
Aritmética Computacional

Sumário

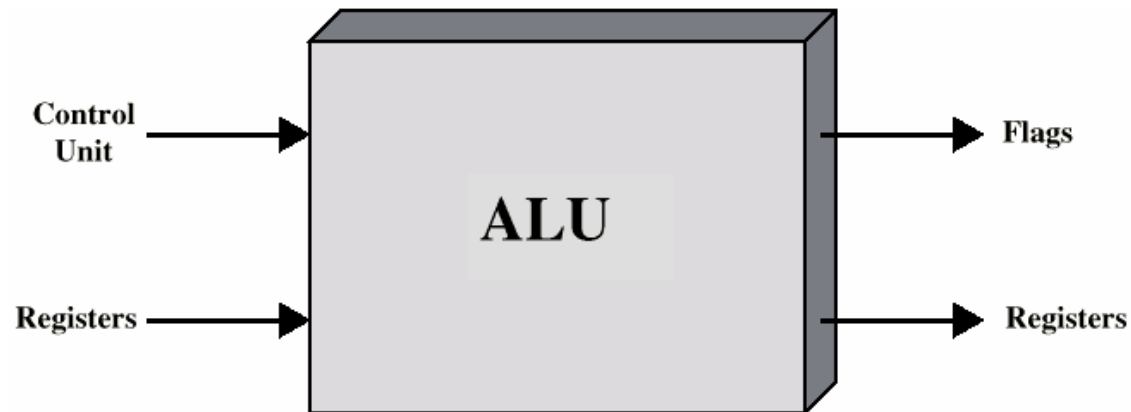
- Introdução;
- Representação de Números Inteiros;
- Aritmética de Números Inteiros;
- Representação de Números de Ponto Flutuante;
- Aritmética de Números de Ponto Flutuante;
- Bibliografia.

Introdução

- A CPU consiste de uma **unidade de controle**, dos registradores, da unidade lógica e aritmética, da unidade de execução de instruções e das interconexões entre esse componentes;
- Vejamos como funciona a unidade lógica e aritmética para implementar as **operações aritméticas**;
- Geralmente, os processadores implementam dois tipos de aritmética: de números **inteiros ou ponto fixo** e de **ponto flutuante**;

Introdução

- A ULA é a parte do computador que de fato executa as operações aritméticas e lógicas sobre os dados. Em suma, a **ULA constitui a essência de um computador**;
- A figura abaixo mostra, em termos gerais, como a **ULA** é conectada com o restante do processador:



Introdução

- Considerações:
 - Os dados são fornecidos à ULA em registradores e os resultados de uma operação armazenados em registradores;
 - A ULA pode ativar bits especiais (*flags*) para indicar o resultado de uma operação. Por exemplo, *overflow*.
 - A unidade de controle fornece sinais para controlar a operação da ULA e a transferência de dados entre a ULA e os registradores;

Representação de Números Inteiros

- Existe diversas formas para representar um número inteiro no computador. Uma delas é através do sinal-magnitude;
- Em uma palavra de n bits, os $n-1$ bits mais à direita representam a magnitude do número inteiro

$$\begin{array}{l} +110_{10} = 01101110 \\ -110_{10} = \mathbf{1}1101110 \end{array}$$

Sinal \dashrightarrow Magnitude

Representação de Números Inteiros

- As desvantagens apresentadas pela representação de sinal-magnitude são:
 - Duas representações para 0:
$$+0_{10} = 00000000$$
$$-0_{10} = 10000000$$
 - Magnitude para operandos e operadores;
- Em conseqüência, o esquema mais utilizado é a representação em complemento de dois;

Representação de Números Inteiros

- Assim como a representação sinal-magnitude, a representação em complemento de dois usa o bit mais significativo como bit de sinal;
 - Porém, os demais bits são interpretados de maneira diferente;
- Considere um inteiro A de n bits. Se A for positivo, então o bit de sinal, n_{a-1} , será igual a zero.
 - O número zero é tratado como um número positivo;
 - Faixa de números inteiros: 0 a $2^{n-1} - 1$

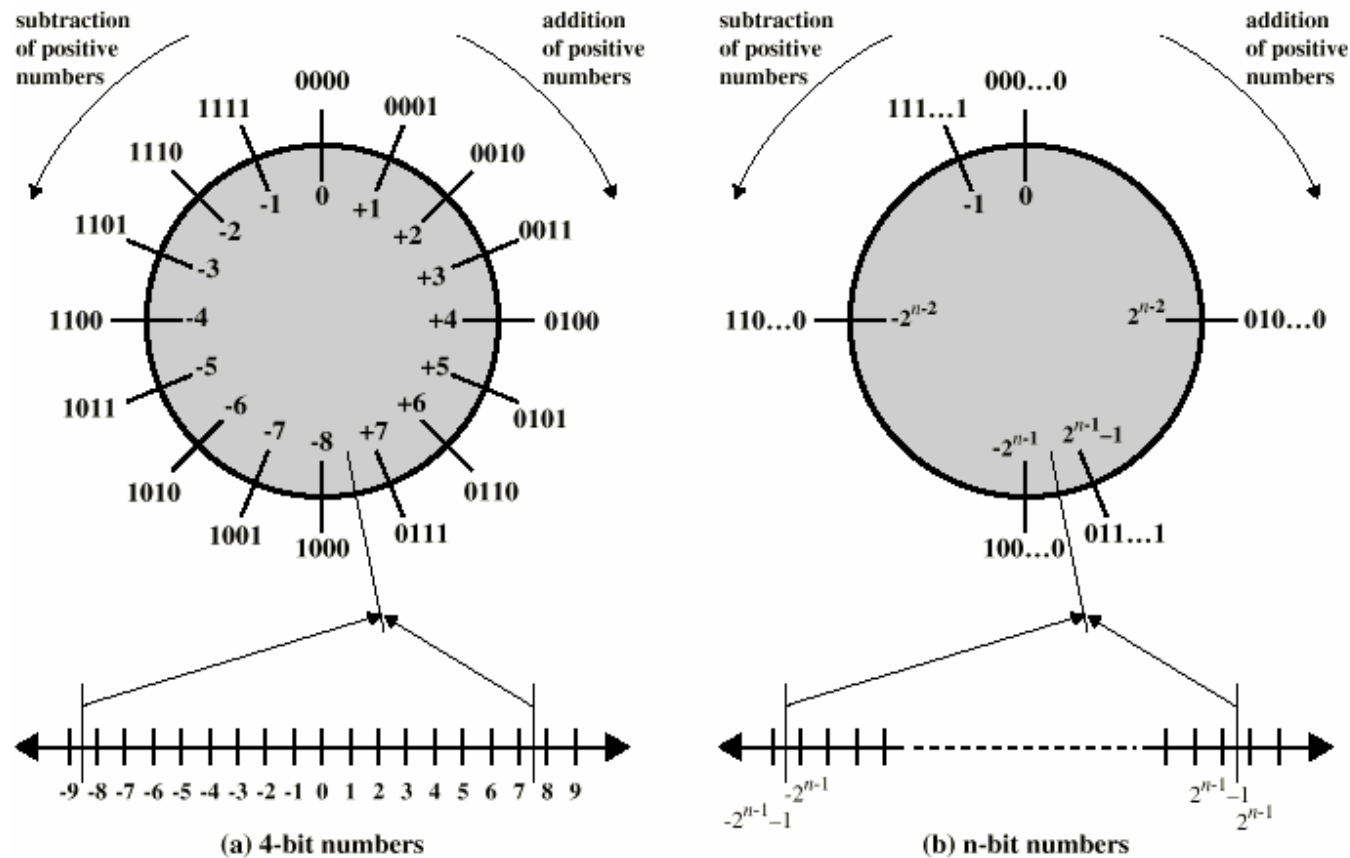
Representação de Números Inteiros

- Se A é um número negativo ($A < 0$), o bit de sinal, n_{a-1} , é 1.
- Vejamos a aplicação do complemento de 2 no inteiro positivo 4 (0100_2), por exemplo:

Inteiro original:	0100	(4 em decimal)
Complemento de 1:	1011	(Inversão dos bits)
	1011	
	+1	(Soma do bit 1 ao
Complemento de 2:	<u>1100</u>	complemento de 1)

Representação de Números Inteiros

- A representação do complemento de dois pode ser visualizada geometricamente, vejamos:



Representação de Números Inteiros

- Vejamos a representação vetorial do complemento de dois:

-128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

$$-128 \qquad \qquad \qquad +2 \quad +1 \quad = -125$$

-128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

$$-128 \qquad \qquad \qquad +8 \qquad \qquad \qquad = -120$$

- Embora a representação em complemento de dois possa parecer pouco natural, ela torna **mais simples a implementação** das operações aritméticas mais importantes (adição e subtração);

Representação de Números Inteiros

- Vejamos a **conversão** entre representação de um número inteiro com **n bits** para sua representação com **m bits**, onde $m > n$;
 - Notação sinal-magnitude:
+10 = 00001010 (8 bits) 0000000000001010 (16 bits)
-10 = 10001010 (8 bits) 1000000000001010 (16 bits)
 - Notação complemento de dois:
+10 = 00001010 (8 bits) 0000000000001010 (16 bits)
-10 = 11110110 (8 bits) 1111111111110110 (16 bits)

Aritmética de Números Inteiros

- Vejamos a implementação das operações aritméticas mais comuns em números representados em complemento de dois:
 - Negação
 - Soma e Subtração
 - Multiplicação
 - Divisão

Aritmética de Números Inteiros


- Negação: Basta aplicar o complemento de 2 ao inteiro original;

Inteiro original:	11111100	(-4 em decimal)
Complemento de 1:	00000011	(Inversão dos bits)
	11111011	
	+1	
Complemento de 2:	<u>00000100</u>	(Soma do bit 1 ao complemento de 1)

Aritmética de Números Inteiros

- Casos especiais da negação:
 - Overflow:

Inteiro original:	00000000	(0 em decimal)
Complemento de 1:	11111111	(Inversão dos bits)
	11111111	
	+1	
Complemento de 2:	<u>100000000</u>	(carry-in igual a 1- valor final = 0)

Ignorado 

Aritmética de Números Inteiros

- Casos especiais da negação:
 - -128:

Inteiro original: 10000000 (-128 em decimal)

Complemento de 1: 01111111 (Inversão dos bits)

Complemento de 2:
$$\begin{array}{r} 01111111 \\ +1 \\ \hline 10000000 \end{array}$$
 (-128 em decimal)

Anomalia que não pode ser evitada.

