MIPS Procedure Call Conventions

Alexander Nelson

February 12, 2021

University of Arkansas - Department of Computer Science and Computer Engineering

Register Conventions

Recall that 31 of 32 MIPS registers are general purpose

R0 is tied to 0

Developer is free to choose how to use these[1]

However

For compatibility, certain conventions (standards) are used

[1] $ra an exception

Register Conventions

Register Name	Number	Usage
$zero	0	Constant 0
$at	1	reserved for assembler
$v0-$v1	2-3	expression evaluation/results
$a0-$a4	4-7	arguments
$t0-$t7	8-15	temporary registers
$s0-$s7	16-23	saved registers
$t8-$t9	24-25	temporary registers
$k0-$k1	26-27	reserved for kernel
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Register Conventions

$at, $k0, $k1 all reserved for assembler and kernel

Do not use these in your assembly code!

$a0-$a3 – arguments passed to routines

Any additional arguments must be passed on the stack

$v0, $v1 – registers to return values from function

Note, two return values (not available by default in C/C++)

Register Conventions

$t0-$t9 – caller-saved registers

Need not be preserved across calls

$s0-$s7 – callee-saved registers

Hold long-lived values that are preserved across calls

Register Conventions

$gp – global pointer
Points to middle of a 64K block of memory in static data segment

$sp – stack pointer
Points to last location on the stack

$fp – frame pointer
Points to beginning of call frame

$ra – return address
Hardware places instruction counter into this register on a jump-and-load

Procedure Calls

Procedure Call Frame (a.k.a stack frame) – block of memory for procedure bookkeeping

Purposes:

- Hold values passed to procedure as arguments
- Save registers that a procedure may modify, but callee does not want changed ($sx regs, etc.)
- Provide space for variables local to procedure

Procedure Calls

Most languages use a LIFO return structure

Memory can be allocated as a stack!

Procedure Calls

Calling function responsible for:

1. Push any arguments past first four
2. Save caller-saved registers (any that it expects to use later $tx, $ax)
3. Execute a jal

Called function responsible for:

1. Allocate memory for the frame by subtracting frame's size from stack pointer
2. Save callee-saved registers in the frame ($sx, $fp, $ra)
3. Establish a frame pointer by adding stack frame size minus 4 to $sp, storing sum in $fp

Procedure Calls

To return, called function must

1. If callee is function that returns value, populate $v0
2. Restore all callee-saved regs that were saved on entry
3. Pop stack frame by adding frame size to $sp
4. Return by jumping to address in $ra

Quick Note

If a language does not allow recursion, do not need a stack

Why?

Quick Note

If a language does not allow recursion, do not need a stack

Can statically allocate memory to each function that may be called