

Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

Факультет ИУ «Информатика и системы управления»

Кафедра ИУ-3 «Информационные системы и телекоммуникации»

Методические указания  
по выполнению лабораторной работы 1  
«Исследование команд обработки сигналов»  
по дисциплине «Микропроцессорные устройства обработки сигналов»

Для студентов-магистров, обучающихся по направлению 09.04.02

Продолжительность работы 4 часа

Составители:

проф. каф. ИУЗ, д.т.н.

В.С. Выхованец

преп. каф. ИУЗ

А.И. Германчук

Москва, 2020

## Содержание

|  |    |
|--|----|
| 1 Общие указания                             | 3  |
| 2 Элементы языка С                           | 3  |
| 2.1 Типы данных                              | 3  |
| 2.2 Ассемблерные вставки                     | 4  |
| 3 Директивы ассемблера                       | 4  |
| 3.1 Секционирование                          | 4  |
| 3.2 Инициализация памяти                     | 5  |
| 3.3 Перекрестные ссылки                      | 5  |
| 4 Задание на самостоятельную подготовку      | 5  |
| 5 Выполнение лабораторной работы             | 6  |
| 5.1 Создание проекта                         | 6  |
| 5.2 Сборка проекта                           | 6  |
| 5.3 Запуск отладчика                         | 6  |
| 5.4 Выполнение индивидуального задания       | 7  |
| 6 Требования к отчету                        | 8  |
| 7 Контрольные вопросы                        | 8  |
| Список литературы                            | 9  |
| Приложение А. Исходный текст модуля main.c   | 10 |
| Приложение Б. Исходный текст модуля test.asm | 11 |

## 1 Общие указания

Целью лабораторной работы является знакомство с интегрированной средой разработки программ Code Composer Studio (CCS) компании Texas Instruments. Среда CCS предназначена для разработки программного обеспечения для микропроцессоров и микроконтроллеров, выпускаемых компанией Texas Instruments, и включает такие инструменты, как редактор исходных текстов, компилятор, компоновщик, отладчик и симуляторы.

Лабораторная работа заключается в написании в соответствии с выданным индивидуальным заданием текстов несложных программ на языке C (Приложение А) и ассемблера (Приложение Б), создании нового проекта в CCS, компиляции и отладке разработанных программ с помощью симулятора микропроцессора TMS320C5515 или оценочной платы TMS320C5515 DSP Evaluation Module (TMDXEVM5515).

Для выполнения лабораторной работы на симуляторе микропроцессора TMS320C5515 необходимо загрузить и установить среду разработки Code Composer Studio версии 5.5.0 (данная версия является последней, включающей в себя симулятор процессора) с сайта компании Texas Instruments. Для этого необходимо сначала зарегистрироваться на сайте компании ([www.ti.com](http://www.ti.com)), затем перейти на страницу загрузки среды разработки. Для этого можно воспользоваться поиском по сайту, либо перейти на страницу продукта через выпадающее меню *Design Resources* на главной странице сайта.

Для подготовки к лабораторной работе 1 необходимо повторить лекционный материал по архитектуре и организации микропроцессора, изучить исходные тексты программ из приложений А-В и разработать функцию на языке ассемблера, реализующую индивидуальное задание, а также модуль на языке C, в котором вызывается разработанная функция и выводятся результаты её работы.

## 2 Элементы языка C

### 2.1 Типы данных

Встроенные типы данных языка C и их разрядность приведены на рис. 1. При написании модулей на языке C и на языке ассемблера необходимо следить за соответствием разрядности переменных (ячеек памяти) и разрядности регистров процессора

