EE2016 - Microprocessor Theory + Lab

Experiment-4: Introduction to AVR Assembly Language Programming

1 Introduction

In this experiment, we will introduce the AVR ATmega8 microcontroller from Atmel and learn to program the microcontroller using assembly language.

The ATmega8 is an 8-bit RISC single-chip microcontroller developed by Atmel. AVR was one of the first microcontroller families to use on-chip flash memory for program storage. Some features of the AVR microcontroller are general-purpose I/O ports, internal instruction flash memory, internal SRAM up to 16 KB, timers etc.

Since flash memory, SRAM and other units are all integrated on to a single chip, many applications can be run without dedicated external memory. Program instructions are stored in flash memory.

2 Programming Environment

The various tasks involve writing assembly language programs. To run these programs, you need to use *Microchip Studio* software on the PCs in the lab. As with most other packages, you need to open a new project. The remaining steps are as follows.

- Choose *Assembler* and specify the folder to save.
- In device list, search for *Atmega8*.
- Write the program and save as *main.asm*.
- Go to *Build* and then select *Build the solution*.
- Go to *Debug* and click on *Step Into* for step-by-step debugging.
- Use the *Processor Status* window (select *Debug* and then *Windows* $--> $ *Processor Status*) to see the contents of the registers.

Note: We may need to do a few additional steps if *Simulator* has not been selected in advance. In particular, after the *Debug* step above, if you get a window with the message: *Please select a connected tool and interface and try again*, do the following:

- Select *Simulator* in *Selected debugger/programmer* (in the window that pops up)
- Then click on *Build* and select *Build a solution*

3 Tasks for the Experiment

Various instructions and their format are in Table 1. You need to identify the ones that are appropriate for performing the tasks listed.

Task 1: Write a program to find the (i) maximum and (ii) minimum of 10 numbers stored in flash. Your numbers will be part of the code. Store the output suitably (in some registers).

Task 2: Consider 10 numbers in flash memory. Add all these numbers and store the result in a register. Assume the numbers are small (so that a register can hold the sum).

Task 3: Sort 5 numbers stored in flash memory in arbitrary order and write the final results to data memory (show the results in the memory window). *Hint:* To perform sorting, you may use five registers.

Task 4: Submit a report on the experiment on Moodle (within a week of this experiment). One report per group (with the names and roll numbers of the group members) is sufficient. The report should contain details of the solution (including the code) and your observations and experience in programming and debugging. Please note that reports that closely match those of other groups will be penalized.

Instruction	Description	Operation
Arithmetic instructions		
ADD Rd,Rr	Add without Carry	Rd ← Rd+Rr
ADC Rd,Rr	Add with Carry	Rd ← Rd + Rr + C
ADIW Rd, K	Add Immediate to Word	Rd+1:Rd ← Rd+1:Rd + K
SUB Rd, Rr	Subtract without Carry	Rd ← Rd - Rr
SUBI Rd, K	Subtract Immediate	Rd ← Rd-K
SBC Rd, Rr	Subtract with Carry	Rd ←Rd-Rr-C
SBCI Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C
SBIW Rd, K	Subtract Immediate from Word	Rd+1:Rd ← Rd+1:Rd - K
Logic instructions		
AND Rd, Rr	Logical AND	Rd ← Rd · Rr
OR Rd, Rr	Logical OR	Rd ← Rd + Rr
EOR Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr
COM Rd	One's Complement	Rd ← FF-Rd
NEG Rd	Two's Complement	Rd ← 00 - Rd
INC Rd	Increment	Rd ← Rd + 1
DEC Rd	Decrement	Rd ← Rd - 1
TST Rd	Test for Zero or Minus	Rd ← Rd · Rd
CLR Rd	Clear Register	Rd ← Rd ⊕ Rd
SER Rd	Set Register Rd	← FF None 1
MUL Rd,Rr	Multiply Unsigned	R1, R0 ← Rd Rr
LSL Rd	Logical Shift Left	Rd(n+1) ← Rd(n),Rd(0) ← 0,C ← Rd(7)
LSR Rd	Logical Shift Right	Rd(n) ←Rd(n+1),Rd(7) ← 0,C ← Rd(0)
ROL Rd	Rotate Left Through Carry	Rd(0) ←C,Rd(n+1) ← Rd(n),C ← Rd(7)
ROR Rd	Rotate Right Through Carry	Rd(7) ← C,Rd(n) ← Rd(n+1),C ← Rd(0)
ASR Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1),$n = 0 \cdots 6$
SWAP Rd	Swap Nibbles	Rd(3..0) << Rd(7..4)
Branch and Jump instructions		
RJMP k	Relative Jump	PC ← PC + k + 1
IJMP	Indirect Jump to (Z)	PC ← Z
JMP k	Jump	PC ← k
RCALL k	Relative Call Subroutine	PC ← PC + k + 1
ICALL	Indirect Call to (Z)	PC ← Z
CALL k	Call Subroutine	PC ← k
RET	Subroutine Return	PC ← STACK
CPSE Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3
CP Rd,Rr	Compare	Rd - Rr
CPC Rd,Rr	Compare with Carry	Rd - Rr - C
SBRC Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3
BREQ k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1
BRNE k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1
Load and Store instructions		
MOV Rd, Rr	Copy Register	Rd ← Rr
LDI Rd, K	Load Immediate	Rd ← K
LDS Rd, k	Load Direct from SRAM	Rd ← (k)
LD Rd, X	Load Indirect	Rd ← (X)
STS k, Rr	Store Direct to SRAM	(k) ← Rr
ST X, Rr	Store Indirect	(X)← Rr
LPM	Load Program Memory	R0 ← (Z)
IN Rd, P	In Port	Rd ← P
OUT P, Rr	Out Port	P ← Rr
PUSH Rr	Push Register on Stack	STACK ← Rr
POP Rd	Pop Register from Stack	Rd ← STACK

Table 1: Instructions and Their Format