Санкт-Петербургский государственный политехнический университет
Факультет технической кибернетики

Кафедра измерительные информационные технологии

Методические указания к практическим и лабораторным работам по теме «Основы программирования микроконтроллерных систем»

Работа №3

«М*акроассемблеры: программные модули, сегменты, макросы, условное ассемблирование»*

по курсам «Аппаратные средства ВТ»,
«Микроконтроллеры в приборостроении»,
«Микропроцессорные системы и языки ассемблеров»,
«Сигнальные процессоры»

Авторы: Евдокимов В.Е., Гродецкий Ю.А.,
Лобан В.И., Цветков В.А.

Санкт-Петербург - 2009

1. ЦЕЛЬ РАБОТЫ

Цель работы - знакомство с одним из компонентов интегрированных средств разработки прикладных программ для микроконтроллерных систем (МК-систем) – макроассемблером, приобретение требуемого уровня компетенций в данной области.

2. ПРЕДМЕТ ИЗУЧЕНИЯ

Понятие макроассемблер, прежде всего, подразумевает язык программирования. По сравнению с ассемблером такой язык обладает большей функциональностью, поддерживая дополнительные конструкции языка, в частности макросы, секции условного ассемблирования, и др. Среда разработки может поддерживать сразу несколько различных макроязыков.

С другой стороны, тот же термин - макроассемблеры применяется для трансляторов ассемблерных программ, которые включаются в среду разработки и могут использоваться ею при построении проекта или отдельно для трансляции исходного текста прикладных программ.

Подобные средства разработки, представленные на рынке для различных МК-семейств, как правило, снабжаются функциями, схожими с функциями транслятора ASM фирмы Intel. Однако в составе среды они могут получать дополнительные свойства и иметь отличительные черты.

Ниже на примере интегрированной среды разработки KEIL µVision2 для 8-разрядных МК семейства x51 исследуются особенности макроассемблера A51, входящего в состав этой среды.

В качестве самостоятельного задания может быть предложено:

- дальнейшее исследование макроассемблера A51: на других примерах, для иной платформы с учетом специфики платформы (например, для МК фирмы SiLabs, МК фирмы Analog Devices), развитие возможностей макроассемблера в среде KEIL µVision3;
- изучение особенностей ассемблеров 8-разрядных МК-семейств с системой команд отличной от MCS-51, например микроконтроллеров семейства AVR фирмы Atmel (среда AVR Studio), МК фирмы Microchip Technology (среда MPLab), МК фирмы Cypress;
- исследование ассемблеров открытой среды разработки IAR Embedded Workbench для 32-разрядных МК, в частности для МК семейства ATmega, ARM-микроконтроллеров (ARM7 в режимах Thump или ARM); в том числе, исследование особенностей применения ассемблеров для сложных МК-приложений – с применением операционной системы реального времени (например, uC/OS-II) для многозадачных приложений;
- исследование трансляторов встроенных беспроводных процессоров, например процессоров ARM964 фирмы WaveComm семейства ARM9;
- изучение ассемблеров сигнальных процессоров, в частности особенностей ассемблера интегрированной среды разработки VisualDSP++ for 21xx для DSP процессоров фирмы Analog Devices.

3. ЗАДАЧИ И ЭТАПЫ РАБОТЫ

3.1 Модифицировать структуру одномодульной ассемблерной демонстрационной программы, сохраняя при этом алгоритм программы, имена переменных и процедур.

Примечание: В качестве демонстрационной программы используется программа timer.a51, имитирующая работу стартового секундомера и рассмотренная в работе 1, или любая другая аналогичная программа по согласованию с преподавателем.

3.1.1. Выделить сегменты памяти программ и данных с относительной и абсолютной адресацией.

3.1.2. Выполнить трансляцию модифицированной программы, сконфигурировав предварительно среду разработки с учетом типа МК, и настроив макроассемблер и линкер (в частности, для генерации исполняемого кода в Intel HEX-формате, генерации MAP-файла).

3.1.3. Проанализировать MAP-файл с результатами компоновки и размещения, генерируемый линкером.

3.1.4. Выполнить и описать процедуру отладки приложения с использованием симулятора среды разработки.

3.1.5. Ввести макросы, в том числе макрос с двумя локальными ссылками.

3.1.6. Добавить фрагменты с условным ассемблированием.

3.1.7. Повторить трансляцию и анализ модифицированной программы, а также отладку приложения с использованием симулятора.

3.1.8. Произвести загрузку и запуск исполняемого кода, полученного в обоих случаях, на программно-аппаратном комплексе и убедиться в правильном функционировании приложений (в данном случае в реализации режимов работы стартового секундомера).

3.2. Разбить модифицированную (например, по пункту 3.1.1.) демонстрационную программу на перемещаемые и неперемещаемые модули и создать исполняемый HEX-файл двумя путями (в соответствии с п. 3.2.1. и п. 3.2.3.).

3.2.1. Построить проект для многомодульной программы, включив в него исходные модули с текстом программы, и выполняя трансляцию и линкование с помощью команды Build Project.

3.2.2. Проанализировать MAP-файл.

3.2.3. Альтернативно, выполнить ассемблирование отдельных модулей программы, а затем осуществить сборку и размещение полученных на первом этапе объектных модулей - в командной строке среды разработки или через команду Build Project.

3.2.4. Проанализировать MAP-файл.

3.2.5. Выполнить процедуру отладки приложения с использованием симулятора.

