Assignment
Here is a simple MIPS program given to you in bytes
01 00 08 20 00 00 09 20 05 00 02 20 0c 00 00 00 03 00 02 11 20 48 09 01 01 00 08 21 04 00 00 08 1a 00 22 01 12 20 00 00 0c 00 00 00

1. Which byte ordering(s) does the MIPS assembly language use? What is bi-endianness and how does it work?

MIPS processors can operate with either big-endian or little-endian byte order. Once again, big endian format means that data is stored big end first. In the little endian format, data is stored little end first. Wikipedia says:

> "One of the special features of the MIPS architecture is that all processors except the R8000 can be configured to run either in big or in little endian mode. Byte order means the way the processor stores multibyte numbers in memory. Big endian machines store the byte with the highest value digits at the lowest address while little endian machines store it at the highest address. Think of it as writing multi-digit numbers from left to right or vice versa."

2. Use the disassembler and tell us whether the byte-code given is in big or little-endian format. HINT: the correctly disassembled code should contain a 'div' instruction.

Little. I think it only makes sense in little endian mode.

3. Convert the byte code to the opposite endianness. That is, if the code is in little-endian format convert it to big-endian and vice versa.

20 08 00 01 20 09 00 00 20 02 00 05 00 00 00 0c 11 02 00 03 01 09 48 20 21 08 00 01 08 00 00 04 01 22 00 1a 00 00 20 12 00 00 00 0c

4. Once you know the endianness, convert the disassembled MIPS code into C and provide a snapshot of your code in your submission.

mips converted into c:

```
int a = 1, b = 0;        # b is the sum
scanf("%d", &read_int);

while (a1 != read_int)   # (if a != the read_int value)
{
    b = a + b
    a++
}

quotient = b / a
```

Notes on conversation:
11020003 hex converts to:
000100 01000 00010 0000000000000011

01094820 in hex converts to:
000000 01000 01001 01001 00000 100000

5. In a few brief sentences, describe what the program does.

The program sums the numbers from 1 to 4 and store the sum in b. If we are dividing two 32-bit number then the result is 64-bits so we need the first 32-bit result that is in lo (low = first 32 bits). The quotient is in a0.