TESSA 0.1

КОМПИЛЯТОР АССЕМБЛЕРА МИКРОКОНТРОЛЛЕРОВ ТЕСЕЙ

Язык **Ассемблер** и компилятор **TESSA 0.1** предназначены для разработки и отладки прикладных программ микроконтроллеров, построенных на основе ядра **ТЕСЕЙ**, в том числе МК **КР1878ВЕ1** (An15E03). Отладка программ осуществляется либо на персональном компьютере в режиме эмуляции, либо на специальном аппаратном отладочном модуле. Для выполнения программирования и отладки необходимы IBM-совместимый персональный компьютер, пакет программ **TESSA 0.1** и любой программатор, удовлетворяющий изложенным в настоящем издании требованиям. При использовании отладочного модуля программатор не требуется.

Алфавит языка

Алфавит языка ассемблера может быть разделен на 4 основные группы:

группа 1 – цифровые символы: **0123456789**

группа 2 – алфавитные символы: **ABCDEFGHIJKLMNOPQRSTUVWXYZ**

a b c d e f g h i j k l m n o p q r s t u v w x y z

группа 3 – символы операций: **_ = - + * / & ! ^ \ ( )**

группа 4 – служебные символы: **. % # : ; $ @**

Структура программы и порядок трансляции

Независимо от модели микроконтроллера первые 16 адресов памяти команд процессора выделены для размещения векторов прерываний. Вектор прерывания содержит команду перехода на подпрограмму обработки прерывания.

Допускается трансляция программы, состоящей из модулей, размещенных в разных файлах. В данной версии транслятора областью видимости констант и меток, объявленных в любом из модулей, является вся программа. Каждый программный модуль, размещенный в отдельном файле, должен заканчиваться инструкцией **.end**. Вся программа должна завершаться инструкцией **.end** или **.endc**.

Расположение в адресном пространстве результирующего кода программных модулей, содержащихся в различных, но совместно транслируемых файлах, определяется последовательностью трансляции модулей. Файлы исходного текста должны иметь расширение **mic**.

Запуск компилятора:

tessa [-key] имя_файла_1[+имя_файла_2[+...]],

где: **имя_файла_1** – имя головного файла

имя_файла_2 – имя файла, содержащего дополнительный программный модуль.

-key – ключ, определяющий режим компиляции.

Компилятор генерирует файлы следующего типа:

.sav – файл исполняемого кода;

.lst – файл листинга;

.sym – служебный файл для обеспечения режима символьной отладки;

.sdf – служебный файл для обеспечения режима символьной отладки;

Ключи компиляции:

l – разрешает генерацию файла листинга;

d – разрешает генерацию служебных файлов для обеспечения режима символьной отладки ;

i:path – определяет путь до директории, содержащей библиотеку **include** файлов.

Метки

Метка содержит не более 32 алфавитно-цифровых символов, должна начинаться алфавитным символом (группа 2) и заканчиваться служебным символом **:**.

Пример: **LABLE:**

labl_1:

Метка может располагаться как в строке, содержащей команду, на которую осуществляется ссылка, так и в предыдущих строках.

Пример: **lab_1:**

lab_2: mov %a1,%a2

Обе метки ссылаются на один и тот же адрес (команду).

Кроме символьных меток, описанных выше, существуют так называемые локальные метки, область действия которых распространяется на пространство адресов между двумя символьными метками. Локальная метка начинается с символа **$** и содержит только цифровые символы (группа 1).

Пример:
```
lab_1:  mov  %a1,%a2
$1:     mov  %b2,%c3
lab_2:  inc  %b1
$1:     ...
```

Локальная метка не должна содержать более 5 цифр.

Особый случай составляют метки в макроопределениях. Первым символом меток данного типа должен быть служебный символ **@**. Данные метки используются в режиме автоматической генерации меток при раскрытии макроопределений.

Константы

Имя символьной константы содержит алфавитно-цифровые символы (группы 1, 2). Первый символ имени должен принадлежать множеству алфавитных символов (группа 2). Длина имени константы не должна превышать 32 символов.