Ключевые слова:

Макроассемблер, макроязык, директивы ассемблера,

сегмент (секция) кода (сегмент памяти программ), сегмент данных (памяти данных), классы (типы) памяти, абсолютный сегмент, перемещаемый (именованный) сегмент, активный сегмент, частичные сегменты,

макрос, макроопределение, локальные ссылки в теле макроса,

код с условным ассемблированием,

программный модуль, неперемещаемый и перемещаемый модуль, межмодульный интерфейс, главный модуль, область видимости битовых переменных.

Задачи и этапы исследования могут корректироваться в зависимости от задания. Выше они в большей степени сориентированы на знакомство с макроассемблером A51 среды KEIL µVision2 x51-совместимых МК.

4. ПРЕДСТАВЛЕНИЕ РЕЗУЛЬТАТОВ РАБОТЫ

Результаты всей работы и отчеты в электронном виде размещаются в директории студента на компьютере учебной лаборатории и предъявляются преподавателю.

Отдельно преподавателю сдаются отчеты на бумажном носителе. Тексты и рисунки отчетов должны быть выполнены на компьютере и представлены в формате Microsoft Word.

Содержание отчета должно включать следующие разделы:

1. Постановка задачи.

   В этом разделе формулируется конкретное полученное задание так, как оно понимается студентом.
2. Задачи и этапы работы.

   За основу принимается соответствующий раздел методических указаний, который корректируется в зависимости от конкретного задания.
3. Основные теоретические сведения.

   Своими словами излагаются почерпнутые из источников информации (предпочтительно из первоисточников) материалы, на которых основываются исследования.
4. Результаты работы и их анализ.

   Это один из наиболее важных разделов отчета. В нем подробно поэтапно описываются и анализируются выполненные исследования с размещением фрагментов модифицированных программ, со ссылками на материалы, размещенные в приложении, приводятся структуры аппаратно-программных комплексов с комментариями и т.д.
5. Выводы.

   Делается общее заключение о достижении (или не достижении по каким-то причинам) цели, поставленной в работе.
6. Источники информации.

   Размещаются ссылки на литературные и электронные источники информации. Для электронных источников необходимо указывать подробный (конечный) электронный адрес, по которому был найден соответствующий материал.
7. Ответы на тестовые вопросы.

   Приводятся и обосновываются ответы на тестовые вопросы, размещенные в соответствующем разделе и дополнительно выдаваемые преподавателем. Указывается номер теста и ответ.
8. Приложение.

   В приложении приводятся исходные тексты программ, полученные при выполнении п. 3.1. и 3.2. с подробными комментариями, а также фрагменты MAP-файлов с результатами компоновки и размещения линкером для всех исходных текстов программ. При этом допускается совмещение модифицированных исходных текстов в одной программе. Каждая измененная или вновь написанная строка исходного текста должна иметь комментарий. Комментарии приводятся на русском языке или все - на английском языке.

5. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ИЗУЧЕНИЮ МАКРОАССЕМБЛЕРА A51

Ниже обсуждаются особенности макроассемблера A51, входящего в состав интегрированной среды разработки KEIL μVision2 и приводятся примеры представления результатов для основных разделов работы.

Данная работа выполняется на том же аппаратно-программном комплексе, что и предшествующие работы, и использует описанную ранее демонстрационную программу, имитирующую стартовый секундомер [1].

5.1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Макросы, условное ассемблирование.

Макрос – это одна из конструкций макроассемблера A51. Среда разработки KEIL поддерживает различные макроязыки и в том числе полностью поддерживает язык Standard Assembler Macros [2].

Макроопределением (или макросом) называется участок программы, которому присвоено имя и который ассемблируется всякий раз, когда ассемблер встречает это имя в тексте программы.

Макрос начинается директивой **MACRO** и заканчивается **ENDM,** то, что находится между двумя указанными директивами, называется телом макроса.

Тело макроса при первом проходе ассемблера полностью подставляется в текст программы, а затем транслируется.

Директива **MACRO** объявляет обязательно имя макроса и возможно формальные параметры [2,3]. Макросы должны быть определены в программе до того, как они смогут быть использованы. В строке вызова может быть размещено до 16 параметров. Параметры должны быть разделены запятой и в определении макроса, и в вызове.

При определении в теле макроса могут использоваться специальные директивы, в частности **LOCAL**, **REPT**, **EXITM** и др. [2].

Директива EXITM в основном используется в условных выражениях для немедленного выхода из тела макроса.

Директива REPT определяет число повторений строк, следующих за ней в макросе. Для ограничения строк также используется директива EXITM.

Метки, используемые внутри тела макроса, по умолчанию являются глобальными, и в случае повторного использования макроса будет генерироваться ошибка.

Поэтому метки в макросе следует объявлять локальными, тогда они будут видны только внутри макроса, и макрос может использоваться повторно.

Для этого в первой строке макроса необходимо расположить директиву **LOCAL** и список всех локальных меток (до 16 меток).

Для локальных идентификаторов, определенных в макросе, ассемблер A51 генерирует внутренний идентификатор, который имеет форму ??0000 и инкрементируется каждый раз, когда вызывается макрос. Таким образом, локальные метки становятся уникальными.

Имена меток можно также передавать в качестве параметров макроса (при этом они должны быть различны при каждом вызове).

.*Условное ассемблирование* предоставляет возможность игнорировать тот или иной участок программы в зависимости от выполнения условий в элементах управления. Это может использоваться, чтобы реализовать разные версии программы или разные модели памяти в одном исходном файле.

Подобный участок программы отделяется директивами условного ассемблирования **IF**, **ELSEIF**, **ELSE** и **ENDIF** [2]:

IF – транслировать блок, если условие верно;

ELSE – транслировать блок, если условие в предыдущем IF было ложным;

ELSEIF – транслировать блок, если условие верно, а в предыдущем IF или ELSEIF условие было ложным;

ENDIF – отмечает конец блока.

