Architecture and Calling Conventions

February 14, 2001

CS 132: Compiler Design

SPARC Architecture Review

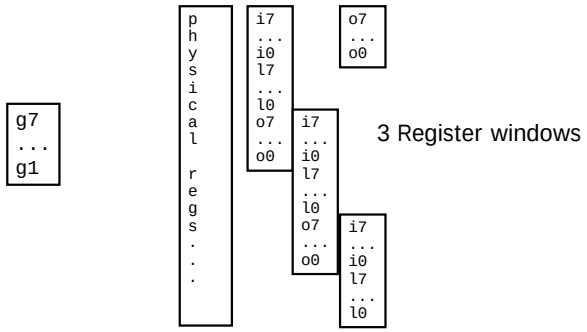
- Registers
  - 32 general-purpose registers (`%r0` -- `%r31`)
  - Each register has at least two names

`%r0`	= `%g0`	always zero
`%r[1-7]`	= `%g[1-7]`	global data
`%r[8-13]`	= `%o[0-5]`	arguments to other routines
`%r14 = %o6`	= `%sp`	stack pointer
`%r15`	= `%o7`	ret. addr. of callees
`%r[16-23]`	= `%l[0-7]`	local data
`%r[24-29]`	= `%i[0-5]`	arguments to current routine
`%r30 = %i6`	= `%fp`	frame pointer
`%r31`	= `%i7`	ret. addr for current routine

Register Windows

- The SPARC actually has ≥128 registers
  - But only 32 are visible at any one time.
  - To access the rest, one must use *register windows*
- Upon executing `save` instruction:
  - "out" registers now accessible with "in" names
  - 32 new registers allocated for "local" and "out"
  - "global" registers unaffected
- Opposite action on `restore`

Pictorial View



Register Window Overflow

- A SPARC processor can have from 64--528 physical GP registers.
  - 8 globals
  - 8 alternate globals
  - (n-16)/16 16-slot register windows
- If call stack is too deep, we run out of fresh physical registers.
  - Contents of oldest window dumped to the stack, and window is reused
  - Register contents must be reloaded upon `restore`

Subroutine Call and Return

- Caller:

```
call foo
nop
```

  - Subroutine call to label "`foo`"
  - Address of the call is put in `%o7`.
  - Program counter is set to address of label `foo`.
  - Recall: SPARC has delay slots, here filled by `nop`.

Subroutine Call and Return

- Callee prologue:

```
foo:  save %sp, -96, %sp
```

  - Instruction `save` updates register window
  - Also acts as an add (subtracts 96 from `%sp`)
    - Arguments are taken from *old* window, result is put in *new* window
    - In general, there will be some other number here depending on how much space the stack frame requires.

Subroutine Call and Return

- Callee epilogue:

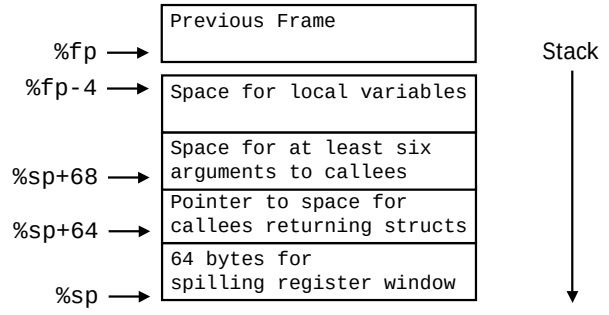
```
ret
restore
```

  - Instruction ret expands to `jmpl %i7+8, %g0`
    - Why `%i7` and not `%o7` ? Why 8 ?
  - Instruction `restore` is in the delay slot; pops register window.

Parameter Passing

- The first 6 arguments are passed in registers
  - %o0 through %o5
- Remaining arguments are passed on the stack.
  - In the caller's stack frame!
  - Caller: addresses %sp+92, %sp+96, ...
  - Callee: addresses %fp+92, %fp+96, ...

SPARC Stack Frame



Sparc Stack

- Minimum stack frame size of 96 bytes.
  64 + 4 + 24 + 4*overflow args + local storage
- Stack must always be doubleword aligned.
  - Why?

Return Values

- Normally a return value, if any, is put in %i0 by the called function.
  - Appears as %o0 to the caller.
- In C, can return a struct rather than pointer to struct.
  - Too big to fit in a single 32-bit register.
  - Trick: caller passes in address of space to create struct as an extra parameter.
  - Won't need to use this for compiling Tiger

Non-SPARC RISC

- No register windows.
  - (Generally) 32 general purpose registers, period.
- Registers are divided into
  - Caller-save:
    - Any subroutine may overwrite these registers.
    - Cannot keep important data here across procedure calls
  - Callee-save
    - Subroutine may modify these registers only if it restores their values before returning.

Alpha Architecture

- GP Registers
  - 32 general-purpose registers (`%r0` -- `%r31`)

`$0`	return value (caller-save)
`$[1-8]`	caller-save registers
`$[9-14]`	callee-save registers
`$15 = $fp`	frame pointer or callee-save
`$[16-21]`	integer arguments (caller-save)
`$[22-25]`	more caller-save registers
`$26 = $ra`	return address
`$27 = $pv`	address of procedure being called
`$28 = $at`	volatile scratch register
`$29 = $gp`	global pointer
`$30 = $sp`	stack pointer
`$31`	always zero

Alpha Call and Return

- Caller:

```
lda $pv, foo
jsr $ra, 0($pv)
ldgp $gp, 0($ra)
```

  - Subroutine call to label "`foo`"
  - Address of the following instruction is put in $ra.
  - Program counter is set to address of label `foo`.
  - No delay slot
  - After code returns, fix up $gp using the address of the `ldgp`

Subroutine Call and Return

- Callee prologue:

```
foo:  ldgp $gp, 0($pv)
      lda $sp, -16($sp)
      stq $26, 0($sp)
```

  - Instruction `save` updates register window
  - No requirement that
    - Generally return address will go on stack, though
    - And any callee-save registers that are modified

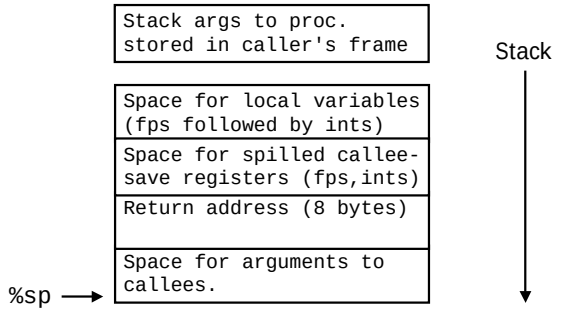
Subroutine Call and Return

- Callee epilogue:

```
ldq $26, 0($sp)
lda $sp, 16($sp)
ret $31, ($26)
```

  - Again, no delay slots

Alpha Stack Frame



Static Links

- Why is there no reference to static links?
  - Because not all languages require static links.
  - In particular, C or C++.
- Implementing static links is the responsibility of the compiler
  - The static link for each procedure will be *passed in* as an extra argument.
  - Each procedure in turn pass along the right static links for each called subroutine.
  - This must be stored on the stack in a known position in order for callees to find it.

Escaping Variables

- A variable is said to "escape" in Appel if it must be stored in memory.
- Variables escape if:
  - The programmer takes their address (C)
  - The variable is passed by reference (C++)
  - It is accessed by a nested function (Tiger)
- Homework for next week:
  - Do the following optional part (only) of Chapter 6: build the `FindEscape` structure.