Organização de Computadores

Conjuntos de instruções: Características e funções

Prof. José Paulo G. de Oliveira
Eng. da Computação, UPE
jpgo@ecomp.poli.br

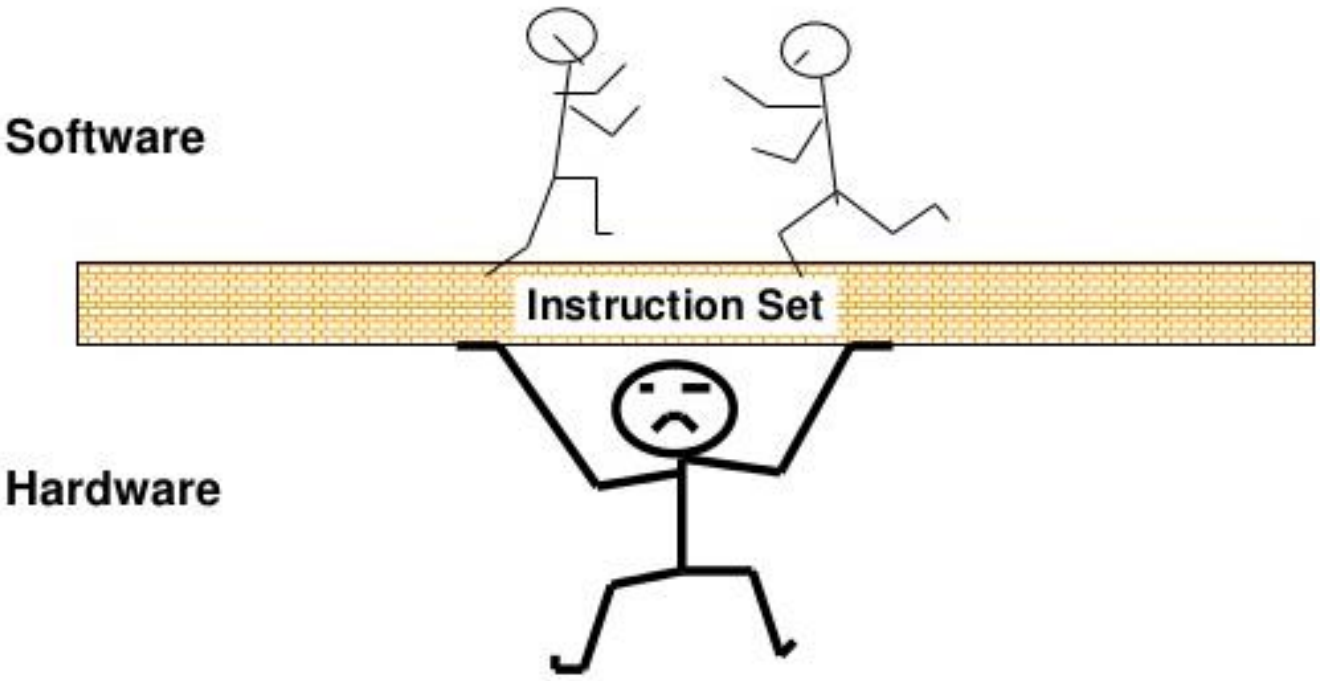
Objetivos

- ISA – *Instruction Set Architecture*
  - Definição
  - Formato e elementos
  - Endereços e registradores
  - Decisões de projeto
  - Tipos de instrução
  - Tipos de dados
  - Operações e OP Codes
- Exemplos de instruções
- Armazenamento (e*ndianess*)

Objetivos

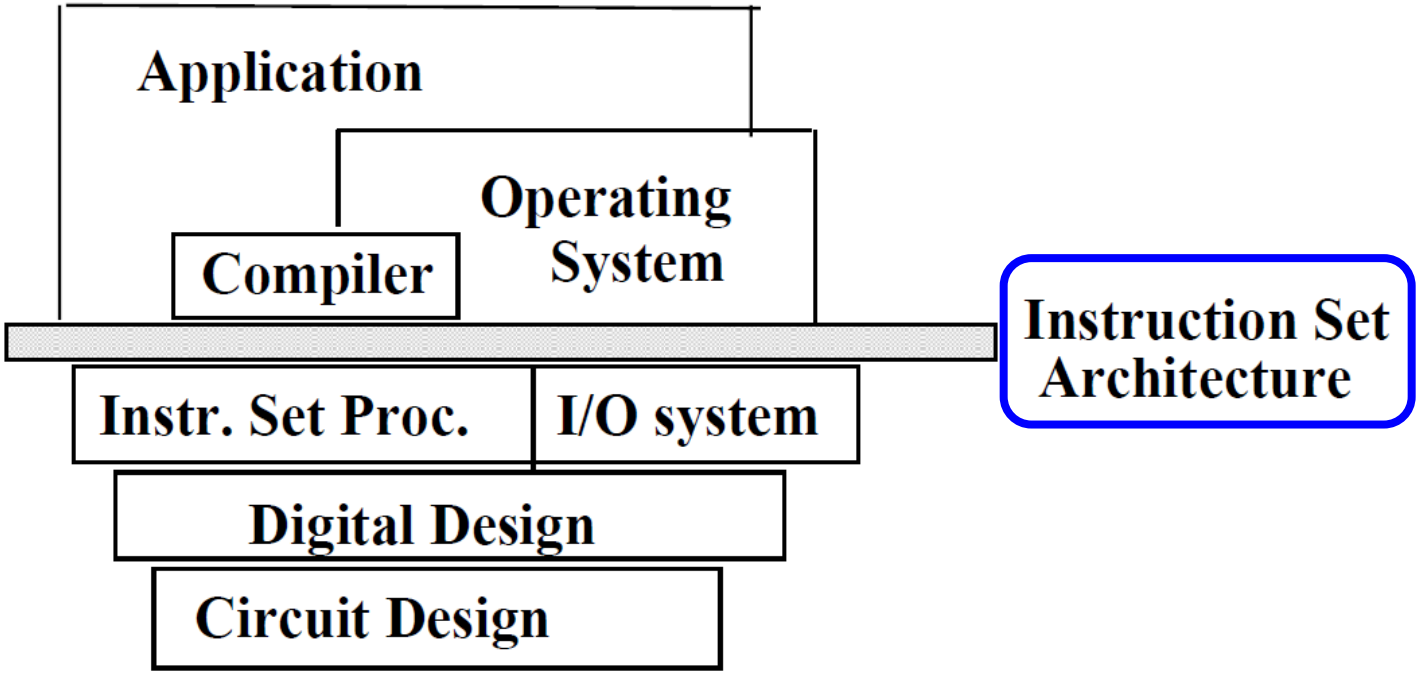
- ISA – *Instruction Set Architecture*
  - **Definição**
  - Formato e elementos
  - Endereços e registradores
  - Decisões de projeto
  - Tipos de instrução
  - Tipos de dados
  - Operações e OP Codes
- Exemplos de instruções
- Armazenamento (e*ndianess*)

