

# PSI3441 – Arquitetura de Sistemas Embarcados

---

## - Ponto Flutuante

---

Escola Politécnica da Universidade de São Paulo



Prof. Gustavo Rehder – [grehder@lme.usp.br](mailto:grehder@lme.usp.br)

Prof. Sergio Takeo – [kofuji@usp.br](mailto:kofuji@usp.br)

Prof. Antonio Seabra – [acseabra@lsi.usp.br](mailto:acseabra@lsi.usp.br)



# Representação de Ponto Flutuante (IEEE 754)

- Representa números entre  $\pm 1.4 \times 10^{-45}$  a  $\pm 3.4 \times 10^{38}$
- Erro de arredondamento
  - 45.45 → 45.4500000000000000284... (double)
  - 45.4500007629... (single)

Single precision (32-bit) form: (Bias = 127)



Double precision (64-bit) form: (Bias = 1023)



$$(-1)^{\text{sign bit}} (1 + \text{fraction}) \times 2^{\text{exponent} - \text{bias}}$$



# Representação de Ponto Flutuante (IEEE 754)

- Sinal:
  - Bit 31 = 0  $\rightarrow (-1)^0 = 1$  (Positivo)
  - Bit 31 = 1  $\rightarrow (-1)^1 = -1$  (Negativo)

Single precision (32-bit) form: (Bias = 127)

(1) sign (8) exponent (23) fraction

$$(-1)^{\text{sign bit}}(1+\text{fraction}) \times 2^{\text{exponent} - \text{bias}}$$



# Conversão de Decimal para Ponto Flutuante (IEEE 754)

• 45.45 (decimal)

– Converter inteiro para binário: 45 → 10 1101

– Converter a fração para binário.  $0.45 \times 2 = 0.9$

45.45 → 101101.011100...

Normalização →  $1.01101011100... \times 2^5$

$$(-1)^{\text{sign bit}} (1 + \text{fraction}) \times 2^{\text{exponent} - \text{bias}}$$

$$(-1)^0 (1.011010...) \times 2^{132 - 127} = 1.011010 \times 2^5$$

0 10000100 01101011100110011001100

- $0.45 \times 2 = 0.9$
- $0.90 \times 2 = 1.8$
- $0.80 \times 2 = 1.6$
- $0.60 \times 2 = 1.2$
- $0.20 \times 2 = 0.4$
- $0.40 \times 2 = 0.8$
- $0.80 \times 2 = 1.6$

Valores se repetem

011100...

Single precision (32-bit) form: (Bias = 127)

(1) sign (8) exponent (23) fraction



# Conversão de Ponto Flutuante para Decimal

Single precision (32-bit) form: (Bias = 127)

(1)sign (8) exponent (23) fraction

0 10000100 01101011100110011001100

binário	decimal
↓	↓
0.1 = 2 <sup>-1</sup>	= 0.5
0.01 = 2 <sup>-2</sup>	= 0.25
0.001 = 2 <sup>-3</sup>	= 0.125
	⋮

- Sinal: 0 → Positivo
- Expoente: 10000100 → 132
- Fração: 01101011100110011001100

$$(-1)^{\text{sign bit}}(1+\text{fraction}) \times 2^{\text{exponent} - \text{bias}}$$

$$(-1)^0(1.011010\dots) \times 2^{132 - 127} = 1.01101011100110011001100 \times 2^5$$

101101.011100110011001100

↓	↓	
45	0 · 2 <sup>-1</sup> + 1 · 2 <sup>-2</sup> + 1 · 2 <sup>-3</sup> + 1 · 2 <sup>-4</sup> + 0 · 2 <sup>-5</sup> + ...	= 0.25 + 0.125 + 0.0625 + ...

45.4500007629...



# Soma ou Subtração com Ponto Flutuante

- Checar se existem zeros
- Igualar expoentes
- Soma ou subtração
- Normalizar resultados

- Exemplo (8 bits : bias = 7)

X+Y

$$X = 7 = 0\ 1001\ 110 \rightarrow 1.110 \times 2^2$$

$$Y = 1 = 0\ 0111\ 000 \rightarrow 1.000 \times 2^0 \rightarrow 0.01000 \times 2^2$$

