



Universidade do Estado do Rio de Janeiro
Instituto de Matemática e Estatística
Departamento de Informática e Ciência da Computação

Arquitetura de Computadores I

Aritmética Computacional
por
Jacques Alves da Silva

Aritmética Computacional

- Palavras (*words*) de bits podem ser representadas como números binários
- Números podem ser inteiros, reais, positivos e negativos
- Manipulação de números:
 - Soma
 - Subtração
 - Multiplicação
 - Divisão

Representação de Números

- Em qualquer base numérica, o valor do i -ésimo dígito d é:

$$d \times \text{base}^i$$

- Os bits são numerados com 0, 1, 2, 3, ... da direita para a esquerda em uma palavra (*word*)
- Exemplo:

$$1011_2 = \{(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)\}_{10} = 11_{10}$$

Representação de Números

- **Bit menos significativo** para designar o bit 0
- **Bit mais significativo** para designar o bit de mais alta ordem
 - Por exemplo, o bit 31 numa palavra de 32 bits

Representação de Números

- Palavra manipulada por um computador
 - Tamanho limitado
 - Os números que podem ser representados também têm tamanho limitado
- Palavra de 32 bits
 - Menor número possível = 0
 - Maior número possível = $2^{32} - 1$

Representação de Números

- Se as operações sobre os números gerarem resultados que não podem ser representados pelos n bits ocorre:
 - *Overflow* (números muito grandes)
 - *Underflow* (números muito pequenos)
- Tanto o **overflow** quanto o **underflow** geram exceções e são tratados pelo sistema operacional

Números Com e Sem Sinal

- Os programas de computador calculam números positivos e negativos
 - Representação de números com sinal
- **Sinal e Magnitude** – Acrescentar um sinal separado, que possa ser representado em um único bit
 - Colocar o bit de sinal à direita ou à esquerda?
 - Representação de 0 positivo e negativo

Números Com e Sem Sinal

- **Complemento a um**

- O negativo de um complemento a um é encontrado invertendo-se cada bit. Por exemplo, para uma palavra de 4 bits:
 - $0111_2 = +7_{10}$
 - $1000_2 = -7_{10}$
- Representação de dois 0s: $00...00_2$ é o 0 positivo e $11...11_2$ é o 0 negativo
- Quantidade: $-2^{n-1} + 1 \leq X \leq 2^{n-1} - 1$
- Etapa extra para subtrair um número

Números Com e Sem Sinal

- **Complemento a dois**

- Convenção

- 0s iniciais significa positivo
 - 1s iniciais significa negativo

- Representação de números positivos e negativos de 32 bits em termos do valor do bit vezes uma potência de 2

$$(X_{31} \times -2^{31}) + (X_{30} \times 2^{30}) + \dots + (X_1 \times 2^1) + (X_0 \times 2^0)$$

- Quantidade: $-2^{n-1} \leq X \leq 2^{n-1} - 1$

Números Com e Sem Sinal

- Vantagens do **complemento a dois**
 - Representar números negativos sempre com o bit 1 em sua posição mais significativa
 - Hardware só precisa testar este bit para verificar se o número é positivo ou negativo (bit de sinal)
 - Menor custo
 - Hardware só para a soma

Números Com e Sem Sinal

- Obtenção do **complemento a dois**
 - Inverte-se o número binário e depois soma-se 1 a este valor
 - $X + \bar{X} = -1 \Rightarrow \bar{X} + 1 = -X$
 - Regra simples
 - Copie da direita para a esquerda todos os bits até encontrar o primeiro bit 1 inclusive, e
 - Inverta todos os demais bits

Números Com e Sem Sinal

- Exemplo para obter o **complemento a dois** de número binário com 4 bits

$$0110_2 = 6_{10}$$

1001 (número binário invertido)

+ 0001 (soma com 1)

1010 (complemento a 2)

Usando a regra:

$$0110 \Rightarrow 1010$$

Números Com e Sem Sinal

- Somar o campo imediato de uma instrução a um registrador de 32 bits
 - Converter esse número de 16 bits para o seu equivalente em 32 bits
- Representar um número binário de n bits em um número maior do que n bits
 - O bit de sinal é replicado nos bits mais significativos do padrão binário com mais bits

Sequência binária em complemento a dois

0000 0000 0000 0000 0000 0000 0000 0000₂ = 0₁₀

0000 0000 0000 0000 0000 0000 0000 0001₂ = 1₁₀

0000 0000 0000 0000 0000 0000 0000 0010₂ = 2₁₀

...

...

