Lab Sheet for Week 6

Lab Instructor: **RSP**

Lab 1: Implement a 24-bit down counter in AVR assembly

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 appropriately to control program execution and support debugging.
- Analyze the execution time and the machine cycle count of an AVR program.

Lab Procedure

In this lab, three AVR assembly code listings are provided as examples. Each program toggles the **PB5** pin of the **ATmega328P** and use a countdown loop to implement a time delay between two consecutive I/O pin toggles.

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 and set the CPU clock to 16MHz.

   3.4 Open simulator / debugging info windows (such as **Processor Status View, CPU Registers View, I/O View** and **Stopwatch,** depending on the IDE used) to observe the MCU state.

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

3.6 Observe the following:

- The number of clock cycles required to execute each instruction
- The cycle count between two consecutive breakpoint events in the main loop (e.g. pin toggles)

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

5. Build the project and upload the .hex file to the microcontroller board.

   5.1 Use a digital oscilloscope or an USB logic analyzer to measure the pulse widths of the signal.

   5.2 Capture the waveforms for inclusion in the lab report.

   5.3 Compare the measured pulse widths with the values determined by the simulation method.

6. Replace the AVR assembly code with the provided code shown in **Code Listing 2** and **Code Listing 3** respectively.

   6.1 Use the simulator to determine the high and low pulse widths of the output signal for each code example.

   6.2 Compare the measured pulse widths with the values determined by the simulation method.

```
.include "m328pdef.inc"
.cseg
.org 0x0000
    rjmp main
main:
    ; configure PB5 as output (LED)
    sbi DDRB, DDB5
main_loop:
    ; Use three registers r20:r19:r18 for 24-bit counter
    ldi r20, 10          ; [1C]
    ldi r19, 0           ; [1C]
    ldi r18, 0           ; [1C]
countdown_loop:
    dec r18              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    dec r19              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    dec r20              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    sbi PINB, PINB5      ; [2C] toggle PB5
    rjmp main_loop       ; [2C] repeat the main loop
```

Code Listing 1: AVR assembly code template for Lab 1

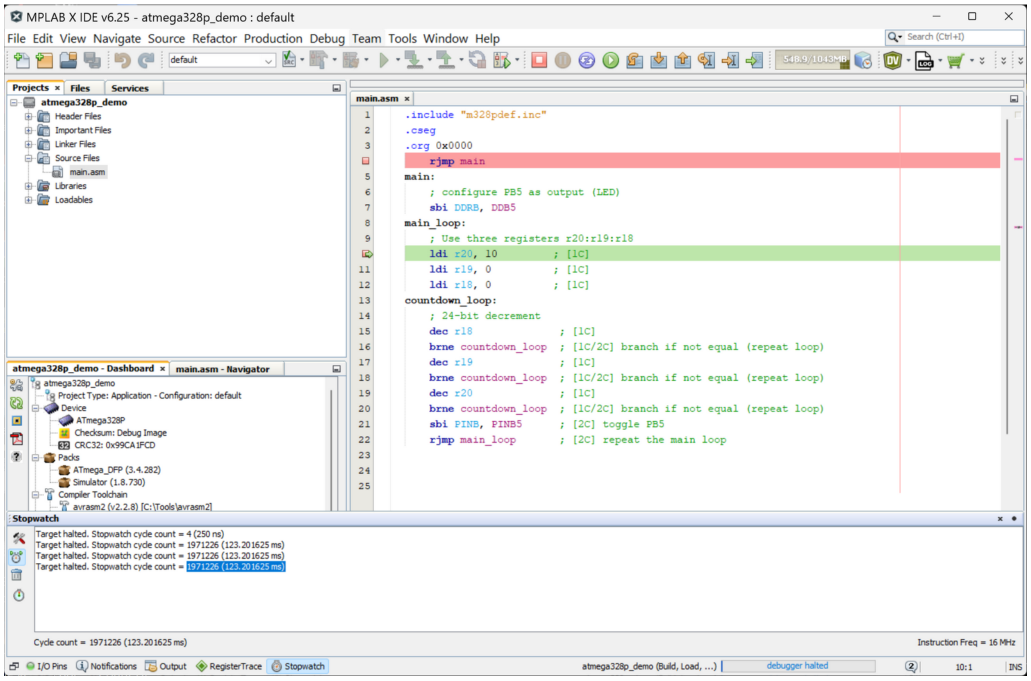
```
.include "m328pdef.inc"
.equ M = 200
.equ N = 200
.equ P = 16
.cseg
.org 0x0000
    rjmp main
main:
    ; configure PB5 as output (LED)
    sbi DDRB, DDB5
main_loop:
    ; Use three registers r20:r19:r18
    ldi r20, P           ; [1C]
    ldi r19, N           ; [1C]
    ldi r18, M           ; [1C]
countdown_loop:
    dec r18              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    ldi r18, M           ; [1C]
    dec r19              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    ldi r19, N           ; [1C]
    dec r20              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    sbi PINB, PINB5      ; [2C] toggle PB5
    rjmp main_loop       ; [2C] repeat the main loop
```