Элементы управления (идентификаторы) для условного ассемблирования могут быть переопределены любое число раз в теле исходного файла.

Макроассемблер A51 допускает различное определение директив IF, ELSEIF, ELSE и ENDIF: с предваряющим знаком доллара ($) или как в компиляторе C51- без него.

Когда предварительно задан знак доллара, условная директива может иметь доступ только к идентификаторам, определенным директивами **SET** и **RESET**.

Директивы SET и RESET могут быть использованы в строке вызова ассемблера, чтобы установить или обнулить условия, которые тестируются директивами IF и ELSEIF.

Остальные инструкции условного ассемблирования разрешены только внутри исходного файла и не могут быть частью строки вызова ассемблера.

Когда условная директива задана без знака доллара, она может иметь доступ ко всем идентификаторам, за исключением определенных посредством директив SET и RESET.

Такие идентификаторы можно определить, например, директивой #define или директивой EQU.

Блоки IF могут быть вложенными (максимально до 10 уровней). Если блок не транслируется, условные блоки, вложенные в него, также пропускаются.

Сегменты, модульное программирование.

Сегменты – это блоки кода или данных, которые создаются ассемблером в памяти на основе исходного текста программы. Сегменты используют, чтобы локализовать код программы, константные данные и переменные в соответствующих областях памяти.

Сегменты могут быть перемещаемыми или абсолютными.

Перемещаемый сегмент имеет имя, тип и может иметь другие атрибуты [2]. Перемещаемые сегменты называют также именованными сегментами. Имя каждого сегмента внутри модуля должно быть уникально. Линкер BL51 соединяет сегменты, имеющие один и тот же тип.

Сегменты одного типа с одинаковыми именами из разных модулей программы являются частями одного и того же сегмента и называются частичными сегментами. Такие сегменты объединяются в один сегмент линкером при компоновке модулей.

Отличие абсолютного сегмента от перемещаемого состоит в указании адреса размещения этого сегмента в памяти явным образом.

Перемещаемые сегменты создаются с помощью директивы **SEGMENT**, которой необходимо указать имя создаваемого сегмента (слева от директивы) и его тип, т.е. класс используемой памяти (справа от директивы). Дополнительно в строке можно указать параметры, определяющие, какие опции могут быть использованы линкером для перемещения и размещения сегмента [2].

Для того чтобы ассемблер считал созданный перемещаемый сегмент активным и использовал его для последующих кода или данных, необходимо воспользоваться директивой **RSEG**. Справа от RSEG указывается имя сегмента.

Сегмент считается активным, пока не встретится очередная директива RSEG или директива, объявляющая абсолютный сегмент.

Абсолютные сегменты создаются с помощью директив **CSEG, DSEG, XSEG, ISEG** и **BSEG** для различных типов памяти. Справа от директивы, как правило, располагается конструкция - AT *address*, определяющая адрес начала сегмента. Если она отсутствует,

адрес будет определяться счетчиком, который создается ассемблером A51 для каждого типа памяти и вначале обнуляется.

Абсолютный сегмент не может быть объединен с другим сегментом.

Абсолютные сегменты используют, в том случае если нужно иметь доступ к определенному месту памяти, или когда хочется поместить код программы или константные данные в определенные адреса памяти.

Директивы CSEG, DSEG, ISEG, BSEG и XSEG выбирают абсолютный сегмент внутри пространств памяти CODE, DATA, IDATA, BIT или XDATA.

Аналогичные ключевые слова используются для класса (типа) памяти в объявлении перемещаемого сегмента:

CODE - кодовое пространство (адрес 0000h.. 0FFFFh).

CONST - то же кодовое пространство, но только для констант; доступ к нему осуществляется инструкцией MOVC.

DATA - пространство данных с прямой адресацией (адрес 0…7Fh и пространство SFR-регистров).

IDATA - пространство данных с косвенной адресацией (адрес 0…0FFh).

BIT - битовое пространство (адрес 20h…2Fh).

XDATA - внешняя память данных (адрес 0000h.. 0FFFFh); доступ к пространству осуществляется посредством инструкции MOVX.

Наибольший эффект от применения сегментов можно получить при написании текста программы с использованием программных модулей.

Модуль – это отдельный файл, содержащий часть кода программы, который может транслироваться независимо. Он может содержать один или более сегментов (или частичных сегментов).

Модули могут быть неперемещаемыми (содержат хотя бы один абсолютный сегмент) и перемещаемыми (содержат только перемещаемые сегменты).

Часто программный модуль оформляется в виде отдельного перемещаемого сегмента. Это позволяет возложить на редактор связей компоновку программы оптимальным образом.

При использовании абсолютных сегментов компоновку и размещение приходится делать вручную. Это не всегда удобно, тем более что в процессе написания программы размер программных модулей постоянно меняется, и в этом случае требуется вводить защитные области неиспользуемой памяти между модулями.

Для корректной сборки модулей в процессе трансляции в единую программу используется межмодульный интерфейс.

Директива **PUBLIC** позволяет экспортировать переменные в другие модули. Метка, объявленная директивой **PUBLIC**, становится глобальной, т.е. доступной для других модулей программы. С помощью данной директивы можно объявлять имена процедур, переменные, а также константы, определенные директивой **EQU**.

