1. (5 pts) How many different values can be represented in 8 hex digits?
   \(16^4 = 65536\)

2. (5 pts) How many different values can be represented in 32 binary digits (bits)?
   \(2^{32} = 4,294,967,296\)

3. (5 pts) Convert the following binary numbers to hexadecimal numbers:
   - 1011 1110 1110 1111 \(\rightarrow\) BEEF
   - 1000 1101 1001 1111 \(\rightarrow\) 8D9D
   - 1111 0001 0000 0001 \(\rightarrow\) F101
   - 1000 0111 0111 0001 \(\rightarrow\) 8771
   - 0010 1001 0101 1000 \(\rightarrow\) 2958

4. (5 pts) Convert the following hexadecimal numbers to binary numbers. Please separate each set of 4 binary digits with a space.
   - 4B4E \(\rightarrow\) 0100 1011 0100 1110
   - EB15 \(\rightarrow\) 1110 1011 0001 0101
   - 9A5D \(\rightarrow\) 1001 1010 0101 1101
   - 7E18 \(\rightarrow\) 0111 1110 0001 1000
   - 97CF \(\rightarrow\) 1001 0111 1100 1111

5. (10 pts) Write MIPS code that subtracts the constant 87 from register $t8 and puts the result in register $a3. You may only use valid MIPS instructions.
   ```
   addi $a3, $t8, -87
   ```

6. (10 pts) Write MIPS code for the following computation. Assume that variable A is in register $t0, B is in register $t1, C is in register $t2, D is in register $t3, E is in register $t4 and F is in register $t5. Your code should not destroy the values of any registers unnecessarily (e.g., you are expected to overwrite $t5, but, do not modify $t0, $t1, etc.).
   \[ F = (C - ((A + D) - B)) - E \]
   ```
   add $t5, $t0, $t3  # t5 = A+D
   sub $t5, $t5, $t1  # t5 = t5 - B = (A+D) - B
   sub $t5, $t2, $t5  # t5 = C - t5 = C -((A+D)-B)
   sub $t5, $t5, $t4  # t5 = t5 - E => F = (C -((A+D)-B)) - E
   ```
7. (10 pts) Explain briefly what an immediate operand is. Why are immediate operands in MIPS always the same size (16 bits) in I-format instructions?

Immediate operands are constant numbers embedded within an instruction, for use by that instruction. In MIPS, immediates are 16-bits. The immediate can be treated as a signed value (e.g., with addi) or as an unsigned value (e.g., as with addiu). We generally would like the immediate to be as large as possible (allowing us to use larger constants), but immediate are always 16-bits because the other fields (opcode, rs, rt) of the instruction take up the remaining 16-bits of the instruction.

8. (5 pts) Write out the MIPS instructions that implements the pseudo-instruction “branch-if-greater-or-equal” (e.g., bge $t0, $t1, LAB).

```
slt $at, $t0, $t1  # is $t0 < $t1?
beq $at, $zero, LAB  # branch if $at == 0
```

9. (5 pts) Write out the MIPS instructions that implements the pseudo-instruction branch-if-greater-than (e.g., bgt $t0, $t1, LAB).

```
slt $at, $t1, $t0
bne $at, $zero, LAB
```

10. (10 pts) Consider the following code, which declares an array of words:

```
.data
wages: .word 100 200 300 400 500 600 700 800 900 1000
```

Write the MIPS assembly code to take the 2\textsuperscript{nd} and 8\textsuperscript{th} elements from the \texttt{wages} array, and place the sum of those two elements into the 5\textsuperscript{th} element. Do not use any pseudo-instructions, except for the “la” pseudo-instruction. This can be done in five MIPS instructions.

```
.data
wages: .word 100 200 300 400 500 600 700 800 900 1000

.text
la $t0, wages
lw $t1, 8($t0)  # load 2\textsuperscript{nd} value
lw $t2, 32($t0) # load 8\textsuperscript{th} value
add $t3, $t1, $t2 # calculate sum
sw $t3, 20($t0) # store into 5\textsuperscript{th} value
```

13. (5 pts) Set bits 26, 27, 28, 29, and 30 to 1 in register $v0. $v0's other bits should not change. This can be done in two MIPS instructions. Do not use any pseudo-instructions or load any data from memory. You may use any registers that you wish.

```
lui $t0, 0x7C00
or $v0, $v0, $t0
```
14. (10 pts) In register $v0, set bits 21 and 20 to 0 and 1 respectively. $v0's other bits should not change. Additionally, set bits 14 and 15 to 1 and 0 respectively. This can be done in six MIPS instructions. Do not use any pseudo-instructions or load any data from memory. You may use any registers that you wish.

```
lui $t0, 0x0010
ori $t0, $t0, 0x4000
or $v0, $v0, $t0  #Turn on bits 14 and 20

lui $t0, 0xFFDF
ori $t0, $t0, 0x7FFF
and $v0, $v0, $t0  #Turn off bits 21 and 15
```