ISA – Instruction Set architecture



Fonte: Computer Architecture: A Quantitative Approach [Hennessy, John L., Patterson, David A.]

ISA – Instruction Set architecture



O que é um conjunto de instruções?

- A coleção completa de instruções que são entendidas por uma CPU
- Código de máquina
- Binário
- Normalmente, representado por códigos em *assembly*

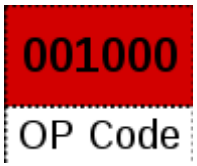
Objetivos

- ISA – *Instruction Set Architecture*
  - Definição
  - **Formato e elementos**
  - Endereços e registradores
  - Decisões de projeto
  - Tipos de instrução
  - Tipos de dados
  - Operações e OP Codes
- Exemplos de instruções
- Armazenamento (e*ndianess*)

Elementos de uma instrução

- Código de operação (Op code):
  - Faça isto

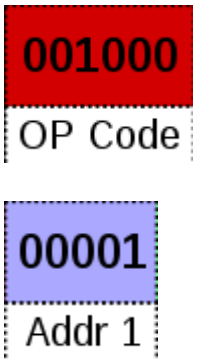
Exemplo:



Elementos de uma instrução

- Código de operação (Op code):
  - Faça isto
- Referência a operando fonte:
  - Neste operando

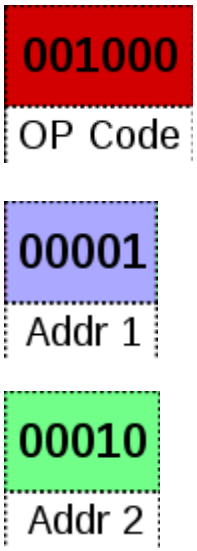
Exemplo:



Elementos de uma instrução

- Código de operação (Op code):
  - Faça isto
- Referência a operando fonte:
  - Neste operando
- Referência a operando de destino:
  - Coloque a resposta aqui

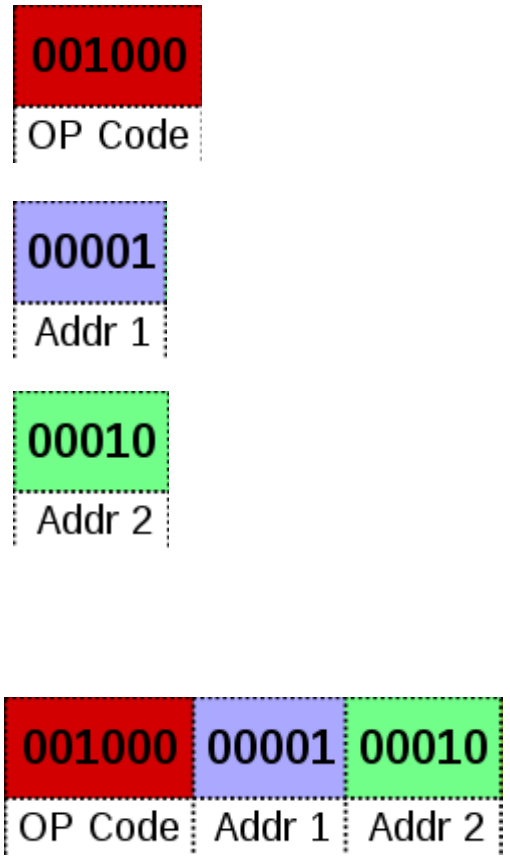
Exemplo:



Elementos de uma instrução

- Código de operação (Op code):
  - Faça isto
- Referência a operando fonte:
  - Neste operando
- Referência a operando de destino:
  - Coloque a resposta aqui
- Referência à próxima instrução:
  - Quando tiver feito isso, faça isto...
  - *Quase sempre implícita: PC*

Exemplo:



Representação da instrução

- Em código de máquina, cada instrução tem um padrão de bits exclusivo

Representação da instrução

- Em código de máquina, cada instrução tem um padrão de bits exclusivo
- Para *consumo humano*, uma representação simbólica da operação é utilizada
  - E.g., ADD, SUB, LOAD

Representação da instrução

- Em código de máquina, cada instrução tem um padrão de bits exclusivo
- Para *consumo humano*, uma representação simbólica da operação é utilizada
  - E.g., ADD, SUB, LOAD
- Operandos também podem ser representados desta maneira:
  - ADD A,B

Ex. de formato de instrução simples

Obs.: Mais de um formato pode ser utilizado na mesma arquitetura

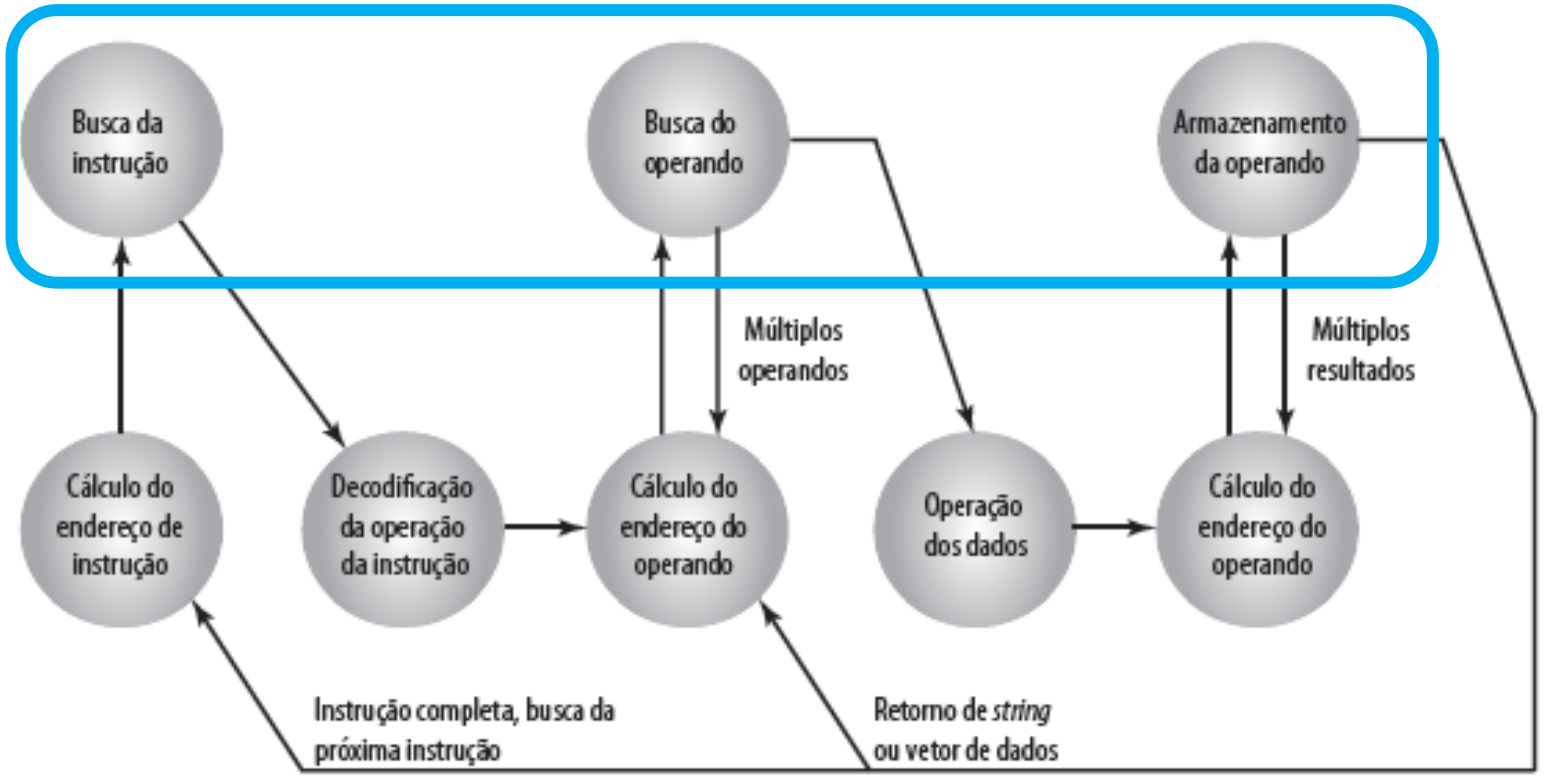
Objetivos

- ISA – *Instruction Set Architecture*
  - Definição
  - Formato e elementos
  - **Endereços e registradores**
  - Decisões de projeto
  - Tipos de instrução
  - Tipos de dados
  - Operações e OP Codes
- Exemplos de instruções
- Armazenamento (e*ndianess*)

Para onde vão (de onde vêm) todos os operandos (e instruções)?

- Memória principal (ou memória virtual ou cache)
- Registrador da CPU
- Dispositivo de E/S

Relembrando: Diagrama de estados do ciclo de instrução



Número de endereços*

*Endereço = posição !!!

1. Operando 1
2. Operando 2
3. Resultado
4. Próxima instrução

→ 4 endereços!

Número de endereços

→ 4 endereços!

