Computer Science 330 - Project 2
Recursive Arithmetic Operations

Due: Wed. Sep. 30, 11:30 a.m.

In the early days of desktop computing, CPUs were quite primitive. None of them supported floating point operations, or complex integer operations like exponentiation or remainder, and many did not even include instructions for integer multiplication. Because of these limitations, many arithmetic operations were performed in software.

We are going to write code to simulate the multiplication and exponentiation of non-negative integers in software using recursion. Translate the program shown at the end of this handout into MIPS assembly language.

What to turn in: When you are ready to turn your program in, upload a copy of your source file to blackboard.

```
#include <stdio.h>
#include <stdlib.h>

int multiply (int a, int b);
int raise (int a, int b);

int main()
{
  int b, e;

  /* Prompt user for base and exponent, calculate and print
     out base ^ exp. */
  printf ("Enter non-negative base:");
  scanf ("%d", &b);
  printf ("Enter non-negative exponent:");
  scanf ("%d", &e);
  if (b == 0 && e == 0) {
    printf ("Error: 0 ^ 0.\n");
    exit (-1);
  }
  printf ("%d ^ %d = %d\n", b, e, raise (b, e));
}
```

```
int multiply (int a, int b)
{
  /* Straightforward recursive definition of a * b. */
  if (a == 0) {
    return 0;
  }
  return b + multiply (a - 1, b);
}

int raise (int a, int b)
{
  /* Calculate a ^ b using optimized recursive def. */

  /* Base case: exponent is 0. */
  if (b == 0) {
    return 1;
  }

  /* Recursive def.  Two cases to consider: b even, and b odd. */
  if ((b & 0x1) == 0x1) { // b odd
    return multiply (a, raise (a, b - 1));
  }

  // b is even
  int temp = raise (a, b/2);
  return multiply (temp, temp);
}
```