Lab Sheet for Week 5

Lab Instructor: **RSP**

Lab 1: AVR Stack Operations

Objectives

- Use Microchip Studio for AVR and the built-in simulator/debugger to analyze AVR assembly code.
- Use the simulator to debug programs and observe the internal state of the ATmega328P microcontroller during code execution.
- Use breakpoints effectively to control program execution and aid in debugging.
- Analyze the stack operations of the AVR.

Lab Procedure

1. Create a new project in **Microchip Studio for AVR** or **Microchip MPLAB X IDE.**

    1.1 Select **AVR Assembly**, enter a project name, and choose a directory for the project.

    1.2 Select **ATmega328P** as the target device and click the "OK" button.

    1.3 Open the **.asm** file to edit the source code.

2. Write AVR assembly code in **main.asm** as follows:

    2.1 Use the provided AVR assembly code as shown in **Code Listing 1**.

    2.2 Build the project to produce the output files.

3. Use the built-in AVR simulator to simulate the assembly program.

    3.1 Start a **debugging session** (choose **Debug → Start Debugging** and **Break** from the menu) using the simulator to begin the code execution.

    3.2 Ensure that the built-in simulator is selected as the debugger.

    3.3 Add some breakpoints to the code before starting the debug process (as shown in **Figure 1**).

    3.4 Open the **Processor Status View, CPU Registers View, I/O View** and **Memory View** to observe the MCU state.

    3.5 Step through the code repeatedly until the next breakpoint is reached.

    3.6 Observe the following:

    - The contents of the CPU registers during the **PUSH** and **POP** operations.

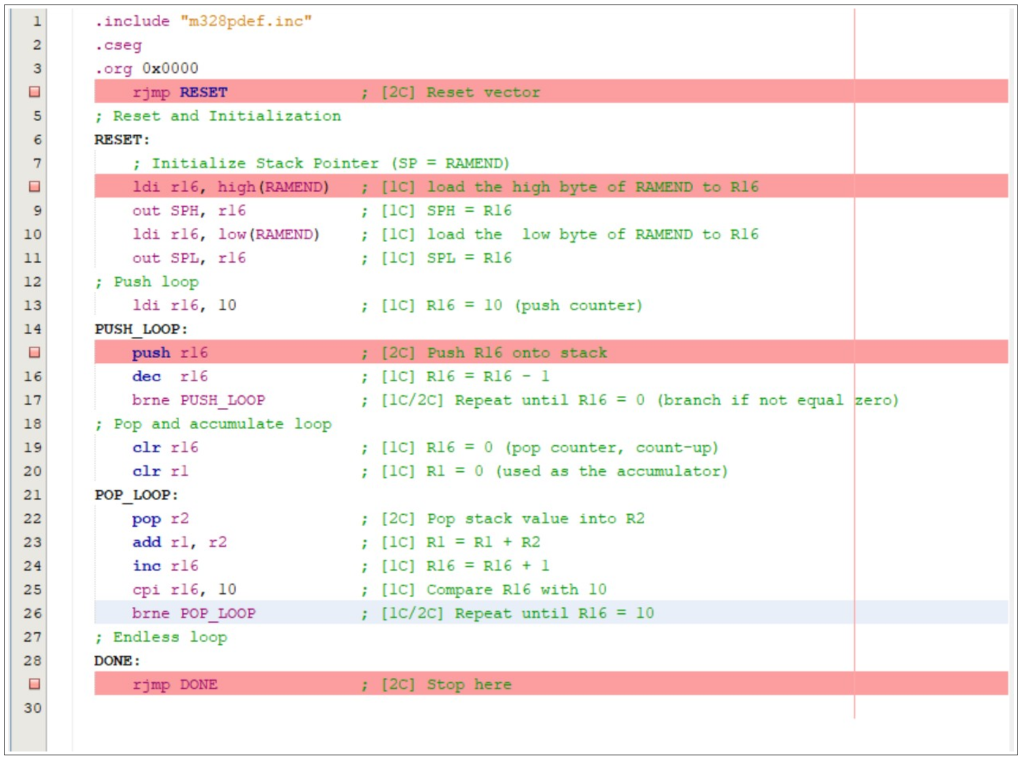
    - How the **stack pointer (SP)** changes during program execution, including the stack growth direction.

