Eletrônica Digital

ULA- Unidade Lógica Aritmética.

Prof. Rômulo Calado Pantaleão Camara

Carga Horária: 60h

Sumário

- ✓ Unidade Lógica Aritmetrica
  - Registradores
  - Unidade Lógica
  - Operações da ULA
  - Unidade de Ponto Flutuante
  - Representação de números inteiros
  - Representação de números ponto flutuante

Registradores

- ✓ O processador contém elementos de memória de pequena capacidade mas de alta velocidade, usados para armazenar resultados temporários, chamados de registradores.
- ✓ O conjunto desses registradores é denominado banco de registradores.
- ✓ Esses registradores são referenciados explicitamente pelas instruções lógicas, aritméticas e de transferência de dados.
- ✓ Existe um registrador especial denominado contador de programa - PC, que contém o endereço da próxima instrução a ser executada.
- ✓ Um outro registrador, chamado de registrador de instrução - IR, contém a instrução que está sendo executada.

Ciclo de Busca das Instruções

✓ O processador executa uma instrução em uma série de etapas:

- 1. Busca a próxima instrução que está localizada na memória para o registrador de instrução;
- 2. Atualiza o apontador de instruções (PC) para que ele aponte para a próxima instrução a ser executada.
- 3. Determina o tipo de instrução;
- 4. Se a instrução faz uso de dados (operandos), determina onde estão localizados;
- 5. Busca os operandos, se houver, para os registradores do processador;
- 6. Executa a instrução;
- 7. Armazena os resultados nos locais apropriados;
- 8. Volta ao passo 1 para executar a próxima instrução.

Unidade Lógica e Aritmética

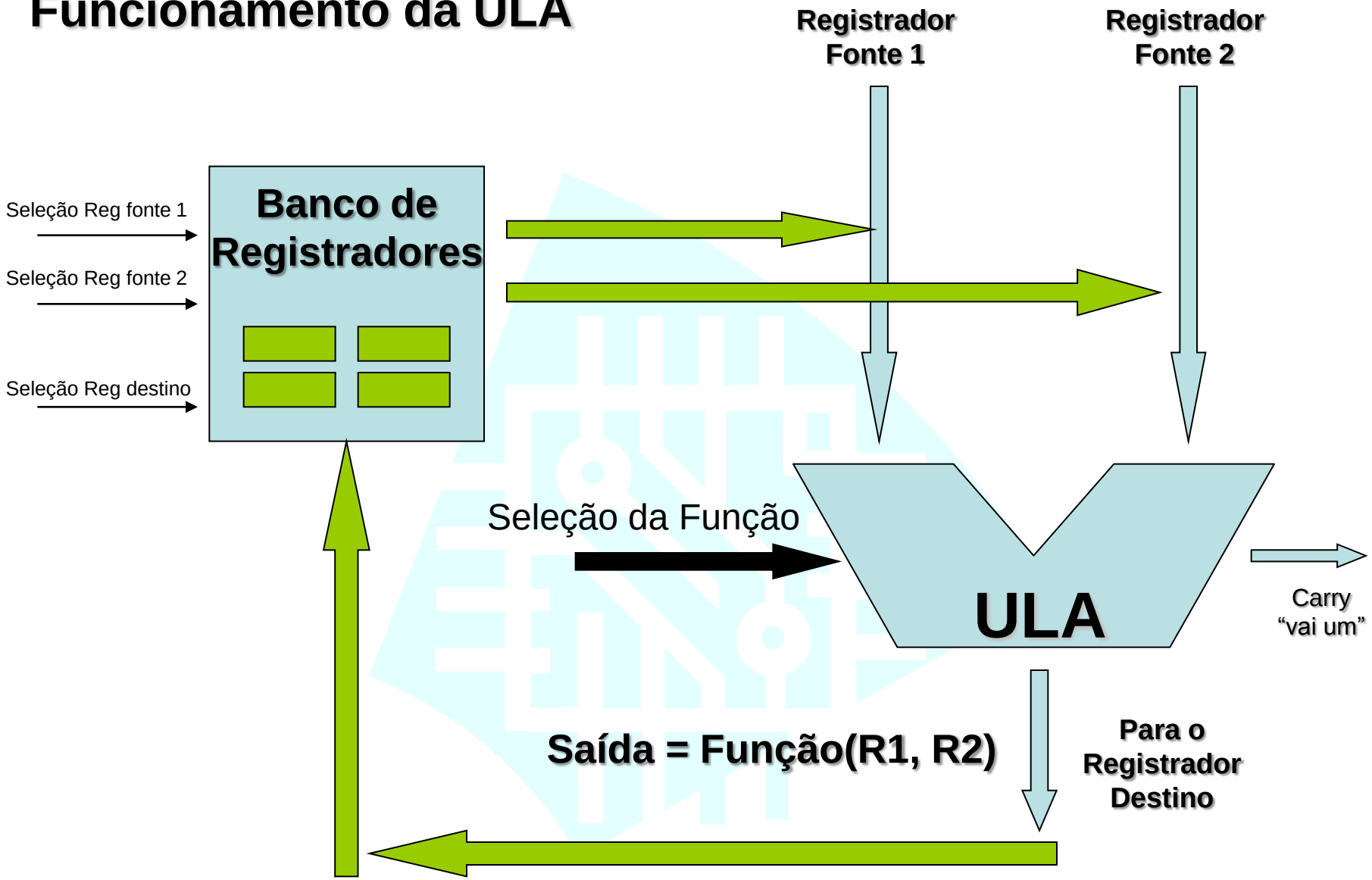
A unidade lógica e aritmética pode realizar diversas operações, entre elas:

