Chapter 3 Solutions: For More Practice

3.8 You should be quite suspicious of both claims. A simple examination yields

6 = 2 + 4
12 = 4 + 8
18 = 2 + 16
24 = 8 + 16
30 = 2 + 4 + 8 + 16 (so we know Harry is wrong)
36 = 4 + 32
42 = 2 + 8 + 32 (so we know David is wrong).

3.11

```
subu $t3, $t5, $t7
sltu $t2, $t5, $t7
add  $t2, $t6, $t2
sub  $t2, $t4, $t2
```

unsigned

```
subu $t3, $t5, $t7
sltu $t2, $t5, $t7
addu $t2, $t6, $t2
subu $t2, $t4, $t2
```

3.14

<<This solution destroys the value of `$s0`>>

```
addu $s1, $s0, $zero
sll  $s0, $s0, 3
addu $s1, $s1, $s0
```

<<This solution preserves `$s0`>>

```
addu $s1, $s0, $zero
sll  $t0, $s0, 3
addu $s1, $s1, $t0
```

3.15 No solution provided.

3.16 No solution provided.

3.17 No solution provided.

3.18 C1 = c4, C2 = c8, C3 = c12, and C4 = c16.

$c4 = G_{3,0} + (P_{3,0} \cdot c0)$.

c8 is given in the exercise.

$c12 = G_{11,8} + (P_{11,8} \cdot G_{7,4}) + (P_{11,8} \cdot P_{7,4} \cdot G_{3,0}) + (P_{11,8} \cdot P_{7,4} \cdot P_{3,0} \cdot c0)$.

$$c16 = G_{15,12} + (P_{15,12} \cdot G_{11,8}) + (P_{15,12} \cdot P_{11,8} \cdot G_{7,4}) + (P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot G_{3,0}) + (P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot P_{3,0} \cdot c0).$$

3.19 The equations for c4, c8, and c12 are the same as those given in the solution to Exercise 3.18. Using 16-bit adders means using another level of carry lookahead logic to construct the 64-bit adder. The second level generate, G0′, and propagate, P0′, are

$$G0' = G_{15,0} = G_{15,12} + P_{15,12} \cdot G_{11,8} + P_{15,12} \cdot P_{11,8} \cdot G_{7,4} + P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot G_{3,0}$$

and

$$P0' = P_{15,0} = P_{15,12} \cdot P_{11,8} \cdot P_{7,4} \cdot P_{3,0}$$

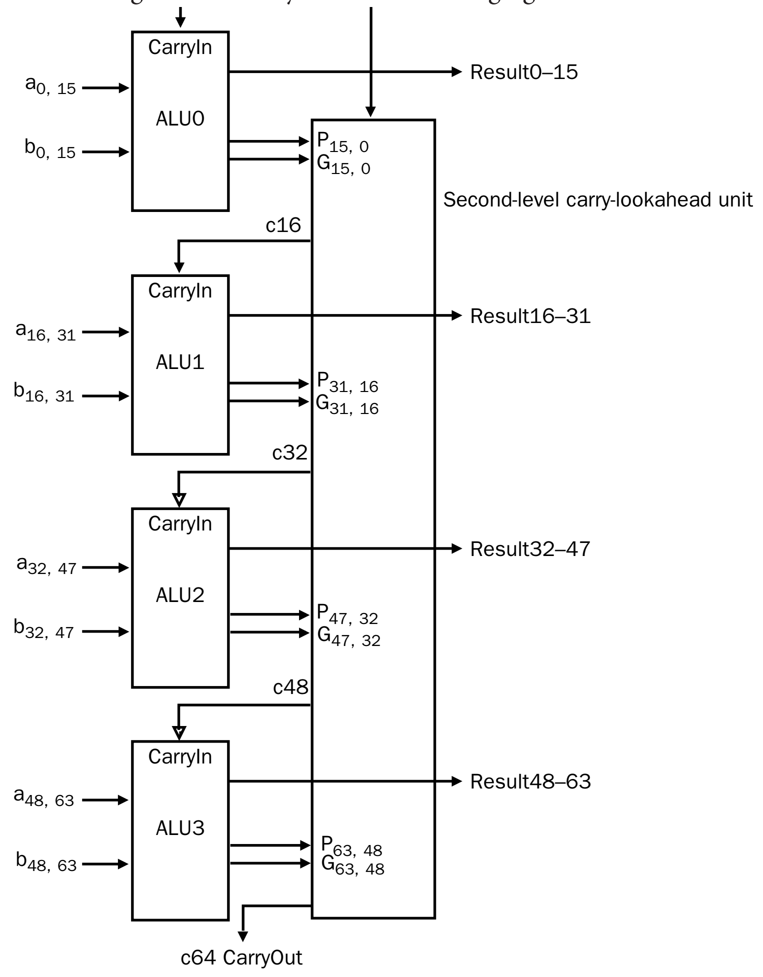
Using G0′ and P0′, we can write c16 more compactly as

