Міністерство освіти і науки України
Національний технічний університет Укараїни
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА №4
з дисципліни “Архітектура комп’ютерів 3”
на тему “РОЗРОБЛЕННЯ ПРОГРАМ ОБРОБКИ ДАНИХ ДЛЯ ПРОЦЕСОРНОГО ЯДРА CORTEX M4”

Виконала:
студентка III курсу ФІОТ
групи ІВ-81
Дьяченко Тетяна

Перевірив:
Нікольський С. С.

Київ – 2021

Мета: Вивчення архітектурних особливостей, системи команд, принципів організації команд умовних та безумовних переходів та переходів на підпрограми, команд роботи з пам'яттю та способів адресації операндів.

Завдання

Номер залікової книжки: 8113

$8113_{10} = 01\ 1111\ 10\mathbf{11\ 0001}_2$

h_4	h_3	h_2	h_1	**Функція**
0	0	0	1	$F=8(X1 \vee X2)+(X3-1-X4)/16$

h_2	h_1	$X1$	$X2$	$X3$	$X4$
0	1	12	2	–10	15

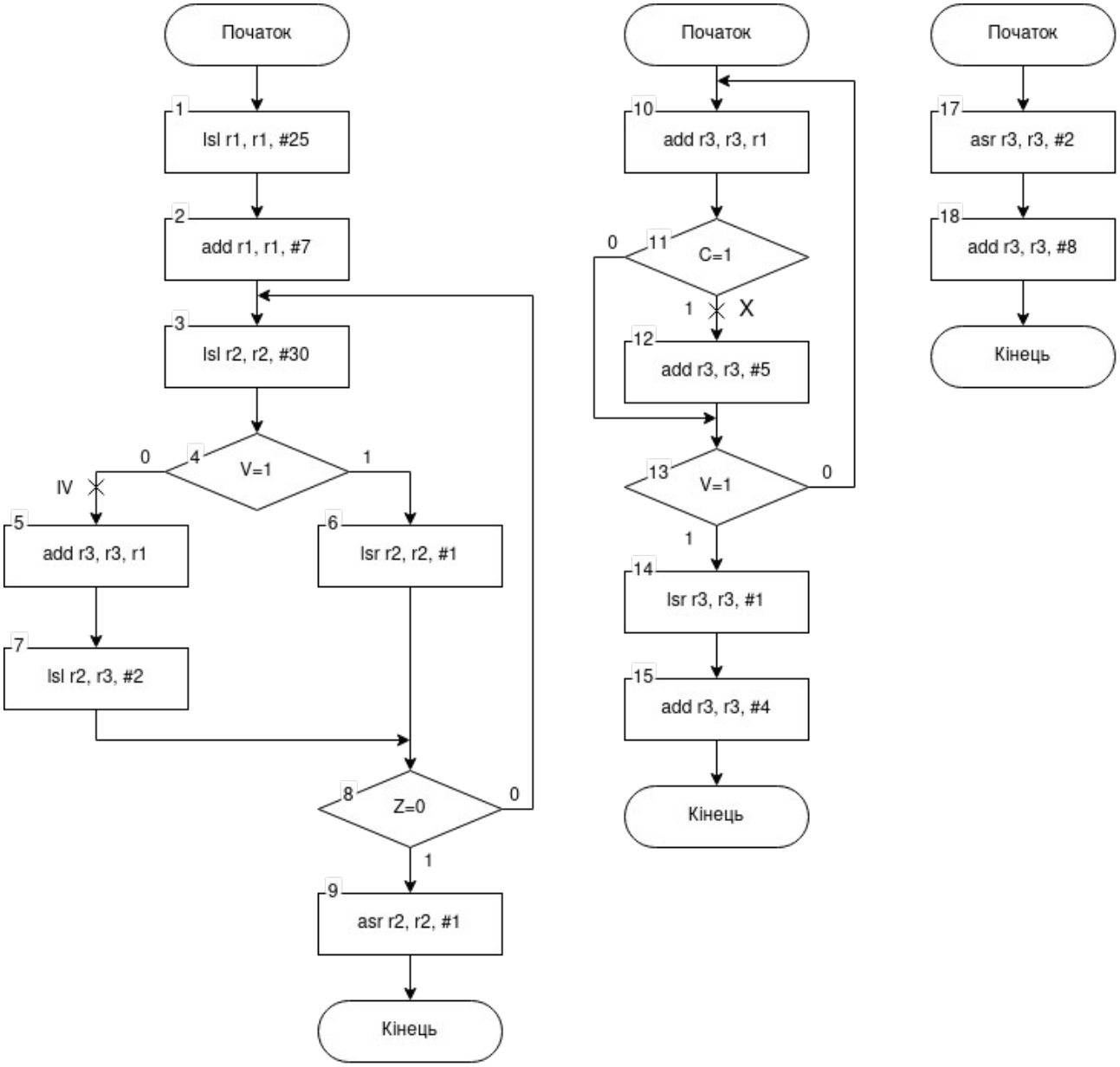
h_4	h_5	h_1	Номер точки переходу на підпрограму
0	1	1	IV

