Министерство образования и науки
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра конструирования узлов и деталей радиоэлектронной аппаратуры
(КУДР)

А.А. Бомбизов, Е.И. Тренкаль

АССЕМБЛЕР. ВЕТВЛЕНИЯ И ЦИКЛЫ

Методические указания к выполнению
лабораторной и самостоятельной работы
по дисциплине «Микропроцессорные устройства»

Томск 2017

1 Введение

В предыдущей работе были рассмотрены базовые основы последовательного (линейного) выполнения инструкций и способ взаимодействия с устройством ввода-вывода микроконтроллера.

Целью настоящей работы является освоение механизмов выполнения условных и безусловных переходов в программе, а также считывания информации с порта ввода-вывода.

2 Краткая теория

Команда перехода – команда процессора, которая нарушает непрерывную последовательность исполнения команд, вынуждая выбирать и исполнять последующие команды с произвольно заданного адреса. Используется для организации условных операторов, циклов, для связи с подпрограммами.

В процессорном наборе команд Thumb, да и вообще в идеологии программирования существует два вида команд перехода: условный и безусловный.

Набор инструкций процессоров с ядром cortex-m4 [1] содержит четыре команды безусловного перехода, описанных в таблице 1.

Таблица 1

THUMB ассемблер	Действия
B метка	Перейти по адресу PC + (Смещение << 1), где адрес метки может быть смещен относительно текущего значения PC в диапазоне +2048 байт.
BL метка	Переход по метке с сохранением точки вызова в регистр LR
BX Rx	Переход по адресу, сохраненному в регистре общего назначения
BLX Rx	Переход по адресу, сохраненному в регистре общего назначения. При этом происходит сохранение адреса точки вызова.

Инструкция *B* в большинстве случаев имеет одно применение – это организация бесконечного цикла.

Использование данной инструкции может быть следующим образом:

```
ANY_LABEL_NAME:
    @ ИНСТРУКЦИЯ 1
    @ ИНСТРУКЦИЯ 2
    ….
    @ ИНСТРУКЦИЯ N
    B ANY_LABEL_NAME
```

где *ANY_LABEL_NAME* – имя метки;

ИНСТРУКЦИЯ 1 – ИНСТРУКЦИЯ N – инструкции, выполняемые внутри бесконечного цикла;

B ANY_LABEL_NAME – вызов команды (инструкции) перехода на метку *ANY_LABEL_NAME.*

Таким образом, все инструкции, которые располагаются между первой и последней строчкой, будут выполняться бесконечное (пока питание не отключат) число раз.

Команда перехода *BX Rx* используется аналогичным образом, только адрес перехода содержится в регистре общего назначения *Rx.*

Другой особо важной сферой применения безусловных переходов – это создание подпрограмм.

Подпрограмма – это отдельная функционально независимая часть программы.

Подпрограммы решают три важные задачи:

- избавляют от необходимости многократно повторять в тексте программы аналогичные фрагменты;
- улучшают структуру программы, облегчая ее понимание;
- повышают устойчивость к ошибкам программирования и непредвиденным последствиям при модификациях программы.

Очень важно понимать, что в подпрограмму может выделяться любой законченный фрагмент программы. В качестве ориентиров могут быть использованы следующие рекомендации:

1) когда в разных частях программы повторяется одна и та же последовательность команд, то она может быть вынесена в подпрограмму;

2) когда длинную программу полезно разбить на составляющие (как книгу разбивают на главу), тогда основная программа будет похожа на оглавление и хорошо подчеркивать основной алгоритм;

3) когда необходимо организовать переносимость алгоритмов в другие программы.

Для создания подпрограмм на языке ассемблер для микроконтроллеров на ядре cortex был введен специальный регистр LR (link register), предназначенный для хранения адреса инструкции, из которого была вызвана подпрограмма. Это было сделано для того, чтобы по завершению подпрограммы можно было вернуться обратно в основную программу в точку вызова и продолжить её выполнение.

Для автоматического помещения адреса в точку вызова в момент выполнения инструкции перехода *B* была введена модификация этой инструкции – *BL метка*. Для BX введена BLX.

