**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СИКОРСЬКОГО»**

**КАФЕДРА КОНСТРУЮВАННЯ ЕЛЕКТРОННО-ОБЧИСЛЮВАЛЬНОЇ АПАРАТУРИ**

**ЗВІТ**

з лабораторної робіти № 1

по курсу «Обчислювальні та МП засоби в РЕА-2»

Виконав:

студент гр. ДК-82

Дмитрук О.О.

Перевірив:

ст. викладач

Бондаренко Н.О.

Київ – 2021

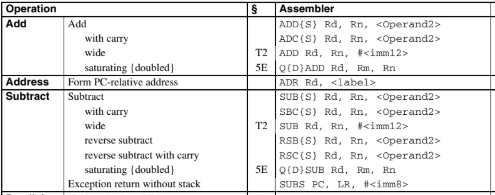
1. **Мета роботи**

**Дізнатися**:  
 - про структуру програми на мові асемблера для мікроконтролера STM32F4xx;  
 - про основні директиви для визначення сегментів програми, констант і змінних;  
 - про команди пересилань, доступу до пам’яті і арифметичної та логічної обробки даних;  
 - про склад прапорців стану програми і особливості впливу на них команд обробки даних.   
**Навчитися**:  
 - встановлювати й налагоджувати IDE Keil μVision5;  
 - створювати проект у Keil μVision5;  
 - створювати програми на мові асемблера для мікроконтролерів STM32F4xx у   
Keil μVision5;  
- налагоджувати програми у режимі емуляції і безпосередньо у мікроконтролері.

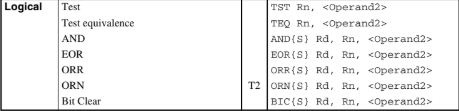
* 1. **Команди, які досліджуються у роботі**

В даної лабораторної роботі досліджуються:

* арифметичні команди – **ADD**/**ADDS** та **SUB** / **SUBS** (сума та різниця без і з впливом на стан прапорців);
* логічні команди - **AND** (ТА), **ORR** (АБО), **EOR** (виключне АБО),
* команди для роботи з пам’яттю - **LDR** (завантажити регістр з пам’яті), **STR** (завантажити вміст регістру у пам’ять);
* команди регістрових пересилань **MOV** (переслати /копіювати) і **MVN** (переслати з інверсією)
  1. **Синтаксис арифметичних команд**



* 1. **Синтаксис команд логічної обробки**



1. **ЗАВдання (Варіант 4)**
2. Створити проект у IDE Keil μVision5 та програму, що виконує

* Арифметичний розрахунок виразу:   
  Q = ((X + Y) – Z – (Z – Y)) – X
* де X = 15h, Y = 35h, Z = 04h
* Логічне перетворення: Q = (!(A+B+C+D))\*(B\*!C\*D ⊕ 0x17) де  
  А = 14h, B = 3Fh, C = 1Dh, D = 03h.

1. Дослідити різні способи завдання вихідних даних, особливості виконання команд обробки даних і їх вплив на стан прапорців з регістру xPSR.
2. **ОПИС програми**

Програма складається з визначення стеку, його вершини та таблиці векторів. Після виходу процесора зі стану скидання (*Reset*) він зчитує із пам’яті два 32-бітних значення і передає керування на основну програму \_main за допомогою процедури-обробника.

Основні директиви, що використовуються, з їх параметрами:

**AREA** [Section\_name], {type,..}, {attr,..}, {align}

***Section\_name*** - ім’я сегменту. Ім’я RESET - зарезервоване. Сегмент з цим ім’ям буде розташовано на початку памяті.

***type* -** Тип сегменту – CODE (команди) або DATA (лише дані, при старті або рестарті ініціалізується нулями).

***attr*** - атрибути сегмента:

*NOINIT* - сегмент даних не ініціалізується лише для даних - тип сегменту можна не писати.

*COMMON* - спільний сегмент даних – для команд і даних,  заповнюється нулями, використовується лише для сегменту даних.

*READ* - сегмент лише для читання.

*READWRITE* - сегмент для читання і запису. (READ - CODE, READWRITE - DATA)

*ALIGN* - вирівнювання сегмента по байтам-  ціле число [0…31]. За замовчуванням ALIGN = 2. Для системи команд THUMB заборонено ALIGN= 0.

**SPACE** *expr -* резервує кількість байт вказану в expr і заповнює цю память нулями.

**DCD** expression{expression}:

Директива DCD виділяє слова в пам'яті, вирівняні на кордонах чотирьох байт.

**EXPORT –** директива, що оголошує символ для дозволу посилань символів в окремих об'єктах і файлах бібліотек. **GLOBAL** є синонімом **EXPORT**.