    - The number of clock cycles required to execute each instruction.

4. Capture screenshots of the Microchip Studio for AVR simulator showing the CPU registers and memory contents at different time steps for inclusion in the lab report.

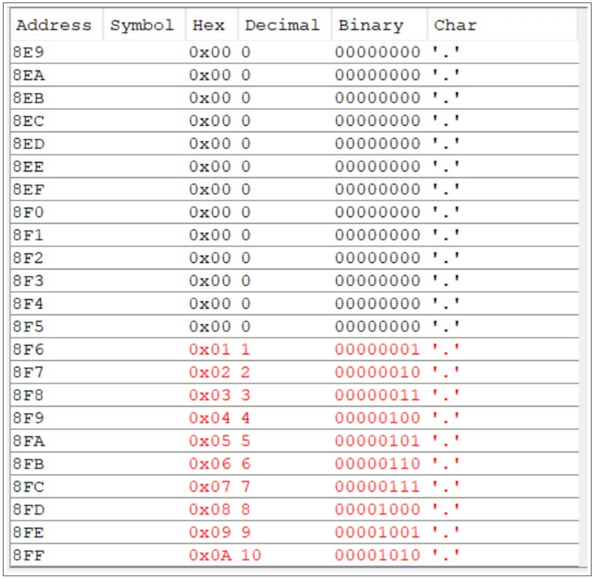
```
.include "m328pdef.inc"
.cseg
.org 0x0000
    rjmp RESET              ; [2C] Reset vector

; Reset and Initialization
RESET:
    ; Initialize Stack Pointer (SP = RAMEND)
    ldi r16, high(RAMEND)   ; [1C] load the high byte of RAMEND to R16
    out SPH, r16            ; [1C] SPH = R16
    ldi r16, low(RAMEND)    ; [1C] load the  low byte of RAMEND to R16
    out SPL, r16            ; [1C] SPL = R16

; Push loop
    ldi r16, 10             ; [1C] R16 = 10 (push counter)
PUSH_LOOP:
    push r16                ; [2C] Push R16 onto stack
    dec  r16                ; [1C] R16 = R16 - 1
    brne PUSH_LOOP          ; [1C/2C] Repeat until R16=0 (branch if not equal zero)

; Pop and accumulate loop
    clr r16                 ; [1C] R16 = 0 (pop counter, count-up)
    clr r1                  ; [1C] R1 = 0 (used as the accumulator)
POP_LOOP:
    pop r2                  ; [2C] Pop stack value into R2
    add r1, r2              ; [1C] R1 = R1 + R2
    inc r16                 ; [1C] R16 = R16 + 1
    cpi r16, 10             ; [1C] Compare R16 with 10
    brne POP_LOOP           ; [1C/2C] Repeat until R16=10

; Endless loop
DONE:
    rjmp DONE               ; [2C] Stop here
```

Code Listing 1: AVR assembly code template for Lab 1



```
.include "m328pdef.inc"
.cseg
.org 0x0000
    rjmp RESET              ; [2C] Reset vector
; Reset and Initialization
RESET:
    ; Initialize Stack Pointer (SP = RAMEND)
    ldi r16, high(RAMEND)   ; [1C] load the high byte of RAMEND to R16
    out SPH, r16            ; [1C] SPH = R16
    ldi r16, low(RAMEND)    ; [1C] load the  low byte of RAMEND to R16
    out SPL, r16            ; [1C] SPL = R16
; Push loop
    ldi r16, 10             ; [1C] R16 = 10 (push counter)
PUSH_LOOP:
    push r16                ; [2C] Push R16 onto stack
    dec  r16                ; [1C] R16 = R16 - 1
    brne PUSH_LOOP          ; [1C/2C] Repeat until R16 = 0 (branch if not equal zero)
; Pop and accumulate loop
    clr r16                 ; [1C] R16 = 0 (pop counter, count-up)
    clr r1                  ; [1C] R1 = 0 (used as the accumulator)
POP_LOOP:
    pop r2                  ; [2C] Pop stack value into R2
    add r1, r2              ; [1C] R1 = R1 + R2
    inc r16                 ; [1C] R16 = R16 + 1
    cpi r16, 10             ; [1C] Compare R16 with 10
    brne POP_LOOP           ; [1C/2C] Repeat until R16 = 10
; Endless loop
DONE:
    rjmp DONE               ; [2C] Stop here

