**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 №1

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**

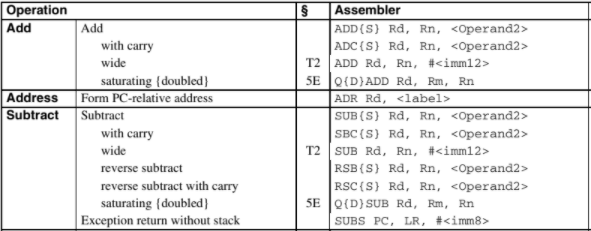
**Дізнатися**:  
 - про структуру програми на мові асемблера для мікроконтролера STM32F4xx;  
 - про основні директиви для визначення сегментів програми, констант і змінних;  
 - про команди пересилань, доступу до пам’яті і арифметичної та логічної обробки даних;  
 - про склад прапорців стану програми 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.

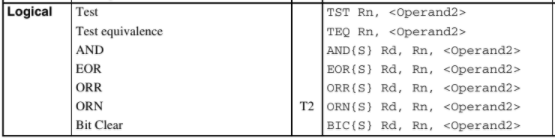
* 1. **Команди, які досліджуються у роботі**

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), **BIC** (АБО-НІ);
* 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**



**1.4.Пріорітет виконання операцій у логічних виразах**

**Завдання (Варіант 7)**

1. Створити проект у IDE Keil μVision5 та програму, що виконує:

* Арифметичний розрахунок виразу Q = (X + Z – Y) + X – Y + Z,

де X = 53h, Y = 34h, Z = 05h;

* Логічне перетворення Q = (AC + BD ⊕ 0x13)⋅!((A ⊕ B)),

де A=55h, B=30h, C=21h, D=E0h.

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

**Опис програми**

* 1. **Структура програми**

Розроблена програма має послідовну структуру з «пустим» безкінечним циклом наприкінці (рис.1).



Рисунок 1 – Блок-схема програми

* 1. **Опис окремих блоків програми**
     1. Будь-яка програма для мікроконтролера STM32F401RE складається з декілька ділянок пам’яті (*сегментів*) трьох типів:  
         - стеку STACK  
         - даних DATA  
         - коду CODE,   
        які відрізняються їх іменами і властивостями.

Останні завдаються параметрами (атрибутами) READWRITE та READONLE (рис.2)

* + 1. Особливе призначення має сегмент типу READONLE DATA із зарезервованим ім’ям RESET, що розташовується з самого початку пам’яті програм (з адреси 0x08000000).

У цьому сегменті розмішені 4-байтні елементи так званої *таблиці векторів переривань (Vector table)*, початок і кінець якої позначається *зарезервованими* іменами

**\_\_Vector** і **\_\_Vector\_End** (вони починаються з подвійного підкреслення).  
Будь яка програма повинна мати при найми хоча б два елементи цієї таблиці:   
перший повинен містити адресу верхівки стеку із зарезервованим ім’ям **\_\_initial\_sp**, а другий ̶ адресу початкової підпрограми **Reset\_Handler**, з якої управління передається основній процедурі, що зазвичай зветься - **main**.   
Ці елементи завдаються директивами асемблера DCD (*Define Constant Data*).

На рис.1 показана частина програми («преамбула»), яка майже без змін (за винятком наповнення таблиці векторів переривань і розміру стеку) є початком кожної програми на асемблері для STM32.

Изображение выглядит как снимок экрана

Автоматически созданное описание

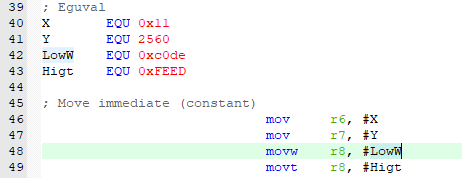
Рисунок 1 – Преамбула програми – визначення сегментів, таблиці векторів переривання і Reset Handler - обробника скидання (RESET)

* 1. **Визначення констант і змінних та способи доступу до них**

STM32 як і будь який RISC-мікроконтролер виконує всю обробку даних лише у регістрах. Данні потрапляють у регістри як *константи* вказані у команді оператором #**imm**, або завантажуються у регістр, як вміст комірок пам’яті, до яких звертається спеціальна команда завантаження регістру **ldr** (*load register*).

