Lab 2: Introduction to Atmel Studio 7 and Basic I/O Assembly Language Programming

CompEng 3151: Digital Engineering Lab II

Last revised: July 16, 2019 (BJZ, RJS)

**Description/Overview**

As you have learned about in your lecture course, the AVR microcontroller is an advanced version minicomputer integrated on a small chip having a processor, memory and programmable input/output peripherals. The main function of an AVR microcontroller is to provide digital control over any type of system (electrical, mechanical or automotive), different devices, industrial plants, and most electronic gadgets and appliances.

All microcontrollers contain flash ROM memory, which is used to store programs that they execute. In order to create program code and write it to the flash memory, some kind of assembler, compiler, and/or programmer is needed. In this course, we will use Atmel Studio 7 to write programs, debug them and then program and run them on the microcontroller board. This lab serves as an introduction to the software and the AVR Simon Board we will use throughout this course to learn about AVR microcontrollers.

**Objectives**

Students will:

1) Become familiar with the functionality of Atmel Studio 7 and how to create an assembly project for an AVR microcontroller
2) Know how to use the built-in debugger and simulator tools to troubleshoot programs
3) Create an Assembly program that uses pushbutton inputs to control LED outputs
4) Implement delay loops/polling loops

**Materials Required**

- Atmel Studio 7 Installed on PC (ECE CLC Computers will have this)
- AVR Simon Board with a USB cable

**Preparation**

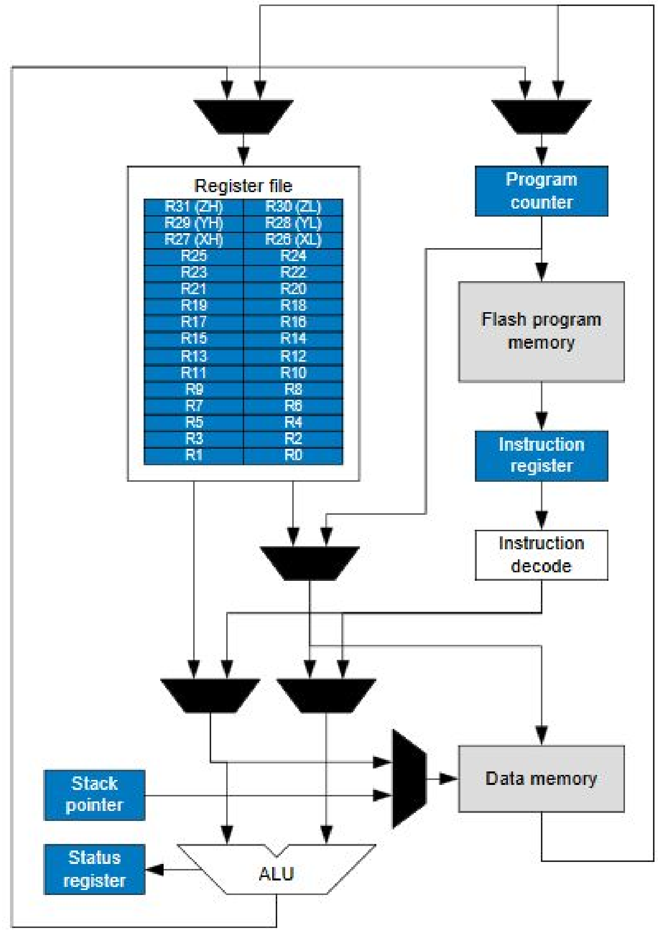
In order to be successful with this lab, students should review Chapters 2 & 3 in the Mazidi textbook.

**Background**

Generally speaking, a microcontroller is a "computer-on-a-chip." In addition to the registers file and all the arithmetic and logic elements (the CPU), the device also incorporates read-only and read-write memory (ROM and RAM), input/output interfaces (I/O ports), and timers.