Для корректной работы модуля, использующего метки, переменные и константы, объявленные в других модулях, необходимо перечислить их после директивы **EXTRN**. При применении этой директивы следует также указывать тип памяти, используемый идентификаторами: **CODE**, **DATA**, **IDATA**, **BIT**, **XDATA**, или для константы **NUMBER.**

В конце каждого модуля должна присутствовать обязательная директива ассемблера **END**.

Модуль должен содержать определения всех идентификаторов, которые используются внутри модуля, явно или с использованием межмодульного интерфейса.

Однако идентификаторы, объявленные внутри файла определения процессора, который подключается директивой **#INCLUDE** к главному модулю, могут не определяться в других модулях программы. Среда разработки при наличии межмодульного интерфейса правильно оценит ситуацию и выполнит трансляцию отдельного модуля, хотя и выдаст предупреждения.

Главным модулем в данном случае назван модуль, которому передается управление при запуске программы, т.е. абсолютный модуль, включающий кодовый сегмент с нулевым адресом.

Следует отметить, что для ассемблера A51 (в отличие от компилятора C51) область видимости битовых переменных не ограничивается текущим модулем, и они могут быть объявлены глобальными.

5.2 РЕЗУЛЬТАТЫ РАБОТЫ И ИХ АНАЛИЗ

Пример модифицированной демонстрационной программы timer.a51 приведен в приложении. Измененные строки программы в приложении и далее по тексту выделяются жирным шрифтом.

В соответствии с задачами исследования, сформулированными выше (п.3.1), во-первых, создадим дополнительные сегменты памяти.

В демонстрационной программе изначально присутствует абсолютный сегмент данных, введенный строкой DSEG at 20h, и абсолютный кодовый сегмент с нулевым начальным адресом, объявленный директивой CSEG.

В начале создадим именованный, а значит перемещаемый, сегмент стека.
Объявление

```
stackArea:  ds  48
```

исходной программы заменим следующим образом. Введем строку

```
STACKAREA  SEGMENT  IDATA
```

,
затем сделаем сегмент активным директивой RSEG:

```
RSEG  STACKAREA
```

и зарезервируем вполне достаточное (вместо 48 байт) место в памяти командой

```
DS 5
```

.

Далее в тексте программы вместо команды mov sp,#StackArea используем команду:

```
mov  sp,#STACKAREA-1
```

.

Во-вторых, для процедуры обработчика прерываний от таймера T0 аналогично введем именованный сегмент в кодовой памяти:

```
int0_code_seg  SEGMENT CODE
               RSEG  int0_code_seg
               USING  1
```

Заметим, что директивой в третьей строке кода резервируется регистровый банк 1 для обработчика прерываний. Какой банк регистров будет реально использоваться в программе, потребуется ли в этом случае дополнительно выполнить явное переключение битов в PSW-регистре, можно проверить, например, в окне memory среды разработки, предварительно запустив симулятор (результаты исследования следует привести в отчете).

Далее, процедуры delay, waitPressKey и waitReleseKey размещаются в сегменте с абсолютной адресацией с адреса 0ACh или перемещаемом сегменте. Это определяется значением константы EXPR1 и использованием директив условного ассемблирования.

Чтобы продемонстрировать два варианта написания условных директив: чисто ассемблерный – со знаком доллара, и заимствованный у компилятора C51 – без знака доллара, ниже по тексту приводится последний вариант (для лучшего понимания фрагмента), а в приложении в примере использован первый вариант.

Для этого зададим начальное значение константы EXPR1, например, с помощью директивы EQU, равным нулю, в тексте программы - явно, и добавим фрагмент кода с условным ассемблированием:

```
EXPR1        EQU        0
IF (EXPR1)
```

```
    CODE_SEG1    SEGMENT  CODE
                 RSEG     CODE_SEG1
    ELSE
    CSEG at 00ACH
    ENDIF
```

Анализ MAP-файла должен показать - какой именно начальный адрес будет присвоен данному кодовому сегменту.

Для процедуры CounterUpdate выделяется перемещаемый сегмент кода с именем CODE_SEG2:

```
  CODE_SEG2          SEGMENT  CODE
                     RSEG     CODE_SEG2
CounterUpdate:
…                              ,
```

а процедура DisplayDecNumber помещается в абсолютный кодовый сегмент с адреса 097h:

```
  CSEG at 0097H
DisplayDecNumber:
                …
```

Дополнительно в процедуре DisplayDecNumber предусмотрена возможность изменения счета таймера с помощью директив условного ассемблирования. При значении константы EXPR1, равным нулю, счет как и в исходной программе ведется от 00 до 99. В противном случае счет будет вестись в обратном порядке.

Кроме этих изменений, в демонстрационной программе вводится также макроопределение delay и модифицируются процедуры waitPressKey и waitReleaseKey. Макрос определяется строкой delay MACRO , вслед за которой непосредственно помещается строка, объявляющая локальные метки delay0 и delay1. Далее тело макроса содержит инструкции процедуры задержки. Заканчивается макрос директивой ENDM.

```
  delay       MACRO
  LOCAL delay0, delay1
        mov   r0,#025h
  delay0:
        mov   r1,#0FFh
  delay1:
        djnz  r1,delay1
        djnz  r0,delay0
              ENDM
```

Вызов макроопределения с использованием его имени delay производится хотя только в одном сегменте (причем с абсолютной адресацией), однако дважды - в процедурах waitPressKey и waitReleseKey. В каждой из этих процедур кодовые метки в конечном счете при подстановке получат уникальные адреса.

В итоге выполненных операций могла бы быть получена модифицированная согласно п.3.1 одномодульная программа timer.a51 (в приложении она не приводится).

