Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №3

з дисципліни «Архітектура комп’ютерів-3»

на тему : «Завантажувач основної програми. Обробка виключень. Вивід даних на відлагоджувальний порт або консоль»

Роботу виконали:

студенти 3 курсу ФІОТ

групи ІВ-71

Молчанова В.С. (ЗК №7110)

Музика О.А. (ЗК №7111)

Роботу перевірив:

Каплунов А.В.

Київ – 2020

**Тема:** Завантажувач основної програми. Обробка виключень. Вивід даних на відлагоджувальний порт або консоль.

**Мета:** Навчитися працювати з оперативною пам’яттю, використовувати інструкції спеціального призначення, використовувати виключення процесора Cortex-M4. Створення мінімального завантажувача системи. Навчитися користуватися виводом даних через відлагоджувальний порт (або консоль).

## Варіант

(7110+7111)%16 = 13 => LDM, STM; інкремент; регістровий зсув; 12  (3 регістра по 4)

## Лістинг

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

#################

# CFLAGS

CFLAGS = -O0 -g3 -mthumb -Wall

##################

# LDFLAGS

LDFLAGS = -Wall --specs=nosys.specs -nostdlib -lgcc

##################

# PATH

APP\_PATH=$(abspath ./)

##################

# add here GNU ASSEMBLY SOURCES .S

GASSRC += start.S

GASSRC += print.S

GASSRC += bootloader.S

##################

SOBJS = $(GASSRC:.S=.o)

COBJS = $(patsubst .c,%.o,$(APP\_SRC))

.PHONY: all clean

# Path to directories containing application source

vpath % $(APP\_PATH)

all: $(TARGET).bin $(COBJS) $(SOBJS) $(TARGET).elf kernel.bin

%.o: %.S

    $(CC) -x assembler-with-cpp  $(CFLAGS) -mcpu=$(CPU\_CC) -c -o $@  $^

bootloader.S: kernel.bin

$(TARGET).elf: $(COBJS) $(SOBJS)

    $(CC) -mcpu=$(CPU\_CC) $(LDFLAGS) -T./lscript.ld -o $@ $^ $(INCFLAGS)

$(TARGET).bin: $(TARGET).elf $(COBJS) $(SOBJS)

    $(OBJCOPY) -O binary   $(TARGET).elf  $(TARGET).bin

kernel.bin:

    $(CC) -x assembler-with-cpp  $(CFLAGS) -mcpu=$(CPU\_CC) -c  kernel.S -o kernel.o

    $(CC) -x assembler-with-cpp  $(CFLAGS) -mcpu=$(CPU\_CC) -c  print.S -o print.o

    $(CC) -mcpu=$(CPU\_CC) $(LDFLAGS) -T./lscript\_kernel.ld -o kernel.elf kernel.o print.o $(INCFLAGS)

    $(OBJCOPY) -O binary  kernel.elf  kernel.bin

qemu:

    $(QEMU)  --verbose --verbose --board $(BOARD) --mcu $(MCU) -d unimp,guest\_errors --image $(TARGET).elf --semihosting-config enable=on,target=native -gdb tcp::$(TCP\_ADDR)  -S

qemu\_run:

    $(QEMU)  --verbose --verbose --board $(BOARD) --mcu $(MCU) -d unimp,guest\_errors --image $(TARGET).elf --semihosting-config enable=on,target=native

clean:

    -rm \*.o

    -rm \*.elf

    -rm \*.bin

flash:

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

kernel.s

// (a-b)\*3 + 2^c

.syntax unified

.cpu cortex-m4

.thumb

#define a #7

#define b #4

#define c #3

.global vtable\_kernel

.global \_\_kernel\_reset\_\_

.type vtable\_kernel, %object

.type \_\_kernel\_reset\_\_, %function

.section .interrupt\_vector

vtable\_kernel:

    .word \_\_stack\_start

    .word \_\_kernel\_reset\_\_+1

    .size vtable\_kernel, .-vtable\_kernel

.section .rodata

    data:  .asciz "kernel started!\n"

    final: .asciz "Value in register #3: "