| Type                        | Size            | Representation | Range                        |                   |
|-----------------------------|-----------------|----------------|------------------------------|-------------------|
|                             |                 |                | Minimum                      | Maximum           |
| char, signed char           | 16 bits         | ASCII          | -32 768                      | 32 767            |
| unsigned char               | 16 bits         | ASCII          | 0                            | 65 535            |
| short, signed short         | 16 bits         | 2s complement  | -32 768                      | 32 767            |
| unsigned short              | 16 bits         | Binary         | 0                            | 65 535            |
| int, signed int             | 16 bits         | 2s complement  | -32 768                      | 32 767            |
| unsigned int                | 16 bits         | Binary         | 0                            | 65 535            |
| long, signed long           | 32 bits         | 2s complement  | -2 147 483 648               | 2 147 483 647     |
| unsigned long               | 32 bits         | Binary         | 0                            | 4 294 967 295     |
| long long, signed long long | 40 bits         | 2s complement  | -549 755 813 888             | 549 755 813 887   |
| unsigned long long          | 40 bits         | Binary         | 0                            | 1 099 511 627 775 |
| enum (C)                    | 16 bits         | 2s complement  | -32 768                      | 32 767            |
| enum <sup>(1)</sup> (C++)   | 16, 32, 40 bits | 2s complement  | -549 755 813 888             | 549 755 813 887   |
| float                       | 32 bits         | IEEE 32-bit    | 1.175 494e-38 <sup>(2)</sup> | 3.40 282 346e+38  |
| double                      | 32 bits         | IEEE 32-bit    | 1.175 494e-38 <sup>(2)</sup> | 3.40 282 346e+38  |
| long double                 | 32 bits         | IEEE 32-bit    | 1.175 494e-38 <sup>(2)</sup> | 3.40 282 346e+38  |
| pointers (data):            |                 |                | 0                            |                   |
| small memory mode           | 16 bits         | Binary         | 0                            | 0xFFFF            |
| large memory mode           | 23 bits         | Binary         | 0                            | 0x7FFFFFF         |
| pointers (function)         | 24 bits         | Binary         | 0                            | 0xFFFFFFFF        |

<sup>(1)</sup> The representation is the same as the underlying type (int, long or long long)

<sup>(2)</sup> Figures are minimum precision.

Рисунок 1 – Типы данных языка C

## 2.2 Ассемблерные вставки

Для вставки в исходный текст на языке C команд микропроцессора используется оператор ассемблирования:

```
asm("текст на ассемблере");
```

## 3 Директивы ассемблера

### 3.1 Секционирование

Для задания секций с неинициализированными данными используются директивы `bss` (неименованная секция) и `usect` (именованная секция):

```
.bss    symbol, size[, blocking[, alignment]]
.usect  "name", size[, blocking[, alignment]]
```

где `symbol` – идентификатор резервируемой области памяти, `size` – размер резервируемой области памяти в словах, `blocking` – необязательный флаг выделения памяти в пределах одной страницы памяти, `alignment` – необязательный флаг выравнивания по границе слова, `name` – имя секции. Элементы синтаксиса директив, заключенные в квадратные скобки, являются необязательными и могут быть опущены.

### 3.2 Инициализация памяти

Для задания секций с инициализированными данными используются директивы `text` (секция кода), `data` (неименованная секция данных) и `sect` (именованная секция данных):

```
.text
.data
.sect    "name"
```

где `name` – имя секции.

Для инициализации данных используются следующие директивы:

```
.[u]byte      value[, value[, value ...]]
.[u]char      value[, value[, value ...]]
.cstring      "string"[, "string"[, "string" ...]]
.double       value[, value[, value ...]]
.field        value[, size]
.float        value[, value[, value ...]]
.[u]half      value[, value[, value ...]]
.[u]int       value[, value[, value ...]]
label: .ivec   address[, stack mode]
.ldouble      value[, value[, value ...]]
.[u]long      value[, value[, value ...]]
.pstring      "string"[, "string"[, "string" ...]]
.[u]short     value[, value[, value ...]]
.string       "string"[, "string"[, "string" ...]]
.[u]word      value[, value[, value ...]]
.xfloat       value[, value[, value ...]]
.xlong        value[, value[, value ...]]
```

где `u` – префикс беззнакового формата чисел;

### 3.3 Перекрестные ссылки

Для объявления перекрестных ссылок используются директивы:

```
.def symbol[, symbol ...]]
.ref symbol[, symbol ...]]
```

Директива `.def` используется для объявления одного или нескольких имен текущего модуля, которые должны быть доступны в других модулях программы. Директива `.ref` используется для объявления одного или нескольких имен, используемых в текущем модуле, но определенных в других модулях.

