Fundamentos de Arquitetura de Computadores

Tiago Alves

Faculdade UnB Gama
Universidade de Brasília

Representação em Ponto Fixo: Comparativo (4 bits)

Decimal	Complemento de 2	Complemento de 1	Signed Representation	Excess $B = 8$
-8	**1000**	-	-	**0000**
-7	**1001**	**1000**	**1111**	**0001**
-6	**1010**	**1001**	**1110**	**0010**
-5	**1011**	**1010**	**1101**	**0011**
-4	**1100**	**1011**	**1100**	**0100**
-3	**1101**	**1100**	**1011**	**0101**
-2	**1110**	**1101**	**1010**	**0110**
-1	**1111**	**1110**	**1001**	**0111**
0	**0000**	**1111** ou **0000**	**1000** ou **0000**	**1000**
1	**0001**	**0001**	**0001**	**1001**
2	**0010**	**0010**	**0010**	**1010**
3	**0011**	**0011**	**0011**	**1011**
4	**0100**	**0100**	**0100**	**1100**
5	**0101**	**0101**	**0101**	**1101**
6	**0110**	**0110**	**0110**	**1110**
7	**0111**	**0111**	**0111**	**1111**

Faixa dinâmica: +7 a -7 (-8) Casas decimais: Q0

Representação de casas decimais em complemento de 2 (8 bits)

Q3: $-1 \times 2^4 2^3 2^2 2^1 2^0, 2^{-1} 2^{-2} 2^{-3}$

- Menor valor: $10000,000 = -2^4 = -16$
- Maior valor: $01111,111 = 2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2} + 2^{-3} = 15,875$

Q1: $-1 \times 2^6 2^5 2^4 2^3 2^2 2^1 2^0, 2^{-1}$

- Menor valor: $1000000,0 = -2^6 = -64$
- Maior valor: $0111111,1 = 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 + 2^{-1} = 63,5$

Q7: $-1 \times 2^0, 2^{-1} 2^{-2} 2^{-3} 2^{-4} 2^{-5} 2^{-6} 2^{-7}$

- Menor valor: $1,0000000 = -1 \times 2^0 = -1$
- Maior valor: $0,1111111 = 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-6} = 0,9921875$

UnB Gama

Ponto Fixo

Operações Matemáticas: da mesma forma que inteiros usando mesmo Q.

- Soma
- Subtração
- Multiplicação
- Divisão (ex.: 5/8)

	Q0	Q2	Q7
01010001	81	20.25	0.6328125
+ 10111001	-71	-17.75	-0.5546875
00001010	10	2.50	0.0781250
00000110	6	1.5	0.046875
x 00001010	x 10	x2.5	x0.078125
00000000 00111100	60	3.75	0.003662109375

Ponto Fixo

Problemas:

- Pequena Faixa Dinâmica
- Precisão depende da faixa dinâmica

Vantagens:

- Aritmética simples e rápida!

Ponto Flutuante

Sistemas computacionais necessitam de uma maneira de representar grande faixa dinâmica:

- números muito pequenos: 0,00000000000000001182721226716
- números muito grandes: 167283876351200000000000000000

Notação Científica (base 10): Mantissa ou Significando / Característica ou Expoente

- Ex.: $1.182721226716 \times 10^{-15}$ e $1.672838763512 \times 10^{29}$
- Sempre normalizado, isto é, apenas 1 dígito não decimal (diferente de zero).