} Mesmo  
Expoente

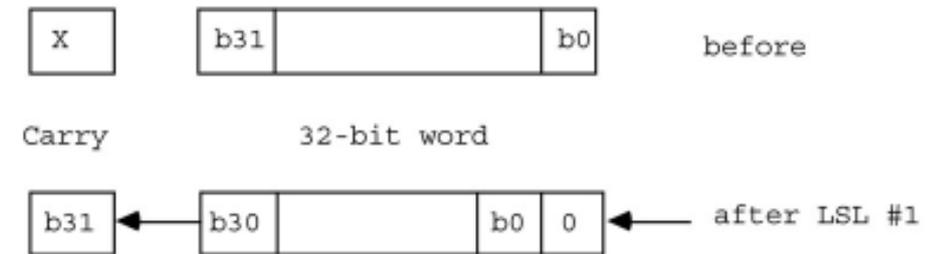
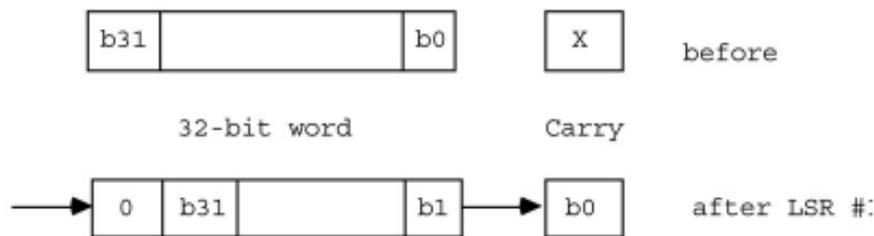
$$\begin{array}{r} 1.110 \times 2^2 \\ + 0.010 \times 2^2 \\ \hline 10.000 \times 2^2 \end{array} \rightarrow 1.000 \times 2^3 \rightarrow 0\ 1010\ 000$$

$$X + Y = 8$$



# Como se iguala os expoentes?

- Logical Shift Right (LSR) ou Logical Shift Left (LSL)



LSR Rd, Rm, Op2

LDR R2, =0x0001 0000

LSR R0, R2, #8

R0 = 0000 0010

C = 0

LSL Rd, Rm, Op2

LDR R2, =0x0001 0000

LSL R0, R2, #8

R0 = 0100 0000

C = 0



# Addição - inteiros e ponto flutuante

- Comparação

```
int a = 2;  
int b = 2;  
int c = 0;  
c = a+b;
```



22 ticks de clock  
com o KL25Z

```
float a = 2.0;  
float b = 2.0;  
float c = 0.0;  
c = a+b;
```



98 ticks de clock  
com o KL25Z

# PSI3441 – Arquitetura de Sistemas Embarcados

---

## - Multiplicação e Divisão

---

Escola Politécnica da Universidade de São Paulo





# Multiplicação (MUL) – Unsigned Int

MUL Rd, Rn, Opt2 ; Rd = Rn x Opt2

- Cortex M0+ possui duas implementações possíveis:
  - Rápida – executa em um único ciclo → Usado pelo KL25Z (Hardware)
  - Pequena – multiplicador iterativo executa em 32 ciclos  
↳ Instruções add/sub/shift usadas quando não existe a opção de 1-ciclo (Software)
- A Instrução MUL realiza a multiplicação de dois números de 32 bits (Rn e Opt2) e retorna em Rd um número de 32 bits (LSBs).
- Multiply and Accumulated (MLA) → Instrução não disponível no Cortex M0+  
MLA Rd, Rm, Rs, Rn ; Rd = Rm x Rs + Rn



# Multiplicação Long

- Instruções que multiplicam números de 32 bits e retornam números de 64 bits

UMULL – Unsigned Multiply Long

UMLAL – Unsigned Multiply with Accumlate Long

SMULL – Signed Multiply Long

SMLAL – Signed Multiply with Accumlate Long

Disponíveis  
a partir do  
Cortex M3



# Multiplicação- inteiros e ponto flutuante (Cortex M0+)

- Comparação

```
int a = 2;  
int b = 2;  
int c = 0;  
c = a x b;
```



22 ticks de clock  
com o KL25Z

Utiliza MUL  
(Hardware)

```
float a = 2.0;  
float b = 2.0;  
float c = 0.0;  
c = a x b;
```



178 ticks de clock  
com o KL25Z

Utiliza add/sub/shift  
(Software)



# Divisão – Signed and Unsigned Int

- Signed Divide:  $\text{SDIV}\{\text{cond}\} \text{ Rd, Rn, Rm}$
- Unsigned Divide:  $\text{UDIV}\{\text{cond}\} \text{ Rd, Rn, Rm}$

