VI Floating 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 Flotante, Fraccionario, Convertidor, IEEE

1 Introducción

En esta práctica se implementa un sistema capaz de convertir números en base decimal al formato IEEE 754, utilizando operaciones aritméticas. El objetivo principal es explorar la representación precisa de números fraccionarios conforme al estándar IEEE 754, evitando el uso directo de operaciones de punto flotante. Esta aproximación resulta especialmente útil en sistemas embebidos con recursos de procesamiento limitados, donde se busca optimizar el rendimiento sin comprometer la exactitud numérica.

2 Objetivos

- Implementar un sistema de conversión de números decimales (base 10) al formato de punto flotante IEEE 754.
- Realizar las operaciones necesarias para codificar con precisión números fraccionarios en el estándar IEEE 754 sin emplear directamente instrucciones de punto flotante.
- Validar y visualizar los resultados de la conversión, asegurando su aplicabilidad en sistemas embebidos o entornos donde se requiere comprender y manipular directamente la representación binaria en punto flotante.

3 Pseudocódigo y Explicación del Código

A continuación, se presenta el pseudocódigo:

Algorithm 1 Conversión de dos números a formato IEEE 754

Función:

```
    // Conversión del primer número
1   Cargar Frac1 en R11  Cargar Decimal0s (10^6) en R3  Cargar 31 en R4 (precisión)
     Llamar a Fract()
2   Cargar Val1 en R2  Llamar a Integer()
3   Cargar Val1 en R2  Llamar a Exponente()
4   Cargar Sign1 en R7  Llamar a Signo()
5   Guardar resultado en la pila  Limpiar registros temporales
    // Conversión del segundo número
6   Repetir los mismos pasos anteriores con Frac2, Val2 y Sign2
7   Extraer ambos resultados de la pila (POP)  Fin del programa (ciclo infinito)
```

Algorithm 2 Construcción del bit de signo

Función:

Desplazar R7 31 bits a la izquierda (bit de signo) OR lógico con R9

Algorithm 3 Cálculo del exponente en formato IEEE

Función:

if *R2 == 0* **then**

Llamar a ZeroExp()

else

Contar ceros a la izquierda en R2 y guardar en R3 Calcular EXP = 158 - CLZ(R2) Desplazar EXP 23 bits a la izquierda OR con R9

Algorithm 4 Conversión de parte entera a mantisa IEEE

Función:

Contar ceros a la izquierda en R2 y ajustar para alinear el bit más significativo Alinear los bits y aislar los 23 bits fraccionarios relevantes (mantisa) OR con R9

Algorithm 5 Conversión de parte fraccionaria

Función:

Convertir parte fraccionaria multiplicando por 2 repetidamente Incluir bits significativos de la fracción hasta alcanzar la precisión

GitHub Repository

3.1 Direcciones y Constantes

- **Val1, Frac1**: Parte entera y fraccionaria, respectivamente, del primer número decimal a convertir.
- **Val2, Frac2**: Parte entera y fraccionaria del segundo número decimal.
- **Decimal0s (1000000)**: Constante utilizada para escalar la fracción decimal como si tuviera seis ceros decimales (equivalente a multiplicar por 10^6).
- **Sign1, Sign2**: Constantes binarias (0 o 1) que representan el signo de cada número. Usado para establecer el bit de signo en el resultado IEEE.

3.2 Inicialización del Proceso

Cuando se ejecuta la rutina `__main`, se preparan los datos para convertir dos números separados (cada uno con parte entera y fraccionaria) al formato de punto flotante IEEE 754 de 32 bits.

- Se cargan los valores de `Frac1` y `Frac2` en `R11`, la constante `Decimal0s` en `R3`, y la precisión en `R4`.
- Se llama a la subrutina `Fract` para convertir la parte fraccionaria a binario.
- Luego se carga `Val1` o `Val2` en `R2` y se llama a `Integer` y `Exponente`, que generan la mantisa y el exponente.
- Finalmente, se carga el valor de signo `Sign1` o `Sign2` en `R7` y se invoca la rutina `Signo` para establecer el bit más significativo.
- El resultado final se construye en `R9` y se guarda temporalmente en la pila.

3.3 Lógica de Conversión a IEEE 754

El programa realiza la conversión manual del número decimal al formato IEEE 754 de precisión simple (32 bits), dividiendo el resultado en tres componentes:

- **Bit de signo**: Determinado por el valor de `Sign1` o `Sign2` (0 para positivo, 1 para negativo), ubicado en el bit 31.
- **Exponente**: Calculado como $127 + (31 - \texttt{CLZ}(parte\_entera))$ y desplazado al campo de 8 bits correspondiente.
- **Mantisa**: Derivada del valor entero y la fracción convertida, normalizados para ajustarse a los 23 bits menos significativos del formato IEEE.