h_2	h_1	Номер точки переходу на підпрограму
0	1	X

h_1	h_3	Умови переходу (ознаки результату виконання операції)		
		***CD*1**	***CD*2**	***CD*3**
1	0	V=1	C=1	Z=1

Виконання роботи

Алгоритм для другого завдання



start.S

```
.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 task1
    bl task2
    _loop: b _loop
    .size __hard_reset__, .-__hard_reset__
```

task1.S

```
.global task1
.syntax unified

#define x1h #0x00000000
#define x1l #0x0000000C
#define x2h #0x00000000
#define x2l #0x00000002
#define x3h #0xFFFFFFFF
#define x3l #0xFFFFFFF6
#define x4h #0x00000000
#define x4l #0x0000000F
#define ZK #0x00001FB1

task1:
        push {lr}
        mov r1, x1h
        mov r0, x1l
        mov r3, x2h
        mov r2, x2l
        //x1 or x2
        orr r4, r2, r0
        orr r5, r3, r1
        //8(x1 or x2) r4, r5
        eor r10, r10
```

```
lsls r4, r4, #1
adc r10, r10, #0
lsls r4, r4, #1
lsl r10, r10, #1
adc r10, r10, #0
lsls r4, r4, #1
lsl r10, r10, #1
adc r4, r4, r10
lsls r5, r5, #3
add r5, r5, r10

mov r1, x3h
mov r0, x3l
mov r3, x4h
mov r2, x4l
//f = (x3 - 1 - x4) r6, r7
subs r6, r0, #1
sbc r7, r1, #0
subs r6, r6, r2
sbc r7, r7, r3
//f = (x3 - 1 - x4)/16 r6, r7
eor r10, r10
asrs r7, r7, #1
adc r10, r10, #0
asrs r7, r7, #1
lsl r10, r10, #1
adc r10, r10, #0
asrs r7, r7, #1
lsl r10, r10, #1
adc r10, r10, #0
asrs r7, r7, #1
lsl r10, r10, #1
adc r10, r10, #0
lsl r10, r10, #28
lsr r6, r6, #4
add r6, r6, r10
//f = 8(x1 or x2) + (x3 - 1 - x4)/16
adds r6, r6, r4
adcs r7, r7, r5

it vs
blVS _correct

cmp r6, #0
it eq
```

```
        blEQ _addZK

        mov r0, ZK
        add r7, r0

        lsrs r0, r7, #30
        it eq
        blEQ _cont
        cmp r0, #3
        it eq
        blEQ _cont
        bl _correct2

_cont:
        pop {pc}

_correct:
        push {lr}
        lsr r6, r6, #1
        lsrs r7, r7, #1
        adc r6, r6, #0
        add r7, r7, #0x80000000
        pop {pc}

_addZK:
        push {lr}
        mov r0, ZK
        lsl r0, r0, #16
        mov r6, r0
        pop {pc}

_correct2:
        push {lr}
        lsr r6, r6, #1
        asrs r7, r7, #1
        it cs
        addCS r6, #0x80000000
        pop {pc}
```

