V Fixed Point

Contreras R Orlando[1,2*] and Garcia B Iker[1,2*†]

[3]Departamento de Sistemas Electrónicos, UAA, Av. Universidad 940, Aguascalientes, 20131, Aguascalientes, México.

*Corresponding author(s). E-mail(s): {al348390,al307630}@edu.uaa.mx;

†Ambos autores contribuyeron de forma equitativa a este trabajo.

Abstract

Esta práctica tiene como objetivo poder convertir números en base 10 (Decimal) en un formato donde se toman 12 bits, es decir nuestro valor mínimo sería $\mathbf{2^{12}}$, ademas de incluir 4 bits para enteros.

Keywords: ASM, ARM, Shift, Registros, Punto Fijo, Fraccionario, Convertidor

1 Introducción

En esta práctica se busca implementar un sistema capaz de convertir números en base 10 a formato Q4.12, utilizando operaciones aritméticas de punto fijo. El objetivo principal es explorar cómo representar números fraccionarios con precisión mediante el uso de una notación de punto fijo, optimizando el uso de recursos y evitando operaciones de coma flotante, lo cual es especialmente útil en sistemas embebidos con capacidades limitadas de procesamiento.

2 Objetivos

• Implementar un sistema de conversión de números decimales (base 10) a formato de punto fijo Q4.12.

• Realizar operaciones aritméticas necesarias para representar con precisión números fraccionarios en formato Q4.12 sin utilizar coma flotante.

• Validar y mostrar los resultados de la conversión, facilitando su uso en aplicaciones que requieren cálculos precisos en sistemas con recursos limitados.

3 Pseudocódigo y Explicacion del codigo

A continuación, se presenta el pseudocódigo que representa la lógica del programa en ensamblador:

Algorithm 1 Conversión de decimal a formato Q4.12

Función:

Cargar `Accuracy (12)` en `R9` Cargar `Integer (27)` en `R0` Cargar `Float (13)` en `R1` Cargar dirección de salida `DataOUT (0x20001000)` en `R7` Llamar a la función `xd` Guardar valor máximo (`R2`) en `R8`

`// Inicio del ciclo de conversión Q4.12`

1 **while** *`R9 > 0`* ***y*** *`R1  0`* **do**

2 Decrementar `R9` Multiplicar `R1` por 2 (`LSL`) **if** *`R1  R8`* **then**

3 Establecer el bit menos significativo de `R6` en 1 Multiplicar `R6` por 2 Restar `R2` de `R1`

4 **else**

5 Multiplicar `R6` por 2

6 Desplazar `R0` a la izquierda por `Accuracy` bits Hacer `OR` entre `R0` y `R6`

`// Almacenar en memoria en formato de 3 bytes`

7 Guardar `R0 >> 16` en `[DataOUT]` Guardar `(R0 >> 8)` en `[DataOUT + 1]` Guardar `(R0)` en `[DataOUT + 2]`

Algorithm 2 Cálculo del valor máximo base 10 en punto fijo

Función:

Cargar el valor `1` en `R2` **while** *`R2  R1`* **do**

Multiplicar `R2` por 2 Multiplicar `R2` por 10 usando: `R2 = R2 + (R2 << 2)`

Retornar con `BX LR`

GitHub Repository

3.1 Direcciones y Constantes

- **DataOUT (0x20001000)**: Dirección de memoria donde se almacena el resultado final de la conversión en formato Q4.12 (3 bytes).
- **Integer (27)**: Parte entera del número decimal que se desea convertir.
- **Float (13)**: Parte fraccional del número decimal.
- **Accuracy (12)**: Precisión en bits que se usará para la parte fraccional del resultado (formato Q4.12).

3.2 Inicialización del Proceso

Al comenzar la ejecución de la rutina `__main`, se cargan en los registros los valores necesarios para realizar la conversión. Además, se invoca una subrutina que permite escalar el número decimal en base 10 a una magnitud que pueda ser utilizada para realizar la conversión a binario.

