The load and store instructions

load → copies data from memory to a register
store → copies from a register to memory

Syntax (load/store word)

lw dest, address
sw src, address

Some GPR

The address can have two forms

address → constant(GPR)
→ (GPR)

0(GPR) is equivalent to just (GPR)

- the effective address is GPR + constant

Examples:

```
lw   $s0, -8($s1)     # copy word from address ...
                      # ... $s0 + offset (-8) to $s0

sw   $s0, 0($s1)      # copy from $s0 to address ...
                      # ... pointed to by $s1
```

can omit the '0'

Array elements, registers and memory

Arrays must reside in memory; array elements are accessed by their memory addresses using address arithmetic

- registers don't have addresses
- No. of registers too small to hold arrays

C-language example

declaration	register allocation
int *p;	$s0
int k;	$s1

Translation of `p[10] = p[20] + k;`

```
lw    $t0, 80($s0)      # $t0 = p[20]
```

→ p[20] is offset 80 bytes from the 0th element
- 4 bytes per array element (important!)

```
add   $t0, $t0, $s1     # $t0 = $t0 + k
sw    $t0, 40($s0)      # p[10] = $t0
```

↑ offset for p[10]

Decision making and branching

To successfully translate a C program with if statements, we need a mechanism to skip machine instructions

Assume:

variable	register
int x;	$s0
int y;	$s1
int z;	$s2

C-code:

```
if (x == y)
    z = 0;
z = z + x;
```

Translation:

```
    bne   $s0, $s1, L1      # if (x != y) goto L1
    add   $s2, $zero, $zero # z = 0
L1:
    add   $s2, $s2, $s0     # z = z + x
```

Notes:

- bne means "branch if not equal"
- L1 is a symbol representing the address to which we wish to branch (the PC will be assigned this address if the test is true)

Two ways to skip instructions : branch and jump

branch ⟶ test a condition ; if true, go to some address
jump ⟶ no condition ; just go !

MIPS instructions available :

beq ⟶ branch if registers equal
bne ⟶ not equal
j ⟶ jump ! (like a "goto")

Loop example :

Coded in the same way as loops in "goto-C". Manage them manually.

declaration	registers	C-code
int *a;	$s0	
int n;	$s1	sum = 0;
int sum;	$s2	i = 0;
int i;	$s3	while (i != n) {
		sum += a[i];
		i++;
		}

Translation

```
        add   $s2, $zero, $zero      # sum = 0
        add   $s3, $zero, $zero      # i = 0
L1:
        beq   $s3, $s1, L2           # if (i==n) goto L2
        sll   $t0, $s3, 2            #  $t0 = i << 2;
        add   $t1, $t0, $s0          # $t1 = &a[i]
        lw    $t2, ($t1)             # $t2 = a[i]
        add   $s2, $s2, $t2          # sum += a[i]
        addi  $s3, $s3, 1            # i++
        j     L1
L2:                                  <- while-loop exit point
```

Multiply i by 4 ("shift-left logical") ⟶ sll

C-language left-shift operator ⟶ <<