Computer Science 210

Computer Systems 1

2007 Semester 1

Lecture Notes Part 2

(Finish Subroutines)

The Assembly Process

Lecture 10
26 Apr 07

James Goodman

Department of Computer Science

Reminders

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

In-class Test

- Test next Wednesday 3May
- During class (this room)
- Coverage: lectures and assignments up to term break

Recommended Readings

For today's lecture

- Chapter 11: function invocation.
- Randy Bryant, *Alpha Assembly Language Guide* (available under Resources at the website) Section 3
- Chapter 12: assembling and disassembling

For the mini-assignment

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

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

- 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
- Argument registers $a0-$a5 are like temporaries
  - Must be preserved if needed after a call
  - If not needed, can be used as a temporary

Use of Stack for Subroutines: Caller

- Caller has allocated space for arguments beyond $a4 in its stack frame
  - Save current (caller's) arguments on stack if needed
  - Save previously returned result $v0 (if needed)
- Assign arguments to registers ($a0-$a5)
- If temporary registers are live, save
- Caller executes bsr instruction
  - Address of subsequent instruction stored in $ra
  - Jumps to beginning of callee
- On return
  - Restore arguments ($a0-$a5) and tmps ($t0-$t5) if/when needed

Use of Stack for Subroutines: Callee

- Allocate space for new activation record
- Saved any saved registers ($s0-$s11) to stack
- Save $ra to stack if any other procedure might be called
- Perform function (possibly invoking other functions)
- Restore saved registers ($s0-$s11, $ra)
- Assign return value to $v0
- Deallocate space for current activation record
- Return to calling procedure via $ra

Accesses to the Stack

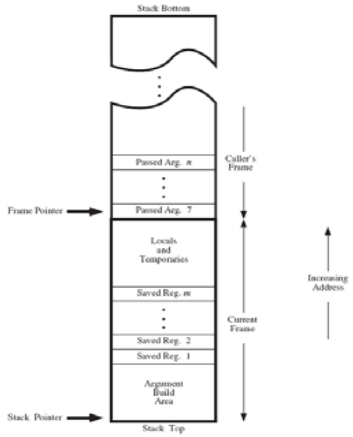
- The layout of a stack frame (activation record) is determined when the method is *compiled*
- At assembly time, when the code is produced
  - the abolute address cannot be fixed (it varies depending on circumstances)
  - the relative address (relative to the top of stack) is known: a small constant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

Example of Stack Access

sw $s1, 32($sp)

Offset from top of stack to save $s3

Top of stack



Optimizations

- Stack is designed to handle worst case:
  - Spilled registers
  - Return address
  - Extra arguments
- In practice stack can be very small
  - If called procedure is a leaf (does not call other procedures), it may not not need a stack at all.
  - Even if it calls other procedures, it needs to save RA, but
    - May not need to save arguments
    - May not need to save registers
    - Could be as little as one word!

Computer Science 210

Computer Systems 1

2007 Semester 1

Lecture Notes Part 2

The Assembly Process

Lecture 10
26 Apr 07

James Goodman

Department of Computer Science

Translation vs. Interpretation

- A program written in language L defines a "machine"
- Problem:
  - We have a program written in language L1
  - We have a computer that understands how to execute language L2
  - How to "execute" program?
- Solution 1: Compilation/Translation/Assembly
  - A "compiler" takes as input a program written in L1 and creates a program written in L2
- Solution 2: Interpretation/Simulation/Emulation
  - A "program" takes as input a program written in L1 and walks through its execution, taking input and creating output as if it were an L1 computer.

The assembly process: Overview

- A computer understands machine code
- People (and compilers) write assembly language
  - Today it's usually compilers
- Goal: create a file describing exactly what memory should look like before starting execution of a program
- Assembly: the process of translating a program written in assembly language into a program written in machine language.
  - Machine language is specific to a computer
  - Need to create memory image that includes
    - instructions (including links to libraries, kernel, etc.)
    - data (constants, static variables, dynamic variables

Steps in Assembly

- Read program and parse
  - Identify declarations of instructions and static data
  - identify pseudo-instructions
  - Allocate space for instructions and data
  - Recognize labels
    - Define address if possible
    - Remember for future reference