Code Listing 2: AVR assembly code template for Lab 1

```
.include "m328pdef.inc"
.cseg
.org 0x0000
    rjmp main
.equ VALUE = (800000-1)
main:
    ; configure PB5 as output (LED)
    sbi DDRB, DDB5             ; [2C]
main_loop:
    ; Load 24-bit countdown, use three registers r20:r19:r18
    ldi r18, low(VALUE)        ; [1C]
    ldi r19, low(VALUE >> 8)   ; [1C]
    ldi r20, low(VALUE >> 16)  ; [1C]
countdown_loop:
    subi r18, 1                ; [1C] r18 = r18 - 1
    sbci r19, 0                ; [1C] r19 = r19 - C
    sbci r20, 0                ; [1C] r20 = r20 - C
    brne countdown_loop        ; [1C/2C] branch if not equal
    sbi PINB, PINB5            ; [2C] toggle PB5
    rjmp main_loop             ; [2C] repeat the main loop
```

Code Listing 3: AVR assembly code template for Lab 1

Figure 1 shows the simulation-based debugging process in **MPLAB X IDE**, with the **Stopwatch** to show the cycle count during two consecutive breakpoint events.

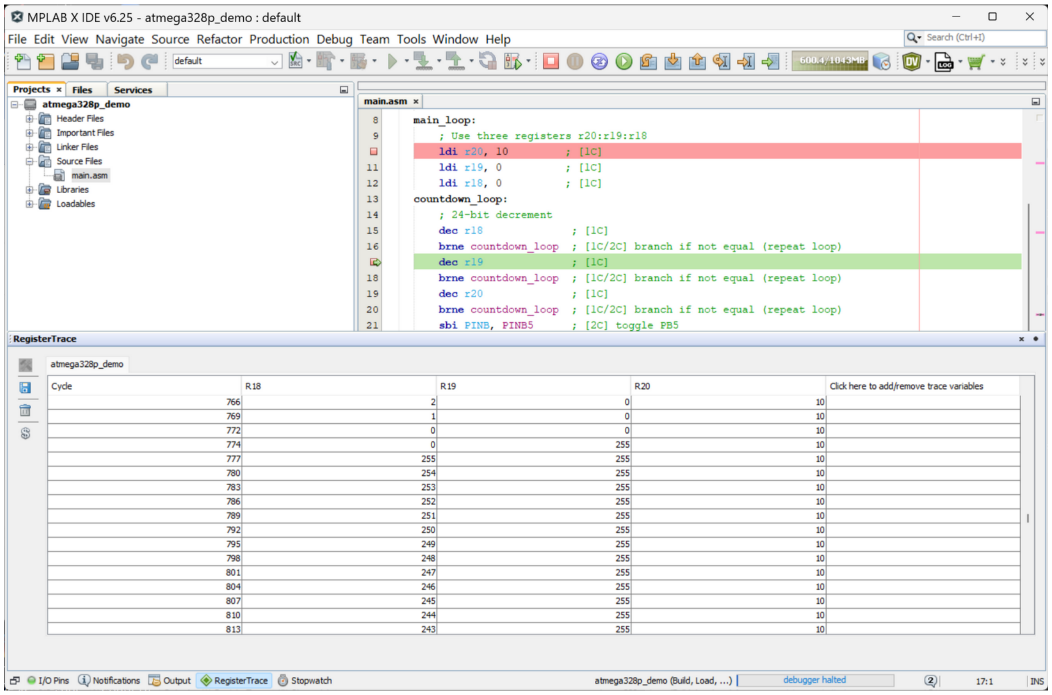


```
.include "m328pdef.inc"
.cseg
.org 0x0000
    rjmp main
main:
    ; configure PB5 as output (LED)
    sbi DDRB, DDB5
main_loop:
    ; Use three registers r20:r19:r18
    ldi r20, 10          ; [1C]
    ldi r19, 0           ; [1C]
    ldi r18, 0           ; [1C]
countdown_loop:
    ; 24-bit decrement
    dec r18              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    dec r19              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    dec r20              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    sbi PINB, PINB5      ; [2C] toggle PB5
    rjmp main_loop       ; [2C] repeat the main loop
```

```
Target halted. Stopwatch cycle count = 4 (250 ns)
Target halted. Stopwatch cycle count = 1971226 (123.201625 ms)
Target halted. Stopwatch cycle count = 1971226 (123.201625 ms)
Target halted. Stopwatch cycle count = 1971226 (123.201625 ms)

Cycle count = 1971226 (123.201625 ms)                    Instruction Freq = 16 MHz
```

