

“Adquira sabedoria e ajuste seu comportamento de acordo. Se esse novo comportamento te deixar temporariamente impopular com seu grupo de colegas ... então eles que vão pro inferno” (Charlie Munger).

Endianness

Paulo Lisboa de Almeida



Relembrando

Valores mais e menos significativos.

Considere o seguinte exemplo em decimal:

$$789_{10} = 7 \times 10^2 + 8 \times 10^1 + 9 \times 10^0$$

O número 7 no exemplo é multiplicado pela maior potência no polinômio.

Tem o “maior impacto” no número.

Dígito **mais significativo**.

O 9 é o valor multiplicado pela menor potência.

Têm o “menor impacto” no número.

Dígito **menos significativo**.

Relembrando

Podemos estender o raciocínio para qualquer base numérica.

Exemplo em binário:

$$11001001_2 = 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

Bit mais significativo

Bit menos significativo

Byte mais significativo

Considere o seguinte valor binário representado na memória.

$$\underbrace{0100001111001001}_{16 \text{ bits}}_2 = 0x2^{15} + 1x2^{14} + 0x2^{13} + \dots + 0x2^1 + 1x2^0$$

Boa parte das memórias são **endereçadas a byte**.

É necessário separar o valor em 2 bytes.

$$\underbrace{01000011}_{8 \text{ bits}} \quad \underbrace{11001001}_{8 \text{ bits}}$$

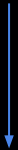
Byte mais significativo

Aplicando ideia de valor mais significativo a nível de byte.

01000011 11001001



Byte mais significativo.



Byte menos significativo.

Big-Endian

Na Memória:

01000011₂ 11001001₂

43₁₆

C9₁₆

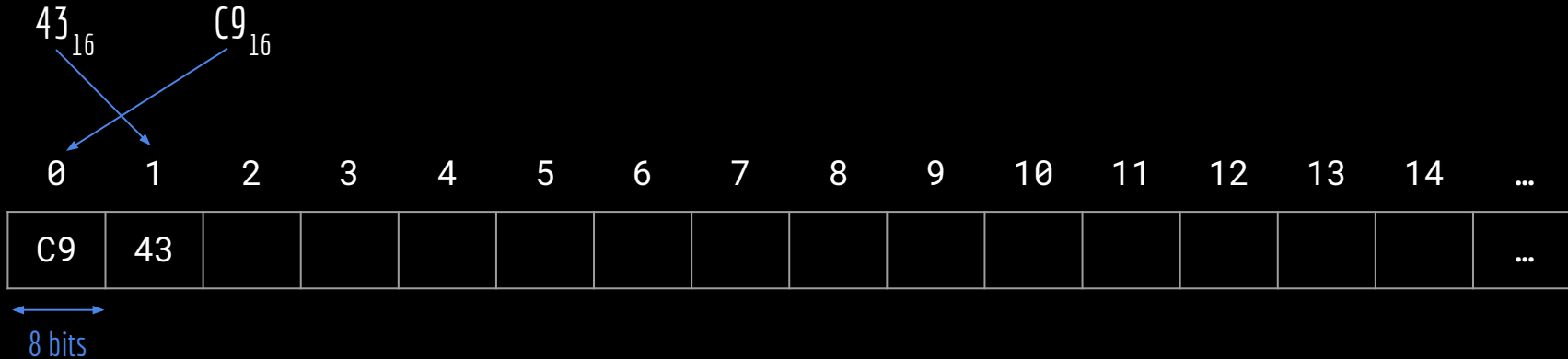
Em Hexa.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 ...

43	C9														...
----	----	--	--	--	--	--	--	--	--	--	--	--	--	--	-----

8 bits

Little-Endian



Podemos escrever os bytes na ordem inversa na memória. Essa forma é chamada de **Little-Endian**.

Little-Endian: O byte menos significativo está no endereço de memória mais baixo.

Máquinas Little-Endian e Big-Endian

Máquinas little-endian:

x86-64 e RISC-V.

Máquinas big-endian:

Motorola 68000 e Sun SPARC.

Máquinas bi-endian (exemplo: O Sistema operacional escolhe o endianness):

MIPS, ARM, PowerPC, IA-64 (Intel Itanium).

Importância

Qual a importância de conhecer esse conceito?

Importância

Qual a importância de conhecer esse conceito?

Ao transferir um dado de uma arquitetura big-endian para uma little-endian, devemos levar o endianness em consideração.

Arquiteturas big/little-endian têm vantagens e desvantagens.

Big-Endian

Vantagens do big-endian?

Big-Endian

Vantagens do big-endian?

- + É mais simples (para nós humanos) entender o valor armazenado.
- + Strings e inteiros (por exemplo) são armazenados na mesma ordem, podendo otimizar comparadores.