0111 1111 1111 1111 1111 1111 1111 1101₂ = 2.147.483.645₁₀

0111 1111 1111 1111 1111 1111 1111 1110₂ = 2.147.483.646₁₀

0111 1111 1111 1111 1111 1111 1111 1111₂ = 2.147.483.647₁₀

1000 0000 0000 0000 0000 0000 0000 0000₂ = -2.147.483.648₁₀

1000 0000 0000 0000 0000 0000 0000 0001₂ = -2.147.483.647₁₀

1000 0000 0000 0000 0000 0000 0000 0010₂ = -2.147.483.646₁₀

...

...

1111 1111 1111 1111 1111 1111 1111 1101₂ = -3₁₀

1111 1111 1111 1111 1111 1111 1111 1110₂ = -2₁₀

1111 1111 1111 1111 1111 1111 1111 1111₂ = -1₁₀

Adição e Subtração

- Soma
 - Os bits são somados um a um da direita para a esquerda
 - Os **carries** são passados para o próximo bit à esquerda
- Subtração
 - Usa a adição
 - Subtraendo é simplesmente negado antes de ser somado ao minuendo
 - A máquina trata os números representados em complemento a 2

Adição e Subtração

- Operações de soma (3+2) e subtração (3-2) entre dois números representados com 4 dígitos binários

Representação Binária	Soma	Subtração
3 = 0011	1 (vai um)	111 (vai um)
2 = 0010	0011	0011
5 = 0101 (Soma)	<u>+0010</u>	<u>+1110</u> (compl. 2)
1 = 0001 (Subtração)	0101	0001

Adição e Subtração

- Condições de *overflow* para adição e subtração

Operação	Operando A	Operando B	Resultado
A+B	≥ 0	≥ 0	< 0
A+B	< 0	< 0	≥ 0
A-B	≥ 0	< 0	< 0
A-B	< 0	≥ 0	≥ 0

- Nenhum *overflow* pode ocorrer ao somar operandos positivos e negativos

Interface Hardware/Software

- O projetista de computador precisa decidir como tratar *overflows* aritméticos
 - Linguagem C ignora o *overflow* de inteiros
 - Linguagem Fortran exige que o programador seja notificado

Interface Hardware/Software

- O MIPS detecta o *overflow* com uma exceção (ou interrupção)
 - Chamada de procedimento não planejada que interrompe a execução do programa
 - O endereço da instrução que causou o *overflow* é salvo no registrador EPC (Exception Program Counter – Contador de Programa de Exceção)
 - O controle é transferido para a rotina que trata essa exceção

Interface Hardware/Software

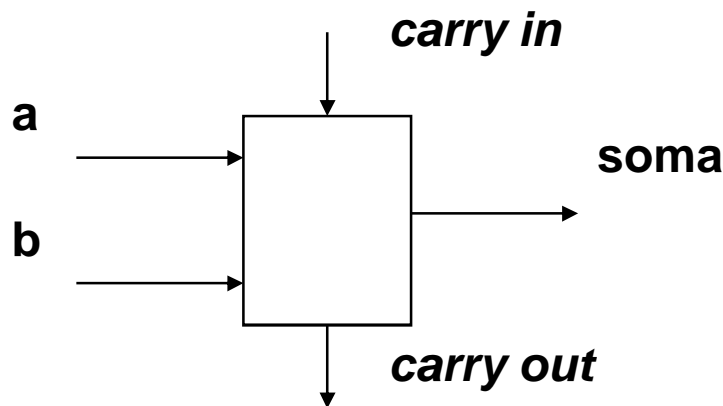
- A instrução `mfc0` (*move from system control*) é usada pelo S.O. para copiar o EPC em um registrador de uso geral
 - Retornar a instrução problemática por meio de uma instrução `jr`
- O hardware deve provocar uma exceção em algumas instruções aritméticas
 - Causam exceção no *overflow*: `add`, `addi` e `sub`
 - Não causam exceção no *overflow*: `addu` (`add unsigned`), `addiu` e `subu`

Construção de Unidade Lógica Aritmética

- *ALU (Arithmetic Logic Unit)*
 - Dispositivo que realiza as operações lógicas e aritméticas, definidas pelo conjunto de instruções, dentro do processador
- ALU é construída basicamente por quatro blocos básicos de hardware: portas **AND**, **OR**, **NOT** e **multiplexadores**
- Operações lógicas são as mais simples de serem realizadas, pois são mapeadas diretamente com componentes do hardware

Construção de Unidade Lógica Aritmética

- Adição – Circuito
 - Duas entradas para os operandos
 - Uma saída para a soma
 - Uma entrada relativa ao ***carry in***
 - Uma saída para o ***carry out***
- Somador de um bit