Figure 1: AVR program simulation using **MPLAB X IDE**



```
main_loop:
    ; Use three registers r20:r19:r18
    ldi r20, 10          ; [1C]
    ldi r19, 0           ; [1C]
    ldi r18, 0           ; [1C]
countdown_loop:
    ; 24-bit decrement
    dec r18              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    dec r19              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    dec r20              ; [1C]
    brne countdown_loop  ; [1C/2C] branch if not equal (repeat loop)
    sbi PINB, PINB5      ; [2C] toggle PB5
```

atmega328p_demo

Cycle	R18	R19	R20	Click here to add/remove trace variables
766	2	0	10	
769	1	0	10	
772	0	0	10	
774	0	255	10	
777	255	255	10	
780	254	255	10	
783	253	255	10	
786	252	255	10	
789	251	255	10	
792	250	255	10	
795	249	255	10	
798	248	255	10	
801	247	255	10	
804	246	255	10	
807	245	255	10	
810	244	255	10	
813	243	255	10	

Figure 2: RegisterTrace window in MPLAB X IDE

Figure 2 show the **RegisterTrace** window in **MPLAB X IDE**, which records the value changes of selected CPU registers during the execution of the AVR program.

Post-Experiment Questions

1. Explain the differences in how the three AVR assembly programs implement the time delay using a countdown loop.
2. What is the toggle interval in milliseconds (at 16 MHz), or measured in cycle count using the simulation method, for each AVR assembly program?
3. For the AVR assembly program in **Code Listing 2**, answer the following questions:

   - Describe the role of registers r20, r19, and r18 in the nested countdown loop.
   - Calculate the total number of clock cycles executed in one complete iteration of the main loop until PB5 is toggled.
   - Adjust the load values for the countdown registers (**P**, **N**, and **M**) so that the toggle interval is very close to **500 ms**.
4. For the AVR assembly program in **Code Listing 3**, answer the following questions:

   - Calculate the total number of clock cycles executed in one complete iteration of the main loop until PB5 is toggled.
   - Adjust the load value (**VALUE**) so that the toggle interval is very close to **100 ms**.

Hint:

For the AVR program in **Coding List 1**, there are three nested countdown loops and we can calculate the cycle count for the countdown loop as follows:

- complete the first countdown loop: c1 = **256***3 **-** 1 = 767 cycles
- complete the second countdown loop: c2 = **256***(c1 + 3) **-** 1 = 197119 cycles
- complete the third countdown loop: c3 = **10***(c2 + 3) **-** 1 = 1971219 cycles
- complete the main loop: c4 = 3 + c3 + 4 = 1971226 cycles (Total main-loop cycle count)

  The expected toggle interval is 1971226 x 1/16MHz = 123201.625 usec (123.201625 msec).

Lab 2: Toggle the LED by clicking the button

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.
- Write AVR assembly code to implement state-based I/O logic using polling.
- Implement a software debounce delay for a mechanical push button.

