**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Arbab Latif  **Dated:** 28/03/2022

**Semester:** 4th **Section:** BEE 12C

**EE-222: Microprocessor Systems**

Lab 6: Timer Programming (In Assembly)

Group Members

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Reg. No** | **Report**  **Marks / 10** | **Viva**  **Marks / 5** | **Total**  **Marks / 15** |
| Danial Ahmad | 331388 |  |  |  |
| Muhammad Umer | 345834 |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Experiment

# Functions, Branches and Delays

## Objectives

1. Control GPIOs of ATmega16A
2. Implement branches in assembly language
3. Implement calculated delays
4. Break the code down to modular functions

## Equipment

Software

* *Atmel Studio*

Hardware

* ATmega16A microcontroller unit
* Universal Programmer
* Seven Segment Display
* Resistance 47Ω
* LEDs (may use from trainer kit)
* Switch or Button (may use from trainer kit)

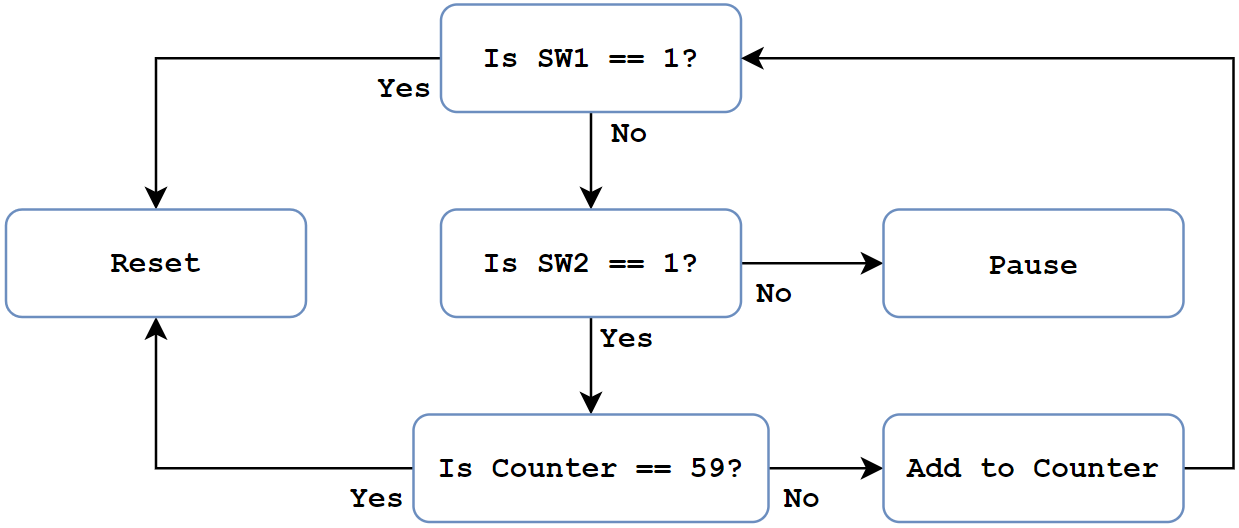
## Introduction

 Atmel Studio IDE is a free development environment for programming Atmel MCUs, sourced by Microchip Technology Inc. It provides us with the means to simulate assembly language codes on specific Microcontrollers and provides an easy and intuitive way of producing .HEX files, which are what makes burning the code on the hardware possible.

In this specific lab, we familiarize ourselves with AVR programming in C language as well as enhancing the applicability of delays through the use of timers. We interface two BCD – 7 Segment displays to create a stopwatch with each increment having a delay of 1s, with reliable accuracy. Common observations include a much concise and a human – readable code relative to programming purely in assembly language.

# Lab Tasks

## Flowchart



## Task A

In this task you are required to create a “Digital Stop Watch” that records the time in seconds precisely (use CTC mode).

