

Study Questions for Patterson & Hennesey, Sections 2.7–13

Section 2.7

The key ideas here are the conventional use of certain registers in subroutine calling, and the memory map of RAM. You are not responsible for \$gp and \$fp. You are responsible conceptually to know what to do with subroutines with more than 2 parameters, but we won't be practicing it for homework.

The major problem that folks encounter is forgetting when a register contains a value and when a register contains an address (or we'll say “pointer”) to a value. In

```
lw $t0, 4($sp)      or      addi $sp, $sp, -4
```

we are storing an address in \$sp, and doing arithmetic on the address. Whereas we are storing a value in \$t0.

The conventions are listed on p. 79 and again in Appendix A.6. Learn them!

To see how valuable employers regard this knowledge, I link here to an exam question ([page 1](#), [page 2](#)) on a long exam that my son-in-law took in applying for a new job to program embedded systems for a government contractor. They surprised him with the exam in the interview. He gives me permission to let you see his hasty answer.

Section 2.8

The usefulness of bytes and halfwords is motivated by ASCII characters (bytes) in strings in C or C++, and Unicode characters (halfwords) in Strings in Java.

Section 2.9

How values are handled that are longer than can fit in an instruction is discussed. Two examples are the load address instruction, which loads a 32-bit constant address specified by a label somewhere,

```
la $t0, Address
```

and the jump instruction,

```
j      NewLocation
```

An end-of-chapter question (2.37, on page 152) explores what would happen for other instructions if the numbers got too big. For example, beq is an operation if it is applied to two registers, but if it is applied to a register and a constant number, like

```
beq   $t5, 3000, QuitLoop
```

then it would have to be assembled into two instructions, one of which gets 3000 into some register, probably \$at [Why?]. What would these instructions be?

Given an instruction, you should be able to write it in binary.

Given binary, you should be able to write the instruction.

Section 2.13

Now we introduce subroutines which call other subroutines. Since always the main program saves \$ra to the stack before it makes a subroutine call and restores \$ra from the stack after return, a subroutine which calls another subroutine behaves as the second subroutine's main program, so it too must save and restore \$ra.

By using the sort example from this section, I was able to illustrate passing an array as a parameter. We pass the *address* of the array. Recall my comment above about the major problem in this section.

We now have two examples of the use of jr: subroutine calling and jump tables.