(6 pts) Exercise 2-8

(See number discussion in Section 2.5)

• What binary number does this hexadecimal number represent:
  \[ \text{7fff fffa}_{\text{hex}} \]?

• What decimal number does it represent?

(10 pts) Exercise 2-9

Show the hexadecimal representation of this MIPS instruction:

\[ \text{add } \$t0 \ , \ \$t1 \ , \ \$zero \]
(10 pts) Exercise 2-10

What MIPS instruction is represented by this binary entry:

1000 1101 0000 1001 0000 0000 0100 0000

(5 pts) Exercise 2-11

What is the MIPS assembly code for the following:

if (g != j)  h = g - h;
else         h = g + h;

Variables f to j are assigned to registers $s0 to $s4
(5 pts) Exercise 2-12

- What is the MIPS assembly code for the following:

```plaintext
if (j == h)  g = i + j;
```

Variables f to j are assigned to registers $s0 to $s4

<table>
<thead>
<tr>
<th>f</th>
<th>$s0</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>$s1</td>
</tr>
<tr>
<td>h</td>
<td>$s2</td>
</tr>
<tr>
<td>i</td>
<td>$s3</td>
</tr>
<tr>
<td>j</td>
<td>$s4</td>
</tr>
</tbody>
</table>

(5 pts) Exercise 2-13

- What is the MIPS assembly code for the following:

```plaintext
if ( (j == h) && (f != i) )  g = i + j;
```

Variables f to j are assigned to registers $s0 to $s4

<table>
<thead>
<tr>
<th>f</th>
<th>$s0</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>$s1</td>
</tr>
<tr>
<td>h</td>
<td>$s2</td>
</tr>
<tr>
<td>i</td>
<td>$s3</td>
</tr>
<tr>
<td>j</td>
<td>$s4</td>
</tr>
</tbody>
</table>
(10 pts) Exercise 2-14

- What is the MIPS assembly code for the following:

```mips
if ((g != h) && (f == i)) || (g == i)
g = i + j;
```

Variables f to j are assigned to registers $s0$ to $s4$
Exercise 2-18: Pseudo-instructions

- Below you will see several problems of the form:
  - `li $t1, small`  # $t1 = small
  - `beq $t1, small, L`  # if ($t1 == small) go to L
  - `beq $t2, big, L`  # if ($t2 == big) go to L
  - `li $t2, big`  # $t2 = big
  - `bge $t5, $t3, L`  # if ($t5 >= $t3) go to L
  - `lw $t5, big($t2)`  # $t5 = Memory[$t2+big]

- This is an example pseudo-instruction, with its meaning given as a comment. The instruction should load 'small' into $t1, where 'small' means a constant that fits within 16 bits. The constant is also called an 'immediate' – 'li' stands for 'load immediate'.

- Your job is to write the real MIPS instruction (or sequence of instructions) that the compiler would produce for each given pseudo-instruction. For instance, the solution for the above pseudo-instruction is:
  - `addi $t1, $zero, small`

- A 'big' constant needs 32 bits to be represented. You will need some notation to talk about the upper (most significant) 16 bits and the lower (least significant) bits of this number. Use UPPER(big) to refer to the most significant bits and LOWER(big) to refer to the other bits.

- To make your answers simpler, you may make use of the 'li' pseudo-instruction where helpful (except when defining li itself). Do this – it will make your job easier!!
(5 pts) Exercise 2-21

- a.) What is the MIPS assembly code for the following:
  do {
    g = g + j;
  } while (g < h);

Variables f to j are assigned to registers $s0 to $s4
Use $v0, $v1 as temporaries if needed

b.) Did your solution use any pseudo-instructions?

(5 pts) Exercise 2-22

- a.) What is the MIPS assembly code for the following:
  do {
    g = g + j;
  } while (g < 100);

Variables f to j are assigned to registers $s0 to $s4
Use $v0, $v1 as temporaries if needed

b.) Did your solution use any pseudo-instructions?
(5 pts) Exercise 2-23

• a.) What is the MIPS assembly code for the following:

```plaintext
while (g < i) {
    g = g + j;
}
```

Variables f to j are assigned to registers $s0 to $s4
Use $v0, $v1 as temporaries if needed
b.) Did your solution use any pseudo-instructions?

(10 pts) Exercise 2-24

• a.) What is the MIPS assembly code for the following:

```plaintext
while (g > i) {
    g = g + 3;
}
```

Variables f to j are assigned to registers $s0 to $s4
Use $v0, $v1 as temporaries if needed
b.) Did your solution use any pseudo-instructions?
The MIPS translation of the C segment:
while (save[i] == k) i = i + 1;
is given on page 107-108 as follows:

Loop:  sll $t1, $s3, 2 # temp reg t1 = i *4
       add $t1, $t1, $s6 # t1 = address of save[i]
       lw $t0, 0($t1) # temp reg t0 = save[i]
       bne $t0, $s5, Exit # goto Exit if save[i] != k
       add $s3, $s3, 1 # loop body: i = i + 1
       j Loop # repeat the loop

This code uses both a conditional branch and an unconditional jump
each time through the loop. Only poor compilers would produce
code with this loop overhead. Rewrite the assembly code so that it
uses at most one branch or jump each time through the loop.