Construção de Unidade Lógica Aritmética

- As saídas **soma** e **carry out** podem ser especificadas através de equações lógicas

$$\text{CarryOut} = (b.\text{CarryIn}) + (a.\text{CarryIn}) + (a.b)$$

$$\text{Soma} = (a . \overline{b} . \overline{\text{CarryIn}}) + (\overline{a} . b . \overline{\text{CarryIn}}) + (\overline{a} . \overline{b} . \text{CarryIn}) + (a . b . \text{CarryIn})$$

Construção de Unidade Lógica Aritmética

- Projeto da ALU de n bits
 - n somadores de um bit
- *Carry outs* dos bits menos significativos podem ser propagados por toda a extensão do somador, gerando um *carry out* no bit mais significativo do resultado da operação
- Este somador é denominado somador de ***carry propagado***

Construção de Unidade Lógica Aritmética

- Subtração
 - Soma o minuendo com a negação do subtraendo
- Acrescenta uma entrada invertida de **b** ao somador e ativa o *carry in* do bit menos significativo
- O somador calcula:
$$a + \overline{b} + 1 = a + (\overline{b} + 1) = a + (-b) = a - b$$

Construção de Unidade Lógica Aritmética

- O problema com o somador de ***carry*** **propagado** está relacionado a velocidade de propagação do *carry*
 - Sequencial
 - Lenta
- Para solucionar este problema existem diversos esquemas para antecipar o *carry*
 - Mais portas lógicas
 - Aumento no custo

Construção de Unidade Lógica Aritmética

- Um dos esquemas para antecipar o *carry*
 - ***Carry Lookahead***
 - Implementação com base em vários níveis de abstração
- Abreviação c_i : i-ésimo bit de *carry*
- Equação do *carry*
$$c_{i+1} = (b_i \cdot c_i) + (a_i \cdot c_i) + (a_i \cdot b_i)$$
$$c_{i+1} = (a_i \cdot b_i) + (a_i + b_i) \cdot c_i$$

Construção de Unidade Lógica Aritmética

- Termos

$$(a_i \cdot b_i) \Rightarrow \text{gerador } (g_i)$$

$$(a_i + b_i) \Rightarrow \text{propagador } (p_i)$$

- Usando estas relações para definir o próximo *carry*

$$c_{i+1} = g_i + p_i \cdot c_i$$

Construção de Unidade Lógica Aritmética

- Qualquer equação lógica pode ser implementada com uma lógica de dois níveis
- Mesmo esta formulação mais simplificada pode gerar equações muito grandes
 - Circuitos lógicos relativamente grandes e caros, dependendo do número de bits a serem somados

Multiplicação

- O primeiro operando é o **multiplicando** e o segundo é o **multiplicador**. O resultado é o **produto**

Multiplicando	1000
Multiplicador	× <u>1001</u>
	1000
	0000
	0000
	<u>1000</u>
Produto	1001000