task2.S

```
  .global task2
.syntax unified
```

```
#define a #5
#define b #8

task2:
	push {lr}

	mov r1, a
	mov r2, b

	//program start
	//step1
	lsl r1, r1, #25
	//step2
	add r1, r1, #7
_step3:
	lsls r2, r2, #30

	//step4
	it vs
	//step6
	lsrVS r2, r2, #1
	bVS _step8

	//else - go to sub-program
	bl subP1
	//step5
	add r3, r3, r1
	//step7
	lsls r2, r3, #2

_step8:
	it eq
	blEQ _step3

	//step9
	asr r2, r2, #1

	pop {pc}

subP1:
	push {lr}
	mov r3, #3

_step10:
	adds r3, r3, r1
```

```
        //step11
        it cc
        blCC _step13

        bl subP2
        //step12
        add r3, r3, #5

_step13:
        it vc
        blVC _step10

        //step14
        lsr r3, r3, #1
        //step15
        add r3, r3, #4


        pop {pc}

subP2:
        push {lr}

        //step17
        asr r3, r3, #2
        //step18
        add r3, r3, #8

        pop {pc}
```

lscript.ld

```
MEMORY
{

FLASH ( rx )
: ORIGIN = 0x08000000, LENGTH = 1M


RAM ( rxw )
: ORIGIN = 0x20000000, LENGTH = 128K
}
__stack_start = ORIGIN(RAM) + LENGTH(RAM);
```

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
	$(CC) -x assembler-with-cpp -c -O0 -g3 -mcpu=$(CPU_CC) -Wall task1.S -o task1.o
	$(CC) -x assembler-with-cpp -c -O0 -g3 -mcpu=$(CPU_CC) -Wall task2.S -o task2.o
	$(CC) start.o task1.o task2.o -mcpu=$(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

flash:
	st-flash write $(TARGET).bin 0x08000000
```

Результати виконання

task 1

$F = 8(12 \vee 2) + (-10 - 1 - 15)/16$ = 8*14 + (-26)/16 = 112 - 2 = 110

Результат у регістрах r6 — молодші розряди і r7 — старші розряди

```
r6             0x6e                110
r7             0x0                 0
r8             0x0                 0
r9             0x0                 0
r10            0xf0000000          -268435456
r11            0x0                 0
r12            0x0                 0
sp             0x2001fffc          0x2001fffc
lr             0x800000d           134217741
pc             0x80000a0           0x80000a0 <task1+142>
```

```
   0x8000076 <task1+100>   mov.w   r10, r10, lsl #1
   0x800007a <task1+104>   adcvs.w r10, r10, #0
   0x800007e <task1+108>   asrs    r7, r7, #1
   0x8000080 <task1+110>   mov.w   r10, r10, lsl #1
   0x8000084 <task1+114>   adc.w   r10, r10, #0
   0x8000088 <task1+118>   asrs    r7, r7, #1
   0x800008a <task1+120>   mov.w   r10, r10, lsl #1
   0x800008e <task1+124>   adc.w   r10, r10, #0
   0x8000092 <task1+128>   mov.w   r10, r10, lsl #28
   0x8000096 <task1+132>   mov.w   r6, r6, lsr #4
   0x800009a <task1+136>   add     r6, r10
   0x800009c <task1+138>   adds    r6, r6, r4
   0x800009e <task1+140>   adcs    r7, r5
  >0x80000a0 <task1+142>   it      vs
   0x80000a2 <task1+144>   blvs    0x80000ca <_correct>
   0x80000a6 <task1+148>   cmp     r6, #0
   0x80000a8 <task1+150>   it      eq
```