Aritmética de Números Inteiros

- Adição: Basta somar e quando ocorrer um "vai-um" para fora do bit mais significativo da palavra, que é ignorado.

$$\begin{array}{r} 1001 \quad (-7) \\ + 0101 \quad (+5) \\ \hline 1110 \quad (-2) \end{array} \qquad \begin{array}{r} 1100 \quad (-4) \\ + 0100 \quad (+4) \\ \hline 10000 \quad (0) \end{array}$$

- Overflow: Overflow sinalizado pela ULA para que o resultado não seja usado


$$\begin{array}{r} 0101 \quad (+5) \\ + 0100 \quad (+4) \\ \hline 1001 \quad (\textit{Overflow}) \end{array} \qquad \begin{array}{r} 1001 \quad (-7) \\ + 1010 \quad (-6) \\ \hline 10011 \quad (\textit{Overflow}) \end{array}$$

Aritmética de Números Inteiros

- Subtração: para subtrair $S - M$, pegue o complemento de dois de M e acrescente esse valor a S ($S + (-M)$).

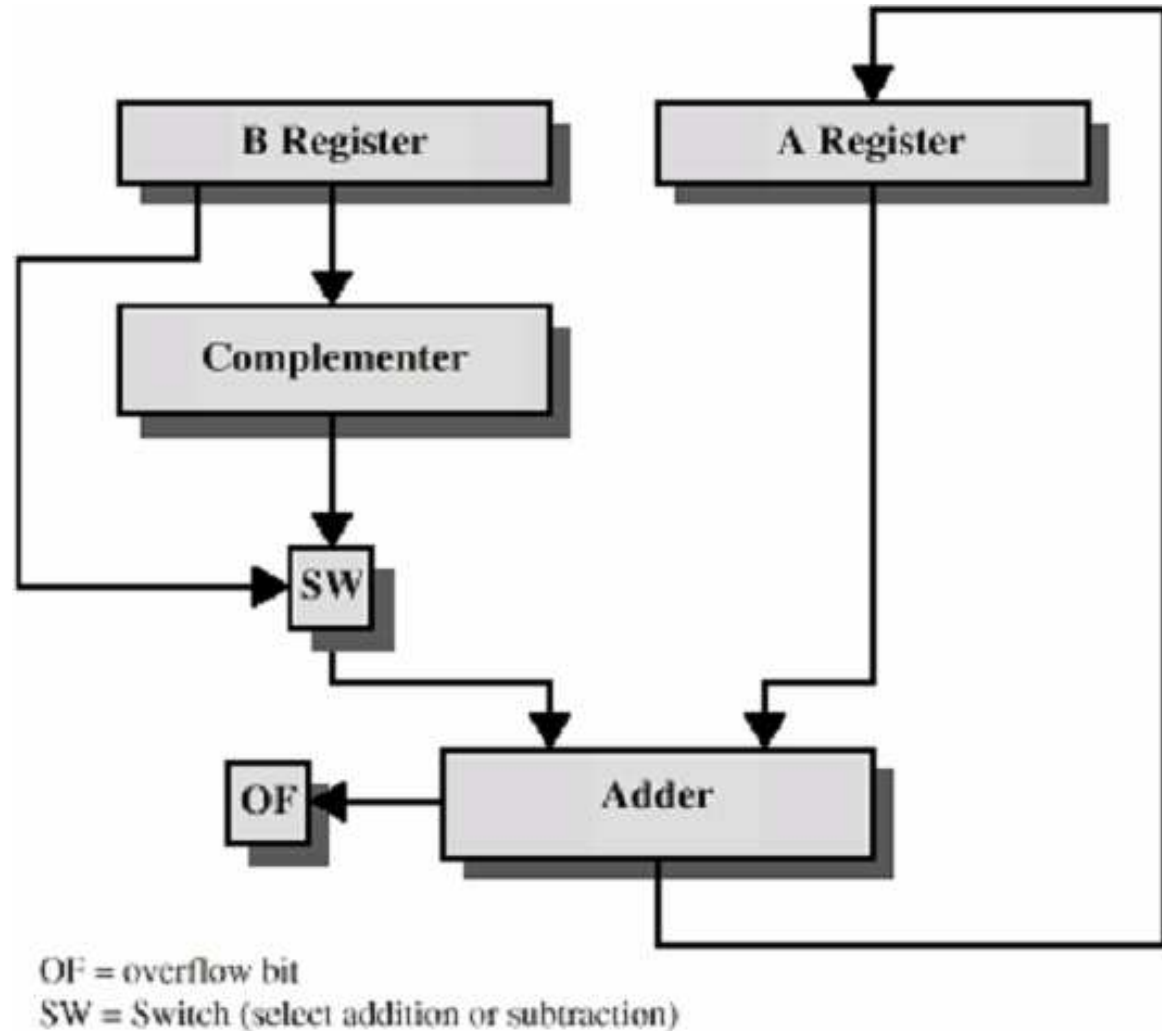
$$\begin{array}{r} 0010 \quad (+2) \\ - 0111 \quad (+7) \\ \hline \end{array} \qquad \begin{array}{r} 0010 \quad (+2) \\ + 1001 \quad (-7) \\ \hline 1011 \quad (-5) \end{array}$$

- Overflow: Overflow sinalizado pela ULA para que o resultado não seja usado

$$\begin{array}{r} 1010 \quad (-6) \\ - 0100 \quad (+4) \\ \hline \end{array} \qquad \begin{array}{r} 1010 \quad (-6) \\ + 1100 \quad (-4) \\ \hline 10110 \quad (\textit{Overflow}) \end{array}$$


Aritmética de Números Inteiros

- Vejamos os caminhos de dados e elementos de hardware necessários para efetuar a adição e a subtração:



Aritmética de Números Inteiros

- Considerações:

- Somador Binário (Meio-Somador): Elemento central que recebe dois números e produz a soma e uma indicação de overflow;
- Operandos: tratados como números inteiros sem sinal;
- Registradores: responsáveis por armazenar os operandos;
- Complemento de dois: usado para calcular o complemento de 2 sobre números negativos;

Aritmética de Números Inteiros

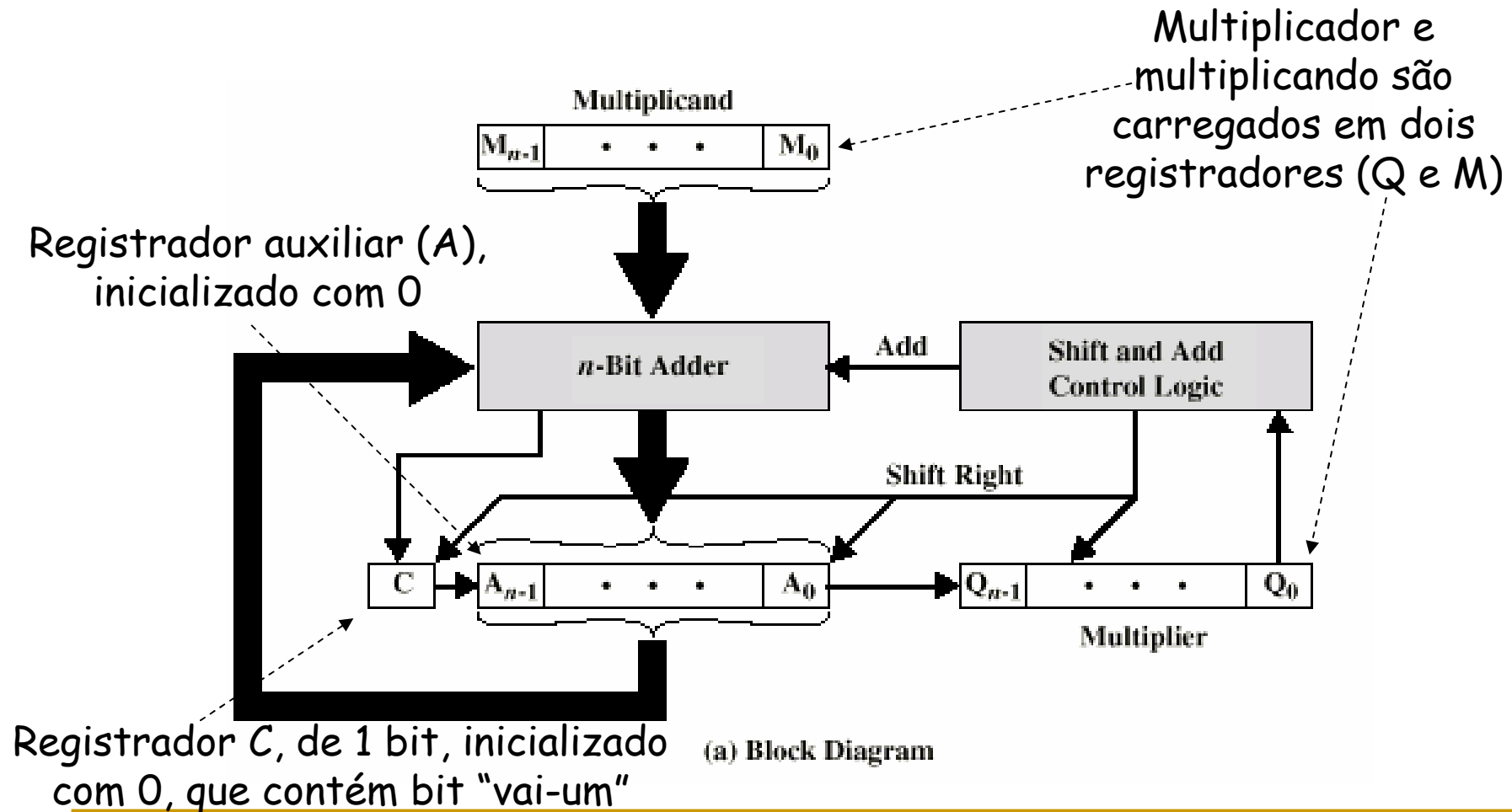
- Para a multiplicação, um grande número de algoritmos tem sido adotados em diversos computadores.
- Analisemos um problema mais simples de multiplicar, dois números inteiros sem sinal (não negativos):

