Лабораторна робота №3. **Пряме програмування портів GPIO.  
Проект на мові асемблера і С.**

**Вступ**

З попередньої лабораторної роботи ми з’ясували, що кожна програма, написана для ARM архітектури, повинна мати декілька основних компонентів: стек, таблиця векторів переривань, ну і звісно тіло самої програми. У стеку будуть зберігатися локальні змінні, які розташовані у регістрах загального призначення, а також в деяких випадках значення адреси повернення з підпрограм, тобто вміст регістру R14(LR), і навіть вміст програмного лічильника R15(PC). У таблиці векторів має бути мінімум дві адреси: перша – адреса з якої починається стек, друга – адреса початку програми,. Приклад найпростішої програми на мові асемблеру можна побачити на рис. 1.

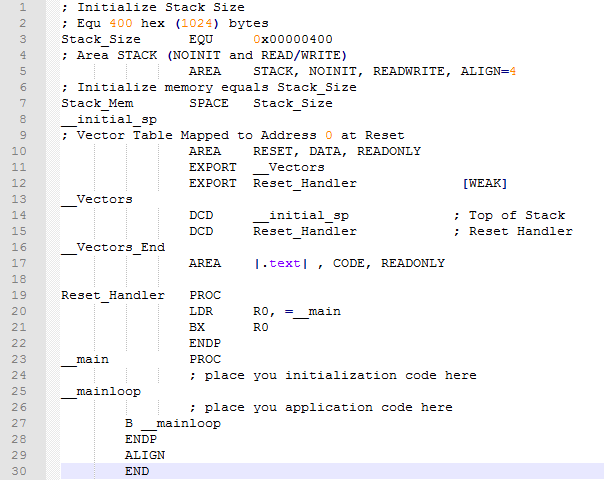


Рис. 1. - Лістинг коду найпростішої програми

**1.Підготовка до створення програми. Огляд необхідної документації**

Однак ця програма нічого не робить. Тож давайте допишемо її так, щоб вона увімкнула світлодіод, який розташований на платі STM32F401 NUCLEO, або STM32F407 Discovery (далі просто NUCLEO i Discovery)[[1]](#footnote-1). Для цього відкриємо інструкцію користувача (user manual UM1724) [1] і знайдемо розділ «LEDs» (ст. 23 ). Тут написано, що користувачу доступний світлодіод LD2 (рис. 2). Керування ним відбувається за допомогою периферійних блоків GPIO (*General Purpose Input/Outputs*), які прийнято називати *портами вводу/виводу*. Вони позначаються англійськими буквами A,B,C, …,J,K . Кожен порт містить 16 виводів (pins - пінив). Отже світлодіодом на платі NUCLEO керує пін «5» порту «A» («PA5»), і щоб його увімкнути необхідно подати на його анод високий рівень. На платі STM32F4 Discovery, згідно документу user manual UM1472 [3], програмам користувача доступні 4 світлодіоди LD3 – LD6, підключені до виводів PD12 – PD15. На рис. 3 надано схему підключення світлодіодів на платах NUCLEO і Discovery.

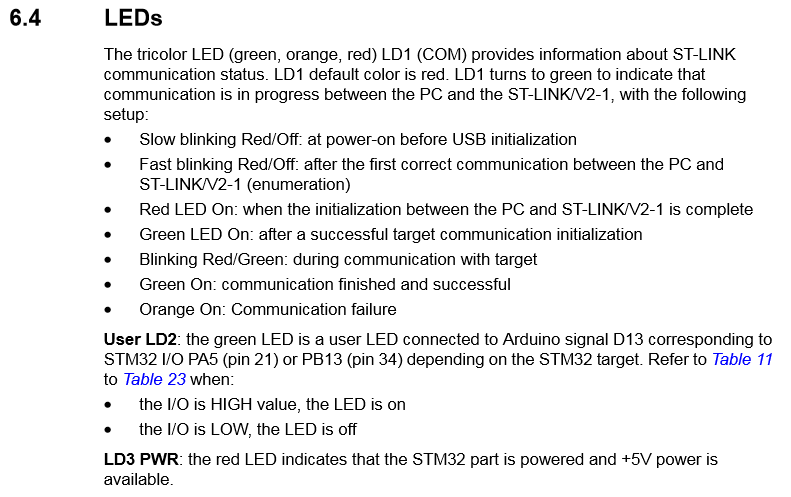
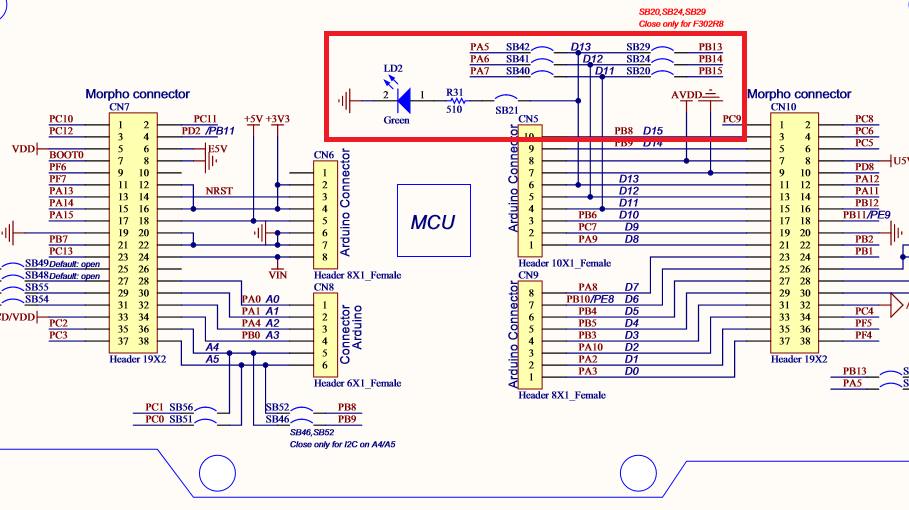
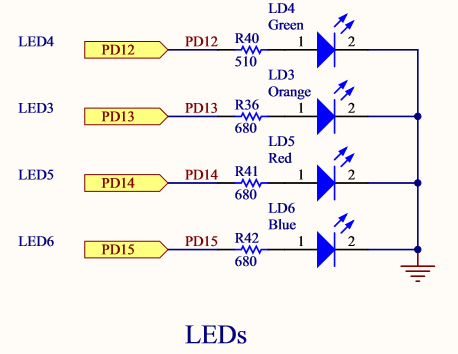