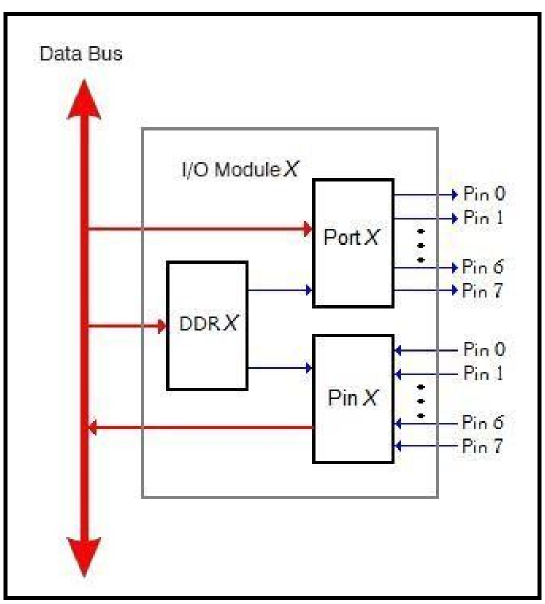
Microcontrollers are frequently used in automatically controlled products and devices, such as automobile engine control systems, office machines, appliances, power tools, and toys.

*Architecture*




AVR microcontrollers are designed after a Harvard architecture, i.e., data memory and program memory are physically separated and signals travel over different buses.

This is the block diagram of the ATmega324PB chip, which is the one we will be using throughout the lab. It is important to mention that in this architecture, instructions are 16-bits long (2 bytes), but only 8-bits (1 byte) are used for data. Therefore, results and operands connect to an 8-bit bus that, in turn, connects with memory and input/output lines. In contrast, instructions travel on a 16-bit bus, which is connected with any instruction-related components like the program memory, the instruction decoder, or the ALU.




*Port Modules*

The microcontroller interfaces with external hardware through ports, a collection of wires and registers to send or detect digital signals. The actual point where hardware and microcontroller connected physically is known as a pin.

The ATmega324 has bidirectional pins, that is, each one can be programmed to work as either data input or as data output. To allow this flexible behavior, they are associated with the three registers, which form an I/O module.

The ATMega324 has 5 ports, each with 8 I/O pins. Ports are identified with the letters from A to E. The figure at the left represents the general structure of any port in the device.

DDRA (Data Direction Register) configures port A through the registers Port A and Pin A. The 8-bit value in DDR indicates what pins will be used for data input (pin register) and what pins will be used for data output (port register). Thus, when the appropriate command is issued, Port A gets data from the data bus and writes it to any pin that has been set as output in DDRA. In a similar way, Pin A gets data from any pin set as an input in DDRA and writes it to the data bus. All of the ports on the AVR are configured the same way using DDRA – DDRE.

*Instruction Set*

The table below summarizes the AVR architecture instruction set:

| Mnemonic | Description | Mnemonic | Description | Mnemonic | Description |
|---|---|---|---|---|---|
| **Flow Control** | | **Bit Manipulation** | | **Load/Store** | |
| JMP | Jump absolute (24-bit) | SEC/CLC | Set/clear C flag (carry) | MOV | Copy register to register |
| RJMP | Branch relative (12-bit) | SEH/CLH | Set/clear H flag (half carry) | LD | Load indirect through X/Y/Z |
| IJMP | Jump indirect (Z) | SEN/CLN | Set/clear N flag (negative) | LD | Load indirect with postincremnt |
| RCALL | Call subroutine | SEZ/CLZ | Set/clear Z flag (zero) | LD | Load indirect with predecremnt |
| ICALL | Call subroutine indirect (Z) | SEI/CLI | Set/clear I flag (interrupt) | LDD | Load indirect with 6-bit offset |
| RET/RETI | Return/from interrupt | SES/CLS | Set/clear S flag (sign) | LDI | Load 8-bit immediate |
| CP/CPC | Compare/with carry | SEV/CLV | Set/clear V flag (overflow) | LDS | Load from 16-bit address |
| CPI | Compare with 8-bit immediate | SET/CLT | Set/clear T bit | LPS | Load from program space |
| CPSE | Compare, skip if equal | SBR/CBR | Set/clear bit in register | ST | Store indirect through X/Y/Z |
| SBRS/SBRC | Skip if register bit set/clear | BSET/BCLR | Set/clear bit in status register | ST | Store indirect with postincremnt |
| SBIS/SBIC | Skip if I/O bit set/clear | SER/CLR | Set/clear entire register | ST | Store indirect with predecremnt |
| BRcc | Conditional branch | SBI/CBI | Set/clear bit in I/O space | STD | Store indirect with 6-bit offset |
| **Logical** | | **Arithmetic** | | STS | Store to 16-bit address |
| AND | Logical AND | ADD/ADC | Add/with carry | IN/OUT | Input/output to/from I/O space |
| ANDI | Logical AND 8-bit immediate | ADIW | Add 6-bit immediate | PUSH/POP | Push/pop stack element |
| OR | Logical OR | SUB/SUBC | Subtract/with borrow | BLD/BST | Load/store T bit |
| ORI | Logical OR 8-bit immediate | SBIW | Subtract 6-bit immediate | **Miscellaneous** | |
| EOR | Logical exclusive-OR | SUBI/SBCI | Subtract 8-bit imm/w borrow | NOP | No operation |
| LSL/LSR | Logical shift left/right by 1 bit | INC/DEC | Increment/decrement register | SLEEP | Wait for interrupt |
| ROL/ROR | Rotate left/right by 1 bit | MUL | Multiply 8 × 8 → 16 | WDR | Watchdog reset |
| ASR | Arithmetic shift right by 1 bit | | | | |
| COM/NEG | One's/two's complement | | | | |
| SWAP | Swap nibbles | | Can use R16–R31 only | | |
| TST | Test for zero or minus | | Can use R24–R31 only | | |