				1	1	1	0	Multiplicando (14)	
			X	1	0	1	0	Multiplicador (10)	
				<hr/>					
				0	0	0	0	} Produtos Parciais	
				1	1	1	0		
		0		0	0	0			
		1		1	1	1	0		
1	0	0	0	0	1	1	0	0	Produto (140)

Aritmética de Números Inteiros

- A multiplicação pode ser feita direta e simples, mas pode ser melhorada nos seguintes aspectos:
 - Armazenamento dos produtos parciais em um único registrador;
 - Na multiplicação pelo bit 1, é necessário, apenas realizar uma operação de soma e um deslocamento;
 - Na multiplicação pelo bit 0, é necessário, apenas o deslocamento.
- Vejamos uma solução que emprega a idéia acima. Multiplicação de números inteiros binários sem sinal;

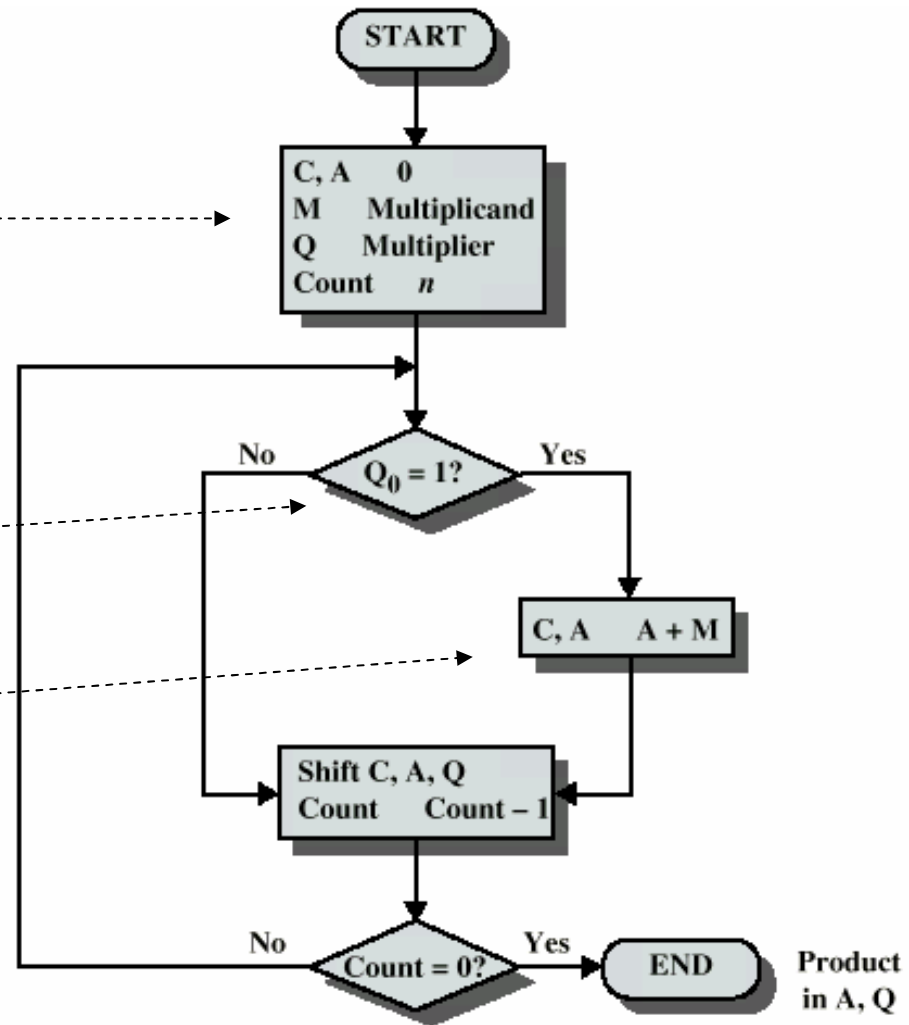
Aritmética de Números Inteiros



Aritmética de Números Inteiros

- A lógica de controle lê os bits do multiplicador, um de cada vez.

- Se Q_0 for 1:
 - o multiplicador será adicionado ao registrador A e o resultado, armazenado nesse registrador;

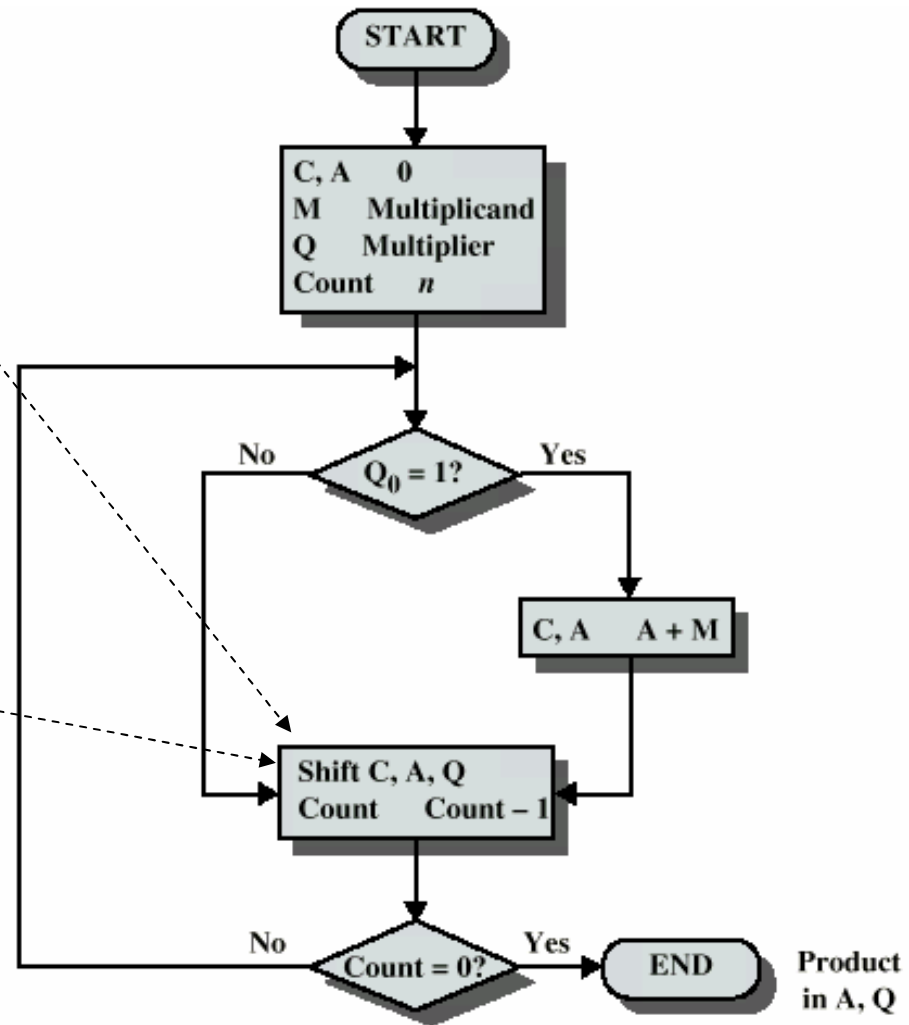


Aritmética de Números Inteiros

- Então, todos os bits dos registradores C , A e Q são deslocados um bit para a direita ($A_{n-1} = C$; $Q_{n-1} = A_0$ e Q_0 eliminado)

- Se Q_0 for 0:

- Nenhuma adição é efetuada, sendo feito apenas o deslocamento dos bits;



Aritmética de Números Inteiros

- Vejamos a aplicação da solução na multiplicação de 13 (Registrador Q) x 11 (Registrador M) :

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	} First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	} Second Cycle
0	1101	1111	1011	Add	
0	0110	1111	1011	Shift	} Third Cycle
1	0001	1111	1011	Add	
0	1000	1111	1011	Shift	} Fourth Cycle

Aritmética de Números Inteiros

- Infelizmente, o esquema anterior não funciona para multiplicação com sinal, vejamos uma analogia:

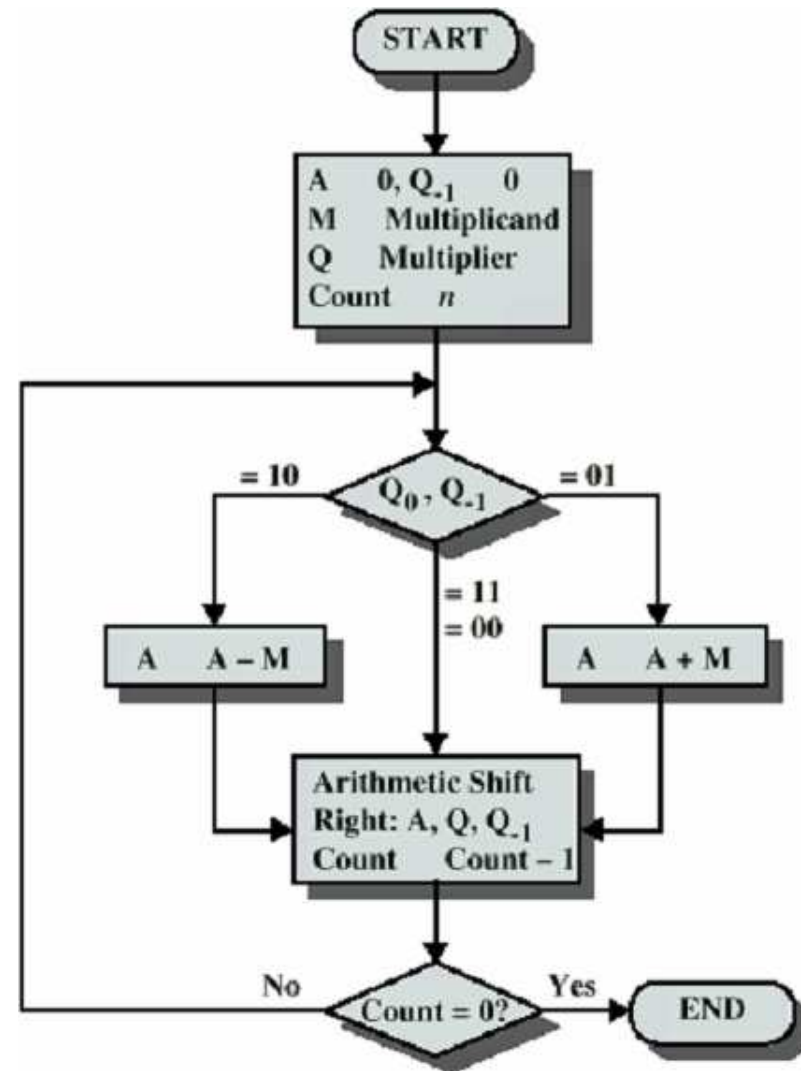
$$\begin{array}{rcccccc} & & & & 1 & 1 & 1 & 0 & \text{Multiplicando (14)} \\ & & & & \times & 1 & 0 & 1 & 0 & \text{Multiplicador (10)} \\ & & & \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & \text{Produto (140)} \end{array}$$

$$\begin{array}{rcccccc} & & & & 1 & 1 & 1 & 0 & \text{Multiplicando (-2)} \\ & & & & \times & 1 & 0 & 1 & 0 & \text{Multiplicador (-6)} \\ & & & \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & \text{Produto (-116)} \end{array}$$

- Existem diversas soluções possíveis para esse e outros dilemas gerados na multiplicação com sinal. Um dos algoritmos mais usados é o de Booth;

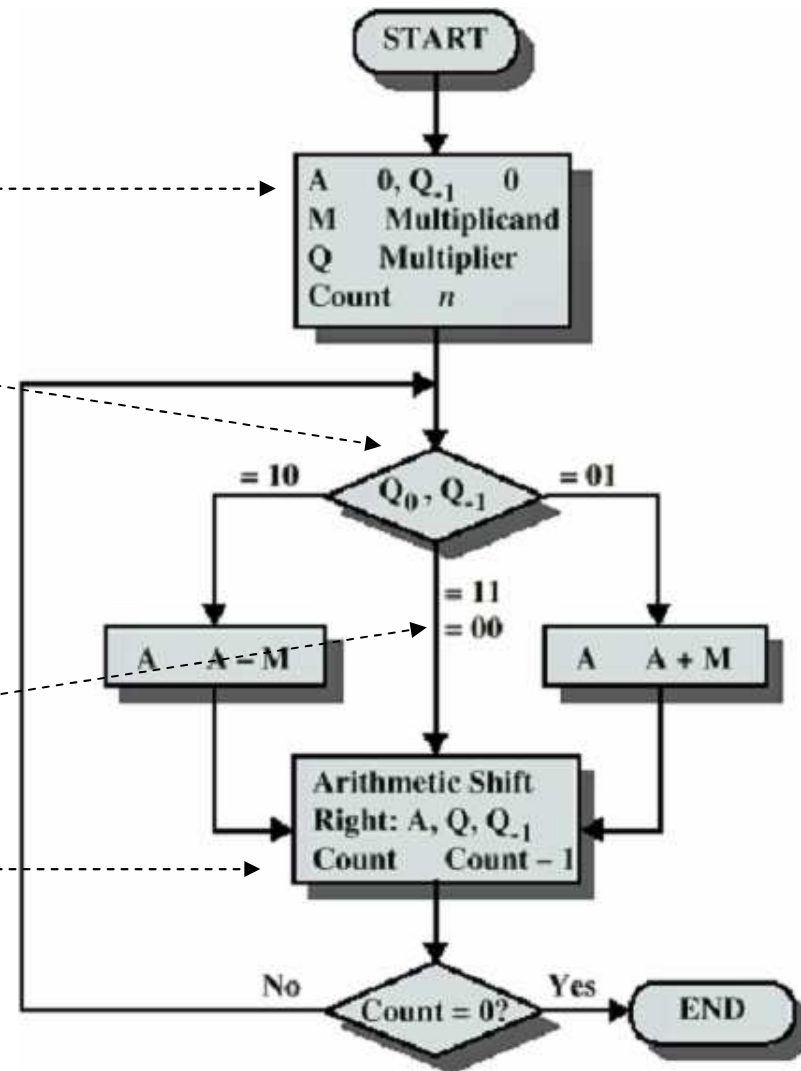
Aritmética de Números Inteiros

- Como antes, multiplicador e multiplicando são armazenados nos registradores Q e M , respectivamente.
- O registrador de 1 bit é posicionado logicamente à direita do bit menos significativo (Q_0) do registrador Q e designado como Q_{-1} ;



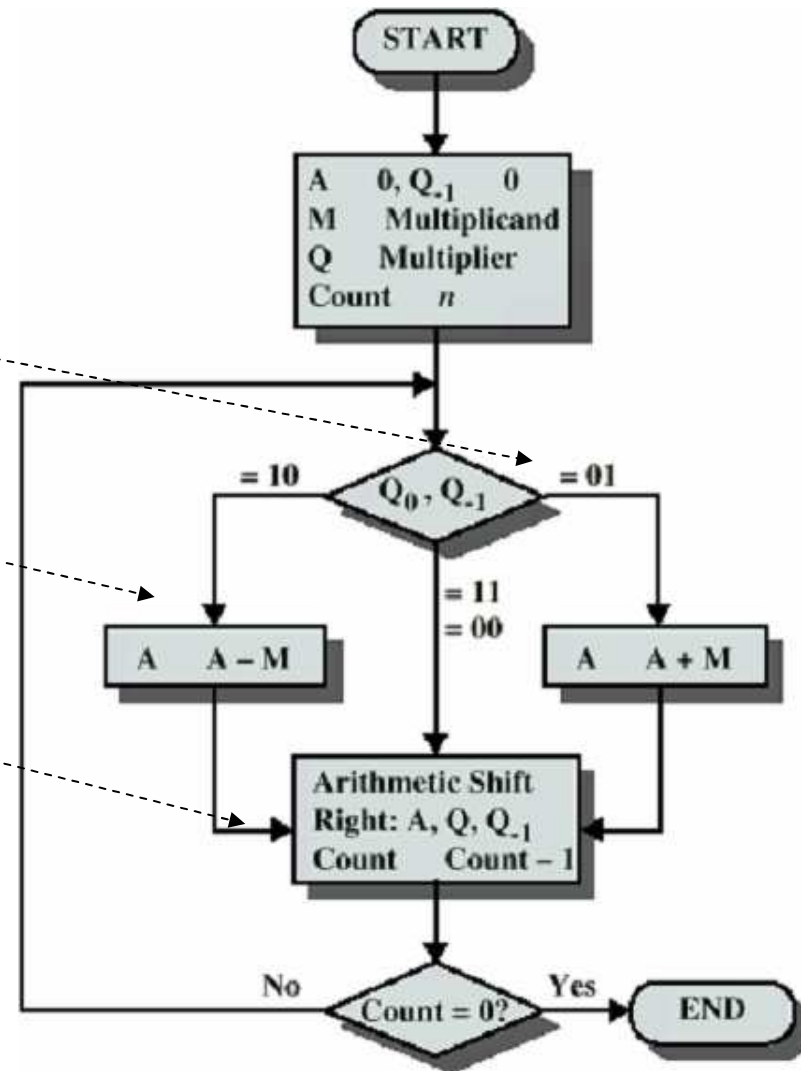
Aritmética de Números Inteiros

- A e Q_{-1} são inicializados com zero.
- Ao analisar cada bit do multiplicador, o bit à sua direita Q_{-1} também é analisado;
- Se esse dois bits forem iguais (1-1 ou 0-0)
 - Então todos os bits dos registradores A , Q e Q_{-1} são deslocados 1 bit para a direita;