Рис. 2. - Опис світлодіодів, розташованих на платі NUCLEO



а) NUCLEO



б) Discovery

Рис. 3. - Схема підключення світлодіодів на платах  
а) NUCLEO, б) Dscovery

Для зниження електроспоживання мікроконтролеру (MCU), після його скидання (RESET), практично всі блоки периферійних пристроїв і утому числі всі порти вимкнуті. Увімкнення або вимкнення їх відбувається подачею або припиненням подачі на них тактуючого сигналу. Керування тактуючим сигналом відбувається за допомогою регістрів «RCC XXX peripheral clock enable» з загальної групи керування скиданням та синхронізацією RCC (*Reset and clock control*) у якої знаходяться всі регістри керування налаштуванням генераторів, блоків ФАПЧ та шин. Замість «XXX» в позначення регістру має бути назва шини «AHB1», «AHB2» або «APB1», яка забезпечує доступ до певного ресурсу MCU. Тому відкриваємо «reference manual RM0368/RM0090» [2]/[4] і в розділі «6.3. RCC registers»/«7.3RCC registers» шукаємо регістр, який відповідає за дозвіл (*enable*) тактування портів «А» i «D». Ця інформація є на ст. 117/178 (рис. 4).

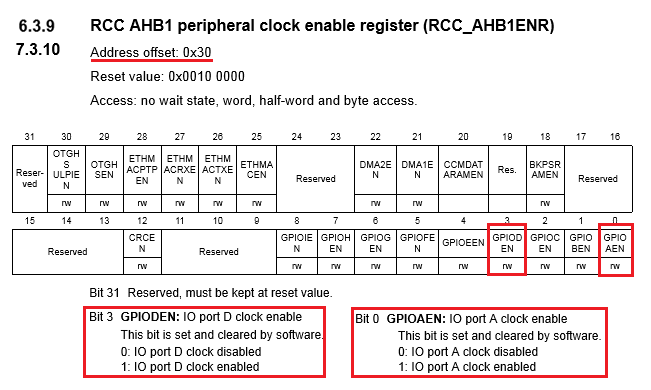


Рис. 4. - Біти дозволу/заборони тактування портів GPIOA та GPIOD

Бачимо, що треба в біт 0 / 3 регістру «RCC\_AHB1ENR» записати 1. Також тут написано, що зміщення цього регістру відносно базової адреси становить «0x30».

Осталось визначити базові адреси «RCC» і портів «GPIOA» та «GPIOD». Оскільки карта пам’яті для однакових ресурсів всіх MCU STM32F4xx співпадає, то відкриємо розділ «2.3 Memory map» (ст.38 [2] або ст.65 [4]) і побачимо, що значення цих адрес становить: «0х40023800» - для RCC, «0х40020000» - для порту «GPIOA», та «0х40020С00»- для порту «GPIOD» (рис. 5).

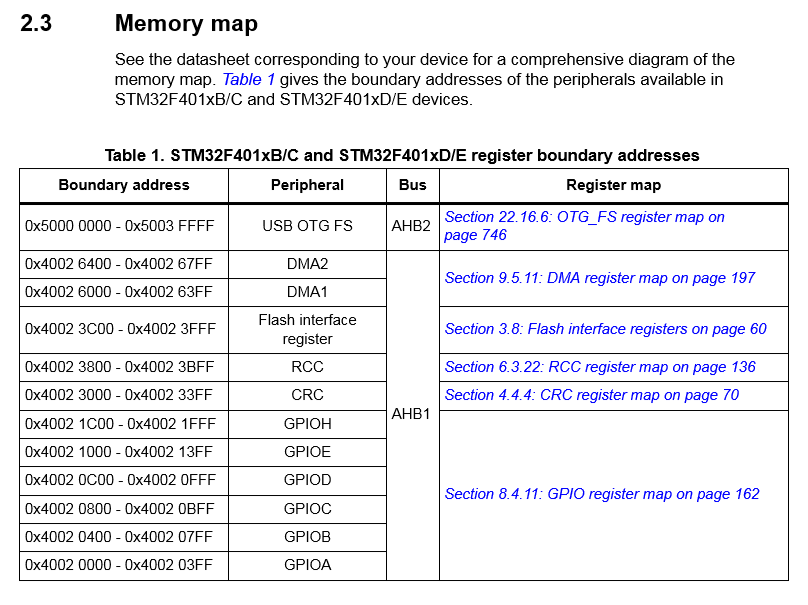


Рис. 5. – Фрагмент опису «memory map»

Більш стисло цю інформацію подано у розділі 6.3.22 [2] /7.3.25 [4] (рис.6):

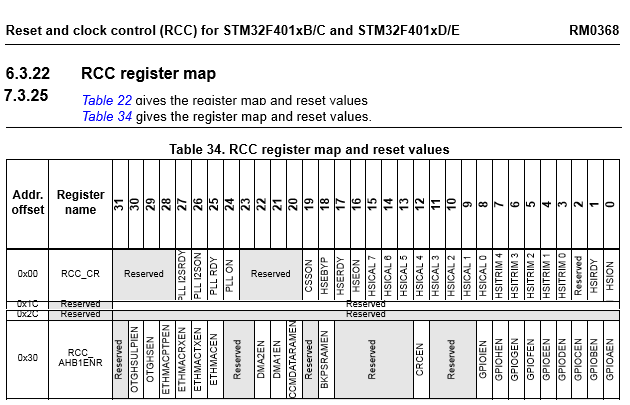


Рис.6 – Карта регістрів RCC групи

Тепер, коли порт «А» / «D» тактується, нам необхідно задати режим його роботи. Це робиться за допомогою регістру «GPIO port mode register» (**GPIOx\_MODER**), що описаний на ст. 157 (рис. 7). Його зміщення становить «0х00», а щоб настроїти пін 5 / 12 на «вивід» даних («output mode») з відповідного порту **GPIOx** (**х=A,B,C,D,…,H/…,K**) необхідно записати в біти «11,10» / «25,24» значення «**01**».

