CS-1216 – Monsoon 2023 – Assignment 2

Rutam Kathale, rutam.kathale@gmail.com

October 15, 2023

Problem 3: For the factorial(n) MIPS code discussed in class, schematically draw the contents of the stack after every recursive call for n = 5.

On call to factorial(5), this happens:
First, factorial(5) is called: In the stack, main and the argument 5 are pushed onto the stack.

factorial(4) is called within factorial(5): The return address of factorial(5) and the argument 4 are pushed onto the stack.

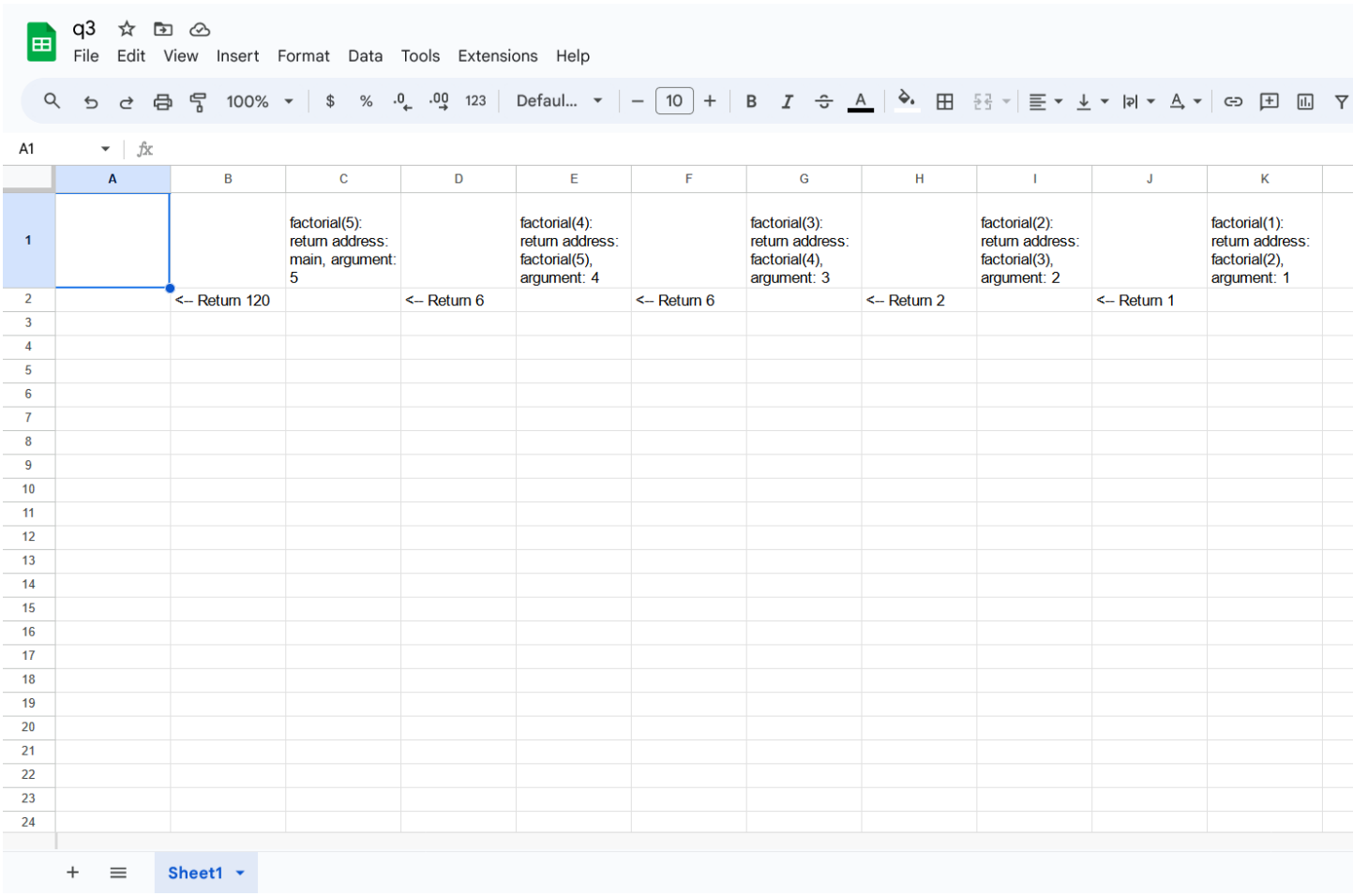
factorial(3) is called within factorial(4): The return address of factorial(4) and the argument 3 are pushed onto the stack.

factorial(2) is called within factorial(3): The return address of factorial(3) and the argument 2 are pushed onto the stack.

factorial(1) is called within factorial(2): The return address of factorial(2) and the argument 1 are pushed onto the stack.

After this, the program starts returning values to get the desired factorial

A representation of the process is attached in the image below:



	A	B	C	D	E	F	G	H	I	J	K
1			factorial(5): return address: main, argument: 5		factorial(4): return address: factorial(5), argument: 4		factorial(3): return address: factorial(4), argument: 3		factorial(2): return address: factorial(3), argument: 2		factorial(1): return address: factorial(2), argument: 1
2		<-- Return 120		<-- Return 6		<-- Return 6		<-- Return 2		<-- Return 1	

Problem 5. How do you handle a 32-bit integer as an immediate operand in MIPS? How do you handle a 32-bit address when making a jump or branch?

For MIPS, we have to handle 32-bit integers since the immediate operands are only supported up to 16-bit. For using 32-bit integers as immediate operands, we have to split it into two 16-bit immediate values and load them into separate registers. The instructions 'lui' and 'ori' enables us to do this. lui is used to load upper 16 bits of the integers into 'hi' part of register, and ori instruction inserts the lower 16 bits into the 'lo' part of register.

MIPS uses PC-relative addressing for jump and branch instructions. In a branch instruction, 16-bit offset is shifted left 2 bits to become a word offset which is added to the address of the instruction following the branch. For jump instructions, MIPS uses pseudo-direct addressing where the 26-bit address field in these instructions is shifted left by 2 bits and concatenated with the upper bits of the current PC. So, jump and branch instructions don't actually work directly with 32-bit addresses where relative addressing would typically accomodate 32-bit addresses and otherwise, we use lui and ori for managing 32-bit operands.