* + 1. У першому випадку (рис.2,а) завантаження константи у регістр здійснюється командами переміщення **mov** (*move*) , або **mvn** (move not). Можливо також, спочатку, за допомогою директиви **EQU** присвоїти числовій константі деяке *ім’я*, а потім завантажити її значення у регістри тимиж командами **mov** (рис.2,б).

а)



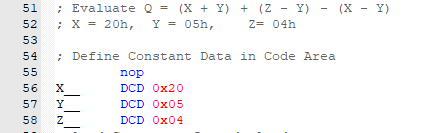
б)

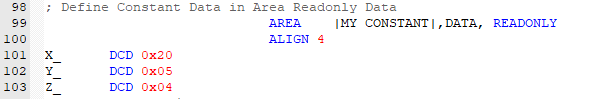
Рисунок 2 - Завантаження у регістр констант,   
а) безпосередньо заданих у команді; б) визначених директивою EQU

Дві останні команди записують 16-бітні числа (*напівслова*) відповідно у молодшу і у старшу половини 32-бітного регістру.

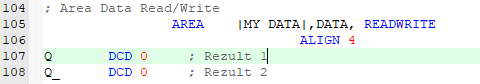
* + 1. Директивою DCD можна записати подвійне слово (32 біти) в пам’ять і завдати цієї конструкції з чотирьох 1-байтових комірок ***ім’я*** , яке буде вказувати на комірку з меншою адресою (це відповідатиме способу розміщення багаторозрядних даних ***little endian***, який прийнято у контролерах на основі Cortex-M за замовчанням).

При цьому ми можемо визначити:   
 − або **константу**, як що задамо її ім’я у сегменті коду (рис.3,а) чи у ділянці пам’яті даних типу READONLY (рис.3,б);  
 − або **змінну**, якщо зробимо це у пам’яті даних типу READWRITE (рис. 3,в.)



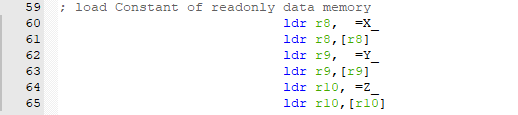
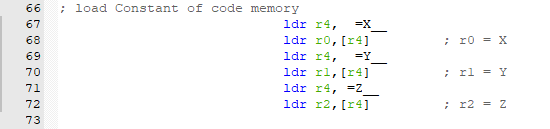
а)

б)



в)

Рисунок 3 ̶ Визначення констант і змінних  
а) в сегменті коду; б) в області READONLY DATA;   
в) в сегменті данихREADWRITE

* + 1. Доступ до цих констант і змінних здійснюється однаково, а саме – командою **LDR** , при завантаженні регістрів (рис.4,а) і командою **STR** (*Store register*) при запису їх вмісту у пам’ять (рис.4,б)

а)



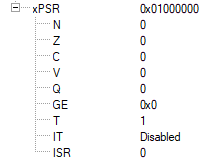
б)

Рисунок 4 – Завантаження констант у регістри (а)  
та копіювання їх вмісту у пам’ять (б)

**Дослідження виконаня програми в Keil μVision5**

* 1. **Стартовий стан програми**

На рис.5 показано стартовий стан регістрів CPU, регістру стану програми (xPSR), вміст таблиці векторів переривань і дизасембльований код першої команди **LDR R0, =\_main**

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 5 – Стан програми після скидання (RESET) мікроконтролера

Елементи таблиці векторів (\_\_initial\_sp = 0x20000408) і Reset\_Handler (0x080000009) завантажено у відповідні регістри (SP) і (PC) процесора. В регістрі стану програми xPSR всі прапорці скинуто у стан «0», за винятком прапорця <T>, що вказує на застосування процесором системи команд Thumb/Thumb2.

* 1. **Розміщення констант і змінних та доступ до них**
     1. Константи визначені директивою EQU не займають ні якого місця у пам’яті. Асемблер зберігає їх у службовій таблиці імен і підставляє їх значення, як частину команди – операнд **#imm,** як і тоді, коли константи безпосередньо задаються в команді (рис.5)

Изображение выглядит как снимок экрана

Автоматически созданное описание

≡>

Рисунок 5 – Способи завантаження констант у регістри процесора

* + 1. Константи, визначені директивою DCD у сегменті code, розміщаються на тому ж місті де вони і визначені. У вікні Disassembly кожна така константа буде визначено двома директивами DCW (*Define Code HalfWord*) (рис.6).