For more information on what each of these instructions does or how they work, you should refer to your textbook or the Atmel 8-bit AVR Instruction Set manual.

**Procedure**

1) *Atmel Studio Tutorial*
    a. In order to become familiar with the basics of Atmel Studio 7 and its debugging functionality, you should complete the Assembly Programming in Atmel Studio 7 – Step by Step Tutorial.

2) *Delay Loops*
    a. The code you created during the tutorial was written to toggle all of the LEDs connected to Port D on and off. After programming the AVR Simon Board with the code in the tutorial, you may notice that the LEDs don't seem to turn on and off. They actually are turning on and off but because of the lack of any delay, the LEDs are flashing very quickly. In order to fix this problem, you need to create a delay subroutine into your assembly code so you can actually

observe the LEDs flashing on and off. Since a register can only hold a value up to 255, you will probably need to use nested loops to do this and CALL it when you need to use it.

3) *Pushbutton Control of LEDs*
   a. Up until now, we have statically programmed the LEDs to turn on and off based on switching PORT D on and off depending on what part of the code it is in. For this last part of the lab, you should choose four of the push buttons on the board and light up their respective LEDs when their button is pushed. The LEDs should remain lit for approx. 1 second before turning off.

**Deliverables**

For this lab, you will submit the three different programs you wrote. Each of these programs should be thoroughly documented so that someone else who has not completed this activity can follow your code and understand how your solution works. Make sure each of your programs has an appropriate comment block at the top of the file (see MST Computer Science Department's Coding Standards). In future labs, you will be required to submit a lab report.

**Questions/Observations**

1. Are the LEDs active low or active high? Did your code use a 0 or a 1 to turn the LED on? Verify your answer on the Simon Board datasheet to determine whether a logical 0 or 1 turns on the LED.
2. Are the push buttons active low or active high? Did your code use a 0 or a 1 to indicate that a button has been pushed? (Or, what is the constant value of the button when it is not pushed?) Verify your answer on the Simon Board datasheet to determine whether a logical 0 or 1 is used to indicate a button push.
3. On the Simon Board datasheet, list which port pin each LED is connected (for all 9 LEDs).

**References**


- Muhammad Ali Mazidi, Sarmad Naimi, and Sepehr Naimi. 2010. *AVR Microcontroller and Embedded Systems: Using Assembly and C (1st ed.)*. Prentice Hall Press, Upper Saddle River, NJ, USA.
- Vasconcelos, Jorge, 2009. *Brief Guide to the AVR Architectures and the AT90USBkey Demonstration Board.* John Hopkins University, Department of Computer Science.