Computer Architecture

Lab 9

In this lab, we'll start to think about patterns of memory references.

You should turn in the source file for the program you're asked to write in Step 3. It should be submitted via the tool available on Sakai for this lab. It is due by the start of the next class period.

Memory Reference Pattern

We're going to take a look at how the order in which you reference memory can matter. In Chapter 5 we'll see the theory behind it, but in this lab we'll start visualizing it using MARS.

Open the row-major.asm file in MARS. It contains a 16x16 word size matrix of .data values. If you go through a two-dimentional array row-by-row we call it row-major order. This is the normal way we traverse a 2D array and in C++ we write the following:

```
for(int i = 0; i < ROWS; i++)
    for(int j = 0; j < COLS; j++)
          arr[i][j] = stuff;
```

We can also access elements in memory in column-major order corresponding to the following C++ loop:

```
for(int i = 0; i < COLS; i++)
    for(int j = 0; j < ROWS; j++)
          arr[j][i] = stuff;
```

It might not seem like it matters, but it turns out that the connection between the datapath and the memory is the single biggest issue for computer performance. Chapter 5 discusses some ways modern machines deal with it. The short of it is that it definitely can matter in what order to access your variables, and in this lab we'll visualize it in MARS.

Step 1

At the top of MARS is a Run Speed slider. It's set by default to "Run speed at max." For this visualization, that's too fast. Change it to about 30 instructions/sec.

Next, assemble the row-major.asm program.

Under Tools, choose Memory Reference Visualization. You will see a 16 x 16 grid representing the 2D array in the .data section of the row-major.asm program. You can change the size of the grid to match another program (and we'll probably do that in the future), but seeing as this is an example provided by the makers of MARS it's set up to work with the defaults.

Click Connect to Mips and then Run the program. You should see the squares in the grid turn blue as the program runs. Blue is the color associated with accessing the memory patterns 1 time. The Counter Value slider shows you the colors corresponding to different access counts.

Step 2

Repeat Step 1 using the column-major.asm file.

Step 3

Create a new program that computes the first 20 Fibonacci numbers. Store them in a 20-element array in the .data section and also save the size of the array (20) in the .data section as well. Store a string (using .asciiz) that prints something like "Here are the Fibonacci numbers" and is printed at the beginning of the program.

Then compute the values of the array, and finally print the values to the console.

Set up the Memory Reference Visualization tool and run it on your program.