1. INRODUCTION:

The objective of this lab is to get familiarized with the development board, understand all the ICs and their connections with the microprocessor. Practice assembly language, and write basic programs. Work with the debugging tools for analysing program flow. In this lab we worked with assembly language, practice connecting basic components such as LEDs and buttons to the board, read from / write to them.

2. PROBLEMS:

2.1. Problem 1:

Write assembly code that will toggle the on-board LED at a rate of 1 second.

2.1.1. Theoretical Research:

WHAT IS GPIO:

Stands for "General Purpose Input/Output." GPIO is a type of pin found on an integrated circuit that does not have a specific function. While most pins have a dedicated purpose, such as sending a signal to a certain component, the function of a GPIO pin is customizable and can be controlled by software.

Not all chips have GPIO pins, but they are commonly found on multifunction chips, such as those used in power managers and audio/video cards. They are also used by system-on-chip (SOC) circuits, which include a processor, memory, and external interfaces all on a single chip. GPIO pins allow these chips to be configured for different purposes and work with several types of components. A popular device that makes use of GPIO pins is the Raspberry Pi, a single-board computer designed for hobbyists and educational purposes. It includes a row of GPIO pins along the edge of the board that provide the interface between the Raspberry Pi and other components. These pins act as switches that output 3.3 volts when set to HIGH and no voltage when set to LOW. You can connect a device to specific GPIO pins and control it with a software program. For example, you can wire an LED to a GPIO and a ground pin on a Raspberry Pi. If a software program tells the GPIO pin to turn on, the LED will light up.

Most computer users will not encounter GPIO pins and do not need to worry about configuring them. However, if you are a hobbyist or computer programmer, it can be helpful to learn what chips have GPIO pins and how to make use of them.

```
/*
 * problem_1.s
 *
 *  Mehmet Akif Gümüş
 *
 */


.syntax unified
.cpu cortex-m0plus
.fpu softvfp
.thumb


/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss


/* define peripheral addresses from RM0444 page 57, Tables 3-4 */
.equ RCC_BASE,         (0x40021000)          // RCC base address
.equ RCC_IOPENR,       (RCC_BASE   + (0x34)) // RCC IOPENR register offset

.equ GPIOC_BASE,       (0x50000800)          // GPIOC base address
.equ GPIOC_MODER,      (GPIOC_BASE + (0x00)) // GPIOC MODER register offset
.equ GPIOC_ODR,        (GPIOC_BASE + (0x14)) // GPIOC ODR register offset


/* vector table, +1 thumb mode */
.section .vectors
vector_table:
      .word _estack             /*     Stack pointer */
      .word Reset_Handler +1    /*     Reset handler */
      .word Default_Handler +1  /*       NMI handler */
      .word Default_Handler +1  /* HardFault handler */
      /* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
      /* set stack pointer */
      ldr r0, =_estack
      mov sp, r0

      /* initialize data and bss
       * not necessary for rom only code
       * */
      bl init_data
      /* call main */
      bl main
      /* trap if returned */
```

```
	/* trap if returned */
	b .


/* initialize data and bss sections */
.section .text
init_data:

	/* copy rom to ram */
	ldr r0, =_sdata
	ldr r1, =_edata
	ldr r2, =_sidata
	movs r3, #0
	b LoopCopyDataInit

	CopyDataInit:
		ldr r4, [r2, r3]
		str r4, [r0, r3]
		adds r3, r3, #4

	LoopCopyDataInit:
		adds r4, r0, r3
		cmp r4, r1
		bcc CopyDataInit

	/* zero bss */
	ldr r2, =_sbss
	ldr r4, =_ebss
	movs r3, #0
	b LoopFillZerobss

	FillZerobss:
		str  r3, [r2]
		adds r2, r2, #4

	LoopFillZerobss:
		cmp r2, r4
		bcc FillZerobss

	bx lr


/* default handler */
.section .text
Default_Handler:
	b Default_Handler


/* main function */
.section .text
main:

	/* sent clock for enable GPIOC, bit2 on IOPENR */

	ldr r6, =RCC_IOPENR
	ldr r5, [r6]
```

```
	movs r4, 0x4  //4
	orrs r5, r5, r4
	str r5, [r6]

	/* the MODER's bits 12-13 setup PC6 for led1*/
	ldr r2, =GPIOC_MODER
	ldr r3, [r2]
	/* cannot do with movs, so use pc relative */
	movs r4,0x3 //0011
	lsls r4,r4,#12
	bics r3,r3,r4
	movs r4, 0x1  //0001
	lsls r4, r4, #12
	orrs r3, r3, r4
	str r3, [r2]
led_loop:
	/* led on C6 in ODR in  */
	ldr r6, =GPIOC_ODR
	ldr r5, [r6]
	movs r4, 0x40 //01000000
	orrs r5, r5, r4
	str r5, [r6]

	ldr r1,=#100000
	bl delay

	/* led off */
	ldr r6, =GPIOC_ODR
	ldr r5, [r6]
	movs r4, 0x40 //01000000
	bics r5, r5, r4
	str r5, [r6]

	ldr r1,=#1000000
	bl delay
	b led_loop
	delay:
		subs r1,r1,#1
		bne delay
		bx lr

	b .

	/* this should never get executed */
	nop
```

