

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота № 4
з дисципліни
“Архітектура комп’ютерів – 3”

Виконала:
студентка групи ІВ-81
ЗК ІВ-8101
Базова Лідія

Київ 2021

Мета роботи: 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.

Варіант: $8101_{10} = 11\ 1010\ 0101_2$

h_4	h_3	h_2	h_1	Функція*
0	1	0	1	$F = 8(X_1 - X_2) + (X_3 \oplus X_4 - 1) / 16$

h_2	h_1	X_1	X_2	X_3	X_4
0	1	12	2	-10	15

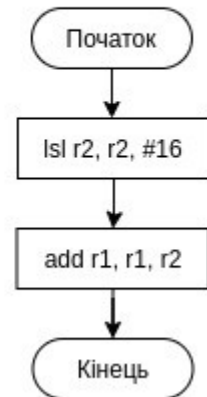
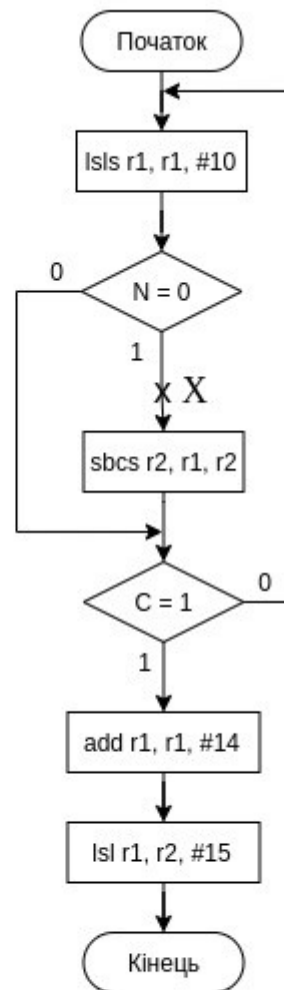
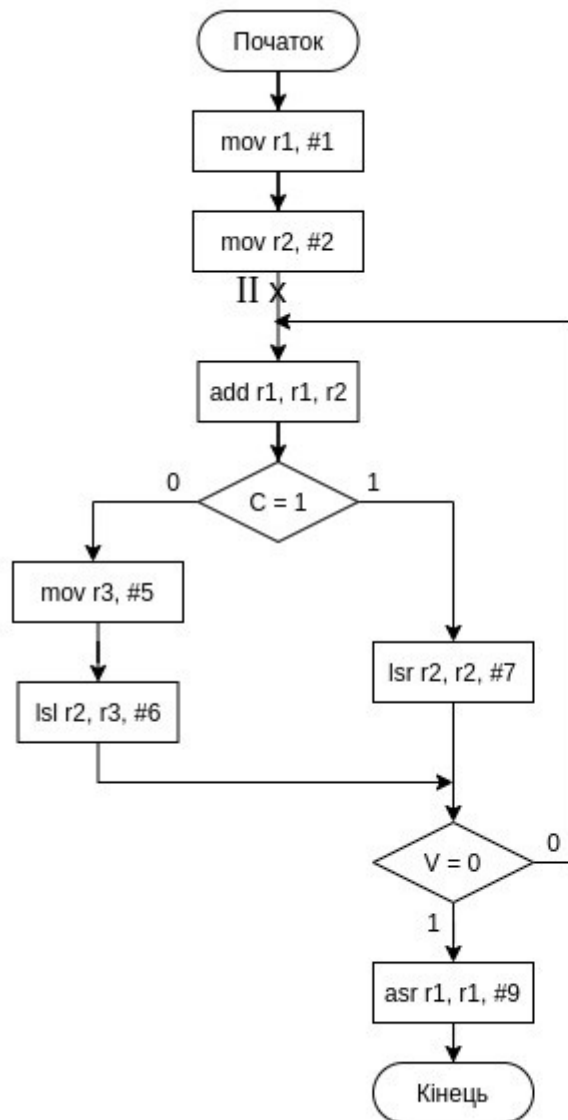
h_4	h_5	h_1	Номер точки переходу на підпрограму
0	0	1	II

h_2	h_1	Номер точки переходу на підпрограму
0	1	X

		Ознаки результату/умови переходу		
h_1	h_3	LC1	LC2	PA/C
1	1	C=1	N=0	V=0

Виконання роботи:

Вихідні алгоритми:



Лістинг коду

1. start.S

```
.syntax unified
```

```
.cpu cortex-m4
```

```
.thumb
```

```
.global vtable
```

```
.global __hard_reset__
```

```
.type vtable, %object
```

```
.type __hard_reset__, %function
```

```
vtable:
```

```

.word __stack_start
.word __hard_reset__+1
.size vtable,.-vtable
__hard_reset__:
    bl part1
    bl part2
    _loop: b _loop
.size __hard_reset__,.-__hard_reset__

```

2. part1.S

```

.global part1
.syntax unified

```

```

#define x1u #0x00000000
#define x1d #0x0000000C
#define x2u #0x00000000
#define x2d #0x00000002
#define x3u #0xFFFFFFFF
#define x3d #0xFFFFFFFF6
#define x4u #0x00000000
#define x4d #0x0000000F
#define ZK #0x00001FA5