Для того ж, щоб встановити на виході високий або низький рівень, треба записати значення «**1»** відповідно в біти «5»/«12» або «21»/«28» регістру «GPIO port bit set/reset register» (**GPIOx\_BSRR**). Інформацію про це можна знайти на ст. 160 (рис. 8).

***Зверніть увагу*** *на те, що запис логічної одиниці «***1***» у молодші біти 0 ÷ 15, позначені як «*BSy,y=0÷15»*, призводить до встановлення («Set») високого рівня на відповідному виводі («піні») порту* **GPIOx.y** (y=0÷15)*, а запис «***1***» у біти 16 ÷ 31, позначених «*BRy,y=0÷15» *– до скидання («Reset») у низький рівень тих самих виводів* **GPIOx.y**(y=0÷15)*.*

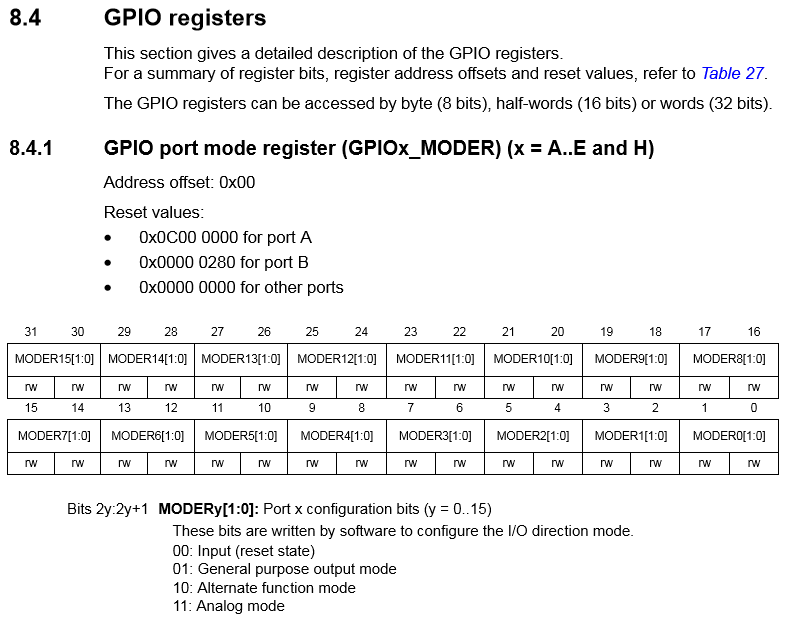


Рис. 7. - Опис регістру «GPIO port mode register»

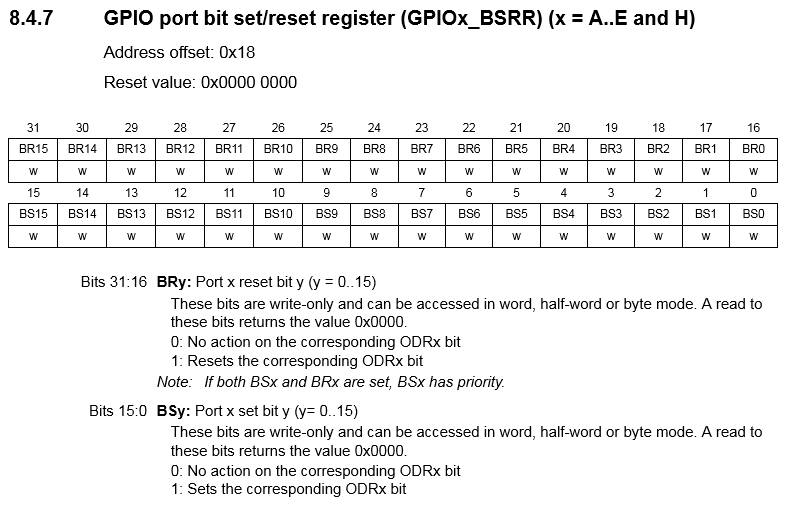


Рис. 8. - Опис регістру «GPIO port bit set/reset register»

**2. Програма на мові ASSEMBLER**

Отже, у нас є вся необхідна інформація і ми можемо починати писати програму. Додаймо декілька констант (рис. 9), щоб код легше будо читати, а «магічних» чисел в ньому було менше.

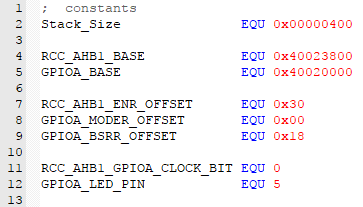
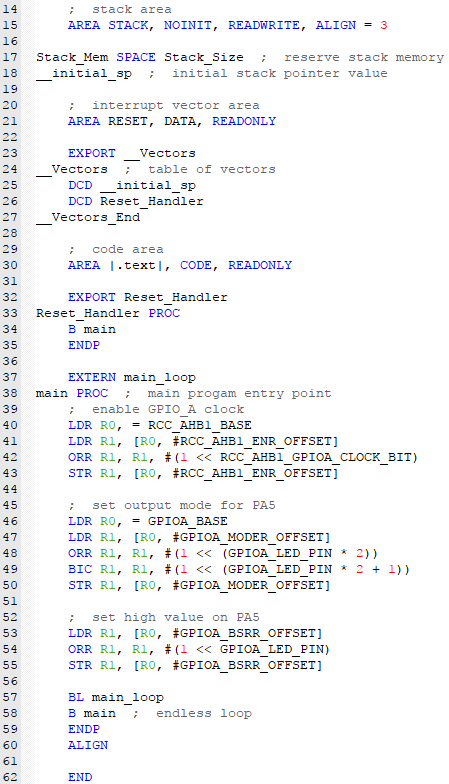


Рис. 9 - Константи програми

Також додаймо код, що вмикає світлодіод перед входом до «вічного» циклу (рис. 10).