```

Figure 1: Example of Breakpoint Settings

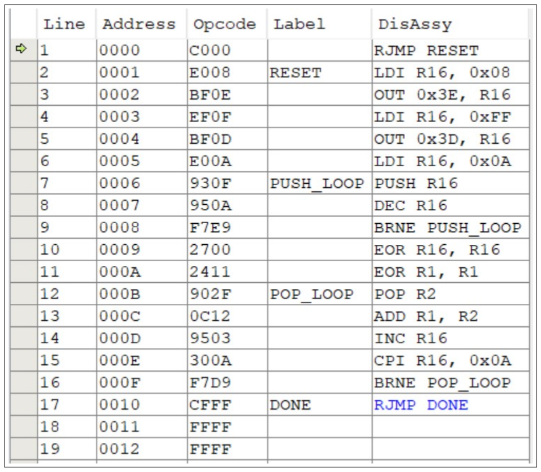


Address	Symbol	Hex	Decimal	Binary	Char
8E9		0x00	0	00000000	'.'
8EA		0x00	0	00000000	'.'
8EB		0x00	0	00000000	'.'
8EC		0x00	0	00000000	'.'
8ED		0x00	0	00000000	'.'
8EE		0x00	0	00000000	'.'
8EF		0x00	0	00000000	'.'
8F0		0x00	0	00000000	'.'
8F1		0x00	0	00000000	'.'
8F2		0x00	0	00000000	'.'
8F3		0x00	0	00000000	'.'
8F4		0x00	0	00000000	'.'
8F5		0x00	0	00000000	'.'
8F6		0x01	1	00000001	'.'
8F7		0x02	2	00000010	'.'
8F8		0x03	3	00000011	'.'
8F9		0x04	4	00000100	'.'
8FA		0x05	5	00000101	'.'
8FB		0x06	6	00000110	'.'
8FC		0x07	7	00000111	'.'
8FD		0x08	8	00001000	'.'
8FE		0x09	9	00001001	'.'
8FF		0x0A	10	00001010	'.'

Figure 2: AVR SRAM Memory View (only some memory locations are shown)

Post-Experiment Questions

1. Which register pair is used to implement the stack pointer (SP) in the AVR architecture?
2. What are SPH and SPL?
3. Which I/O registers (Special Function Registers, SFRs) are used to store the high byte and low byte of the stack pointer?
4. During program execution, how does the stack pointer (SP) change when a PUSH instruction is executed?
5. How does the stack pointer (SP) change when a POP instruction is executed?
6. In which direction does the stack grow in the SRAM address space of an AVR microcontroller?
7. How many machine cycles are required to execute a PUSH instruction?
8. How many machine cycles are required to execute a POP instruction?
9. What is the highest possible value of the stack pointer after reset, and why is this value used?
10. What is the value of `RAMEND` for the ATmega328P device?
11. Why is it necessary to initialize the stack pointer before using PUSH or POP instructions?
12. When program execution reaches the instruction "`rjmp DONE`", what is the final content of register R1? Explain how this value is obtained.
13. If the simulator executes the program starting from the instruction " `rjmp RESET`" and then first halts at the instruction "`rjmp DONE`", how many CPU cycles have been executed up to that point? Explain how you calculate the total number of CPU cycles for your answer.
14. What is the program memory address (word address) of the instruction "`rjmp DONE`" in the program memory?
15. Based on the program memory listing shown in **Figure 3**. How many instructions does this program have? Which instructions are 16-bit and which are 32-bit?



