Computer Science 210 Computer Systems 1 2007 Semester 1 Lecture Notes Part 2 Subroutines

Lecture 9 24 Apr 07

James Goodman



Reminders

- Mini-assignment 2 is due on Monday, 30April at noon.
- There will be a tutorial Thursday, probably in this room (not yet confirmed), at 3.30. [No tutorial tomorrow]

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.

Correction to Template

CS210

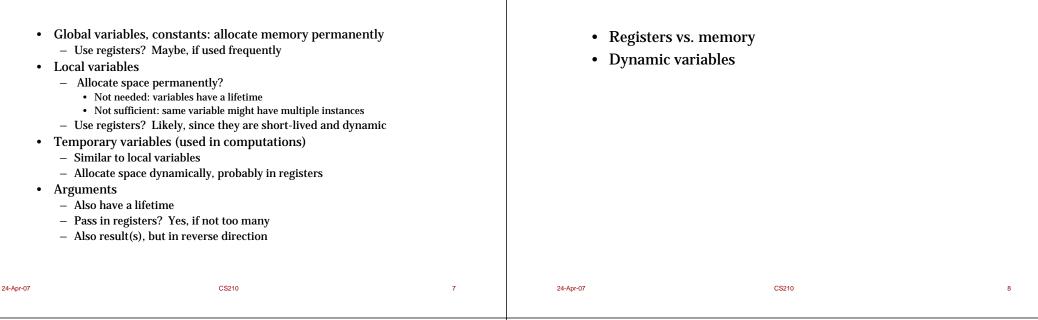
Assignment 2 template (available from website)

- File Assignment2.user.s has two extraneous lines:
 - 39 ldiq \$a0, 2;
 - 40 call_pal 0x83;
- Delete both lines (or download revised template)

24-Apr-07

2

Memory Allocation for a Variable



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)

The Stack

Two Distinct Storage Issues

- 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

Caller vs. Callee

- Who should allocate space?
 - Callee knows how much space it needs
 - Arguments and return are special: they are shared
- Who should save registers?
- Caller should save
 - Don't need to save registers not being used
 - Only caller knows this
- Callee should save
 - Don't need to save registers that won't be touched
 - Only callee knows
- Solution: do both!

Caller/Callee Register Allocation

- Temporary registers for callee
 - \$t0-\$t11
 - Free for use, but not preserved
- Saved registers for caller
 - \$s0-\$s5
 - Free to use, but responsible for saving/restoring value
- Every method is potentially both a caller and callee
 - Leaves (methods that invoke no other methods) often don't need to use S registers—no spills
 - Other nodes save registers they use exactly once: on invocation

Dealing	with	Arguments
Doamig		/ ii gainonto

CS210

- Used for communication between caller and callee
- No limit to number of allowed arguments
 - Pass arguments in registers: \$a0-\$a5
 - Pass additional arguments through stack

Use of Stack for Subroutines: Caller

CS210

- Caller has allocated space for arguments in its stack frame
 - Save current (caller's) arguments on stack
 - Save previously returned result (if needed)
- Assign arguments to registers (\$a0-\$a5)
- If temporary registers are live, save
- Caller executes bsr instruction
 - Address of subsequent instruction stored in \$ra
 - Jumps to beginning of callee
- On return
 - Restore arguments (if needed)

24-Apr-07

11

24-Apr-07

12

Use of Stack for Subroutines: Callee

- Callee allocates new stack frame
 - Space for local variables
 - Space to save S/T registers if needed
 - Space to save return address (if not a leaf)
 - Space for parameters (if not a leaf)
- Execute code
 - May invoke other subroutines
- Assign result
- Restore registers
- Deallocate stack space
- Return to caller

Accesses to the Stack

- The layout of a stack frame (activation record) is determined when the method is *compiled*
- At assembly time, when the code is produced
 - the abolute address cannot be fixed (it varies depending on circumstances)
 - the relative address (relative to the top of stack) is known: a small constant
- Addressing mode of base register + displacement is perfect
 - base: frame pointer (or stack pointer)
 - displacement (computed when the stack frame is laid out.

24-Apr-07	CS210	16	24-Apr-07	CS210	17