Якщо ви хочете випробувати роботу отриманої програми вже зараз, тобто у «чисто асемблерному варіанті», то закоментуйте рядки 37 і 57, які відносяться до матеріалу який буде розглянуто в наступному пункті[[2]](#footnote-2),. Далі побудуйте проект так, як ви це вже робили у лабораторній роботі №1, і відслідкуйте його виконання у Debugger. А щоб перевірити працездатність програми на налагоджувальної платі Nucleo, завантажте в неї отриманий код (цю дію ви також робили в першій роботі), і запустіть програму на виконання. Як що ви будете досліджувати програму у debugger (крок за кроком), то світлодіод повинен загорітися після виконання останньої команди з блоку встановлення високого рівню на виводі РА5 (див. Рис. 10):

str r1, [ro,#GPIOA\_BSRR\_OFFSET]



Вилучите ці рядки при випробуванні ASM+C проекту

Закоментуйте ці рядки при випробуванні асемблерного варіанту програми

Рис. 10 - Код для увімкнення світлодіоду

**3.Увімкнення світлодіоду з програми на мові C**

Давайте спробуємо також написати увімкнення світлодіоду на мові «С», щоб потім порівняти з уже написаним на мові асемблеру в дебагері. Саме для цього у 37 рядку ми кажемо компілятору, що десь за межами даного файлу є **EXTERN**-функція, яка називається «**main\_loop**», а в рядку 57 ̶ переходимо на цю функцію. Тепер створимо файл з розширенням «.с» і напишемо там код, зображений на рис. 11. Тут видно, що ми створили функцію «**main\_loop**» і саме сюди перейде програма, після виклику функції з рядку 57 асемблерного файлу..

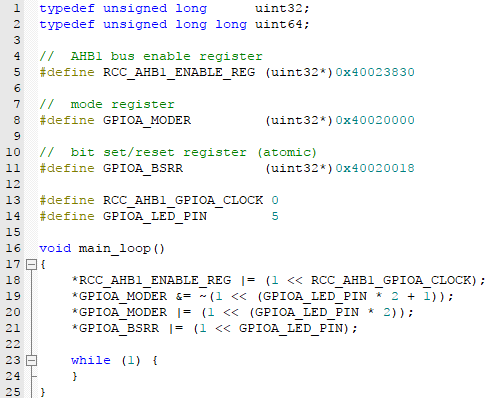


Рис. 11. Код програми на мові «С»

Взагалі-то тепер ( «для чистоти експерименту») у асемблерному файлі main.s рядки з 39 по 56 (див. рис. 10) бажано вилучити, або закоментувати. Однак ми залишимо їх, щоб за один раз провести порівняння інструкцій вихідного асемблерного коду із інструкціями, які створюються при компіляції відповідних операцій на мові С.

Додайте до проекту створений файл **main\_loop.c**[[3]](#footnote-3). Структура нашого проекту з двома файлами зображена на рис. 12.

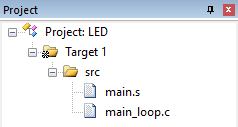


Рис. 12. Структура проекту

Щоб скомпілювати проект можна натиснути клавішу «F7» або на відповідну піктограму. Однак, як що проект вже було компільовано з іншими файлами у складі, то краще виконати команду <Rebuild>, натиснувши іншу кнопку (рис. 13).



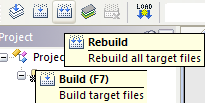


Рис. 13. Піктограма для побудови проекту

Якщо все зробити правильно, то має бути 0 помилок та 0 попереджень.

Відлагодження може відбуватися як на платі, так і у симуляторі. Це залежить від того, що обрано у налаштуваннях проекту «Project -> Options for Target…» у вкладці «Debug» (рис. 14).

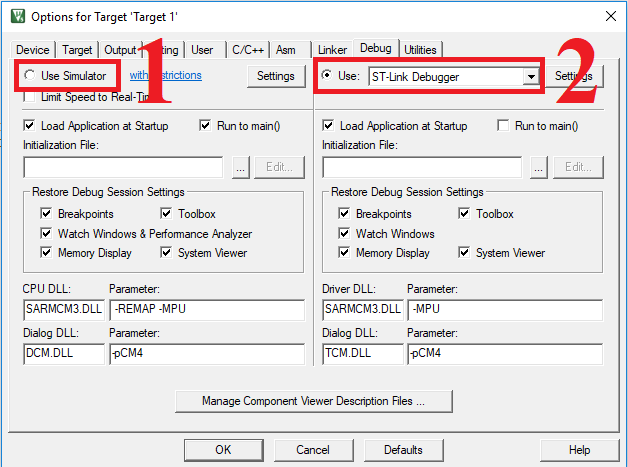


Рис. 14. Вибір способу відлагодження

Виберемо варіант 2, завантажимо програму ( або «Ctrl-F5») і почнемо покрокове відлагодження програми. При цьому відкриваються нові вікна (рис. 15).