	Line	Address	Opcode	Label	DisAssy
➪	1	0000	C000		RJMP RESET
	2	0001	E008	RESET	LDI R16, 0x08
	3	0002	BF0E		OUT 0x3E, R16
	4	0003	EF0F		LDI R16, 0xFF
	5	0004	BF0D		OUT 0x3D, R16
	6	0005	E00A		LDI R16, 0x0A
	7	0006	930F	PUSH_LOOP	PUSH R16
	8	0007	950A		DEC R16
	9	0008	F7E9		BRNE PUSH_LOOP
	10	0009	2700		EOR R16, R16
	11	000A	2411		EOR R1, R1
	12	000B	902F	POP_LOOP	POP R2
	13	000C	0C12		ADD R1, R2
	14	000D	9503		INC R16
	15	000E	300A		CPI R16, 0x0A
	16	000F	F7D9		BRNE POP_LOOP
	17	0010	CFFF	DONE	RJMP DONE
	18	0011	FFFF		
	19	0012	FFFF		

Figure 3: AVR Program Memory View

Lab 2: I/O Bit-Banging

Objectives

- Use Microchip Studio for AVR and the built-in simulator/debugger to analyze AVR assembly code.
- Use the simulator to debug programs and observe the internal state of the ATmega328P microcontroller during code execution.
- Use breakpoints effectively to control program execution and aid in debugging.
- Verify the program using real hardware.

Lab Procedure

1. Create a new project in **Microchip Studio for AVR** or **Microchip MPLAB X IDE.**

    1.1 Select **AVR Assembly**, enter a project name, and choose a directory for the project.

    1.2 Select **ATmega328P** as the target device and click the "OK" button.

    1.3 Open the **.asm** file to edit the source code.

2. Write AVR assembly code in **main.asm** as follows:

    2.1 Use the provided AVR assembly code as shown in **Code Listing 2**.

    2.2 Build the project to produce the output files.

3. Use the built-in AVR simulator to simulate the assembly program.

4. Use the **AVRDUDE** program to upload the .hex file to the Arduino board and use a digital oscilloscope or a USB logic analyzer to analyze the output waveform at the **PB5** pin.

    4.1 Measure the positive pulse width and the frequency of the waveform.

    4.2 Capture the signal waveforms for inclusion in the lab report.

5. Revise the AVR assembly program so that the **LSB is shifted out first** (instead of MSB first), and use the constant value **0xED** for `DATA_BYTE`.

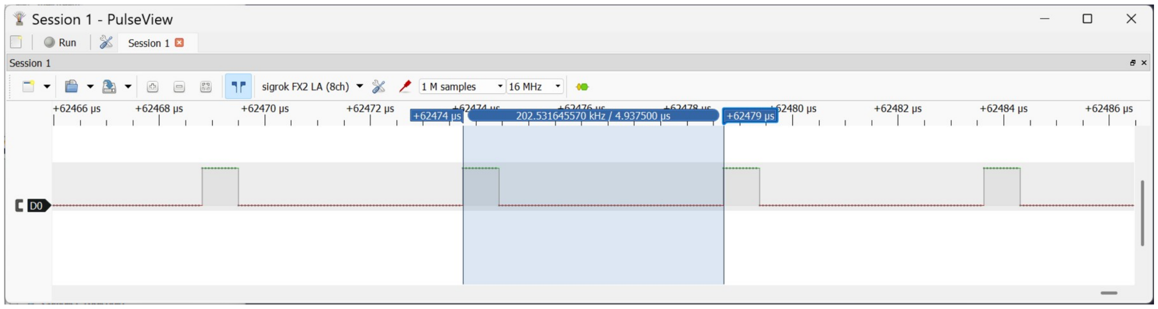
    5.1 Build the project and upload the **.hex file** to the Arduino board to verify the correctness of the program.