Значение константе может быть присвоено в любом месте программы, но только один раз. Далее значение, присвоенное имени, действует на протяжении всей программы, включая совместно транслируемые модули.

Значение константы может быть присвоено как в явном числовом виде, так и в виде арифметического выражения. Задание константы в явном числовом виде:

const=12 – задание константы в десятиричной системе счисления;

const=12o – задание константы в восьмиричной системе счисления;

const=01010101b – задание константы в двоичной системе счисления;

const=12h – задание константы в шестнадцатиричной системе счисления.

Шестнадцатеричная константа должна начинаться с цифры. В случае необходимости используйте в качестве первого символа шестнадцатеричной константы незначащий **0**. Константа может быть определена арифметическим выражением.

Значение константы не должно превышать 8 разрядов. В случае использования констант в теле команд переходов разрядность констант увеличивается до 12.

Определение служебных регистров

Для обращения к служебным регистрам используется конструкция: **#n**, где:

– префикс обращения к служебному регистру;

n – номер служебного регистра **0** ¸ **7**.

Номер служебного регистра определяет соответствующий регистр из множества {**SR0** ¸ **SR7**}. Для служебных регистров **SR0**, **SR1**, **SR2** и **SR3** допускается задание номера символами **a**, **b**, **c**, **d** соответственно. **SR0** ¸ **SR3** являются регистрами баз сегментов, **SR4** и **SR5** – регистрами адреса косвенного регистра **IR0**, а **SR6** и **SR7** – регистрами адреса косвенного регистра **IR1**.

Разряды **6** и ***7*** в **SR5** определяют режим работы косвенного регистра **IR0**:

00 – отключение косвенного режима; обращение к регистру **%d6** аналогично обращению к любому другому регистру памяти данных;

01 – косвенный автоинкрементный режим;

10 – косвенный автодекрементный ражим;

11 – косвенное обращение без изменения значения адреса.

Разряды **6** и **7** в **SR7** определяют режим работы косвенного регистра **IR1**.

00 – режим адресации к памяти команд;

01 – косвенный автоинкрементный режим;

10 – косвенный автодекрементный ражим;

11 – косвенное обращение без изменения значения адреса.

Пример: **#1** – определение регистра **SR1**;

#b – определение регистра **SR1**.

Определение регистров памяти данных

Доступ к ячейкам памяти данных в микроконтроллере осуществляется через механизм окон. Действительный адрес формируется как объединение содержимого сегментного регистра и сегментного индекса. Признаком непосредственного обращения к регистрам памяти является префикс **%**. Используемый сегментный регистр задается в явном виде одним из четырех символов {**a**, **b**, **c** и **d**}. Символы, задающие сегментные регистры, определяют служебные регистры **SR0**, **SR1**, **SR2** и **SR3** соответственно. Сегментный индекс лежит в диапазоне **0** ¸ **7**. Таким образом, для непосредственного обращения к регистру памяти необходимо применить следующую форму записи операнда:

%Xn,

где: **%** – префикс обращения к регистру памяти;

X – символ, определяющий сегментный регистр {**a**, **b**, **c** или **d**};

n – сегментный индекс **0** ¸ **7**.

Пример : содержимое **SR1=8**

%b5

В качестве регистра, задающего сегментный адрес, выбран служебный регистр **SR1**.

Содержимое служебного регистра **SR1** является старшей частью адреса сегмента.

Полный адрес сегмента = **40h**.

Сегментный индекс равен 5. Таким образом выбран адрес **45h**.

Пример программного фрагмента:

ldr **#1,40h;** – запись сегментного адреса в **SR1**; содержимое **SR1** = **8**,

movl **%b5,1;** – обращение к адресу **45h** в команде пересылки литеры.

Обращение к регистрам устройств ввода/вывода выполняется аналогично обращениям к регистрам памяти данных.

Следует особо отметить, что при обращении к регистрам **6** и **7** в сегменте **D** реально происходит обращение к косвенным регистрам **IR0**, **IR1** соответственно.