- Adição
- Subtração
- Operações lógicas (E, OU, XOR, INVERSÃO)
- Deslocamento (à esquerda e à direita)
- Comparação

As unidades aritméticas e lógicas mais modernas realizam também as operações de multiplicação e divisão.

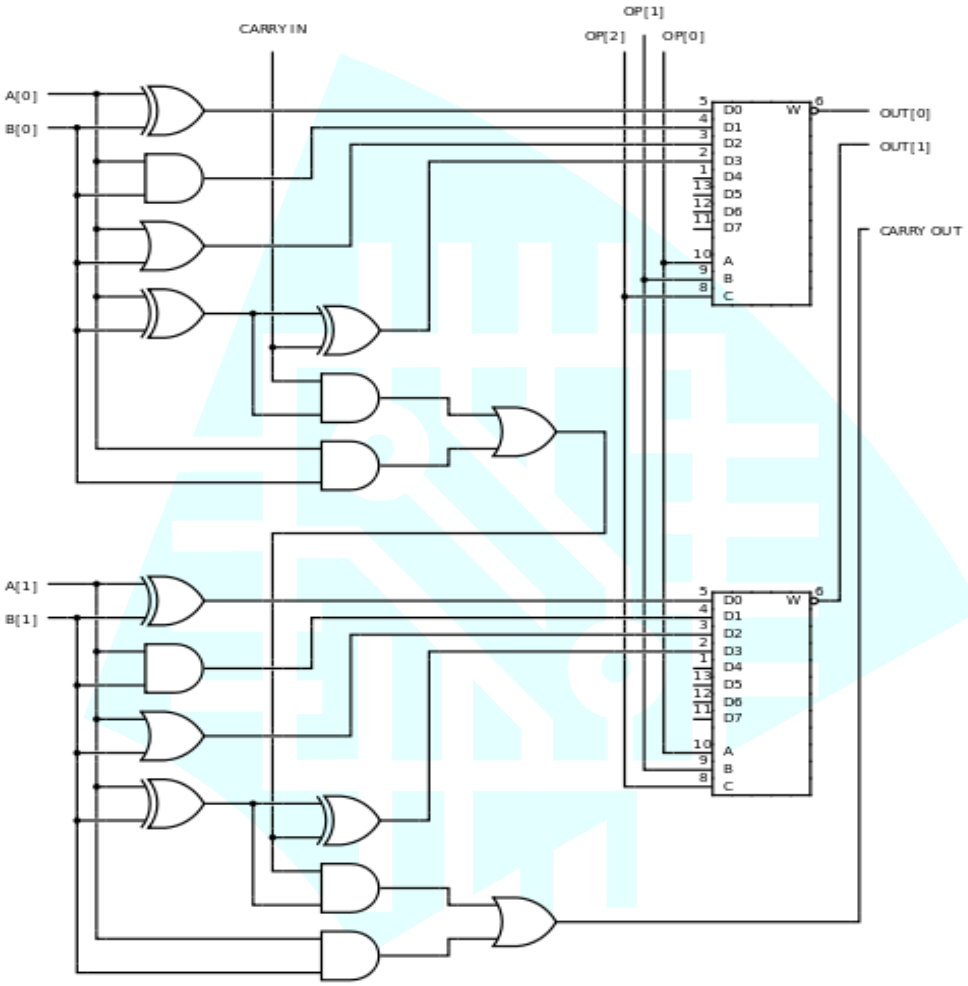
As operações são realizadas pela leitura de dois registradores fontes do banco de registradores, e com a escrita do resultado no registrador de destino.