Изображение выглядит как снимок экрана

Автоматически созданное описание

Рисунок 6 – Розміщення констант визначених директивою DCD

* 1. **Організація доступу до констант і змінних, розміщених у пам’яті**

Изображение выглядит как снимок экрана, держит, комната, экран

Автоматически созданное описаниеВище в п.3.3.3. (див. рис.4) показано, що доступ до констант і змінних, визначених як у READONLY, так і у READWRITE областях пам’яті, виконується однаковою конструкцією з двох команд. При цьому, перша команда ініціює регістр вказівник адресою комірки пам’яті, а друга здійснює непряме читання вмісту цієї комірки у певний регістр, або запис вмісту регістру у комірку, на яку вказує регістр-вказівник (рис.7):

Рисунок 7 – Конструкції команд для доступу до пам’яті

Причому, як показано у інструкції к даної роботі ([1], ст. 18) перший елемент конструкції, тобто запис типу

ldr rt, = Label

перетворюється асемблером у команду з відносною адресацією

ldr.w rt, [pc, #offset]

Для підтримки роботи цієї конструкції у сегменті коду, зразу за останньою командою програми, асемблер розміщує *показники* на константи і змінні, тобто записує в них *адреси* комірок пам’яті, у яких лежать константи та змінні (рис.8). При цьому зміщення #offset в команді дорівнює відстані до комірки-показника

pc + 4 + #offset

Изображение выглядит как снимок экрана

Автоматически созданное описание

pc + 4 + #80 = 0x08000044+4+#80 (80 = 0x50)

a)

Изображение выглядит как снимок экрана

Автоматически созданное описание

Адреса константи

б)

Изображение выглядит как снимок экрана, комната

Автоматически созданное описание

Константа 0x20

в)

Рисунок 8 – Організація доступу до константи у пам’яті

Тут важливо нагадати, що MCU Cortex-M3/M4 мікроконтролерів STM32 мають в своєму складі 3-х ступінчастий конвеєр: **fetch**🡪**decode**🡪**execution** (рис.9). Тобто, під час дешифровки однієї команди, виконується вибірка наступної.

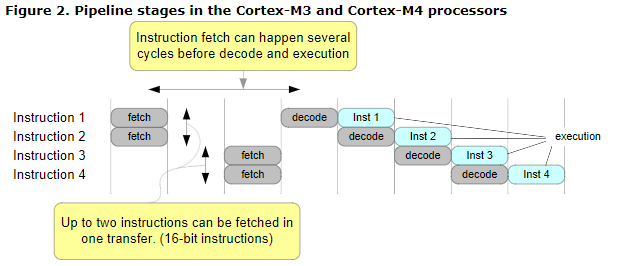


Рисунок 9 – Конвеєр в архітектурі Cortex-M3 та Cortex-M4.

Це яскраво видно при налагоджуванні програми на командах роботи з пам’яттю (рис.10) – дизасемблер вказує на адресу в пам’яті на 4 байти більше очікуваної



Рисунок 10 – Приклад розбіжності показань дизасемблера та визначення адреси [pc,#72]

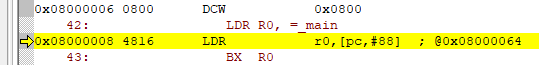
Вміст регістра РС перед початком команди дорівнює 0х08000024. Тоді в R0 мало би записатись число, що лежить за адресою 0х08000024+0х00000048=0х0800006С (48h=72), але мітка у коментарі вказує, що запис в R0 буде здійснюватися з адреси 0х08000070.

Це пояснюється тим, що у фазі ***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 (РС=0х08000024+4=0х08000028). Тому у фазі дешифрування (***decode***) і фазі виконання (***execution***) першої команди адреса комірки пам’яті, вміст якої буде зчитано у регістр r0, визначатиметься як PC + 0х00000048 = 0х08000028 + 0х00000048 = 0х08000070, на що і вказує коментар @0х08000070.

