BIT
                RSEG SegMyBits
UFlag:          DBIT    1               ; управляющий бит в BIT пространстве
P1              DATA 90H                ; 8051 SFR порт №1.
GreenLed        BIT P1.2                ; GreenLed в порту P1.2
SegMyData SEGMENT DATA                  ; сегмент класса DATA.
                RSEG SegMyData
Value1:         DS 1                    ; резервирует 1 байт в DATA.
IO_PORT2        DATA 0A0H               ; SFR регистр.
Value2          DATA 20H                ; абсолютная адресация переменной в памяти DATA.
SegMyIData SEGMENT IDATA                ; сегмент класса IDATA
                RSEG SegMyIData
MyBuffer:       DS      100             ; резервировать 100 байт
SegMyPD SEGMENT XDATA INPAGE            ; определяет страницу в XDATA
                RSEG SegMyPD
VarPD:          DS 1                    ; резервирует байт.
SegMyProgr SEGMENT CODE                 ; сегмент кода.
                RSEG SegMyProgr
                ; ===== непосредственная адресация
                MOV A,P1                ; прямая адресация в DATA
                MOV A,#0E0h             ; непосредственно 0xE0 загрузить в аккумулятор
                MOV DPTR,#0x8000        ; непосредственно 0x8000 загрузить в указатель внешней памяти
                ANL A,#128              ; побитовое AND с аккумулятором и непосредственно 128
                XRL A,#0FFh             ; исключающее или с аккумулятором и непосредственно 0ffh
                MOV R5,#MyVal           ; загрузить R5 числом MyVal
```

