beq $s0, $s1, L1 → ± 2^15

if L1 ≥ 2^15 → bne $s0, $s1, L2
j L1
L2: ....

MIPS Addressing Modes

- Immediate addressing, where the operand is a constant within the instruction itself. addi ...
- Register addressing, where the operand is a register. add $s0, $s1, $s2
- Base or displacement addressing, where the operand is at the memory location whose address is the sum of a register and a constant in the instruction. lw $t0, 4($s0) EA: 4 + $s0
- PC-relative addressing, where the branch address is the sum of the PC and a constant in the instruction. PC+4 beq L1
- Pseudo-direct addressing, where the jump address is the 26 bits of the instruction concatenated with the upper bits of the PC

append 00, to make it word aligned → 28 bits

max jump address = 2^28

Procedure call essentials

Step(1)
→ Caller at call time
→ put arguments in $a0 - $a3
→ save any caller-save temporaries
→ jal ... → jump and link, ret address saved in $ra

Step(2)
→ Caller at entry
→ allocate stack space
→ save $ra, $s0 ... $s7 if necessary

return address

MIPS Convention

Temporary & Saved registers
Caller saved registers + Callee saved registers

$Vx → for func. retval
$a0-$a9 - for passing args
$tx - Caller saved regs (saved before call)
$Sx - Callee saved regs

Step (3)
↳ Callee at exit

We save return addr in $ra & stack because when we call a function in a function
A → B → C

⇒ restore $ra, $Sx
⇒ deallocate $ra, $Sx
⇒ dellocate all stack space
⇒ Put all return value $Vx

Step (4) ⇒ caller after return
⇒ retrieve return values from $Vx
⇒ restore $tx