Notação de Engenharia (expoente múltiplo de 3, prefixo YZPTGMk_mμnpfazy) :

- Ex.: $0.00015 = 1.5 \times 10^{-4} = 150 \times 10^{-6} = 150\mu$
- Ex.: $15000 = 1.5 \times 10^{4} = 15 \times 10^{3} = 15k$

Notação Científica (base 2):

- Ex.: $1.010_2 \times 2^{-2} = 0.01010_2 = (1 + 0.25) \times 2^{-2}$

Representação em Ponto Flutuante

A notação mais usada para representar números fracionários é a notação em ponto flutuante padrão IEEE 754.

$$v = -1^s \cdot m \cdot b^e$$

Onde:

- s representa o sinal (0 para positivo, 1 para negativo)
- m representa a mantissa (*significand*)
- b representa a base. No padrão IEEE 754 a base pode ser 10 ou 2, mas em sistemas computacionais utiliza-se sempre a base 2 (por que?)
- e representa o expoente. Usaremos notação em excesso; dessa forma $e = (\text{exp\_cod} - \text{bias})$

Representação em Ponto Flutuante

Existem dois tipos principais de números:

	Single Precision (float)	*Double Precision* (double)
Sinal	1 bit	1 bit
Expoente	8 bits	11 bits
Mantissa	23 bits	52 bits
Total	32 bits	64 bits

Representação em Ponto Flutuante: IEEE 754

- O bit 1 inicial do significando está implícito (aumenta a precisão)
- O expoente possui um offset para facilitar a ordenação
  - Offset de 127 para precisão simples e de 1023 para precisão dupla.
- Formato:

$$-1^{\text{sinal}} \times (1 + \text{fração}) \times 2^{(\text{exp\_cod} - \text{offset})}$$

Exemplo:

- decimal: -0.75
- ponto fixo (sinal e magnitude): $-0.11_2 = -1.1_2 \times 2^{-1}$ em notação científica
- ponto flutuante:

  exp_cod $- 127 = -1 \Longrightarrow$ exp_cod $= 126 =$ 01111110 (é o código em precisão simples para representar o expoente -1)

  exp_cod $=$ 1022 $=$ 01111111110 (precisão dupla)

  precisão simples IEEE: 1**01111110**10000000000000000000000

  precisão dupla IEEE: 1**01111111110**10000000000000000000000000000000000000000000000000

Representação em Ponto Flutuante: IEEE 754

- O bit 1 inicial do significando está implícito (aumenta a precisão)
- O expoente possui um offset para facilitar a ordenação
  - Offset de 127 para precisão simples e de 1023 para precisão dupla.
- Formato:

$$-1^{\text{sinal}} \times (1 + \text{fração}) \times 2^{(\text{exp\_cod} - \text{offset})}$$

Exemplo:

- decimal: 5.0
- ponto fixo (sinal e magnitude): $0101.0_2 = 1.01_2 \times 2^2$ em notação científica
- ponto flutuante:

  exp_cod $- 127 = 2 \Longrightarrow$ exp_cod $= 129 =$ 10000001 (precisão simples)

  exp_cod $=$ 1025 $=$ 10000000001 (precisão dupla)

  precisão simples IEEE: 0**10000001**01000000000000000000000

  precisão dupla IEEE: 0**10000000001**0100000000000000000000000000000000000000000000000000

Representação em Ponto Flutuante: IEEE 754

Fazendo o caminho em sentido reverso....

Dado o número em FP IEEE754: 0xC1100000 qual o número decimal representado?

- 1100 0001 0001 0000 0000 0000 0000 0000
- 1**100 0001 0**001 0000 0000 0000 0000 0000
- exp_cod $- 127 = 130 - 127 = 3 \Longrightarrow 2^{130-127} = 2^3$
- Mantissa $= 1.001_2$
- Logo: $(-1)^1 \times 1.001_2 \times 2^3 = (-1001.0)_2 = -9$

Representação em Ponto Flutuante: IEEE 754

Como representar o 0 ?