Maioria dos arquiteturas não usa → no máximo 3 endereços:

Próxima instrução - PC

Número de endereços (a)

- 3 endereços:
  - Operando 1, Operando 2, Resultado
  - a = b + c

Número de endereços (a)

- 3 endereços:
  - Operando 1, Operando 2, Resultado
  - a = b + c
  - Não é comum
  - Precisa de palavras (formato das intruções) muito longas para acomodar tudo
- ADD, A, B, C

Número de endereços (b)

- 2 endereços:
  - Um endereço servindo como operando e resultado
  - a = a + b

Número de endereços (b)

- 2 endereços:
  - Um endereço servindo como operando e resultado
  - a = a + b
  - Reduz tamanho da instrução
  - Requer algum trabalho extra
    - Armazenamento temporário para manter alguns resultados
- ADD, A, B

Número de endereços (c)

- 1 endereço:
  - Segundo endereço implícito
  - Normalmente, um registrador
  - E.g., acumulador
    - Comum nas máquinas mais antigas
- ADD, B

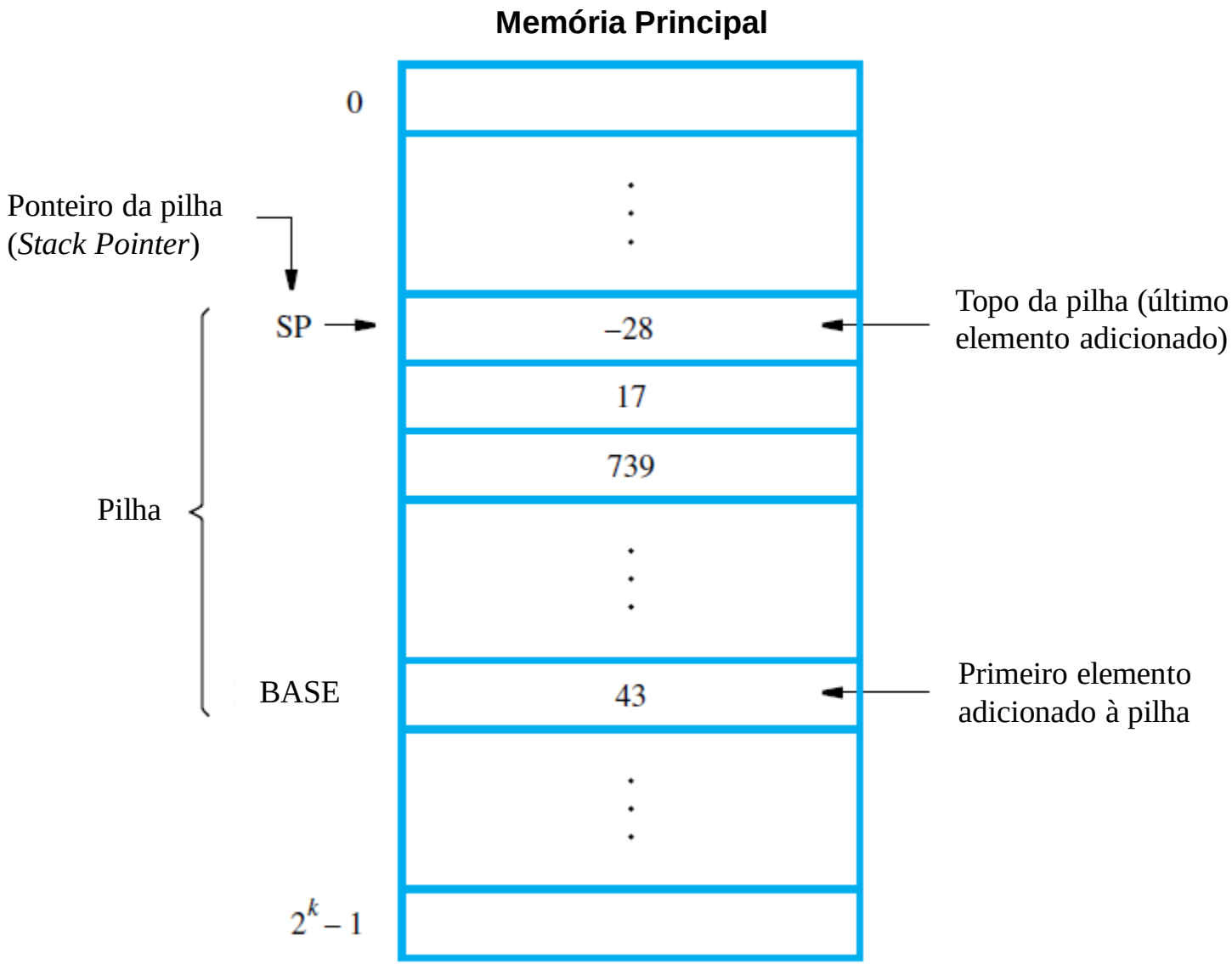
Número de endereços (d)

- 0 (zero) endereços:
  - Todos os endereços implícitos
  - Usa uma **pilha** (estrutura LIFO)

    - push a
    - push b
    - ADD
    - pop c

    - c = a + b

Pilha (*Stack*)


Memória Principal
0
Ponteiro da pilha
(Stack Pointer)
SP
–28
17
739
Pilha
BASE
43
$2^k - 1$
Topo da pilha (último
elemento adicionado)
Primeiro elemento
adicionado à pilha

Número de endereços

Número de endereços	Representação simbólica	Interpretação
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T – 1) OP T

AC = acumulador
T = topo da pilha
(T – 1) = segundo elemento da pilha
A, B, C = locais de memória ou registradores