Multiplicação

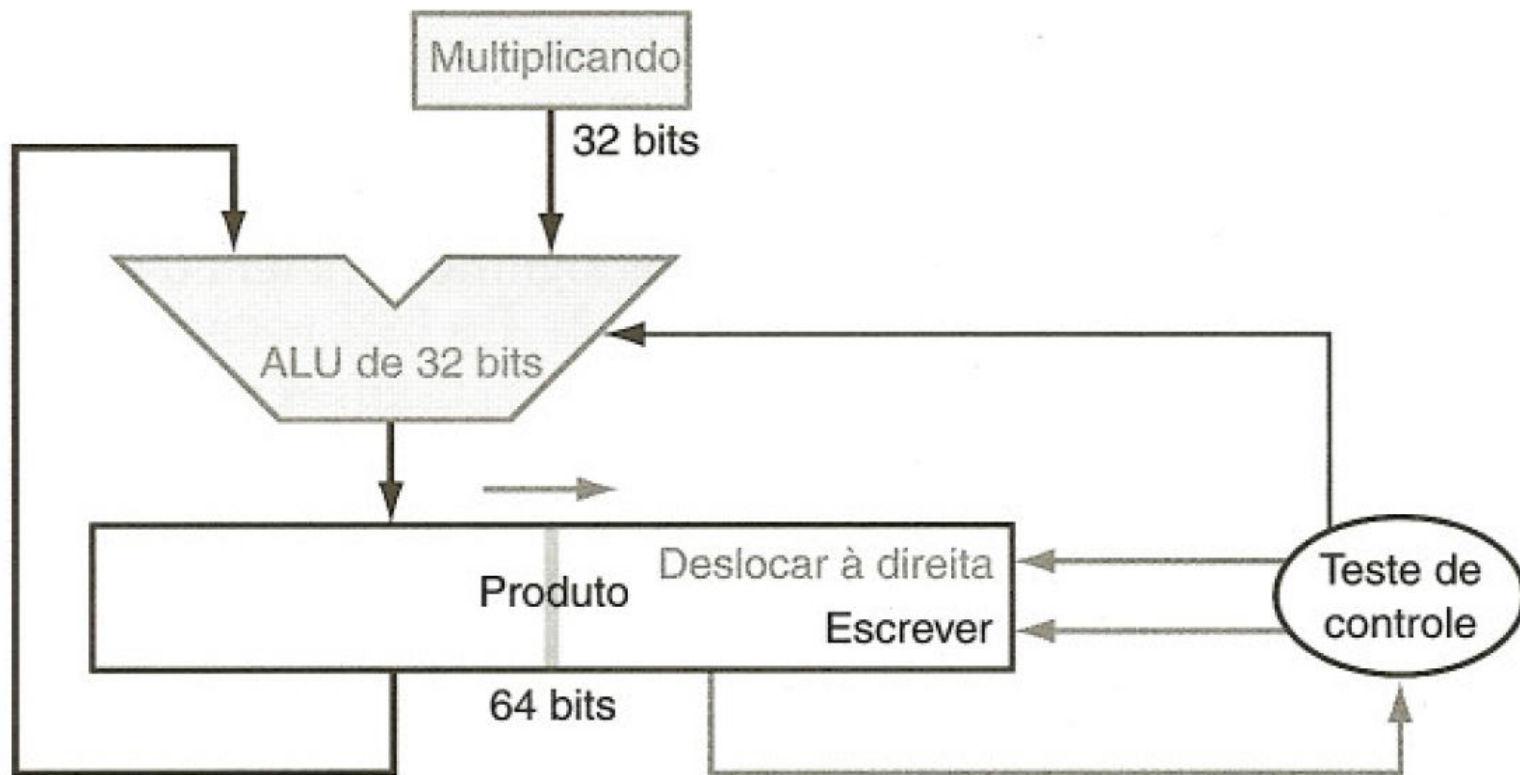
- A multiplicação de um multiplicando de n bits por um multiplicador de m bits é um produto que possui $n + m$ bits (ignorando o bit de sinal)
- Existem somente duas opções durante a multiplicação
 - Colocar uma cópia do multiplicando no lugar apropriado se o dígito do multiplicador for 1
 - Colocar 0 no lugar apropriado se o dígito for 0

Multiplicação

- Os registradores **Produto** e **Multiplicador** são combinados em um só
- O algoritmo começa com o Multiplicador na metade à direita do registrador Produto, e 0 na metade à esquerda

Multiplicação

- Hardware de multiplicação



Multiplicação

- Algoritmo da multiplicação

- 1) Testa se Produto0 é igual a 0 ou 1
- 2) Produto0 = 0, passa ao item 4
- 3) Produto0 = 1, soma o Multiplicando à metade esquerda do Produto e coloca o resultado na metade à esquerda do Produto
- 4) Desloca o Produto 1 bit à direita
- 5) Verifica se foram realizadas todas as repetições necessárias de acordo com o tamanho da palavra, se não volta ao item 1
- 6) Fim

Multiplicação

- Multiplicação com sinal
 - Converter o multiplicador e o multiplicando em números positivos
 - Depois lembrar dos sinais originais
 - Produto será negativo se os sinais originais forem diferentes
- Multiplicação no MIPS
 - O MIPS oferece um par separado de registradores de 32 bits, chamados **Hi** e **Lo**, para conter o produto de 64 bits

Divisão

- É a operação recíproca da multiplicação
- No algoritmo são utilizados dois operandos, o dividendo e o divisor, e produzidos dois resultados, o quociente e o resto

