Question 2 (30 points): The 16-bit *half precision* floating point representation has the following specification:

15	14 — 10	9 — 0
S	*exponent*	*fraction*

$$N = \begin{cases} (-1)^S \times 0.\mathit{fraction} \times 2^{-14} & \text{if } \mathit{exponent} = 0 \\ (-1)^S \times 1.\mathit{fraction} \times 2^{\mathit{exponent}-15} & \text{if } 0 < \mathit{exponent} < 31 \\ (-1)^S \times \infty & \text{if } \mathit{exponent} = 31 \text{ and } \mathit{fraction} = 0 \\ NaN & \text{if } \mathit{exponent} = 31 \text{ and } \mathit{fraction} \neq 0 \end{cases}$$

a) **(4 points)** What is the binary representation of -37.375 in the half-precision floating-point representation? $-37.375_{10} = -100101.011_2 \Rightarrow 1.00101011 \times 2^5$

sign = 1 exponent = 5+15 = 20 ⇒ 10100

fraction = 00101011

$\boxed{1\ 10100\ 00101011}$

b) **(6 points)** A = 0x6404 and B = 0x4790 are two half-precision floating-point numbers. What is the true value of $A + B$ expressed in decimal notation? That is, if we have infinite precision to do the addition and store the result, what is $A + B$?

A = 0|110 01|00 0000 0100 ⇒ $1.0000000001 \times 2^{25-15} \Rightarrow 1028_{10}$

B = 0|100 01|11 1001 0000 ⇒ $1.111001 \times 2^{17-15} \Rightarrow 7.5625_{10}$

⇒ $\boxed{1035.5625}$

c) **(5 points)** What is the decimal value of $A + B$ computed on a machine with one guard bit, one round bit and one sticky bit?

```
  1.0000000001
+ 0.0000000001111001
  1.0000000010111001
```

fraction, guard, round, sticky = 1

⇒ $1.0000000100 \times 2^{10}$

$= \boxed{1036_{10}}$

d) (**15 points**) MIPS has division instructions for single and double precision floating point values, but not for half-precision. Write a MIPS procedure to implement division of a half-precision floating point by an integer multiple of 2. The input to your procedure is the address of a half-precision floating point value (in `$a0`), and the unsigned integer representation of a power of 2 (in `$a1`). Your procedure should perform the division and update the value in memory at the address in `$a0`.

Your code should follow calling conventions and should not use any pseudoinstructions. You may assume that the result of the division can be represented as a half-precision floating point value.

Note that if \$a1 contains the unsigned int representation of 2^k, then it will contain a 1 followed by k 0's. We need to count the 0's, then subtract that from the half's exponent.

```
halfdiv:  addi $t0, $zero, -1
loop:     beq $a1, $zero, next      # repeats until we
          srl $a1, $a1, 1           # have made k+1
          addi $t0, $t0, 1          # shifts
          j loop
next:     lhu $t1, 0($a0)           # load the half
          andi $t2, $t1, 0x7C00     # mask out exponent
          srl $t2, $t2, 10          # shift, add k, and
          add $t2, $t2, $t0         # shift back
          sll $t2, $t2, 10
          andi $t1, $t1, 0x83FF     # clear exponent
          or $t1, $t1, $t2          # update exponent
          shu $t1, 0($a0)           # update memory
          jr $ra
```