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.

1.1.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)

1.2.Синтаксис арифметичних команд

Operation		§	Assembler
Add	Add		`ADD{S} Rd, Rn, <Operand2>`
	with carry		`ADC{S} Rd, Rn, <Operand2>`
	wide	T2	`ADD Rd, Rn, #<imm12>`
	saturating {doubled}	5E	`Q{D}ADD Rd, Rm, Rn`
Address	Form PC-relative address		`ADR Rd, <label>`
Subtract	Subtract		`SUB{S} Rd, Rn, <Operand2>`
	with carry		`SBC{S} Rd, Rn, <Operand2>`
	wide	T2	`SUB Rd, Rn, #<imm12>`
	reverse subtract		`RSB{S} Rd, Rn, <Operand2>`
	reverse subtract with carry		`RSC{S} Rd, Rn, <Operand2>`
	saturating {doubled}	5E	`Q{D}SUB Rd, Rm, Rn`
	Exception return without stack		`SUBS PC, LR, #<imm8>`

1.3.Синтаксис команд логічної обробки

Logical	Test		`TST Rn, <Operand2>`
	Test equivalence		`TEQ Rn, <Operand2>`
	AND		`AND{S} Rd, Rn, <Operand2>`
	EOR		`EOR{S} Rd, Rn, <Operand2>`
	ORR		`ORR{S} Rd, Rn, <Operand2>`
	ORN	T2	`ORN{S} Rd, Rn, <Operand2>`
	Bit Clear		`BIC{S} Rd, Rn, <Operand2>`

2. ЗАВДАННЯ (ВАРІАНТ 4)

1. 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 A = 14h, B = 3Fh, C = 1Dh, D = 03h.
2. Дослідити різні способи завдання вихідних даних, особливості виконання команд обробки даних і їх вплив на стан прапорців з регістру xPSR.

3. 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* – для розміщення і доступу до даних і стеку (стартова адреса - 0x20000000);
- *Peripherals region* – для доступу до периферії (стартова адреса - 0x40000000);
- *Internal Peripherals* – для внутріпроцесорних компонент управління і налагодження (0xE0000000)

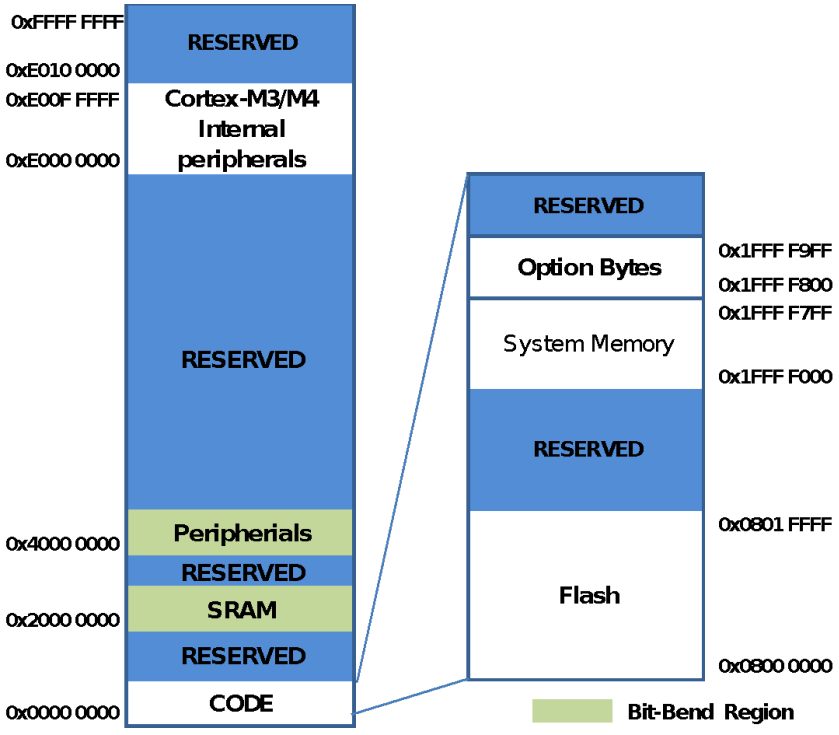

Рис. 1 - Карта пам’яті мікроконтролерів STM32F4xx

Після написання програми збираємо проект командою Build (F7). Натискаємо Ctrl-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.

```
A__     DCD     0x14
B__     DCD     0x3F
C__     DCD     0x1D
D__     DCD     0x03
```