Чтобы узнать, по каким адресам после построения проекта будут расположены сегменты, можно посмотреть карту памяти в MAP-файле:

```
INPUT MODULES INCLUDED:
  timer.obj (timer)

LINK MAP OF MODULE:  lab3 (timer)
            TYPE    BASE      LENGTH    RELOCATION   SEGMENT NAME
            -----------------------------------------------------
            * * * * * * *   D A T A   M E M O R Y   * * * * * * *
            REG     0000H     0008H     ABSOLUTE     "REG BANK 0"
```

```
REG     0008H     0008H     ABSOLUTE     "REG BANK 1"
        0010H     0010H                  *** GAP ***
DATA    0020H     0006H     ABSOLUTE
IDATA   0026H     0005H     UNIT         ?STACK
* * * * * * *   C O D E   M E M O R Y   * * * * * * *
CODE    0000H     011EH     ABSOLUTE
        011EH     3EE2H                  *** GAP ***
CODE    4000H     0020H     UNIT         INT0_CODE_SEG
```

Из карты памяти видно, что в программе задействовано два регистровых банка. Дополнительный банк 1, вероятно, будет обслуживать обработчик прерывания в кодовом сегменте `INT0_CODE_SEG,` что и предполагалось при введении директивы USING 1.

Байтовые переменные расположены с абсолютного адреса 20h. После них с адреса 26h линкер поместил стек.

В кодовой памяти код располагается с нулевого адреса. Сегмент INT0_CODE_SEG линкер расположил с адреса 4000h (16К). Перед загрузкой кода программы можно вручную поменять этот адрес. Однако в нашем случае объем памяти программ МК-системы в программно-аппаратном комплексе - 32К. Поэтому загрузка кода выполнится без каких-либо изменений.

Продолжим исследование далее, и в соответствии с задачей раздела 3.2 разобьем модифицированную программу на модули, а в приложение поместим уже окончательный вариант исходного текста модифицированной программы (приложение 1).

Создадим четыре модуля (файла) программы.

В первом модуле с именем module_1, показанном ниже в приложении 1, оставим определение переменных, констант; резервирование стека, таблицу векторов прерываний; процедуру StartUp и main, а также обработчик прерывания от таймера 0 – процедуру Clock. В этом сегменте будут определены абсолютный сегмент данных, перемещаемый сегмент стека и два сегмента кода: абсолютный с нулевого адреса – точка входа для исполняемого кода, и перемещаемый int0_code_seg - для обработчика прерывания Clock. Наличие в этом модуле точки входа делает этот модуль главным модулем. Поэтому только в нем подключаются файлы определения процессора Reg515.inc и add_on.inc.

Для корректной работы линкера при сборке и размещении модулей в абсолютном объектном файле необходимо использовать межмодульный интерфейс для каждого модуля.

Межмодульный интерфейс первого модуля с помощью директивы EXTRN перечисляет кодовые метки (ключевое слово CODE), объявленные в других модулях, распространяя поле их видимости на данный модуль, а директива PUBLIC делает глобальными переменные, объявленные в этом модуле (для директивы PUBLIC тип переменных не указывается):

EXTRN CODE (DisplayUpdate,waitReleaseKey,waitPressKey,CounterUpdate, DisplayDecNumber)

PUBLIC Digit0, Digit1, DisplayPhase, LCD1, LCD0, BUTTON, Go, Tick, Sec

Заканчивается модуль обязательной директивой **END**.

Во втором модуле в кодовом сегменте размещаем процедуру обновления изображения на ЖК-индикаторе DisplayUpdate и обязательно вместе с таблицей перекодировки DisplayTable кода символа в семи сегментный код изображения.

Интерфейс модуля выглядит следующим образом:

EXTRN DATA (Digit0, Digit1, DisplayPhase, LCD1, LCD0)

PUBLIC DisplayUpdate

В директиве EXTRN для переменных указывается их тип – DATA.

Третий модуль будет содержать процедуры нажатия waitPressKey и отпускания waitReleaseKey клавиш вместе с подпрограммой задержки delay, размещаемые в абсолютном или относительном сегменте в зависимости от константы EXPR1. Интерфейс модуля импортирует битовую переменную BUTTON из первого модуля и экспортирует две метки:

```
EXTRN   BIT (BUTTON)
PUBLIC      waitPressKey, waitReleaseKey
```

В четвертый модуль останется поместить процедуру инкремента двоичного счетчика секунд CounterUpdate - в сегменте CODE_SEG2 и модернизированную процедуру кода изображения десятичного числа DisplayDecNumber - в абсолютном сегменте CSEG at 0097H.

Интерфейс модуля в данном случае аналогично объявляет внешние байтовые переменные типа DATA и делает глобальными две кодовые метки:

```
EXTRN       DATA (Go, Tick, Sec, Digit1, Digit0)
PUBLIC CounterUpdate, DisplayDecNumber
```

После выполнения задания в соответствии с п.3.2.3. и п.3.2.4 будет получен MAP-файл, представленный в приложении 2.

