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 №3

з дисципліни “Архітектура комп’ютерів 3”
на тему “Завантажувач основної програми. Обробка виключень. Вивід даних на відлагоджувальний порт або консоль.”

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

Перевірив:
Нікольський С. С.

Київ – 2021

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).

Завдання

Номер залікової книжки: 8113
8113 % 16 = 1

No	Команди для роботи з пам’яттю	Інкремент/Декремент регістру адреси	Вид зсуву	Кількість байт для зсуву
1	LDR, STR	інкремент	регістровий	4

Виконання роботи

bootloader.S

```
.syntax unified
.cpu cortex-m4
//.fpu softvfp
.thumb

.global bootload
.section .rodata
	image: .incbin "kernel.bin"
	end_of_image:
	str_boot_start: .asciz "bootloader started"
	str_boot_end: .asciz "bootloader end"
	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
	mov r4, #4

loop:
	ldr r3, [r1, r4]
	str r3, [r2, r4]
	add r1, r4
	add r2, r4
```

```
        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
ldr r0, [r2]
bx r0
bootload_end:
b bootload_end
```

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
}
```

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}
```

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
}
```

kernel.S

```
.syntax unified
.cpu cortex-m4
.thumb
#define A #5
#define B #11
```

```
#define C #2

.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 r0, A
        mov r1, B
        mov r2, C
        cmp r0, r1
        ITE GE
        addGE r3, r0, r1
        subLT r3, r0, r1
        sdiv r3, r2


        ldr r0, =final
        bl dbgput_line
        mov r0, r3
        bl dbgput_num

        end:
        b end
```

Makefile

```
SDK_PREFIX?=arm-none-eabi-
CC = $(SDK_PREFIX)gcc
LD = $(SDK_PREFIX)ld
SIZE = $(SDK_PREFIX)size
OBJCOPY = $(SDK_PREFIX)objcopy
QEMU = qemu-system-gnuarmeclipse
BOARD ?= STM32F4-Discovery
MCU=STM32F407VG
TARGET=firmware
CPU_CC=cortex-m4
TCP_ADDR=1234
##################
# CFLAGS
CFLAGS = -O0 -g3 -Wall
####################
# LDFLAGS
LDFLAGS = -Wall --specs=nosys.specs -nostdlib -lgcc
####################
# PATH
APP_PATH=$(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
```

Результати виконання

a=11, b=5, c=2
r3 = (a+b)/c = 8

```
Cortex-M4 r0p0 core reset.

starting

bootloader started

bootloader end
kernel started!

Value in register #3:
0x00000008
Graphic window closed. Quit.
make: *** [Makefile:49: qemu_run] Error 1
```

a=5, b=5, c=2
r3 = (a+b)/c = 5

```
Cortex-M4 r0p0 core reset.

starting

bootloader started

bootloader end
kernel started!

Value in register #3:
0x00000005
Graphic window closed. Quit.
make: *** [Makefile:49: qemu_run] Error 1
```

a=5, b=11, c=2
r3 = (a-b)/c = -3

```
Cortex-M4 r0p0 core reset.

starting

bootloader started

bootloader end
kernel started!

Value in register #3:
0xFFFFFFFD
Graphic window closed. Quit.
make: *** [Makefile:49: qemu_run] Error 1
```

Висновок

В ході виконання лабораторної роботи було вивчено асемблерні інструкції ядра Cortex-M4, що використовуються для доступу до пам’яті та створено проект на асемблері, що виконує завантаження програми, визначеної в файлі bootloader.S в оперативну пам’ять для її виконання (розрахунку за заданою формулою). Виконання проекту було перевірено за допомогою відлагоджувача gdb. Також було створено Makefile для автоматизації створення прошивки.