2.2 PROBLEM-2

Connect a button to the board, and turn on the on-board LED when the button is pressed. When the button is released, the LED should turn off.

```
/*
 * problem_2.s
 *
 * Mehmet Akif Gümüş
 *
 */


.syntax unified
.cpu cortex-m0plus
.fpu softvfp
.thumb


/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss


/* define peripheral addresses from RM0444 page 57, Tables 3-4 */
.equ RCC_BASE,         (0x40021000)          // RCC base address
.equ RCC_IOPENR,       (RCC_BASE   + (0x34)) // RCC IOPENR register offset

.equ GPIOA_BASE,       (0x50000000)          // GPIOA base address
.equ GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOA MODER register offset
.equ GPIOA_ODR,        (GPIOA_BASE + (0x14)) // GPIOA ODR output offset
.equ GPIOA_IDR,        (GPIOA_BASE + (0x10)) // GPIOA ODR input offset

/* vector table, +1 thumb mode */
.section .vectors
vector_table:
       .word _estack             /*     Stack pointer */
       .word Reset_Handler +1    /*     Reset handler */
       .word Default_Handler +1  /*       NMI handler */
       .word Default_Handler +1  /* HardFault handler */
       /* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
       /* set stack pointer */
       ldr r0, =_estack
       mov sp, r0

       /* initialize data and bss
        * not necessary for rom only code
```

```
         * */
        bl init_data
        /* call main */
        bl main
        /* trap if returned */
        b .


/* initialize data and bss sections */
.section .text
init_data:

        /* copy rom to ram */
        ldr r0, =_sdata
        ldr r1, =_edata
        ldr r2, =_sidata
        movs r3, #0
        b LoopCopyDataInit

        CopyDataInit:
                ldr r4, [r2, r3]
                str r4, [r0, r3]
                adds r3, r3, #4

        LoopCopyDataInit:
                adds r4, r0, r3
                cmp r4, r1
                bcc CopyDataInit

        /* zero bss */
        ldr r2, =_sbss
        ldr r4, =_ebss
        movs r3, #0
        b LoopFillZerobss

        FillZerobss:
                str  r3, [r2]
                adds r2, r2, #4

        LoopFillZerobss:
                cmp r2, r4
                bcc FillZerobss

        bx lr


/* default handler */
.section .text
Default_Handler:
        b Default_Handler
```

```
/* main function */
.section .text
main:
	/* sent clock for enable GPIOC, bit2 on IOPENR */

	ldr r6, =RCC_IOPENR
	ldr r5, [r6]
	movs r4, 0x1
	orrs r5, r5, r4
	str r5, [r6]

	/* setup PA7 as output pin for led_one 01 for bits 14-15 in MODER */
	ldr r6, =GPIOA_MODER
	ldr r5, [r6]
	/* cannot do with movs, so use pc relative */
	movs r4,0x3 //0011
	lsls r4,r4,#14
	bics r5,r5,r4
	movs r4, 0x1 //0001
	lsls r4, r4, #14
	orrs r5, r5, r4
	str r5, [r6]

	/* setup PA4 as input pin for button 00 for bits 8-9 in MODER */
	ldr r3, =GPIOA_MODER
	ldr r5, [r3]
	/* cannot do with movs, so use pc relative */
	movs r4,0x3
	lsls r4,r4,#8
	bics r5,r5,r4
	str r5, [r3]
button_control:

	/* press button connected to PA4 in IDR*/
	ldr r6, =GPIOA_IDR
	ldr r5, [r6]
	lsrs r5, r5, #4
	movs r4, #0x1 //0001
	ands r5, r5, r4


	cmp r5, #0x1
	beq led_on
	bne led_off

led_on:

	/* turn on led connected to PA7 in ODR */
	ldr r6, =GPIOA_ODR
	ldr r5, [r6]
	movs r4, 0x80  //100 0000
	orrs r5, r5, r4
	str r5, [r6]
	b button_control
led_off:
/* turn off led connected to PA7 in ODR */
	ldr r6, =GPIOA_ODR
	ldr r5, [r6]
```

```
led_off:
/* turn off led connected to PA7 in ODR */
      ldr r6, =GPIOA_ODR
      ldr r5, [r6]
      movs r4, 0x80
      bics r5, r5, r4
      str r5, [r6]
      b button_control

      /* for(;;); */


      /* this should never get executed */
      nop
```

