Pregunta 1

Emulación de una multiplicación por dos de un número real expresado en formato IEEE 754 de simple precisión.

a) Se debe sumar uno al exponente sin realizar ninguna comprobación adicional. El convenio de paso de parámetros se ha de cumplir, de modo que la función llamada **mults_2** recibirá como parámetro por copia a través del registro $f12 el número real, y la función retornará el resultado de la multiplicación a través del registro $f0.

$f0 = mult_s(float $f12)

b) Añada la comprobación de los números especiales, cero, infinito y Nan. Además si el resultado de la operación excede el formato que se retorne infinito.

Pregunta 2

Programe una subrutina que indique si un número real es Nan.

- El número se pasará por copia a través del registro $f12, y el resultado se retornará a través del registro $v0, será 0 si es falso y 1 si es cierto.

$v0 = EsNan(float $f12)

- Programe también el programa principal que realice correctamente la llamada a dicha función.

Pregunta 3

- Subrutina que calcule la media aritmética de un vector de elementos de tipo float. En $a0 se pasará la dirección del vector y en $a1 la dimensión. El resultado se retornará en $f0.

$f0 = media(float *a0, int $a1)

- Implemente también el programa principal realizando la llamada a la función creada.

Pregunta 4

Implemente una función que recorra un vector de float y calcule su inversa, siempre que sea el elemento distinto de cero. El procedimiento sería en alto nivel como se muestra a continuación:

```
void inversa ( float A, int max) {
int i;
for (i=0; i<max; i++) {
     if A[i] <> 0.0 {
        A[i] = 1/A[i];
     }
}
}
```

El paso de parámetros será según convenio: $a0 la dirección del vector y $a1 el número de elementos del vector.

Solución 1

Apartado a)

```
#    Operación simulada con instrucciones de enteros.

#    $f0 =  mult_s(float $f12)

#     Retorna en $f0 = $f12 * 2

mults_2:mfc1 $t0, $f12          # copia $f12 en $t0
        li $t1, 0x00800000      # máscara: un 1 en la 23a posición
        add $t0, $t0, $t1       # suma un 1 al exponente
        mtc1 $t0, $f0           # copia el resultado en $f0
        jr $ra
```

Apartado b)

```
mults_2:mfc1 $t0, $f12          # copia $f12 en $t0
        li $t1, 0x7FFFFFFF      # máscara para  E y M
        and $t1, $t0, $t1       # extrae E y M en $t1
        beq $t1, $zero, eixir   # si E=M=0 es +0.0 o  -0.0

        li $t1, 0x7F800000      # máscara para  E
        and $t2, $t0, $t1       # extrae E en $t2
        beq $t1, $t2, eixir     # si exp==255 es NaN, +Infty o -Infty

        li $t1, 0x00800000      # máscara: un 1 en la 23a posición
        add $t0, $t0, $t1       # suma 1 un al exponente
        li $t1, 0x7F800000      # máscara para E
        and $t2, $t0, $t1       # extrae E en $t2
        bne $t2, $t1, eixir     # si exp<>255 es un número "normal"
                                # si exp==255 ha desbordado->infinito
        li $t1, 0xFF800000      # máscara  M a zero
        and $t0, $t0, $t1       # M = zero
```

```
eixir:  mtc1 $t0, $f0
        jr $ra
```

Solución 2

```
   Apartado a) Función esNan

#    $v0 =  EsNan(float $f12)

#     Retorna en $v0 = 0 si  $f12 ≠ Nan y $v0 = 1 si $f12 = Nan

#      Nan lleva Exponente = 255 y Mantisa ≠ 0


es_NaN: mfc1 $t0, $f12          # copia $f12 en $t0
        li $t1, 0x7F800000      # màscara per a l'exponent
        and $t2, $t0, $t1       # exponent en $t2
        bne $t1, $t2, noes      # si exp<>255 no ésNaN
        li $t1, 0x007FFFFF      # màscara per a la mantissa
        and $t2, $t0, $t1       # mantissa en $t2
        beq $t2, $zero, noes    # si mantissa==0 no ésNaN
sies:   li $v0, 1
        j fin
noes:   li $v0, 0
fin:    jr $ra


Apartado b) Llamada desde el programa principal

            .data

numero:  .float  12.0


            .text

__start:    la $t0, numero

            lwc1 $f12, 0($t0)

            jal es_Nan

            .....
```

Solución 3

```
        .data 0x10000000
A:      .float 39.8, 36.2, 41.7, 40.5

        .text
        la $a0, A
        li $a1, 4
        jal media
        ......


#     $f0 =  media(float *a0, int $a1)
#


media:  or $t0, $zero, $a0      # copia de $a0 en $t0
        or $t1, $zero, $a1      # copia de $a1 en $t1
        mtc1 $zero, $f0         # posa 0.0 en $f0

bucle:  beqz $t1, fin
        lwc1 $f4, 0($t0)        # $f4 ← A[i]
        add.s $f0, $f0, $f4     # $f0 = $f0 + A[i]
        addiu $t0, $t0, 4       # dirección elemento siguiente
        addi $t1, $t1, -1       # decremento del contador
        b bucle                 #  repite el bucle

  fin:  mtc1 $a1, $f4           #   $f4 ← $a1
        cvt.s.w $f4, $f4        # convierte de entero a float
        div.s $f0, $f0, $f4     # divide y calcula la media
        jr $ra
```

Solución 4

Si solo tuviéramos que recorrer el vector tendríamos las siguientes instrucciones:

```
#
#     void Inversa (float *$a0, int $a1)
#
Inversa:   move $t0, $a0
           move $t1, $a1
    for:   beqz $t1, fin
           lwc1  $f4, 0($t0)  # $f4 ← A[i]
           ....
           swc1  $f4, 0($t0)  # $f4 → A[i]
           addiu $t0, $t0, 4
           addi  $t1, $t1, -1
           b for
fin:       jr $ra
```

Ahora le añadimos las instrucciones que comparan A[i] con 0.0 y luego realizan las operaciones de inversión.

```
#
#     void Inversa (float *$a0, int $a1)
#
Inversa:   move $t0, $a0
           move $t1, $a1
           li.s  $f0, 0.0      #  mtc1 $zero, $f0 también
           li.s  $f6, 1.0      #  $f6 = 1.0
    for:   beqz $t1, fin
           lwc1  $f4, 0($t0)   # $f4 ← A[i]
           c.eq.s $f4,$f0      #  ¿ $f4 = 0.0?
           bc1t seguir         #  Salta si cierto
           div.s $f4, $f6,$f4  # $f4 = 1.0/$f4
           swc1  $f4, 0($t0)   # $f4 → A[i]
   seguir: addiu $t0, $t0, 4
           addi  $t1, $t1, -1
           b for
fin:       jr $ra
```