Objetivos

- ISA – *Instruction Set Architecture*
  - Definição
  - Formato e elementos
  - Endereços e registradores
  - **Decisões de projeto**
  - Tipos de instrução
  - Tipos de dados
  - Operações e OP Codes
- Exemplos de instruções
- Armazenamento (e*ndianess*)

Decisões de projeto (ISA)

- Repertório de operações*
  - Quantas operações?
  - O que podem fazer?
  - Qual sua complexidade?
- Tipos de dados*
- Formatos de instrução**
  - Tamanho do campo de código de operação
  - Número de endereços

*Abordados mais adiante

** Já abordado

Decisões de projeto (ISA)

- Registradores:
  - Número de registradores da CPU disponíveis
  - Quais operações podem ser realizadas sobre quais registradores?
- Modos de endereçamento:
  - Como o endereço dos operandos é especificado

- RISC vs. CISC

Quantos endereços, então?

- Mais endereços:
  - Instruções mais complexas
  - Mais registradores
    - Operações entre registradores são mais rápidas
  - Menos instruções por programa

Quantos endereços, então?

- Mais endereços:
  - Instruções mais complexas
  - Mais registradores
    - Operações entre registradores são mais rápidas
  - Menos instruções por programa
- Menos endereços:
  - Instruções menos complexas
  - Busca/execução de instruções mais rápida
  - Mais instruções por programa

Arquiteturas Load/Store

Permitido:

LOAD r3, M(endereço)

e

ADD r1=r2+r3

⇒ força dependência de registradores

Arquiteturas Load/Store

Permitido:

LOAD r3, M(endereço)

e

ADD r1=r2+r3

⇒ força dependência de registradores

NÃO é permitido:

ADD r1 = r2 + M(endereço)

⇒ mais instruções, implementação mais rápida

Arquiteturas Load/Store

Permitido:

LOAD r3, M(endereço)

e

ADD r1=r2+r3

NÃO é permitido:

ADD r1 = r2 + M(endereço)