```
                ; ====== адресация в DATA
                MOV A,IO_PORT2      ; прямая адресация к DATA.
                ADD A,Value1
                MOV Value2,A
                MOV R1,#Value1      ; регистровая + непосредственная, загрузить адрес Value1 в R1.
                ADD A,@R1           ; косвенная адресация к Value1 в DATA.
                ; ===== адресация в BIT
                SETB GreenLed       ; P1.2 = 1
                JB UFlag,LabIs_on   ; прямая адресация бита в DATA и адрес программы.
                SETB UFlag
                CLR ACC.5           ; сбросить бит 5 в аккумуляторе.
LabIs_on:       CLR UFlag
                CALL My_Function    ; адрес программы.
                ; ===== адресация в IDATA
                MOV R0,# MyBuffer   ; загрузить адрес буфера в R0
                MOV A,@R0           ; читать память MyBuffer
                INC R0              ; адрес следующего байта в R0
                MOV @R0,A           ; писать в память MyBuffer+1
                ; ===== адресация в XDATA
                MOV R1,#VarPD       ; загрузить адрес VarPD.
                MOVX @R1,A          ; доступ через R0 или R1.
                . . . . . .
Seg1MyProgr SEGMENT CODE INBLOCK ; сегмент в блокеа 2 Кбайта.
                RSEG Seg1MyProgr
LabL0:          CALL Func1          ; вызов -> ACALL.
LabL1:          CALL My_1Func       ; внешний вызов -> LCALL.
                MOV A,my_var
                JNZ LabL1
                RET
Func1:          CLR UFlag
                MOV R0,#20
LabL2:          CALL My_1Func       ; внешний вызов -> LCALL.
                DJNZ R0,LabL2
                RET
```

2.5. Операторы (операции) Ассемблера.

Операторы задают операции, обозначают их, являются их знаками. **Операции** – суть сами действия. Эти вычисления проводятся на этапе компиляции исходного текста. Вычисления проводятся в двух проходах, и результаты этих вычислений должны совпадать между собой для случаев, связанных с выделением памяти. Если такого совпадения нет, то возникает ошибка фазы компиляции.

Операции выполняются над 16-разрядными числовыми величинами. (Для семейства расширений MCS-C51 действия происходят над 32-разрядными величинами.)

Ассемблером общепринято поддерживаются **4 категории** операций с соответствующими им операторами. Это – арифметические, побитовые операции, операции отношений и атрибутивные операции. Содержание первых трёх категорий ясно определяется их названием. Последняя категория включает оставшиеся операции.

Арифметические операции

Операция	Синтаксис	Описание
+	+ *expression*	Унарный знак плюс.
–	– *expression*	Унарный знак минус.
+	*expression* + *expression*	Сложение бинарное.
–	*expression* – *expression*	Вычитание бинарное.
*****	*expression* * *expression*	Умножение бинарное.
/	*expression* / *expression*	Целочисленное деление.
MOD	*expression* MOD *expression*	Остаток от целого деления.
(и **)**	(*expressio*n)	Упорядочивает последовательность действий при вычислении.

Побитовые операции.

Операция	Синтаксис	Описание
NOT	NOT *expression*	Побитовое дополнение.
SHR	*expression* SHR *count*	Логический сдвиг вправо.
SHL	*expression* SHL *count*	Логический сдвиг влево.
AND	*expression* AND *expression*	Побитовое AND (логическое И)
OR	*expression* OR *expression*	Побитовое OR (логическое ИЛИ)
XOR	*expression* XOR *expression*	Побитовое исключающее OR (сумма по модулю 2)

Операции отношений.

Величина ложь = 0000h, а величина истина – ненулевое значение. Сравнения знаковые и беззнаковые.

Операция	Синтаксис	Описание
GTE	*expression1* GTE *expression2*	Истина, если *expression1* больше или равно *expression2*.
LTE	*expression1* LTE *expression2*	Истина, если *expression1* меньше или равно *expression2*.
NE	*expression1* NE *expression2*	Истина, если *expression1* не равно *expression2*.
EQ	*expression1* EQ *expression2*	Истина, если *expression1* равно *expression2*.
LT	*expression1* LT *expression2*	Истина, если *expression1* меньше *expression2*.
GT	*expression1* GT *expression2*	Истина, если *expression1* больше *expression2*.
>=	*expression1* >= *expression2*	Истина, если *expression1* больше или равно *expression2*.
<=	*expression1* <= *expression2*	Истина, если *expression1* меньше или равно *expression2*.
<>	*expression1* <> *expression2*	Истина, если *expression1* не равно *expression2*.
=	*expression1* = *expression2*	Истина, если *expression1* равно *expression2*.
<	*expression1* < *expression2*	Истина, если *expression1* меньше *expression2*.
>	*expression1* > *expression2*	Истина, если *expression1* больше *expression2*.

Атрибутивные операторы работают с определёнными атрибутами операндов и либо подавляют соответствующие атрибуты, либо возвращают значения атрибутов операндов. Данная категория более специализирована под конкретную архитектуру МК (МП). Ассемблер поддерживает три вида операций данного класса:

Операторы класса памяти.

Назначают **класс памяти** своему выражению.

Основной набор:

Оператор	Синтаксис	Описание
BIT	BIT *expression*	Назначает класс памяти BIT для выражения.
CODE	CODE *expression*	Назначает класс памяти CODE для выражения.
DATA	DATA *expression*	Назначает класс памяти DATA для выражения.
IDATA	IDATA *expression*	Назначает класс памяти IDATA для выражения.
XDATA	XDATA *expression*	Назначает класс памяти XDATA для выражения.

Набор для расширений семейства MCS-51:

Оператор	Синтаксис	Описание
CONST	CONST expression	Назначает класс памяти CONST для выражения.
EBIT	EBIT *expression*	Назначает класс памяти EBIT для выражения.
ECODE	ECODE *expression*	Назначает класс памяти ECODE для выражения.
ECONST	ECONST *expression*	Назначает класс памяти ECONST для выражения.
EDATA	EDATA *expression*	Назначает класс памяти EDATA для выражения.
HCONST	HCONST *expression*	Назначает класс памяти HCONST для выражения.
HDATA	HDATA *expression*	Назначает класс памяти HDATA для выражения.

Операторы типа данных.

Назначают **тип данных** своему выражению. Набор для расширений семейства MCS-51:

Оператор	Синтаксис	Описание
BYTE	BYTE *expression*	Назначает новый тип BYTE для выражения.
WORD	WORD *expression*	Назначает новый тип WORD для выражения.
DWORD	DWORD *expression*	Назначает новый тип DWORD для выражения.
NEAR	NEAR *expression*	Назначает новый тип NEAR для выражения.
FAR	FAR *expression*	Назначает новый тип FAR для выражения.

Операторы долей (разные).

Атрибут – само значение операнда. Возвращают кратную в единицах байта **часть значения** выражения. Выделять можно отдельные байты или смежные байты, образующие слова. Несмежные байты не группируются.

Ст. байт			Мл. байт
BYTE3	**BYTE2**	**BYTE1**	**BYTE0**
WORD2		**WORD0**	
		HIGH	**LOW**

Основной набор:

Оператор	Синтаксис	Описание
LOW	LOW expression	Младший байт 16-разрядного выражения.
HIGH	HIGH expression	Старший байт 16-разрядного выражения.

Данная категория операторов наиболее употребительна для расширений семейства MCS-51, где и представлена существенно большим количеством операторов.

Набор для расширений семейства MCS-51:

Оператор	Синтаксис	Описание
BYTE0	BYTE0 expression	Байт 0 32-разрядного выражения (аналог LOW).
BYTE1	BYTE1 expression	Байт 1 32-разрядного выражения (аналог HIGH).
BYTE2	BYTE2 expression	Байт 2 32-разрядного выражения.
BYTE3	BYTE3 expression	Байт 3 32-разрядного выражения.

WORD0	WORD0 expression	Младшее слово 0 32-разрядного выражения.
WORD2	WORD2 expression	Старшее слово 0 32-разрядного выражения.
MBYTE	MBYTE expression	**AX51** только. Для совместимости с библиотеками C51 при доступе к типу памяти **far**.

Приоритет операций.

Операции, относящиеся к 1-му уровню, вычисляются самыми первыми. Последним случаем – уровень 10.

Уровень	Операторы одного приоритета