Aritmética de Números Inteiros

- Se esse dois bits forem diferentes:
 - O multiplicando será somado (01) do registrador A;
 - ou subtraído (10) do registrador A;
- Em seguida, ocorre o deslocamento de um bit para a direita, porém:
 - O bit mais a esquerda de A (A_{n-1}), é deslocado para A_{n-2} mas também permanece em A_{n-1} ;



Aritmética de Números Inteiros

- Vejamos do **algoritmo de Booth** na multiplicação de 3 (Registrador Q) x 7 (Registrador M) :

A	Q	Q ₋₁	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A	A - M } First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	} Second Cycle
0101	0100	1	0111	A	
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	} Fourth Cycle

Resultado armazenado nos registradores A e Q

Aritmética de Números Inteiros

- A **divisão** é um pouco mais complicada que a multiplicação, vejamos um simples exemplo de números binários sem sinal:

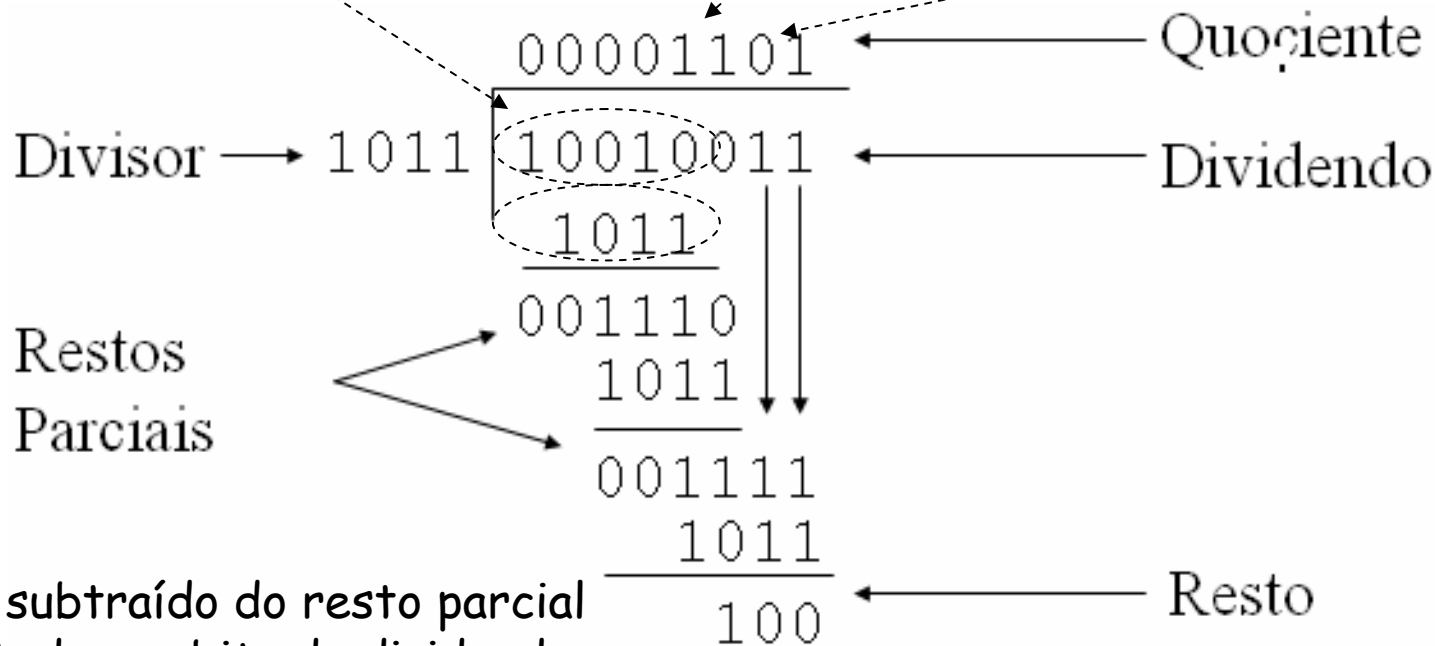
$$\begin{array}{r} \text{Quoçiente} \leftarrow 00001101 \\ \text{Divisor} \rightarrow 1011 \overline{) 10010011} \leftarrow \text{Dividendo} \\ \underline{1011} \\ 001110 \\ \underline{1011} \\ 001111 \\ \underline{1011} \\ 100 \leftarrow \text{Resto} \end{array}$$

Restos Parciais

Aritmética de Números Inteiros

Bits analisados, da esquerda para a direita, até o número de bits seja igual ou maior que o divisor.

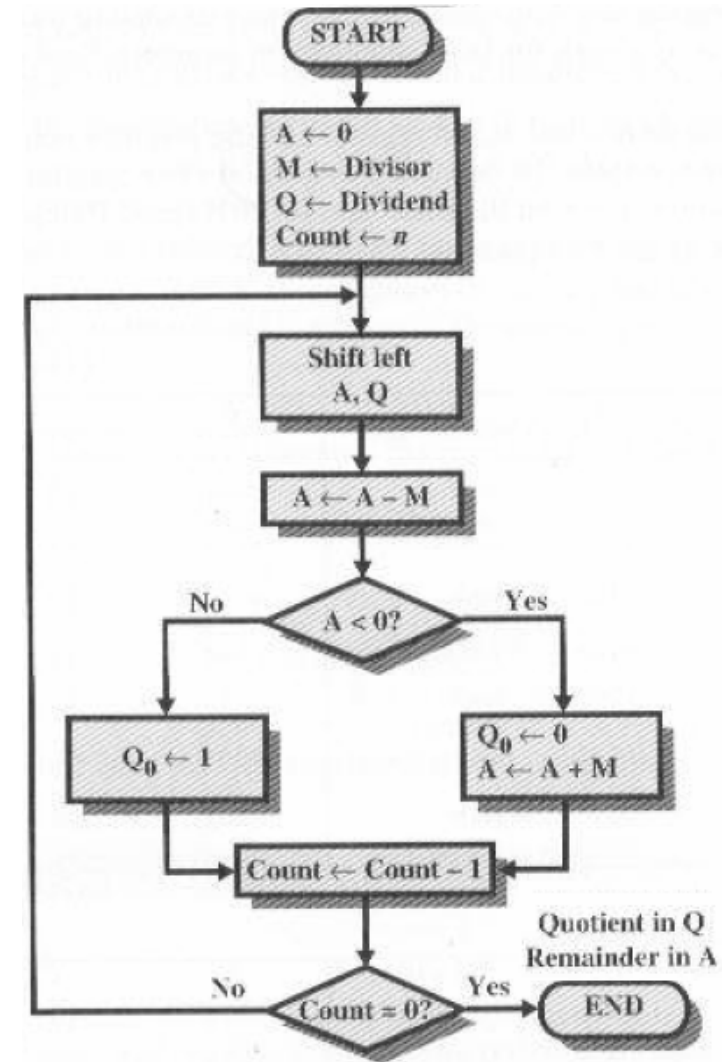
Quando esse número é encontrado é colocado 1 no quociente. Caso contrário, coloca 0 no quociente



Divisor é subtraído do resto parcial até que todos os bits do dividendo seja tenham sido analisados.

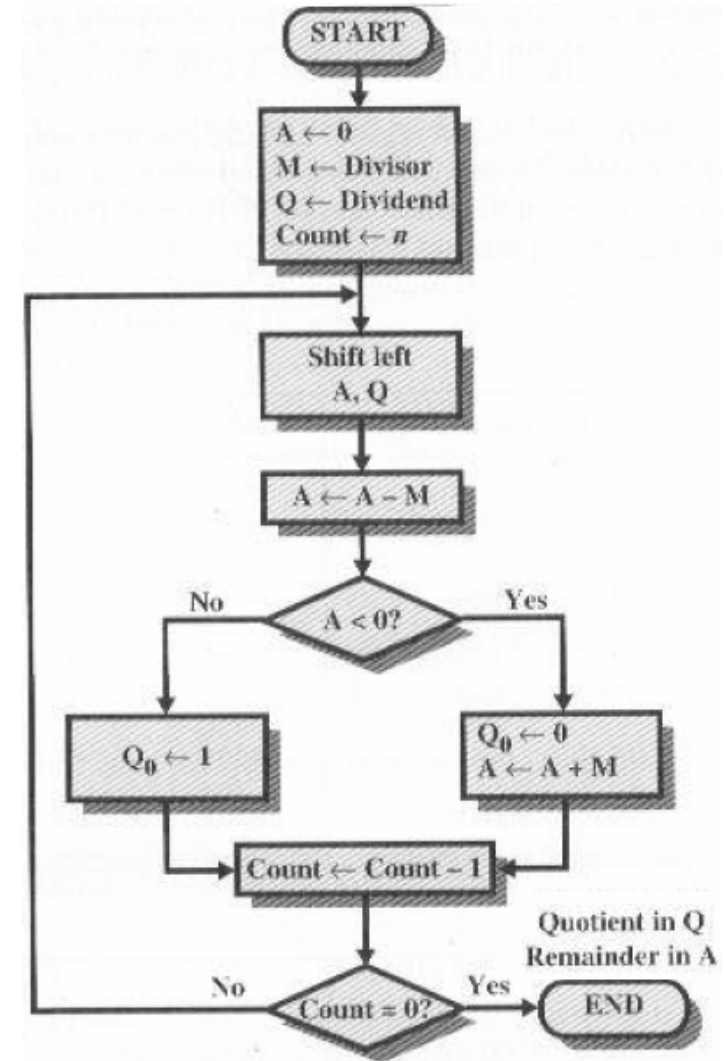
Aritmética de Números Inteiros

- Vejamos um algoritmo de máquina para o processo de divisão usando o complemento de 2:
 - Passo 1:
 - Divisor, no registrador M;
 - Dividendo, nos registradores A e Q;
 - O dividendo deve ser expresso como um número em complemento de dois com $2n$ bits. Ex: 1001, como 11111001;