Область пам’яті поділена на :

* *Code region* - для розміщення і доступу до програмного коду (стартова адреса - 0x00000000);
* *SRAM-region* - длярозміщення і доступу до даних і стеку (стартова адреса - 0x20000000);
* *Peripherals region* - длядоступу до периферії (стартова адреса - 0x40000000);
* *Internal Peripherals* - длявнутріпроцесорних компонент управління і налагодження (0xE0000000)

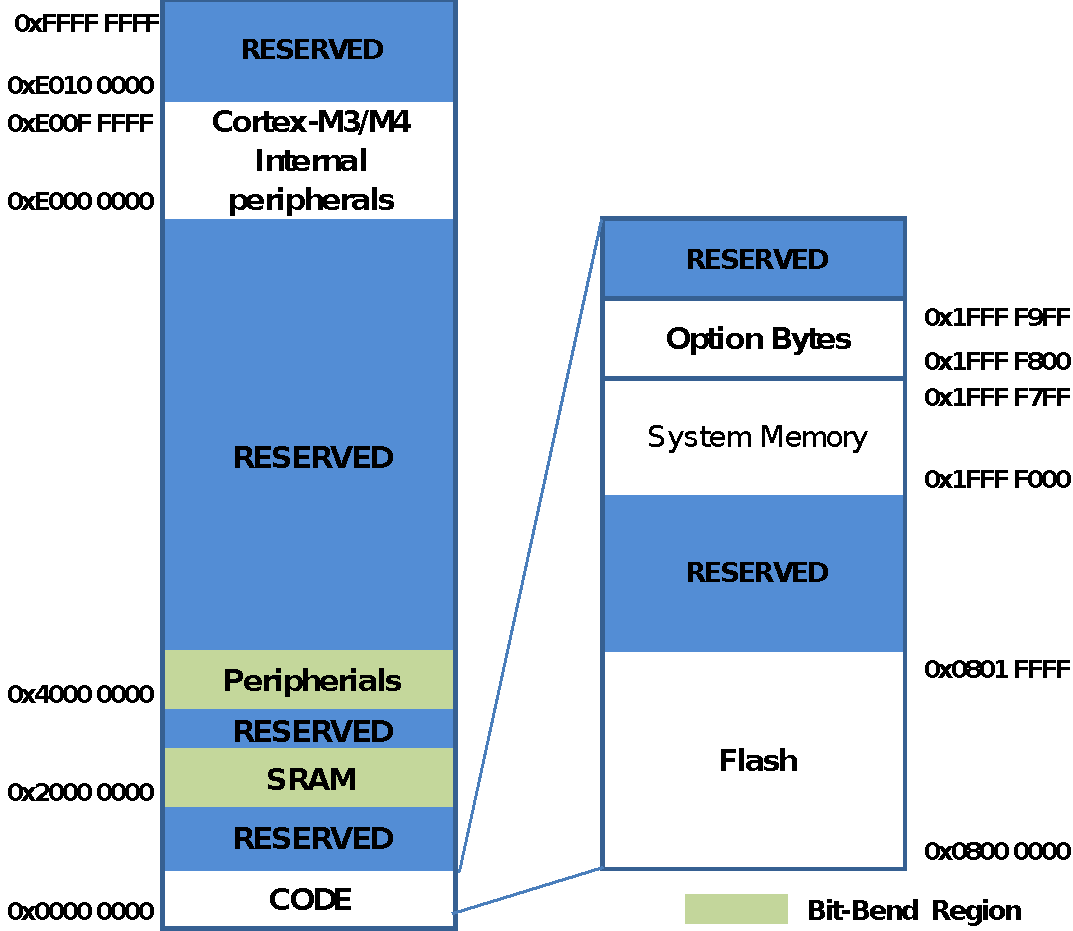


Рис. 1 - Карта пам’яті мікроконтролерів STM32F4xx

Після написання програми збираємо проект командою Build (F7). Натискаємо Сtrl-F5 для відлагодження програми. У вікні Registers бачимо значення, що змінюються під час покрокового виконання інструкцій програми.

Ініціалізація значень реалізується командою DCD, яка встановлює 32 бітне значення

Розберемось з визначенням констант і змінних, та способу адресації до них.STM32 являється мікроконтролером архітектури Register-Register, тому операндами над якими виконуються інструкції є регістри. Дані загружаються до регістрів через директиву завантаження даних LDR. Також операндами можуть бути певні константи типу imm , які містяться в регістрах, тощо.