1	**( ),** < >
2	**NOT, HIGH, LOW, BYTE0, BYTE1, BYTE2, BYTE3, WORD0, WORD2**
3	**BIT, CODE, CONST, DATA, EBIT, EDATA, ECONST, ECODE, HCONST, HDATA, IDATA, XDATA**
4	**BYTE, WORD, DWORD, NEAR, FAR**
5	**+** (унарный)**, –** (унарный)
6	***, /, MOD**
7	**+, –** (бинарные)
8	**SHR, SHL**
9	**AND, OR, XOR**
10	**>=, <=, =, <>, <, >, GTE, LTE, EQ, NE, LT, GT**

2.6. Выражения Ассемблера.

В выражения входят явные константы, счётчик адреса, символические имена и операторы в различных комбинациях. В общем случае, выражения состоят из подвыражений соединённых операциями. Результат оценки выражения – 16-разрядное число. (Для семейства расширений 80C51 выражения дают 32-разрядный результат.)

Выражения имеют свои **атрибуты**, которые характеризуют заданное выражения в отношении класса памяти и переместимости в этом адресном пространстве. Класс памяти (тип) выражения приводится в листинге в списке символических имён. После работы редактора связей все значения выражений становятся абсолютными.

Класс (тип) выражения.

Класс выражения	Описание
N NUMB	Чисто числовое значение.
C ADDR	Адрес символического имени в памяти **CODE.**
D ADDR	Адрес символического имени в памяти **DATA.**
I ADDR	Адрес символического имени в памяти **IDATA.**
X ADDR	Адрес символического имени в памяти **XDATA.**
B ADDR	Адрес символического имени в памяти **BIT.**
CO ADDR	Адрес символического имени в памяти **CONST.**
EC ADDR	Адрес символического имени в памяти **ECONST.**
CE ADDR	Адрес символического имени в памяти **ECODE.**
ED ADDR	Адрес символического имени в памяти **EDATA.**
EB ADDR	Адрес символического имени в памяти **EBIT.**
HD ADDR	Адрес символического имени в памяти **HDATA.**
HC ADDR	Адрес символического имени в памяти **HCONST.**

Чаще всего выражения имеют **класс NUMB**, т.к. составлены из числовых операндов. Класс NUMB – универсальный класс, он применим в любом выражении и в любом допустимом месте программы. Остальные выражения с именованными классами применяются только в тех местах, где допустим их тип. Мешать нельзя.

Если операнды выражения имеют класс памяти, то действуют **три правила:**

1). Результат унарной операции имеет класс памяти своего операнда.

2). Все бинарные операции возвращают класс NUMB, кроме сложения и вычитания.

3). Для + и – приложимы два случая. Если только один операнд имеет класс, то результирующее выражение имеет тот же самый класс. Если оба операнда имеют класс, то результат будет – NUMB.

Например:	data_address - 10	результирующая величина класса data_address.
	10 + edata_address	результирующая величина класса edata_address.
	(data_address - data_address)	результирующая величина класса number.
	code_address + (data_address - data_address)	результирующая величина класса code_address.

Абсолютная адресация задаёт адрес размещения объекта в своём классе памяти.

Перемещаемая адресация задаёт смещение объекта в соответствующем перемещаемом сегменте. Окончательный адрес вычисляет и устанавливает редактор связей.

Перемещаемое адресное выражение получается из перемещаемого символического имени и константного выражения посредством операций сложения и вычитания. Внешний символ также допустим:

- перемещаемое выражение + константное выражение
- константное выражение + перемещаемое выражение
- перемещаемое выражение - константное выражение

Перемещаемое адресное выражение может быть простым и расширенным. В **расширенном** перемещаемом адресном выражении могут использоваться имена перемещаемых сегментов и внешние имена.

Простое перемещаемое адресное выражение применяется:

1). в директиве ORG;
2). в директивах EQU и SET;
3). в директивах инициализации данных DB;
4). в операндах машинных инструкций.

Расширенное перемещаемое адресное выражение применяется в двух последних пунктах.

3. ДИРЕКТИВЫ АССЕМБЛЕРА.

Краткий обзор директив Ассемблера A51 и его расширений (преимущественно Philips 80C51MX и MCS-251).

Базовый набор директив.

BIT	*symbol* BIT *bit_address*	Определяет адрес бита в битовом пространстве.
BSEG	BSEG [AT *absolute address*]	Определяет абсолютный сегмент в пределах битового пространства.
CODE	*symbol* CODE *code_address*	Определяет имя символа в адресном пространстве кода.
CSEG	CSEG [AT *absolute address*]	Определяет абсолютный сегмент в пределах пространства кода.
DATA	*symbol* DATA *data_address*	Определяет имя символа в адресном пространстве данных на кристалле.
DSEG	DSEG [AT *absolute address*]	Определяет абсолютный сегмент в пределах пространства данных на кристалле.
DB	[*label*:] DB *expression* [, *expr* ...]	Генерирует список байтовых величин.
DBIT	[*label*:] DBIT *expression*	Резервирует биты в битовом пространстве.
DS	[*label*:] DS *expression*	Резервирует байты (без типа).
END	END	Отмечает конец программы.
EQU	*symbol* EQU *expression*	Задаёт невычисляемый псевдоним величины.
_ _ERROR_ _	_ _ERROR_ _ *text*	Генерирует сообщение об ошибке.
EVEN	EVEN	Выравнивает на чётную границу слова.
EXTRN	EXTRN *class (symbol* [, ...])	Определяет внешний символ.
IDATA	*symbol* IDATA *idata_address*	Определяет имя символа в адресном пространстве косвенных данных.
ISEG	ISEG [AT *absolute address*]	Определяет абсолютный сегмент в пределах пространства косвенных данных.
NAME	NAME *modulname*	Задаёт имя текущего модуля.
ORG	ORG *expression*	Устанавливает счётчик адреса в текущем сегменте.
PUBLIC	PUBLIC *symbol* [, *symbol* ...]	Объявляет символические имена глобальными.
RSEG	RSEG *seg*	Выделяет перемещаемый сегмент.
SEGMENT	*seg* SEGMENT *class* [*reloctype*] [*alloctype*]	Объявляет перемещаемый сегмент.
SET	*symbol* SET *expression*	Задаёт вычисляемый псевдоним величины.
sfr,	sfr *symbol = address*;	Определяет имена регистров специальных функций 8 разрядов.
sfr16	Sfr16 *symbol = address*;	Определяет имена регистров специальных функций 16 разрядов.
sbit	sbit *symbol = address*;	Определяет имена регистров специальных функций битовые.
USING	USING *expression*	Объявляет банк регистров для прямой адресации AR0 – AR7.
XDATA	*symbol* XDATA *xdata_address*	Определяет имя символа в адресном пространстве внешних данных.
XSEG	XSEG [AT *absolute address*]	Определяет абсолютный сегмент в пределах пространства внешних данных.

Расширенный набор директив. **AX51** и **A251**.

DD	[*label*:] DD *expression* [, *expr* ...]	Генерирует список величин в два слова.
DSB	[*label*:] DSB *expression*	Резервирует байты.
DSD	[*label*:] DSD *expression*	Резервирует двойные слова.
DSW	[*label*:] DSW *expression*	Резервирует слова.
DW	[*label*:] DW *expression* [, *expr.* ...]	Генерирует список величин размером в слово.
EXTERN	EXTERN *class* [:*type*] (*symbol* [, ...])	Определяет внешний символ.
LABEL	*name*[:] LABEL [*type*]	Определяет метку в адресном пространстве сегмента.
LIT	*symbol* LIT *'literal string'*	Назначает псевдоним строке символов.
PROC	*name* PROC [*type*]	Определяет начало процедуры.
ENDP	*name* ENDP	Определяет конец процедуры.

Примеры применения некоторых директив смотри в лекции №1 для контроллера термометрии.

4. СРЕДСТВА УСЛОВНОЙ ТРАНСЛЯЦИИ.

Это директивы управления для реализации различных версий программы из одного исходного файла. В командной строке они запрещены, применимы только внутри исходного текста для выделения блоков текста:

IF	Начало условного блока, если следующее выражение истинно.
ELSIF	Предыдущий блок неверен. Начало следующего блока, если следующее выражение истинно.
ELSE	Предыдущий блок неверен. Блок всегда транслируется, если предыдущий блок фальшив.
ENDIF	Конец условного блока.
SET	Устанавливает символические имена истинными значениями для блоков IF и ELSIF. (0FFFFh по умолч.)
RESET	Устанавливает символические имена ложными значениями для блоков IF и ELSIF. (0000h по умолч.)