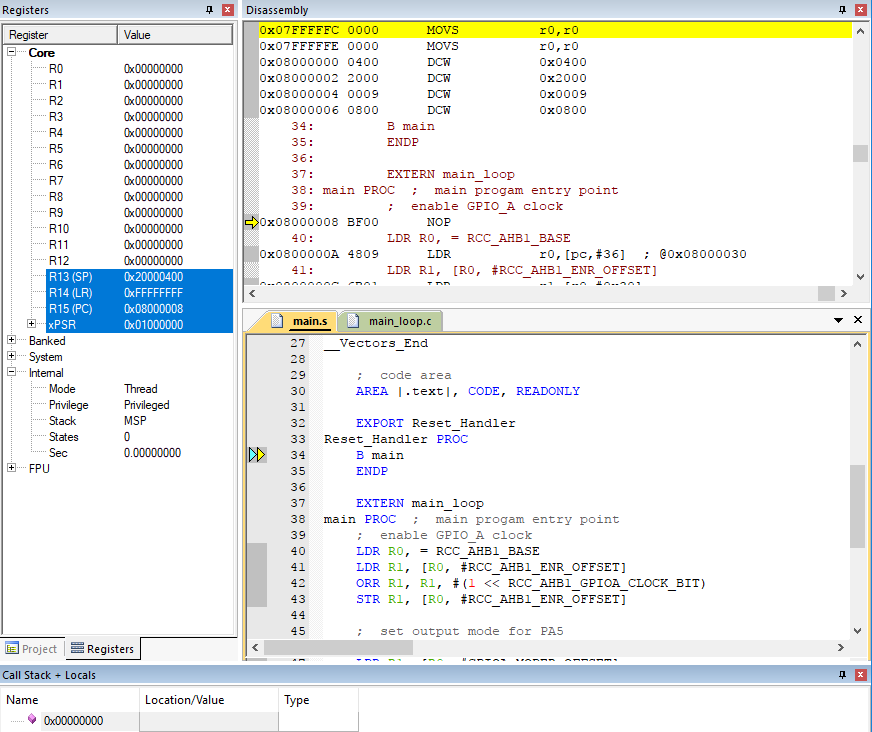


Рис. 15. Вікна з інформацією про відлагодження

Як і у роботі №1 праворуч зверху можна бачити вікно із результатом дизасемблювання скомпільованої програми, тобто саме ті інструкції, які і буде виконувати мікроконтролер. Жовта стрілка вказує на поточну інструкцію. Тут можна побачити, що починаючи з адреси «0х08000000», тобто у сегменті коду, записані 2 подвійних слова (у вигляді 4 слів, завданих командами DCW – Define Code Word). Це, як ви вже знаєте з першої роботи, ̶ перші два елементи таблиці векторів переривань. Нагадаємо, що перше подвійне слово (перші два слова), це адреса вершини стеку, а друге подвійне слово (два наступних слова) – адреса, з якою починається виконання програми. Зліва вікно, в якому можна подивитися поточний вміст кожного з регістрів. І як можна бачити в регістрах «SP» та «PC» («R13» та «R15» відповідно) вже знаходяться значення з таблиці векторів переривань.

У нижньої частині вікна Debugger, на відміну від роботи №1, розташовано дві вкладки з вихідними текстами файлів **main.s** та **main\_loop.c**.

Поглянувши на дизасембльований код рядків 48-49 файлу «main.s» (рис. 16), можна побачити, що замість бітових зсувів, множень і додавань, завданих у так званому **flexible operand 2** :  
 **#(1 << (GPIOA\_LED\_PIN\*2))**  та **#(1 << (GPIOA\_LED\_PIN\*2+1))** компілятор вставив реальні числа «0х400» та «0х800», з якими безпосередньо і виконуються прості операції «порозрядного або» - **ORR** (рядок 48):

ORR r1,r1,#0x400 ;r1:=r1ORR #0x400

та «очищення біту» Bit Clear – **BIC** (рідок 49):

BIC r1,r1,#0x800 ;r1:=r1 AND NOT #0x800

Таким чином код залишається зрозумілим, а мікроконтролеру не доводиться робити зайві операції.

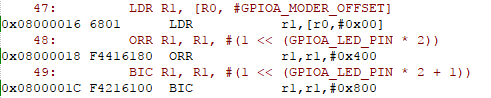
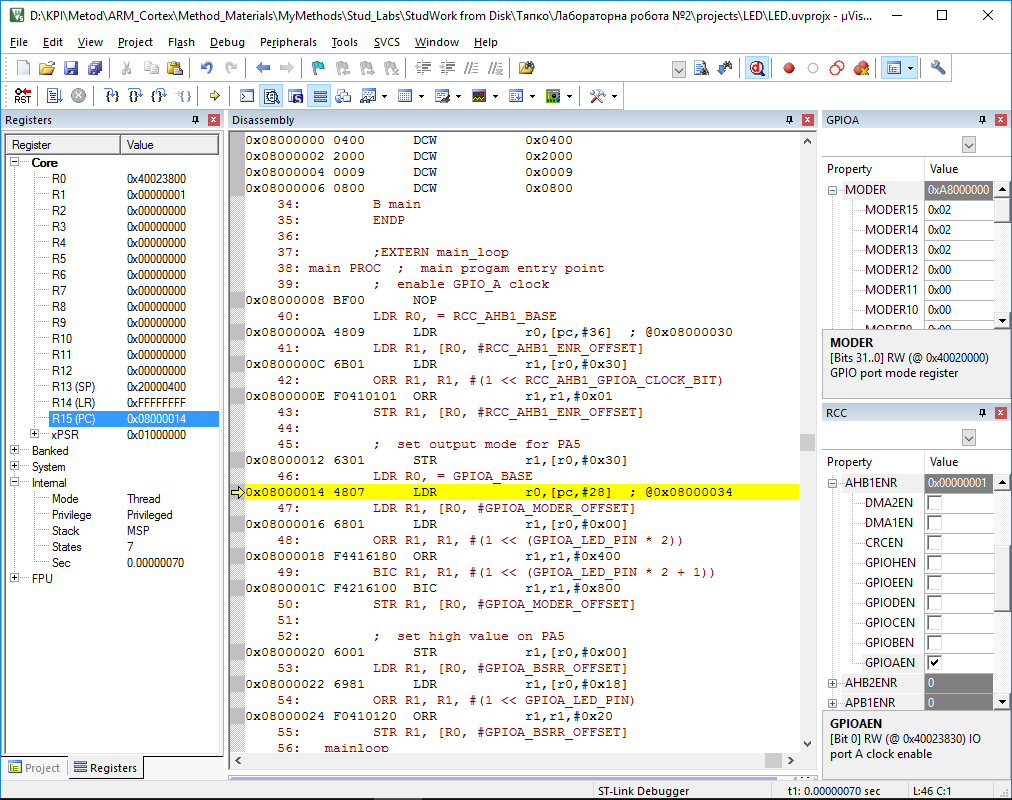


Рис. 16. Дизасембльований код

. Натискаючи клавішу «F11» будемо виконувати інструкції по черзі («крок за кроком»).

