

CS 415: Lecture 17

- Runtime (Cont'd)

Schedule

- Midterm Monday 3/27
- Project phase 3 due Monday 4/3
- HW3 due Monday 4/10

- Extra office hours: Th 9-10:30

Procedure Call

- To implement a procedure call
 - Caller may save some registers
 - Caller starts callee's activation record
 - Caller evaluates actual parameters and stores them in callee's activation record
 - Caller stores return address and stack_top value in callee's activation record
 - Caller sets access link (or manipulate display)
 - Caller increments stack_top to point within callee's activation record to beginning of storage for local variables
 - Callee may save some more registers into its activation record
 - Callee initializes local data and begins execution

Procedure Return

- What happens on the return?

Return Address

- Address of code instruction right after the call statement
- Put in a designated register by the calling procedure or on stack
- Return value of a function is also usually returned in a register

Parameter Passing

- Often, convention is defined so that first few parameters are passed in specific registers (4-6)
- May need to save registers to put the argument values into them. Why?
 - Most procedures are leaves of calling structure
 - Inter-procedural register allocation allows parameter passing in different registers
 - Needn't ever save dead variables
 - Register windows give fresh set of registers to each called function

Parameter Passing

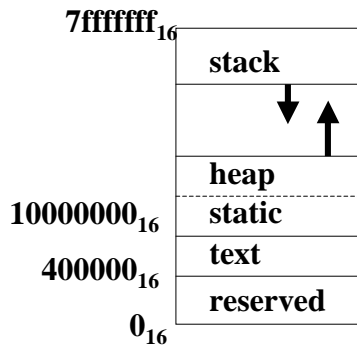
- *By value*
- By value result (copy in, copy out)
- By result
- *By reference*
- By name (by thunk)

Parameters

- **Some conventions are troublesome**
 - C requires all parameters be in consecutive storage words
 - C allows parameters to have their address taken (dangling pointer problem)

Case Study - SPIM

- A simulator that executes MIPS programs
- Target machine for our compiler project
- Memory model
 - Text - program code
 - Data
 - Static (globals)
 - Heap
 - Stack (runtime stack)



SPIM Calling Conventions

- MIPS CPU has 32 general purpose registers numbered 0-31
 - Register \$0 always contains the value 0
 - Registers \$at, \$k0, \$k1 are reserved, cannot be used by users
 - Registers \$a0-\$a3 are used to pass first 4 arguments to routines
 - Registers \$v0, \$v1 are used to return values from functions

SPIM Calling Conventions

- Registers $\$t0-\$t9$ are caller-save registers used to hold temporaries not preserved across calls
- Registers $\$s0-\$s7$ are callee-save registers, used to hold long-lived values across calls
- Register $\$gp$ is used to point to middle of 64K block of memory in the static data segment
- Register $\$sp$ is stack pointer, which points to first free location in stack; $\$fp$ is frame pointer; jai instruction writes $\$ra$, the return address from a procedure call

SPIM Register Usage - Summary

<u>Register</u>	<u>Number</u>	<u>Use</u>
$\$zero$	0	constant 0
$\$at$	1	reserved for assembler
$\$v0-\$v1$	2-3	expr eval and return values
$\$a0-\$a3$	4-7	arguments 1-4
$\$t0-\$t7$	8-15	temporary (not preserved)
$\$s0-\$s7$	16-23	saved temporary across calls
$\$t8-\$t9$	24-25	temporary (not preserved)
$\$k0-\$k1$	26-27	reserved for os
$\$gp$	28	pointer to global data area
$\$sp$	29	stack pointer
$\$fp$	30	frame pointer
$\$ra$	31	return address

SPIIM Procedure Call

- **Caller on executing the call**
 - Pass first 4 arguments in registers \$a0-\$a3; put rest of arguments at beginning of callee's frame
 - Save caller-save registers (\$a0-\$a3, \$t0-\$t9)
 - Execute *jai* which jumps to callee's first instruction and saves return address in \$ra