.section .text

\_\_kernel\_reset\_\_:

    ldr r0, =data

    bl dbgput\_line

    // calculate

    mov r3, a

    mov r1, b

    mov r2, c

    //r3 = a - b

    sub r3, r1

    //r3 = (a - b) \* 3

    mov r1, #3

    mul r3, r1

    //r4 = 2^c

   mov r4, #2

   pow:

    cmp r2, #1

    beq sum

    lsl r4, r4, #1

    sub r2, #1

    bne pow

    sum:

        //r3 = (a-b)\*3 + 2^c (result)

    add r3, r4

    ldr r0, =final

    bl dbgput

    mov r0, r3

    bl dbgput\_num

    end:

    b end

bootloader.S

.syntax unified

.cpu cortex-m4

.thumb

.global bootload

.section .rodata

    image: .incbin "kernel.bin"

    end\_of\_image:

    str\_boot\_start: .asciz "bootloader started"

    str\_boot\_end: .asciz "bootload ended"

    str\_boot\_indicate: .asciz "#"

.section .text

bootload:

    ldr r0, =str\_boot\_start

    bl dbgput\_line

    ldr r0, =end\_of\_image

    ldr r1, =image

    ldr r2, =\_ram\_start

loop:

    ldm r1!, {r5, r6, r7}

    stm r2!, {r5, r6, r7}

    cmp r0, r1

    bhi loop

    bl newline

    ldr r0, =str\_boot\_end

    bl dbgput\_line

    ldr lr, =bootload\_end

    add lr, #1

    ldr r2, =\_ram\_start

    add r2, #4 // go to \_\_reset\_kernel\_\_

    ldr r0, [r2]

    bx r0

    bootload\_end:

    b bootload\_end

print.S

.thumb

.syntax unified

.cpu cortex-m4

#define SEMIHOSTING\_SYS\_WRITE0  #0x04

#define SEMIHOSTING #0xAB

.section .data

    str\_hex: .asciz "0xXXXXXXXX\n"

.text

.global dbgput\_line

.global dbgput

.global newline

.global dbgput\_num

// param: @str

dbgput:

    push {lr}

    // move str to r1

    mov r1, r0

    mov r0, SEMIHOSTING\_SYS\_WRITE0

    bkpt SEMIHOSTING

    pop {pc}

\_newline\_sym: .asciz "\n\r"

.align 4

dbgput\_line:

    push {lr}

    // move str to r1

    mov r1, r0

    mov r0, SEMIHOSTING\_SYS\_WRITE0

    bkpt SEMIHOSTING

    ldr r1,=\_newline\_sym

    mov r0, SEMIHOSTING\_SYS\_WRITE0

    bkpt SEMIHOSTING

    pop {pc}

newline:

    push {lr}

    ldr r1,=\_newline\_sym

    mov r0, SEMIHOSTING\_SYS\_WRITE0

    bkpt SEMIHOSTING

    pop {pc}

dbgput\_num:

    push {lr}

    mov r2, #9

    mov r3, #0x0000000F

    ldr r1, =str\_hex

    next:

        push {r0}

        and r0, r3

        add r0, #48

        cmp r0, #58

        blo store

        add r0, #7

    store:

        strb r0, [r1, r2]

        pop {r0}

        lsr r0, r0, #4

        sub r2, #1

        cmp r2, #2

        bge next

        ldr r1, =str\_hex

        mov r0, SEMIHOSTING\_SYS\_WRITE0

        bkpt SEMIHOSTING

        pop {pc}

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

.section .interrupt\_vector

vtable:

    .word \_\_stack\_start

    .word \_\_hard\_reset\_\_+1

    .size vtable, .-vtable

.section .text

\_\_hard\_reset\_\_:

// initialize stack here

// if not initialized yet

    .data

    d: .asciz "starting \n"

    .text

    ldr r0, =d

    bl dbgput\_line

    bl bootload

    \_loop: b \_loop

    .size \_\_hard\_reset\_\_, .-\_\_hard\_reset\_\_

lscript.ld

/\* linker script for stm32f1

\* platforms

\*/

MEMORY

{

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

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

}

\_\_stack\_start = ORIGIN(RAM) + LENGTH(RAM);

\_ram\_start = ORIGIN(RAM);