Funcionamento da ULA



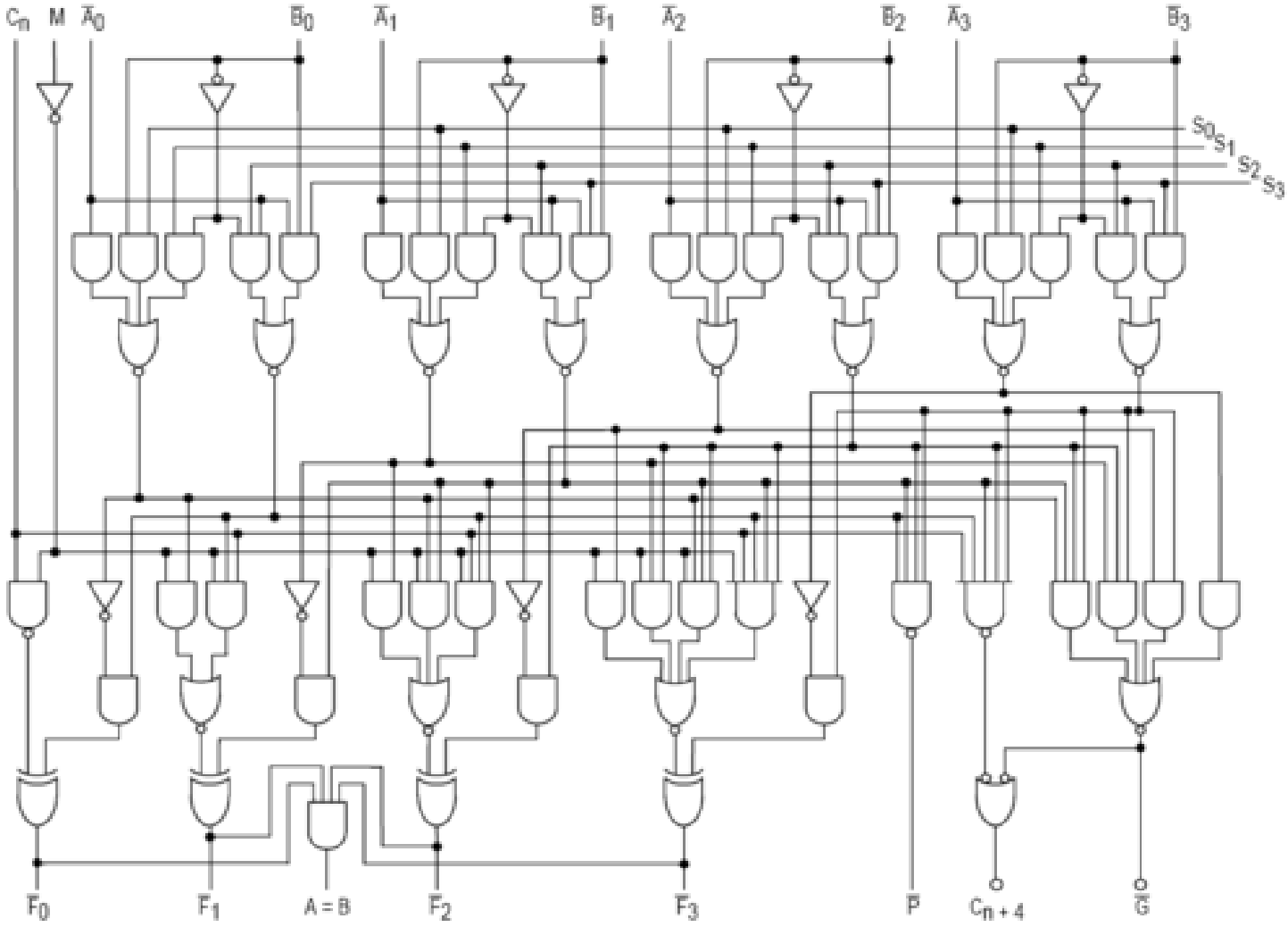
Design da ULA

Ula 2-bits



Design da ULA

Ula 4-bits

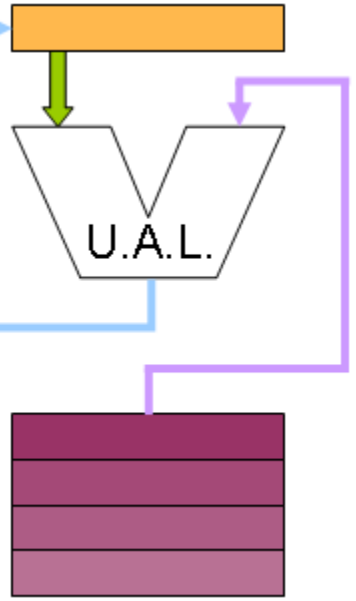


Tipos de Design da ULA

- ✓ Accumulator:
  - Há um registrador especial chamado de Accumulator;
  - O Accumulator guarda todas as operações da ULA;
- ✓ Vantagens:
  - ISA mais simples apenas especificando um operando ao invés de dois operandos e o destino;
  - Mais rápidas;
- ✓ Desvantagens
  - Necessita de software adicional para carregar valores;
  - São difíceis de trabalhar com pipeline.

Tipos de Design da ULA

- ✓ Accumulator:
  - Geralmente esta arquitetura é utilizada em calculadoras de mão.

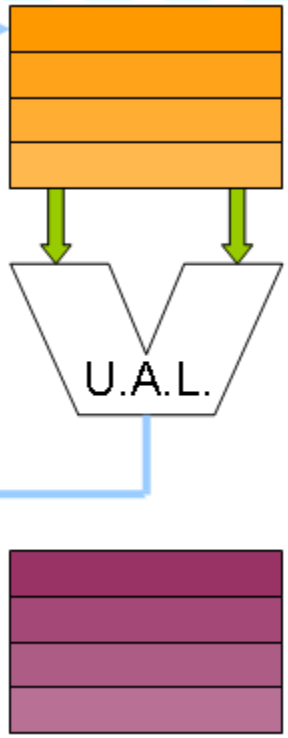


Tipos de Design da ULA

- ✓ Register-to-Register:
  - Uma das mais comuns arquiteturas register-to-register é chamada de máquina de três operadores. (three register operand machine);
- ✓ Desvantagens
  - ISA precisa ser expandida para incluir instruções de "source" e de destino.
  - Requer um longo tamanho de instrução;
  - Requer esforço adicional para informar o registrador que será feito a escrita do resultado; (Write-back)
  - O passo write-back pode gerar problemas na sincronização no processo de pipeline.

