**Лабораторна робота №1**

**Тема:** Створення мінімального програмного проекту на мові асемблера.

**Мета:** Створити мінімальний проект, перевірити виконання відлагоджувачем.

**1 Порядок виконання**

**1)** Створіть файл **start.S** у директорії проекту(створіть директорію проекту).

Цей файл потрібен для зберігання таблиці векторів виключень (про неї детальніше буде в лабораторній роботі №3) та для мітки **hard\_reset** з якої починається виконання програми.

Зверніть увагу на розширення файлу, воно має значення для компілятора:

|  |
| --- |
| .syntax unified  .cpu cortex-m4  //.fpu softvfp  .thumb  // Global memory locations.  .global vtable  .global reset\_handler  /\*  \* vector table  \*/  .type vtable, %object  **vtable:**  .word \_\_stack\_start  .word \_\_hard\_reset\_\_+1  .size vtable, .-vtable  **\_\_hard\_reset\_\_:**  ldr r0, =\_\_stack\_start  mov sp, r0  b \_\_hard\_reset\_\_ |

**2)** Створіть файл **lscript.ld**. Це скрипт лінкера, який вказує скільки пам'яті може використовувати програма:

|  |
| --- |
| /\* linker script for stm32f1  \* platforms  \* Define the end of RAM and limit of stack memory  \*/  MEMORY  {  /\* We mark flash memory as read-only, since that is where the program lives. STM32 chips map their flash memory to start at 0x08000000, and we have 32KB of flash memory available. \*/  FLASH ( rx ) : ORIGIN = 0x08000000, LENGTH = 1M  /\* We mark the RAM as read/write, and as mentioned above it is 4KB long starting at address 0x20000000. \*/  RAM ( rxw ) : ORIGIN = 0x20000000, LENGTH = 128K  }  \_\_stack\_start = ORIGIN(RAM) + LENGTH(RAM); /\* Start of the stack address \*/ |

**3)** Зберіть проект, для цього виконайте:

>>> arm-none-eabi-gcc -x assembler-with-cpp -c -O0 -g3 -mcpu=cortex-m4 -mthumb -Wall start.S -o start.o

>>> arm-none-eabi-gcc start.o -mcpu=cortex-m4 -mthumb -Wall --specs=nosys.specs -nostdlib -lgcc -T./lscript.ld -o firmware.elf

>>> arm-none-eabi-objcopy -O binary -F elf32-littlearm firmware.elf firmware.bin

На виході маємо бінарний файл.

**4)** Після цього можна виконати цей код в qemu, використавши відлагоджувач **gdb**.

Записати в **PATH** де знаходиться qemu, зверніть увагу, це потрібно робити кожного разу коли запускаєте консоль, тому краще всього додати наступний рядок в, наприклад, **/home/user/.bashrc** файл:

PATH=$PATH:~/opt/xPacks/qemu-arm/2.8.0-7/bin/

Виконати:

>>> qemu-system-gnuarmeclipse --verbose --verbose --board STM32F4-Discovery --mcu STM32F407VG -d unimp,guest\_errors --image firmware.bin --semihosting-config enable=on,target=native -s -S

З флагами **-s -S** qemu очікує підключення зовнішнього відлагоджувального ПО з портом **tcp::1234**. Відкриваємо нове вікно терміналу та виконуємо (в залежності від того, який тулчейн із “вступу” було інстальовано):

>>> gdb-multiarch firmware.elf

**АБО**

>>> arm-none-eabi-gdb firmware.elf

В разі успіху:

For help, type "help".

Type "apropos word" to search for commands related to "word"...

Reading symbols from firmware.elf...

(gdb)

Вводимо **target extended-remote:1234**. Тепер програма готова для відлагодження.

Далі програма очікує введення команд **gdb**. Для того щоб виконати по кроково вводимо:

>>> step  
Далі натискаємо Enter та виконуємо програму.

**5)** Тепер можна автоматизувати створення прошивки.

Створіть GNU Makefile**:**

|  |
| --- |
| SDK\_PREFIX?=arm-none-eabi-  CC = $(SDK\_PREFIX)gcc  LD = $(SDK\_PREFIX)ld  SIZE = $(SDK\_PREFIX)size  OBJCOPY = $(SDK\_PREFIX)objcopy  QEMU = qemu-system-gnuarmeclipse  BOARD ?= STM32F4-Discovery  MCU=STM32F407VG  TARGET=firmware  CPU\_CC=cortex-m4  TCP\_ADDR=1234  deps = \  start.S \  lscript.ld  all: target  target:  $(CC) -x assembler-with-cpp -c -O0 -g3 -mcpu=$(CPU\_CC) -Wall start.S -o start.o  $(CC) start.o -mcpu=$(CPU\_CC) -Wall --specs=nosys.specs -nostdlib -lgcc -T./lscript.ld -o $(TARGET).elf  $(OBJCOPY) -O binary -F elf32-littlearm $(TARGET).elf $(TARGET).bin  qemu:  $(QEMU) --verbose --verbose --board $(BOARD) --mcu $(MCU) -d unimp,guest\_errors --image $(TARGET).bin --semihosting-config enable=on,target=native -gdb tcp::$(TCP\_ADDR) -S  clean:  -rm \*.o  -rm \*.elf  -rm \*.bin |

Команда **make** створить прошивку, команда **make qemu** запустить емулятор з вищезазначеними налаштуваннями.

**3.2 Додаткові матеріали**

1. Документація на qemu: <https://gnu-mcu-eclipse.github.io/qemu/options/>
2. Команди для відлагоджування: <https://web.stanford.edu/class/archive/cs/cs107/cs107.1196/resources/gdb>

<https://linux.die.net/man/1/gdb>

<http://users.ece.utexas.edu/~adnan/gdb-refcard.pdf>