2.3 PROBLEM-3

Connect 8 external LEDs to the board, and toggle all the LEDs at the same time at a rate of 1 second.

```
/*
 * problem_3.s
 *
 * Mehmet Akif Gümüş
 *
 */


.syntax unified
.cpu cortex-m0plus
.fpu softvfp
.thumb


/* make linker see this */
.global Reset_Handler

/* get these from linker script */
.word _sdata
.word _edata
.word _sbss
.word _ebss


/* define peripheral addresses from RM0444 page 57, Tables 3-4 */
.equ RCC_BASE,         (0x40021000)          // RCC base address
.equ RCC_IOPENR,       (RCC_BASE   + (0x34)) // RCC IOPENR register offset

.equ GPIOA_BASE,       (0x50000000)          // GPIOA base address
.equ GPIOA_MODER,      (GPIOA_BASE + (0x00)) // GPIOA MODER register offset
.equ GPIOA_ODR,        (GPIOA_BASE + (0x14)) // GPIOA ODR output offset
.equ GPIOA_IDR,        (GPIOA_BASE + (0x10)) // GPIOA ODR input offset
```

```
/* vector table, +1 thumb mode */
.section .vectors
vector_table:
	.word _estack             /*     Stack pointer */
	.word Reset_Handler +1    /*     Reset handler */
	.word Default_Handler +1  /*       NMI handler */
	.word Default_Handler +1  /* HardFault handler */
	/* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
	/* set stack pointer */
	ldr r0, =_estack
	mov sp, r0

	/* initialize data and bss
	 * not necessary for rom only code
	 * */
	bl init_data
	/* call main */
	bl main
	/* trap if returned */
	b .


/* initialize data and bss sections */
.section .text
init_data:

	/* copy rom to ram */
	ldr r0, =_sdata
	ldr r1, =_edata
	ldr r2, =_sidata
	movs r3, #0
	b LoopCopyDataInit

	CopyDataInit:
		ldr r4, [r2, r3]
		str r4, [r0, r3]
		adds r3, r3, #4

	LoopCopyDataInit:
		adds r4, r0, r3
		cmp r4, r1
		bcc CopyDataInit

	/* zero bss */
	ldr r2, =_sbss
	ldr r4, =_ebss
	movs r3, #0
	b LoopFillZerobss

	FillZerobss:
		str  r3, [r2]
		adds r2, r2, #4
```

```
/* vector table, +1 thumb mode */
.section .vectors
vector_table:
	.word _estack             /*     Stack pointer */
	.word Reset_Handler +1    /*     Reset handler */
	.word Default_Handler +1  /*       NMI handler */
	.word Default_Handler +1  /* HardFault handler */
	/* add rest of them here if needed */


/* reset handler */
.section .text
Reset_Handler:
	/* set stack pointer */
	ldr r0, =_estack
	mov sp, r0

	/* initialize data and bss
	 * not necessary for rom only code
	 * */
	bl init_data
	/* call main */
	bl main
	/* trap if returned */
	b .


/* initialize data and bss sections */
.section .text