```
extended-r Thread 1 In: task1
(gdb) step
task1 () at task1.S:15
(gdb)
```

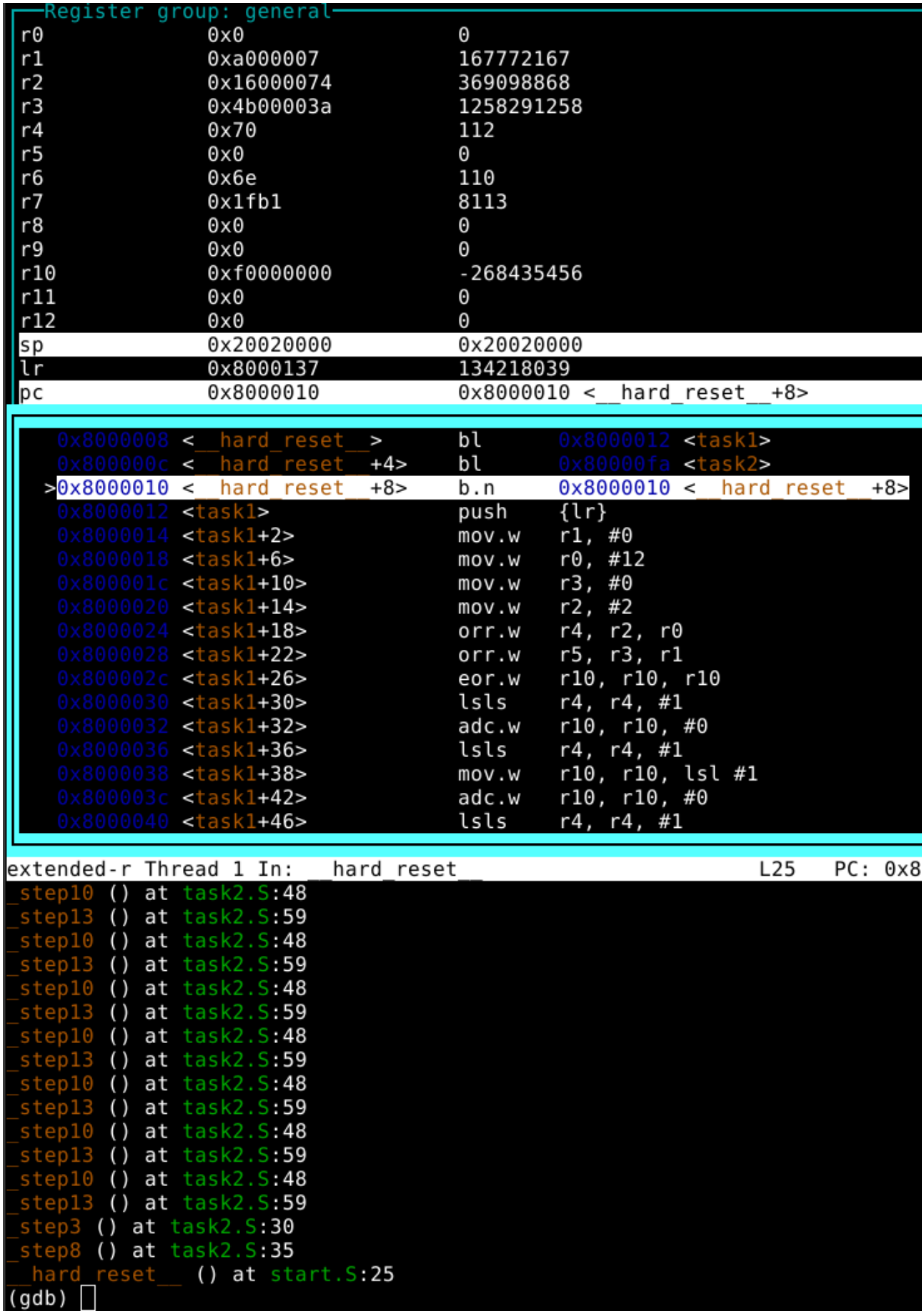
task 2

r1 = 5, r2 = 8

1) r1 = lsl(r1, 25) = 0x0A000000
2) r1 = r1 + 7 = 0x0A000007
3) r2 = lsl(r2, 30) = 0
4) V = 1

   r3 = 3

10) r3 = r3 + r2 = 0x0A00000A
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x14000011
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x1E000018
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x2800001F
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x32000026
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x3C00002D
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x46000034
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x5000003B
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x5A000042
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x64000049
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x6E000050
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x78000057
11) C=0
13) V=0
10) r3 = r3 + r2 = 0x8200005E
11) C=0
13) V=1
14) r3 = lsr(r3, 1) = 0x4100002F
15) r3 = r3 + 4 = 0x41000033
5) r3 = r3 + r1 = 0x4B00003A
7) r2 = lsl(r3, 2) = 0x2C0000E8
8) Z = 0
9) r2 = asr(r2, 1) = 0x16000074


Register group: general
r0 0x0 0
r1 0xa000007 167772167
r2 0x16000074 369098868
r3 0x4b00003a 1258291258
r4 0x70 112
r5 0x0 0
r6 0x6e 110
r7 0x1fb1 8113
r8 0x0 0
r9 0x0 0
r10 0xf0000000 -268435456
r11 0x0 0
r12 0x0 0
sp 0x20020000 0x20020000
lr 0x8000137 134218039
pc 0x8000010 0x8000010 <__hard_reset__+8>
0x8000008 <__hard_reset__> bl 0x8000012 <task1>
0x800000c <__hard_reset__+4> bl 0x80000fa <task2>
