Questions

1. The `add` function has opcode 000000. At most how many different MIPS operations are there with this opcode?

2. Write each of the following *pseudo*instructions using MIPS instructions `slt, beq, bne`.

```
ble  $s1, $s2,  Exit
bgt  $s1, $s2,  Exit
bge  $s1, $s2,  Exit
```

3. What is the value of `$s0` after each instruction below?

```
lui   $s0, 0x322b
srl   $s0,  $s0,  4
srl   $s0,  $s0, 24
sll   $s0,  $s0,  1
sll   $s0,  $s0,  1
sll   $s0,  $s0,  1
```

4. MIPS has immediate and unsigned versions of the `add` instruction. Why doesn't it have immediate and signed versions of the `sub` instruction, namely `subi` and `subiu` ?

5. Convert each of the MIPS pseudoinstructions into real MIPS instructions:

```
a) not   $s0, $s1      #  Hint use one of:  'and', 'or', 'nor'

b) rol   $s0, $s1, 7   #  rotate left i.e. wraparound the bits
                       #  The 7 most significant bits of $s1
                       #  become the 7 least significant bits of $s0
                       #  Hint:  use 'sll' and 'srl' (shift left and right)

c) li    $s0, 20       #  load immediate

d) beqz  $s0, label    #  branch if equal to zero

e) neg   $s0, $s1      #  negate $s1
```

6. When two integers are multiplied in MIPS, the result is put in the `Hi` and `Lo` registers which are accessed using `mfhi` and `mflo` instructions. For example,

```
mult    $s0, $s1
mfhi    $t0
mflo    $t1
```

(a) Why didn't the MIPS designers use three registers for the `mult` instruction, as they did for (say) `add` ?

(b) Suppose that three integers i, j, k have their values stored in three MIPS registers. How many words are needed to store the product $i * j * k$?

Solutions

1. The `funct` field of the R-format instruction has 6 bits and so there are $2^6 = 64$ possible funct codes for each opcode.

2. ```
   ble  $s1, $s2,  Exit                    slt  $t0, $s2, $s1,
   beq  $t0, $zero,  Exit

   bgt  $s1, $s2,  Exit                    slt  $t0, $s2, $s1
   bne  $t0, $zero,  Exit

   bge  $s1, $s2,  Exit                    slt  $t0, $s1, $s2
   beq  $t0, $zero, Exit
   ```

3. ```
   lui      $s0, 0x322b              # $s0 = 0x322b0000
   srl      $s0, $s0, 4              # $s0 = 0x0322b000
   srl      $s0, $s0, 24             # $s0 = 0x00000003
   sll      $s0, $s0, 1              # $s0 = 0x00000006
   sll      $s0, $s0, 1              # $s0 = 0x0000000c
   sll      $s0, $s0, 1              # $s0 = 0x00000018
   ```

4. The `addi` and `addiu` require you to write an immediate argument explicitly. If you want to subtract instead of add, you don't need a special `subi` or `subiu` instruction: you can just add the negative of the number you wanted to subtract. There is no need to waste opcodes on these immediate subtraction operations.

5. ```
   a) nor  $s0, $s1, $0   #  not $s0, $s1

   b) srl  $t0, $s1, 25   #  rol $s0, $s1, 7
      sll  $s0, $s1, 7
      or   $s0, $s0, $t0

   c) addi $s0, $0, 20    #  li    $s0, 20

   d) beq  $s0, $0, label #  beqz  $s0, label

   e) sub  $s0, $0, $s1   #  negate $s1
   ```

6. (a) `mult $s1, $s2, $s3` wouldn't work, since `$s1` in only 32 bits whereas the result needs a 64 bit register.

   (b) The product i*j*k would require 3 words. Why? Each of the variables is 32 bits (unsigned int), i.e. each value is at most $2^{32}-1$. Thus, the product $j*k$ is less than 2^{64}. Multiplying by i gives a result that is less than 2^{64+32}, that is, $(2^{32})^3$. So, $i * j * k$ requires at most $3 * 32$ bits, or 3 words.
   ```