ECE-3226/CSCI-3451 - Lab #3: Control Flow, Arrays, and Strings

Contents:

Overview

Purpose: The lab serves two purposes. First, it will provide you with experience in understanding and writing assembly programs that employ control flow (which is essentially all non-trivial programs). Second, it introduces you to arrays and strings. Finally, it also demonstrates the use of constants stored in program memory.

In particular, you will learn:

- understanding and creating assembly programs that use control flow instructions (compares and branches)
- declaring constants, arrays, and strings in program memory, and working with arrays and strings
- the function of some additional assembly instructions, including:
    - compare instructions (`CP`, `CPI`, etc.)
    - branch instructions (`BREQ`, `BRLO`, `BRGE`, etc.)
    - the load program memory instruction (`LPM`)
- some assembly directives for allocating bytes and words in program memory:
    - the declare byte directive (`.DB`)
    - the declare word directive (`.DW`)

Pre_lab

Please turn in the following parts at the beginning of the lab hour.

Due: The first week of the experiment. Part1c , Part1d , part1f , Part2a, Part2b

Due: The second week of the experiment. Part2c , Part2d

Lab Assignment

- **Part 1: Introducing Control Flow, Arrays, and Strings**

  For the first part of the lab, you are given the assembly program, lab3_p1.asm. (Provided at the end of this handout.) Create a project for this program in AVR Studio, and begin stepping through to understand what it is doing.

  As indicated by the comments, this program contains three parts. The first part sums the elements from an array of four 8-bit unsigned numbers, and saves the result as a 16-bit word in the register pair `R10:R11`. The second part sums two elements from an array of two 16-bit numbers, and saves the result as a 16-bit word. Finally, the last part of the program iterates (loops) over an ASCII string.

  a. Question1_1: In this program, what is the purpose of the `LPM` instruction? For this instruction the second operand is always `Z+`. Which register(s) does `Z` constitute, and what is the `+` for? Why is this program using register `Z` instead of registers `X` or `Y` with the `LPM` instruction?

  Question1_2: When using the `.DB` and `.DW` (and similar) directives, it is best to place such constant declarations at the end of the program code (after the `rjmp End` instruction). What happens if you place (cut & paste) them before the `rjmp Init` instruction? Does the program still work? If so, will it necessarily work with any constant declarations?

  b. Question1_3: In the first the first part of the program, we are iterating (looping) over an array of four 8-bit unsigned numbers, and saving the sum over all the numbers as a 16-bit value in the register pair `R10:R11`. When we load an element of the array from program memory, what register is it being stored in?

  Question1_4: What is the purpose of register `R1`?

  Question1_5: Using this method, would it be possible to take the sum of an array of signed numbers? Why or why not?

  Question1_6: This part of the program uses a loop that iterates (loops) four times. What register(s) and instruction(s) are used to enable the loop to iterate (loop) the desired number of times?

  c. Problem: Create a new version of the first part of the program (computing the sum of the four 8-bit array elements) that iterates the appropriate number of times in a different fashion.

  Note: Delete all portions of the assembly program not relevant to the first part of the program.

d. **Problem:** The second part of the program computes the sum of two 16-bit words. Create a new version of the second part of the program that computes the sum of the two 16-bit words using a loop (that iterates two times).

   Note: Delete all portions of the assembly program not relevant to the second part of the program.

e. **Question1_7:** The third part of the program iterates over an ASCII string. Which register shows the value of the current ASCII character that in the string being read?

   Question1_8: Strings of ASCII characters commonly end in a terminator, which signifies the end of the string. What is the hexadecimal value of the terminator used in this ASCII string? Why is it desirable to use an unprintable character as the string terminator?

   Question1_9: This part of the program uses a loop to iterate over the string, but the loop iteration condition here is different than used previously. What test is being performed to determine when the loop should stop iterating?

f. **Problem:** Create a new version of the last part of the program that computes the length of the string (the number of characters contained in the string, excluding the string terminator).

   Note: Delete all portions of the assembly program not relevant to the third part of the program.