Для того щоб завантажити значення за допомогою директиви LDR необхідно вказати регістр, у який необхідно зберегти значення, а також адресу памяті, в якій находяться дані . Для такого способу, можна створювати змінні за допомогою директиви DCD (Define double Word), яка створює змінну у пам’яті.

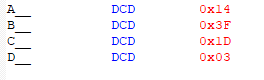


Рис. 2. Створення змінної (DCD)

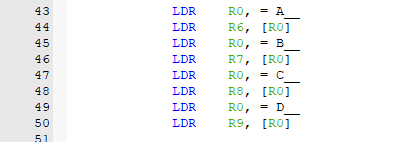


Рис. 3. Використання директив LDR

Оператор = каже загрузити адресу змінної!

Тобто запис типу:

**LDR RO, = Label**

перетворюється асемблером в команду типу

ldr.w R0, [pc,#offset]

Де PC – адреса інструкції

#offset – зміщення

Щоб дана конструкція працювала в сегменті коду після останньої команди програми, асемблер розміщує показники на константи і змінні, тобто записує в них адреси памяті, у яких лежать константи та змінні. Зміщення “#offset” рівне відстані до комірки-показника.

MCU Cortex-M3/M4 мк. STM32 має в собі 3-х ступінчастий конвеєр, який виконує такі дії: fetch – decode – execution, тобто присутній паралелізм, під час декодування виконується фаза fetch . За одну транзакцію з пам’яті одночасно витягуються 4 байти (одна 32-бітова, або дві 16-бітові інструкції) і після їх вибірки РС збільшиться на 4 Тому у фазі дешифрування і фазі виконання першої команди адреса комірки пам’яті, вміст якої буде зчитано у регістр r0, визначатиметься як

PC+offset+4

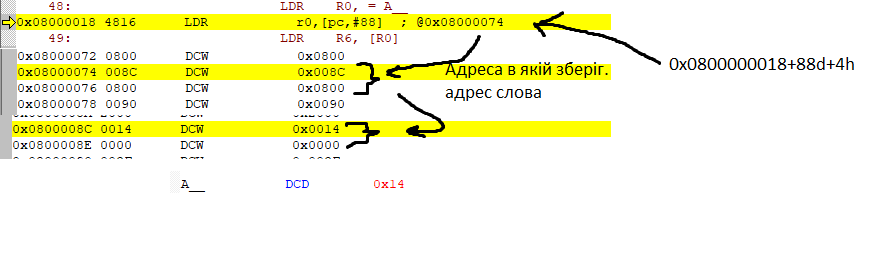


Рис.4. Орган.доступу до змінної у памяті

Молодші біти значень векторів (і взагалі будь яких покажчиків адрес у Code-сегменті) повинні містити лог.1. що вказує процесорові на застосування набору команд Thumb/Thumb-2. Як що ж цей біт буде скинутим у 0, то процесор зробить спробу переключитися до виконання команд з набору ARM.

Директива STR зберігає значення регістра до пам’яті за певною адресою.

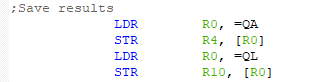


Рис. 5. Використання директиви STR

Також можна створити змінну за допомогою директиви EQU та використовувати її як аргумент типу Imm.



рис.6.Створення константи (EQU)



рис.7. Запис в регістри констант директивою MOV

Також в STM32 є базовий регістр флагів, за допомогою якого можна обробляти результати логічних і арифметичних операцій.

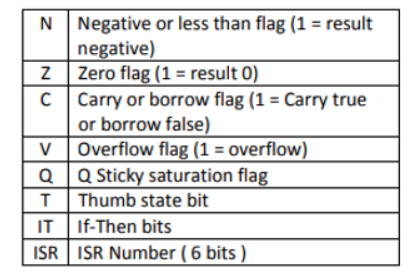


рис.8. Регістр флагів

Розглянувши базові операції, структуру програми STM32 та розібравшись з базовими поняттями, можемо приступити до реалізації самої програми.

1. Арифметичні операції:

Q = ((X + Y) – Z – (Z – Y)) – X

де X = 15h, Y = 35h, Z = 04h;

Для обчислення використаємо асемблерні інструкції, відповідно: додавання - ADD, віднімання SUB. Порядок дій звичайний, дужки вказують на першочерговість виконання дій.