Single precision		Double precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	Nonzero	0	Nonzero	± denormalized number
1–254	Anything	1–2046	Anything	± floating-point number
255	0	2047	0	± infinity
255	Nonzero	2047	Nonzero	NaN (Not a Number)

Operações em Ponto Flutuante

Adição e Subtração:

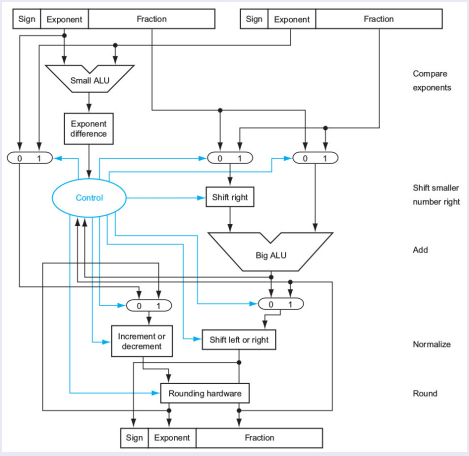
Em notação científica decimal com limite de representação de 4 dígitos (significado) e 2 dígitos (expoente):

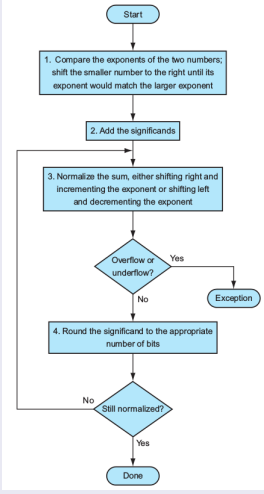
- $9,999 \times 10^2 + 1,61 \times 10^{-1} = 9,999 \times 10^2 + 0,00161 \times 10^2 =$
- $= 9,999 \times 10^2 + 0,0016 \times 10^2 = 10,015 \times 10^2 = 1,0015 \times 10^3 =$
- $= 1,002 \times 10^3$

Em notação científica binária com 4 bits de precisão: 0,5+(-0.4375)

- $0.5 = 0.1_2 = 1.0_2 \times 2^{-1}$
- $-0.4375 = -0.0111_2 = -1.11_2 \times 2^{-2} = -0.111_2 \times 2^{-1}$
- $1.0_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = (1.000 - 0.111)_2 \times 2^{-1} = 0.001_2 \times 2^{-1} = 1.0_2 \times 2^{-4} = 0.0625$

Adição em Ponto Flutuante





Multiplicação em Ponto Flutuante

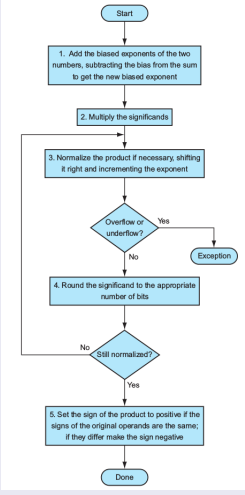
Decimal

- $3.23 \times 10^2 \times 3.415 \times 10^{-1} = 11.03045 \times 10^1$
- (4 dígitos) $= 1.103 \times 10^2$

Binário

- $0.5 \times (-0.4375)$
- $1.000_2 \times 2^{-1} \times (-1.110_2 \times 2^{-2}) = -1.110000_2 \times 2^{-3}$
- (4 bits) $= -1.110_2 \times 2^3$

Obs.: IEEE 754 Expoentes com offset, logo é necessário diminuir 1 offset da soma dos expoentes.



Ponto Flutuante: Arredondamento

O IEEE754 permite 4 tipos de arredondamentos:

- Sempre para $+\infty$ (cima, ceil): 2.11 $\Longrightarrow$ 2.2 e 2.15 $\Longrightarrow$ 2.2 e 2.19 $\Longrightarrow$ 2.2
- Sempre para $-\infty$(baixo, floor): 2.11 $\Longrightarrow$ 2.1 e 2.15 $\Longrightarrow$ 2.1 e 2.19 $\Longrightarrow$ 2.1
- Truncamento: Despreza os bits menos significativos (trunc) $+1.01101_2 = 1.40625 \Longrightarrow +1.011_2 = 1.375$
- Ao mais próximo (round): 2.11 $\Longrightarrow$ 2.1 e 2.19 $\Longrightarrow$ 2.2 e 2.15 $\Longrightarrow$?$\Uparrow\Downarrow$?