Карта памяти в MAP-файле выглядит следующем образом:

```
TYPE    BASE      LENGTH    RELOCATION   SEGMENT NAME
-----------------------------------------------------
* * * * * * *  D A T A   M E M O R Y  * * * * * * *
REG     0000H     0008H     ABSOLUTE     "REG BANK 0"
REG     0008H     0008H     ABSOLUTE     "REG BANK 1"
        0010H     0010H                  *** GAP ***
DATA    0020H     0006H     ABSOLUTE
IDATA   0026H     0005H     UNIT         ?STACK
* * * * * * *  C O D E   M E M O R Y  * * * * * * *
CODE    0000H     0097H     ABSOLUTE
CODE    0097H     0015H     ABSOLUTE
CODE    00ACH     0018H     ABSOLUTE
        00C4H     3F3CH                  *** GAP ***
CODE    4000H     0020H     UNIT         INT0_CODE_SEG
CODE    4020H     0020H     UNIT         MODULE_2
CODE    4040H     001CH     UNIT         MODULE_4_1
```

Как можно заметить, адреса абсолютных кодовых сегментов соответствуют описанным выше адресам с учетом установленного значения константы EXPR1 равным нулю. Адреса именованных сегментов INT0_CODE_SEG, MODULE_2 и MODULE_4_1 в абсолютном объектном коде выбраны линкером.

5.3. ИСТОЧНИКИ ИНФОРМАЦИИ

1. «Подготовка и трансляция МП программ с использованием простейшего Intel совместимого кросс ассемблера». - Методические указания к лабораторным работам по курсу «Микропроцессорные средства и системы», Авт.: Лобан В.И., Цветков В.А., Электрон. текст. дан. – СПб: Б. и., 2004.
URL:ftp://ftp.lunilib.neva.ru/dl/local/874.pdf.
2. http://www.keil.com/support/man/docs/a51/default.htm
3. http://www.keil.com/support/man/docs/uv3/uv3_cm_asm.htm

5.4. ПРИЛОЖЕНИЕ

Приложение 1. Исходные тексты модулей программы (программа timer.a51 модифицирована в соответствии с заданием п.3.1 и п.3.2).

Модуль 1:

```
$NOMOD51
$INCLUDE (Reg515.inc)
$INCLUDE (add_on.inc)

NAME        module_1

EXTRN CODE  (DisplayUpdate, waitReleaseKey, waitPressKey, CounterUpdate, DisplayDecNumber)
PUBLIC Digit0, Digit1, DisplayPhase, LCD1, LCD0, BUTTON, Go, Tick, Sec

T0COUNT EQU    50000        ; коэффициент  пересчета таймера 0 для частоты
                            ; (кварц - 12 МГц) возникновения прерывания (тиков) в процедуре clock -
                            ; 20 Гц (12MHz/12/50=50000)

BUTTON  BIT    P5.7         ; состояние кнопки (0 - нажата)
LCD0    DATA   P1           ; младшая цифра ЖК-индикатора
LCD1    DATA   P4           ; старшая цифра ЖК-индикатора

     DSEG   at 020h
Tick:         ds     1 ; счетчик прерывания от таймера
Sec:          ds     1 ; счетчик секунд
Go:           ds     1 ; флаг запуска счетчиков
DisplayPhase:  ds     1 ; флаг инверсии вывода данных на ЖК-индикатор
Digit0:       ds     1 ; код изображения младшего разряда индикатора
Digit1:       ds     1 ; код изображения старшего разряда индикатора

STACKAREA     SEGMENT IDATA        ; Стек создается в памяти IDATA RAM.
              RSEG   STACKAREA             ;переключение в STACKAREA сегмент.
              DS     5             ; резервируем для стека
                                   ; 5 байт в этом примере.

     CSEG
     ORG    RESET         ;
     jmp    StartUp       ;
                          ;
     ORG    EXTI0         ;
     reti                 ;
                          ;
     ORG    TIMER0        ;
     jmp    Clock         ;

                          ;
     ORG    EXTI1         ;
     reti                 ;
                          ;
     ORG    TIMER1        ;
     reti                 ;
                          ;
     ORG    SINT          ;
     reti                 ;
                          ;
     ORG    TIMER2        ;
     reti                 ;
                          ;
     ORG    ADCONV        ;
     reti                 ;
                          ;
     ORG    EXTI2         ;
     reti                 ;
                          ;
     ORG    EXTI3         ;
```

```
        reti            ;
                        ;
        ORG     EXTI4   ;
        reti            ;
                        ;
        ORG     EXTI5   ;
        reti            ;
                        ;
        ORG     EXTI6   ;
        reti            ;
                        ;
            org     50h
;-------------------------------------------------------------------------------
StartUp:                ;
        mov     IEN0,#00000000b ; запрещаем прерывания на
                        ; время инициализации МКС
                        ;
        mov     IP0,#00010010b  ; устанавливаем приоритеты прерываний:
                        ; в порядке убывания - T0,SP,EXT0,EXT1,T1
        mov     sp,#STACKAREA-1  ; инициализируем указатель стека
                        ; программируем таймеры:
        mov     TCON,#00000101b ; TR0 = 0, TR1 = 0, INT0,1 front
        mov     TMOD,#00010001b ; T0: C/T=0, MODE=1, GATE=0
                        ; T1: C/T=0, MODE=1, GATE=0
                        ;
        setb    ET0             ; разрешаем прерывания от таймера T0
        setb    EAL             ; разрешаем обработку прерываний
                        ;
        clr     a               ; a = 0
        call    DisplayDecNumber;

        mov     Go,#00          ; инициализируем
        mov     Sec,#00         ; переменные
        mov     DisplayPhase,#00;
                        ;
        call    clock           ; включаем часы, вызывая процедуру обработчика прерывания ,
                                ; допускается вассемблере
main:                   ;
        call    waitPressKey    ; ждем нажатия клавиши


        mov     Sec,#00         ;
        mov     Tick,#00        ;
        mov     Go,#01          ;   включаем режим пуска секундомера
        call    waitReleaseKey  ; ждем отпускания клавиши
                        ;
        call    waitPressKey    ; ждем нажатия клавиши
        mov     Go,#00          ;   сбрасываем секундомер
        call    waitReleaseKey  ; ждем отпускания клавиши
                        ;
        call    waitPressKey    ; ждем нажатия клавиши
        clr     a               ; a = 0
        call    DisplayDecNumber ; обнуляем секундомер
        call    waitReleaseKey  ; ждем отпускания клавиши
                        ;
        jmp     main            ;  бесконечный цикл
;-------------------------------------------------------------------------------
int0_code_seg   SEGMENT CODE            ; сегмент для обработчика прерывания
            RSEG    int0_code_seg   ; переключаемся в этот кодовый сегмент
            USING   1               ; регистровый банк для подпрограммы прерывания

;-------------------------------------------------------------------------------
```

```
; Обработчик прерываний от таймера 0 (системные часы)
Clock:                  ;
      push   psw        ; запоминаем  текущее состояние процессора
      push   acc        ;
                        ;
      call   DisplayUpdate   ; обновляем данные на ЖК-индикаторе
      call   CounterUpdate   ; обновляем состояние счетчиков
                        ;
      clr    C          ;
      clr    TR0        ; запрещаем работу таймера T0
      mov    a,TL0      ; программируем T0
      subb   a,#LOW(T0COUNT-7)
      mov    TL0,a      ;
      mov    a,TH0      ;
      subb   a,#HIGH(T0COUNT-7)
      mov    TH0,a      ;
      setb   TR0        ;
      pop    acc        ;
      pop    psw        ; восстанавливаем состояние процессора до
      reti              ; прихода прерывания

          END           ; конец текущего модуля
```