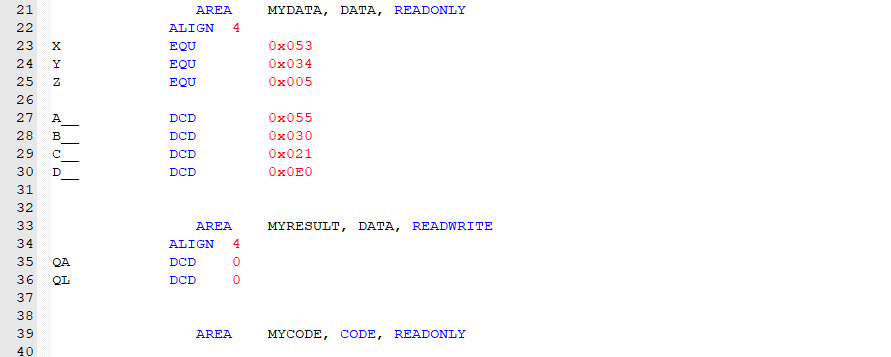
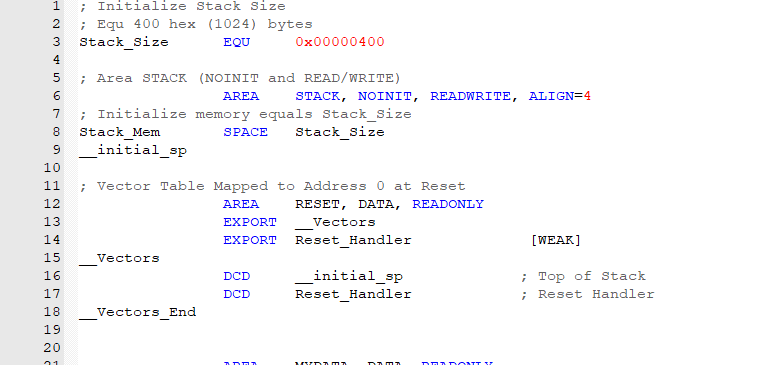
Запис результатів роботи програми здійснюється в сегмент пам’яті програм, що дозволяє як читання, так і запис даних (це визначено атрибутом сегменту - READWRITE).

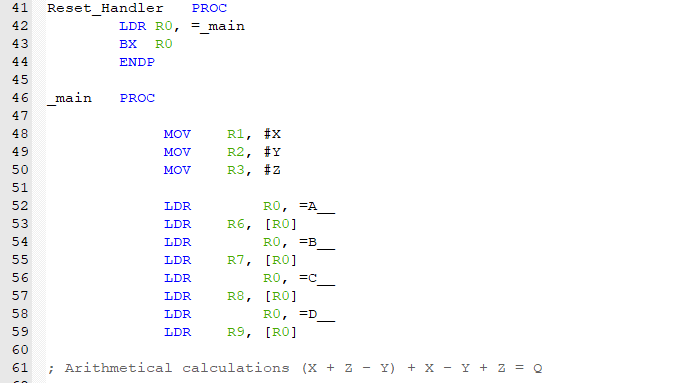
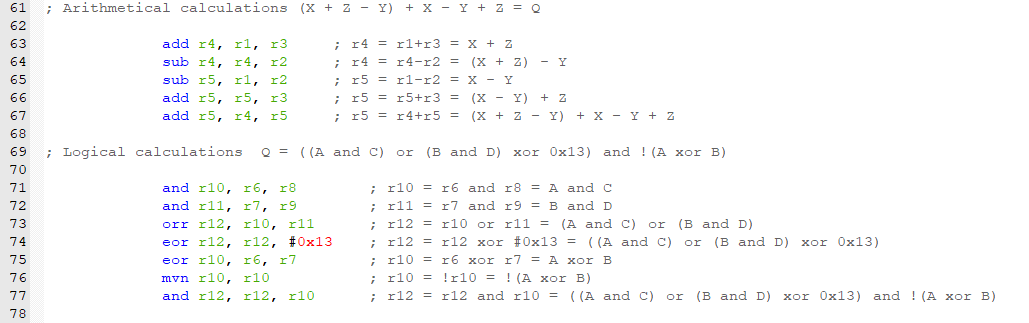
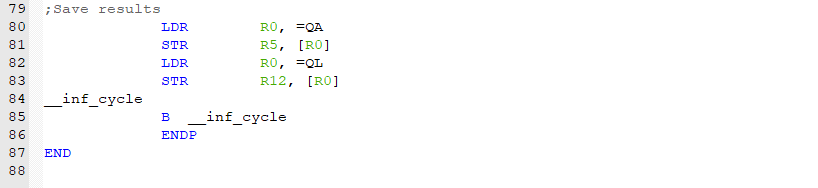
**Додатково**

Молодші біти значень векторів (і взагалі будь яких покажчиків адрес у Code-сегменті) повинні містити «1» («одиницю»), 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.** Це цілком природно для ранніх моделей ARM-процесорів, в яких застосовувалися різні набори команд. Але ж у процесорів Cortex-M, які виконують лише команди **Thumb/Thumb-2**, це недопустимо і визве генерацію виключення відмови (Hard Fault).