Пример: **содержимое SR6 = 97h SR7 = 40h**

%d7

Обращение к регистру памяти с адресом **97h** с использованием косвенного регистра **IR1** в косвенном автоинкрементном режиме.

Допустимо задание имени регистра памяти без использования префиксного служебного символа **%**. Таким образом, записи **%a1** и **a1** идентичны.

Размещение данных в памяти команд

Архитектура процессора допускает хранение пользовательских данных в памяти команд. Данные могут быть организованы в виде 16-разрядных слов или байтов.

[метка:] .byte константа [,константа].

[метка:] .word константа/метка [,константа/метка].

При данной форме записи количество констант, следующих за инструкцией резервирования, ограничено только длиной строки. Константы должны быть определены перед их первым использованием.

Текущий счетчик команд

Терм **.** определяет в процессе трансляции текущий счетчик команд. Он может быть использован при определении констант и арифметических выражений.

Пример: **jmp .+1**

В результате выполнения команды будет осуществлен переход к следующей команде.

Использование терма в левой части выражения приводит к изменению счетчика команд на этапе трансляции программы.

Пример: **.=100h**

Новый счетчик команд при трансляции будет равен **100h**.

.=.+100h

Новый счетчик команд примет значение, равное текущему счетчику команд **+100h**.

Арифметические выражения

При определении констант, включая константы, задаваемые в теле команды, а также при изменении значения текущего счетчика команд допускается использование арифметических выражений.

Допустимые арифметические операции:

( ) – изменение приоритета операций;

***** – умножение;

/ – деление;

& – поразрядное **И**;

+ – сложение;

- – вычитание;

**** – поразрядное исключающее **ИЛИ**;

! – поразрядное **ИЛИ**;

^ – инверсия.

При выполнении арифметических операций принят следующий их приоритет:

1 – **( )**

2 – *** / &**

3 – **+ - \ !**

Допустимы унарные операции: **+ - ^** .

Комментарии

Комментарии начинаются с символа **;**. Комментарии могут располагаться в любой позиции строки программы.

Команды

В настоящем разделе рассмотрены структура команд процессора, особые случаи их применения и псевдокоманды. Подробно команды описаны в разделе "Система команд".

Набор команд микроконтроллера может быть разделён на следующие группы:

Двухоперандные команды:

mov – пересылка операнда;

cmp – сравнение операндов;

add – сложение операндов;

sub – вычитание операнда;

and – логическое **И**;

or – логическое **ИЛИ**;

xor – исключающее **ИЛИ**.

Двухоперандные команды выполняют операции над двумя операндами, при этом один из операндов не изменяет своего значения. Операнд, изменяющий свое значение, будем называть приемником (по тексту **dst**), а операнд, не изменяющий значения, назовем источником (по тексту **src**) .

Структура двухопрандной команды:

мнемоника_команды dst,src.

Пример: **mov %a1**,**%b2**

%b2 – операнд источник;

%a1 – операнд приемник;

mov – мнемоника команды, осуществляющей пересылку данных из операнда источника в операнд приемника.

Литеральные команды:

movl – пересылка литеры;

cmpl – сравнение с литерой;

addl – сложение с литерой;

subl – вычитание литеры;

bic – очистка бита;

bis – установка бита;

btg – инверсия бита;

btt – проверка бита;

Литеральные команды производят операции над операндом-приемником и литерой. Результат операции помещается в операнд-приемник. В качестве литеры может использоваться заданное в явном виде числовое значение, ранее определенная константа или арифметическое выражение.

Структура команды:

мнемоника_команды dst, литера.

Пример:
```
const=10h
movl    %b1,const+1
```

где: **const** – константа **10h**;

%b1 – операнд-приемник;

const+1 – литера, заданная арифметическим выражением.

Литеральные команды **movl** и **cmpl** оперируют с 8-разрядными литерами, команды **addl** и **subl** – с 5-разрядными литерами, а команды **bic**, **bis**, **btg** и **btt** – с 4-разрядными литерами.