$$\text{Dividendo} = \text{Quociente} \times \text{Divisor} + \text{Resto}$$

onde o resto é menor que o divisor

Divisão

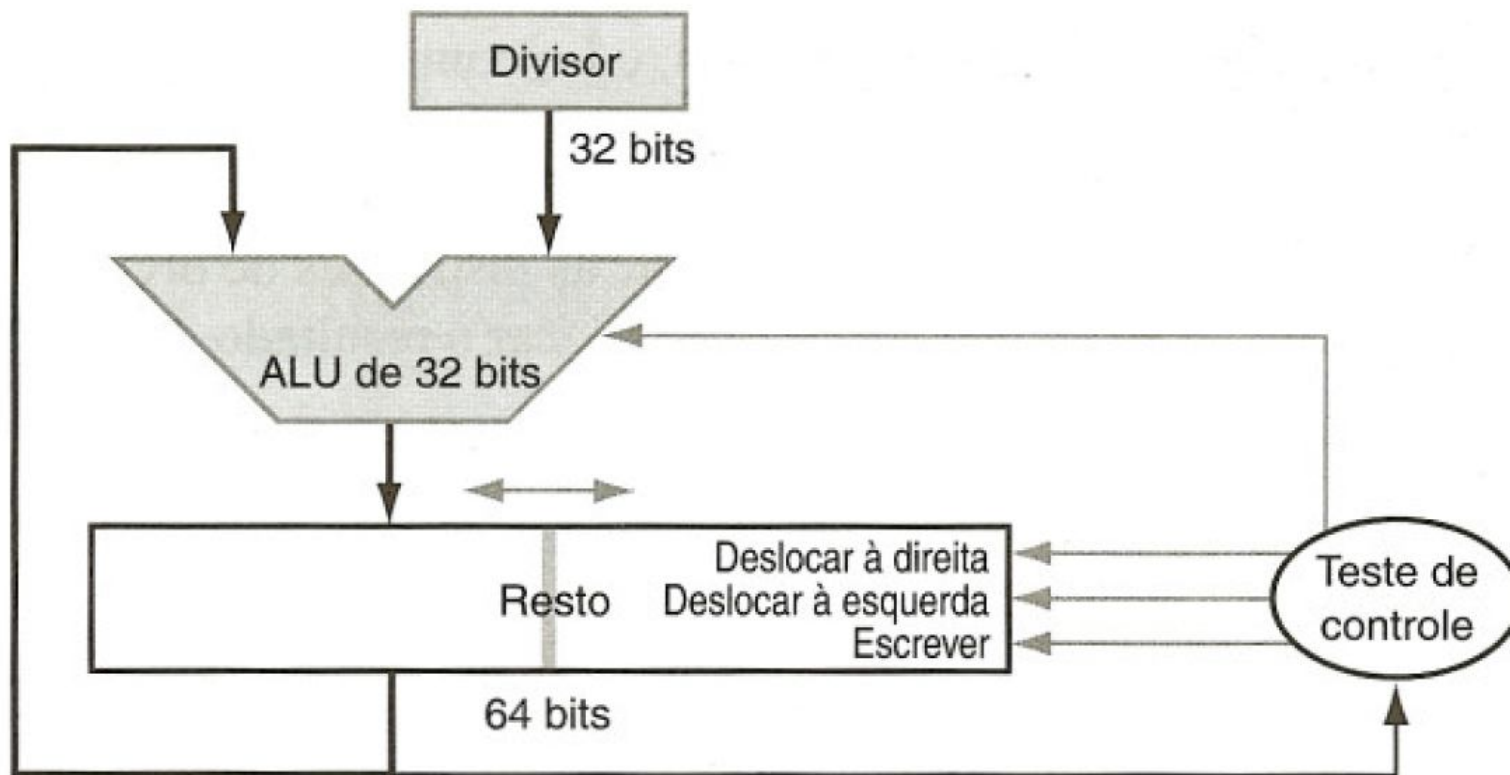
- É necessário que seja detectada a divisão por zero, que é matematicamente inválida
- O algoritmo básico da divisão tenta ver o quanto um número pode ser subtraído, criando um dígito do quociente em cada tentativa
 - O computador não sabe, com antecedência, se o divisor é menor do que o dividendo

Divisão

- Os registradores **Resto** e **Quociente** são combinados em um só
- O algoritmo começa com o Dividendo na metade à direita do registrador Resto, e 0 na metade à esquerda
- Ao final do algoritmo, o resto estará na metade à esquerda do registrador Resto, e o Quociente na metade à direita

Divisão

- Hardware de divisão



Divisão

- Algoritmo da divisão

- 1) Desloca o registrador Resto 1 bit à esquerda
- 2) Subtrai o Divisor da metade à esquerda do registrador Resto e armazena o resultado na metade esquerda do registrador Resto
- 3) Testa se Resto é menor do que 0
- 4) $\text{Resto} < 0$, restaura valor original com Divisor + metade esquerda do Resto, armazenando na metade esquerda do registrador Resto e deslocando 1 bit à esquerda, inserindo 0 no novo bit menos significativo, passa ao item 6
- 5) $\text{Resto} \geq 0$, desloca o registrador Resto 1 bit à esquerda, inserindo 1 no novo bit mais à direita
- 6) Verifica se foram realizadas todas as repetições necessárias de acordo com o tamanho da palavra, se não volta ao item 2
- 7) Desloca a metade a esquerda do registrador Resto 1 bit à direita, Fim