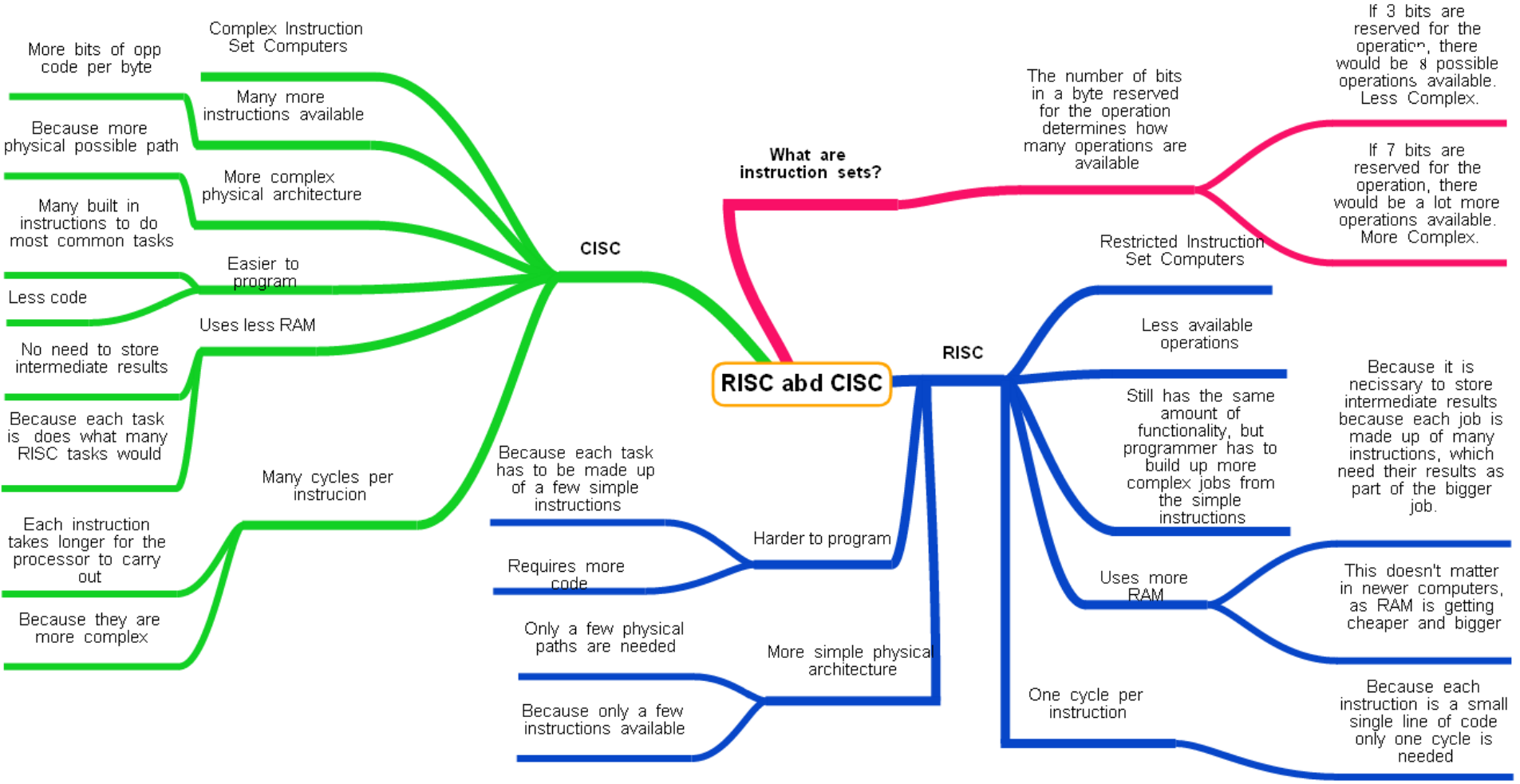
⇒ força dependência de registradores

⇒ mais instruções, implementação mais rápida

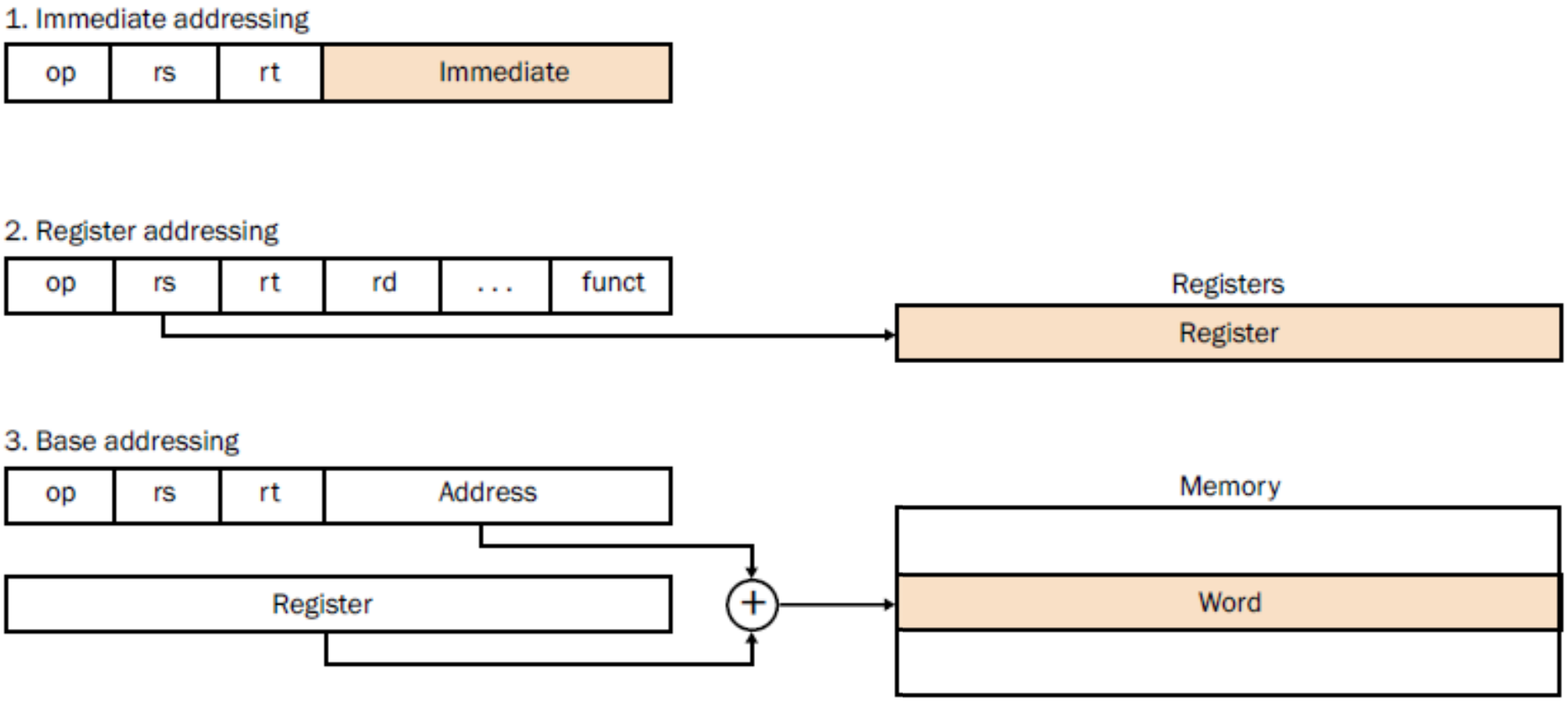
é o que se busca em projetos atuais!!!

