

## Study Questions for Patterson & Hennesey, Section 2.6

### Section 2.6

In this section, you are introduced to the I-format instructions **bne**, **beq**, the R-format instruction **slt**, and the J-format instruction **j**.

1. Can you represent the pseudo-ops **blt**, **ble**, **bgt**, **bge**, and **b** using only the above-named instructions? Be sure to use no additional temporary registers except possibly \$at.
2. What is the name of the register in which the address of the next instruction to be executed is always stored?
3. Why is that register always automatically incremented by 4?
4. The unconditional branch pseudo-op, **b**, and the jump instruction, **j**, both transfer control unconditionally to a labeled location. In the case of branching, what value is stored in the low-order 16 bits of the branch instruction? [If you answered “offset” you are only partly correct. Offset from where?]
5. In the case of jumping, transfer of control is also made unconditionally to the labeled location, but now the address of the location is stored in the jump instruction. How can a 32-bit address fit in the low order 26 bits of the jump instruction? More precisely, where does the jump instruction get its other 6 needed bits? [It gets the bits in two different ways. Mention both ways.]
6. Let's do the arithmetic. If a **page** is defined as all the memory locations which share the same high order (most significant) 4 bits, then the number of pages that can be addressed at all in MIPS is  $2^4$ . Since there are a total of  $2^{32}$  memory locations, how many memory locations are there on each page, as a power of 2? As a decimal number? [You can approximate the decimal number, but remember that we usually use the kilo- prefix to mean  $2^{10}$  for RAM and  $10^3$  for disk space, even though  $2^{10} \approx 10^3$ .]
7. Which of **b** or **j** can transfer control across a page boundary in RAM?
8. Which of **b** or **j** can transfer control to the RAM location farthest away?
9. If you wrote an assembler and you were lazy, you might flag as an error any attempt to jump to an off-page label, say on the next page. Such a “far jump” can't be done directly. But as I hinted hastily in the last 2 minutes of my class lecture, one can do a “jump fix.”<sup>1</sup> If a jump instruction is on RAM page 5, and the label is near the beginning of page 6, the assembler could translate the jump into two instructions, the first a jump to the last memory location on page 5; and the second, a branch from there across the page directly to the labeled instruction on page 6. This only works if the assembler makes decisions about the last word of page 5. The assembler could get to decide if we agree to specify addresses using only **names** for addresses. Can we specify numerical addresses instead? Sure! We can execute `j 0xFFFFF0FC` and thereby go directly to the last instruction on a page. But we are not wise to do that. That would prevent the assembler from using that RAM location to do a far jump. There is one exception: In the course Operating Systems, we will learn that sometimes we need to address exact memory locations for relating to input/output devices that use something called “memory mapped” i/o.

Why is `0xFFFFF0FC` the last address on a page, and not `0xFFFFFFF` ?

lect06.pdf v1.0

---

<sup>1</sup> Microsoft Assembler 6.0 (MASM) does jump fixes. Many C compilers allow you to hint that data may be placed on a distant page if it is convenient. To see examples, search Google for `const char FAR*`