Laboratory 4

Format Conversion

Due Date: Beginning of Week 6 Lab

Introduction

The first demonstration in this lab shows why one should multiply before dividing instead of dividing before multiplying. The programming assignment involves several different methods of representing numbers in memory.

We have mostly been using hexadecimal to store values in memory. This method requires the fewest number of bits to store a given value. For example, storing 65535_{10} in memory appears as "FF FF" and requires two bytes. However, this is not a convenient format for reporting the number to a human user.

One common user-friendly format is binary coded decimal (BCD). This method stores each digit of the decimal number as a four-bit value, two digits per byte. This makes the value being stored appear to a human as the decimal representation but at a cost of more space. Storing the value 65535_{10} in memory appears as "06 55 35". This format is often used in conjunction with 7-segment displays.

Another common format is ASCII (American Standard Code for Information Exchange). In ASCII, each item is given a 7-bit code that is often stored as a two-digit (1-byte) hexadecimal number. ASCII has codes assigned to letters, numbers, and non-printable items like backspaces and carriage returns. This method is used to communicate with devices like text-based terminals and printers. Note that the other systems represent the value of the number being stored while ASCII represents the appearance of the number. In the hexadecimal representation for the ten numbers in ASCII, the first digit is a "3" and the second digit is the number. For example, storing 65535_{10} in memory appears as "36 35 35 33 35".

A "null-terminated" string is one that ends with the byte $00. The value $00 is a non-printable character call "null", and this value is used as a special byte to indicate the end of the string (much like an earlier program that counted $64's used $FF to indicate the end of an array). Many ASCII strings are stored as null-terminated strings so that the length does not need to be stored also. For example, storing 65535_{10} as a null-terminated string in memory appears as "36 35 35 33 35 00".

Assignment

The importance of ordering multiplication before division is shown in the following section. The assembly files mentioned are available on Blackboard. Each implements a version of the exponential filter discussed in Lecture 12.

- Download the files multdiv.asm and divmult.asm to your thumb drive, assemble them, and download them to the Star12.
- Using the information in the assembly files, run each program. You should not have to enter any inputs.
- Enter the command **md 1000 102f**.
- The 16 values on the line beginning with $1000 are the unfiltered input values. The line beginning with $1010 contains filtered values from the exponential filter that multiplies before dividing (from multdiv.asm), and the line beginning with $1020 contains filtered values from the exponential filter that divides before multiplying (from divmult.asm).
  Question 1: On the 10^{th} sample (addresses $1009, $1019, and $1029), what is the percentage difference between the input and each of the filtered versions?
- For each of the two filter programs, generate a breakpoint trace by placing breakpoints at the "ABA" instruction in each program (You will have 16 lines per program). You do not need to submit the traces in the lab report.
  Question 2: Using the data from the two traces, explain briefly but specifically why one version more closely follows the input.

Write a flowchart and the corresponding assembly program that meets the following requirements. Demonstrate the working program to the instructor.

1. The program converts the 2-byte hexadecimal number to a 3-byte BCD number. The **address** of the hexadecimal number is supplied in addresses $1000 and $1001, and the **address** of the BCD number is supplied in addresses $1002-$1003.
2. The program converts the 2-byte hexadecimal number to a 6-byte ASCII string. The **address** of the hexadecimal number is supplied in addresses $1000 and $1001, and the **address** of the BCD number is supplied in addresses $1002-$1003.
3. After the last ASCII byte that represents a number the program must store $00. For example, if $00FF is supplied as the hexadecimal number, your program should store the bytes "30 30 32 35 35 00".
4. 10 points Extra Credit = Minimize the length of the generated ASCII string. When converting non-zero values, do not output leading 0's. For example, if $00FF is supplied as the hexadecimal number, your program should store only the bytes "32 35 35 00". If the value is 0, the ASCII string should be "30 00".

What to Demonstrate/Submit

- Answers to questions.
- Listing file of the number formatting program.
- Instructor check-off.