Branch Addressing (PC-relative addressing)

It is convenient for hardware to increment PC early to point to the next instr.

MIPS addr → relative to the addr of the following instr (PC+4)

Target Addr = PC + offset × 4

(PC: incremented by 4)

* MIPS instr are 4-bytes long.
PC relative addressing refer to number of words to the next instructions instead of number of bytes.
(16-bit field → branch four times further distance)

Jump Addessing (Pseudo-direct Addr.)

Target Addr = $PC_{31..28}$: (addr x 4)

↘ concatenate

$t0 - $t7 (8-15)	$s0 - 16
$s0 - $s7 (16-23)	$1 - 17
	$s2 - 18
	$s3 - 19

Slide 6.9 - Examples

```
addi  $s4, $s4, 100   (Imm)
add   $s1, $s2, $s3   (Reg)
lw    $s1, 20($s2)    (Base Addr)
beq   $s1, $s2, label (PC-rel.)
j     label           (Pseudo-direct)
```

```
void swap(int v[], int k)
{
  int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

- v in $a0, k in $a1, temp in $t0

```
void sort (int v[], int n)
{
  int i, j;
  for (i = 0; i < n; i += 1) {
    for (j = i - 1;
         j >= 0 && v[j] > v[j + 1];
         j -= 1) {
      swap(v,j);
    }
  }
}
```

- v in $a0, k in $a1, i in $s0, j in $s1

Move : Pseudoinstruction

move $s0, $zero

↳ add $s0, $zero, $zero

i → $s0 V → $a0

J → $s1 k → $a1

(Note: $s0, $s1, $s2, $s3 used in procedure)

Slide 87

Impact of compiler opt on sort program (performance, IC, CC, CPI)

CPU time = IC x CPI x CC

Slide 92

Array Version (on the left)

array → $a0
size → $a1
i → $t0

(Assumes size is greater than 0)

Multiply & add inside the loop.

(i is incremented & each addr must be recalculated from the new index)

Pointer Version (on the right)

array → $a0
size → $a1
p → $t0

Assign p to the addr. of first element of array

increment pointer "p" directly.

increment pointer by 1 means move pointer to the next sequential object.

Since p is a pointer to integers, each in 4 bytes ⇒ Compiler increments p by 4.

Comparing two versions:

array version → execute 6 instr per iteration

pointer version → execute 4 instr per iteration.

Compiler opt

strength reduction (shift instead of multiply)

induction variable elimination (eliminate array address calculations within loops)