By Sarwan Shah
ID: ss03595
Section: T2 (Junaid Memon)

Microcontrollers and Interfacing
Submitted: 3/3/2019

Assignment 1 – Report

Abstract:

This report aims to provide an analysis of the code and procedure used to perform 8-bit arithmetic operations on a sequence of 10 value pairs, as indicated by another sequence of 10 arithmetic operators (+, -, *, /) in ASCII; all stored in the ATMEGA328 Program Flash Memory (ROM) in sequential manner, starting at the address 0x200. Once the values have been operated upon, the results are stored in sequential manner in the Internal Ram of the ATMEGA328 starting at address 0x100.

We will assume the following data set for explaining the procedure:

DATA1: 96,70,47,30,42,30,10,48,100,47
DATA2: '+','-','+','-','*','/','+','-','*','/'
DATA3: 3,4,7,9,2,3,1,1,10,0

Procedure and Results:

Understanding How ROM Data is organized:

The first task was to extract from ROM the data values, which include the two numeric values and the ASCII character representing the operation to be performed between them. The data is stored in such a manner that if 96, from DATA 1 is stored at 0x200, then '+', from DATA 2 is stored at 0x200 + 0x0A i.e. 10 locations ahead. Similarly, '3' from DATA3 is stored at 0x200 + 0x14 i.e. locations ahead. The Z pointer is used to access this data in the ROM and the X pointer is used to store data to IRAM.

Note: The AVR actually stores the ROM data at 0x400 since each ROM location is 2-byte sized chunks, and when accessed using the Z register, the address stored in Z needs to be bit shifted to the left by one, since the first bit signifies whether the high bit or the low bit needs to be accessed.

Extraction of Data:

The Z pointer is used such that, the values from the same index in all three data sets are extracted and stored in General Purpose Registers (by getData routine), operated upon (by operateData routine), and then the results are stored in the IRAM, before moving to the next set of values.

A main loop, that runs the same number of times (GPR r20) as the data set size helps repeat this process over for the 10 perceptively parallel values. After each loop run, the addresses pointed by the Z and X pointers are incremented such that they point to the next set of perceptively parallel values in the Data Set when accessed. This achieved using an 'incrementerZ' and 'incrementerX' function that increment these pointers in an inversely proportional relation to the number of times the main loop has run, as the operation of these functions depends on the value stored in GPR r20. Hence, these pointer land at

sequential/incremental positions after each loop run. The working of the 'incrementer' routines has been explained and demonstrated in code's comments.

Operation of Data:

Once the data has been extracted from the ROM and stored to GPR's, selecting which operation to perform is done by the 'operateData' routine. It compares the decimal values of the already extracted operator characters, with the ASCII decimal values of the relevant operations. If there is a match, it branches to the relevant operation routine that performs the subsequent operation on the two numeric values already extracted from ROM, now stored in GPR's.

For addition, subtraction and division the results are initially stored in to r17, and then moved to the IRAM using the 'storeData' routine, whereas the multiplication results are stored in r0 and r1, since the result can be 16-bits (i.e. 100 * 10), before being moved to the IRAM using the special 'storeData16bit' routine.

Storage of Data:

The 'storeData' routine makes use of the X pointer, incrementerX routine and ST operation to move data from GPR r17 to the relevant IRAM location. It works in a similar manner, as the 'getData' routines.

The 'storeData16bit', helps increments the X pointer an additional step than usual for the accommodating an additional byte (stored in r1), if the multiplication result is greater than one byte, in which case GPR's r0 and r1 are moved to their relevant sequential positions in the IRAM.

Arithmetic Operations:

Addition, Multiplication and Subtraction are achieved using built-in AVR instructions.

Division is achieved using repeated subtraction, until a remainder of 0 is achieved, using branching instructions. The procedure assumes that the numbers perfectly divide each other. In the case that either of the two numbers is a zero, the resulting answer is a zero.