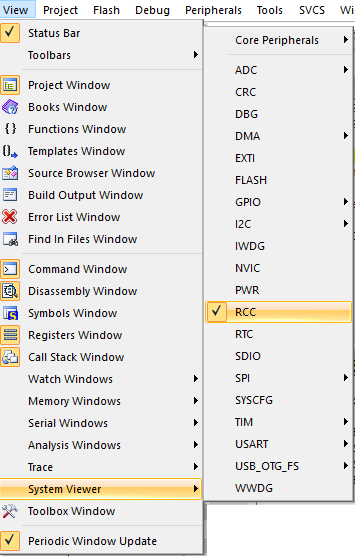
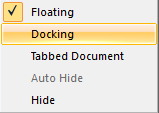
На рис. 17 у вікні Disassembly показано результат виконання програми після встановлення тактування порту GPIOA, тобто після встановлення в одиницю RCC\_AHB1ENR[0] = GPIOAEN командою STR за адресою 0х8000012.



Тактування порту А встановлено

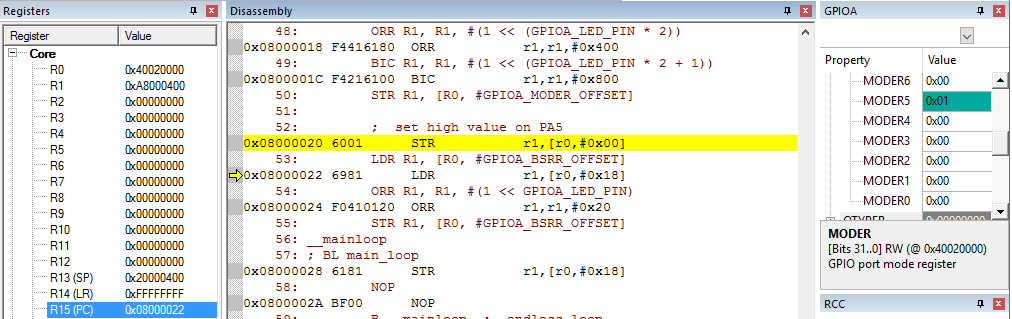
Початковий стан регістру  
GPIOA\_MODER

Рис. 17. Відлагодження асемблерної частини програми.  
Тактування порту GPIOA включено.

Для нагляду за станом регістрів RCC\_AHB1\_ENR і GPIO\_MODER командою View🡪System Viewer можна додати відповідні вікна. А маніпулюючи інструментами розміщення вікон, можна встановити їх у зручному місці вікна Debugger. На рис.17 вони встановлені так званим способом Docking, який задається з контекстного меню, що відкривається «правим кліком» мишки и по заголовку вікна.

Праворуч зверху (див. рис. 17) бачимо початкове налаштування пінив PA15, PA14, PA13 регістру GPIOA\_MODER. Поясніть стан цих виводів.

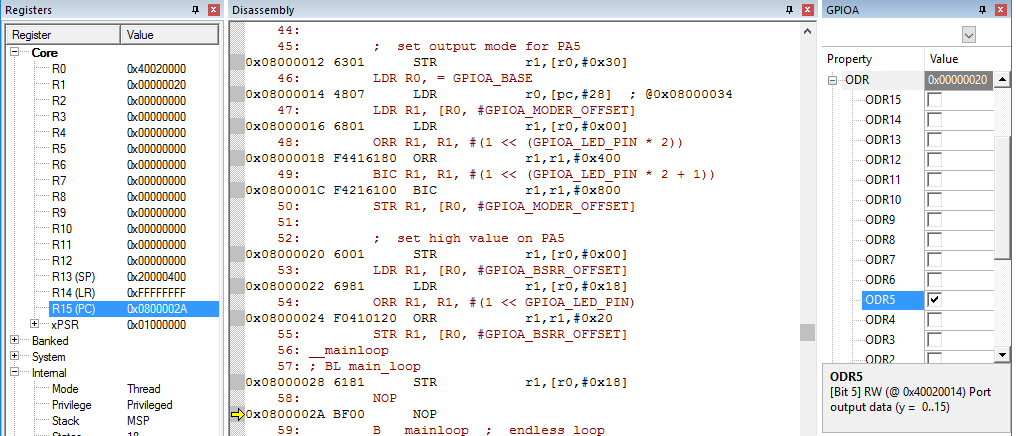
Виконайте фрагмент програми з налаштуванням виводу PA5 на «вивід» GPIOA\_MODER[11:10] = MODER5=01, тобто зупиніть виконання програми після виконання команди STR – на адресі 0х8000022 (рис. 18).



PA5 - налаштовано на вивід

Рис. 18. Порт PA5 налаштовано на вивід

На рис. 19 показано вікно дізасемблеру при завершенні всього фрагменту включення світлодіоду (встановлення високого рівню «1» на виході порту РА5).



На виводі PA5  
встановлено «1»

Рис. 19 - Завершення асемблерної частини програми

При цьому світлодіод LED2 засвітиться зеленим світлом.

**Зверніть увагу** на те, що, хоча ми маніпулювали битому регістру BSRR, стан відповідного біта регістра ODR також змінилося.

