Lecture 10

On functions -- how to implement them in assembly language, and the commands and protocols associated with them (JAL, JR, $ra, $a0-a3, $v0-v1, persisting registers, etc)

Functions/Function Calls

- The basic idea: a function will be defined as some code following a label (with the function name). The caller jumps to the label and executes the following code until it receives an instruction to return control to the instruction after the original function call. Passing parameters and return values is handled by storing them in registers meant for the purpose.

- `JAL` - jump and link
    - When we jump to a function, we need to know the location of the next instruction (before jumping), so we know where to return control to afterwards
    - JAL automatically stores addres of the next instruction in the `$ra` register when called
    - Call with `JAL function_name`
    - return control with `JR $ra` *(jump to the address specified in $ra)*

  Caller - the function doing the calling (e.g. the *main* function calls the *pow* function)

  Callee - the function being called

How do we pass parameters?

- registers `a0 - a3` are used to accept parameters
- registers `v0 - v1` are used to return the result
- *Specify in a comment above your function definition which registers should be loaded with parameters (in what order), and similarly with return values.*

A note on convention -- persistent registers.

- By convention, we decide that some register types have their state/value preserved after calling a function, while others don't. For example, s registers cannot have their value mangled after a call to a function -- they are guaranteed to be unchanged. t registers, on the other hand, have no guarantee of maintaining their state -- functions can freely change and repurpose them without preserving their values upon return to the caller.

A note on pseudoinstructions

- A pseudoinstruction is something that isn't a direct mnemonic to a machine instruction, but is allowed by the assembler anyhow. A common example is the `LI` (load immediate) command, which is equivalent to using `ADDI $target, $zero, val`, without the burden of the $zero register.

EXAMPLE

Create the function INCREMENT

- accepts integer in a0
- increases integer by 1
- returns result in v0

CALLER

```
ADDI $a0, $zero, 5          #load value into param
JAL INCREMENT               #jumps to function (calls)
...
```

CALLEE

```
INCREMENT:                  #label/function name
ADDI $v0, $a0, 1              #Increment, load result into return register
JR $ra                      #jump to next instruction in caller
```