Литеральные команды **bic**, **bis**, **btg** и **btt**, оперирующие с тетрадами, в трансляторе реализованы в качестве псевдокоманд, явно определяющих положение тетрады в регистре-приемнике, над которой выполняется операция.

В данных командах литера является маской разрядов регистра-приемника над которыми выполняется операция. То есть операция будет выполнена только над теми разрядами регистра-приемника, где соответствующий разряд литеры установлен в **1**.

bich – очистка битов, задаваемых литерой, ведется в старшей тетраде операнда приемника.

bicl – очистка битов, задаваемых литерой, ведется в младшей тетраде операнда приемника.

bish – установка битов, задаваемых литерой, ведется в старшей тетраде операнда приемника.

bisl – установка битов, задаваемых литерой, ведется в младшей тетраде операнда приемника.

btgh – инверсия битов, задаваемых литерой, ведется в старшей тетраде операнда приемника.

btgl – инверсия битов, задаваемых литерой, ведется в младшей тетраде операнда приемника.

btth – проверка битов, задаваемых литерой, ведется в старшей тетраде операнда приемника.

bttl – проверка битов, задаваемых литерой, ведется в младшей тетраде операнда приемника.

Введены дополнительные псевдокоманды загрузки в регистр памяти данных адресов инструкций (команд) и данных, расположенных в памяти команд.

Структура команд загрузки адреса инструкции:

mial dst, адрес_инструкции

Команда загружает младшую часть адреса команды в регистр памяти.

miah dst, адрес_инструкции

Команда загружает старшую часть адреса команды в регистр памяти.

```
Пример:  mial %a1,lab_1      ;загрузка младшей части адреса команды
         miah  %a2,lab_1     ;загрузка старшей части адреса команды
                . . .
lab_1:   mov %a1,%b2
```

Структура команд загрузки адреса данных, расположенных в памяти команд:

mdal dst, адрес_данных

Команда загружает младшую часть адреса данных в памяти команд (с точностью до байта) в регистр памяти.

mdah dst,адрес_данных

Команда загружает старшую часть адреса данных в памяти команд (с точностью до байта) в регистр памяти. Разряды **6** и **7 dst** автоматически устанавливаются в **0**.

```
Пример:  lab_1:   .byte   1,2,3,4,5     ;массив данных в памяти команд
                  .even                 ;выравнивание на границу слова
         lab_2:   mdal    %a1,lab_1     ;загрузка младшей части адреса данных
                  mdah    %a2,lab_1     ;загрузка старшей части адреса данных
```

Однооперандные команды

Однооперандные команды выполняют операции только над одним операндом – регистром в памяти данных, внешних устройств или косвенным регистром.

Структура команды:

мнемоника_команды dst

Пример: **not %a1**

Класс однооперандных команд расширен псевдокомандами, реализованными на литеральных командах:

inc – увеличение на 1 содержимого операнда (**addl %Xn,1**).

dec – уменьшение на 1 содержимого операнда (**subl %Xn,1**).

clr – очистка операнда (**movl %Xn,0**).

Следует отметить, что команды **inc** и **dec** изменяют состояние бита **C** в регистре состояния микроконтроллера.

Команды работы со служебными регистрами

Команда загрузки служебных регистров.

Формат команды: **ldr #n , значение**

где: **n** - номер служебного регистра.

Номер служебного регистра лежит в диапазоне **0** ¸ **7**, определяя соответственно один из регистров **SR0** ¸ **SR7**. Загружаемое значение может быть константой, меткой или выражением, задающим адрес в памяти данных или памяти команд.

При загрузке регистров **SR0** ¸ **SR3**, загружаемое значение является адресом в памяти данных. Константа или выражение, загружаемое в регистр, преобразуется в адрес сегмента путем отсечения младших 3-х разрядов значения.

Пример: **ar_adr=11h**

ldr #1,ar_adr

В результате выполнения команды в регистр **SR1** будет загружено значение **2**, определяющее сегментный адрес **10h**.