Рис. 2. Створення змінної (DCD)

```
43          LDR     R0, = A__
44          LDR     R6, [R0]
45          LDR     R0, = B__
46          LDR     R7, [R0]
47          LDR     R0, = C__
48          LDR     R8, [R0]
49          LDR     R0, = D__
50          LDR     R9, [R0]
51
```

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 PC 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

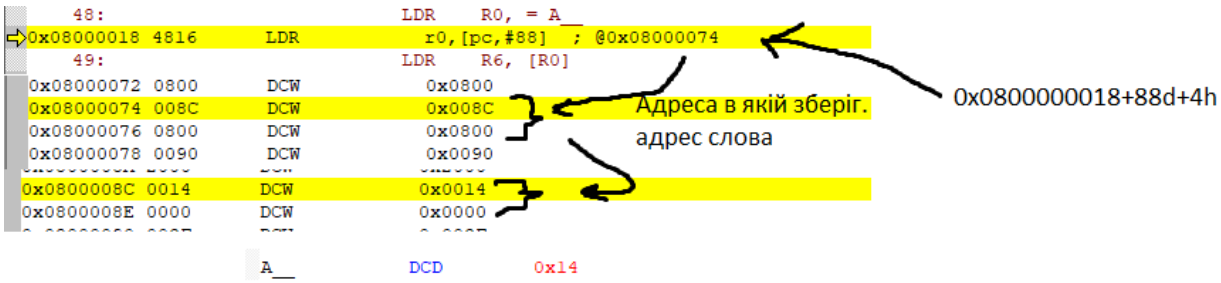

Рис.4. Орган.доступу до змінної у памяті

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.

```
;Save results
                LDR         R0, =QA
                STR         R4, [R0]
                LDR         R0, =QL
                STR         R10, [R0]
```

Рис. 5. Використання директиви STR

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

```
27  X       EQU     0x15
28  Y       EQU     0x35
29  Z       EQU     0x04
```

рис.6.Створення константи (EQU)

```
39          MOV     R1, #X
40          MOV     R2, #Y
41          MOV     R3, #Z
```

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.

N	Negative or less than flag (1 = result negative)
Z	Zero flag (1 = result 0)
C	Carry or borrow flag (1 = Carry true or borrow false)
V	Overflow flag (1 = overflow)
Q	Q Sticky saturation flag
T	Thumb state bit
IT	If-Then bits
ISR	ISR Number (6 bits)

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:

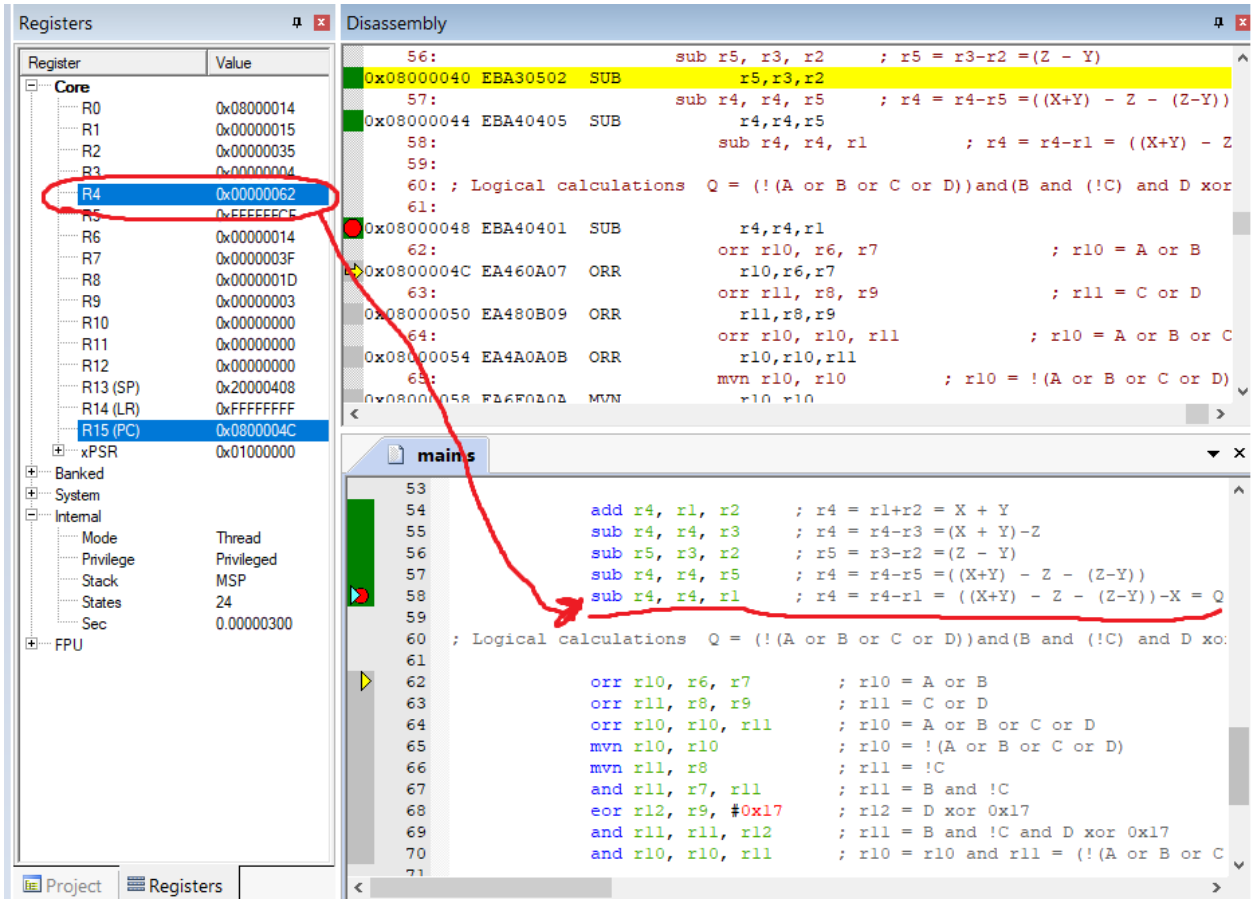

рис.9. Отриманий результат арифметичних дій

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

Виконаємо логічні операції:

2. Логічні операції:

Q = (!(A+B+C+D))*(B*!C*D ⊕ 0x17) де

A = 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 = A = 14h

0011_1111 = B = 3Fh

0001_1101 = C = 1Dh

0000_0011 = D = 03h

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

0011_1111 +
0001_1101 +
<u>0000_0011}</u>
0011_1111 =>(A+B+C+D)

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

!C => 1110_0010

B*!C*D = {0011_1111 *
1110_0010 *
<u>0000_0011</u> }
0000_0010 => B*!C*D

(B*!C*D) ⊕ 0x17 = { 0000_0010 ⊕
<u>0001_0111</u> }
0001_0101 => (B*!C*D) ⊕ 0x17

(!(A+B+C+D))*(B*!C*D ⊕ 0x17) = { 1100_0000 *
<u>0001_0101 }</u>
0000_0000 = результат

Звіримо результат розрахований власноруч з результатом виконання програми:

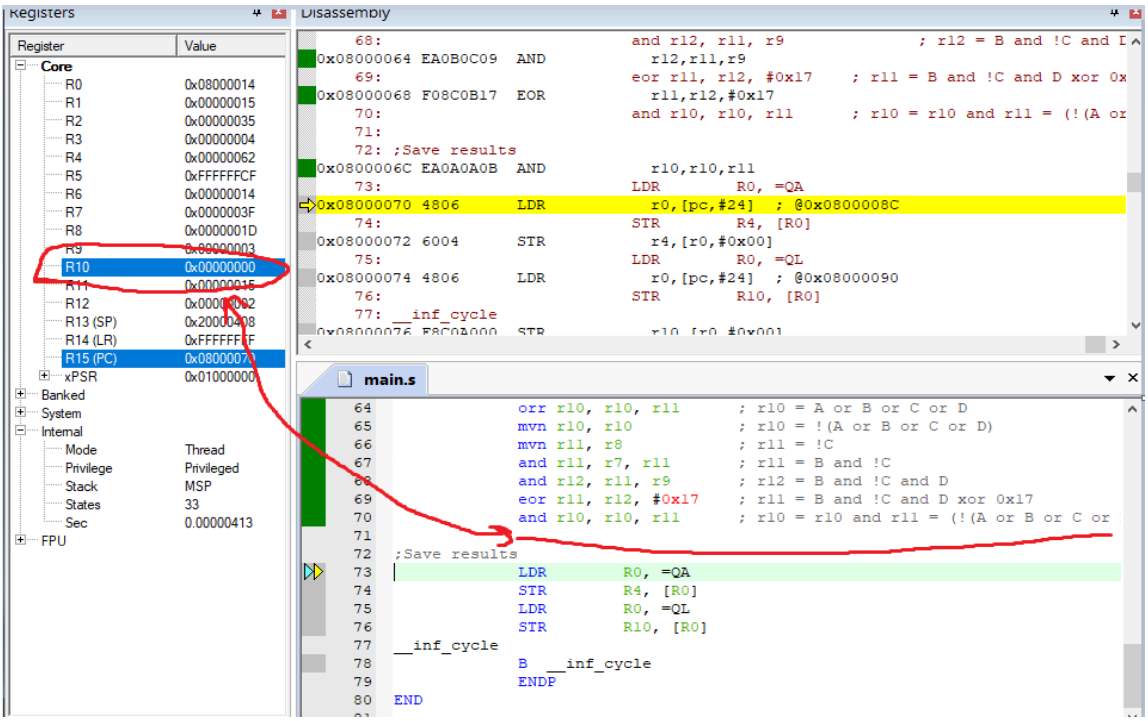

рис.10. Отриманий результат логічних операцій

Як бачимо результат збігається з отриманим власноруч результатом.

4. ТЕКСТ ПРОГРАМИ

```
; Initialize Stack Size
; Equ 400 hex (1024) bytes
Stack_Size      EQU     0x00000400

; Area STACK (NOINIT and READ/WRITE)
                AREA    STACK, NOINIT, READWRITE, ALIGN=3
; Initialize memory equals Stack_Size
Stack_Mem       SPACE   Stack_Size
__initial_sp

; Vector Table Mapped to Address 0 at Reset
                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors
                EXPORT  Reset_Handler             [WEAK]
__Vectors
                DCD     __initial_sp            ; Top of Stack
                DCD     Reset_Handler           ; Reset Handler
__Vectors_End

                AREA    MYDATA, DATA, READWRITE
             ALIGN  4
QA           DCD    0
QL           DCD    0

             AREA    MYCODE, CODE, READONLY

X            EQU        0x15
Y            EQU        0x35
Z            EQU        0x04

A__          DCD        0x14
B__          DCD        0x3F
C__          DCD        0x1D
D__          DCD        0x03


Reset_Handler   PROC
```

```
38
39              MOV     R1, #X
40              MOV     R2, #Y
41              MOV     R3, #Z
42
43              LDR     R0, = A__
44              LDR     R6, [R0]
45              LDR     R0, = B__
46              LDR     R7, [R0]
47              LDR     R0, = C__
48              LDR     R8, [R0]
49              LDR     R0, = D__
50              LDR     R9, [R0]
51
52  ; Arithmetical calculations  ((X+Y) - Z - (Z-Y))-X = Q
53
54              add r4, r1, r2      ; r4 = r1+r2 = X + Y
55              sub r4, r4, r3      ; r4 = r4-r3 =(X + Y)-Z
56              sub r5, r3, r2      ; r5 = r3-r2 =(Z - Y)
57              sub r4, r4, r5      ; r4 = r4-r5 =((X+Y) - Z - (Z-Y))
58              sub r4, r4, r1      ; r4 = r4-r1 = ((X+Y) - Z - (Z-Y))-X = Q
59
60  ; Logical calculations  Q = (!(A or B or C or D))and(B and (!C) and D xor 0x17 ) , де A=0x14h, B=3Fh, C=1Dh, D=03h
61
62              orr r10, r6, r7         ; r10 = A or B
63              orr r11, r8, r9         ; r11 = C or D
64              orr r10, r10, r11       ; r10 = A or B or C or D
65              mvn r10, r10            ; r10 = !(A or B or C or D)
66              mvn r11, r8             ; r11 = !C
67              and r11, r7, r11        ; r11 = B and !C
68              and r12, r11, r9        ; r12 = B and !C and D
69              eor r11, r12, #0x17     ; r11 = B and !C and D xor 0x17
70              and r10, r10, r11       ; r10 = r10 and r11 = (!(A or B or C or D))and(B and !C and D xor 0x17)
71
72  ;Save results
73              LDR       R0, =QA

74                  STR         R4, [R0]
75                  LDR         R0, =QL
76                  STR         R10, [R0]
77  __inf_cycle
78                  B   __inf_cycle
79                  ENDP
80  END
81
```

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.