\_ram\_end = ORIGIN(RAM) + LENGTH(RAM);

SECTIONS

{

.text :

{

. = ALIGN(4);

KEEP(\*(.interrupt\_vector))

\*(.text)

\*(.text\*)

\*(.rodata)

\*(.rodata\*)

. = ALIGN(4);

} > FLASH

}

lscript\_kernel.ld

/\* linker script for stm32f1

\* platforms

\*/

MEMORY

{

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

}

\_\_stack\_start = ORIGIN(RAM) + LENGTH(RAM);

SECTIONS

{

.text :

{

. = ALIGN(4);

KEEP(\*(.interrupt\_vector))

\*(.text)

\*(.text\*)

\*(.rodata)

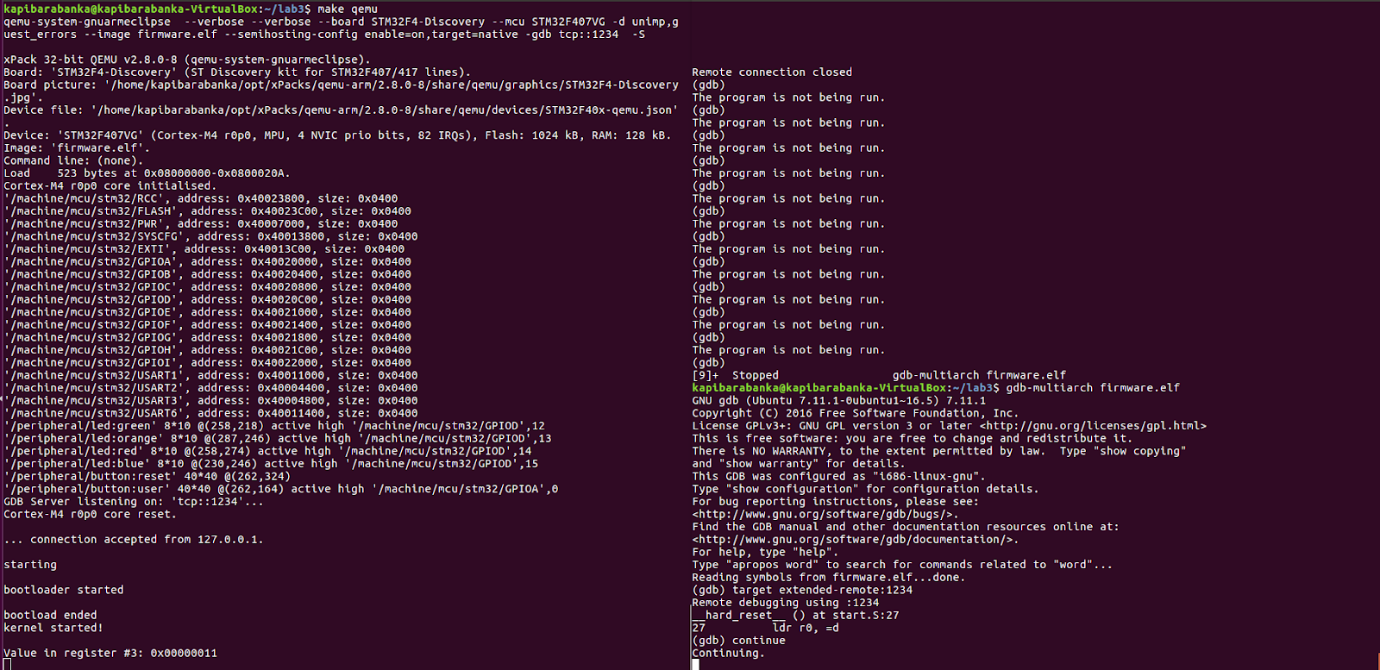
\*(.rodata\*)

. = ALIGN(4);

} > RAM

}

## Виконання

****

## Github

Посилання на git: <https://github.com/Kapibarabanka/CA_Lab3>

## Висновок

Під час даної лабораторної роботи ми навчилися працювати з оперативною пам’яттю, використовувати інструкції спеціального призначення, використовувати виключення процесора Cortex-M4, користуватися виводом даних через відлагоджувальний порт (або консоль). Створено мінімальний завантажувач системи.