```
$SET (PROG = 0, DEB0 = 1, DEB2 = 2)
$RESET (DEB1)
IF IDATALEN <> 0
        MOV    R0,#IDATALEN - 1
        CLR    A
IDATALOOP:    MOV    @R0,A
        DJNZ   R0,IDATALOOP
ENDIF
```

Блоки группы **IF** могут быть вложены до 10 уровней.

Перед директивами управления группы **IF** может ставиться, и может не ставиться знак **$**. Если префикс доллара стоит, то в условном выражении участвуют только символы директив **SET** и **RESET**. Без доллара в условном выражении участвуют все символы исходной программы.

5. МАКРОСРЕДСТВА.

Могут поддерживаться раздельно три средства.

1). **Макросредства стандартного Ассемблера.** Используются стандартные директивы:

MACRO	Определение макроса с его именем и параметрами (до 16 параметров). Вложение до 9 уровней.
ENDM	Конец блока макроса или блока повторений.
LOCAL	Определяет локальные метки для макроса (до 16 символов).
EXITM	Выход из макроса при генерации макрорасширения.
REPT	Блок повторения фиксированное число раз.
IRP	Блок повторения для каждого параметра из заданного списка.
IRPC	Блок повторения для каждого параметра из заданного списка, где параметр – отдельный символ.

Стандартные операторы: **NUL** – проверка на пропущенный параметр при вызове**; &** - слияние параметров; **<** > - содержимое передавать литерально (какова вложенность, столько раз и повторять); **%** - вычислить выражение; **;;** - игнорировать содержимое; **!** – символ воспринимать литерально (например, запятая и угловые скобки).