Модуль 2:

```
NAME        module_2

EXTRN   DATA (Digit0, Digit1, DisplayPhase, LCD1, LCD0)  ; устанавливаем межмодульный
PUBLIC          DisplayUpdate                            ; интерфейс

MODULE_2   SEGMENT CODE          ; вводим перемещаемый кодовый сегмент
       RSEG    MODULE_2   ; переключаемся в этот кодовый сегмент
;-------------------------------------------------------------------------------
; Процедура обновления изображения на ЖК-индикаторе
DisplayUpdate:              ;
      mov    a,Digit0      ;
      add    a,#(DisplayTable-$-2-1) ; адресуем код изображения знака
      movc   a,@a+pc       ; заносим в Acc семи сегментный код изображения младшей десятичной цифры
      xrl    a,DisplayPhase  ; при необходимости инвертируем код
      mov    LCD0,a        ; выводим код для младшего разряда
                           ; индикатора
      mov    a,Digit1      ;
      add    a,#(DisplayTable-$-2-1) ; адресуем код изображения знака
      movc   a,@a+pc       ; заносим в Acc семи сегментный код изображения старшей десятичной цифры
      xrl    a,DisplayPhase  ; при необходимости инвертируем код
      mov    LCD1,a        ; выводим код для старшего разряда
                           ; индикатора
      xrl    DisplayPhase,#0FFh; инвертируем флаг I
      ret                  ;

;-------------------------------------------------------------------------------
; Таблица перекодировки кода символа в семи сегментный код изображения.
; Таблица должна располагаться непосредственно за процедурой DisplayUpdate,
; чтобы не возникало возможных ошибок при трансляции
DisplayTable:
;            hgfedcba          код  DIG LAT RUS
      DB     00111111b      ;| 00h | 0 | O | O |
      DB     00000110b      ;| 01h | 1>| I>|   |
      DB     01011011b      ;| 02h | 2 |   |   |
      DB     01001111b      ;| 03h | 3 |   | З Э |
      DB     01100110b      ;| 04h | 4 |   | Ч |
      DB     01101101b      ;| 05h | 5 | S |   |
      DB     01111101b      ;| 06h | 6 |   | Б |
```

```
        DB      00000111b       ;| 07h | 7 |   |   |
        DB      01111111b       ;| 08h | 8 | B | B |
        DB      01101111b       ;| 09h | 9 |   |   |
;       DB      01110111b       ;| 0Ah |   | A | A |
;       DB      01111100b       ;| 0Bh |   | b | Ь |
;       DB      00111001b       ;| 0Ch |   | C | C |
;       DB      01011110b       ;| 0Dh |   | d |   |
;       DB      01111001b       ;| 0Eh |   | E | E |
;       DB      01110001b       ;| 0Fh |   | F |   |
            END                                     ; конец текущего модуля
```

Модуль 3:

```
NAME            module_3

EXTRN   BIT (BUTTON)                            ; устанавливаем межмодульный
PUBLIC          waitPressKey, waitReleaseKey    ; интерфейс

$IF (EXPR1)
CODE_SEG1   SEGMENT     CODE        ; вводим перемещаемый кодовый сегмент
            RSEG    CODE_SEG1       ; переключаемся в этот кодовый сегмент
$ELSE                                ; иначе
CSEG at 00ACH                       ; вводим абсолютный кодовый сегмент
$ENDIF

;-------------------------------------------------------------------------------
; Задержка, устраняющая дребезг контактов
delay   MACRO                       ; вводим макроопределение
        LOCAL delay0, delay1        ; устанавливаем локальные ссылки
    mov     r0,#025h        ;
delay0:                 ;
    mov     r1,#0FFh        ;
delay1:                 ;
    djnz    r1,delay1       ;
    djnz    r0,delay0       ;
        ENDM                        ; конец макроса
;-------------------------------------------------------------------------------
; Ожидание нажатия клавиши
waitPressKey:
    delay                           ;вызываем макрос
    jb      BUTTON,waitPressKey
    ret

;-------------------------------------------------------------------------------
; Ожидание отпускания клавиши
waitReleaseKey:
        delay                       ;вызываем макрос
    jnb     BUTTON,waitReleaseKey
    ret

        END                         ; конец текущего модуля
```

