Prelim Review

Kevin Walsh
CS 3410, Spring 2010
Computer Science
Cornell University

FAQ

- caller/callee saved registers
- CPI
- writing assembling
- reading assembly

Caller-save: If necessary… ($t0 .. $t9)

- save before calling anything; restore after it returns

Callee-save: Always… ($s0 .. $s7)

- save before modifying; restore before returning

Caller-save registers are responsibility of the caller

- Caller-save register values saved only if used after call/return
- The callee function can use caller-saved registers with concern

Callee-save register are the responsibility of the callee

- Values must be saved by callee before they can be used
- Caller can assume that these registers will be restored

Caller-save: If necessary… ($t0 .. $t9)

- save before calling anything; restore after it returns

Callee-save: Always… ($s0 .. $s7)

- save before modifying; restore before returning

eax, ecx, and edx are caller-save…

- … a function can freely modify these registers
- … but must assume that their contents have been destroyed if it in turns calls a function.

ebx, esi, edi, ebp, esp are callee-save

- A function may call another function and know that the callee-save registers have not been modified
- However, if it modifies these registers itself, it must restore them to their original values before returning.

Caller-save: If necessary… ($t0 .. $t9)

- save before calling anything; restore after it returns

Callee-save: Always… ($s0 .. $s7)

- save before modifying; restore before returning

A caller-save register must be saved and restored around any call to a subprogram.

In contrast, for a callee-save register, a caller need do no extra work at a call site (the callee saves and restores the register if it is used).

Caller-save: If necessary… ($t0 .. $t9)

- save before calling anything; restore after it returns

Callee-save: Always… ($s0 .. $s7)

- save before modifying; restore before returning

CALLER SAVED: MIPS calls these temporary registers, $t0-t9

- the calling program saves the registers that it does not want a called procedure to overwrite
- register values are NOT preserved across procedure calls

CALLEE SAVED: MIPS calls these saved registers, $s0-s8

- register values are preserved across procedure calls
- the called procedure saves register values in its AR, uses the registers for local variables, restores register values before it returns.

Caller-save: If necessary… ($t0 .. $t9)

- save before calling anything; restore after it returns

Callee-save: Always… ($s0 .. $s7)

- save before modifying; restore before returning

Registers $t0-$t9 are caller-saved registers

- … that are used to hold temporary quantities
- … that need not be preserved across calls

Registers $s0-s8 are callee-saved registers

- … that hold long-lived values
- … that should be preserved across calls

caller-saved register

- A register saved by the routine being called

callee-saved register

- A register saved by the routine making a procedure call

CPI

Cycles Per Instruction

A measure of latency (delay)?

"ADD takes 5 cycles to finish"

or

A measure of throughput?

"N ADDs are completed in N cycles"

CPI = weighted average *throughput* over all instructions *in a given workload*

CPI = 1.0 means that on average…

… an instruction is completed every 1 cycle

CPI = 2.0 means that on average…

… an instruction is completed every 2 cycles

CPI = 5.0 means that on average…

… an instruction is completed every 5 cycles

CPI = 1.0 means that on average…

… an instruction is completed every 1 cycle

Single cycle CPU

≈ Your MIPS CPU

if pure arith. workload

CPI = 2.0 means that on average…

… an instruction is completed every 2 cycles

- Your CPU: if every inst. still is once. (illegal)
- Multi-cycle CPU

CPI = 0.5 means that on average…

… an instruction is completed every 0.5 cycles

→ Dual Core

Suppose 10 stage pipeline and…

- 1 instruction zapped on every taken jump or branch
- 3 stalls for every memory operation

Q: What is CPI?

… for pure arithmetic workload? 1.0

… for pure memory workload? 4.0

… for pure jump workload? 2.0

… for 50/50 arithmetic/jump workload? 1.5

… for 50%/25%/25% arith/mem/branch? ×

… if one fifth of the branches are taken? 1.8

50% 1
25% 4
20% 1
5.2