```
DELAY       MACRO   CNT    ;; определение
              REPT  CNT    ;; блок повторения
              NOP
              ENDM
            ENDM

LOOP:       MOV     P1, #0FEh
            DELAY   7
            MOV     P1, #0FFh
            DELAY   5
            JMP     LOOP
```

2). **Макросредства Си**. Соответствуют стандартному препроцессору Си. Можно использовать одни файлы заголовков в одном проекте.

3). **Макросредства Intel ASM-51.** Разрешается директивой **MPL**. При этом препроцессор Си не работает.

ЛАБОРАТОРНАЯ РАБОТА 3. «ИЗУЧЕНИЕ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ПРОГРАММИРОВАНИЯ MCS-51 НА ЯЗЫКЕ СИ И АССЕМБЛЕРА (KEIL ELEKTRONIK GmbH)»

Часть 1.

1). Исследовать программу контроллера термометрии, приведенную в лекции №1.

2). Написать программу для чтения сообщений в коде «Манчестер 2». Цифровой сигнал 16-разрядов получен обычным преобразованием АЦП процесса. Провести компиляцию программы на языке Си и данных для неё на языке ассемблера. Скомпоновать программу. Получить и прочитать текст сообщения!

Файлы сообщений имеют имена Man2.ds, Man2.d1s, Man200.ds, Man201.ds, и т.п..

Часть 2.

1). Провести медианную фильтрацию процесса с фильтрами разных порядков. Использовать различные алгоритмы сортировок. Провести компиляцию программы на языке Си и данных для неё на языке ассемблера. Скомпоновать программу.

2). Все варианты программ свести в единую программу, где каждое задание использует свой банк памяти в модели с переключаемыми банками.

Примеры кодов передачи информации

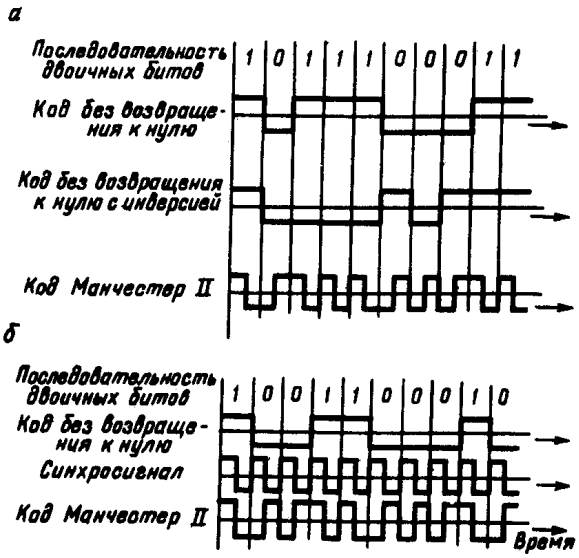


Рис. 1

На рис. 1 приведены некоторые примеры последовательной передачи данных. Примечателен самосинхронизирующийся код Манчестер 2.

Пример декодирования из кода Манчестер-2

Pascal

```
{+---------- Manch2Decode ----------------------------------+
 ¦ Процедура декодирует из кода Манчестер-2 в байты.      ¦
 ¦ M2AdrSour - адрес анализируемого массива;              ¦
 ¦ M2porog - порог для импульса, битовое кодирование;     ¦
 ¦ M2granica - граница индекса исходного (word);          ¦
 ¦ M2interval - интервал 3/2T-1; T-половина;              ¦
 ¦ M2AdrDest - смещение приёмника кода (сегмент DS);      ¦
 ¦ M2dlina - длина текста в байтах.                       ¦
 ¦ M2shift - число пустых бит в последнем байте.          ¦
 +-------------------------------------------------------+}
procedure Manch2Decode;              { ( PtrS : pointer;
                                         PtrD : Integer;
                                         Porog, Interval12 : Word ); }
{var
  M2AdrSour : pointer absolute PtrS;
  M2AdrDest : Integer absolute PtrD;
  M2porog : Word absolute Porog;
  M2interval : Word absolute Interval12; }
begin {----------------------- body procedure ---------------------------}
  asm
    mov     ax,M2porog
    mov     si,M2AdrDest
    mov     dx,M2granica
    les     di,M2AdrSour
    push    bp
    mov     bp,M2interval
    cld
@WorkLoop:
    mov     cx,8
@ByteLoop:
    scasw                    { получить текущий бит в CF   }
    rcr     bl,1             { сохранить бит               }
    js      @GetByte1        { SF=1 если CF=1, отклонение }
@GetByte0:                   { меньше 1/2 диапазона        }
    scasw
    jnc     @GetByte0
    jmp     @YesByte
@GetByte1:
    scasw
```

