Lab #3 – Practice with 2-d arrays and with strings

Arrays are used extensively in assembly language programming. RAM is essentially one giant array of bytes or words. Every data variable we declare will occupy some portion of it. Today, we will practice with a 2-dimensional array of integers, and a string (array of bytes).

Create a new folder called lab3 on your USB drive or account, and do all of today's work in there. From the class Web site, you should see a folder called lab3. Download 2d.s and string.s. You will modify these programs.

Part 1: The 2-D array

Invoke MARS as you did in the previous lab. Open the source file 2d.s. This program defines an array called "a". Logically, we would like to conceive in our mind that this array has 6 rows and 8 columns of integers. However, in reality these 48 numbers simply reside consecutively in memory. The assembly programmer (or compiler) is responsible for converting between the 2-dimensional way of identifying a cell into the cell's actual address. This is what you will practice in this program.

The purpose of this program is to display on the screen one of the values in the array. We ask the user for the desired element: what is its row and column number? The I/O has already been done for you. The first part of the program obtains the row and column number from the user.

However, in the middle of the program, I have left out the code necessary to compute the address of this element in the data segment, and to place the contents of this cell into a register. This is what you need to finish. As a guide for your implementation, please refer to the suggested register allocation in the comment that appears before the text segment. Assume that this array is stored in row-major order, and that the "rows" and "columns" should be numbered starting from zero.

When you are ready, let's assemble and run the program! Try these input cases, and verify that each one produces the appropriate output. √

Row 0, Column 3 = _____

Row 3, Column 1 = _____

Row 4, Column 6 = _____

1. Next, run your program for two more input cases:

Row 2, Column 9 = _____

Row 6, Column 8 = _____

What can you conclude from these results? How would this program have responded differently if this had been a HLL (e.g. Java) program?

2. After you have run the program, notice that MARS has an Edit and an Execute tab. The Edit tab allows you to view the source code. The Execute tab shows the contents of the text and data segments. At the bottom of the data segment, you see options where I recommend that you off the "hexadecimal values". Now, you can see the array contents displayed in the data segment. Based on what you see:

   What is the base address of the array a? _____

   What is the address of variable b? _____

   Subtract the two above addresses: _____. How would we have anticipated this answer based on the source code?

3. At the bottom of the data segment, tick the box labeled ASCII. This will allow you to see the strings that we declared after variables a and b.

   What is the base address of the string prompt_row? _____

   What is the base address of the string prompt_col? _____

   Subtract these two addresses: _____. Explain how this compares with the length of the string prompt_row. √

Part 2: Manipulating strings

Next, inside MARS, open the program string.s. Assemble and run this program. It is designed to accept a string as input. Describe what is being printed.

For us, a string is just an array of single bytes. Each byte represents a character that can be printed out on the screen. But internally, a character is represented as a number in ASCII code. Here are some examples of ASCII codes:

Character	code	Character	code	Character	code	Character	code
'0'	48	'A'	65	'a'	97	Space	32
'1'	49	'B'	66	'b'	98	Newline	10
'2'	50	'C'	67	'c'	99		
'3'	51	'D'	68	'd'	100		
'8'	56	'Y'	89	'y'	121		
'9'	57	'Z'	90	'z'	122		

Notice that among the digits, and among the lowercase and capital letters, that the ASCII codes are consecutive. This makes it straightforward for the computer to alphabetize (sort) strings. Take a close look at the ASCII codes assigned to capital versus lowercase letters, for example 'B' versus 'b'. For any letter of the alphabet, what is the numerical difference between its capital and lowercase ASCII code?

The answer to this question will help you to modify the program. Notice that the program has a loop in which we visit each individual character of the string. Modify the body of the loop so that we capitalize all of the letters in the string. In other words, inside the loop, we want to say "if this character is a lowercase letter, change its ASCII code so that it becomes the capital version of the same letter." Because this is an if-then-else situation, you will need to use branch instructions. Characters that are not lowercase letters should be unchanged in the final output of the program.

Run your program to verify that it produces the correct output. ✓

Once again, let's take a look under the hood of this program. Select the Execute tab, and study the contents of the data segment.

1. Let's check some addresses.

   What is the base address of the input `string`? _____

   What is the base address of the `prompt`? _____

   Subtract these two addresses: _____. How does this difference compare with how the input string was declared in our program?

2. Modify the program so that string is allocated as ".space 1" instead of ".space 80". Assemble and run the program. Enter a single letter string as your input. After the program is finished, carefully inspect the contents of the data segment. I recommend that you tick the ASCII box at the bottom of the data segment so that you can see individual characters.

   What are the first 4 bytes in the data segment? _____, _____, _____, _____

   Based on what you see, how has the prompt changed?

3. Continue to have the input string allocated as ".space 1". Experiment running this program. Can you come up with an input string that will corrupt the contents of the output string `out_str`? The sizes of the strings in the data segment should give a clue as to how large a string to attempt. √

After you have your result above checked, you should change the input string allocation back to ".space 80", and re-run your program to verify that all is well with ordinary input.