Exercise classes computer architecture and system software

Yves Younan
DistriNet, Department of Computer Science
Katholieke Universiteit Leuven
Belgium
Yves.Younan@cs.kuleuven.ac.be

Overview

- Functions
- Stack
- Registers
- Extra instructions
  - div instruction
  - lui instruction
- Recursion vs. tail recursion

Functions

- Functions take parameters, perform a calculation and return a result
- Parameters to a function are passed in arguments $a0 to $a3
- When a function ends it must resume execution at the next instruction from where it was called.
- To enable resuming of execution even after multiple function calls, we use a return address, stored in $ra

Functions

- Return values for functions are stored in $v0 to $v1
- When calling a function, registers might be changed, so we need to save them somewhere.
- Functions are called with the instruction:
  - jal function_name
  - This will save the return address to $ra and then call the function
- Returning from a function:
  - jr $ra

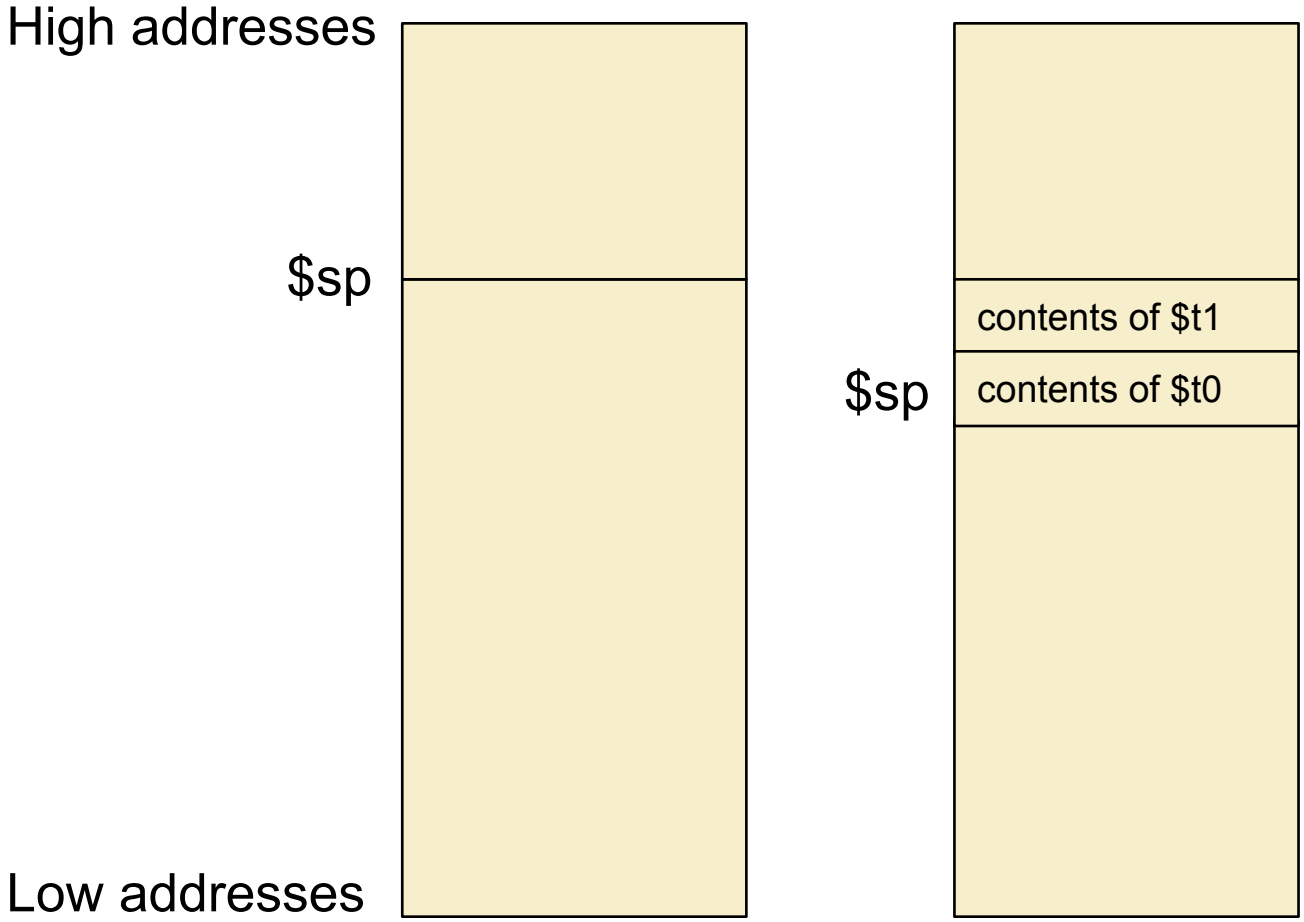
Stack

- The stack is a last-in, first out (LIFO) structure.
- We place values on the stack one by one (push) and remove them one by one (pop)
- On Mips the stack is a memory region which grows down from high addresses to low addresses
- It is controlled by the $sp register which is a pointer to the top of the stack (i.e. it's lowest address) at any given time
- Use for local variables and to store registers

Stack

- Push: substract from stack pointer
- Pop: add to stack pointer
- Example:
  - addi $sp, $sp, -8
  - sw $t1, 4($sp)
  - sw $t0, 0($sp)
- Access to a variable on the stack occurs as follows:
  - lw $t1, 4($sp)

Stack


High addresses
$sp
Low addresses
$sp
contents of $t1
contents of $t0

Stack

- Because the stack pointer is constantly changing, you must always recalculate the position of a stack-stored variable
- Some functions use a frame pointer to access local variables to avoid this
- The frame pointer is set to the value of $sp when entering the function
- When you need to access a local variable, it will be at a static location from $fp

Registers

- A function needs to use registers, however the function which is calling the function may also use these registers
- Some registers must be preserved over function calls (i.e. a function must store them somewhere safe before using them and restore them when done)
- Other registers are not preserved and if a function needs them it must save them before calling a function

Registers

- Preserved registers are called callee-save registers (because the function being called must save them):
  - The saved registers: $s0 - $7
  - The stack pointer: $sp
  - The return address: $ra
  - The frame pointer: $fp

Registers

- Registers which are not preserved are called caller-save registers (because the calling function must save them):
  - Temporary registers: $t0-$t9
  - Argument registers $a0-$a3
  - Return value registers $v0-$v1

Extra instructions

- Divide
  - div $t0, $t1
    - Divides $t0 by $t1
    - Result is stored in 2 separate 32-bit registers called hi and lo
    - hi contains the remainder
    - lo contains the quotient
  - mflo and mfhi copy these registers into general purpose ones
    - mfhi $t0

Extra instructions

- Load upper immediate
  - lui $t0, 1000
  - Loads 1000 into the first 16 bits of $t0
  - Easier than
    - addi $t0, $zer0, 1000
    - sll $t0, $t0, 16

Recursion vs. tail recursion

- Tail recursion:
  - Recursion at the end of the function, no other recursive calls
  - Considered to be an iterative function
- Real recursion:
  - Requires stack state to be saved