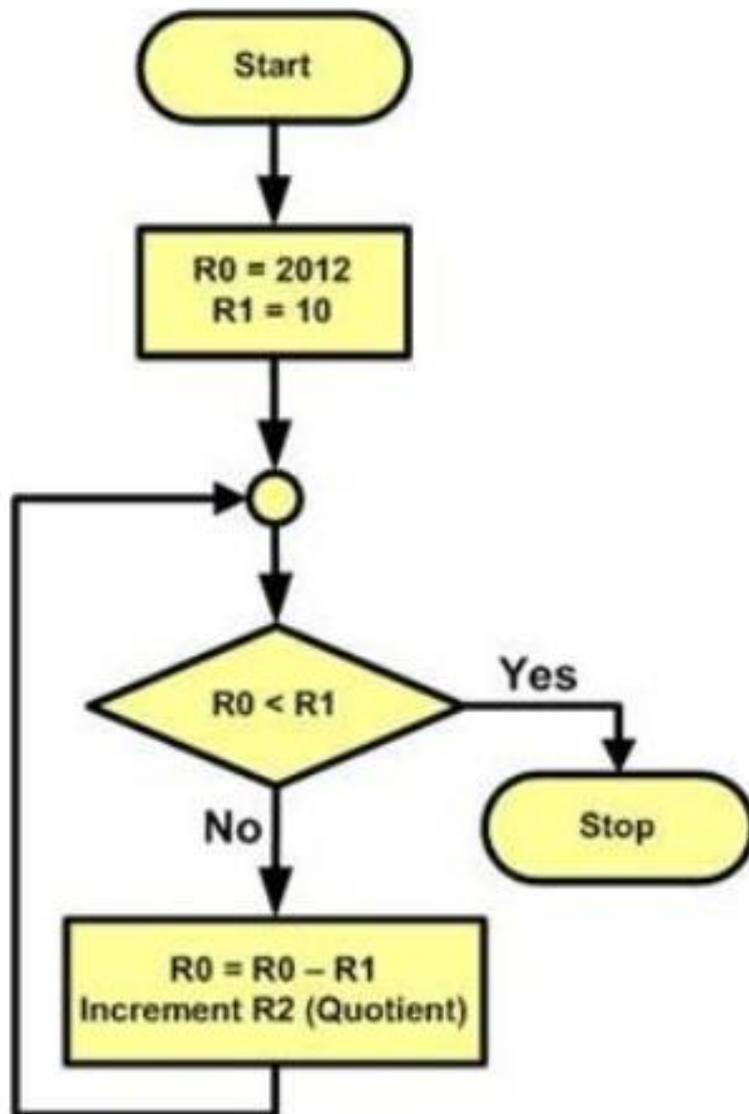
Disponíveis a partir do Cortex M3  
(Hardware)

Como fazer divisão no  
Cortex M0+ ?





# Divisão Inteiro – Cortex M0+ (Software)



Exemplo:  $R2 = R0/R1$

```
int R0 = 2012;  
int R1 = 10;  
int R2 = 0;
```

```
while (R0 >= R1)
```

```
{  
  R0 = R0 - R1;  
  R2 = R2 + 1;  
}
```

Quando R0 for maior ou igual a R1, R2 = 201 e R0 = 2



# Divisão - inteiros e ponto flutuante (Cortex M0+)

- Comparação

```
int a = 2;  
int b = 2;  
int c = 0;  
c = a / b;
```



88 ticks de clock  
com o KL25Z

Software

```
float a = 2.0;  
float b = 2.0;  
float c = 0.0;  
c = a / b;
```



429 ticks de clock  
com o KL25Z

Software

# PSI3441 – Arquitetura de Sistemas Embarcados

---

## - Pseudo-Instruções

---

Escola Politécnica da Universidade de São Paulo





# Assembly Pseudo-Instruções

- Auxiliam na programação
- Não utilizam memória
- São indicações para o Assembler traduzir o código de Assembly para Código de Máquina

Directive	Description
<b>AREA</b>	Instructs the assembler to assemble a new code or data section
<b>END</b>	Informs the assembler that it has reached the end of a source file.
<b>ENTRY</b>	Declares an entry point to a program.
<b>EQU</b>	Gives a symbolic name to a numeric constant, a register-relative value or a PC-relative value.
<b>INCLUDE</b>	It adds the contents of a file to our program.