⇒ RISC

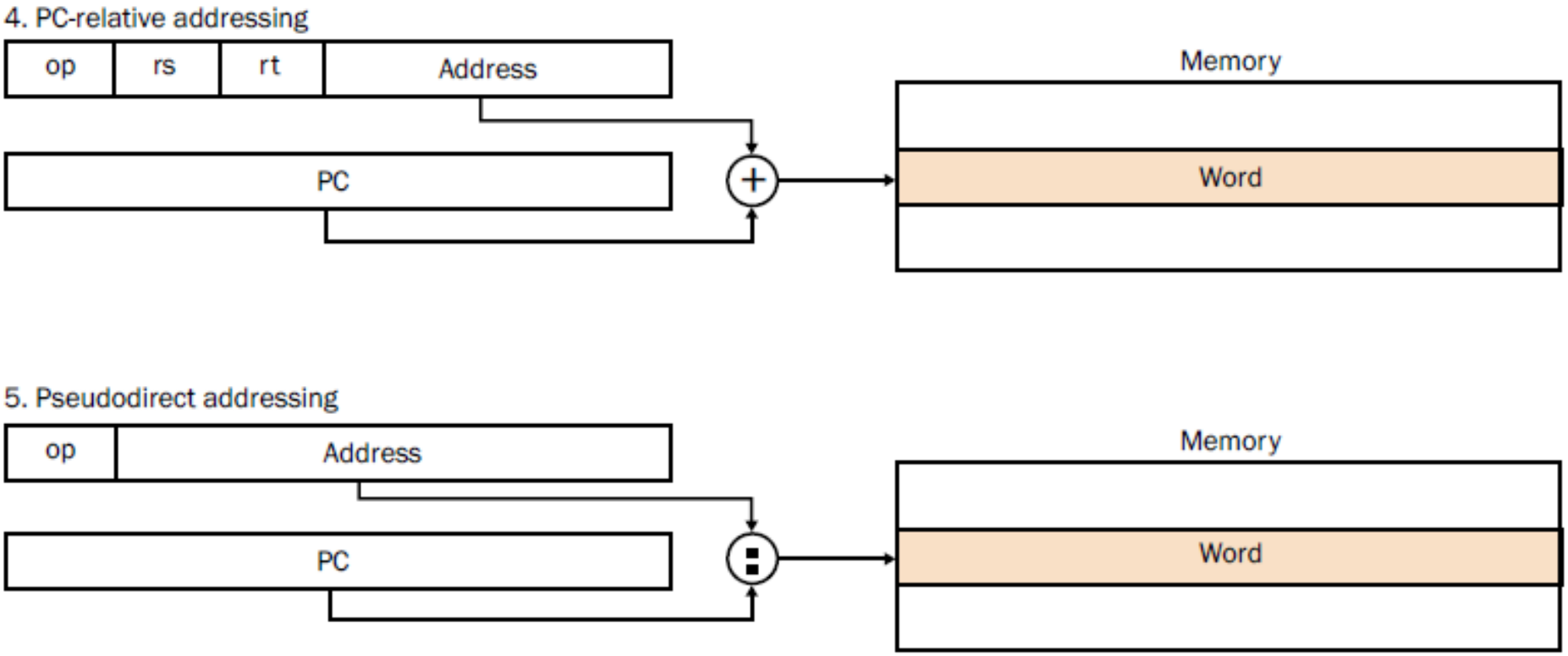
RISC vs. CISC


RISC abd CISC
What are instruction sets?
The number of bits in a byte reserved for the operation determines how many operations are available
If 3 bits are reserved for the operation, there would be 8 possible operations available. Less Complex.
If 7 bits are reserved for the operation, there would be a lot more operations available. More Complex.
CISC
Complex Instruction Set Computers
More bits of opp code per byte
Many more instructions available
Because more physical possible path
More complex physical architecture
Many built in instructions to do most common tasks
Easier to program
Less code
Uses less RAM
No need to store intermediate results
Because each task is does what many RISC tasks would
Many cycles per instrucion
Each instruction takes longer for the processor to carry out
Because they are more complex
RISC
Restricted Instruction Set Computers
Less available operations
Still has the same amount of functionality, but programmer has to build up more complex jobs from the simple instructions
Because it is necissary to store intermediate results because each job is made up of many instructions, which need their results as part of the bigger job.
Uses more RAM
This doesn't matter in newer computers, as RAM is getting cheaper and bigger
One cycle per instruction
Because each instruction is a small single line of code only one cycle is needed
Harder to program
Because each task has to be made up of a few simple instructions
Requires more code
More simple physical architecture
Only a few physical paths are needed
Because only a few instructions available

Formas de endereçamento



Formas de endereçamento



Objetivos

- ISA – *Instruction Set Architecture*
  - Definição
  - Formato e elementos
  - Endereços e registradores
  - Decisões de projeto
  - **Tipos de instrução**
  - Tipos de dados
  - Operações e OP Codes
- Exemplos de instruções
- Armazenamento (e*ndianess*)

Tipos de instrução

- Processamento de dados
  - Aritmética e Lógica
- Transferência de dados
  - Registradores $\Leftrightarrow$ Memória
  - E/S
- Controle de fluxo do programa
  - Teste e desvio
- Controle do sistema

Tipos de operação

- **Transferência de dados**
- Aritmética
- Lógica
- Conversão
- E/S
- Controle do sistema
- Transferência de controle

Transferência de dados

- Especificam:
  - Origem
  - Destino
  - Quantidade de dados
- Podem ser instruções diferentes para diferentes movimentações
  - Ex., IBM 370 (Menos compacta)
- Ou uma única instrução e diferentes endereços
  - Ex., VAX. (Menos mnemônicos para aprender)
- Operações de E/S (mapeamento na memória)

Tipos de operação

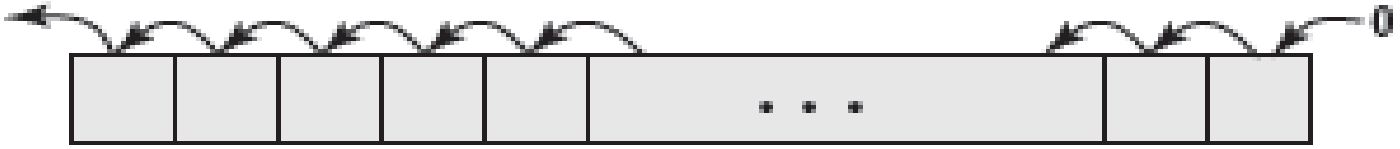
- Transferência de dados
- **Aritmética**
- Lógica
- Conversão
- E/S
- Controle do sistema
- Transferência de controle