init_data:

	/* copy rom to ram */
	ldr r0, =_sdata
	ldr r1, =_edata
	ldr r2, =_sidata
	movs r3, #0
	b LoopCopyDataInit

	CopyDataInit:
		ldr r4, [r2, r3]
		str r4, [r0, r3]
		adds r3, r3, #4

	LoopCopyDataInit:
		adds r4, r0, r3
		cmp r4, r1
		bcc CopyDataInit

	/* zero bss */
	ldr r2, =_sbss
	ldr r4, =_ebss
	movs r3, #0
	b LoopFillZerobss

	FillZerobss:
		str  r3, [r2]
		adds r2, r2, #4
```

```
	LoopFillZerobss:
		cmp r2, r4
		bcc FillZerobss

	bx lr


/* default handler */
.section .text
Default_Handler:
	b Default_Handler


/* main function */
.section .text
main:
		/* sent clock for enable GPIOC, bit2 on IOPENR */

	ldr r6, =RCC_IOPENR
	ldr r5, [r6]
	movs r4, 0x1 //1
	orrs r5, r5, r4
	str r5, [r6]
	/* the MODER's bits 12-13 setup PC6 for led01 */
	ldr r6, =GPIOA_MODER
	ldr r5, [r6]
	/* cannot do with movs, so use pc relative */
	movs r4,0x3 //0000 0011
	lsls r4,r4,#12
	bics r5,r5,r4
	movs r4, 0x1  //0000 0001
	lsls r4, r4, #12
	orrs r5, r5, r4
	str r5, [r6]
led_loop:
	/* led on connected with PA6 in ODR */
	ldr r6, =GPIOA_ODR
	ldr r5, [r6]
	movs r4, 0x40 //0000 0100 0000
	orrs r5, r5, r4
	str r5, [r6]

	ldr r1,=#1000000
	bl delay

	/* led off */
	ldr r6, =GPIOA_ODR
	ldr r5, [r6]
	movs r4, 0x40 //0100 0000
	bics r5, r5, r4
	str r5, [r6]

	ldr r1,=#1000000
	bl delay
	b led_loop
	delay:
		subs r1,r1,#1
		bne delay
		bx lr
```

```
        subs r1,r1,#1
        bne delay
        bx lr

b .

/* this should never get executed */
nop
```

3. CONCLUSION

The purpose of the experiment are understand all the ICs and their connections with the microprocessor. Practiced assembly language, and written basic programs. Worked with the debugging tools for analysing program flow. In this experiment, worked on assembly code. STM32 board is used for the first time. In order to use the pins on the card, a clock signal was sent to those pins first. After the pins were opened for use, GPIO IDR and ODR registers were assigned according to the datasheet. Necessary connections have been made on the card. Connections were made according to the information form (A6 = PC6) that came with the card. Leds were on as desired in the experiment sheet.

4. References

- https://www.raspberrypi.org/documentation/usage/gpio/
- https://iotality.com/blink-led-arm-assembly/
- http://www.icbase.com/File/PDF/STM/STM48041812.pdf

5. VİDEO LİNKİ:

PROBLEM-1: https://youtu.be/sO4FYZDNV6Y
PROBLEM-2: https://youtu.be/aE-yHzuBYGc
PROBLEM-3: https://youtu.be/zDQSo6f07i0