  Estatisticamente coerente: Ao dígito par: 2.15 $\Longrightarrow$ 2.2 e 2.25 $\Longrightarrow$ 2.2

Obs.: Em precisão limitada $(x + y) + z \neq x + (y + z)$

$x + (y + z) = -1,5 \times 10^{38} + (1,5 \times 10^{38} + 1,0) = 0,0$

$(x + y) + z = (-1,5 \times 10^{38} + 1,5 \times 10^{38}) + 1,0 = 1,0$

Ponto Flutuante no MIPS: Coprocessador 1

32 Registradores de ponto flutuante: $f0, $f1, ..., $f30 e $f31 ou

16 Registradores de precisão dupla: $f0, $f2, $f4, ..., $f30

Permite operações com precisão simples e dupla:

- Adição: `add.s` e `add.d` # `add.s $f0,$f1,$f2`
- Subtração: `sub.s` e `sub.d` # `sub.d $f0,$f2,$f4`
- Multiplicação: `mul.s` e `mul.d` # `mul.s $f0,$f1,$f2`
- Divisão: `div.s` e `div.d` # `div.d $f0,$f4,$f8`
- Comparação: `c.x.s` e `c.x.d` # `c.eq.s 0,$f1,$f2`
  onde x pode ser: eq, neq, lt, le, gt, ge
  Seta um bit do byte de flag como V ou F
- Desvio se flag V (bc1t) desvio se F (bc1f) # `bc1t 0,LABEL`
- Load e Store: `lwc1 e swc1, ldc1 e sdc1` # `lwc1 $f0,Imm($s2)`
- Move: `mfc1, mtc1` # `mfc1 $t0, $f0`
- `cvt.x.y` converte de y para x (s,d,w) # `cvt.s.w $f2,$f4`

Conversor de Temperatura

```
float f2c(float fahr)
{
  return ((5.0/9.0)*(fahr-32.0));
}
```

Considerando que o argumento de entrada em $f12 e saída em $f0.

```
#constantes na memória ($gp+off)
fahr:
  lwc1 $f16, const5($gp)
  lwc1 $f18, const9($gp)
  div.s $f16,$f16,$f18
  lwc1 $f18, const32($gp)
  sub.s $f18, $f12, $f18
  mul.s $f0, $f16, $f18
  jr $ra
```

```
#constantes segmento de dados
.data
const5:   .float 5.0
const9:   .float 9.0
const32: .float 32.0
.text
fahr:
  l.s $f16, const5
  l.s $f18, const9
  div.s $f16,$f16,$f18
  l.s $f18, const32
  sub.s $f18, $f12, $f18
  mul.s $f0, $f16, $f18
  jr $ra
```

Complexidade em operações com Ponto Flutuante

As operações aritméticas são mais complexas

Além do overflow podemos ter underflow

A precisão pode ser um grande problema:

- O IEEE 754 mantém dois bits extras, guarda e arredondamento
- quatro modos de arredondamento
- positivo dividido por zero produz infinito
- zero dividido por zero produz um não-número (NaN)
- outras complexidades...

Implementar o padrão pode ser arriscado

Não usar o padrão pode ser ainda pior

- 80x86 e o bug do Pentium! (07, 09, 11, 12 de 1994)

Resumo

- A aritmética de computador é restrita por uma precisão limitada Os padrões de bit não têm um significado inerente mas existem padrões
  - complemento de dois
  - ponto flutuante IEEE 754
- As instruções de computador determinam o “significado” dos padrões de bit
- O desempenho e a precisão são importantes; portanto, existem muitas complexidades nas máquinas reais
- A escolha do algoritmo é importante e pode levar a otimizações de hardware para espaço e tempo (por exemplo, multiplicação)