## 4 Задание на самостоятельную подготовку

- 1) Повторите материал лекций по архитектуре и организации микропроцессора TMS320C5515.
- 2) Изучите исходные тексты программ из Приложений А, Б.
- 3) Разработайте модули на языке C и на языке ассемблера, реализующие индивидуальное задание.

## 5 Выполнение лабораторной работы

### 5.1 Создание проекта

5.1.1 Для создания проекта выберите пункты меню *File* → *New* → *Project*. Откроется форма выбора типа проекта.

5.1.2 В списке типов проекта необходимо выбрать пункт *Code Composer Studio* и подпункт *CCS Project*. Откроется форма нового проекта.

5.1.3 В поле *Project Name* введите имя проекта, например, *MPUOS\_LAB1*. При установке флага *Use default location* проект будет создан в папке текущего пользователя. В противном случае место размещения папки проекта в файловой системе следует выбрать после нажатия на кнопку *Browse*.

5.1.4 В ниспадающем списке *Output Type* выберите *Executable* для создания исполняемой программы.

5.1.5 В разделе *Device* в ниспадающем списке выберите семейство микропроцессоров *C5500*, вариант *C551x* и *EVM5515*, соединение *Texas Instruments Simulator*.

5.1.6 В разделе *Project Template and Examples* выберите *Empty Project (with main.c)*. Нажмите кнопку *Finish*. CCSv5 создаст проект, который отобразится в окне *Project Explorer*.

5.1.7 Для включения в проект новых файлов в контекстном меню, открывающемся при нажатии правой кнопки мыши на имени созданного проекта, выберите пункт *New* и создайте файл *lab1.asm*. Для добавления в проект существующих файлов выберите пункт меню *Project* → *Add Files* и в появившемся окне *New Source File* задайте место размещения файла *lab1.asm* при установленном флаге *Copy File* для копирования этих файлов в папку проекта.

5.1.8 Введите подготовленные исходные тексты программ в окна текстового редактора *main.c* и *lab1.asm*.

### 5.2 Сборка проекта

5.2.1 Выполните сборку проекта, выбрав пункт меню *Project* → *Build Project*. Исправьте обнаруженные ошибки компиляции (асемблирования) и повторите сборку снова.

5.2.2 В случае необходимости измените параметры сборки, выбрав в контекстном меню проекта (окно *Project Explorer*) пункт *Build Options*. Аналогичные настройки через контекстное меню, открывающееся при нажатии правой кнопки мыши на имени файла, можно выполнить и для каждого файла проекта.

### 5.3 Запуск отладчика

5.3.1 Установите фокус ввода на имени проекта в окне *Project Explorer* и выберите пункт меню *Run → Debug*. По этой команде запустится отладчик и откомпилированный ранее загрузочный файл с именем проекта и расширением *.out* загрузится для исполнения в память целевой платформы.

5.3.2 Для выполнения загруженной программы по шагам используйте пункты меню *Target → Step Into (F5)* или *Target → Step Over (F6)*, а для запуска программы – *Target → Run (F8)*.

5.3.3 При пошаговой отладке программы содержимое регистров процессора контролируйте в окне *Registers*, содержимое ячеек памяти – в окне *Memory*, текст программы на языке C с подсвеченным текущим оператором – в окне редактора соответствующего исходного файла, а исполняемый код программы на языке ассемблера – в окне *Disassembly*.