    5.2 Measure the frequency of the waveform.

    5.3 Capture the signal waveforms for inclusion in the lab report.

```
.include "m328pdef.inc"
.def BIT_COUNT = r17        ; bit counter
.def DATA_BYTE = r18        ; data byte to shift out

.cseg
.org 0x0000
    rjmp RESET              ; [2C] Reset vector

; Reset and Initialization
RESET:
    ; Initialize Stack Pointer (SP = RAMEND)
    ldi r16, high(RAMEND)   ; [1C]
    out SPH, r16            ; [1C]
    ldi r16, low(RAMEND)    ; [1C]
    out SPL, r16            ; [1C]

    ; Configure PB5 as output
    sbi DDRB, PB5           ; [2C]

; Main loop: shift out the MSB first on PB5
MAIN_LOOP:
    cbi  PORTB, PB5         ; [2C] clear PB5
    ldi DATA_BYTE, 0x80     ; [1C] load DATA_BYTE
    ldi BIT_COUNT, 8        ; [1C] set BIT_COUNT (8 bits)
    nop                     ; [1C] no operation
    nop                     ; [1C] no operation

SHIFT_LOOP:
    ; Output MSB of DATA_BYTE to PB5
    sbrc DATA_BYTE, 7       ; [1C/2C] skip next if MSB = 0
    sbi  PORTB, PB5         ; [2C] set PB5 (output logic 1)
    sbrs DATA_BYTE, 7       ; [1C/2C] skip next if MSB = 1
    cbi  PORTB, PB5         ; [2C] clear PB5 (output logic 0)

    lsl DATA_BYTE           ; [1C] logical shift left the data byte
    dec BIT_COUNT           ; [1C] BIT_COUNT--
    brne SHIFT_LOOP         ; [1C/2C] loop until BIT_COUNT is zero

    rjmp MAIN_LOOP          ; [2C] repeat the main loop
```

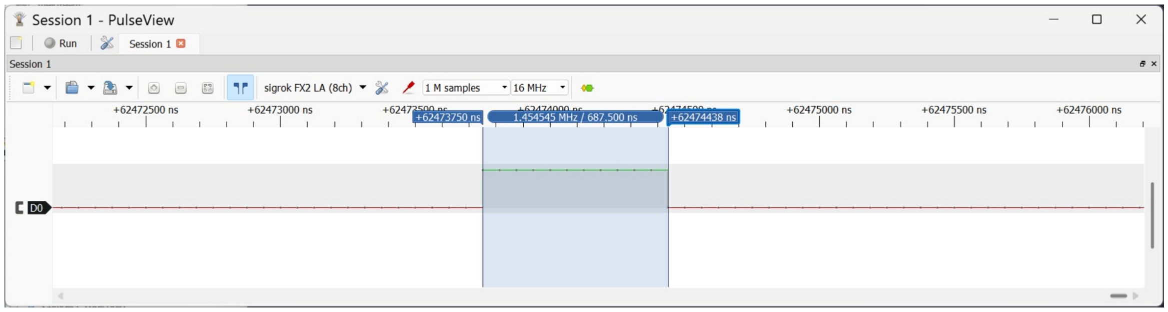
Code Listing 2: AVR assembly code template for Lab 2


Session 1 - PulseView

sigrok FX2 LA (8ch)   1 M samples   16 MHz

+62474 µs   202.531645570 kHz / 4.937500 µs   +62479 µs

Figure 4: Period Measurement using FX2 USB Logic Analyzer / PulseView


Session 1 - PulseView

sigrok FX2 LA (8ch)   1 M samples   16 MHz

+62473750 ns   1.454545 MHz / 687.500 ns   +62474438 ns

Figure 5: Positive Pulse Width Measurement using FX2 USB Logic Analyzer / PulseView

Post-Experiment Questions

1. How many machine cycles does the CPU take to complete one iteration of the MAIN loop? How is this number of cycles related to the period and frequency of the output signal?
2. If the CPU clock frequency is 16 MHz, what is the output signal frequency at the **PB5** pin? What are the duty cycle and the positive pulse width of the signal?
3. If the data byte is changed from **0x80** to **0xFE**, what is the positive pulse width of the output signal at the **PB5** pin? Confirm your answer by capturing and analyzing the signal waveforms using a digital oscilloscope or a USB logic analyzer.