Таким образом, организация подпрограммы и ее вызов подпрограммы может быть осуществлен следующим образом:

Start:

ИНСТРУКЦИЯ 1

ИНСТРУКЦИЯ 2

……..

BL SUB_PROGRAM

……..

ИНСТРУКЦИЯ N

B Start

SUB_PROGRAM:

ИНСТРУКЦИЯ ПОДПРОГРАММЫ 1

……..

ИНСТРУКЦИЯ ПОДПРОГРАММЫ N

BX LR

где *BL SUB_PROGRAM* – инструкция безусловного перехода на метку подпрограммы с сохранением адреса точки вызова; *BX LR* – переход на инструкцию по сохраненному адресу, то есть в точку вызова подпрограммы.

Архитектура арифметико-логического устройства ядра cortex позволяет выполнять операции с данными, которые находятся в регистрах общего назначения. То есть это своего рода переменные, с которыми позволено работать. При этом этих регистров в системе команд thumb только восемь. Отсюда обеспечение сохранности данных при разработке сложных алгоритмов, а тем более с использованием функций является важной задачей. Для устранения дефицита регистров общего назначения, особенно при вызове функций, был придуман стек (абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in – first out, «последним пришёл — первым вышел»).). Принцип работы со стеком изображен на рисунке 2, где push – это операция помещения данных в стек, а pop – операция извлечения из стека. В процессоре с ядром cortex стек расположен в некоторой области оперативной памяти, причем вершина стека размещена по большему адресу, а все помещающиеся в стек элементы располагаются по уменьшающимся адресам. В процессоре для работы со стеком введен регистр общего назначения SP (Stack Pointer), который содержит адрес на текущую занятую ячейку памяти. Если в стеке нет элементов, то SP будет указывать на вершину стека.

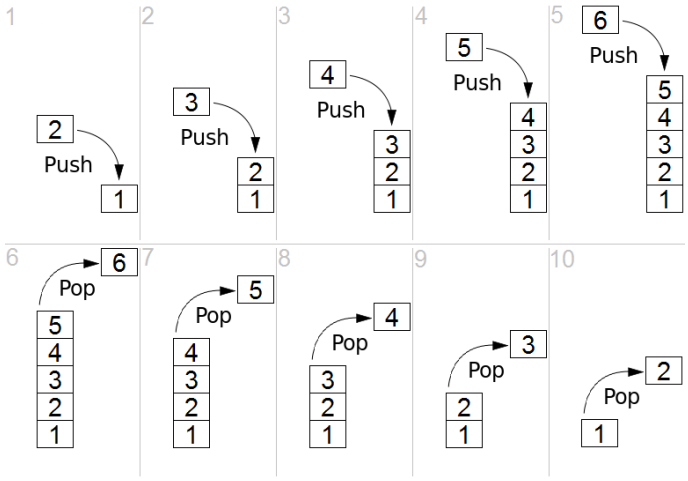

Рисунок 1 – Организация работы со стеком

В системе команд микропроцессоров на ядре cortex существуют соответствующие инструкции для работы со стеком [1]:

1) помещение регистров стек:

PUSH reglist ,

где reglist – список сохраняемых регистров, указывается в фигурных скобках и может выглядеть, например, следующим образом {R0-R3,R7}. То есть можно указывать диапазоны.

2) загрузка значений из стека:

POP reglist

Здесь использование аналогично предыдущей команде. Не стоит забывать, что чем больше регистров сохраняется в стек, тем больше тактов процессора на это тратится.

Опираясь на знания работы со стеком, разработчик должен организовывать подпрограммы по принципу «не навреди». То есть, подпрограмма перед использованием регистров общего назначения должна сохранить их значения в стек, а по окончанию работы вернуть значения из стека обратно в регистры общего назначения. Допустим, подпрограмма в своих вычислениях использует регистры общего назначения *R0*, *R1*. Заготовка подпрограммы будет выглядеть следующим образом:

```
SUB_PROGRAM:
        PUSH {R0, R1}
        ИНСТРУКЦИЯ ПОДПРОГРАММЫ 1
……..……..……..……..……..……..
        ИНСТРУКЦИЯ ПОДПРОГРАММЫ N
        POP {R0, R1}
        BX LR
```

В данном фрагменте вначале работы подпрограммы значения из регистров *R0* и *R1* помещаются в стек, а по окончанию работы перед возвратом в точку вызова возвращаются обратно из стека в регистры.