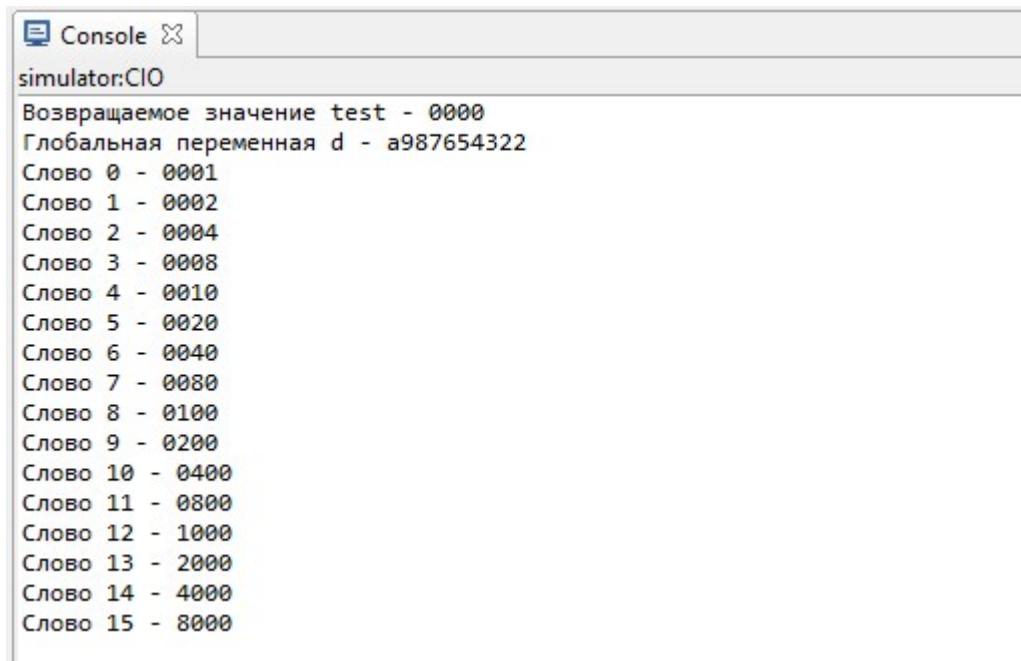
5.3.4 Для ускорения прохода отладчика до требуемого оператора (команды) используйте точки останова. Для задания точки останова установите фокус ввода на требуемый оператор (команду) и выберите в контекстном меню пункт *New Breakpoint* или *Toggle Breakpoint*. После установки точек останова используйте команду запуска программы *Target → Run (F8)*, которая приведет к выполнению программы до тех пор, пока не наступит очередь выполнения команды, помеченной точкой останова.

## 5.4 Выполнение индивидуального задания

5.4.1 Используя отладчик, протрассируйте выполнение разработанной программы. Убедитесь в корректности работы разработанной программы и ее соответствии заданию.

5.4.2 В случае необходимости остановите отладку, внесите изменение в исходные тексты программ и повторите процесс отладки. При внесении изменений воспользуйтесь справочными данными, приведенными в литературе [1-4].

5.4.3 Сохраните результаты работы программы (вывод в консоль среды разработки при помощи функции `printf`). Пример сохраненного результата приведен на рис. 2.



```
simulator:CIO
Возвращаемое значение test - 0000
Глобальная переменная d - a987654322
Слово 0 - 0001
Слово 1 - 0002
Слово 2 - 0004
Слово 3 - 0008
Слово 4 - 0010
Слово 5 - 0020
Слово 6 - 0040
Слово 7 - 0080
Слово 8 - 0100
Слово 9 - 0200
Слово 10 - 0400
Слово 11 - 0800
Слово 12 - 1000
Слово 13 - 2000
Слово 14 - 4000
Слово 15 - 8000
```

Рисунок 2 – Пример сохраненного результата работы программ из приложений А и Б

## 6 Требования к отчету

Отчет по лабораторной работе должен содержать:

- титульный лист;
- содержание;
- описание задания с текстами программ;
- результаты выполнения задания (результат работы программы);
- содержательные выводы и рекомендации;
- список использованной литературы;
- приложения (при необходимости).

Оформление отчета осуществляется в соответствии с ГОСТ 2.105-95 [5], список литературы по ГОСТ Р 7.0.5–2008 [6].

## 7 Контрольные вопросы

Вопрос 1 – Какие регистры ядра сигнального процессора используются для хранения аргументов функции при ее вызове? Для хранения каких типов данных используются регистры каждой группы?

Вопрос 2 – Какие специальные функции и команды сигнального процессора использовались или могли быть использованы для реализации задания?

Вопрос 3 – Предложите рекомендации по оптимизации разработанной функции.



## Список литературы

- [1] TMS320C55x. CPU. Reference Guide. – Texas Instruments, 2009. – 265 p.
- [2] TMS320C55x. Mnemonic Instruction Set. Reference Guide. – Texas Instruments, 2009. – 863 p.
- [3] TMS320C55x. Assembly Language Tools. User's Guide. – Texas Instruments, 2011. – 366 p.
- [4] TMS320C55x. Optimizing C/C++ Compiler. User's Guide. – Texas Instruments, 2011. – 181 p.
- [5] ГОСТ 2.105-95. Единая система конструкторской документации. Общие требования к текстовым документам. – М.: Изд-во стандартов, 2012. – 26 с.
- [6] ГОСТ Р 7.0.5–2008. Библиографическая ссылка. Общие требования и правила составления. – М.: Изд-во стандартов, 2009. – 23 с.

