Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Архітектура комп'ютерів 3. Мікропроцесорні системи

Лабораторна робота 2

«Основні інструкції 32-бітного ARM процесора для мікроконтролерів»

Виконав:
студент групи ІО-23
Корбут М. Я.
Залікова книжка №2313
Перевірив
Каплунов А.В.

Київ - 2025

Лабораторна робота №2

Тема: «Основні інструкції 32-бітного ARM процесора для мікроконтролерів»

Мета: Навчитися використовувати асемблерні інструкції ядра Cortex-M4, працювати з процедурами і базово зрозуміти архітектуру ядра. Навчитися розуміти синтаксис мови асемблера GAS (GNU Assembly), що є частиною стандартного пакету тулчейну GCC (GNU Compiler Collection) для арм (arm-none-eabi-). Навчитися працювати з GDB відлагоджувачем.

Хід роботи:

1. **Підготовка**
   Для цієї лабораторної роботи я вирішив розробити власну функцію для обчислення. В результаті вийшла ось така функція:
   {a*b/c, c>0; (a+b) << 3, c = 0; a & b + c, c < 0}

2. **Створення файлу start.S**
   У створеному каталозі проєкту був створений файл start.S, який містить таблицю векторів виключень і мітку __hard_reset__:

```
.syntax unified
.cpu cortex-m4
//.fpu softvfp
.thumb
// Global memory locations.
.global vtable
.global __hard_reset__
/*
* vector table
*/
.type vtable, %object
.type __hard_reset__, %function
vtable:
    .word __stack_start
    .word __hard_reset__+1
    .size vtable, .-vtable
__hard_reset__:
    // initialize stack here
    // if not initialized yet

    bl lab2
    _loop: b _loop
    .size __hard_reset__, .-__hard_reset__
```

3. Створення скрипта лінкування lscript.ld

Скрипт визначає розміщення пам’яті:

```
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);
```

4. Основний файл Lab2.S:

Далі було створено основний скрипт Lab2.S, який записує операнди у регістри r0, r1 та r2, після чого порівнює r2 з 0 і перестрибує на розрахунок потрібної функції в залежності від результату:

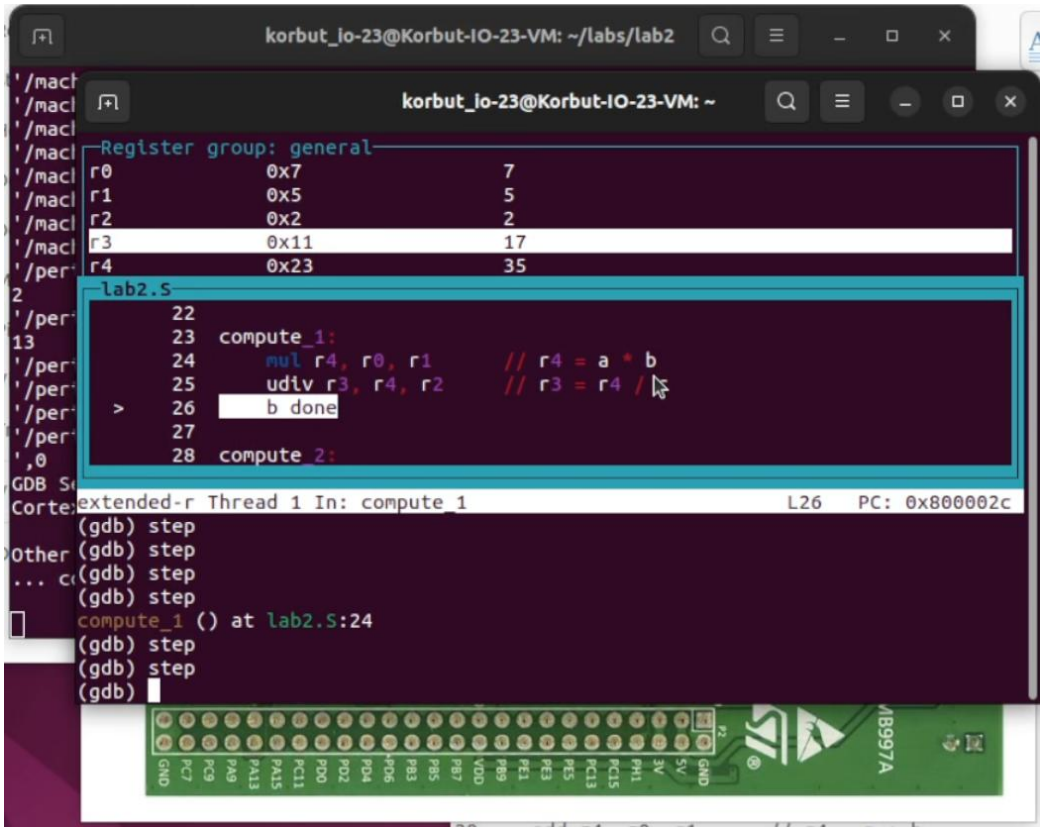
```
.global lab2
.syntax unified

// Define a, b, c
#define A #7
#define B #5
#define C #-3

// {a * b / c, c > 0; ((a + b) << 3), c = 0; a & b + c,  c < 0}

lab2:
    push {lr}   // Save registers and return address

    mov r0, A
    mov r1, B
    mov r2, C

    cmp r2, #0
    bgt compute_1  // if c > 0
    beq compute_2   // if c == 0
    blt compute_3  // if c < 0

compute_1:
    mul r4, r0, r1      // r4 = a * b
    udiv r3, r4, r2     // r3 = r4 / c
    b done

compute_2:
```

```
    add r4, r0, r1      // r4 = a + b
    lsl r3, r4, #3      // r3 = r4 << 3
    b done

compute_3:
    and r4, r0, r1      // r4 = a & b
    add r3, r4, r2      // r3 = r4 + c
    b done

done:
    pop {pc}    // Return
```

5. **Запуск та відлагодження:**
   Аналогічно до попередньої роботи, буро використано Makefile для автоматичної збірки проекту та Qemu для емуляції, а стан регістрів було відображено за допомогою команди layout regs:

```
qemu-make
make qemu
arm-none-eabi-gdb firmware.elf
(gdb) target extended-remote :1234
(gdb) layout regs
(gdb) step
```

Скріншот роботи програми:



Репозиторій

Код було завантажено до репозиторію GitHub. Переглянути його можна за посиланням.

Висновки:

У результаті виконання лабораторної роботи:

- Ознайомився з базовими арифметичними, логічними та умовними інструкціями ARM.
- Розробив програму з використанням умовного переходу та арифметичних обчислень.
- Навчився створювати власні асемблерні функції.
- Відпрацьовано налагодження програми з допомогою gdb, перегляд вмісту регістрів.
- Оформлено збірку та запуск через Makefile.