4.13 Key Terms

These terms reflect the key ideas in the chapter. Check the Glossary for definitions of the terms you are unsure of.

AND gate  floating point  round
AND operation  guard  scientific notation
arithmetic logic unit (ALU)  hexadecimal  significant
biased notation  least significant bit  single precision
devisor  most significant bit  sticky bit
double precision  normalized  underflow
exclusive OR gate  overflow  units in the last place (ulp)
exponent

4.14 Exercises

Never give in, never give in, never, never, never—in nothing, great or small, large or petty—never give in.

Winston Churchill, address at Harrow School, 1941

4.1 [§4.2] Convert \(512_{\text{ten}}\) into a 32-bit two’s complement binary number.

4.2 [§4.2] Convert \(-1,023_{\text{ten}}\) into a 32-bit two’s complement binary number.

4.3 [§4.2] Convert \(-4,000,000_{\text{ten}}\) into a 32-bit two’s complement binary number.
4.4 [5] <§4.2> What decimal number does this two’s complement binary number represent: \text{11111111111111111111111000001100}_\text{two}? \\
4.5 [5] <§4.2> What decimal number does this two’s complement binary number represent: \text{11111111111111111111111111111111}_\text{two}? \\
4.6 [5] <§4.2> What decimal number does this two’s complement binary number represent: \text{01111111111111111111111111111111}_\text{two}? \\
4.7 [5] <§4.2> What binary number does this hexadecimal number represent: \text{fff ffa}_{\text{hex}}? What decimal number does it represent? \\
4.8 [5] <§4.2> What hexadecimal number does this binary number represent: \text{1100101011111011101011001110}_\text{two}? \\
4.9 [5] <§4.2> Why doesn’t MIPS have a subtract immediate instruction? \\
4.10 [10] <§4.2> Find the shortest sequence of MIPS instructions to determine the absolute value of a two’s complement integer. Convert this instruction (accepted by the MIPS assembler):

\text{abs $t2, $t3}

This instruction means that register $t2$ has a copy of register $t3$ if register $t3$ is positive, and the two’s complement of register $t3$ if $t3$ is negative. (Hint: It can be done with three instructions.)

4.11 [10] <§4.2> Two friends, Harry and David, are arguing. Harry says, “All integers greater than zero and exactly divisible by six have exactly two 1s in their binary representation.” David disagrees. He says, “No, but all such numbers have an even number of 1s in their representation.” Do you agree with Harry or with David, or with neither? (Hint: Look for counterexamples.)

4.12 [15] <§4.4> Consider the following code used to implement the instruction

\text{sllv $s0, $s1, $s2}

which uses the least significant 5 bits of the value in register $s2$ to specify the amount register $s1$ should be shifted left:

\begin{verbatim}
.data
mask: .word 0xfffff83f
.text
start: lw $t0, mask
lw $s0, shifter
and $s0,$s0,$t0
andi $s2,$s2,0xf
sll $s2,$s2,6
or $s0,$s0,$s2
sw $s0, shifter
shifter: sll $s0,$s1,0
\end{verbatim}
Add comments to the code and write a paragraph describing how it works. Note that the two `lw` instructions are pseudoinstructions that use a label to specify a memory address that contains the word of data to be loaded. Why do you suppose that writing "self-modifying code" such as this is a bad idea (and oftentimes not actually allowed)?

4.13 [10] <§4.2> If $A$ is a 32-bit address, typically an instruction sequence such as

```assembly
lui $t0, A_upper
ori $t0, $t0, A_lower
lw $s0, 0($t0)
```

can be used to load the word at $A$ into a register (in this case, $s0$). Consider the following alternative, which is more efficient:

```assembly
lui $t0, A_upper_adjusted
lw $s0, A_lower($t0)
```

Describe how $A_{\text{upper}}$ is adjusted to allow this simpler code to work. (Hint: $A_{\text{upper}}$ needs to be adjusted because $A_{\text{lower}}$ will be sign-extended.)

4.14 [15] <§§3.4, 4.2, 4.8> The Big Picture on page 299 mentions that bits have no inherent meaning. Given the bit pattern:

```
1000 1111 1110 1111 1100 0000 0000 0000
```

what does it represent, assuming that it is

- a two's complement integer?
- an unsigned integer?
- a single precision floating-point number?
- a MIPS instruction?

You may find Figures 3.18 (page 153), 4.48 (page 292), and A.19 (page A-54) useful.

4.15 [10] <§§4.2, 4.4, 4.8> This exercise is similar to Exercise 4.14, but this time use the bit pattern:

```
0000 0000 0000 0000 0000 0000 0000 0000
```

4.16 [10] <§4.3> One of the differences between Sun's SPARC architecture and the MIPS architecture we've been studying is that the load word instruction on the SPARC can specify the address either as the sum of two registers'