1. Connect two 7-segment-Displays with your ATmega16A.
2. Connect two switches (say Sw1 and Sw2) with your ATmega16A.
3. If Sw1 is high, the Stop Watch must get reset to zero, no matter what is the state of Sw2.
4. If Sw2 is high and Sw1 is low, the Stop Watch must display the seconds passed.
5. If Sw2 is low and Sw1 is also low, the Stop Watch must pause its time and if Sw2 is raised again, it should resume from where it was paused

**Code**

**;**

**; Lab6.asm**

**;**

**; Created: 30/03/2022 1:40:44 am**

**; Author : Danial and Umer**

**;**

**; initializing stack pointer**

**.ORG 0x00**

**LDI R31, HIGH(RAMEND)**

**OUT SPH, R31**

**LDI R31, LOW(RAMEND)**

**OUT SPL, R31**

**CLR R31**

**; Setting up ports/pin**

**LDI R31, 0X00**

**OUT DDRA, R31**

**LDI R31, 0XFF**

**OUT DDRB, R31**

**LDI R31, 0XFF**

**OUT DDRC, R31**

**CLR R31**

**; Timer**

**LDI R31,0x00**

**OUT TCNT1H, R31**

**OUT TCNT1L, R31 ;TCNT1 = 0**

**CLR R31**

**; Output Compare**

**LDI R20, 0X3D**

**OUT OCR1AH, R20**

**LDI R20, 0X09**

**OUT OCR1AL, R20 ; OCR1A = 0X3D09**

**LDI R20, 0x00**

**OUT TCCR1A, R20 ; WGM11:10 = 00**

**LDI R20, 0x0B**

**OUT TCCR1B, R20 ; WGM13:12 = 01, CTC mode, prescaler = 64**

**; Initially, display 00**

**LDI R16, 0 ; Counter**

**CALL SevenSegment**

**OUT PORTB, R17**

**OUT PORTC, R17**

**START:**

**IN R18, PINA**

**CPI R18, 2**

**BRSH RESET**

**CPI R18, 0**

**BREQ PAUSE**

**CPI R18, 1**

**BREQ INCREMENTER**

**RJMP START**

**INCREMENTER:**

**AGAIN0:**

**IN R20, TIFR ; read TIFR**

**SBRS R20, OCF1A ; if OCF1A is set skip next instruction**

**RJMP AGAIN0**

**LDI R20, 1<<OCF1A**

**OUT TIFR, R20 ; clear OCF1A flag**

**CPI R16, 59**

**BREQ OF**

**CONT:**

**INC R16 ; Store and save the current counter value**

**STS 0X400, R16**

**CALL SPLITTER ; To split a decimal number into units and tens**

**MOV R16, R24**

**CALL SevenSegment**

**OUT PORTB, R17**

**MOV R16, R23**

**CALL SevenSegment**

**OUT PORTC, R17**

**CLR R23**

**CLR R24**

**LDS R16, 