Divisão

- Divisão com sinal
 - Converter o divisor e o dividendo em números positivos e depois lembrar dos sinais originais
 - Negar o quociente se os sinais dos operandos forem opostos e fazer o sinal do resto (diferente de 0) igual ao sinal do dividendo
- Divisão no MIPS
 - Utiliza os registradores **Hi** e **Lo** de 32 bits
 - Ao final do algoritmo, **Hi** contém o resto e **Lo** contém o quociente

Ponto Flutuante

- As linguagens de programação admitem números fracionários (números reais)
- Além disso, muitas vezes os números são maiores do que aqueles representados com um inteiro de 32 bits com sinal
- Exemplos:

$$3,14159265..._{10} \ (\pi)$$

$$2,71828..._{10} \ (e)$$

$$0,000000001_{10} = 1,0_{10} \times 10^{-9}$$

$$3.155.760.000_{10} = 3,15576_{10} \times 10^9$$

Ponto Flutuante

- **Notação científica**

- Um único dígito à esquerda da vírgula
- Por exemplo: $3,15576_{10} \times 10^9$

- Um número em notação científica que não possui 0s na parte inteira é denominado de número **normalizado**

$1,0_{10} \times 10^{-9}$ está normalizado

$0,1_{10} \times 10^{-8}$ e $10,0_{10} \times 10^{-10}$ não estão

Ponto Flutuante

- Em binário, o formato utilizado é:

$$1,xxxxxxx_2 \times 2^{yyyy}$$

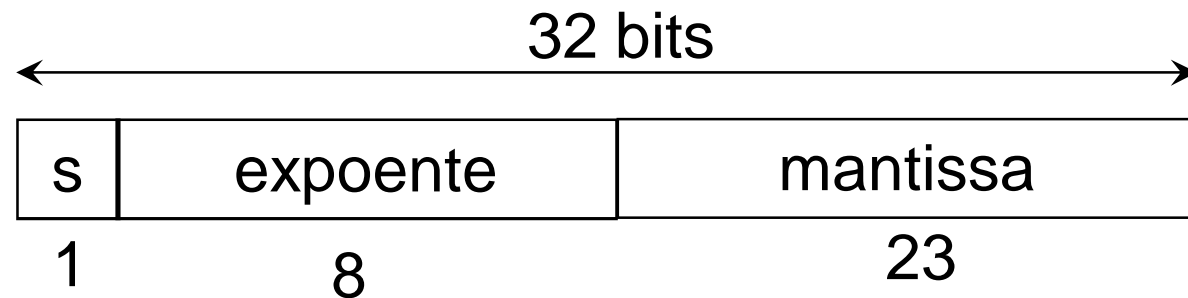
- A aritmética computacional que manipula esses números é denominada de **ponto flutuante (pf)**

Ponto Flutuante

- A notação científica normalizada oferece três vantagens
 - Simplifica a troca de dados, que incluem números em **pf**, entre diferentes computadores
 - Simplifica os algoritmos aritméticos de **pf**, pois os números sempre estarão nessa forma
 - Aumenta a precisão dos números que podem ser armazenados em uma palavra, pois os 0s desnecessários são substituídos por dígitos reais à direita da vírgula

Ponto Flutuante

- Números em ponto flutuante no MIPS (**precisão simples**) têm o formato:



- s: sinal do número (1 significando negativo)
- expoente: campo para o expoente (incluindo o sinal)
- mantissa: campo para a parte fracionária

Ponto Flutuante

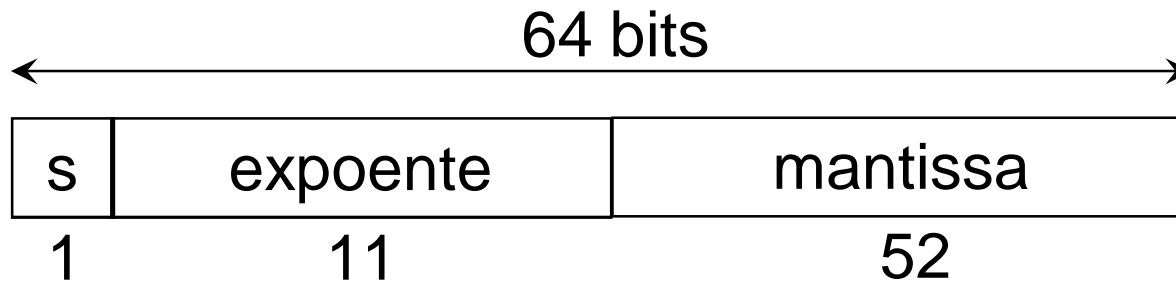
- Com 8 bits para o expoente e 23 bits para a mantissa, é possível representar:
 - Frações quase tão pequenas quanto $2,0 \times 10^{-38}$
 - Números quase tão grandes quanto $2,0 \times 10^{38}$
- O **overflow** significa que o expoente é muito grande para ser representado no campo de expoente