Помимо безусловных переходов, с помощью которых можно организовать бесконечные циклы и подпрограммы в архитектуре ядра cortex существуют условные переходы. **Условный переход** – команда программируемому вычислительному устройству на изменение порядка выполнения программы в соответствии с результатом проверки некоторого условия.

Идеология условных переходов в процессорах ARM основана на регистре PSR. **Program Status Register** (PSR) – содержит флаги, описывающие текущее состояние процессора. Модифицируется при выполнении логических, арифметических и др. инструкций.

Сам регистр содержит следующие биты состояния [1]:

Рисунок 2 – Program Status Register

Для выполнения условных переходов необходимо учитывать только биты 27–31, расшифровка которых приведена в таблице 2.

Таблица 2

Наименование бита	Описание
N – флаг отрицания	0 – если результат последней операции положительный; 1 – если результат последней операции отрицательный (<0). Это происходит при установке старшего бита в результирующем регистре в 1.
Z – флаг нуля	0 – если результат последней операции не равен нулю; 1 – если результат последней операции равен нулю (==).
C – флаг переноса	1 – если в результате операции произошел выход за границы слова. То есть 0xFFFFFFFF+1=0x00000000, при этом флаг C будет установлен, то есть «один в уме».
V – флаг переполнения	0 – если в результате последней операции не возникло переполнения регистра; 1 – если в результате последней операции возникло переполнения регистра.

Таблица 2 – продолжение

Наименование бита	Описание
Q – флаг переполнения для DSP инструкций	0 – если в результате последней операции не возникло переполнения регистра; 1 – если в результате последней операции возникло переполнения регистра;

Наиболее часто условный переход имеет две стадии: на первой происходит сравнение между собой некоторых величин, определяющих условие перехода, на второй выполняется сам переход.

Для сравнения двух величин применяются следующие инструкции:

- *TST Rd, Rs*

Устанавливает флаг Z состояния по результату выполнения побитовой операции «И» между регистрами Rd и Rs (Rd AND Rs). Применяется для определения состояния битов в регистре Rd, определенных маской в регистре Rs. К примеру, *TST Rd, 1<<7* проверяет состояние седьмого бита.

- *CMP Rd, Rs*

Устанавливает флаги состояния по результату выполнения *Rd - Rs.*

- *CMN Rd, Rs*

Устанавливает флаги состояния по результату выполнения *Rd + Rs.*

Соответственно для побитового сравнения применяется *TST*, а для арифметического наиболее часто *CMP*.

В ассемблере команда условного перехода формируется путем соединения команды безусловного перехода *B* со специальным суффиксом, то есть *Bxx*, где *xx* – необходимый суффикс. Всего в ядре arm 14 [2] инструкций условного перехода (таблица 3).

Таблица 3

Инструкция	Описание (при выполнении инструкции CMP)
BEQ метка	Если флаг Z установлен (операнд1 == операнд2). Для TST флаг будет установлен, если все биты не совпали.
BNE метка	Если флаг Z сброшен (операнд1 != операнд2). Для TST флаг будет сброшен, если хотя бы один бит совпал.
BCS метка	Если флаг C установлен (беззнаковый операнд1>= беззнаковый операнд2)
BCC метка	Если флаг C сброшен (беззнаковый операнд1< беззнаковый операнд2)
BMI метка	Если флаг N установлен (результат отрицательный)
BPL метка	Если флаг N сброшен (результат >=0)
BVS метка	Если флаг V установлен (переполнение)
BVC метка	Если флаг V сброшен (отсутствует переполнение)
BHI метка	Если C установлен, а Z сброшен (беззнаковый операнд1>беззнаковый операнд2)

Таблица 3 – продолжение

Инструкция	Описание
BLS метка	Если C сброшен, а Z установлен (беззнаковый операнд1<=беззнаковый операнд2)
BGE метка	Если N установлен, а V сброшен (операнд1>=операнд2)
BLT метка	Если N сброшен, а V установлен (операнд1< операнд2)
BGT метка	Если Z сброшен, либо N установлен и V установлен, а N сброшен, V установлен (операнд1> операнд2)
BLE метка	Если Z установлен, либо N установлен и V сброшен, а N сброшен, V установлен (операнд1<= операнд2)

3 Порядок выполнения работы

В ходе данной работы будет создана программа моргания одного светодиода с задержкой. Причем переключение моргающего светодиода будет происходить по нажатию кнопки. Задача будет решаться в четыре этапа: 1) инициализация портов ввода/вывода; 2) создание основного рабочего цикла; 3) разработка функции задержки; 4) разработка функции опроса состояния кнопки.