```
    jc     @GetByte1
  @YesByte:
    add    di,bp
    cmp    di,dx
    ja     @Exit
    loop   @ByteLoop
    mov    [si],bl
    inc    si
    jmp    @WorkLoop
  @Exit:
    pop    bp
    dec    cl
    shr    bl,cl
    mov    [si],bl
    sub    si,M2AdrDest
    inc    si
    mov    M2dlina,si
    mov    M2shift,cx
   end
  end; {***** procedure *****}
```

ЛАБОРАТОРНАЯ РАБОТА 3. ПРОГРАММИРОВАНИЕ MCS-51 (ДОПОЛНЕНИЕ.)

Прочитать сообщения – **Man2.ds**, **Man2.d1s** (по 4000 байт каждое), **Man200.ds** и т.п.. Пример второго приведен ниже. Перекодировать из таблицы 866 в таблицу среды Windows 1251.

Цифровой сигнал (10-разрядный АЦП) сообщения в коде «Манчестер 2».

Фрагмент сигнала Man2.d1s

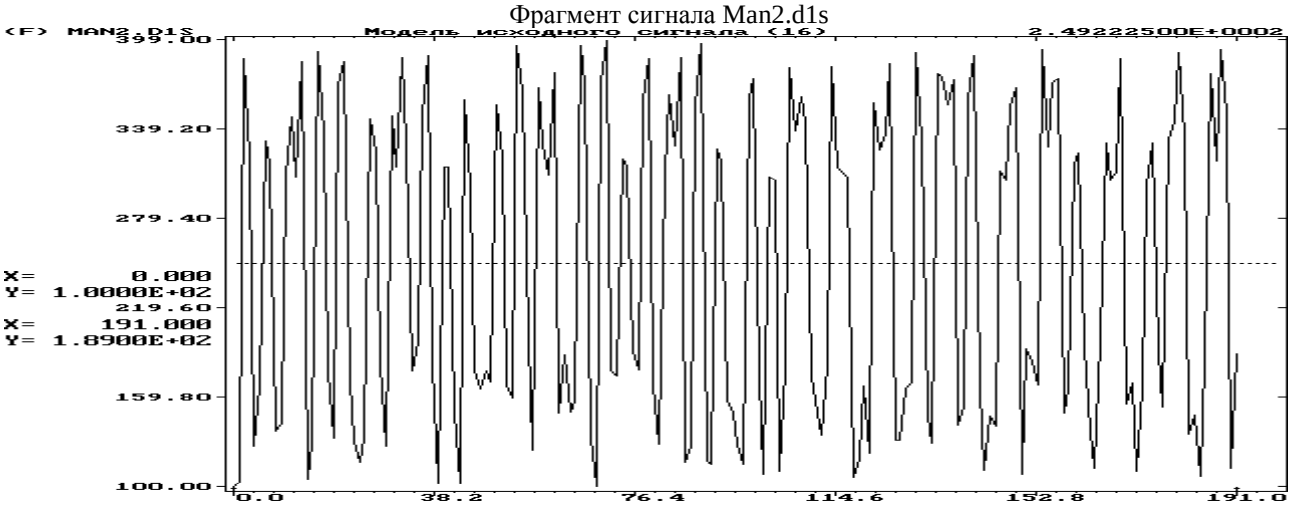


Рис. 1

4 отсчёта сигнала передают один бит текстового сообщения. Это показывает рис. 2.

Начальный участок фрагмента сигнала Man2.d1s в дискретных отсчётах

Рис. 2

Двоичный фрагмент этого файла следует ниже:

```
000:  64 00 67 00 82 01 40 01 ¦ 7F 00 A7 00 4B 01 3C 01
010:  89 00 8E 00 34 01 5B 01 ¦ 33 01 80 01 69 00 81 00
020:  87 01 50 01 B1 00 84 00 ¦ 71 01 80 01 AB 00 82 00
030:  74 00 84 00 5A 01 44 01 ¦ B6 00 7F 00 5C 01 3A 01
040:  83 01 48 01 B1 00 C5 00 ¦ 5D 01 84 01 B6 00 66 00
050:  3A 01 3A 01 96 00 66 00 ¦ 67 01 2C 01 B1 00 A5 00
060:  B1 00 AA 00 63 01 40 01 ¦ A8 00 9F 00 8B 01 6C 01
070:  C7 00 7C 00 6F 01 49 01 ¦ 34 01 79 01 95 00 BC 00
080:  96 00 9D 00 8B 01 70 01 ¦ 85 00 64 00 72 01 8E 01
090:  B1 00 AE 00 3F 01 3A 01 ¦ BF 00 B2 00 66 01 82 01
0A0:  A8 00 80 00 35 01 6A 01 ¦ 48 01 83 01 74 00 7F 00
0B0:  62 01 8C 01 75 00 73 00 ¦ 46 01 3D 01 9D 00 95 00
0C0:  80 00 73 00 68 01 75 01 ¦ AE 00 6C 00 33 01 31 01
0D0:  6E 00 B3 00 7C 01 52 01 ¦ 69 01 57 01 AE 00 95 00
0E0:  86 00 B5 00 7D 01 3A 01 ¦ 35 01 32 01 6A 00 77 00
0F0:  A7 00 7A 00 65 01 45 01 ¦ 50 01 7F 01 83 00 83 00
100:  A5 00 A9 00 86 01 4B 01 ¦ 93 00 81 00 78 01 76 01
   . . . . . .
```

Отсчёты представлены словами в формате 16-разрядных процессоров (мл. байт по младшему адресу).

С помощью 16-го отладчика получим текст файла в формате 16-ых слов, который преобразуем к исходному тексту на языке Ассемблера (файл **Man2d1s.src**). (Точно такие же действия необходимо сделать и с родственным файлом **Man2.ds**.).

1). Поставим на выполнение отладчик, например: G:\TOOLSKPP\SYMDEB.EXE MAN2.D

2). 4000 байт – это FA0h. Получим дамп файла в 16-разрядных словах по команде -dw 100 l 7d0

3). Чтобы дамп получить в текстовый файл, перенаправим консоль на дисковый файл > man2.txt

4). Затем восстановим вывод на консоль.

5). В текстовом редакторе с помощью команд найти и заменить получим ассемблерные директивы для данных размером в слово. В итоге получим:

```
; Набор данных для инициализации. /файл источник Man2.ds/
; Сообщение в коде Манчестер-2
$PAGELENGTH(5000)
NAME    MAN2D1S

?C_INITSEG      SEGMENT CODE
?XD?MAN2D1S     SEGMENT XDATA
        PUBLIC  iMasMan2

        RSEG  ?XD?MAN2D1S
iMasMan2:       DS    4000
        RSEG  ?C_INITSEG
        DB      06FH          ; Заголовок 00-DATA, 01-XDATA, 10-PDATA
        DB      0A0H
        DW      iMasMan2
        DW      00064h, 00067h, 00182h, 00140h, 0007Fh, 000A7h, 0014Bh, 0013Ch
        DW      00089h, 0008Eh, 00134h, 0015Bh, 00133h, 00180h, 00069h, 00081h
        DW      00187h, 00150h, 000B1h, 00084h, 00171h, 00180h, 000ABh, 00082h
        DW      00074h, 00084h, 0015Ah, 00144h, 000B6h, 0007Fh, 0015Ch, 0013Ah
        . . . . . .
```

Инициализация глобальных переменных в Си. Используется сегмент **?C_INITSEG** с блоком параметров для инициализации (смотри лекцию №2).

Программа декодера имеет примерно следующий вид:

Пример 1

```
//#pragma code //===== Листинг ===== ассембл. расшифровка
#pragma nocond //===== исключить ложные условные.
#pragma symbols
#pragma warninglevel(2)
#pragma nointvector
#pragma registerbank(0)
#pragma small

extern unsigned int xdata IMASMAN2[];
 #include <reg51.h>
 #include <stdio.h>
```

```
static char pdata cpMas[] = "1 ЊЋ‹…ЌњЉЋц 2 ЦЂЋґЂЋЊЋЂ 3 Їа®Ў  4\n\
ЋЃ,ґ„…†‡€‰Љ‹ЊЌЋЦ Ђ‘’“”•––□™љ›њќћц\n\
 ЎўJ¤Ґ¦§Ё©Є«¬-®Ї абвгдежзийклмноп рс ь";
static char code cpConvertToW[] =
                                { 0xC0, 0xC1, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7,
                                  0xC8, 0xC9, 0xCA, 0xCB, 0xCC, 0xCD, 0xCE, 0xCF,
                                  0xD0, 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD6, 0xD7,
                                  0xD8, 0xD9, 0xDA, 0xDB, 0xDC, 0xDD, 0xDE, 0xDF,
                                  0xE0, 0xE1, 0xE2, 0xE3, 0xE4, 0xE5, 0xE6, 0xE7,
                                  0xE8, 0xE9, 0xEA, 0xEB, 0xEC, 0xED, 0xEE, 0xEF,
                                  0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7,
                                  0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF,
                                  0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7,
                                  0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF,
                                  0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7,
                                  0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF,
                                  0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,
                                  0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF,
                                  0xA8, 0xB8, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7,
                                  0x8, 0x9, 0xA, 0xB, 0xB9, 0xD, 0xE, 0xF };
unsigned char bdata cbByte;
sbit bBit7 = cbByte ^ 7;