0X400 ; Reload counter**

**RJMP START**

**OF:**

**LDI R16, 0**

**CALL SevenSegment**

**OUT PORTB, R17**

**OUT PORTC, R17**

**AGAIN1:**

**IN R20, TIFR ; read TIFR**

**SBRS R20, OCF1A ; if OCF1A is set skip next instruction**

**RJMP AGAIN1**

**LDI R20, 1<<OCF1A**

**OUT TIFR, R20 ; clear OCF1A flag**

**RJMP START**

**RESET:**

**LDI R16, 0 ; As R16 gets auto incremented by 'INCREMENTER'**

**CALL SevenSegment**

**OUT PORTB, R17**

**OUT PORTC, R17**

**RJMP START**

**PAUSE:**

**IN R18, PINA**

**CPI R18, 0**

**BREQ PAUSE**

**RJMP START**

**.ORG 0X100**

**; Subroutine for sending BCD decoder output upon input**

**SevenSegment:**

**CPI R16, 0**

**BRNE A**

**LDI R17, 0X01**

**RET**

**A:**

**CPI R16, 1**

**BRNE B**

**LDI R17, 0X4F**

**RET**

**B:**

**CPI R16, 2**

**BRNE C**

**LDI R17, 0X12**

**RET**

**C:**

**CPI R16, 3**

**BRNE D**

**LDI R17, 0X06**

**RET**

**D:**

**CPI R16, 4**

**BRNE E**

**LDI R17, 0X4C**

**RET**

**E:**

**CPI R16, 5**

**BRNE F**

**LDI R17, 0X24**

**RET**

**F:**

**CPI R16, 6**

**BRNE G**

**LDI R17, 0X20**

**RET**

**G:**

**CPI R16, 7**

**BRNE H**

**LDI R17, 0X0F**

**RET**

**H:**

**CPI R16, 8**

**BRNE I**

**LDI R17, 0X00**

**RET**

**I:**

**CPI R16, 9**

**BRNE J**

**LDI R17, 0X0C**

**RET**

**J:**

**RET**

**.ORG 0x200**

**SPLITTER:**

**MOV R23, R16 ; Unit**

**CLR R24 ; Tens**

**LOOP:**

**CPI R23, 10**

**BRLO FINISH ; If Unit < 10, end**

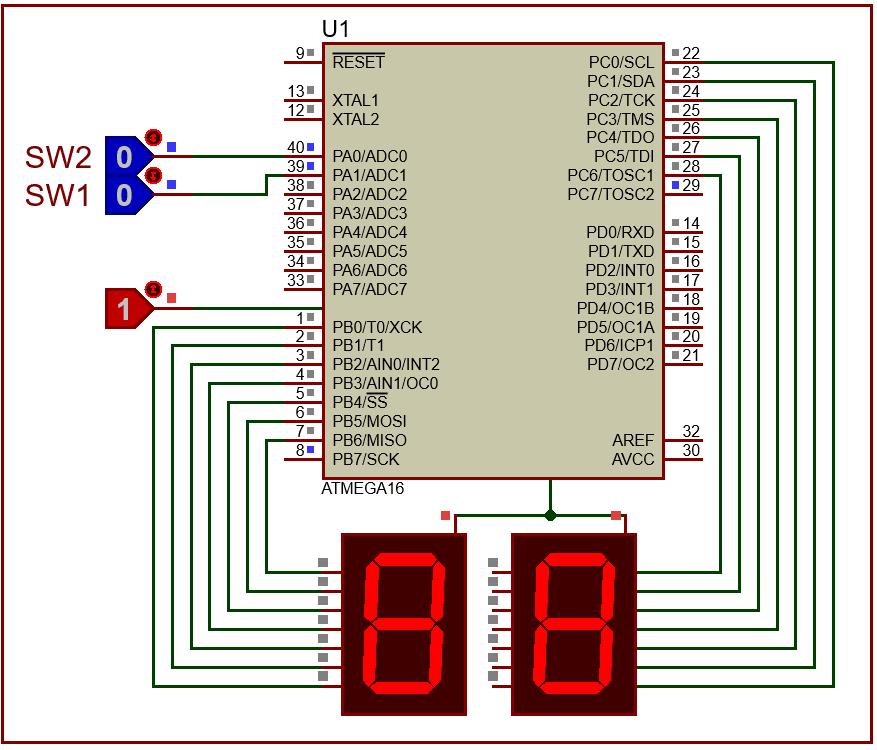
**SUBI R23, 10 ; Unit = Unit - 10**

**INC R24 ; Ten = Ten + 1**

**RJMP LOOP**

**FINISH:**

**RET**



**Proteus Simulation**

## Conclusion

After the conduction of this lab, we have learnt how to utilize the timer registers in AVR and implemented a stop watch through means of a simple program and two BCD – 7 Segment decoders. We observe that implementing delays in such a way results in a concise and much readable code, but perhaps the biggest advantage of timers is the accuracy in delays relative to coding through nested loops.