- `R0`: Contiene la parte entera del número decimal.
- `R1`: Contiene la parte fraccional del número decimal.
- `R2`: Se usará para guardar el valor escalado (por potencias de 10) como referencia de comparación.

- `R6`: Se utiliza para acumular los bits fraccionales resultantes de la conversión a binario.
- Se llama a la subrutina `xd`, que multiplica iterativamente el número por 10 hasta superar el valor de la parte fraccional.

3.3 Lógica de Conversión Q4.12

El objetivo del algoritmo es representar el número decimal en formato punto fijo Q4.12, el cual utiliza 4 bits para la parte entera y 12 bits para la parte fraccional. El procedimiento implementado realiza esta conversión utilizando desplazamientos y comparaciones sucesivas:

- Se desplaza a la izquierda la parte fraccional y se compara con un valor máximo (guardado en `R2`) para decidir si se añade un bit 1 o 0 en el resultado final.
- Cada iteración representa un bit de la parte fraccionaria (hasta completar los 12 bits definidos por `Accuracy`).
- El valor resultante se construye combinando:
  - La parte entera desplazada 12 bits a la izquierda.
  - La parte fraccionaria acumulada en `R6`.

El resultado final (24 bits) se almacena en memoria (dirección `DataOUT`) en tres bytes consecutivos.

3.4 Subrutinas

- `xd`: Multiplica iterativamente el número base por 10 utilizando instrucciones `LSL` y `ADD`, con el objetivo de obtener un valor de referencia que permita realizar comparaciones sucesivas durante la conversión binaria.

Este es el código original en ensamblador que corresponde al pseudocódigo anterior:

```
; -------------- Iker | Das -----------
; ------------ Fixed Point ------------
; ------------- 16/05/2025 ------------
; ------------- Variables -------------
; ---------------- Main ---------------

; ---- Registers Used ----
;R4 iteration
;R5 Float
; r6 log_2(x)=31-clz(x)
; r7 log_10(2)
;R8 10
;r9

;Q 4.12
Integer  EQU 27
Float    EQU 13
Accuracy EQU 12
DataOUT       EQU 0x20001000

        AREA data, DATA, READWRITE
        AREA juve3dstudio,CODE,READONLY
        ENTRY
        EXPORT __main

__main
;    BL      confRCC
    LDR     R9,=Accuracy
    LDR     r0,=Integer
    LDR     r1,=Float
    LDR     r7,=DataOUT
    BL      xd
    mov     r8,r2 ;max val

ciclo
    ; r9 iterations, r8 log(max val)
    ; r2 log(x) r6 fractional x r5 val
    SUBS    r9,#1
    BEQ     print
        adds         r1,#0
        beq         print

    lsl     r1,#1
    cmp     r1,r8
    bhs     poner1
        lsl     r6,#1
        b ciclo
poner1
    orr r6,#1
        lsl r6,#1
    sub r1,r2

    b ciclo

print
        lsl r0,#Accuracy
        orr r0,r6,r0
    lsr r1, r0, #16     ; get MSB
```

```
    strb r1, [r7]
    lsr r1, r0, #8
    strb r1, [r7, #1]
    strb r0, [r7, #2]

        B .

xd
    ldr     r2,=1
otrabes
```

```
    cmp     r2, r1 ; r2-r1
    bls     por10
        bx  lr

por10
    ;r2>r1
    lsl     r2, r2, #1      ; r2 = 2x
    add     r2, r2, r2, lsl #2 ; r2 = 2x + (2x << 2) = 2x +
    b       otrabes

    end
```

4 Conclusiones

El mayor problema que tuvimos fue al momento de convertir la parte fraccionaria del número decimal al formato Q4.12. Al principio, no lográbamos que los bits fraccionales se generaran correctamente, y eso nos llevó a revisar varias veces la lógica de desplazamientos y comparaciones.

Esta práctica nos enseñó que, al trabajar con formatos como Q4.12, es fundamental comprender no solo la aritmética binaria, sino también cómo se representan los números en memoria. Además, reforzamos la importancia de leer cuidadosamente el manual y de probar paso a paso nuestras ideas para encontrar errores lógicos en el código.