Exemplo:

```
int meuInteiro = 0xAABBCC00;  
char meuVetor[4] = {0xAA, 0xBB, 0xCC, 0x00};
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	...
Big-Endian	AA	BB	CC	00	AA	BB	CC	00						...
Little-Endian	00	CC	BB	AA	AA	BB	CC	00						...

Little-Endian

Vantagens do little-endian?

Considere o seguinte código C.

Considere que inteiros ocupam 4 bytes na memória, e chars ocupam 1 byte.

```
int main(){
    int meuInt = 0x00000061; //código da letra 'a' em hexa
    char meuChar = (char)meuInt; //fazer um cast para pegar o byte mais baixo

    printf("%c", meuChar); //vai imprimir a letra 'a' (código 61 hexa ASCII)
    return 0;
}
```

Little-Endian

```
int main(){
    int meuInt = 0x00000061; // código da letra 'a' em hexa
    char meuChar = (char)meuInt; // fazer um cast para pegar o byte mais baixo

    printf("%c", meuChar); // vai imprimir a letra 'a' (código 61 hexa ASCII)
    return 0;
}
```

[illegible]

Castings

Fazer um *cast* para um tipo menor é o mesmo que carregar os n bytes mais baixos a partir de um endereço.

No exemplo um inteiro possui 4 bytes, e um char 1 byte.

Para realizar o cast, carregamos somente o byte mais baixo do inteiro para dentro do char.

Little-Endian

Considere as instruções RISC-V

`lw REG, ENDEREÇO`

Carrega 4 bytes a partir do endereço especificado.

`lb REG, ENDEREÇO`

Carrega 1 byte a partir do endereço especificado.

Big-Endian

0	1	2	3	4	...
00	00	00	61		...

Little-Endian

0	1	2	3	4	...
61	00	00	00		...

Little-Endian

Considerando Big-Endian.

Qual o endereço inicial de *meuInt* se o carregarmos como um inteiro?

Qual o endereço inicial de *meuInt* se o carregarmos como um char?

Considerando Little-Endian.

Qual o endereço inicial de *meuInt* se o carregarmos como um inteiro?

Qual o endereço inicial de *meuInt* se o carregarmos como um char?

	0	1	2	3	4	...
Big-Endian	00	00	00	61		...
	0	1	2	3	4	...
Little-Endian	61	00	00	00		...

Little-Endian

Vantagens do little-endian.

- + Os **endereços não precisam ser recalculados** quando fazemos castings.

Somente a quantidade de bytes carregados é alterada!

Fazemos castings o tempo todo em nossos programas, mesmo que muitas vezes isso não esteja explícito.

Little-Endian

Vantagens do little-endian.

- + Os **endereços não precisam ser recalculados** quando fazemos castings.
 - Somente a quantidade de bytes carregados é alterada!
 - Fazemos castings o tempo todo em nossos programas, mesmo que muitas vezes isso não esteja explícito.
- + É mais rápido/fácil realizar operações aritméticas
 - Operações partem do bit de ordem mais baixa.
 - Procurar por esse bit é mais fácil, pois ele necessariamente está no primeiro byte.

Little-Endian

Vantagens do little-endian.

- + Os **endereços não precisam ser recalculados** quando fazemos castings.
 - Somente a quantidade de bytes carregados é alterada!
 - Fazemos castings o tempo todo em nossos programas, mesmo que muitas vezes isso não esteja explícito.
- + É mais rápido/fácil realizar operações aritméticas
 - Operações partem do bit de ordem mais baixa.
 - Procurar por esse bit é mais fácil, pois ele necessariamente está no primeiro byte.
- + É mais fácil construir uma máquina sem restrições de alinhamento (como o x86-84) utilizando little-endian.

Little-Endian vs Big-Endian

As viagens de Gulliver (Jonathan Swift, 1726).

Os Lilliputeanos (o povo pequenino) se dividiam em dois grupos.

Um grupo pregava que ovos cozidos deveriam ser abertos a partir da extremidade menor.

Os little-endians.

O outro, pregava que os ovos cozidos deveriam ser abertos a partir da extremidade maior.

Os big-endians.

A extremidade pela qual os moradores comiam ovos cozidos era motivo de disputas e guerras.



Little-Endian vs Big-Endian

As viagens de Gulliver (Jonathan Swift, 1726).

Os Lilliputeanos (o povo pequenino) se dividiam em dois grupos.

Um grupo pregava que ovos cozidos deveriam ser abertos a partir da extremidade menor.

Os little-endians.

O outro, pregava que os ovos cozidos deveriam ser abertos a partir da extremidade maior.

Os big-endians.

A extremidade pela qual os moradores comiam ovos cozidos era motivo de disputas e guerras.

Conclusão: Cientistas/Engenheiros têm um humor estranho.



Ordem dos bits

Quando falamos em *endianess*, geralmente estamos nos referindo a **ordem dos bytes**.

Também podemos aplicar o conceito nos bits.

Máquinas podem ser little-endian em seus bytes, e big-endian em seus bits.

Exemplo

O seu x86-64 é little-endian.

Armazena os bytes na “ordem reversa”.

No entanto, os bits de cada byte são armazenados da “maneira convencional”.

bits em big-endian.

Exercício

Crie o seguinte programa em C e compile normalmente com o GCC no Linux.

gcc meuPrograma.c -o nomeBinario

Atenção: não inclua nenhuma biblioteca no programa.