- **Part 2: Creating Programs using Control Flow, Arrays, and Strings**
  a. **Problem:** *ABC* Program

     Create a program that iterates through the capital letters of the alphabet (`A - Z`), outputting the ASCII value for each letter on Port B of the processor.

     Question2_1: There are various ways (checks) you could use to terminate your loop for this program. What is an alternate method for terminating the loop for this program?

  b. **Problem:** *Thresholding* Program

     In many real-life situations (e.g. monitoring sensors), programs are often instructed to ignore input values that are below a certain threshold (hence, this type of program performs "thresholding"). Create such a program, which iterates over the following array of 16-bit unsigned numbers, and adds the number to the sum if it is higher than 100, or otherwise ignores it.

     573, 16, 8, 39, 8192, 483, 1602, 198, 2607, 215, 101, 33, 598, 63, 882, 100, 120

Question2_2: Is your final sum what you would expect? State the result.

Question2_3: How would your program need to be different if the values were signed values, such that you ignore values between -100 and 100, otherwise the number is added to the sum? Write the modified program.

c. Problem: *Powers* Program (x^y)

Write a program that calculates the powers of unsigned integers. Using the `.DB` directive, create **two 8-bit values**, one called (labeled) `Base`, and the other called `Expon`. Load these two values from program memory and compute the result of `Base` raised to the `Expon` power, with the result saved as a **16-bit value** (e.g. if `Base` = 5 and `Expon` = 4, $5^4 = 625$).

Try your program with various values in `Base` and `Expon` to ensure that it works as expected.

d. Problem: *String Compare* Program

Write a program that compares two strings to see if they are the same. Using the `.DB` directive, create two strings, labeled `String1` and `String2`, both of which are appropriately terminated (see Part 1). Compare the two strings, byte by byte, to determine whether the two strings are identical. If the two strings are identical, end the program with the hexadecimal value of `0x01` in register `R0`, otherwise have register `R0` be `0x00`.

Note: The two strings are not necessarily of the same length, so ensure that your program works even if the two strings are of different lengths.

Try your program with various ASCII strings in `String1` and `String2` to ensure that it works in all cases.

Last modified: Monday, 15 September 2008

```
;***********************************************
; written by:                 Jason Fritts
; date:                       9/15/08
; Updated by:                 Armineh Khalili 8/25/2014
; file saved as:              lab3_p1.asm
; for AVR:                    ATmega32
; clock frequency:
;***********************************************

; Program Function:   <describe purpose of program here>
```

```
; .device      ATmega32              ; don't need because it's in .inc
file below
.nolist
.include       "C:\Program Files\Atmel\AVR
Tools\AvrAssembler2\Appnotes\m32def.inc"
.list


;=================
; Start of Program

        jmp     Init                           ; first line executed

;=========

Init:
        ; <insert code here to initialize ports, as needed>

;====================
; Main body of program

Start:
        ldi  zl, LOW(2*Byte_Array)
        ldi  zh, HIGH(2*Byte_Array)

        ; Summing four 8-bit words
        clr      r10
        clr  r11
        clr  r1
        clr  r16

Loop1:
        lpm  r0, z+

        add  r10, r0
        adc  r11, r1

        subi r16, -1
        cpi  r16, 4
        brlo Loop1

        ; Summing two 16-bit words
        ldi  zl, LOW(2*Word_Array)
        ldi  zh, HIGH(2*Word_Array)

        lpm  r24, z+
        lpm  r25, z+

        lpm  r0, z+
        lpm  r1, z+

        add  r24, r0
        adc  r25, r1

        ; Iterating through an ASCII string
        ldi  zl, LOW(2*String1)
```

```
        ldi  zh, HIGH(2*String1)

Loop2:
    lpm  r16, z+

        cpi  r16, 0
        brne Loop2


End:
        rjmp End


;=============
; Declarations


Byte_Array:
.DB 123,45,67,89                        ; a list of four bytes

Word_Array:
.DW 2137,984, 0                         ; a wordwise list of labels

String1:
.DB "This is a text.", 0       ; a list of byte characters
```