```
1  # MIPS Implementation of multiplication and exponentiation
2  # Brandon Ingli
3  # 25 March 2019
4
5  .data
6  prompt1:  .asciiz "Enter non-negative base: "
7  prompt2:  .asciiz "Enter non-negative exponent: "
8  newline:  .asciiz "\n"
9  exp_sym:  .asciiz " ^ "
10 equals:   .asciiz " = "
11 err_0:    .asciiz "Error: 0^0.\n"
12           .align 2 #Make sure subsequent words line up correctly
13
14 #Begin Code
15 .text
16
17 # b -> $s0
18 # e -> $s1
19
20 main:
21         #Prompt for base
22         la $a0, prompt1
23         li $v0, 4
24         syscall
25
26         #Read base
27         li $v0, 5
28         syscall
29         move $s0, $v0
30
31         #Prompt for exponent
32         la $a0, prompt2
33         li $v0, 4
34         syscall
35
36         #Read exponent
37         li $v0, 5
38         syscall
39         move $s1, $v0
40
41         #Print a new line
42         la $a0, newline
43         li $v0, 4
44         syscall
45
46         #Check for invalid case (0^0)
```

```
47          bne $s0, $zero, main2
48          bne $s1, $zero, main2
49          #Print error message
50          la $a0, err_0
51          li $v0, 4
52          syscall
53          #Exit with -1 result
54          li $v0, 17
55          li $a0, -1
56          syscall
57
58  main2:  #call raise(b,e)
59          move $a0, $s0
60          move $a1, $s1
61          jal raise
62
63          move $s2, $v0
64
65          #Print result
66          li $v0, 1
67          move $a0, $s0
68          syscall
69
70          li $v0, 4
71          la $a0, exp_sym
72          syscall
73
74          li $v0, 1
75          move $a0, $s1
76          syscall
77
78          li $v0, 4
79          la $a0, equals
80          syscall
81
82          li $v0, 1
83          move $a0, $s2
84          syscall
85
86          li $v0, 4
87          la $a0, newline
88          syscall
89
90          #exit
91          li $v0, 10
92          syscall
93
94  ###### raise(b, e) = b ^ e
```

```
95  # Used $s0
96
97  raise:
98          addi $sp, $sp, -8
99          #save ra
100         sw $ra, 4($sp)
101         #save s-regs
102         sw $s0, 0($sp)
103
104         #Base Case: exponent is zero
105         bne $a1, $zero, raise2
106         li $v0, 1
107         j raise_return
108
109 raise2:
110         #recursive case 1: exponent is odd
111         andi $t0, $a1, 0x1
112         beq $t0, $zero, raise3
113         #Save the base
114         move $s0, $a0
115         #Calculate raise(b, e-1)
116         addi $a1, $a1, -1
117         jal raise
118         #Calculate mult(b, raise(b, e-1))
119         move $a0, $s0
120         move $a1, $v0
121         jal multiply
122         j raise_return
123
124 raise3:
125         #recursive case 2: e is even
126         #Call raise(b, e/2)
127         srl $a1, $a1, 1
128         jal raise
129         #Call mult(temp, temp)
130         move $a0, $v0
131         move $a1, $v0
132         jal multiply
133
134 raise_return:
135         #cleanup stack and return
136         lw $s0, 0($sp)
137         lw $ra, 4($sp)
138         addi $sp, $sp, 8
139         jr $ra
140
141 ###### multiply(a, b) = a * b
142 #Used $s0
```

```
143
144 multiply:
145         addi $sp, $sp, -8
146         #save ra
147         sw $ra, 4($sp)
148         #save s-regs
149         sw $s0, 0($sp)
150
151         #Base Case: a is zero
152         bne $a0, $zero, multiply2
153         move $v0, $zero
154         j multiply_return
155
156 multiply2:
157         #Recursive Case
158         #Save b
159         move $s0, $a1
160         #Call mult(a-1, b)
161         addi $a0, $a0, -1
162         jal multiply
163         # Add to b
164         add $v0, $s0, $v0
165
166 multiply_return:
167         #cleanup stack and return
168         lw $s0, 0($sp)
169         lw $ra, 4($sp)
170         addi $sp, $sp, 8
171         jr $ra
```