## Приложение А

### Исходный текст модуля main.c

```
/* Включаемые файлы */
#include "stdio.h"

/* Макроопределения */
#define RESULTS 16

/* Глобальные данные */
long long d = 0xFEDCBE9876;
int r[RESULTS];

/* Объявление внешней функции */
extern long test(int*, int);

/* Точка входа в программу */
void main(void)
{
    /* Локальные переменные */
    int i, q;
    long t = 0x12345678;
    /* Подготовка исходных данных в формате Q15 */
    # pragma MUST_ITERATE(RESULTS, RESULTS, 1)
    for( i = 0, q = 32767; i < RESULTS; i++ )
    {
        r[i] = q, q >>= 1; /* 0,5, 0,25, 0,125, ... */
    }
    /* Вызов внешней функции */
    t = test(r, RESULTS);
    /* Вывод результата */
    printf( "Возвращаемое значение test - %08x\n", t );
    printf( "Глобальная переменная d - %010x\n", d );
    # pragma MUST_ITERATE(RESULTS, RESULTS, 1)
    for( i = 0; i < RESULTS; i++ )
    {
        printf( "Слово %d - %04x\n", i, r[i] );
    }
}
```

## Приложение Б

### Исходный текст модуля test.asm

```
; Объявление перекрестных ссылок
.def      _test
.ref      _d
; Секция инициализированных данных
.data
w  .word  0FFFFh, 0FFA9h, 8765h, 4320h
; Секция кода
.text
; Аргументы функции test:
;  AR0          - указатель int* на массив
;  T0           - длина массива int
; Состояние стека:
;  *SP(#0)      - локальная переменная функции
;  *SP(#1)      - сохраненный регистр T2
;  *SP(#2)      - сохраненный регистр AR5
;  *SP(#3)      - адрес возврата из функции
;  *SP(#4)      - первый аргумент, не поместившийся в регистрах
; Определение локальной переменной
.asg      *SP(#0), var
; Точка входа в функцию test
_test:
; Использование указателя стека при прямой адресации
.cpl_on
; Сохранение в стеке модифицируемых регистров
PSH      mmap(ST1_55)
PSH      mmap(ST2_55)
PSH      T2, AR5
; Выделение в стеке памяти для локальной переменной
AADD     #-1, SP
; Задание режима работы микропроцессора
BSET     M40                      ; ST1 40-битные операции АЛУ
BSET     SXMD                     ; ST1 расширение знака при
загрузке
BCLR     ARMS                     ; ST2 сигнальный режим адресации
.arms_off
; Сохранить длину массива в локальной переменной
MOV      T0, var
; Занести длину массива в счетчик простого повторения
ADD      #-1, T0
MOV      T0, CSR
; Скопировать указатель на массив в AR5
MOV      AR0, AR5
; Инициализировать T1 числом 2^(-15)
MOV      #1, T1
; Выполнить CSR+1 раз запись и модификацию данных
RPT      CSR
MOV      T1, *AR5+
||SFTL   T1, #1
; Загрузить в AC0 данные из области памяти w
MOV      #w, AR0
MOV      dbl(*AR0+), AC1
MOV      dbl(*AR0+), AC0
MOV      AC1, mmap(AC0G)
; Прибавить к AC0 2
ADD      #2, AC0
```

```

; Сохранить AC0 в глобальной переменной типа long long
MOV          #_d, AR1
SFTS         AC0, #-32, AC1
MOV          AC1, dbl(*AR1+)
MOV          AC0, dbl(*AR1+)
; Освобождение локальной памяти
AADD         #1, SP
; Восстановление регистров из стека
POP          T3, AR5
POP          mmap(ST2_55)
POP          mmap(ST1_55)
; Возврат из подпрограммы, AC0 - возвращаемое значение long
MOV          #0, AC0
MOV          T1, HI(AC0)
RET

```