```
int main(){
    int a = 0xAABBCCDD;
    int b = 0x10111213;
    a = a + b;

    return 0;
}
```

Exercício

No binário gerado, execute os seguintes comandos:

```
xxd -p nomeBinario > nomeBinario.hex
```

O comando vai pegar os **bits do binário em ordem de 4 em 4**, e vai gerar uma saída em hexadecimal.

0000 se torna 0, 0001 se torna 1, ... 1111 se torna F.

O resultado será salvo no arquivo nomeBinario.hex.

```
objdump -d -M intel nomeBinario > nomeBinario.s
```

O comando vai fazer uma **engenharia reversa do seu binário**.

Vai mostrar o Assembly x86-64 do programa no formato compatível com os manuais da Intel.

Exercício

Compare os arquivos do seu programa em C, o hexadecimal gerado, e o código assembly.

Procure pelo trecho `int a = 0xAABBCCDD`.

Provavelmente a primeira coisa que seu compilador fez foi salvar esse valor na pilha.

O quadro na pilha pode ser indicado pelo registrador RBP no x86-64.

Frame pointer.

O opcode do `mov` (dependendo de qual versão do `mov` o seu compilador vai utilizar) pode ser $C7_{16}$.

Pode ser diferente dependendo do programa e do compilador.

Procure por esse *int* no assembly e no hexadecimal e entenda como ele ficou organizado na memória.

Exercício

Você deve:

Marcar no arquivo assembly (gerado pelo objdump) claramente onde está o trecho que salva o valor 0xAABBCCDD na memória, ou que carrega esse valor para um registrador.

Marcar no arquivo hexadecimal claramente onde está o trecho que salva o valor 0xAABBCCDD na memória, ou que carrega esse valor para um registrador.

Baseado nos dois arquivos, responda:

O x86-64 é little ou big-endian?

O *endianess* do assembly é necessariamente o mesmo *endianess* do hexadecimal?

Qual dos arquivos representa o *endianess* real da máquina?

Associe suas respostas com a diferença entre assembly e linguagem de máquina.

Resposta - Na minha máquina

Hexadecimal.

Assembly.

c7 45 f8 dd cc bb aa mov DWORD PTR [rbp-0x8] 0xaabbccdd

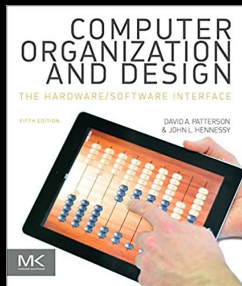


-0x8 em complemento de 2.

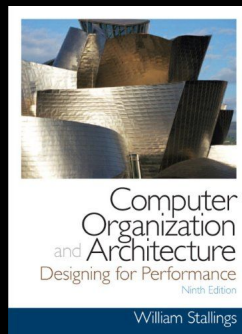
Será feito um deslocamento a partir de rbp.

Referências

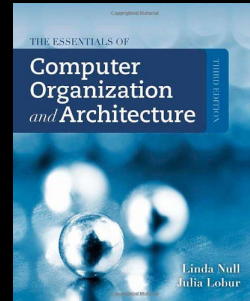
Patterson, Hennessy .
Arquitetura e Organização de
Computadores: A interface
hardware/software. 2014.



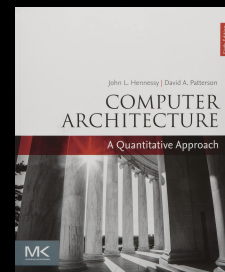
Stallings, W. Organização
de Arquitetura de
Computadores. 2012.



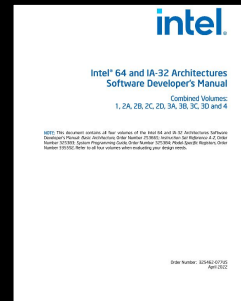
Null, Lobur. The Essentials of
Computer Organization and
Architecture. 2014.



Hennessy, Patterson.
Arquitetura de Computadores:
uma abordagem quantitativa.
2019.



Intel. Manual x86-64.
<https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>.



Licença

Esta obra está licenciada com uma Licença [Creative Commons Atribuição 4.0 Internacional](https://creativecommons.org/licenses/by/4.0/).