Инициализация портов ввода/вывода.

3.1 Изучите предложенный в п. 2 теоретический материал.

3.2 Для начала работы скопируйте папку template_Ветвления_и_циклы.base в свой рабочий каталог.

3.3 Откройте в блокноте (или в программе Notepad++) файл main.s, содержащийся в каталоге template_ Ветвления_и_циклы.base. Как видно, это базовый проект, созданный на первом занятии.

3.4 Определите по [3] или по внешнему виду отладочной платы порт ввода/вывода и номера бит пользовательских светодиодов и кнопки.

Определение констант порта ввода вывода:

3.5 В файле main.s после определения секции .text опишите следующие константы: 1) GPIO_LED и GPIO_BTN, содержащие базовые адреса для портов ввода/вывода светодиодов и кнопки; 2) константы с номерами бит для красного (LED_RED) и зеленого (LED_GREEN) светодиодов, соответственно, и BTN_PIN с номером бита для кнопки; 3) GPIO_LED_DIR для переключения режима (на вход, на выход…) для светодиодов и GPIO_BTN_DIR – для кнопки; 4) GPIO_LED_OUT – регистр выходных данных для порта со светодиодами и GPIO_BTN_IN – регистр входных данных для порта с кнопкой.

3.6 Определите название шины для тактирования портов ввода/вывода светодиодов и кнопки.

3.7 Опишите константу RCC_BUS с базовым адресом источника тактирования.

3.8 Опишите константу RCC_BUS_EN с адресом регистра для включения тактирования порта ввода/вывода светодиодов и кнопки.

3.9 Опишите константу RCC_LED_EN с номером бита для включения тактирования порта ввода/вывода светодиодов и RCC_BTN_EN для кнопки.

3.10 В тексте программы (после метки Start:) включите тактирование портов ввода-вывода для светодиодов и кнопки.

3.11 В тексте программы переключите выводы микроконтроллера, к которым подключены светодиоды на выход, а к которому подключена кнопка – на вход.

3.12 Включите один красный светодиод.

3.13 Откомпилируйте программу путём запуска командного файла make_project.bat и выполните программирование микроконтроллера.

Создание основного рабочего цикла:

3.14 Занесите в регистры R6 и R7 следующие маски

LDR R6, =((1<<(LED_GREEN))|(1<<(LED_RED)))

LDR R7, =(1<<(LED_GREEN))

Они понадобятся далее для выполнения операции переключения при помощи инструкции EOR (исключающее ИЛИ).

3.15 Опишите в тексте программы метку MAIN_CYCLE и организуйте бесконечный цикл.

3.16 Внутри главного цикла выполните переключение бит в регистре R1 по маске R7. В регистре R1 должно содержаться значение регистра GPIO_LED_OUT, которое было получено в п. 3.11, а в регистре R0 адрес GPIO_LED_OUT.

3.17 После переключения битов сохраните значение из регистра R1 в ячейку по адресу R0.

3.18 Откомпилируйте программу путём запуска командного файла make_project.bat и выполните программирование микроконтроллера. В результате должны гореть 2 светодиода, но зеленый с меньшей яркостью.

Разработка функции задержки.

3.19 За пределами основного цикла организуйте пока еще пустую функцию sub_DELAY. А её вызов разместите внутри основного цикла.

3.20 Так как функция задержки внутри себя будет использовать регистры R0 и R1, то их значения необходимо сохранить, чтобы не нарушить логику вызывающей программы. Поэтому после метки sub_DELAY поместите в стек значения регистров R0 и R1. А перед возвратом в точку вызова выгрузите их обратно.

Далее будет реализован алгоритм аналогичный циклу типа for(R0=0;R0<R1;R0++), где R1=2000000. С его помощью будет «прокручено вхолостую» определенное число тактов, тем самым будет организована задержка.