void main( void ) {
  int iI, iI2, iMO;
//  int iI, iI2;
  unsigned long lAcc = 0;
//  #define iMO (*( int * )&lAcc )
  #define iInterval 3
//  unsigned char cWork;
  unsigned char pdata *pPD = cpMas;
  //***** вычислить порог для сигнала
  for ( iI = 0; iI < 2000; iI++ )
    lAcc += IMASMAN2[ iI ];
  iMO = lAcc / 2000;
  //----- декодировать сигнал
 #if 1 // 1, если нужно декодировать
  iI = 0;
  do {
    for ( iI2 = 0; iI2 < 8; iI2++ ) {
      cbByte >>= 1;
      if ( IMASMAN2[ iI++ ]> iMO ) {
//        bBit7 = 1;
        cbByte |= 0x80;
        while ( IMASMAN2[ iI++ ]> iMO ) ;
      } else while ( IMASMAN2[ iI++ ]<= iMO ) ;
      if (( iI += iInterval)>= 2000 ) goto labExit;
    }
    *pPD++ = cbByte;
  } while ( iI < 2000 );
labExit:
  if ( iI2 = 8 - iI2  ) {
    cbByte >>= ( iI2 - 1 );
    *pPD++ = cbByte;
  }
  *pPD = '\0';
 #endif
  //***** конверсия в таблицу WINDOWS
  pPD = cpMas;
  TI = 1;
  printf( "КОНВЕРСИЯ в таблицу WINDOWS ->\n[%s]\n", cpMas );
 #if 1 //----- вариант 1
  while ( cbByte = *pPD ) {
    if ( bBit7 )
```

```
      *pPD = cpConvertToW[ cbByte ^ 0x80 ];
    pPD++;
  }
 #else //----- другие варианты
  while ( cWork = *pPD ) {
    if ( cWork & 0x80 )
      *pPD = cpConvertToW[ cWork ^ 0x80 ];
/* 3-ий
    if ( cWork >= ( unsigned char )0x80 ) {
      cWork &= 0x7F;    // cWork ^= 0x80;
      *pPD = cpConvertToW[ cWork ]; } */
/* 4-ый
    if ( cWork >= ( unsigned char )0x80 ) {
      cWork ^= 0x80;
      *pPD = cpConvertToW[ cWork ]; } */
/* 5-ый
    if ( cWork & 0x80 ) {
      cWork &= 0x7F;
      *pPD = cpConvertToW[ cWork ]; } */
    pPD++;
  }
 #endif
  printf( "Результат конверсии ->\n[%s]\n", cpMas );
  while ( 1 );
}
```

Второе сообщение имеет вид:

Фрагмент сигнала Man2.ds

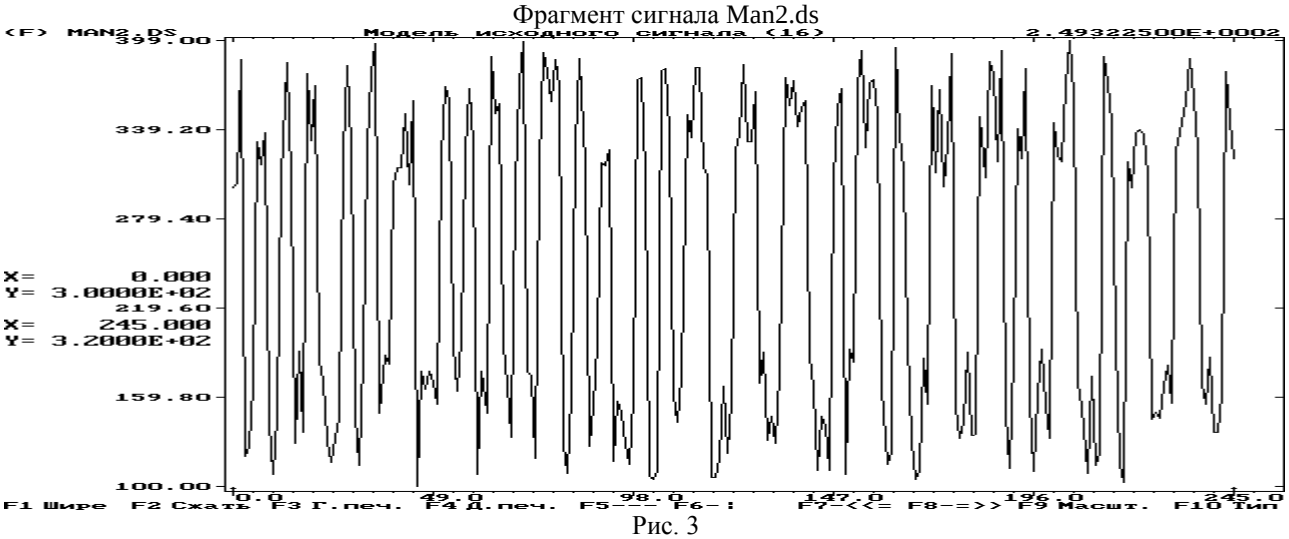


Рис. 3

Кодировки русских букв:

	866	**1251**
8	А Б В Г Д Е Ж З И Й К Л М Н О П	
9	Р С Т У Ф Х Ц Ч Ш Щ Ъ Ы Ь Э Ю Я	
A	а б в г д …	
B		
C		А Б В Г Д Е Ж З …
D		Р С Т У Ф Х Ц …
E	р с т у ф …	а б в г д …
F	Ё ё …	р с т у ф …