Адреса початку програми:

Адреса яка записана у векторі Reset\_Handler:

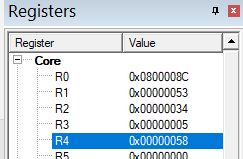
**Текст програми**

****

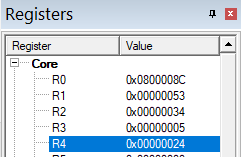
**Перевірка вірності результатів арифметичних і логічних команд**

Арифметичні обрахунки ( (X + Z – Y) + X – Y + Z = Q):

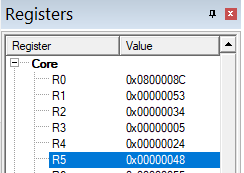
1. r4 = r1+r3 = X + Z = 53 + 5 = 58h

Значення регістрів

1. r4 = r4-r2 = (X + Z) – Y = 58 – 34 = 24h

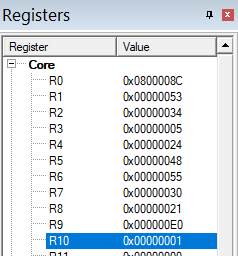
Значення регістрів

1. r5 = r1-r2 = X – Y = 53 – 34 = 1Fh
2. r5 = r5+r3 = (X – Y) + Z = 1F + 5 =24h
3. r5 = r4+r5 = (X + Z – Y) + X – Y + Z = 24 + 24 = 48h

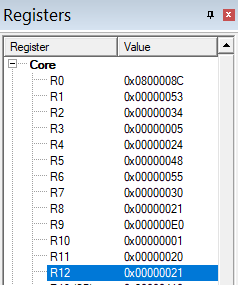
Значення регістрів

Логічні обрахунки ( ((A and C) or (B and D) xor 0x13) and !(A xor B) = Q):

1. r10 = r6 and r8 = A and C = 01010101b and 0010001b = 00000001b = 1h

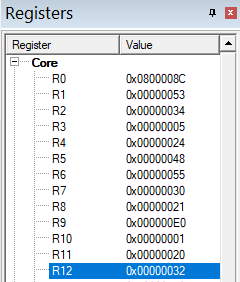
Значення регістрів

1. r11 = r7 and r9 = B and D = 00110000b and 11100000b = 00100000b = 20h
2. r12 = r10 or r11 = (A and C) or (B and D) = 00000001b or 00100000b = 00100001b = 21h

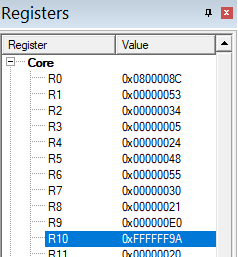
Значення регістрів

1. r12 = r12 xor #0x13 = ((A and C) or (B and D) xor 0x13) = 00100001b xor 00010011b = 00110010b = 32h

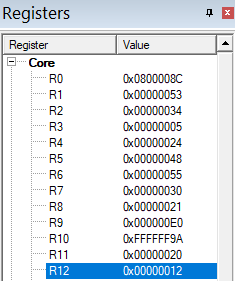
Значення регістрів



1. r10 = r6 xor r7 = A xor B = 01010101b xor 00110000b = 01100101b = 65h
2. r10 = !r10 = !(A xor B) = !01100101b = 10011010b = 9Ah

Значення регістрів

1. r12 = r12 and r10 = ((A and C) or (B and D) xor 0x13) and !(A xor B) = 00110010b and 10011010b = 00010010b = 12h

Значення регістрів

Перевірка чи значення обрахунків записалися в змінні QA і QL:

* Адреса QA: 0x20000000, результат який потрібно записати Q = 48h

Значення за даною адресою

* Адреса QL: 0x20000004, результат який потрібно записати Q = 12h

Значення за даною адресою

**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.