Ponto Flutuante

- Se uma fração é pequena demais para ser representada, significa que o expoente negativo é muito grande para caber no campo de expoente
 - Essa situação é denominada de ***underflow***
- Para reduzir as chances de *underflow* ou *overflow* é necessário oferecer outro formato com um expoente maior

Ponto Flutuante

- Ponto flutuante com **precisão dupla**:



- Utiliza 2 palavras no MIPS
- Frações quase tão pequenas quanto $2,0 \times 10^{-308}$
- Números quase tão grandes quanto $2,0 \times 10^{308}$
- Principal vantagem é sua maior precisão

Ponto Flutuante

- O padrão IEEE 754 para ponto flutuante, usado por computadores desde 1980, não representa o bit 1 inicial dos números binários normalizados
 - Considera-o implícito, já que todos os números binários normalizados possuem antes da vírgula o bit ligado

Ponto Flutuante

- O padrão IEEE 754 garante uma mantissa de 24 bits (1 implícito + 23) em precisão simples e 53 bits (1 implícito + 52) em precisão dupla
- A representação dos números em ponto flutuante corresponde a:

$$(-1)^s \times (1 + \text{Mantissa}) \times 2^{\text{Expoente}}$$

Ponto Flutuante

- Padrão de ponto flutuante IEEE 754
 - Símbolos especiais para representar eventos incomuns

Precisão simples		Precisão dupla		Objeto representado
Expoente	Mantissa	Expoente	Mantissa	
0	0	0	0	0
0	não zero	0	não zero	\pm número desnormalizado
1-254	qualquer valor	1-2046	qualquer valor	\pm número ponto flutuante
255	0	2047	0	\pm infinito
255	não zero	2047	não zero	NaN (Not a Number)

Ponto Flutuante

- O padrão IEEE 754 também busca uma representação de **pf** que possa ser facilmente processada por comparações de inteiros, especialmente para ordenação
 - O campo de sinal está no bit mais significativo, permitindo um teste rápido de menor que, maior que ou igual a 0
 - Colocar o expoente antes da mantissa simplifica a ordenação, já que números com expoentes maiores são maiores que os que tem expoentes menores (se expoentes com o mesmo sinal)

Ponto Flutuante

- Expoentes negativos impõem um desafio à ordenação simplificada
- Complemento a dois ou qualquer outra notação em que os expoentes negativos possuem um 1 no bit mais significativo do campo expoente
 - Expoente negativo parecerá número grande
 - Por exemplo:

$$1,0_2 \times 2^{-1} = 0 \ 11111111 \ 000 \dots$$

$$1,0_2 \times 2^{+1} = 0 \ 00000001 \ 000 \dots$$

Ponto Flutuante

- A notação desejável deve representar o expoente mais negativo como $00...00_2$ e o mais positivo como $11...11_2$
 - Essa convenção é denominada **notação deslocada** (*biased notation*)
 - Sendo o **bias**, o número subtraído do expoente, sem sinal, para determinar o valor real

Ponto Flutuante

- O padrão IEEE 754 usa um bias de 127 para a precisão simples
 - Expoente -1: $-1 + 127_{10} = 126_{10} \Rightarrow 01111110_2$
 - Expoente +1: $+1 + 127_{10} = 128_{10} \Rightarrow 10000000_2$
- O bias do expoente para a precisão dupla é 1023
- A representação do número em ponto flutuante é:

$$(-1)^s \times (1 + \text{Mantissa}) \times 2^{\text{Expoente} - \text{bias}}$$

Ponto Flutuante

- Exemplo: Mostre a representação binária do padrão IEEE 754 para o número $-0,75_{10}$ em precisão simples
 - Em decimal: $-0,75 = -3/4 = -3/2^2$
 - Em binário: $-(11/100) = -0,11 \times 2^0 = -1,1 \times 2^{-1}$
 - Expoente (precisão simples) é: $-1 + 127 = 126$
 - Resultado:
 $(-1)^1 \times (1 + 0,1000\ 0000\ 0000\ 0000\ 0000\ 000_2) \times 2^{(126-127)}$
 - A representação binária é:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1 bit									8 bits								23 bits															

