**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-XX

< П.І.Б>

Перевірив:

ст. викладач

Бондаренко Н.О.

Київ – 2021

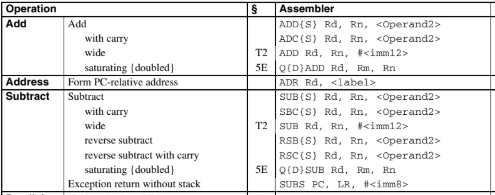
1. **Мета роботи**

**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.

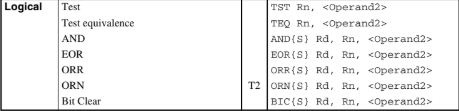
* 1. **Команди, які досліджуються у роботі**

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), **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. **Завдання (Варіант N)**
2. Створити проект у IDE Keil μVision5 та програму, що виконує

* Арифметичний розрахунок виразу:   
  Q = (X + Y) + (Z - Y) - (X – Y), де X = 20h, Y = 05h, Z= 04h;
* Логічне перетворення: Q = (0x34 + !A⋅B⋅C ⊕ D)⋅!(A⋅!D⋅C),   
  де A=44h, B=12h, C=13h, D=C1h

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

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



Рисунок 1 – Блок-схема програми

Повний текст програми надано наприкінці звіту, у розділі 5.

* 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)

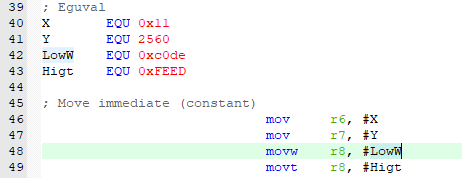
* 1. **Визначення констант і змінних та способи доступу до них**

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 тімі ж командами **mov** (рис.2,б).



а)



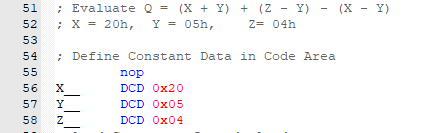
б)

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

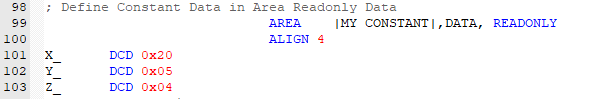
Дві останні команди записують 16-бітні числа (*напівслова*) відповідно у молодшу і у старшу половини 32-бітного регістру.

* + 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.)

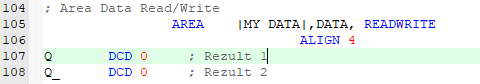




а)



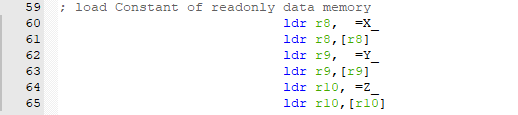
б)

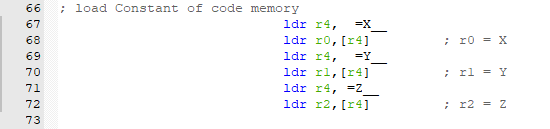


в)

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)

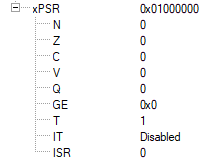


б)

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

1. **Дослідження виконання програми у 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**

Вище 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):

Изображение выглядит как снимок экрана, держит, комната, экран

Автоматически созданное описание



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

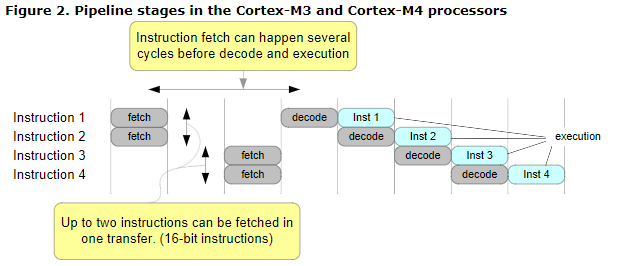
Автоматически созданное описание

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.



Рисeунок 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

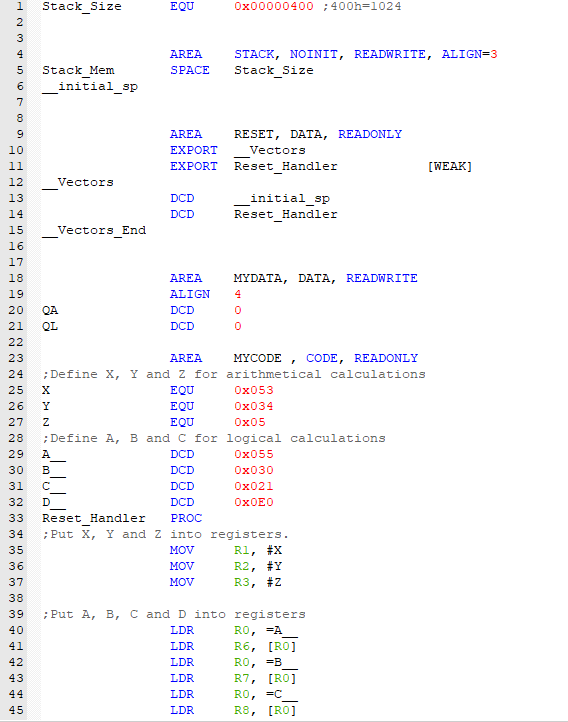
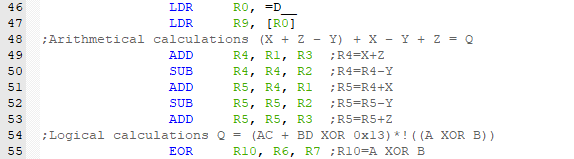


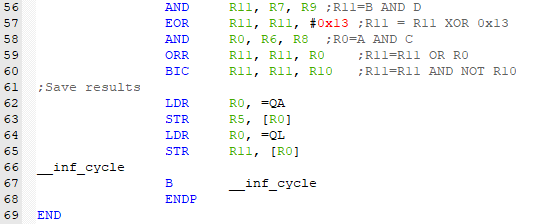
Рисунок 10 – Приклад розбіжності показань дизасемблера та визначення адреси [pc,#72]

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).

1. **Текст програми**



**Висновок**

В даній лабораторній роботі було розглянуто основи роботи з язиком асемблеру STM32F4 (окрім умовних переходів, переривань та ін.). Розглянуто роботу 3-х ступінчастого конвеєра та структуру програми для STM32F4.