Lab 3: Functions, Branches & Delays

EE222: Microprocessor Systems

Contents

1 Acknowledgements

This lab exercise is prepared by Mohammad Azfar Tariq and Muhammad Usman under the supervision of Dr. Rehan Ahmed for the course EE-222 Microprocessor Systems. Reporting any error or discrepancy found in the text is appreciated.

2 Administrivia

2.1 Objective

By the end of this lab you will be able to

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

2.2 Deliverable

You are required to submit

- Appropriately Commented Code
- Image of assembled hardware
- Comment on the issues you faced in Developing the Solution.

in the beginning of next lab

2.3 Hardware Resources

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

3 Introduction

3.1 AVR subroutines

When a task is to be performed repetitively, programmers usually create a separate code block to write the respective code only once in it and jump to that block as many times, the task is to be performed. Such programming practice saves the cost of writing the same piece of code again and again. In the context of assembly language, it is called "subroutine".
In AVR special instructions are present to jump and return back from a subroutine which are `CALL, RCALL, RET,`, etc. Details of these instructions can be read from Microchip AVR ISA Manual[1]. For example a subroutine that adds 4 to R16 can be written as,

```
.ORG 0x00
    LDI   R16, 0x11
    CALL  PlusFour

plusFour:
    SUBI  R16, -4
    RET
```

3.2 AVR Stack

Stack is a data structure which follows the rule of LIFO (Last In First Out). In AVR we can implement a stack using RAM and SP (Stack Pointer) register specified by the AVR which points to the top of stack. SP is a 16-bit register whose upper 8-bits can be accessed through SPH and lower through SPL. Using `PUSH` and `POP` instruction we can load and unload data from stack. The internal circuitry automatically subtracts 1 from SP when a data chunk is pushed over stack and adds 1 after it is unloaded. Thus, the stack must be initialized at some higher memory location so that when the data is pushed over it, the SP can be decremented safely otherwise if it is initialized say at location 0, after first push, the stack pointer will become -1 which is not a memory address. For this purpose we usually initialize the stack with the last location of RAM. For example to push 0x11 and 0x12 on stack we may use the following code,

```
.ORG 0x00
    LDI   R16, HIGH(RAMEND)    ; Initialize Stack Pointer
    OUT   SPH, R16
    LDI   R16, LOW(RAMEND)
    OUT   SPL, R16
    LDI   R16, 0x11
    PUSH  R16
    LDI   R16, 0x12
    PUSH  R16
```

Whenever a function is called, the address of instruction after the call is pushed on the stack and when RET instruction is executed, the address is popped back to program counter so "**for a safe function call, you must set the stack pointer to RAMEND in the start of program**".

[1] https://www.microchip.com/webdoc/avrassembler/avrassembler.wb_instruction_list.html

3.3 ATmega16A Clock

ATmega16A has many options for clock. We can provide clock to it through external crystal or RC oscillator. However, an internal RC oscillator is also present which can be calibrated to provide clock of 1,2,4 or 8 MHz. The factory default configurations have already set the internal RC oscillator to provide a **"1MHz"** clock. In this lab we will be using ATmega16A with factory default configurations. For more information see chapter 8 in data sheet[2].

3.4 Delay Calculation

In delay calculation one must keep in view two things,

1. CPU clock cycle frequency (F_{cpu})
2. Instruction cycles consumed

$$Delay\ time = (no.\ of\ instruction\ cycles)/F_{cpu}$$

3.5 Digital I/O

There are four I/O ports in ATmega16, A, B, C and D. Three registers are specific to each port,

- Data Direction Register (`DDRx`).
- Port Output Register (`PORTx`).
- Port Input Register (`PINx`)

`DDRx` register decides wether the port pins will act as input or output. If any bit of `DDRx` is loaded with 1, the corresponding pin of the port will be activated in output mode and vice versa for input mode. For example to activate pins of Port B as output pins, we have to load 0xFF in `DDRB`.
To send some binary value out at the pins (which are set as output by `DDRx`), we have to store it in `PORTx` register. For example to send 0xAA out at the pins of port B, we have to store it in `PORTB`.
To read the status of pins (high or low) which are set as input by `DDRx`, we have to read register `PINx`. For example, if we have to read what logic values are present at the pins of port B, we must read the values in `PINB` register, they will be same.
Under normal condition a pin in a port cannot perform dual operations (–that is input and output at the same time).
Use specific instructions "`IN` and `OUT`" to read and deliver data to I/O registers.

[2]http://ww1.microchip.com/downloads/en/devicedoc/atmel-8154-8-bit-avr-atmega16a$_d$*atasheet.pdf*

3.6 Bit Addressing

AVR ISA provides a set of special instructions to manipulate individual bits in the registers. A few of these instructions are,

Instruction	Description
`SBI R, n`	Set bit number n in the register R to 1
`CBI R, n`	Clear bit number n in the register R.
`BST R, n`	Store bit number n from register R to T-Flag
`BLD R, n`	Load bit number n of register R with the value of T-Flag

For further explanation of these instructions see Microchip AVR ISA Manual[3]

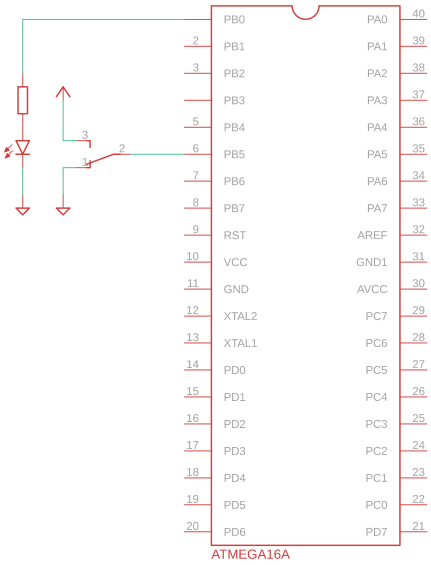
[3]https://www.microchip.com/webdoc/avrassembler/avrassembler.wb$_{i}nstruction_{l}ist.html$

4 Lab Tasks

4.1 Controlling LED

4.1.1 Task A

Turn an LED on or off by sensing a switch connected to a pin as schematic suggests.



Logic low at pin should turn off the LED while logic high should immediately turn it on.

4.2 Delay Function

4.2.1 Task B

Implement a delay function for 500 milliseconds delay. Write an assembly program that utilizes this function to turn on an LED and wait 500ms and then turn it off and wait 500ms repeatedly in a loop. Include your delay calculations in the report.

4.3 Subroutines and Stack

4.3.1 Task C

The following program stores seven characters in memory. Write a subroutine named CHECK_PALINDROME that return 1 in R16 if this seven letter word is a palindrome else return 0 in R16. Utilize the property of AVR stack in your function.
Replace these seven characters with "ROTATOR" and then with "JUMPOFF" to check wether your function is general or not.

```
.ORG 0x00
    LDI R16, 'R'
    STS 0x200, R16
    LDI R16, 'A'
    STS 0x201, R16
```

```
6       LDI R16, 'C'
7       STS 0x202, R16
8       LDI R16, 'E'
9       STS 0x203, R16
10      LDI R16, 'C'
11      STS 0x204, R16
12      LDI R16, 'A'
13      STS 0x205, R16
14      LDI R16, 'R'
15      STS 0x206, R16
16
17      CALL CHECK_PALINDROME
18  Here: RJMP Here
19
20  .ORG 0x50
21  ;------ Your Code Here------
22  CHECK_PALINDROME:
23
24  ;----------------------------
```