**Міністерство освіти і науки України Національний технічний університет України «Київський Політехнічний Інститут імені Ігоря Сікорського» Кафедра конструювання електронно-обчислювальної апаратури**

Звіт з виконання лабораторної роботи №2

з дисципліни “ Обчислювальні та МП засоби в РЕА-2”

Виконав: студент групи ДК-82

Рудюк Б. Б.

Перевірила: ст.вик.

Бондаренко Н. О.

**Київ – 2021**

**Мета роботи:** ознайомлення зі структурою і технологією створення проекту, в якому основна програма, написана на С, викликає функції (підпрограми), написані на asm; написання і компілювання коду на С і asm, і налагоджування образу програми на отладочній платі (а саме, на платі «Nucleo» або «Discovery») за допомогою інструменту Keil μVision5 MDK-ARM.

1. **Плюси і мінуси програмування на С.**

Більшість програм для мікроконтролерів пишуться на мові С, оскільки це простіше і швидше (а значить і дешевше), ніж на мові асемблера. Але іноді буває так, що деякі критичні до часу виконання частини програм необхідно писати на асемблері. Використовуючи асемблер, ви можете по максимуму оптимізувати код програми так, як вважаєте за потрібне. Але все ж більшість з систем закодовано на мові C з мовою асемблера, який використовується тільки для критично важливої обробки. Також мова С (при використанні таких бібліотек, як CMSIS), забезпечує переносимість програм на різні мікроконтролери.

З іншого боку, при написанні програми на мові С може виникнути необхідність у використанні асемблера для реалізації функцій, які не можуть бути написані на С, наприклад:

* функцій, що безпосередньо маніпулюють вмістом стека або використовують спеціальні команди, які не можуть бути отримані за допомогою стандартних конструкцій мови;
* для написання процедур, критичних до часу виконання;
* при обмежених обсягах пам'яті, що вимагає написання частини програми на асемблері для отримання якомога більш компактного коду.

1. **Техніки змішування мови ASM і С.**

Можливі три техніки, змішування коду С і асемблера:

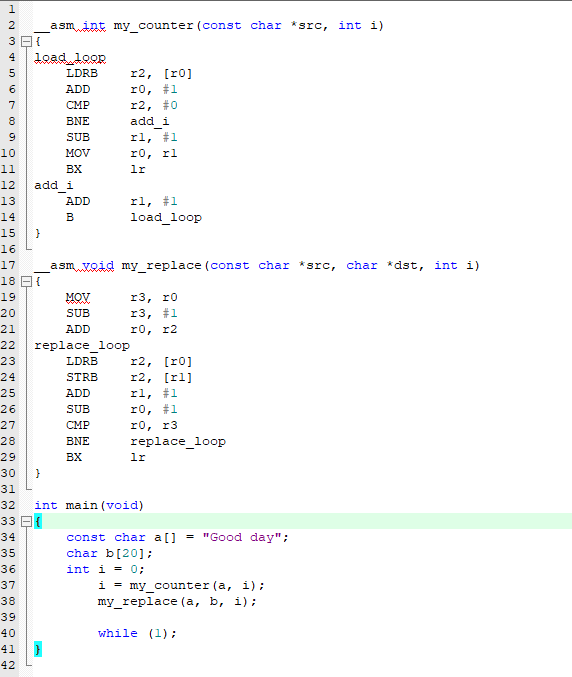
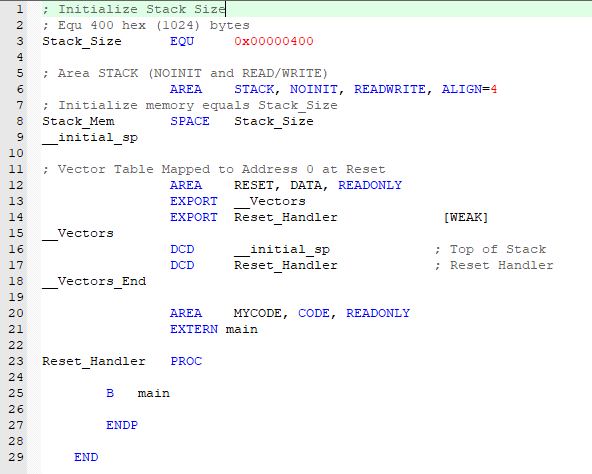
* при використанні так званого вбудованого асемблера (або, в разі засобів GNU, inline-асемблера) в тексті програми на С;
* при виклику з програми на С функції або процедури, написаної на асемблері і реалізованої в окремому файлі;
* при виклику з асемблерної програми процедури або функції, написаної на С.