>0x8000010 <__hard_reset__+8> b.n 0x8000010 <__hard_reset__+8>
0x8000012 <task1> push {lr}
0x8000014 <task1+2> mov.w r1, #0
0x8000018 <task1+6> mov.w r0, #12
0x800001c <task1+10> mov.w r3, #0
0x8000020 <task1+14> mov.w r2, #2
0x8000024 <task1+18> orr.w r4, r2, r0
0x8000028 <task1+22> orr.w r5, r3, r1
0x800002c <task1+26> eor.w r10, r10, r10
0x8000030 <task1+30> lsls r4, r4, #1
0x8000032 <task1+32> adc.w r10, r10, #0
0x8000036 <task1+36> lsls r4, r4, #1
0x8000038 <task1+38> mov.w r10, r10, lsl #1
0x800003c <task1+42> adc.w r10, r10, #0
0x8000040 <task1+46> lsls r4, r4, #1
extended-r Thread 1 In: __hard_reset__ L25 PC: 0x8
_step10 () at task2.S:48
_step13 () at task2.S:59
_step10 () at task2.S:48
_step13 () at task2.S:59
_step10 () at task2.S:48
_step13 () at task2.S:59
_step10 () at task2.S:48
_step13 () at task2.S:59
_step10 () at task2.S:48
_step13 () at task2.S:59
_step10 () at task2.S:48
_step13 () at task2.S:59
_step10 () at task2.S:48
_step13 () at task2.S:59
_step3 () at task2.S:30
_step8 () at task2.S:35
__hard_reset__ () at start.S:25
(gdb)

Висновок

В ході виконання лабораторної роботи було вивчено архітектурні особливості, системи команд, принципів організації команд умовних та безумовних переходів та переходів на підпрограми, команд роботи з пам’яттю та способів адресації операндів для Cortex-M4. Також було створено проект на асемблері, що розрахунок по формулі з операндами подвійної довжини, а також іншу підпрограму, що демонструє умовні та безумовні переходи. Виконання проекту було перевірено за допомогою відлагоджувача gdb. Також було створено Makefile для автоматизації створення прошивки.