Tipos de Design da ULA

- ✓ Vantagem:
  - Pode-se utilizar pipeline;
  - Software mais simples;



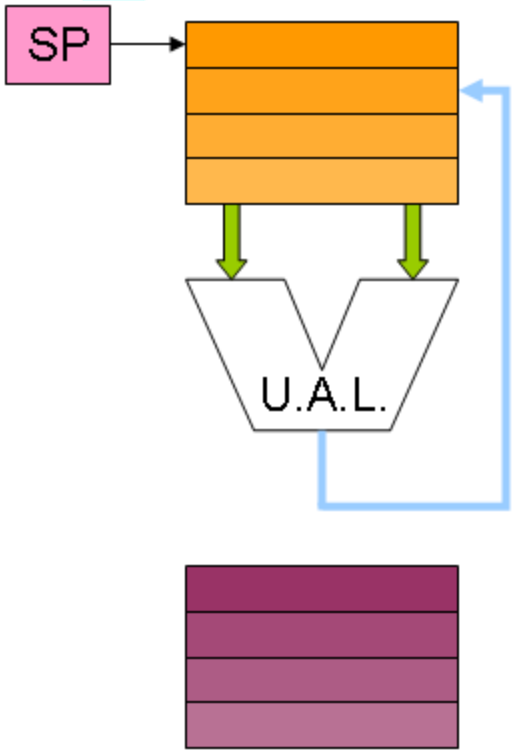
Tipos de Design da ULA

- ✓ Register Stack:
  - Combinação entre a Accumulator e a Register-to-Register;
- ✓ Funcionamento:
  - ALU lê o operando do topo da pilha e o resultado é inserido no topo da pilha;
- ✓ Desvantagens
  - Operações matemáticas complicadas requer decomposição dentro da expressão (Reverse-Polish form;
  - Dificulta a programação caso o compilador não auxilie;

Tipos de Design da ULA: Register Stack

- ✓ Desvantagens
  - Hardware precisa ser criado para implementar o PUSH e POP;
  - Hardware precisa ser criados para a detecção e o tratamento de erros de pilha. (***pushing*** quando a pilha estiver cheia, ou ***popping*** quando a pilha estiver vazia)
- ✓ Vantagens
  - Muito compiladores dão suporte e fazem o reverse-polish facilmente utilizando árvores binárias;
  - ISA muito simples;
- ✓ Essa máquina é chamada de "0-operand" ou máquina de endereço zero, porque não precisa especificar o local da instrução.

Tipos de Design da ULA

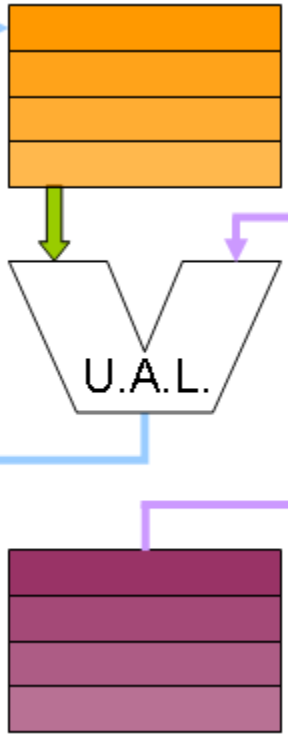


Tipos de Design da ULA

- ✓ Register-and-Memory
  - Um operando vem do registrador e o outro vem da memória externa;
- ✓ Desvantagens:
  - ISA complicada por causa da busca na memória. Deverá trazer toda a palavra localizada no endereço de memória.
- ✓ Não é utilizado diretamente: é integrada com outro esquema register-to-register;
- ✓ Utilizados com ISA CISC;

Tipos de Design da ULA

- ✓ Register-and-Memory



Exercício

- ✓ Pesquise outros tipos de design de ULA e especifique 4 outras arquiteturas. (Valor 0,5 extra na primeira nota.)
- ✓ Deve ser especificado a arquitetura, vantagens, desvantagens e funcionamento.