При загрузке регистров **SR4** или **SR5** загружаемое значение является адресом в памяти данных. Никаких преобразований загружаемого значения не выполняется.

Пример: **ldr #4,41h**

ldr #5,40h

В результате выполнения связки команд косвенный регистр **IR0** указывает на адрес **41h** в памяти данных и работает в автоинкрементном режиме.

При загрузке регистров **SR6** или **SR7** загружаемое значение может быть:

– адресом в памяти данных;

– адресом данных в памяти команд;

– адресом команды для выполнения косвенного перехода (вызова подпрограммы).

Данные, загруженные в регистры **SR6** и **SR7**, интерпретируются в зависимости от метода использования косвенного регистра **IR1**.

В автоинкрементном, автодекрементном режиме и режиме чистой косвенности загруженные данные интерпретируются как адрес в памяти данных. Этот случай не отличается от описанного выше для косвенного регистра **IR0**.

В режиме адресации к памяти команд содержимое регистров интерпретируется как адрес данных в памяти команд с точностью до байта. Адрес данных в памяти команд, конечно, может быть задан явно числовым значением, но предпочтительнее использовать псевдокоманды загрузки адреса данных.

Структура псевдокоманд загрузки адреса данных:

ldal #6, метка_данных – загрузка младшей части адреса данных (с точностью до байта);

ldah #7, метка_данных – загрузка старшей части адреса данных (с точностью до байта).

```
Пример: lab_1:   .byte 0,5,6,9,7,7,9   ;массив данных
                 .even                 ;выравнивание на границу слова
        lab_2:   ldal #6,lab_1         ;загрузка младшей части адреса данных
```

ldah #7,lab_1 ;загрузка старшей части адреса данных

При данной форме загрузки **SR7 6**-й и **7**-й разряды константы, задающей старшую часть адреса, установлены в **0**, что приводит к автоматической установке режима обращения к памяти команд.

При выполнении косвенного перехода или косвенного вызова подпрограммы данные, загруженные в регистры **SR6** и **SR7**, интерпретируются как адрес следующей выполняемой команды. Для загрузки адреса перехода рекомендуется использовать специальные псевдокоманды.

Структура псевдокоманд загрузки адреса перехода:

lia #6,метка_перехода – загрузка младшей части адреса перехода;

liah #7,метка_перехода – загрузка старшей части адреса перехода.

Пример:

```
lia     #6,lab_1  ;загрузка младшей части адреса перехода
liah    #7,lab_1  ;загрузка старшей части адреса перехода
ijmp              ;переход
        ...
lab_1:            ;метка перехода
```

Команды чтения и записи служебных регистров

mfpr dst,#n

Команда копирует содержимое служебного регистра с номером **n** в регистр памяти.

mtpr #n,src

Команда копирует содержимое регистра памяти в служебный регистр с номером **n**.

Имеется возможность использовать связку команд загрузки адреса данных или инструкций в память с командами записи служебных регистров.

```
Пример: lab_1:  .byte   5,6,7,8,9,10   ;масссив данных
        lab_2:  mdal    %a1,lab_1      ;запись в регистр памяти младшей
                                       ;части адреса массива данных
                mdah    %a2,lab_1      ;запись в регистр памяти старшей
                                       ;части адреса массива данных
                mtpr#6,%a1             ;запись в SR6 младшей части адреса
                                       ;массива
                mtpr#7,%a2             ;запись в SR7 старшей части адреса
                                        массива
```

push #n

Команда записывает содержимое служебного регистра с номером **n** в стек данных.

pop #n

Команда восстанавливает из стека данных содержимое служебного регистра с номером **n**.

Команды работы с битами регистра статуса

Команды установки и сброса битов в регистре статуса.

Формат команды: **мнемоника_команды константа**

Значение **1** в соответствующих разрядах константы определяет разряды в регистре статуса процессора, над которыми выполняется операция.

