Solution to Selected Problems Set #1
COE608: Computer Organization and Architecture

Introduction, Instruction Set Architecture and Computer Arithmetic
Chapters 2 and **3**

4th Edition Problems from Chapter 2

2.6

2.6.1 a.
```
lw   $s0, 4($s7)
sub  $s0, $s0, $s1
add  $s0, $s0, $s2
```

b.
```
sll  $t0, $s1, 2
add  $t0, $s7, $t0
lw   $t0, 0($t0)
sll  $t0, $t0, 2
add  $t0, $t0, $s6
lw   $s0, 4($t0)
```

2.6.2. **a.** 3 **b.** 6

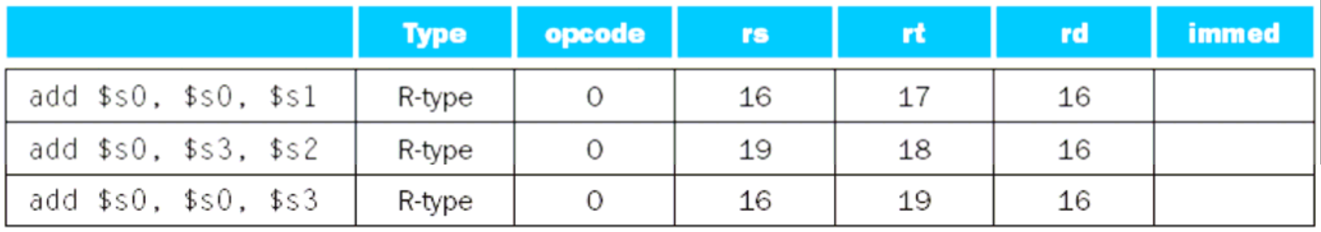
2.6.3. **a.** 4 **b.** 5

2.6.4. **a.** f = 2i + h; **b.** f = A[g – 3];

2.6.5. **a.** $s0 = 110 **b.** $s0 = 300

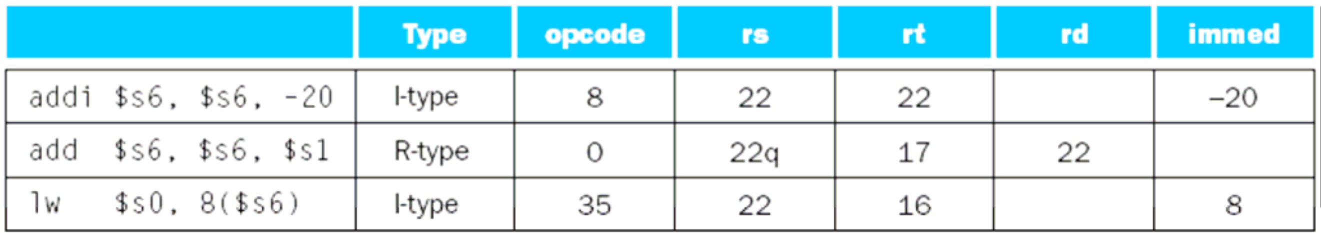
2.6.6.

a.



	Type	opcode	rs	rt	rd	immed
`add $s0, $s0, $s1`	R-type	0	16	17	16	
`add $s0, $s3, $s2`	R-type	0	19	18	16	
`add $s0, $s0, $s3`	R-type	0	16	19	16	

b.



	Type	opcode	rs	rt	rd	immed
`addi $s6, $s6, -20`	I-type	8	22	22		–20
`add  $s6, $s6, $s1`	R-type	0	22q	17	22	
`lw   $s0, 8($s6)`	I-type	35	22	16		8

2.12

2.12.1.

	Type	opcode	rs	rt	rd	shamt	funct	
a.	R-type	6	3	3	3	5	6	total bits = 26
b.	R-type	6	5	5	5	5	6	total bits = 32

2.12.2

	Type	opcode	rs	rt	immed	
a.	I-type	6	3	3	16	total bits = 28
b.	I-type	6	5	5	10	total bits = 26

2.12.3

a. less registers → less bits per instruction → could reduce code size
less registers → more register spills → more instructions

b. smaller constants → more lui instructions → could increase code size
smaller constants → smaller opcodes → smaller code size

2.12.4 **a.** 17367056 **b.** 2366177298

2.12.5 **a.** add $t0, $t1, $0 **b.** lw $t1, 12($t0)

2.12.6 **a.** R-type, op=0×0, rt=0×9 **b.** I-type, op=0×23, rt=0×8

Computer Arithmetic

Questions from 3rd Edition

3.12. Suppose that all of the conditional branch instructions except beq and bne were removed from the MIPS instruction set along with slt and all of its variants (slti, sltu, sltui). Show how to perform.

```
slt  $t0, $s0, $s1
```

using the modified instruction set in which slt is not available .

Solution:
To detect whether $s0 < $s1, it's tempting to subtract them and look at the sign of the result. This idea is problematic, because if the subtraction results in an overflow, an exception would occur! To overcome this, there are two possible methods:
You can subtract them as unsigned numbers (which never produces an exception) and then check to see whether overflow would have occurred. This method is acceptable, but it is lengthy and does more work than necessary.
An alternative would be to check signs. Overflow can occur if $s0 and (–$s1) share the same sign; that is, if $s0 and $s1 differ in sign. But in that case, we don't need to subtract them since the negative one is obviously the smaller! The solution in pseudo-code would be
if ($s0<0) and ($s1>0) then $t0 : =1

```
    else if ($s0>0) and ($s1<0) then $t0 : = 0
            else $t1 : = $s0 – $s1
if ($t1<0) then $t0 : = 1 else $t0 : = 0
```

3.45. The internal representation of floating point numbers in IA-32 is 80 bits wide. This contains a 16-bit exponent. However, it also advertises a 64-bit significand. How is this possible?

Solution:

It implied 1 is counted as a significant bits. So, 1 sign bit, 16 exponent bits, and 63 fraction bits.

Additional Question:

2. Subtract $(13)_{10}$ from $(39)_{10}$ using 2's-complement arithmetic.

Solution:
Requires a 6-bit plus 1 sign bit (39 needs 6-bit to represent) 2's complement system for proper addition or subtraction

```
        2's complement of  39   = 0     1 0 0 1 1 1
        2's Complement of 13    = 0     0 0 1 1 0 1
        2's complement of –13   = 1     1 1 0 0 1 1
Add 2's complement of  39 and 2's complement of -13

            +39                 0       1 0 0 1 1 1
            -13                 1       1 1 0 0 1 1
            ______________________________________
            + 26        ~~1~~       0       0 1 1 0 1 0
          carry discard it
          Result is a +ve number (0)   011010
```