Laboratory 4: Postlab

Date 3/1/2022 Section 02

Name MINGXI XIA

Recursive Procedures (cont'd)

Now that you have enough experience with the procedure call mechanism in MIPS it is time to try something harder on your own.

A famous example of a recursive function is *Ackermann's function* $A(x, y)$:

- if $x = 0$ then $y+1$
- else if $y = 0$ then $A(x\text{-}1, 1)$
- else $A(x\text{-}1, A(x, y\text{-}1))$

Step 1

Write a program in C/C++ (named *lab4.6.c*) which:

- asks the user for two non-negative integers, x and y
- ouputs the value of the Ackermann's function for those two values, $A(x, y)$

Run the program and fill out the missing parts of the following test table

Test plan for the C/C++ Ackermann's function

x	y	$A(x, y)$
0	0	1
0	2	3
1	0	2
2	3	9
2	4	11
2	5	13
2	6	15
3	1	13
3	2	29

Test plan for the C/C++ Ackermann's function

x	*y*	*A*(*x*, *y*)
3	3	61

Note: Do not take large values (especially for *x*) if you want your program to finish in a reasonable amount of time (or to finish at all). It takes 128 seconds to compute *A*(3, 12) on an Indigo[2] running at 250 MHz, with 64 MB main memory, with the code compiled using the cc compiler and the -O2 flag for optimization. On the same machine *A*(4, 2) exhausts not only the main memory space but also the 100 MB disk swap space.

Step 2

Create a program named *lab4.7.asm* which

- in 'main' prompts the user to enter two non-negative integers; store them in **`$t0`** and **`$t1`**
- check if any of the numbers entered is negative: if it is negative, then print a message saying so and prompt the user again for numbers
- call the procedure named 'Ackermann' whose parameters will be the integers read from the user, and which returns the value of the Ackermann's function for those two integers
- pass the parameters in registers
- prints a message and the value returned by 'Ackermann'

Run your program and fill out the missing spaces in the following test plan

Test plan for the MIPS assembly Ackermann's function

x	*y*	*A*(*x*, *y*)
0	0	1
0	2	3
1	0	2
2	3	9
2	4	11
2	5	13
2	6	15
3	1	13
3	2	29
3	3	61

Make sure you obtain the proper values when running your program.

Q 1:

How many recursive calls are made to calculate *A*(2, X) where X is the first digit of your SSN?

X = 3	Recursive calls = 2x^2+7x+5= 62

Q 2:

Every time a recursive call is made, the stack grows. By how many bytes per call?

8 bytes

Step 3

Return to your lab instructor copies of *lab4.6.c* and *lab4.7.asm* together with this postlab description. Ask your lab instructor whether copies of programs must be on paper (hardcopy), e-mail or both.