## In More Depth: The IBM/Motorola PowerPC

The PowerPC, made by IBM and Motorola and used in the Apple Macintosh, shares many similarities to MIPS: both have 32 integer registers, instructions are all 32 bits long, and data transfer is possible only with loads and stores. The primary difference is two more addressing modes plus a few operations.

### Indexed Addressing

In the examples above we saw cases where we needed one register to hold the base of the array and the other to hold the index of the array. PowerPC provides an addressing mode, often called *indexed addressing*, that allows two registers to be added together. The MIPS code

```
add $t0,$a0,$s3   # $a0 = base of array, $s3 = index
lw  $t1,0($t0)    # reg $t1 gets Memory[$a0+$s3]
```

could be replaced by the following single instruction in PowerPC:

```
lw  $t1,$a0+$s3   # reg $t1 gets Memory[$a0+$s3]
```

Using the same notation as Figure 2.24 on page 101, Figure 2.1.1 shows indexed addressing. It is available with both loads and stores.

### Update Addressing

Imagine the case of a code sequence marching through an array of words in memory, such as in the array version of `clear1` on page 130. A frequent pair of operations would be loading a word and then incrementing the base register to point to the next word. The idea of *update addressing* is to have a new version of data transfer instructions that will automatically increment the base register to point to the next word each time data is transferred. Since the MIPS architecture uses byte addresses and words are 4 bytes, this new form would be equivalent to this pair of MIPS instructions:

```
lw  $t0,4,($s3)   # reg $t0 gets Memory[$s3+4]
addi$s3,$s3,4     # $s3 = $s3 + 4
```

The PowerPC includes an instruction like this:

```
lwu $t0,4($s3)    # reg $t0=Memory[$s0+4]; $s3 = $s3+4
```

That is, the register is updated with the address calculated as part of the load.

Figure 2.1.1 also shows update addressing. PowerPC has update addressing options for both base and indexed addressing, and for both loads and stores.

## Unique PowerPC Instructions

The PowerPC instructions follow the same architecture style as MIPS, largely relying on fast execution of simple instructions for performance. Here are a few exceptions.

The first is load multiple and store multiple. These can transfer up to 32 words of data in a single instruction and are intended to make fast copies of locations in memory by using load multiple and store multiple back to back. They also save code size when saving or restoring registers.

A second example is loops. The PowerPC has a special counter register, separate from the other 32 registers, to try to improve performance of a *for* loop.

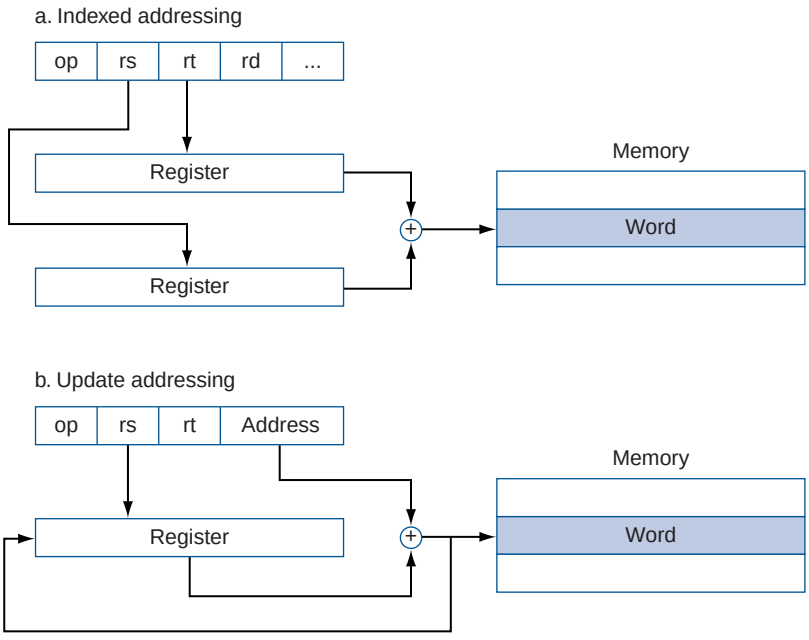


**FIGURE 2.1.1 Illustration of (a) indexed and (b) update addressing mode.** The operand is shaded in color.

Suppose we wanted to execute the following C code:

```
for (i = n; i != 0; i = i - 1)
    {. . .};
```

If we want to decrement a register, compare to 0, and then branch as long as the register is not 0, we could use the following MIPS instructions:

```
Loop:...
  addi  $t0,$t0,-1       # $t0 = $t0 - 1
  bne   $t0,$zero, Loop  # if $t0 != 0 go to Loop
```

In PowerPC we could use a single instruction instead:

```
  bc    Loop,$ctr!=0     # $ctr = $ctr - 1;
                         # if $ctr != 0 go to Loop
```

**2.45** [10] <§2.12> Consider an architecture that is similar to MIPS except that it supports update addressing (like the PowerPC) for data transfer instructions. If we run SPEC2000int using this architecture, some percentage of the data transfer instructions shown in Figure 2.48 on page 146 will be able to make use of the new instructions, and for each instruction changed, one arithmetic instruction can be eliminated. If 25% of the data transfer instructions can be changed, which will be faster for SPEC2000int, the modified MIPS architecture or the unmodified architecture? How much faster? (You can assume that both architectures have CPI values as given in Exercise 2.51 and that the modified architecture has its cycle time increased by 10% in order to accommodate the new instructions.)

**2.57** [5] <§2.19> Implement the following C code in MIPS and in PowerPC using both indexed addressing and update addressing:

```
for (i=0; i != 10; i++) {
    a[2i] = b[i] + i;
}
```

Assume that the base address at `a[]` is stored in `$a0`, and the base address of `b[]` is stored in `$a1`.

**2.58** [15] <§2.19> Implement the following C code in MIPS and in PowerPC taking advantage of the PowerPC's addressing modes and unique instructions:

```
    freq = 0;
    for  (i=0; i != 100; i++) {
        x = 0;
        for (j=100; j != 0; j--) {
             if (a[i] == a[j])
                        x++;
        }
        if (x >= freq)
             freq = x;
}
```

Assume that `freq` corresponds to `$v0`, `x` corresponds to `$s0`, and the base address of `a[]` is stored in `$a0`.