Para números que no tienen parte entera (i.e., igual a cero), se activa una lógica especial en la rutina `ZeroExp`, que ajusta el exponente para representar números subnormales y normaliza la fracción directamente.

3.4 Subrutinas

- `Fract`: Convierte la parte fraccionaria en una secuencia binaria desplazando bits con precisión de 31 ciclos.
- `Integer`: Alinea el bit más significativo de la parte entera y extrae los bits más relevantes para la mantisa.
- `Exponente`: Calcula el valor del exponente usando la instrucción `CLZ` y lo codifica en su posición correspondiente.
- `ZeroExp`: Maneja el caso en que la parte entera es cero, normalizando la fracción y ajustando el exponente como subnormal.

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

```
Sign1        EQU 1
Val1         EQU 0
Frac1        EQU 0
;Decimal0s   EQU 1000
Sign2       EQU 0
Val2        EQU 0
Frac2       EQU 0
Decimal0s   EQU 1000 ;
        AREA data, DATA, READWRITE
        AREA juve3dstudio,CODE,READONLY
        ENTRY
        EXPORT __main

__main
    EOR     R2, R2
    LDR     R11, =Frac1
    LDR     R3, =Decimal0s
    LDR     R4, =31
    BL      Fract
    LDR     R2, =Val1
    BL      Integer
    LDR     R2,=Val1
    BL      Exponente
    LDR     R7,=Sign1
    BL      Signo
    PUSH{R9}
    EOR         R9,R9
    BL                  Limpiar
    EOR     R2, R2
    LDR     R11, =Frac2
    LDR     R3, =Decimal0s
    LDR     R4, =31
    BL      Fract
    LDR     R2, =Val2
    BL      Integer
    LDR     R2,=Val2
    BL      Exponente
    LDR     R7,=Sign2
    BL      Signo
    PUSH{R9}
    EOR         R9,R9
    BL                  Limpiar
    EOR  R7,R7
    POP {R2}
    POP {R1}

ciclo
        b ciclo

Signo
    LSL     R7, #31
        ORR         R9, R7
    BX      LR

Exponente
    PUSH{LR}
    CMP R2, #0
    BLEQ        ZeroExp
    CLZ     R3, R2
    RSB     R3, #158
    LSL     R3, #23
    ORR     R9,R3
    POP{LR}
    BX      LR

ZeroExp
        POP{LR}
        LSL     R9, R7, #31   ;Signo

        CMP                 R11, #0
        BXEQ        LR

        EOR         R9, R9
        CLZ     R3, R12
        ADD                 R3, #1

        RSB                 R3, #127
        LSL     R3, #23

        CLZ                 R4, R12
        ADD         R4, #1

        LSL         R12, R4

        LSR                 R12, #9

        ORR     R12,R3
        ORR                 R9, R12
        BX LR

Integer
    CLZ     R3, R2
    ADD     R3, #1

    LSL     R2, R3
    LSR     R2, R3 ; aca tronamos el mas significativo

    ;CLZ     R3, R2
    RSB     R3, R3, #32 ; numero de digitos

        EOR     R12, R12
        ADD     R12, R5

    LSR     R5, R3

        RSB     R6, R3, #32

    LSL     R2, R6
    ORR     R5, R2

    LSR     R5, #9

    ORR     R9, R5

    BX      LR

Fract
        EOR  R9,R9
    ;R2 valor fraccionario
    ;R3 Presiocion
    ;R5 valor
    LSL  R11,#1
    CMP  R11,R3
    BLE  Zero
    ORR  R5,#1
    SUB  R11,R3

Zero
    LSL  R5,#1
    SUBS R4, #1
    BNE  Fract
    BX   LR

    end
```

4 Conclusiones

El mayor reto que enfrentamos en esta práctica fue comprender y aplicar correctamente el proceso de conversión de la parte fraccionaria de un número decimal al formato IEEE 754. Inicialmente, tuvimos dificultades para generar con precisión la representación binaria, especialmente al calcular la parte fraccionaria del número. Esto nos obligó a revisar con detenimiento la lógica de desplazamientos y redondeo.

Esta práctica nos permitió entender mejor cómo funciona la representación en punto flotante conforme al estándar IEEE 754 y por qué es crucial conocer tanto la estructura del formato (signo, exponente y mantisa) como las reglas que rigen su codificación.

Esto nos permitió tener un entendimiento mas extenso para poder aplicarlo a la calculadora.