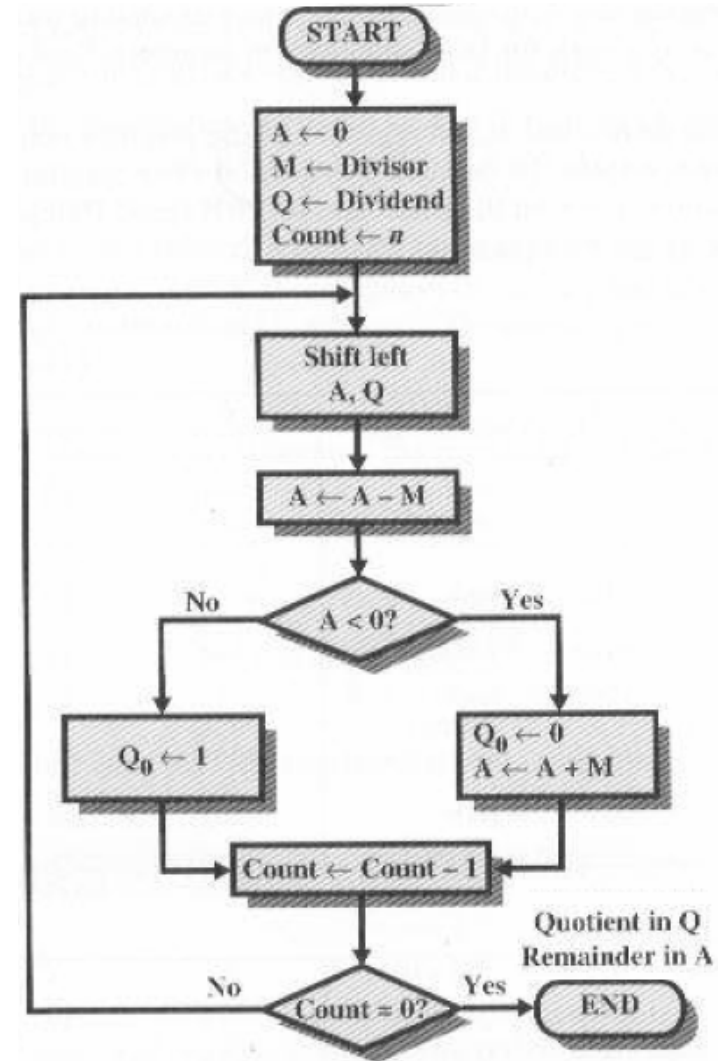
Aritmética de Números Inteiros

- ❑ **Passo 2:** Deslocar o conteúdo dos registradores A e Q , juntos, um bit para a esquerda;
- ❑ **Passo 3:** Se M e A têm o mesmo sinal, fazer $A = A - M$; caso contrário, $A = A + M$;
- ❑ **Passo 4:** Se o sinal de A for o mesmo, antes e depois da operação, então, operação bem sucedida;



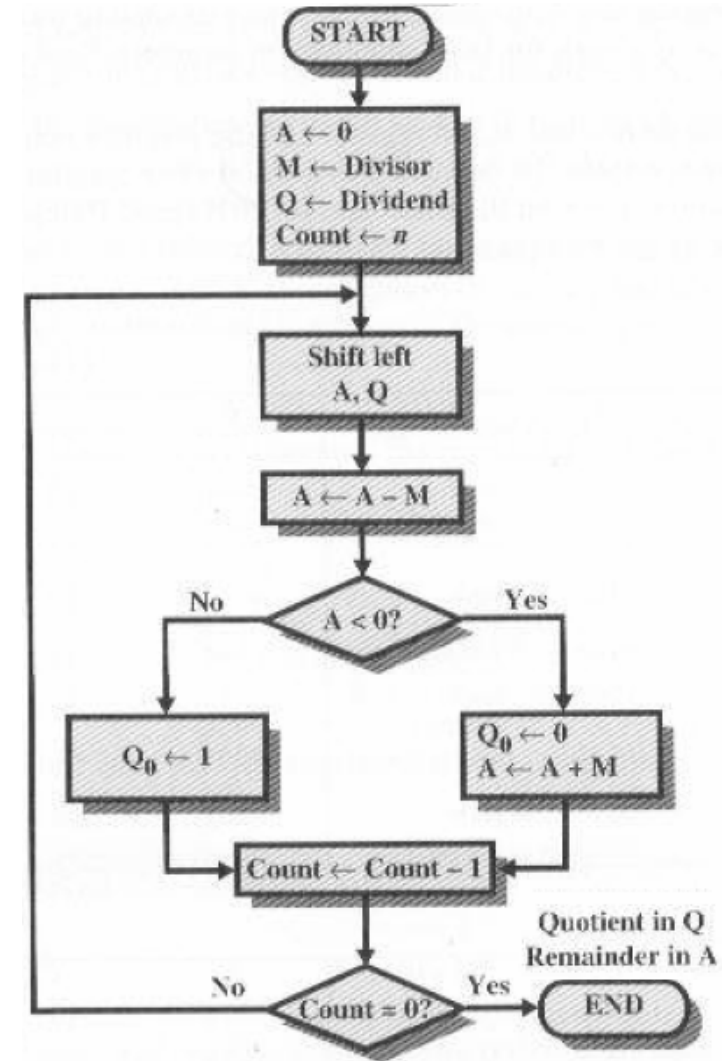
Aritmética de Números Inteiros

- ❑ **Passo 4.a:** Se a operação for bem-sucedida ou se ($A = 0$ e $Q = 0$), então faça $Q_0 = 1$;
- ❑ **Passo 4.b:** Se a operação não for bem sucedida e se ($A = 0$ ou $Q = 0$), então faça $Q_0 = 0$ e restaure o valor antigo de A (somando M a A)
- ❑ **Passo 5:** Repetir os passos 2 e 4 enquanto houver bits a examinar em Q ;



Aritmética de Números Inteiros

- ❑ Passo 6: Ao final, o resto estará em A .
- ❑ Se o divisor e o dividendo tiverem o mesmo sinal, o quociente estará em Q ;
- ❑ Caso contrário, o quociente correto é o complemento de dois no número armazenado em Q ;



Aritmética de Números Inteiros

- Vejamos a aplicação do algoritmo em um exemplo: $(7) \div (3)$

A	Q	M=0011
0000	0111	Valor inicial
0000	1110	Deslocar
1101		Subtrair
0000	1110	Restaurar
0001	1100	Deslocar
1110		Subtrair
0001	1100	Restaurar
0011	1000	Deslocar
0000		Subtrair
0000	1001	Fazer $Q_0 = 1$
0001	0010	Deslocar
1110		Subtrair
0001	0010	Restaurar

Representação de Números de Ponto Flutuante

- A notação de ponto fixo **não possibilita** representar **números muito grandes** nem **frações muito pequenas**.
- Além disso, em uma divisão de dois números muito grandes, a parte fracionária do quociente pode ser perdida;
- Essas limitações são superadas com a representação de ponto flutuante;

Representação de Números de Ponto Flutuante

- Um número pode ser representado da seguinte forma:

$$\pm M \times B^{\pm E}$$

Onde:

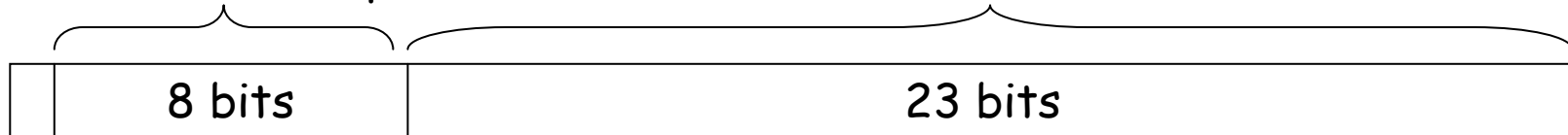
- Sinal: mais ou menos
- Mantissa: M
- Expoente: E
- Base: B (Implícita e não precisa ser armazenada porque é a mesma para todos os número)

Representação de Números de Ponto Flutuante

- A figura abaixo mostra a representação de números binários de ponto flutuante:

Expoente Polarizado. Um valor fixo é subtraído ao valor desse campo de 8 bits para se obter o verdadeiro valor do expoente

Mantissa. Armazena o número normalizado, ou seja, $\pm 0,1 \text{ bbb...b} \times 2^{\pm E}$



Sinal da Mantissa
(0 = Positivo; 1 = Negativo)

Representação de Números de Ponto Flutuante

- Vejamos os exemplos a seguir:

$$\begin{aligned} 0,11010001 \times 2^{10100} &= 0 \ 10010011 \ 101000100000000000000000000000 \\ -0,11010001 \times 2^{10100} &= 1 \ 10010011 \ 101000100000000000000000000000 \\ 0,11010001 \times 2^{-10100} &= 0 \ 01101011 \ 101000100000000000000000000000 \\ -0,11010001 \times 2^{-10100} &= 1 \ 01101011 \ 101000100000000000000000000000 \end{aligned}$$

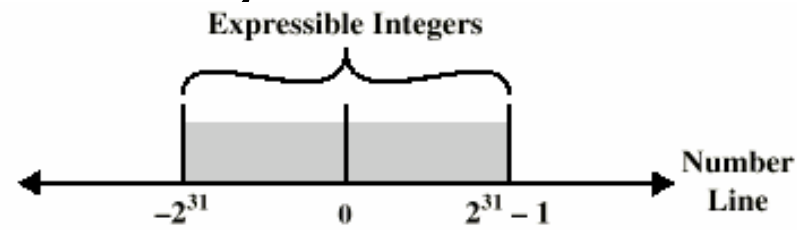
- O sinal é armazenado no primeiro bit da palavra;
- O primeiro bit da mantissa é sempre q (após normalização);
- O valor 127 é adicionado ao expoente verdadeiro e armazenado no campo do expoente;
- A base é 2.