# PSI3441 – Arquitetura de Sistemas Embarcados

---

## - Processadores ARM

---

Escola Politécnica da Universidade de São Paulo





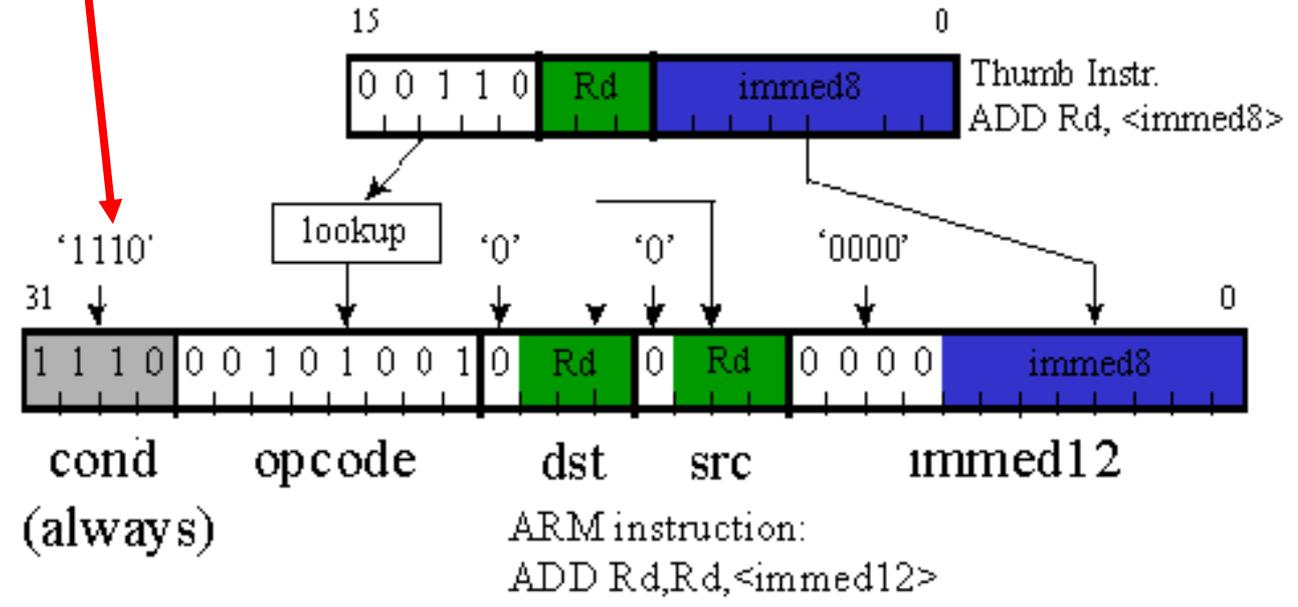
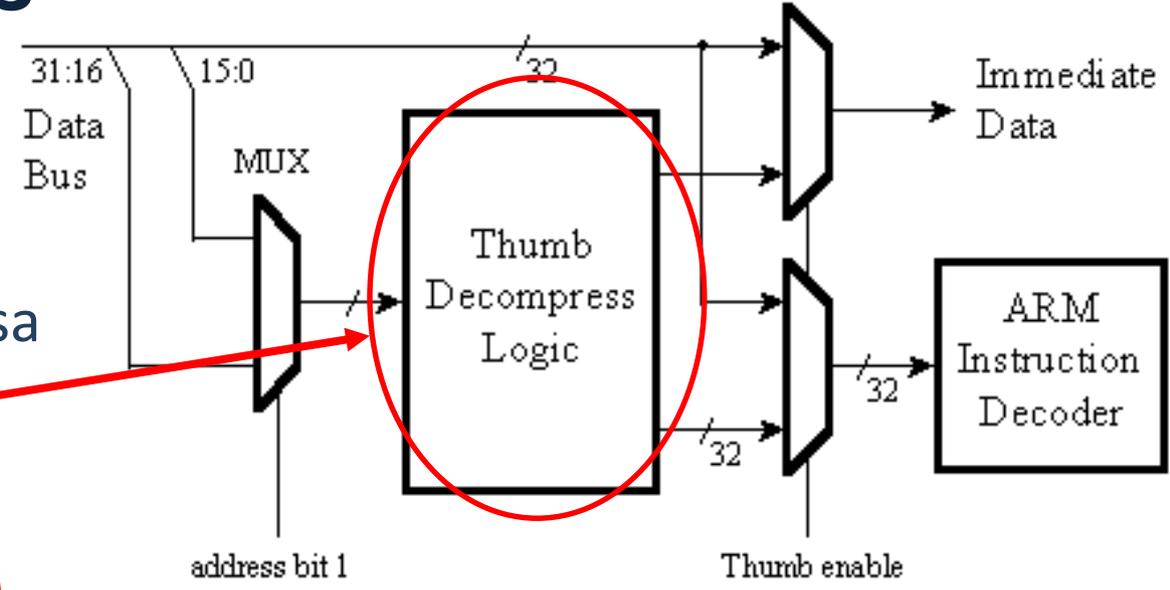
# Instruction Sets - ARM

- Processadores ARM possuem diversos grupos de instruções dependendo da versão do processador:
  - 32-bit ARM instruction set
  - 16-bit *Thumb* instruction set
  - 16/32-bit *Thumb-2* instruction set } Cortex M0+
- *Jazelle DBX* for Java byte codes (multi-tasking Java Virtual Machine)
- *NEON* 64/128-bit SIMD instruction set (Media ARM Cortex A e R52)
- *VFP* vector floating point instruction set (FPU)



# Instruction Sets

- Thumb – instrução precisa ser descomprimida
- Todas as instruções ARM podem ser condicionais





# Comparação entre Instruction Sets

Dhrystone 2.1/sec  
@ 20MHz