Q = ((15h + 35h) – 04h – (04h – 35h)) – 15h =>

(4Ah-04h-(04h-35h))-15h =>

46h – (04h-35h)-15h =>

46h-FFFFFFCFh-15h =>

77h – 15h = 62h

Звіримо результат розрахований власноруч з результатом виконання програми:

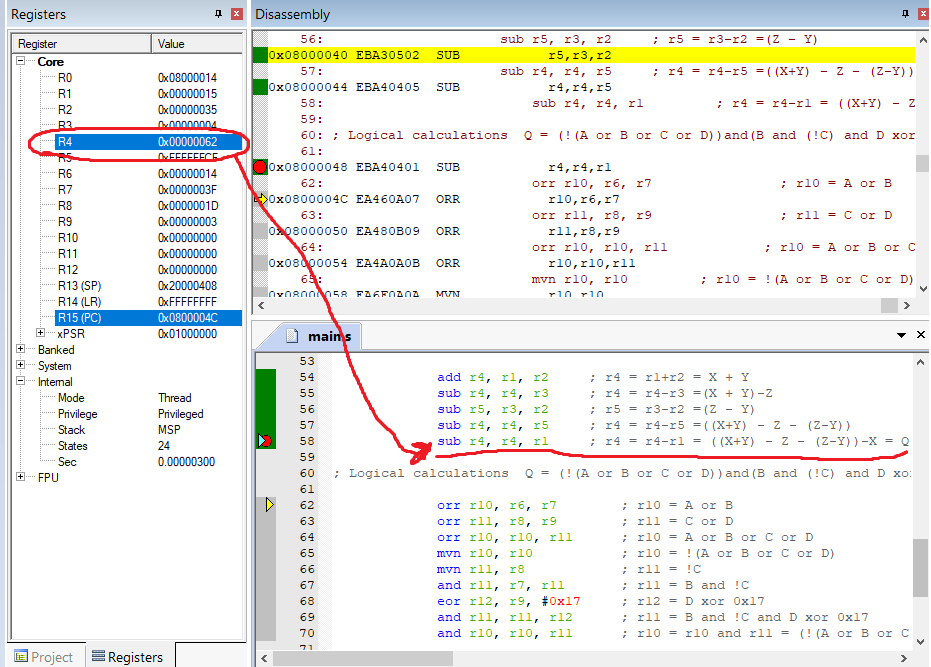


рис.9. Отриманий результат арифметичних дій

Як бачимо, в регістр R4 записався результат арифметичних дій, вміст якого збігається з власноруч отриманим результатом.

Виконаємо логічні операції:

1. Логічні операції:

Q = (!(A+B+C+D))\*(B\*!C\*D ⊕ 0x17) де  
А = 14h, B = 3Fh, C = 1Dh, D = 03h.

Мнемоніки логічних операцій і порядок виконання за спаданням (зверху більший пріоритет):

Дужки вказують на першочерговість виконання дій.

Інверсія - MVN

кон’юнкція – AND

{диз’юнкція – ORR

ділення по модулю на 2 – EOR}.

Виконаємо логічні операції власноруч:

0001\_0100 = А = 14h

0011\_1111 = B = 3Fh

0001\_1101 = C = 1Dh

0000\_0011 = D = 03h

(A+B+C+D) => { 0001\_0100 +

0011\_1111 +

0001\_1101 +

0000\_0011}

0011\_1111 =>(A+B+C+D)

!(A+B+C+D) => 1100\_0000

!C => 1110\_0010

B\*!C\*D = { 0011\_1111 \*

1110\_0010 \*

0000\_0011 }

0000\_0010 => B\*!C\*D

(B\*!C\*D) ⊕ 0x17 = { 0000\_0010 ⊕

0001\_0111 }

0001\_0101 => (B\*!C\*D) ⊕ 0x17

(!(A+B+C+D))\*(B\*!C\*D ⊕ 0x17) = { 1100\_0000 \*

0001\_0101 }

0000\_0000 = результат

Звіримо результат розрахований власноруч з результатом виконання програми:

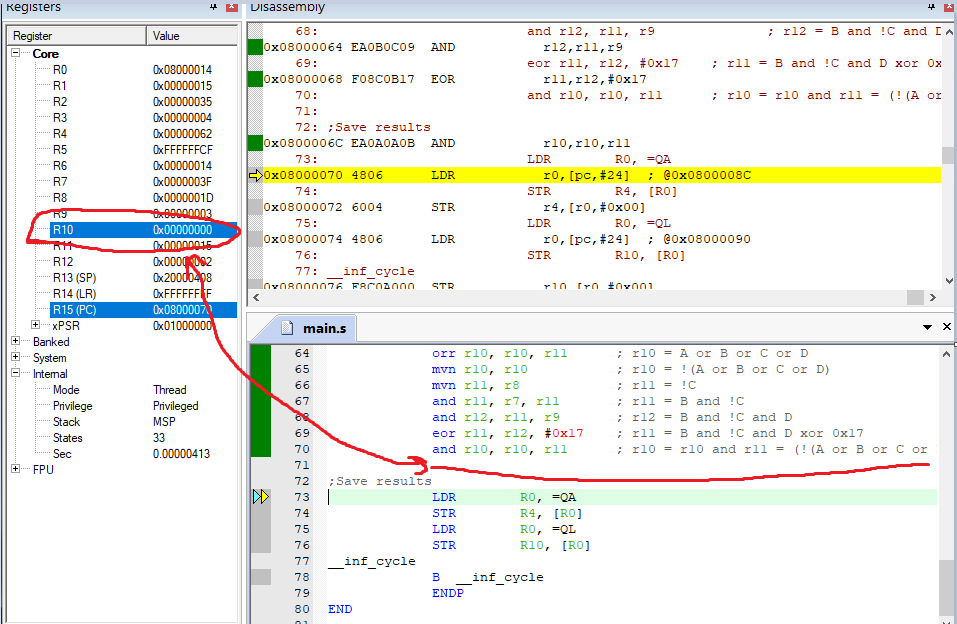
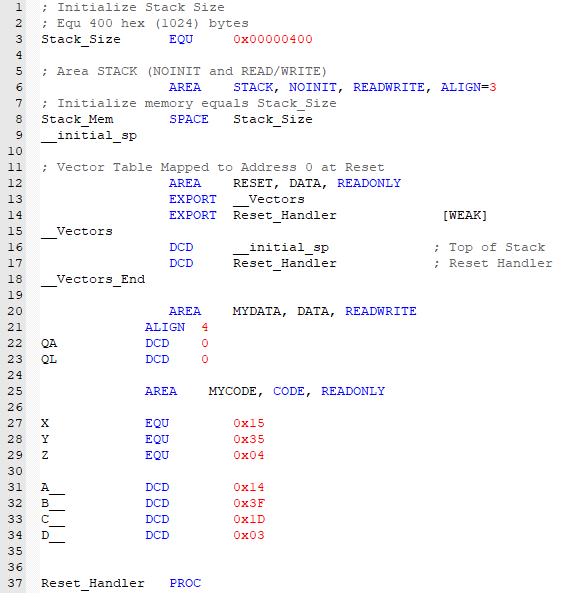


рис.10. Отриманий результат логічних операцій

Як бачимо результат збігається з отриманим власноруч результатом.

1. **Текст програми**



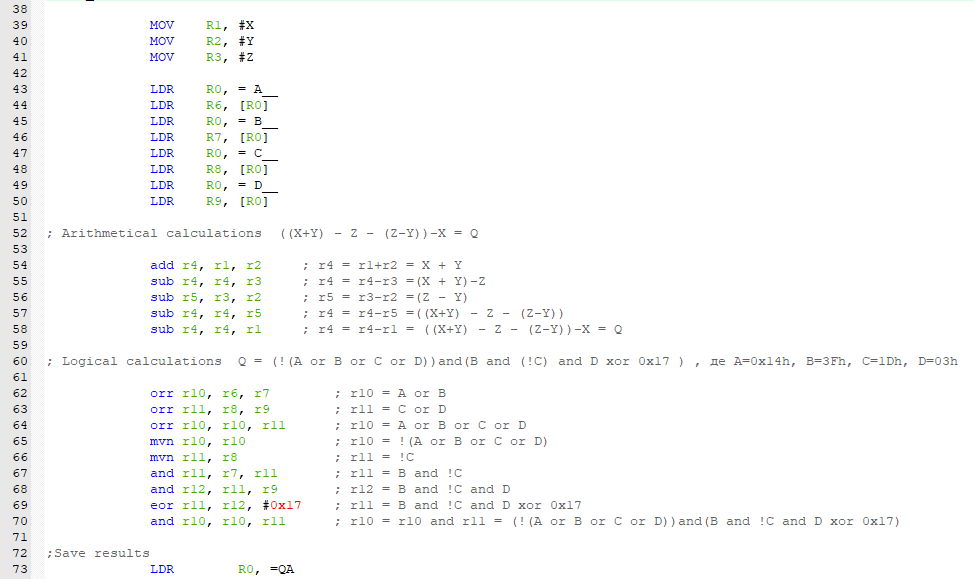
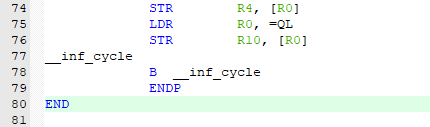
****

Рис. 11. Текст програми

**Висновок**

В даній лабораторній роботі було розглянуто основи роботи з язиком асемблеру STM32F4 (окрім умовних переходів, переривань та ін.). Розглянута структура програми для STM32F4.