Representação de Números de Ponto Flutuante

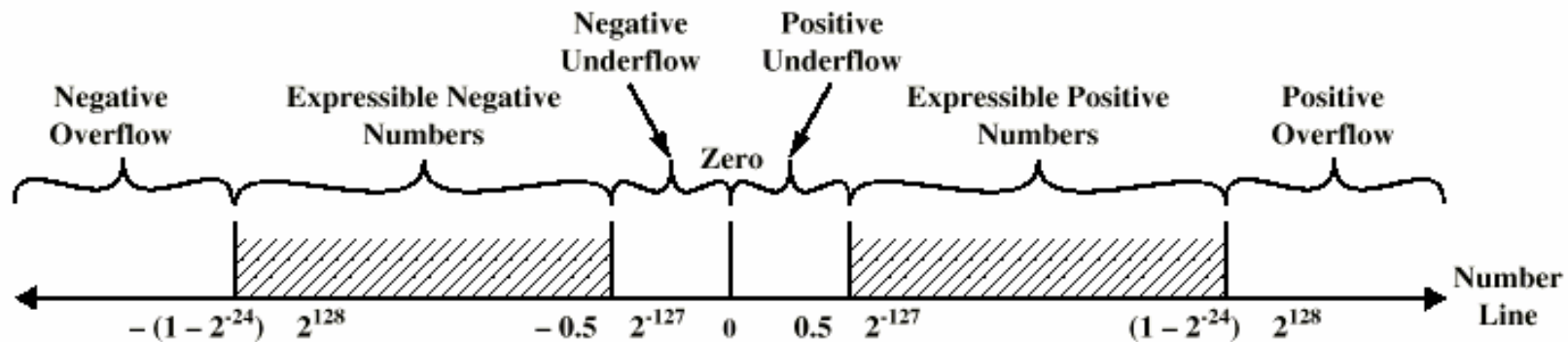
- Na notação em complemento de dois, podem ser representados todos os números inteiros de -231 a 231-1 (total de 232 números distintos);
- No formato de ponto flutuante, números nas seguintes faixas podem ser representados:
 - Número negativos entre $-(1 - 2^{-24}) \times 2^{128}$ e $-0,5 \times 2^{-127}$
 - Número positivos entre $0,5 \times 2^{-127}$ e $(1 - 2^{-24}) \times 2^{128}$

Representação de Números de Ponto Flutuante

- Vejamos a representação dos limites de representação na figura abaixo:



(a) Twos Complement Integers



(b) Floating-Point Numbers

Representação de Números de Ponto Flutuante

- Cinco regiões na reta de números não estão incluídas nessas faixas:
 - **Overflow em número negativos:** números negativos menores que $-(1 - 2^{-24}) \times 2^{128}$
 - **Underflow em números negativos:** números negativos maiores que $-0,5 \times 2^{-127}$;
 - **Zero.**
 - **Underflow em números positivos:** números positivos menores que $0,5 \times 2^{-127}$;
 - **Overflow em números positivos:** números positivos maiores que $(1 - 2^{-24}) \times 2^{128}$;

Representação de Números de Ponto Flutuante

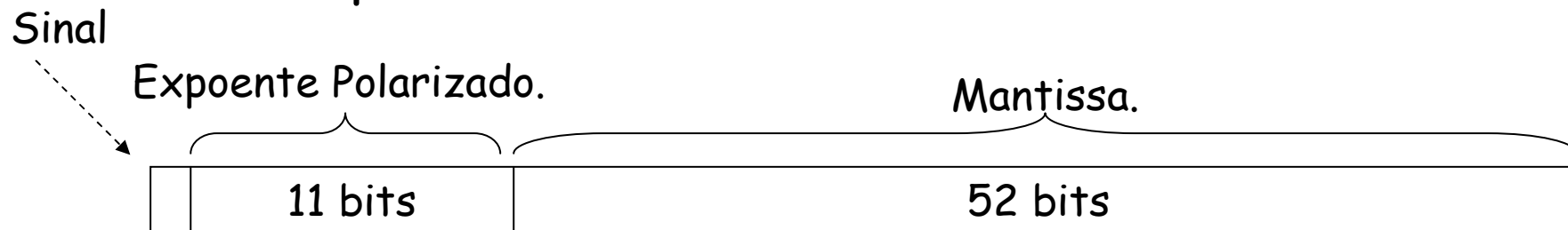
- A representação de ponto flutuante terá que prever um código especial para o 0 (zero);
- Ocorre **overflow** quando a magnitude do resultado é maior que o maior valor que pode ser expresso com expoente igual a 128 (por exemplo, $2^{120} \times 2^{100} = 2^{220}$);
- Ocorre **underflow** quando a magnitude é muito pequena (por exemplo, $2^{-120} \times 2^{-100} = 2^{-220}$);

Representação de Números de Ponto Flutuante

- Analisemos também que os números não são distribuídos igualmente ao longo da reta de números:
 - Maior quantidade de valores representáveis próximo à origem;
 - Quantidade diminui com a distância à origem;
- Se aumentarmos o número de bits do expoente, expandimos a faixa de valores representáveis. Porém teremos uma menor precisão;
- Os cálculos em ponto flutuante, tendem a serem arredondados para os valores mais próximos que a notação possibilitar;

Padrão IEEE para representação de números binários de ponto flutuante

- A mais importante representação de ponto flutuante é definida no padrão IEE 754 de 1985;
- O padrão define um formato simples de 32 bits e um formato duplo de 64 bits com expoentes de 8 e 11 bits respectivamente;



- Diminui overflow em operações intermediárias;
- Diminui erros por arredondamento;

Padrão IEEE para representação de números binários de ponto flutuante

- Alguns bits e combinações de bits são interpretados de forma especial no formato IEEE, vejamos a tabela:

	Single Precision (32 bits)				Double Precision (64 bits)			
	Sign	Biased exponent	Fraction	Value	Sign	Biased exponent	Fraction	Value
Positive zero	0	0	0	0	0	0	0	0
Negative zero	1	0	0	0	1	0	0	-0
Plus infinity	0	255 (all 1s)	0	∞	0	2047 (all 1s)	0	∞
Minus infinity	1	255 (all 1s)	0	$-\infty$	1	2047 (all 1s)	0	$-\infty$
Quiet NaN	0 or 1	255 (all 1s)	$\neq 0$	NaN	0 or 1	2047 (all 1s)	$\neq 0$	NaN
Signaling NaN	0 or 1	255 (all 1s)	$\neq 0$	NaN	0 or 1	2047 (all 1s)	$\neq 0$	NaN
Positive normalized nonzero	0	$0 < e < 255$	f	$2^{e-127}(1.f)$	0	$0 < e < 2047$	f	$2^{e-1023}(1.f)$
Negative normalized nonzero	1	$0 < e < 255$	f	$-2^{e-127}(1.f)$	1	$0 < e < 2047$	f	$-2^{e-1023}(1.f)$
Positive denormalized	0	0	$f \neq 0$	$2^{-126}(0.f)$	0	0	$f \neq 0$	$2e^{-1022}(0.f)$
Negative denormalized	1	0	$f \neq 0$	$-2^{-126}(0.f)$	1	0	$f \neq 0$	$-2e^{-1022}(0.f)$

Padrão IEEE para representação de números binários de ponto flutuante

■ Considerações:

- O expoente 0 junto com uma fração 0 representa 0;
- Expoente totalmente preenchido com uns junto com uma fração igual a zero representa infinito, positivo ou negativo, (dependo do bit de sinal).
- Expoente 0 junto com uma fração diferente de 0 representa um número não-normalizado;
- Um expoente totalmente com uns junto com uma fração diferente de zero representa o valor NaN (Not a Number).