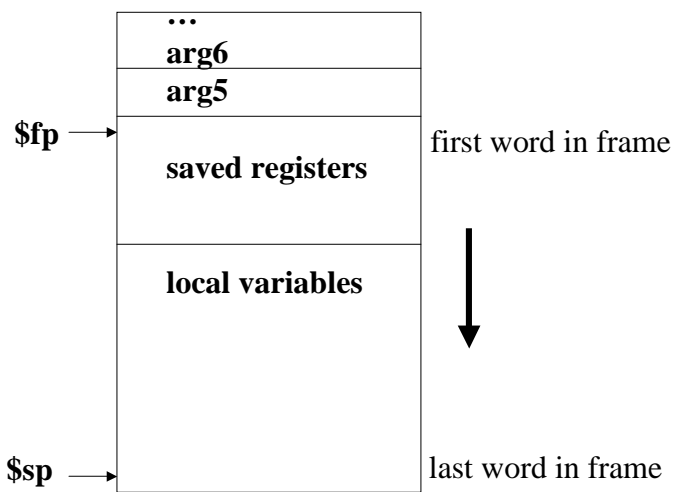
MIPS Calling Context Switch-2

- **Callee, before it starts running**
 - Allocate memory for frame (subtract frame size from stack pointer)
 - Save callee-save registers in frame (\$s0-\$s7, \$fp, and \$ra if callee makes a call)
 - Establish frame pointer in \$fp by subtracting stack frame's size minus 4 to \$sp

MIPS Calling Context Switch-3

- Before returning to the caller, the callee
 - Places return value in \$v0
 - Restores all callee-save registers
 - Pops stack frame by adding the frame size to \$sp
 - Return by jumping to address in \$ra

MIPS Frame



Factorial Example (SPIM manual, p A-26 ff)

```
C code for computing 10! :
main(){
    printf("the factorial of 10 is %d\n",fact(10));
}

int fact (int n){
    if (n < 1) return(1);
    else return (n*fact(n-1));
}
```

Example

Minimum frame size=24 bytes. Prologue code from main:

```
.text
.globl main
main:
    subu    $sp,$sp,32 #stack frame 32 bytes long
    sw     $ra,20($sp) #save return address in $ra
    sw     $fp,16($sp) #save old frame pointer
    addu   $fp,$sp,28 #setup frame pointer
```

Example

main calls factorial routine, passing it a single argument 10. after factorial returns, main calls library print routine `printf` and passes it both a format string and the result from factorial.

```
li    $a0,10      #put argument into $a0
jai   fact        #call factorial routine

la    $a0,$LC     #put format string into $a0
move  $a1,$v0     #move factorial result into $a1
jai   printf      #call print function
```

Example

main's epilogue restores saved registers, pop its stack frame and return.

```
lw    $ra,20($sp) #restore return address
lw    $fp,15($sp) #restore frame pointer
addu  $sp,$sp,32  #pop stack frame
jr    $ra         #return to caller

.rdata
$LC:
.ascii "The factorial of 10 is %d\n\000"
```

Example

factorial routine structured similarly to main. factorial prologue:

```
.text
fact:
    subu    $sp,$sp,32    #stack frame is 32 bytes
    sw     $ra,20($sp)    #save return address
    sw     $fp,16($sp)    #save frame pointer
    addu   $fp,$sp,28     #setup frame pointer
    sw     $a0,0($fp)     #save argument n
```

Example

computation within factorial routine:

```
    lw     $v0,0($fp)    #load n
    bgtz   $v0,$L2       #branch if n>0
    li     $v0,1          #return 1
    j      $L1            #jump to return code
$L2:
    lw     $v1,0($fp)    #load n
    subu   $v0,$v1,1     #compute n-1
    jai    fact           #recursive call of factorial
    lw     $v1,0($fp)    #load n
    mul    $v0,$v0,$v1   #compute factorial(n-1) * n
```

Example

factorial routine epilogue:

```
$L1:  
lw      $ra, 20($sp) #result in $v0  
lw      $fp, 16($sp) #restore $ra  
addu   $sp, $sp, 32 #pop stack  
j       $ra          #return to caller
```

Next Week

- Monday - Midterm
- Wednesday - Intermediate representation, ASU 8.1-8.3