Computer Science 210

Computer Systems 1

2007 Semester 1

Lecture Notes Part 2

Registers & Subroutines

Lecture 8
5 Apr 07

James Goodman

Reminders

- **Mini-assignment 2 is due on Monday, 30April.**
- **There is a tutorial today in this room, at 3.30.**

Recommended Readings

For the mini-assignment

- Chapter 6: program structure
- Chapter 7: strings
- Chapter 8: running the simulator
- Chapter 10: writing and debugging in assembly language

For today's lecture

- Chapter 5: use of registers
- Chapter 11: function invocation.

I/O Instructions

- Privileged Architecture Library (PAL)
  - A set of functions of arbitrary complexity invoked by a special call_pal instruction
  - Performs privileged operations such as accessing disk, reading and printing, etc.
- Form: call_pal constant

```
call_pal CALL_PAL_CALLSYS
call_pal CALL_PAL_BPT
```

Simple I/O

- getchar (result in $v0)

```
ldiq $a0, 0x1  // CALLSYS_GETCHAR
call_pal 0x83  // CALL_PAL_CALLSYS
```

- Equivalent operation: **`bsr Sys.putChar.enter;`**

- putchar (character in $a1)

```
ldiq $a0, 0x2  // CALLSYS_PUTCHAR
call_pal 0x83  // CALL_PAL_CALLSYS
```

- Equivalent operation: **`bsr Sys.getChar.enter;`**

Registers Named

$0	$v0
$1-$8,	$t0-$t9
$9-$14	$s0-$s5
$15	$fp
$16-$21	$a0-$a5
$22-$25	$t8-$t11
$26	$ra
$27	$pv
$28	$at
$29	$gp
$30	$sp
$31	$zero (special)

Register Names

$t0-$t11	Temporary registers, used to hold temporary values, when evaluating expressions, etc.
$s0-$s5	Saved registers, used to hold the values of local variables in functions.
$a0-$a5	Argument registers, used to pass parameters to functions.
$v0	Value register, used to return the result of a function.
$ra	Return address register, used to hold the return address of a function.
$gp	Global pointer register, used to point to the table of constants.
$sp	Stack pointer register, used to point to the top of the stack used to allocate space for functions.
$zero	Zero register, that always contains the value zero. Attempting to write to this register has no effect.

Memory Allocation for a Variable

- Global variables, constants: allocate memory permanently
  - Use registers? Maybe, if used frequently
- Local variables
  - Allocate space permanently?
    - Not needed: variables have a lifetime
    - Not sufficient: same variable might have multiple instances
  - Use registers? Likely, since they are short-lived and dynamic
- Temporary variables (used in computations)
  - Similar to local variables
  - Allocate space dynamically, probably in registers
- Arguments
  - Also have a lifetime
  - Pass in registers? Yes, if not too many
  - Also result(s), but in reverse direction

Two Distinct Storage Issues

- Registers vs. memory
- Dynamic variables

Dynamic Variables

Variables have a lifetime

- A variable is defined within a scope
- Variables do not need space allocated if they aren't assigned a value
- Different variables can be assigned to the same memory location at different times
- The same variables in different instances requires two different memory locations if they overlap (recursion)

Extreme Case: Write-once variables

- A variable requires storage when it is written
- A variable does not require storage if it will not be read again before it is written
- If we know a variable will not be read, we can deallocate storage on the last read, allocate it on write.
  - We must be certain that the variable will not be read again
  - This is often possible in controlled situations, e.g., loops
- In effect, each write creates a new variable, written only once
- Hardware implications
  - with multiple instructions being executed, a dead variable can be inferred on each write (previous instructions may still need to read it)
  - A different buffer can be assigned the new value while the old value is still live!

The Stack

- Modern programming languages require the ability to allocate space for an indefinite number of variables
- Each instance of a method requires its own space for variables, arguments, and temps.
- The *Stack of Activation Records* is a data structure that satisfies this requirement.
  - On invocation
    - Allocate space for arguments, temps, local variables: a *Frame*
    - Save (spill) some registers to allocate for subroutine
    - Save linkage information (how to return)
    - Transfer control to subroutine
  - On return
    - Assign return value
    - Restore spilled registers
    - Deallocate space
    - Jump back to original code