0-й разряд константы – разряд **C** в регистре статуса;

1-й разряд константы – разряд **Z** в регистре статуса;

2-й разряд константы – разряд **S(N)** в регистре статуса;

3-й разряд константы – разряд **IE** в регистре статуса.

sst – установка битов в регистре статуса;

cst – сброс битов в регистре статуса.

Данный класс команд расширен псевдокомандами явного обращения к разрядам регистра статуса, а именно:

stc – установка бита **C**;

stz – установка бита **Z**;

stn – установка бита **S(N)**;

stie – установка разрешения прерывания;

ctc – сброс бита **C**;

ctz – сброс бита **Z**;

ctn – сброс бита **S(N)**;

ctie – сброс разрешения прерывания.

Команды проверки разрядов регистра статуса

tof – проверка бита переполнения **OF**; значение разряда **OF** транслируется в разряд **Z** регистра статуса.

tdc – проверка бита тетрадного переноса **DC**; значение разряда **DC** транслируется в разряд **Z** регистра статуса.

Команды не имеют операндов.

Команды передачи управления

Команды перехода по адресу, заданному в теле команды:

jmp – безусловный переход;

jsr – переход к подпрограмме;

jnz(jne) – переход по **Z** = **0** (не равно);

jz(jeq) – переход по **Z** = **1** (равно);

jns – переход по **S(N)** = **0**;

js – переход по **S(N)** = **1**;

jnc (jae) – переход по **C** = **0**;

jc (jb) – переход по **C** = **1**.

Формат команды: **мнемоника_команды адрес_перехода**

Адрес перехода может быть задан меткой, константой, арифметическим выражением.

Пример: **jmp lab_1+1** ;переход по адресу **lab_1+1** (**lab_2**)

...

lab_1: **jmp .+1** ;переход по адресу= **PC +1** (**lab_2**)

lab_2: **mov %a1,%a2** ;оба перехода будут выполнены на эту команду

Команды передачи управления с использованием косвенного регистра

ijmp – команда выполняет безусловный переход по адресу, содержащемуся в косвенном регистре **IR1**;

ijsr – команда выполняет переход к подпрограмме по адресу, содержащемуся в косвенном регистре **IR1**.

Команды возврата из подпрограмм и прерываний

rts – возврат из подпрограммы;

rti – возврат из прерывания;

rtsc – возврат из подпрограммы с изменением состояния разряда **C** в регистре статуса.

Формат команды: **rtsc c**

c – значение разряда **C**, которое будет установлено в регистре статуса после выполнения команды. Принимает значение **0** или **1**.

Специальные команды

nop – нет операции;

wait – ожидание;

slp – останов;

sksp – прогон стека;

RST – программный **RESET**.

Команды не имеют операндов.

Инструкции транслятора и команды препроцессора

Инструкции размещения данных в памяти команд

[метка:] .byte константа [,константа].

[метка:] .word константа/метка [,константа/метка].

Данные в памяти команд могут быть организованы в виде последовательности байтов или 16-разрядных слов. При использовании словной организации данных в качестве константы может использоваться адрес в памяти команд (метка). Константа может быть представлена арифметическим выражением. Символьные константы и метки, используемые при задании констант, должны быть определены ранее.

Инструкции завершения трансляции

.end – завершение трансляции.

.endc – завершение трансляции программы и вычисление контрольной суммы.

Исходный текст программы в каждом из транслируемых файлов должен содержать инструкцию завершения трансляции. Текст, содержащийся в файле после инструкции **.end**, трансляции не подлежит. При необходимости вычисления контрольной суммы всей программы в целом в качестве последней в программе инструкции завершения трансляции должна быть использована **.endc**. Контрольная сумма содержит 3 байта и вычисляется сложением байтов кода программы с учетом переносов. Контрольная сумма записывается в тело результирующего кода программы. Допускается обращение к контрольной сумме программы как к данным в памяти команд.

Инструкция обеспечения четности указателя адреса

.even

Применяется при байтовой организации данных в памяти команд для обеспечения четности указателя адреса.

Пример: **dat1: .byte 1,2,3,4,5**

.even

Инструкции условной трансляции

.if константа

...