Algumas operações da ULA

- ✓ NOT – Inversão
- ✓ AND – E lógico
- ✓ OR – OU lógico
- ✓ XOR – OU exclusivo
- ✓ Shift
- ✓ +, -, *, /
- ✓ =, <, >, etc

Tabela Verdade

A	B	AND	OR	NOT A	XOR
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Representação de grandeza com sinal

- ✓ O bit mais significativo representa o sinal:
  - **0** (indica um **número positivo**)
  - **1** (indica um **número negativo**)
- ✓ Os demais bits representam a **grandeza (magnitude)**.

sinal	magnitude

- ✓ O valor dos bits usados para representar a magnitude independe do sinal (sendo o número positivo ou negativo, a representação binária da magnitude será a mesma).

Exemplos: (8 bits)

- ❑ $00101001_2 = +41_{10}$
- ❑ $10101001_2 = -41_{10}$

Exemplo

Valor decimal	Valor binário com 8 bits (7 + bit de sinal)
+9	00001001
-9	10001001
+127	01111111
-127	11111111

Assim, uma representação em binário com **n** bits teria disponível para a representação do número ***n-1*** bits (o bit mais significativo representa o sinal).

Representação de grandeza com sinal

- ✓ Apresenta uma grande **desvantagem**: ela exige um grande número de testes para se realizar uma simples soma de dois números inteiros.

- ✓ Requer que na ULA existam dois circuitos distintos para a adição e a subtração.

- ✓ Existem **duas representações** para o zero.

Representação Complemento de 2

- **Representação de números inteiros positivos**
  - igual à representação de grandeza com sinal.
- **Representação de números inteiros negativos**
  - mantém-se os bits menos significativos da direita para a esquerda até à ocorrência do primeiro bit igual a 1 (inclusive), sendo os bits restantes complementados de 1.
  - Esta operação equivale a realizar: complemento de 1 + 1.

Exemplo : (8 bits)

$00001100_2 = +12_{10}$

$11110100_{c2} = -12_{10}$

Exemplo : (8 bits)

$00101001_2 = +41_{10}$

$11010111_{c2} = -41_{10}$

Representação Complemento de 2

- ✓ **Exemplo:** Números inteiros codificados em **binário de 8 bits** em um sistema que utiliza complemento de 2:

(-128, -127, ..., -2. -1, 0, +1, +2,..., +127)

{10000000, 10000001, ..., 11111110, 11111111, 00000000, 00000001, 00000010, ..., 01111111}

- ✓ Bit mais significativo ⇨ informação de sinal

(0 = positivo e 1 = negativo)

Representação Complemento de 2

- Requer **um só circuito** (somador) para fazer a adição e a subtração.
- Há apenas uma representação para o valor **0** (disponibilidade para **mais uma representação)** - mais um número negativo pode ser representado (para 8 bits, pode-se representar o número $-128_{10} \Rightarrow 10000000_2$) .
- A quantidade de números positivos é **diferente** da quantidade de números negativos.

Outras formas de representação

- ✓ **Complemento de 1**: para negar o valor de um número deve-se inverter os bits do sinal (obsoleta) e **Excesso de 2^{m-1}**: representação do número é dada pela soma de seu valor absoluto com 2^{m-1}. Exemplo: Um sistema de 8 bits é chamado de excesso de 128 e um número é armazenado com seu valor real somado a 128. Ex.:-3=01111101_2 (-3+128=125)

- ❑ Exercício de fixação:

- ✓ Escreva os números decimais abaixo nas seguintes representações: sinal e magnitude; representação em complemento de 1; representação em complemento de 2 e excesso de 128 (utilizando 8 bits, se existir representação).
  a) -1
  b) -20
  c) -127
  d) -128