$$c16 = G_{15,0} + P_{15,0} \cdot c0$$

and

$$c32 = G_{31,16} + P_{31,16} \cdot c16$$
$$c48 = G_{47,32} + P_{47,32} \cdot c32$$
$$c64 = G_{63,48} + P_{63,48} \cdot c48$$

A 64-bit adder diagram in the style of the following figure would look like this:



3.20 No solution provided.

3.21 No solution provided.

3.22 No solution provided.

3.24 No solution provided.

3.32

```
#include <stdio.h>
void main(void)
{
         double d;
         scanf("%lf",&d);
         printf("%llx\n",d);
}
```

3.33

```
import java.io.*;
public class FloatingConvert {
  public static void main(String[] args) throws IOException {
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    String line = in.readLine();
    if(line != null)
    {
      double d = Double.parseDouble(line);
      System.out.println(Long.toHexString(Double.doubleToLongBits(d)));
    }
  }
}
```

3.34

```
    # multiply by four by adding 2 to the exponent;
    # ignores exp. overflow and special exp. values

       lw  $t0, X($zero)       # $t0 = single prec. fp value
       beq $t0, $zero, Exit    # mp change if zero
       lui $t1, 0x01000000     # constant 2 in exp. field
       add $t0, $t0, $t1       # add to exponent
       sw  $t0, X(zero)        # replace

Exit:
```

3.47 No solution provided.

3.48 The guard bit is present to preserve the full precision (number of significant bits) of the significand during the operation. It can be shown that a single bit is sufficient for +, –, ×, and / to preserve the precision of a result that can be expressed in the number of bits available in the significand. If the result has more bits of precision than available in the floating-point format, then rounding is used to yield an expressible approximation (fits in the available number of bits) of the result.

The bits guard (g), round (r), and sticky (s) all participate in right shifts of the significand to make the exponent of the smaller number match that of the larger number. Right shifting moves the LSB into g, g into r, and the OR of bit(s) shifted through r into s. During left shifts, s does not participate, zeros are shifted into r as needed, r shifts into g, and g shifts into the LSB.

The following tables list the actions needed for each rounding mode.

Round Toward Negative Infinity:

g	r	s	Action
0	0	0	Exact result; significand correctly rounded
X	X	1	Inexact result, correctly rounded if Sum is positive, round by adding 1 to LSB if Sum is negative (makes Sum more negative, moving toward negative infinity)
X	1	X	Inexact result, correctly rounded if Sum is positive, round by adding 1 to LSB if Sum is negative (makes Sum more negative, moving toward negative infinity)
1	X	X	Inexact result, correctly rounded if Sum is positive, round by adding 1 to LSB if Sum is negative (makes Sum more negative, moving toward negative infinity)

Round Toward Positive Infinity:

g	r	s	Action
0	0	0	Exact result; significand correctly rounded
X	X	1	Inexact result, correctly rounded if Sum is negative, round by adding 1 to LSB if Sum is positive (makes Sum more positive, moving toward positive infinity)
X	1	X	Inexact result, correctly rounded if Sum is negative, round by adding 1 to LSB if Sum is positive (makes Sum more positive, moving toward positive infinity)
1	X	X	Inexact result, correctly rounded if Sum is negative, round by adding 1 to LSB if Sum is positive (makes Sum more positive, moving toward positive infinity)

