Procedure Call

CS 217

Procedure Call

- Involves following actions

  pass arguments
  save a return address
  transfer control to <u>callee</u>
  transfer control back to <u>caller</u>
  return results

- Simplest example: leaf procedure (**`a=b*c;`**)

```
ld   b,%o0      ld   b,%o0
ld   c,%o1      call .mul
call .mul       ld   c,%o1
nop             st   %o0,a
st   %o0,a
```

Call/Return Instructions

- Procedures are called with either…

  `call` *label*

01	disp30
31	29

leaves PC (location of **`call`**) in **`%o7 (%r15`**)

  `jmpl` *address,reg*

10	reg	111000	rs1	0	0	rs2
10	reg	111000	rs1	1	simm13	
31	29	24	18	13	12	4

leaves PC in *reg*

Call/Return (cont)

- Indirect calls

  `jmpl` *reg***`,%r15`**

  jumps to the 32-bit address specified in *reg*

  leaves PC (return address) in **`%r15`**

  e.g., for function pointers **`a = (*apply)(b,c);`**

```
ld   b,%o0
ld   c,%o1
ld   apply,%o3
jmpl %o3,%r15; nop
st   %o0,a
```

Call/Return (cont)

- Procedure call return

  `jmpl %r15+8,%g0`

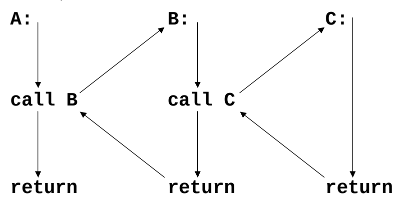
  transfers control from caller to callee

  other instructions: **`ret`** and **`retl`**

  why **`+8`**?

Nested/Recursive Calls

- **`A`** calls **`B`**, which calls **`C`**



must work when **`B`** is **`A`**

Nested/Recursive Calls (cont)

- Other requirements
  - pass a variable number of arguments
  - pass and return structures
  - allocate and deallocate space for local variables
  - save and restore caller’s registers
- <u>Entry</u> and <u>exit</u> sequences collaborate to implement these requirements

Fall 2001 7

Stack

- Procedure call information stored on stack
  - locals, including compiler temporaries
  - caller’s registers, if necessary
  - callee’s arguments, if necessary
- Sparc’s stack grows “down” from high to low address
- The stack pointer (**`%sp`**) points to top word on the stack (must be multiple of 8)

Fall 2001 8

Arguments and Return Values

- By convention
  - caller places arguments in the “out” registers
  - callee finds its arguments in the “in” registers
  - only the first 6 arguments are passed in registers
  - the rest are passed on the stack
- Registers at call time

<u>caller</u>	<u>callee</u>	
`%o7`	**`%i7`**	return address -8
`%o6`	**`%i6`**	stack/frame pointer
`%o5`	**`%i5`**	sixth argument
…	…	
`%o0`	**`%i0`**	first argument

Fall 2001 9

Arguments/Return Value (cont)

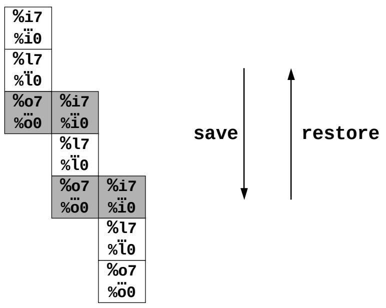
- Registers at return time

callee	caller	
`%i5`	`%o5`	sixth return value
`%i4`	`%o4`	fifth return value
…	…	
`%i0`	`%o0`	first return value

Register Windows

- Each procedure gets 16 “new” registers
- The window “slides” at call time
  - caller’s out registers become synonymous with callee’s in registers
- Instructions
  - **`save`** slides the window forward
  - **`restore`** slides the window backwards
  - decrement/increments CWP register
- Finite number of windows (usually 8)

Register Windows (cont)



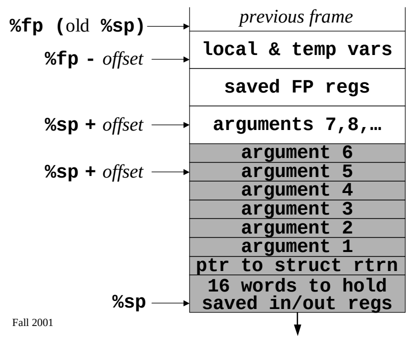
Window Managment

- Call time (**`save`**)
  - **`save %sp,`***N***`,%sp`** e.g., **`save %sp,-4*16,%sp`**
  - current window becomes previous window
  - decrements CWP and checks for <u>overflow</u>
  - adds *N* to the stack pointer (allocates *N* bytes if *N*<0)
  - if overflow occurs, save registers on the stack (must be enough stack space)
- Return time (**`restore`**)
  - previous window becomes current window
  - increments CWP and checks for <u>underflow</u>

Window Management (cont)

- In both **`save`** and **`restore`**
  - <u>source</u> registers refer to <u>current</u> window
  - <u>destination</u> registers refer to <u>new</u> window

Stack Frame



Example Stack Frames

```
main() {
   t(1,2,3,4,5,6,7,8);
}
t(int a1, int a2, int a3, int a4,
  int a5, int a6, int a7, int a8) {
    int b1 = a1;
    return s(b1,a8);
}
s(int c1, int c2) {
   return c1 + c2;
}
```

Fall 2001 16

Example (cont)

```
_main: save %sp,-104,%sp
       set 1,%o0
       set 2,%o1
       set 3,%o2
       set 4,%o3
       set 5,%o4
       set 6,%05
       set 7,%i5
       st %i5,[%sp+4*6+68]
       set 8,%i5
       st %i5,[%sp+4*7+68]
       call _t; nop
       ret; restore
```

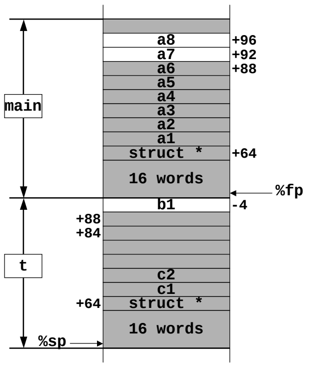
Fall 2001 17

Example (cont)

```
_t: save %sp,-96,%sp
    st %i0,[%fp-4]
    ld [%fp-4],%o0
    ld [%fp+96],%o1
    call _s; nop
    mov %o0,%i0
    ret; restore

_s: add %o0,%01,%o0
    retl; nop
```

Fall 2001 18


a8
+96
a7
+92
a6
+88
a5
a4
main
a3
a2
a1
struct *
+64
16 words
%fp
b1
-4
+88
+84
t
c2
c1
+64
struct *
16 words
%sp