Exemplos

- ✓ **Números negativos de 8 bits expressos em 4 sistemas diferentes**

N (decimal)	N (binário)	-N (sinal-magnitude)	-N (comple-mento de 1)	-N (comple-mento de 2)	-N (excesso de 128)
1	00000001	10000001	11111110	11111111	01111111
2	00000010	10000010	11111101	11111110	01111110
3	00000011	10000011	11111100	11111101	01111101
4	00000100	10000100	11111011	11111100	01111100
10	00001010	10001010	11110101	11110110	01110110
20	00010100	10010100	11101011	11101100	01101100
100	01100100	11100100	10011011	10011100	00011100
127	01111111	11111111	10000000	10000001	00000001
128		Não existe represen-tação	Não existe represen-tação	10000000	00000000

Representação de Números Reais

- ✓ Em alguns tipos de cálculo, a faixa de variação dos números envolvidos é muito grande.
- ✓ **Exemplo**:
  - 1) Massa do elétron - da ordem de 9×10^{-28} gramas
  - 2) Massa do Sol - aproximadamente igual a 2×10^{33} gramas
  - Faixa de variação: $> 10^{60}$
  - Exemplo de representação (34 dígitos à esquerda do ponto decimal e 28 dígitos à direita do mesmo)

1) 0000000000000000000000000000000000.0000000000000000000000000009

2) 2000000000000000000000000000000000.0000000000000000000000000000

- ✓ Como representar esses números num computador?

Representação de Números Reais

- ✓ Forma usual de representação de números reais: **parte inteira, vírgula (ou ponto), parte fracionária.**

- ✓ Esta representação, embora cômoda para cálculos no papel, não é adequada para processamento no computador.

- ✓ **Exemplo: 45,724**
- ✓ O número **45,724** pode ser expresso como:
  - **$45{,}724 \times 10^{0}$**
  - **45724×10^{-3}**
  - **$0{,}45724 \times 10^{2}$**

Representação de Números Flutuante

- ✓ É necessário o uso de um sistema de representação de números no qual a faixa de variação dos números seja independente do número de dígitos significativos dos números representados.
- ✓ Uma maneira de separar a faixa de variação dos números de sua precisão consiste em representá-lo na notação científica.

$$n = f \times 10^{e}$$

f - fração ou significando (ou mantissa)
e - expoente (inteiro positivo ou negativo)

Representação de Números Flutuante

- ✓ Qualquer número (inteiro ou fracionário) pode ser expresso no formato **número x base^{expoente}**, podendo-se variar a posição da vírgula e o expoente.
- ✓ Denominação (computacional): **representação em ponto flutuante** (o ponto varia sua posição, modificando, em consequência, o valor representado).
- ✓ Representação pode variar ("**flutuar**") a posição da vírgula, ajustando a potência da base.

Representação de Números Flutuante

- ✓ **Exemplos**:
  - $3,14 = 0,314 \times 10^{-1} = 3,14 \times 10^{0}$
  - $0,000001 = 0,1 \times 10^{-5} = 1,0 \times 10^{-6}$
  - $1941 = 0,1941 \times 10^{4} = 1,941 \times 10^{3}$

- ✓ A **faixa de variação** dos números é determinada pela quantidade de dígitos do expoente e a **precisão** é determinada pela quantidade de dígitos do significando.

Representação de Números Flutuante

- ✓ **Forma normalizada:** usa um único dígito antes da vírgula, diferente de zero (*).
- ✓ Na representação computacional de números em ponto flutuante, a representação normalizada é, em geral, melhor que a não-normalizada.
  - **Forma normalizada**: só existe uma forma de representar um número.
  - **Forma não normalizada**: um mesmo número pode ser representado de diversas maneiras.

(*) Padrão IEEE 754 para números em ponto flutuante – **significando normalizado** – começa com um bit 1, seguido de um ponto (vírgula) binário e pelo resto do significando (número = ± 1,_ _ ... x 2^{exp})

Mantissa normalizada - começa com o ponto (vírgula) binário seguido por um bit 1 e pelo resto da mantissa (bit antes da vírgula igual a zero).

