Procedures

Lecture 11
Section 2.8

Robb T. Koether

Hampden-Sydney College

Wed, Sep 18, 2019

Outline

Calling a Function

Calling a Function

```
        jal     myFunc          # Call myFunc()
         :
myFunc:
```

- To call a function, we must give it a name, i.e., a label.
- Then use the jump-and-link (`jal`) instruction to jump to the function.
- The `jal` instruction will store the current program counter `$pc` in the return-address register `$ra`.

Calling a Function

Calling a Function

```
        jal     myFunc          # Call myFunc()
         :
myFunc:
```

- To call a function, we must give it a name, i.e., a label.
- Then use the jump-and-link (`jal`) instruction to jump to the function.
- The `jal` instruction will store the current program counter `$pc` in the return-address register `$ra`.
- And that is REALLY IMPORTANT!

Returning from a Function

Returning from a Function

```
        jr      $ra                 # Return
```

- To return from a function, use the jump register (`jr`) instruction.
- The `jr` instruction will load the address stored in the specified register (normally `$ra`) into the program counter `$pc`.
- See `jump_return.asm` as an example.

Outline

Argument Passing

- The registers `$a0` - `$a3` are intended to be used to send arguments to functions.
- The registers `$v0` and `$v1` are intended to be used to return values from functions.
- See `max_func.asm` as an example.
- Write a program `list_max.asm` that uses `max()` to find the largest integer in a list of integers.

Outline

The Stack

- Typically, the called function must use some of the registers `$s0` - `$s7`.
- Yet, the calling function assumes that the called function will not alter the values in these registers.
- How can we protect these registers from being modified by the called function?

The Stack

Pushing onto the Stack

```
addi    $sp, $sp, -16    # Reserve 16 bytes on stack
sw      $s0, 12($sp)     # Push $s0
sw      $s1, 8($sp)      # Push $s1
sw      $s2, 4($sp)      # Push $s2
sw      $s3, 0($sp)      # Push $s3
```

- There is no `push` instruction or `pop` instruction.
- To push, we have to adjust the stack pointer to create space on the stack.
- Then we have to store the data in that space.
- The stack "grows" towards lower memory, so we subtract.
- It is the responsibility of the called function to do this. (Why?)

The Stack

Popping off of the Stack

```
lw      $s3, 0($sp)      # Pop $s3
lw      $s2, 4($sp)      # Pop $s2
lw      $s1, 8($sp)      # Pop $s1
lw      $s0, 12($sp)     # Pop $s0
addi    $sp, $sp, 16     # Release 16 bytes from stack
```

- Popping is the reverse of pushing.
- To pop, we load the value from the stack into a register.
- Then we adjust the stack pointer to free up the space.

Outline

Assignment

Assignment

- Read Section 2.8.