.else

...

.endif

.if – начало основного блока условной трансляции. Если значение константы равно **0**, то производится трансляция блока.

.else – начало блока альтернативной трансляции. Блок транслируется если значения константы отлично от **0**.

.endif – конец блока условной трансляции.

Инструкции управления отладкой

.bpt – определяет точку останова программы в процессе отладки. Допускается установка не более 4-х точек останова.

Инструкции задания макроопределений

.macro имя_макроопределения параметр1,...,параметрN

имя_макроопределения – последовательность длиной не более 32 символов. **Параметр** – формальный параметр макроопределения; последовательность длиной не более 32 символов.

.endm – инструкция завершает блок макрорасширения.

Инструкции формирования листинга

.list режим

режим – код режима определяет битовые флаги разрешения / запрещения раскрытия содержимого файлов **include**, макроопределений **define**.

Разряд **0** – разрешение раскрытия содержимого файлов **include**.

Разряд **1** – разрешение раскрытия макроопределений.

Разряд **2** – разрешение раскрытия **define**.

По умолчанию установлен режим полного разрешения.

Команды препроцессора

#include – включение содержимого файла в тело исходного текста программы.

#include "имя файла" – строка заменяется на содержимое файла с указанным именем. Если путь к файлу не задан явно, то поиск файла производится в текущем каталоге.

#include <имя файла> – поиск включаемого файла производится в каталоге, указанном во входной строке транслятора (**-i**).

Команда может употребляться в любом месте программы.

#define идентификатор строка – команда устанавливает соответствие строки символов заданному идентификатору. В теле программы идентификатор будет заменён на указанную строку символов. Размер идентификатора не должен превышать 32 символа.

Макроопределения

Задание макроопределения осуществляется с помощью соответствующих инструкций транслятора:

.macro имя_макроопределения параметр 1,..., параметр N

тело макроопределения

.endm

Размер имени макроопределения не должен превышать 32 символа. Вызов макроопределения осуществляется использованием в поле команды имени макроопределения. Не допускается использовать в качестве имени макропределения имен команд процессора. В роли фактических параметров макроинструкции могут выступать любые конструкции языка (метки, константы, выражения, определения, имена регистров). Тело макроопределения не может включать инструкции **.macro**, **#include**. Использование макроинструкций в теле макроопределений допускается.

В макроопределениях имеется режим автоматической генерации уникальных меток. Метки в макроопределениях должны начинаться с символа **@**. Метка содержит до 5 цифровых символов. Автоматически генерируются только локальные метки.

Пример:

```
.macro name par1,par2
        cmpl    par2,0
        jne     @1
        mov     par1,par2
@1:     inc     par2
        .endm
        name    %a1,%a2
```

ПРИМЕР ПРОГРАММЫ

Сообщения об ошибках включаются в листинг программы и выдаются на экран монитора с указанием номера строки-источника ошибки.

Macro.mlb

```
;-------------------------------------------------------------------------
; макроопределение формирования конфигурации тамера
; inter_h – значение старшего байта интервала
; inter_l – значение младшего байта интервала
;-------------------------------------------------------------------------
   .macro timer_config inter_h,inter_l
      movl      ta_dr,10h       ;работа с регистром интервала
      movl      ta_ci,40h       ;16-разрядный регистр
      movl      ta_dr,0         ;работа с младшим байтом регистра интервала
      movl      ta_ci,inter_l   ;значение младшего байта интервала
      movl      ta_dr,4         ;работа со старшим байтом регистра интервала
      movl      ta_ci,inter_h   ;значение старшего байта регистра интервала
   .endm
port_cf=00011011b   ;значение регистра управления порта для задания
                    ;конфигурации в автоинкрементном режиме
;-------------------------------------------------------------------------
; макроопределение формирования конфигурации портов
; p_dr-регистр управления порта
;-------------------------------------------------------------------------
    .macroport_config  p_dr
      movl      p_dr,port_cf    ;заполнение регистра управления порта для
                                ;формирования конфигурации порта в
                                ;автоинкрементном режиме
      movl      p_dr,0ffh       ;3 – все вход/выход
      movl      p_dr,0          ;4 – открытый сток
      movl      p_dr,0          ;5 – резисторы отключены
      movl      p_dr,0          ;6 – прерывания запрещены
      movl      p_dr,0          ;7 – прерывания запрещены
    .endm
;-------------------------------------------------------------------------
; макроопределение формирования цикла
; reg – счетчик цикла
; lab – метка цикла
;-------------------------------------------------------------------------
    .macro loop  reg,lab
      subl  reg,1
      jne   lab
    .endm
```