```

```

part1:
    push {lr}
    // calculate
        mov r1, x1u
        mov r0, x1d
        mov r3, x2u
        mov r2, x2d

        //x1 - x2
        subs r4, r0, r2
        sbc r5, r1, r3
        //8(x1 - x2)
        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, x3u
mov r0, x3d
mov r3, x4u
mov r2, x4d
```

```
//x3  $\oplus$  x4 - 1
subs r6, r2, #1
sbc r7, r3, #0
eors r6, r0, r6
eors r7, r1, r7
//(x3  $\oplus$  x4 - 1)/16
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 - x2) + (x3 ⊕ x4 - 1)/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}
```

3. part2.S

.global part2

.syntax unified

part2:

```
    push {lr}

    // calculate
    mov r1, #1
    mov r2, #2
    //go to sub
    bl subPr1
```

_step3:

```
    adds r1, r1, r2

    it cc
    //C = 1
    lsrCC r2, r2, #7
    bCC _step8
```

```
//C != 1
mov r3, #5
lsls r2, r3, #6
```

```
_step8:
    //if (V != 0 -> V = 1)
    it vs
    blVS _step3

    //if V = 0
    asr r1, r1, #9
    pop {pc}
```

```
subPr1:
    push {lr}
_step1_1:
    lsls r1, r1, #10
    //if(N != 0 -> N = 1)
    it mi
    blmi _step1_4
    //else go to subpr2
    bl subPr2
    sbcs r2, r2, r1
```

```
_step1_4:
    it cc
    blCC _step1_1

    add r1, r1, #14
    lsl r1, r2, #15

    pop {pc}
```


subPr2:

push {lr}

lsl r2, r2, #16

adds r1, r1, r2

pop {pc}

4. lscript.ld

MEMORY

{

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

RAM (rxw) : ORIGIN = 0x20000000, LENGTH = 128K

}

__stack_start = ORIGIN(RAM) + LENGTH(RAM);

5. 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 part1.S -o part1.o
$(CC) -x assembler-with-cpp -c -O0 -g3 -mcpu=$(CPU_CC) -Wall part2.S -o part2.o
$(CC) start.o part1.o part2.o -mcpu=$(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
```

Скріншоти роботи:

частина 1. Результат на r7..r6

```
Register group: general
r0      0xfffffffff6      -10
r1      0xfffffffff      -1
r2      0xf              15
r3      0x0              0
r4      0x50             80
r5      0x0              0
r6      0x4f             79
r7      0x0              0

0x800009a <part1+136> adcs    r7, r5
0x800009c <part1+138> it      vs
0x800009e <part1+140> blvs   0x80000c6 <_correct>
>0x80000a2 <part1+144> cmp    r6, #0
0x80000a4 <part1+146> it      eq
0x80000a6 <part1+148> bleq   0x80000d8 <_addZK>
0x80000aa <part1+152> movw   r0, #8101      ; 0x1fa5
0x80000ae <part1+156> add     r7, r0
0x80000b0 <part1+158> lsrs   r0, r7, #30

extended-r Thread 1 In: part1      L72      PC: 0x80000a2
(gdb) step
part1 () at part1.S:15
(gdb)
```

Після корекції:

r6	0x4f	r5	79
r7	0x1fa5	r6	8101

Перевірка.

$x1 = 12$; $x2 = 2$; $x3 = -10$; $x4 = 15$

$$F = 8(x_1 - x_2) + (x_3 \oplus x_4 - 1) / 16 = 8 * (12 - 2) + (-10 \oplus 15 - 1) / 16 =$$

$$= 8 * 10 + (-10 \oplus 14) / 16 = 80 + (-8) / 16 = 79.5$$

Після другої частини:

```
Register group: general
r0      0x0      0
r1      0x3fdf7f  4185983
r2      0x140    320
r3      0x5      5
r4      0x50     80
r5      0x0      0
r6      0x4f     79
r7      0x1fa5   8101
r8      0x0      0
r9      0x0      0

0x8000008 <__hard_reset__> bl      0x8000012 <part1>
0x800000c <__hard_reset__+4> bl      0x80000f6 <part2>
>0x8000010 <__hard_reset__+8> b.n     0x8000010 <__hard_reset__+8>
0x8000012 <part1>          push     {lr}
0x8000014 <part1+2>        mov.w    r1, #0
0x8000018 <part1+6>        mov.w    r0, #12
0x800001c <part1+10>       mov.w    r3, #0
0x8000020 <part1+14>       mov.w    r2, #2
0x8000024 <part1+18>       subs     r4, r0, r2
0x8000026 <part1+20>       sbc.w    r5, r1, r3

extended-r Thread 1 In:  hard_reset      L16  PC: 0x8000010
_step1_1 () at part2.S:38
subPr2 () at part2.S:57
_step1_1 () at part2.S:44
_step1_4 () at part2.S:47
_step1_1 () at part2.S:38
subPr2 () at part2.S:57
_step1_1 () at part2.S:44
_step1_4 () at part2.S:47
_step3 () at part2.S:14
_step8 () at part2.S:27
__hard_reset__ () at start.S:16
```

Висновок

Ми вивчили архітектурні особливості, системи команд, принципи організації команд умовних та безумовних переходів та переходів на підпрограми, команди роботи з пам'яттю та способів адресації операндів.

На основі отриманих знаній було зроблено програмний проект на мові асемблера, що вирішує заданий математичний вираз з заданими умовами. Отримали очікувані результати. Також програма виконує арифметичні операції у відповідності з алгоритмом за варіантом. Забезпечили звернення з основної програми в підпрограми згідно з алгоритмом та варіантом. Налаштували умови розгалуження алгоритму. Отримали коректні результати.