При цьому необхідно чітко уявляти собі, яким чином здійснюється передача параметрів в функцію, що викликається, і повернення результату її роботи в програму, яка її викликала. Механізм такої взаємодії детально описаний у [стандарті AAPCS](http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042f/IHI0042F_aapcs.pdf). Коротко це виглядає так:

1. Вхідні параметри передаються в регістрах R0-R3. Якщо ж обсяг переданих даних більше 4 \* 32 біт (важливо не кількість переданих, а їх загальний розмір), то вони передаються через R0-R3 + стек.
2. Значення, що повертається, зазвичай зберігається в R0, а якщо це значення займає 2\*32 біта, то в R0-R1.
3. Регістри R0-R3 можуть змінюватися в функції, тоді як вміст R4-R11, R12 і LR слід зберігати при вході в підпрограму і відновлювати при виході з неї. Зазвичай для цього використовують стек.
4. Значення SP має бути вирівняно за подвійним словом.

Завдання №1. Реалізувати копіювання рядка символів у другий рядок «з заду на перед».

**Лістинг програми**

* main.c
* mystartup.s

**Виконання програми**

1. Ініціалізуємо масив символів і тимчасову змінну:

const char a[] = "Good day"; - строка в якій записане початкове слово.

char b[20]; - строка в яку буде записуватися відповідь, початкове слово

перероблене з заду на перед.

int i = 0; - використовується для збереження значення кількості символів в масиві ‘а’, яке буде обраховане в функції my\_counter.

1. Викликаємо функцію my\_counter, передаємо в неї адресу масиву ‘а’ (в регістр r0) та змінну ‘і’ (в регістр r1). Вона виконує обрахунок символів у масиві ‘а’. У регістр r2 завантажується байт за адресою, яка записана в регістрі r0, тобто перший символ масива ‘а’. Після цього до регістру r0 додається 1, щоб перейти на наступний символ. Записане в регістр r2 значення порівнюється з 0 (команда CMP), якщо воно не дорівнює нулю то програма перейде до мітки add\_і, в іншому випадку програма продовжить своє виконання. Коли програма переходить до мітки add\_і, наступна команда буде додавання до регістру r1 одиниці, тобто зарахування цього символу, після цього виконується перейде до міти load\_loop. Слова в масивах, мають на кінці слова значення 0, що інформує про закінчення слова, тому коли функція my\_counter дійде до кінця слова, команда порівняння з нулем, встановить Z флаг в логічну 1 і тому команда BNE не виконає перехід до мітки add\_і. Далі буде виконана команда SUB r1, #1 (яка слугує для певного корегування) і отримана порахована кількість елементів масива записується в регістр r0, так як функція my\_counter повертає значення. Отримане значення записується в змінну ‘і’.
2. Викликаємо функцію my\_replace, передаємо в неї адресу масиву ‘а’ (в регістр r0), адресу масиву ‘b’ (в регістр r1) та змінну ‘і’ (в регістр r2). Вона виконує запис слова з масива ‘а’ в масив ‘b’ з заду на перед. Спочатку у регістр r3 зберігається адреса масиву ‘а’ і відбувається корегування цієї адреси, відніманням одиниці, це потрібно для того, щоб записана адреса в регістрі r3 вказувала на початок слова записаного в масиві ‘а’, а не на перший його символ.

**Висновок**

В даній лабораторній роботі було розглянуто основи роботи асемблеру STM32F4. Досліджено синтаксис арифметичних і логічних команд. Розглянуто як відбувається розміщення констант і доступ до них. Було ознайомлено з роботою 3-х ступінчастого конвеєра та структури програми для STM32F4.