Модуль 4:

```
NAME            module_4

EXTRNDATA   (Go, Tick, Sec, Digit1, Digit0)     ; устанавливаем межмодульный
PUBLIC CounterUpdate, DisplayDecNumber          ; интерфейс

CODE_SEG2   SEGMENT CODE        ; вводим перемещаемый кодовый сегмент
        RSEG    CODE_SEG2       ; переключаемся в этот кодовый сегмент
```

```
;------------------------------------------------------------------------------

; Процедура увеличивает значение счетчика Sec на единицу каждую секунду, после
; установки Go != 0. При достижении значения Sec = 100, Sec обнуляется.
CounterUpdate:
      mov    a,Go          ;
      jz     c3            ; переход, если GO == 0
      mov    a,Tick        ; Tick -> a
      inc    a             ; a = Tick + 1
      cjne   a,#20,c1      ; если a != 20, то переход
      clr    a             ; a = 0
c1:   mov    Tick,a        ; a -> Tick
      jnz    c3            ; переход, если a != 0
      mov    a,Sec         ; Sec -> a
      inc    a             ; a = Sec + 1
      cjne   a,#100,c2     ; если a != 100, то переход
      clr    a             ; a = 0
c2:   mov    Sec,a         ; a -> Sec
      ;call   DisplayDecNumber; отображаем Sec на ЖКИ
          call      DisplayDecNumber
c3:   ret
;------------------------------------------------------------------------------

; Формирование в видеобуфере 7-сегментного кода изображения десятичного
; числа из аккумулятора (00 <= A <= 99)
CSEG at 0097H           ; вводим абсолютный кодовый сегмент
DisplayDecNumber:           ;
      push   b            ;
      mov    b,#100       ; выделяем два младших десятичных
      div    ab           ; разряда из отображаемого числа
      mov    a,b          ;
      mov    b,#10        ; загружаем 10 в регистр B
      div    ab           ; делим исходное число на B

     mov    Digit1,a      ; в A старший, в B младший десятичный разряд
$IF (EXPR1)
          mov     a, #9           ; фрагмент программы
          subb    a, Digit1       ; с установкой обратного счета
          mov     Digit1, a       ; секундомера
$ENDIF
          mov    a,b          ; сохраняем разряды
      mov    Digit0,a      ; в видеобуфере
$IF (EXPR1)
          mov     a, #9           ; фрагмент программы
          subb    a, Digit0       ; с установкой обратного счета
          mov     Digit0, a       ; секундомера
$ENDIF
     pop    b            ;
      ret                ;
          END             ; конец текущего модуля
```

Приложение 2. Часть map-файла timer.M51, сгенерированного линкером при сборке многомодульной программы, текст которой приведен в приложении 1 (в соответствии с заданием п.3.2.3 и п.3.2.4).

```
BL51 BANKED LINKER/LOCATER, INVOKED BY:
D:\KEIL\C51\BIN\BL51.EXE .\module4.OBJ, .\MODULE1.OBJ, .\module2.OBJ, .\module3.OBJ TO lab3
RAMSIZE (256)

INPUT MODULES INCLUDED:
  .\module4.OBJ (MODULE_4)
```

```
 .\MODULE1.OBJ (MODULE_1)
 .\module2.OBJ (MODULE_2)
 .\module3.OBJ (MODULE_3)

LINK MAP OF MODULE:  lab3 (MODULE_4)

        TYPE    BASE     LENGTH    RELOCATION   SEGMENT NAME
        -----------------------------------------------------

        * * * * * * *  D A T A   M E M O R Y   * * * * * * *
        REG     0000H    0008H    ABSOLUTE     "REG BANK 0"
        REG     0008H    0008H    ABSOLUTE     "REG BANK 1"
              0010H    0010H             *** GAP ***
        DATA    0020H    0006H    ABSOLUTE
        IDATA   0026H    0005H    UNIT         ?STACK

        * * * * * * *  C O D E   M E M O R Y   * * * * * * *
        CODE    0000H    0097H    ABSOLUTE
        CODE    0097H    0015H    ABSOLUTE
        CODE    00ACH    0018H    ABSOLUTE
              00C4H    3F3CH             *** GAP ***
        CODE    4000H    0020H    UNIT         INT0_CODE_SEG
        CODE    4020H    0020H    UNIT         MODULE_2
        CODE    4040H    001CH    UNIT         MODULE_4_1
```

6. ПРИМЕР ТЕСТОВЫХ ВОПРОСОВ

Тест 1.

Ознакомьтесь с фрагментами ассемблерной программы, в которую вводятся макроопределения. Программа разрабатывается с использованием макроассемблера A51 интегрированной среды Keil µVision2.

Выберете фрагмент программы без синтаксических ошибок из следующих вариантов:

А)

```
delay:  MACRO
        mov     r0,#025h
delay0:
        mov     r1,#0FFh
delay1:
        djnz    r1, delay1
        djnz    r0, delay0
        ret
        ENDM
```

Б)

```
delay:  MACRO
        mov     r0,#025h
local   delay0
delay0:
        mov     r1,#0FFh
local   delay1
delay1:
        djnz    r1, delay1
        djnz    r0, delay0
```

```
        ret
        ENDM
```

B)

```
delay:  MACRO
local   delay0, delay1
        mov     r0,#025h
delay0:
        mov     r1,#0FFh
delay1:
        djnz    r1, delay1
        djnz    r0, delay0
        ret
        ENDM
```