3.21 После загрузки значений в стек обнулите R0, а в R1 поместите константу 2000000. Последняя будет определять задержку. Чем больше значение, тем больше задержка.

3.22 Далее опишите метку DELAY_CYCLE.

3.23 Инкрементируйте значение в R0.

3.24 Для проверки достигло ли значение в R0 целевого значения, определенного в регистре R1, выполните операцию арифметического сравнения R0 и R1.

3.25 Если R0 не достиг своего целевого значения, то выполните команду условного перехода на метку DELAY_CYCLE. На этом цикл будет закончен. И в случае достижения целевого значения будет закончена функция задержки.

3.26 Откомпилируйте программу путём запуска командного файла make_project.bat и выполните программирование микроконтроллера. В результате красный светодиод должен гореть постоянно, а зеленый моргать с установленной задержкой.

Разработка функции опроса состояния кнопки.

3.27 За функцией sub_DELAY опишите функцию sub_TESTBTN с сохранением в стек регистров R0 и R1.

3.28 Внутри основного цикла сделайте вызов функции sub_TESTBTN.

3.29 Далее будет организована проверка состояния бита регистра данных, отвечающего за кнопку. Алгоритм будет аналогичен условию выполненному на языке высокого уровня

*if(*GPIO_BTN_IN*&(1<<BTN_PIN))*

R7^=R6;

Если в регистре GPIO_BTN_IN бит под номером *BTN_PIN* равен 1, то выполнить переключение бит в регистре R7 по маске R6. А так как регистр R7 содержит 0010000000000000, а R6 = 0110000000000000, то после выполнения операции исключающего или в R7 будет 0100000000000000. При повторном выполнении значение вернется в исходное состояние 0010000000000000.

3.30 В функции опроса кнопки после загрузки значений в стек загрузите в R0 адрес GPIO_BTN_IN.

3.31 Далее в R1 загрузите значение, размещенное по адресу GPIO_BTN_IN.

3.32 Выполните проверку бита BTN_PIN в регистре R1.

3.33 Определите метку END_SUB перед выгрузкой значений из стека в регистры R0, R1. Она потребуется, если бит кнопки не будет установлен и никаких действий с регистром R7 не потребуется выполнять.

3.34 После проверки бита выполните переход на метку END_SUB, а если бит не установлен, то будет выполнена следующая команда после операции перехода.

3.35 После операции перехода выполните операцию исключающего ИЛИ регистра R7 по маске R6.

3.36 Откомпилируйте программу путём запуска командного файла make_project.bat и выполните программирование микроконтроллера. В результате красный светодиод должен гореть постоянно, а зеленый моргать с установленной задержкой. По нажатию на кнопку начнет моргать красный, а зеленый перестанет.

3.37 Оформите отчет, содержащий титульный лист, введение, ход выполнения работы, ответы на контрольные вопросы и выводы.

3.38 Защитите отчет у преподавателя.

4 Контрольные вопросы

4.1 Что такое безусловный переход?

4.2 Опишите пустую функцию и её вызов.

4.3 Какие флаги состояния существуют в процессоре?

4.4 Для чего нужен стек и как им пользоваться?

4.5 Что такое условный переход?

4.6 Каким образом организуется команда условного перехода?

4.7 Какой регистр отвечает за входные данные порта ввода/вывода?

4.8 Как переключить биты в регистре общего назначения?

Список литературы

1. PM0214. Programming manual. STM32F3, STM32F4 and STM32L4 Series. Cortex®-M4 programming manual.– URL: www.st.com/resource/en/programming_manual/DM00046982.pdf (дата обращения: 10.01.2017).

2. THUMB Instruction Set. – URL:https://ece.uwaterloo.ca/~ece222/ARM/ARM7-TDMI-manual-pt3.pdf.

3. MB1075. STM32F429I-DISCO schematics.– URL: http://www.st.com/resource/en/schematic_pack/stm32f429i-disco_sch.zip.

4. RM0090. Reference manual. STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM®-based 32-bit MCUs.–

URL: www.st.com/resource/en/reference_manual/DM00031020.pdf (дата обращения: 10.01.2017).