Example.mic

```
;:::::::::::::::::::::::::::::::::::::::::::::::::
;:                                              ::
;:     Тестовая программа                       ::
;:                                              ::
;:::::::::::::::::::::::::::::::::::::::::::::::::

#define    ta_dr      %d4 ;регистр управления таймера A
#define    ta_ci      %d5 ;интервальный (конфигурации) регистр
                          ;таймера A
#define    pa_dr      %b1 ;регистр управления порта A
#define    pb_dr      %b2 ;регистр управления порта B
#define    pa_wr      %d1 ;рабочий регистр порта A
#define    pb_wr      %d2 ;рабочий регистр порта D

work=40h                  ;область рабочих переменных программы

#include "macro.mlb"

;- область векторов прерываний процессора
        jmp   START       ;<0> Начальный пуск программы
        nop               ;<1> Сторожевой таймер (не используется)
        jmp   exit        ;<2> выход за границу стека
        jmp   ITMRA       ;<3> Таймер_A
        nop               ;<8> не используется
        nop               ;<5> не используется
        nop               ;<6> Порт A  (прерывания не используются)
        nop               ;<7> Порт B  (прерывания не используются)

;-  область векторов прерываний устройств, отсутствующих в данной версии
;микроконтроллера (может быть использована для размещения полезного кода
;программы.

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%:
; процедура обработки прерывания от таймера A                              :
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%:

 ITMRA:  push      #d          ;сохранение содержимого сегментного регистра
         ldr       #d,0        ;сегментный адрес регистра управления таймера
                               ; A
         bich      ta_dr,0Eh   ;снятие причины прерывания (сбросом битов
                               ;ошибки)
         pop       #d          ;восстановление содержимого сегментного
                               ; регистра
         rti
         nop                   ;<13>
```

```
        nop             ;<14>
        jmp   exit      ;<15> EEPROM (прерывание не используется)

;#########################################################:
;     Точка старта программы                              :
;#########################################################:
start: ldr              #a,work         ;сегмент рабочих переменных программы
       ldr              #b,18h          ;сегмент регистров конфигурации портов
       port_config      pa_dr           ;формирование конфигурации порта A
       port_config      pb_dr           ;формирование конфигурации порта B
       ldr              #d,0            ;
       movl             pa_wr,0FFh      ;запись в порт A
       movl             pb_wr,0FFh      ;запись в порт B
       movl             a1,20           ;счетчик цикла
$53:   movl             a0,0            ;счетчик внутреннего цикла
$54:   movl             pa_wr,055h      ;запись в порт A
       movl             pb_wr,055h      ;запись в порт B
       movl             pa_wr,0AAh      ;запись в порт A
       movl             pb_wr,0AAh      ;запись в порт B
       loop             a0,$54          ;организация внутреннего цикла
       loop             a1,$53          ;организация внешнего цикла
       timer_config     0ffh,0ffh       ;
       movl             a4,055h         ;данные для формирования шахматного
       movl             a5,0FFh         ;кода на порту B
       movl             ta_dr,3         ;пуск таймера
       movl             %a6,0           ;параметр цикла
$10:   wait                             ;ожидание прерывания  от таймера
       mov              pb_wr,%a4       ;запись в порт B
       xor              a4,a5           ;формирование шахматного кода
       loop             %a6,$10         ;цикл
exit:  slp                              ;останов программы
       .END
```