Truncate:

g	r	s	Action
X	X	X	Significand correctly rounded

(Unbiased) Round to Nearest:

Sum_p	g	r	s	Action
X	0	0	0	Exact result; significand correctly rounded
X	0	X	1	Inexact result, but significand is correctly rounded
X	0	1	0	Inexact result, but significant is correctly rounded
0	1	0	0	Tie case with even significand; significand correct
1	1	0	0	Tie case with odd significand; round to nearest even by adding 1 to LSB
X	1	1	X	Inexact result, round to nearest by adding 1 to LSB
X	1	X	1	Inexact result, round to nearest by adding 1 to LSB

Note that unbiased round to nearest is the only mode that requires the value of the LSB. Neither unbiased round to nearest nor truncate predicate their action on the sign of the significand. Round to plus and minus infinity are the only modes that require the sign bit.

Filling in the table given in the exercise:

Rounding mode	Sum > = 0	Sum < 0
Toward Negative Infinity	blank	OR(g,r,s)
Toward Positive Infinity	OR(g,r,s)	blank
Truncate	blank	blank
(Unbiased) Round to Nearest	g (Sum_p + r + s)	g (Sum_p + r + s)

3.49 No solution provided.

3.50

1	`lw`	24%
2	`addiu`	12%
3	`sw`	9%
4	`addu`	7%
4	`or`	7%
6	`beq`	6%
7	`bne`	5%
8	`subu`	3%
8	`andi`	3%
8	`nor`	3%

3.51

1	`addu`	21%
2	`lw`	15%
2	`l.d`	15%
4	`add.d`	8%
4	`mul.d`	8%
6	`s.d`	7%
7	`lui`	5%
8	`sub.d`	3%
9	`addiu`	2%
9	`subu`	2%
9	`or`	2%
9	`sw`	2%
9	`beq`	2%

3.52

a. `lw`, `addiu`, `sw`, `addu`, `or`, `beq`, `subu`
b. 68%
c. 79%
d. 46%
e. 92%

3.53 `lw`, `addiu`, `addu`, `l.d`, `sw`

`lw`, `addiu`, `addu`, `sw` are tops with combined %.

`l.d` is very important to FP benchmarks.

3.54 1.0 cycle ops (not in Figure 3.11.1) and their percentages:

`addu`	7
`addiu`	12
`subu`	3
`and`	1
`andi`	3
`or`	7
`ori`	2
`nor`	3
`sll`	1
`lui`	2
`slt`	2
`slti`	1
`sltu`	1
`sltiu`	1
`sra`	1
`misc`	2
49%	

1.3 cycle ops (loads and stores) and their percentages:

`lw`	24
`sw`	9
`lbu`	1
`sb`	1
`lhu`	1
36%	

1.6 cycle ops (conditional branches) and their percentages:

`beq`	6
`bne`	5
`bltz`	1
`bgez`	1
13%	

1.2 cycle ops (jumps) and their percentages:

`jal`	1
`jr`	1
2%	

Avg: .49 + 1.3 * .36 + 1.6 * .13 + 1.2 * .02 = 1.19 CPI

3.55 1 cycle ops (not in Figure 3.11.1) and their percentages:

`addu`	21
`addiu`	2
`subu`	2
`or`	2
`nor`	1
`sll`	1
`lui`	5
`misc`	3
37%	

1.3 cycle ops (loads and stores) and their percentages:

`lw`	15
`sw`	2
`l.s`	15
`s.s`	7
39%	

1.6 cycle ops (conditional branches) and their percentages:

`beq`	2
`bne`	1
`blez`	1
4%	

1.0 cycle ops (FP add) and their percentages:

`add.d`	8
`sub.d`	3
11%	

3.0 FP mul 8%

5.0 Int mult 1%

Avg: .37 + 1.3 * .39 + 1.6 * .04 +.11 + 3.0 * .08 + 5.0 * .01 = 1.341 CPI