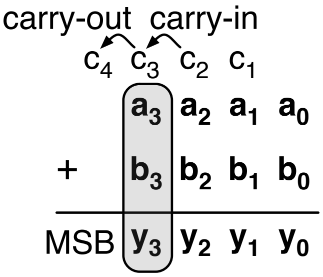
CS 61C
Spring 2010

Signed Arithmetic & MIPS

Scott Beamer (cs61c-ta)

Week 5 (2/16)

Overflow for Signed Numbers



- *Overflow* - if the result of an arithmetic operation is incorrect due to not enough bits in the result
- *Carry-out* - if the carry-out of the MSB is 1
- Overflow is impossible if the sign of the two inputs is different
- Can you think of a way to detect overflow from the carry-in and carry-out of the MSB?

Unsigned Instructions

- As mentioned in lecture, the term ‘unsigned’ is overloaded in MIPS
- Do/don’t sign extend loaded byte - lb / lbu
- Do/don’t detect overflow - add, addi, sub, mult, div / addu, addiu, multu, divu
- Do signed / unsigned compare - slt, slti / sltu, sltiu

Decisions

- We can make any comparison (<, >, <=, >=, ==, !=) using slt and beq/bne
- Can reverse the arguments to slt to get sget
- Fill in the gaps in table:

C	MIPS

int total = 0;
int *end = start + n;
while(start != end) {
  total += start[0];
  start++;
}
return total;
	sll $a1, $a1, 2
	add $a1, $a1, $a0
	add $v0, $0, $0
	beq $a1, $a0, L2
	L1: lw $t0, 0($a0)
	add $v0, $v0, $t0
	addi $a0, $a0, 4
	bne $a1, $a0, L1
	L2: jr $ra

switch(op) {
  case '+':
    result = x + y; break;
  case '-':
    result = x - y; break;
  case '&':
    result = x & y; break;
  case '|':
    result = x | y; break;
}
return result;
 	
// Mappings
result→$v0 x→$a0 y→$a1 op→$a2
	addi $t0, $0, 43
	addi $t1, $0, 45
	addi $t2, $0, 38
	addi $t3, $0, 124
	beq $a2, $t0, plus
	beq $a2, $t1, sub
	beq $a2, $t2, and
	beq $a2, $t3, or
	plus: add $v0, $a0, $a1
	j done
	sub: sub $v0, $a0, $a1
	j done
	and: and $v0, $a0, $a1
	j done
	or: or $v0, $a0, $a1
	j done
	done: jr $ra

MIPS Register Conventions

- *Caller* - the calling function
- *Callee* - the function being called
- *Saved* - $s0-s7, $sp
- *Volatile* - $ra, $v0-v1, $a0-a3, $t0-t9, $at
- *Responsibility of Caller* - save any volatile registers it will want after the function call
- *Responsibility of Callee* - save any saved registers it wants to use during function call

Steps to Save Registers

- *Prologue* - move $sp down, store registers into memory offset from $sp
- *Body* - do whatever required saving
- *Epilogue* - load registers from memory offset from $sp, move $sp back up

MIPS Registers (registers in italics aren't needed for CS 61C)

Register #	Register Name	Use
$0	$zero	constant source of 0
$1	$at	used by the assembler
$2-3	$v0-v1	used to return values from a function
$4-7	$a0-a3	used to pass argument into a function
$8-15	$t0-t7	store *temporary* values
$16-23	$s0-s7	store *saved* values
$24-25	$t8-t9	store *temporary* values
$26-27	*$k0-k1*	*used by the kernel*
$28	*$gp*	*global pointer*
$29	$sp	stack pointer
$30	*$fp*	*frame pointer*
$31	$ra	return address

MIPS Register Saving Practice

- Fill in the prologue and epilogue of the caller and callee
- Caller uses: $t0, $s2, $a0, ($ra)
- Callee uses: $s0, $s1, $t2, $v0

Caller	Callee

`# Prologue`
`addi $sp, $sp, -12`
`sw   $t0, 8($sp)`
`sw   $a0, 4($sp)`
`sw   $ra, 0($sp)`
`# Body`
`jal CALLEE`
`# Epilogue`
`lw   $t0, 8($sp)`
`lw   $a0, 4($sp)`
`lw   $ra, 0($sp)`
`addi $sp, $sp, 12`
	`# Prologue`
	`addi $sp, $sp, -8`
	`sw   $s0, 4($sp)`
	`sw   $s1, 0($sp)`
	
	`# Body`
	`# Epilogue`
	`lw   $s0, 4($sp)`
	`lw   $s1, 0($sp)`
	`addi $sp, $sp, 8`