Якщо, виконуючи команди, дойти до частини коду, написаної на С, то у вікні Disassembly можна побачити багато «магічних» чисел не дуже зрозумілих людині, але зручних для виконання безпосередньо для мікроконтролера. Однак загалом цей код схожий на наш. Просто свої числові константи компілятор записав як частину програми, в такій ділянці пам’яті, до якої, за нормальних обставин, мікроконтролер не дійде, і не почне інтерпретувати їх як коди команд. Звертається компілятор до них завдяки зміщенню відносно регістру «PC» записаному у квадратних дужках в операції завантаження в регістр («LDR»).

Поглянемо на рис. 20 на ньому зображено першу інструкцію з функції «**main\_loop**» та результат її дизасемблювання. Видно, щось завантажується у регістр «R0» з пам’яті за адресою на 44 байти більшою ніж поточне значення регістру «PC». Також, як коментар, правіше, після знаку «собачка», написана вже порахована та адреса «0х08000068». Якщо в цьому ж вікні дизасемблеру перейти до цієї адреси (рис. 21), то можна побачити два подвійних слова, це наша константа «RCC\_AHB1\_ENABLE\_REG», яка мала значення адреси однойменного регістру в пам’яті мікроконтролеру, отже в регістрі «R0» тепер знаходиться ця адреса. Далі йде вже зчитування значення за адресою, що зберігається в регістрі «R0», тобто значення самого регістру «RCC\_AHB1\_ENABLE\_REG» і записування прочитаних даних у регістр «R0». Потім відбувається операція «логічного побітового І» для регістру «R0» з константою «1». Значення цієї константи отримане шляхом зсуву «1» на кількість бітів, записану в константу «RCC\_AHB1\_GPIOA\_CLOCK», тобто на «0» бітів. Тепер адреса регістру «RCC\_AHB1\_ENABLE\_REG» вже записується в регістр «R1», і останньою операцією значення з регістру «R0» записується за адресою, яка зберігається в регістрі «R1».

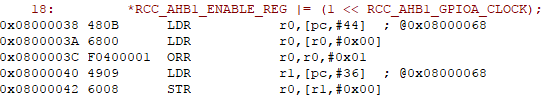


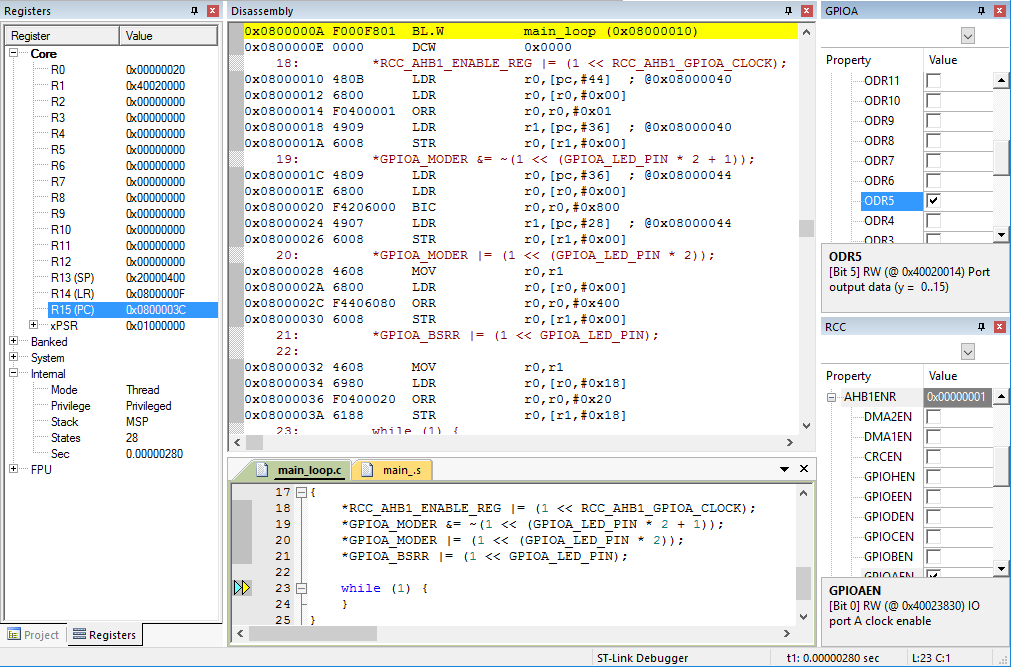
Рис. 20 – Дизасемблювання інструкції написаної на мові «С»



Рис. 21 - Числові константи, записані як частина програми

На рис. 22 надано результат дизасемблювання всієї процедури **main\_loop**.

По завершенні виконання всіх команд програми, побачимо, що стан регістрів GPIOA той же, що і у асемблерному варіанті програми. До речи, як що ви у вікні GPIOA відкриєте регістр BSRR, то виявите, що стан його бітів (у тому числі і BS5) не змінилося! Але ж біт ODR5 (PA5) очікувано встановлено у «1» і зелений світлодіод LED2 знову засвітився.



PA5 = 1

Рис. 22 - Результат дизасемблювання процедури **main\_loop**

Результат роботи програми зображений на рис. 23.

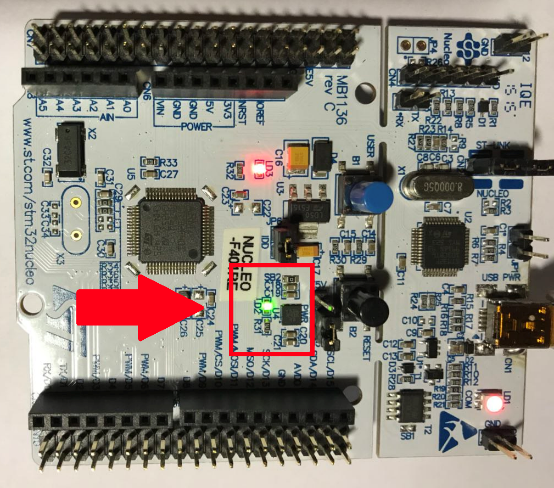


Рис. 23 - Результат роботи програми у NUCLEO

Сподіваюсь, що з набутими знаннями та за допомогою інструментів створення і відлагодження програм, які надає IDE «Keil µVision5» з MDK-ARM Keil, ви зможете самостійно створити подібні проекти та здійснити аналіз роботи відповідних програм, що складає суть аудиторного і індивідуального завдань.