Aritmética de Números de Ponto Flutuante

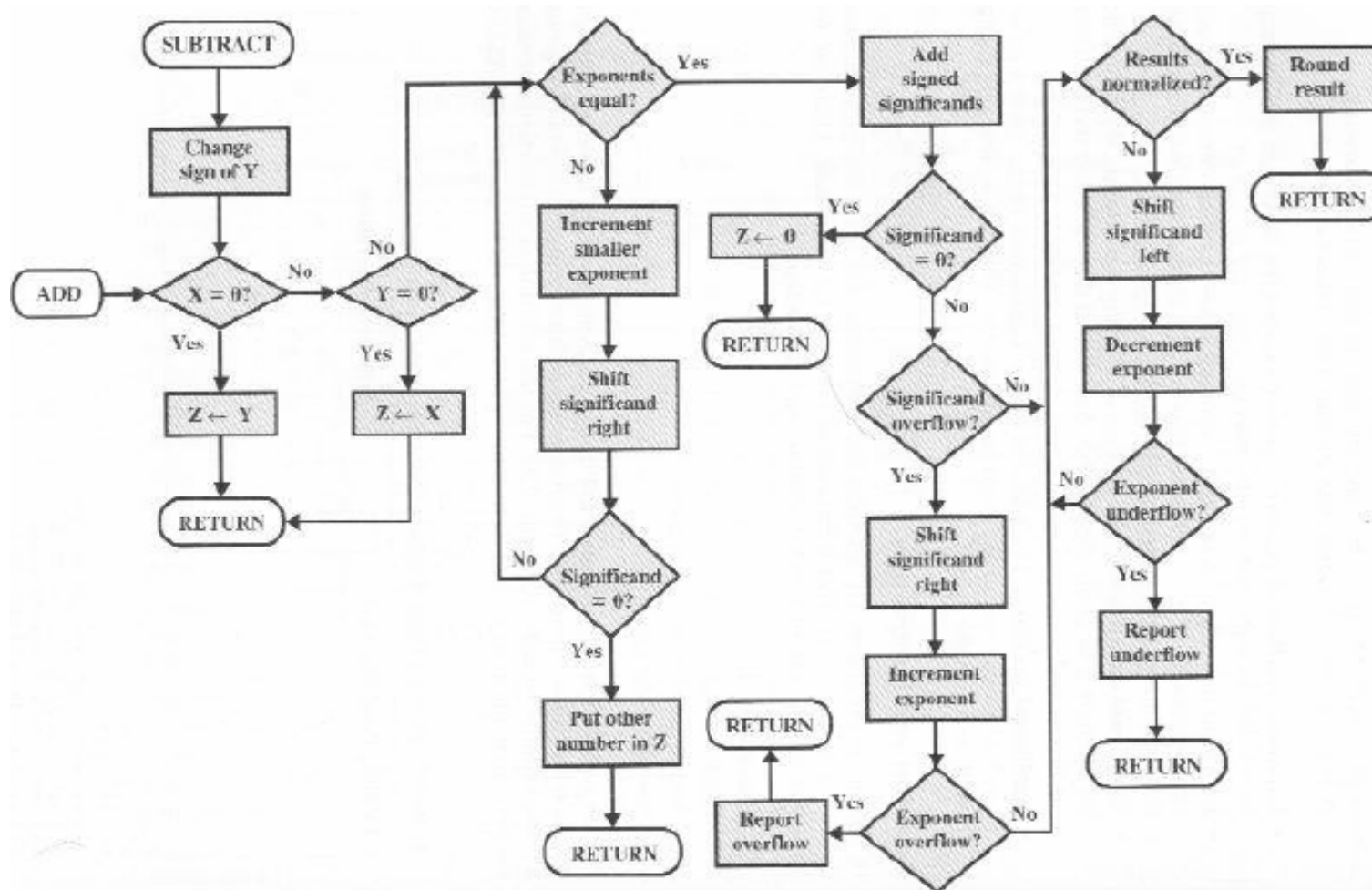
- Na aritmética de ponto flutuante, a adição e a subtração são operações mais complexas que a multiplicação e a divisão;
- Isso ocorre devido à necessidade de alinhar os operandos, tornando-os iguais.
- O algoritmo tem as seguintes fases:
 - Verificar se algum operando é zero;
 - Alinhar as mantissas;
 - Adicionar ou subtrair as mantissas;
 - Normalizar o resultado.

Aritmética de Números de Ponto Flutuante

$$\begin{aligned}(987 \times 10^0) + (654 \times 10^{-2}) &= \\(987 \times 10^0) + (6,54 \times 10^0) &= \\6454,98 \times 10^0 &= \end{aligned}$$

- ❑ Em caso de subtração, mudar o sinal do subtraendo;
- ❑ Igualar os expoentes e deslocar os bits para efetuar a soma (alinhamento dos números, por exemplo: 6+7);
- ❑ Somar as duas mantissas
- ❑ Normalizar o resultado. A normalização consiste em deslocar os dígitos da mantissa para a esquerda, até que o dígito mais significativo seja diferente de 0;
- ❑ Resultado ainda pode ser arredondado;

Aritmética de Números de Ponto Flutuante

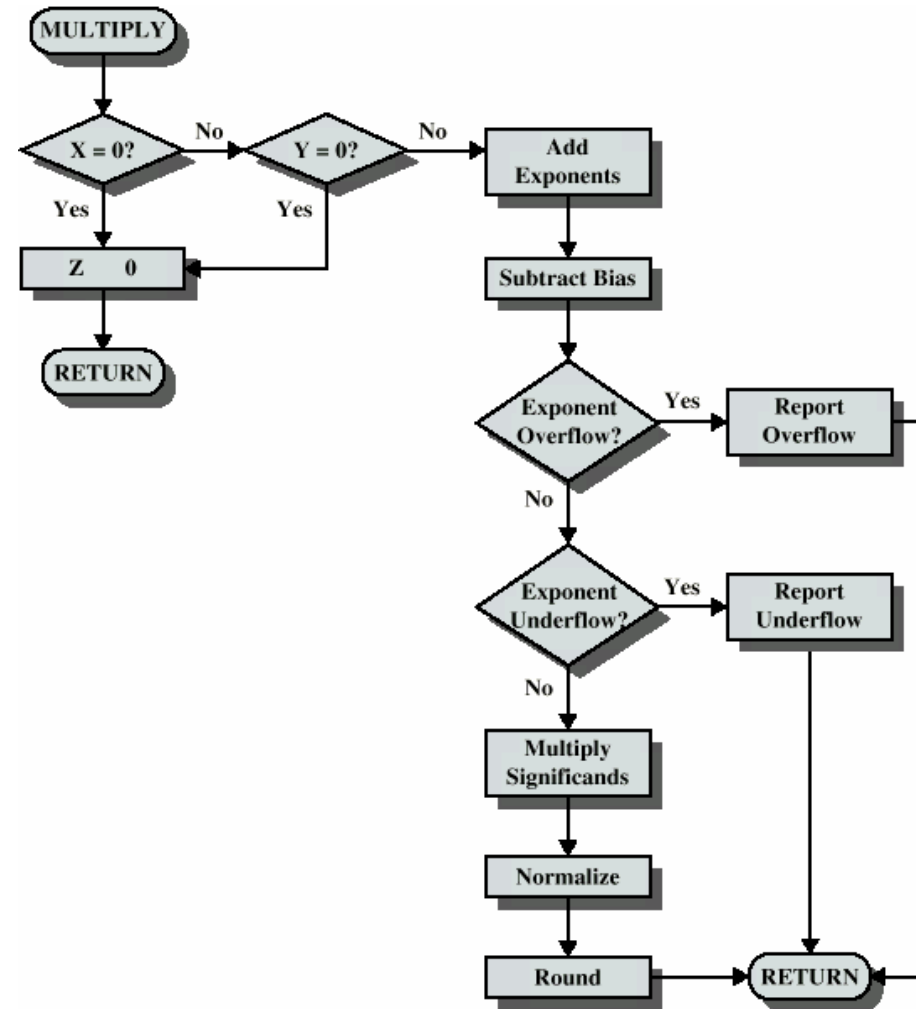


Aritmética de Números de Ponto Flutuante

- A multiplicação é mais simples do que a adição e a subtração:
- Considerações:
 - Caso um dos operadores é 0, o resultados será 0.
 - Soma-se os expoentes. Se eles forem armazenados na forma polarizada, a soma dobrará a polarização;
 - Portanto, o valor da polarização deve ser subtraído da soma dos expoentes;
 - Se o expoente do produto estiver dentro da faixa de números representáveis, as mantissas devem ser multiplicadas (mesma forma dos números inteiros), levando em conta seus sinais;

Aritmética de Números de Ponto Flutuante

- Vejamos o fluxograma da multiplicação de números de ponto flutuante:
 - $z = x \cdot y$



Aritmética de Números de Ponto Flutuante

- Vejamos os passos para efetuarmos uma divisão bem sucedida com ponto flutuante:
 - Caso o divisor seja 0, erro ou representação de infinito;
 - Dividendo 0, resultado 0;
 - Caso não ocorra nenhum desses casos, o expoente do divisor será subtraído do expoente do dividendo;
 - Em seguida, são efetuados testes de overflow e underflow no expoente;
 - O passo seguinte é dividir as mantissas;
 - Por fim a normalização e o arredondamento;

Aritmética de Números de Ponto Flutuante

- Diversas técnicas para arredondamento foram exploradas, De fato, o padrão IEEE relaciona quatro abordagens alternativas:
 - Arredondamento para o mais próximo: arredondado para o número representável mais próximo;
 - Arredondamento para cima ($+\infty$): arredondado na direção do infinito positivo;
 - Arredondamento para baixo ($-\infty$): arredondado na direção do infinito negativo;
 - Arredondamento para 0: arredondado na direção de zero;

Aritmética de Números de Ponto Flutuante

- O padrão IEEE traz também procedimentos específicos para que a aritmética de pnto produza resultados uniformes, previsíveis e independente de plataformas:
 - Infinito:
 $5 + (+\infty) = +\infty$ $5 - (-\infty) = -\infty$ $5 \cdot (-\infty) = -\infty$
 - NaN silencioso e NaN sinalizador: se propaga sem gerar exceção e gera exceção de operação inválida, respectivamente
 - Números não-normalizados: trata casos de underflow de expoente;

Bibliografia

- Stallings, W. *Arquitetura e Organização de Computadores*, Pearson Hall, 5 ed. SP: 2002.