Ponto Flutuante

- Adição e multiplicação em ponto flutuante
 - Utilizam as operações inteiras correspondentes para operar as mantissas
 - Manipulação extra nos expoentes e para a normalização dos resultados

Ponto Flutuante

- Algoritmo da adição

- 1) Compare o expoente dos dois números. Desloque o menor número à direita até que seu expoente se iguale ao maior número
- 2) Some as mantissas
- 3) Normalize a soma, deslocando à direita e incrementando o expoente ou deslocando à esquerda e decrementando o expoente
- 4) Teste se *overflow* ou *underflow*
- 5) Sim, gera exceção
- 6) Não, arredonde a mantissa para o número de bits apropriado
- 7) Testa se resultado está normalizado
- 8) Sim, Fim
- 9) Não, retorna ao passo 3

Ponto Flutuante

- Exemplo da adição: Sejam os números $9,999_{10} \times 10^1$ e $1,610_{10} \times 10^{-1}$. Suponha a restrição de 4 dígitos para a mantissa e 2 dígitos para o expoente
 - 1) A vírgula do número com menor expoente é alinhada: $1,610 \times 10^{-1} = 0,01610 \times 10^1$. É preciso manter 4 dígitos na mantissa: 0,016
 - 2) As mantissas são somadas: $9,999 + 0,016 = 10,015$. A soma é $10,015 \times 10^1$. Nesse passo é determinado o sinal do resultado

Ponto Flutuante

- 3) O número é normalizado: $10,015 \times 10^1 = 1,0015 \times 10^2$ (é necessário verificar se ocorre *overflow* ou *underflow* no expoente)
- 4) A mantissa é arredondada para 4 dígitos: $1,0015 \times 10^2 = 1,002 \times 10^2$.

A regra pode ser: se o último dígito está entre 0 e 4, o número é truncado, se está entre 5 e 9, o número é truncado e soma-se 1 ao último dígito. Caso exista uma série de 9s então o passo 3 deve ser repetido

Ponto Flutuante

- Algoritmo da multiplicação

- 1) Soma os expoentes com peso dos dois números, subtraindo o valor do peso da soma para obter o novo expoente
- 2) Multiplique as mantissas
- 3) Normalize o produto se necessário, deslocando à direita e incrementando o expoente
- 4) Teste se *overflow* ou *underflow*
- 5) Sim, gera exceção
- 6) Não, arredonde a mantissa para o número de bits apropriado
- 7) Testa se resultado está normalizado
- 8) Não, retorna ao passo 3
- 9) Sim, faça o sinal do produto positivo se ambos os sinais dos operandos originais são os mesmos, caso contrário o sinal é negativo, Fim

Ponto Flutuante

- Exemplo da multiplicação: Sejam os números $1,110_{10} \times 10^{10}$ e $9,200_{10} \times 10^{-5}$. Suponha a restrição de 4 dígitos para a mantissa e 2 dígitos para o expoente
 - 1) Os expoentes são somados: $10 + (-5) = 5$
(para os expoentes deslocados $137 + 122 - 127 = 132 = 5 + 127$)
 - 2) As mantissas são multiplicadas: $1,110 \times 9,200 = 10,212000 = 10,212$. Desta forma, o produto é $10,212 \times 10^5$

Ponto Flutuante

- 3) O número é normalizado: $10,212 \times 10^5 = 1,0212 \times 10^6$ (é necessário verificar se ocorre *overflow* ou *underflow* no expoente)
- 4) A mantissa é arredondada para 4 dígitos:
 $1,0212 \times 10^6 = 1,021 \times 10^6$
- 5) O sinal do produto é positivo, pois os sinais dos 2 operandos são iguais. O produto é:
 $+1,021 \times 10^6$

Aritmética Computacional

- Padrões de bits podem representar inteiros com sinal, inteiros sem sinal, números de ponto flutuante, instruções, etc. O que um padrão de bits representa depende da instrução que o opera

Aritmética Computacional

- A principal diferença entre os números do mundo real e os números do computador é que esses últimos possuem tamanho limitado e, portanto, uma precisão limitada
 - É possível calcular números muito grandes ou muito pequenos para serem representados em uma palavra (*word*)
 - Os programadores devem estar cientes desses limites e escrever programas de acordo com essas restrições

Bibliografia

- [1] Maria Clicia S de Castro. "Capítulo 6: Aritmética Computacional.", Apostila disponível em <http://www.ime.uerj.br/professores/Mariaclicia/Oc1/oc1.htm>
- [2] D. A. Patterson e J. L. Hennessy, "Organização e projeto de computadores: a interface hardware/software", Ed. Campus, 2005.