CS 214: Introduction to computing systems
Fall Term, 2025

Lab 2

The purpose of this lab is to practice writing functions in assembly.

Assembly functions

As promised, I'd like you to start by implementing a recursive function to compute Fibonacci numbers by translating the following code:

```
int fib(int n) {
  if(n < 2)
    return n;

  int val1 = fib(n-1);
  int val2 = fib(n-2);
  return val1 + val2;
}
```

Begin with the following "main function" that calls the `fib` function and prints its return value:

```
.text
        #read an integer and call fib with it
        addi $v0, $zero, 5
        syscall
        addi $a0, $v0, 0
        jal fib

        #print return value
        add $a0, $v0, $zero
        addi $v0, $zero, 1
        syscall

        #exit
        addi $v0, $zero, 10
        syscall
```

Then I suggest that you write the `fib` function in the following steps:

1. Begin with a skeleton that allocates 16 bytes, has room for the body, and concludes with code to return:

```
fib:    addi  $sp, $sp, -16
        #TODO: store values onto stack

        #TODO: body of function

return: #part of fib that returns
        #TODO: restore values from stack
        addi $sp, $sp, 16
        jr   $ra
```

2. Make the function store `$ra`, `$s0`, and `$s1` onto the stack at the beginning of the function (right after the `addi`) and load them back at the end (right before the final `addi`). The addresses to use are `$sp`, `$sp + 4`, and `$sp + 8`.

3. Implement the body of the function, using `$s0` and `$s1` to store the values of `n` and `val1`. The `return` statements should involve putting the return value into `$v0` and then branching to the part of the function that removes the new stack frame and then returns. In order to make the recursive calls to `fib`, put the argument into `$a0`, use `jal` to make the call, and then take the return value from `$v0`.

Once you've done those things, test your function to make sure that it works. Either way, I suggest that you step through the code as well to familiarize yourself with the debugging capabilities of MARS. If you have trouble with your program, raise your hand; this can be tricky to debug so I am happy to help.

Here are some correct return values to check against:

n	0	1	2	3	4	5	6	7
fib(n)	0	1	1	2	3	5	8	13

When you finish that, translate the code below to write a function to compute n multichoose k, which is sometimes written as $\left(\binom{n}{k}\right)$. This is the number of different unordered collections of size k that can be selected from n things. Again, start with the "main" from class.

```
int multichoose(int n, int k) {
  if(k == 1)
    return n;
  if(n == 1)
    return 1;

  int val1 = multichoose(n-1, k);
  int val2 = multichoose(n, k-1);
  return val1 + val2;
}
```

For testing, here are some values

		n				
		1	2	3	4	5
	1	1	2	3	4	5
	2	1	3	6	10	15
k	3	1	4	10	20	35
	4	1	5	15	35	70
	5	1	6	21	56	126