Representação de Números Flutuante

Ilustração:

- ✓ No sistema binário:
  - **110101** = **110,101x2^3** = **1,10101x2^5** = **0,0110101x2^7**
- ✓ Números armazenados em um computador - os expoentes serão também gravados na base dois
  - Como $3_{10} = 11_2$ e $7=111_2$
  - $110,101 \times (10)^{11} = 1,10101x(10)^{101} = 0,0110101x(10)^{111}$
- ✓ Representação normalizada - há apenas um “1” antes da vírgula
  - **Exemplo**: **$1,10101x(10)^{101}$**

Armazenamento em *floats*

- ✓ **Na organização/arquitetura do computador, deve-se definir**:
  - Número de bits do significando (precisão, ***p*** ou ***f***)
  - Número de bits do expoente (***e***)
  - Um bit ("**0**" para + e "**1**" para -) de sinal (tipicamente o primeiro, da esquerda)

Armazenamento em *floats*

- ✓ **Ilustração (8 bits)**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Sinal	Expoente (+/-)			Significando			

- ✓ Sinal do número: 0 = + e 1 = -
- ✓ Expoentes: 8 combinações possíveis
  - **OBS**: Não seguem aritmética normal
    (p.ex.: Utiliza notação em excesso)

Armazenamento em *floats*

000	Caso especial
001	Expoente –2
010	Expoente -1
011	Expoente 0
100	Expoente 1
101	Expoente 2
110	Expoente 3
111	Caso especial

Abaixo de zero

(*bias* = polarização)

Acima de zero

Armazenamento em *floats*

Exemplo: Realize as conversões abaixo:

$6{,}75_{10}$ = (?)$_2$ (ponto flutuante, com 8 bits)

$1110\ 1001_2$ (ponto flutuante, com 8 bits) = (?)$_{10}$

Armazenamento em *floats*

Solução:

- ✓ $\mathbf{6,75_{10}} = 110,11_2 = 1,1011 \times 2^2$

  sinal: 0

  expoente: $2_{10} + 3_{10} = x_{10}$, $x_{10} = 5_{10} = 101_2$

  significando: 1011

  Número (ponto flutuante, com 8 bits):

  $$\mathbf{01011011_2}$$

Armazenamento em *floats*

Solução:

- ✓ **1110 1001_2** (ponto flutuante, 8 bits)

  sinal: 1

  expoente: $110_2 = 6_{10}$, $x_{10} + 3_{10} = 6_{10}$,

  $x_{10} = 3_{10}$

  significando: 1001_2 =

  Número:(negativo) $1{,}1001_2 \times 2^3 = 1100{,}1_2 = -12{,}5_{10}$

Armazenamento em *floats*

- ✓ **Observações:**

- ✓ **Maior número positivo** (lembre do bit escondido):
  0 110 1111 = + 2^3 x 1,1111 = 1111,1 = **15,5 decimal**

- ✓ **Menor número positivo** (lembre do bit escondido):
  0 001 0000 = + 2^{-2} x 1,0000 = 0,01 ou **0,25 decimal**

Armazenamento em *floats*

Combinações especiais dos expoentes na ilustração...

- ✓ **000** – representação NÃO normalizada
  - Significando passa a ser 0,_ _ _ ...
  - Expoente (000) = -2
  - **Menor número positivo passa a ser**
  - 0 000 0001 = 2^{-2} x 0,0001 = 2^{-2} x 2^{-4} = 2^{-6} = **0,015625 decimal**

A norma IEEE prevê o *underflow* gradual (a mantissa deixa de ser normalizada), permitindo obter números bem mais próximos de zero.

Armazenamento em *floats*

Ainda as combinações especiais...

- ✓ **Normalização não permite representar zero!**
- ✓ **000** – representação NÃO normalizada
  - 00000000 = + 0 decimal
  - 10000000 = - 0 decimal (iguais em comparações)
- ✓ **111** - representações de infinito
  - 01110000 = + infinito
  - 11110000 = - infinito
  - 11111000 = indeterminação
  - Outras combinações 11111_ _ _ = *Not A Number* (**NANs**)

Padrão IEEE 754

?