**4.Завдання для виконання в аудиторії**

1. Створить вихідні файли main.s і main\_loop.c проекту для плати NUCLEO відповідно до цієї інструкції.
2. Виконайте всі етапи створення і відлагодження проекту.

**Індивідуальні завдання**

1. Переробіть вихідні файли main.s і main\_loop.c для проекту із платою Discovery. Варіанти:
   1. Увімкніть та здійсніть помітне для очей мерехтіння світлодіоду LED 3.
   2. Увімкніть та здійсніть помітне для очей мерехтіння світлодіоду LED 4
   3. Увімкніть та здійсніть помітне для очей мерехтіння світлодіоду діод LED 5
   4. Увімкніть та здійсніть помітне для очей мерехтіння світлодіоду LED 6
   5. Засвічуйте світлодіоди почергово із деякою помітною для очей затримкою.

**Примітка.** Затримку завдайте програмно.

1. Варіанти завдань для лабораторних макетів пристроїв на основі плати **NUCLEO/Discovery та стенду №1** (кнопки та світлодіоди).
   1. Здійсніть почергове увімкнення світлодіодів LED 1, Led 2, Led 3
   2. Увімкнення світлодіоду (LED №) відповідно до натиснутої кнопки
   3. Зміна послідовності увімкнення світлодіодів натисненням кнопки
   4. Зменшення/збільшення затримки горіння світлодіоду натисненням кнопок SW1 (”L”) та SW3 (“R”).  
      Кнопка SW2 (“OK”) підтверджує вибір значення величини затримки.

**Примітка.** Послідовність зміни тривалості затримки   
 зменшення “OK”🡪”L”{“L”… “L”}🡪”OK” ;  
 збільшення “OK”🡪”R”{“R”…”R”}🡪”OK”;  
 повернення значення затримки до початкового стану

1. Варіанти завдань для лабораторних макетів пристроїв на основі плати **NUCLEO/Discovery та стенду №2** (кнопки та семісегментні індикатори).

Більш детально варіанти завдань див. у файлі «Теми і варіанти робіт.docx»

**Сворить звіт з виконання «аудіторного» і індівідуального завдань.**

1. Додайте у звіт з виконання роботи «скріншоти» з основних етапів відлагодження проектів і ваші коментарі.

**5.Список літератури**

1. Лекція 7. GPIO/- презентація на Google Disk: \\Lessons\ Less 7 GPIO
2. User\_manual\_Nucleo(UM1724).pdf ̶ Руководство пользователя по плате Nucleo   
   / ̶ документ на Google Disk: \\Matherials\Documentations\STM32 NUCLEO
3. Reference\_manual\_STMF401(RM0368).pdf ̶ Руководство по STM32F401   
   / ̶ документ на Google Disk: \\Matherials\Documentations\STM32 NUCLEO  
   [Електронний ресурс] Режим доступа:  
   <http://www.st.com/web/en/resource/technical/document/reference_manual/DM00096844.pdf>
4. Discovery\_User\_manual(UM1472 ).pdf ̶ Руководство пользователя по плате Discovery / ̶ документ на Google Disk:\\Matherials\Documentations\STM32 Discovery
5. Reference Manual \_STM405-07-43xxx (RM0090) ̶ Руководство по STM32F407   
   / ̶ документ на Google Disk:\\Matherials\Documentations\STM32 Discovery  
   [Електронний ресурс] Режим доступа:  
   [www.st.com/resource/en/reference\_manual/dm00031020.pdf](http://www.st.com/resource/en/reference_manual/dm00031020.pdf)
6. Programming\_manual\_STM32F3, STM32F4 and STM32L4 (PM0214).pdf  
    ̶ Руководство по программированию МК STM32хх / ̶ документ на Google Disk:\\Matherials\Documentations\Cortex-M4  
   [Електронний ресурс] Режим доступа:  
   [www.st.com/resource/en/programming\_manual/dm00046982.pdf](http://www.st.com/resource/en/programming_manual/dm00046982.pdf)
7. ARMv7-M\_ARM Architecture (RM 0403).pdf – Описание архитектуры ARMv7-M  
   / ̶ документ на Google Disk:\\Matherials\Documentations\Cortex-M4
8. Лабораторна робота №1. Ваш перший проект для STM32.  
   / ̶ Google Disk:\\ 1 семестр 2017\Labs\Lab1
9. Начинаем изучать Cortex-M на примере STM32 (Часть 1)   
   /[Електронний ресурс] Режим доступа: <https://habrahabr.ru/post/216843/>
10. Олег Вальпа «Современные 32-разрядные ARM-микроконтроллеры серии STM32: порты общего назначения GPIO» / CHIP NEWS Украина, #9 (129), ноябрь, 2013, с. 72 – 75.

1. Далі буде йти про обидві плати NUCLEO і Discovery. Аналогічні дані будемо подавати у тексті через «/». [↑](#footnote-ref-1)
2. Ця дія не є обов’язковою, оскілки ім’я підпрограми main\_loop , що викликається у рядку 57 визначено директивою EXTERN у рідку 37 як зовнішнє, тобто таке, що завдане у іншому файлі. Тому асемблер не виявить синтаксичної помилки типу «не описане ім’я» [↑](#footnote-ref-2)
3. Назва файлу не обов'язково має збігатися з ім'ям функції, що описано у ньому. [↑](#footnote-ref-3)