Lab Procedure

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

   1.1 Select AVR Assembly as the project type.

   1.2 Select **ATmega328P** as the target device.

   1.3 Open the **main.asm** file for editing.

2. Write AVR assembly code that implements the following functionality:

   2.1 Configure **PD2** as a digital input with the internal pull-up resistor enabled (active-low).
   A push button connected to **PD2** provides the input signal.

   2.2 Configure **PB5** as a digital output connected to the onboard LED.

   2.3 Use a polling loop to continuously read the state of the **PD2** input pin.

   2.4 Detect a valid button click (button press followed by release).

   2.5 Verify that the LED toggles once and only once per button click.

   2.6 Use a software delay subroutine to debounce the push button and avoid false triggering due to mechanical bouncing. Implement a delay subroutine based on a countdown loop for debouncing.

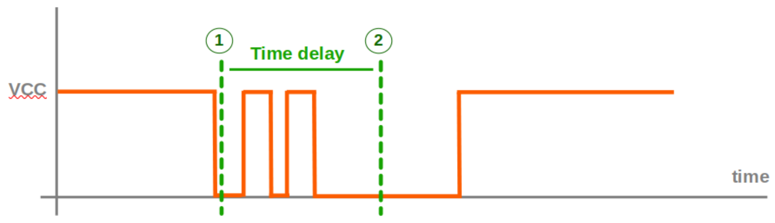


Figure 3: A sample timing diagram for button-input signal

According to the sample timing diagram in **Figure 3**, (1) the MCU detects a signal transition when the input changes from high to low. Some signal bouncing occurs after this transition.
(2) After a fixed time delay (e.g. 10~40msec), the MCU rechecks the input to determine whether it is still logic low. During this delay, rapid high/low transitions caused by contact bounce are ignored. If the signal is still in the pressed state (logic low), it is accepted as a valid button press.

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

   3.1 Manipulate the **PIND** register to simulate the input transitions during the simulation process.

   3.2 Observe bit value changes in **PORTB**.

4. Build the project and upload the generated .hex file to the microcontroller board.

5. Test the program using real hardware

   5.1 Upload the program to the microcontroller board.

   5.2 Connect a push button to **PD2** and observe the LED behavior on **PB5**.

   5.3 Verify that the LED toggles once per button click.

   5.4 Use a digital oscilloscope to capture the input and output signals when pressing the button.

Lab 3: Implement the logical function of an RS latch

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.
- Write AVR assembly code to implement or emulate the RS latch using an MCU.

Lab Procedure

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

   1.1 Select AVR Assembly as the project type.

   1.2 Select **ATmega328P** as the target device.

   1.3 Open the **main.asm** file for editing.

2. Write AVR assembly code in **main.asm** that implements the following functionality:

   2.1 Configure **PD2** and **PD3** as digital inputs (**/R** and **/S,** respectively) with internal pull-up resistors enabled (active-low inputs).

   2.2 Configure **PB4** and **PB5** as digital outputs (**Q** and **/Q**, respectively).

   2.3 Implement RS latch logic with the following behavior:

   - **Reset**: /R = 0, /S = 1
   - **Set**: /S = 0, /R = 1
   - **Hold (no state change)**: /R = 1, /S = 1
   - **Invalid state:** If **/R** and **/S** are both active low, both **Q** and **/Q** are driven high.

3. Simulate the AVR assembly program.

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

   3.2 Manipulate the input pin values (**PD2** and **PD3**) in the simulator and observe the output values to verify correct RS latch behavior.

4. Build the project and upload the generated **.hex** file to the microcontroller board.

5. Test the programs using real hardware

   5.1 Upload the program to the microcontroller board.

   5.2 Connect push buttons to **PD2** (**/R**) and **PD3** (**/S**).

   5.3 Observe the LED behavior on **PB4** (**Q**) and **PB5** (**/Q**).

   5.4 Verify correct RS latch operation for all valid and invalid input combinations.

   5.5 Use a digital oscilloscope to measure the input-to-output propagation delays for any input transition that causes an output transition. Capture waveforms for inclusion in the lab report.

Post-Experiment Questions

1. What is the typical range of input-to-output delay values observed in the experiment?
2. How was the input-to-output propagation delay measured using the digital oscilloscope? Describe the trigger setup and the signals observed.
3. Explain the limitations of simulation-based verification and discuss how these limitations differ from testing on real hardware.