Function Support in Hardware

2-10-2003

Opening Discussion

- What did we talk about last class?
- Have you seen anything interesting in the news?
- Suppose that you write a program in which main calls a function A which then calls a function B. What happens in the computer so that this can work? What if B is recursive and calls itself a variable number of times?

Procedures/Functions

- Obviously, the ability to implement procedure/functions is critical for any modern programming language. Even in assembly we really like to have it, because it allows us to get some code reuse by calling the same function from multiple places.
- The question is of course, what really goes into implementing a function at a low level.

Jump and Link

- The most basic type of code reuse is something like an old BASIC subroutine (gosub). This was nothing more than a jump that remembered where it came from so it could jump back.
- MIPS provides an instruction to help with calling functions , the jal instruction. It jumps to the label, but also stores the address of the next instruction in $ra.

Program Counters

- The way that stored program computers typically work is to have an extra register called a program counter (PC) that stores the address of the next instruction that is to be executed.
- On a normal instruction, this value is simply incremented to the next instruction (always 4 bytes away on a MIPS machine). A jump/branch just puts a new value in the PC.

Stack and the Stack Pointer

- The old BASIC gosub was really limited in what it could do though because all that it did was keep track of where it should return to. For many things we want more power.
- This extra power comes from being able to store values in memory in a way where each new function gets its own little bit of memory. This is implemented with the stack. The address of the "top" of the stack is stored in $sp.
- The stack grows from higher to lower addresses.

Saving Registers

- When we want to "save" a value of a register so that it can be used for something else, we push it onto the stack. This is done by decrementing the stack pointer and storing the proper value into that memory.
- When the values need to be taken off the stack, popped, they are loaded and $sp is incremented or set back up.

What and When to Save

- A function can determine what it wants to push to the stack, but there are certain "rules" that it is supposed to follow.
  - It has to preserve the values in $s0-$s7, $gp, $sp, and $fp. This means whatever was in those when the function is called should be back there when it ends.
  - It also needs to store any other registers it might need before calling another function. This only matters for non-preserved registers.

Burden on Functions

- One thing to note about the way that function calls are done in MIPS assembly is that the burden of preserving the saved registers, stack pointer, and the return address is completely on the procedure.
- If you don't do it correctly you can mess things up far worse than is ever possible in higher level languages.

Data Space on the Stack

- You are probably more familiar with the stack as where local variables are stored. As you have seen so far, this isn't as simple as what we might tell you in PAD2, you only have to put values on the stack that can't be kept in registers.
- Using the frame pointer register can help when you have a function that puts things on the stack and varies the stack size.

Character Data

- MIPS provides instructions to load and store individual bytes as well as the 32-bit words. These instructions are lb and sb and they have the same format as the lw and sw instructions only they take the indicated byte and put it in the first byte of the indicated register.

Code

- Let's write some Fibonacci code in MIPS assembly.

Minute Essay

- Why do you think that MIPS put the burden of dealing with the stack squarely on procedures instead of introducing assembly language instructions that could do a lot of the work for you?
- Keep reading the book. I'm going to be posting assignment #3 shortly.