Aritmética

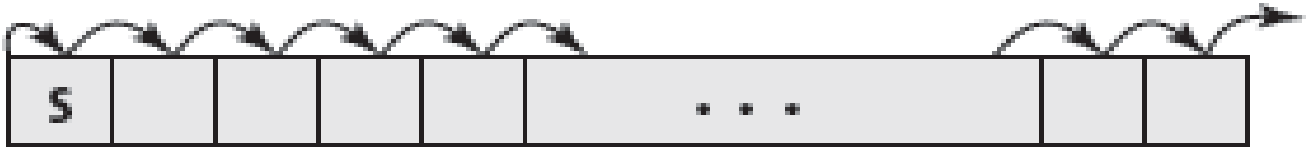
- Adição, Subtração, Multiplicação, Divisão
- Inteiro com sinal
- Ponto flutuante?
- Pode incluir:
  - Incremento (a++)
  - Decremento (a--)
  - Negação (-a)
  - Deslocamento/rotação

Operações de deslocamento e rotação

(a) Deslocamento lógico à direita

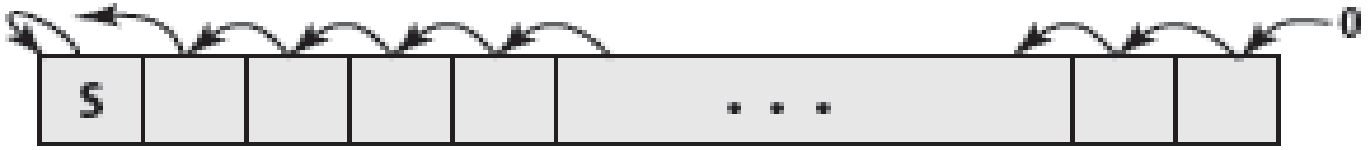


(b) Deslocamento lógico à esquerda

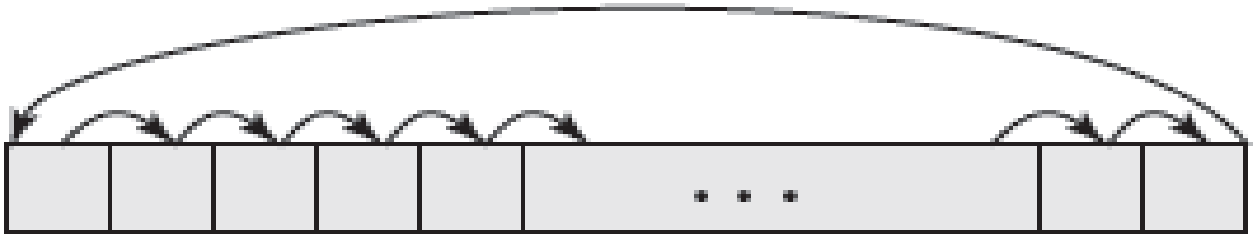


(c) Deslocamento aritmético à direita

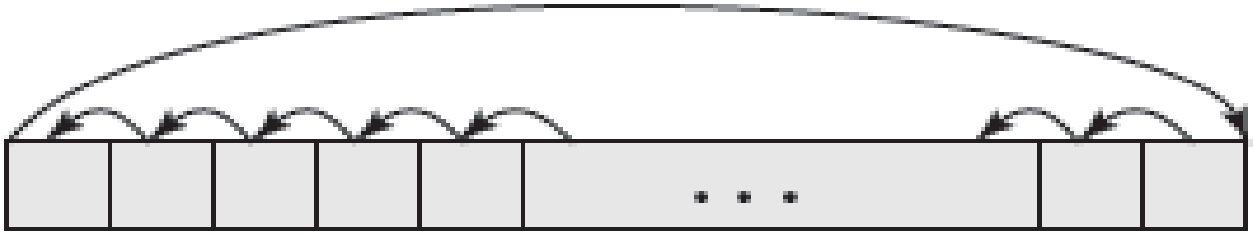
Operações de deslocamento e rotação



(d) Deslocamento aritmético à esquerda



(e) Rotação à direita



(f) Rotação à esquerda

Tipos de operação

- Transferência de dados
- Aritmética
- **Lógica**
- Conversão
- E/S
- Controle do sistema
- Transferência de controle

Lógica

- Operações bit a bit
- AND, OR, NOT

P	Q	NOT P	P AND Q	Q OR Q	P XOR Q	P = Q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Tipos de operação

- Transferência de dados
- Aritmética
- Lógica
- **Conversão**
- E/S
- Controle do sistema
- Transferência de controle

Conversão

- Exemplo: binário para decimal
  - 1000 → 8 (inteiro)
  - 1000 → 8,0 (real)

Tipos de operação

- Transferência de dados
- Aritmética
- Lógica
- Conversão
- **E/S**
- Controle do sistema
- Transferência de controle

Entrada/saída

- Podem ser instruções específicas
  - mapeamento direto
- Pode ser feita usando instruções de leitura/escrita
  - mapeadas na memória
- Pode ser feita por um controlador separado
  - processador de E/S

Tipos de operação

- Transferência de dados
- Aritmética
- Lógica
- Conversão
- E/S
- **Controle do sistema**
- Transferência de controle

Controle do sistema

- Instruções privilegiadas
- CPU precisa estar em estado específico
  - modo kernel
- *Para uso com sistemas operacionais*

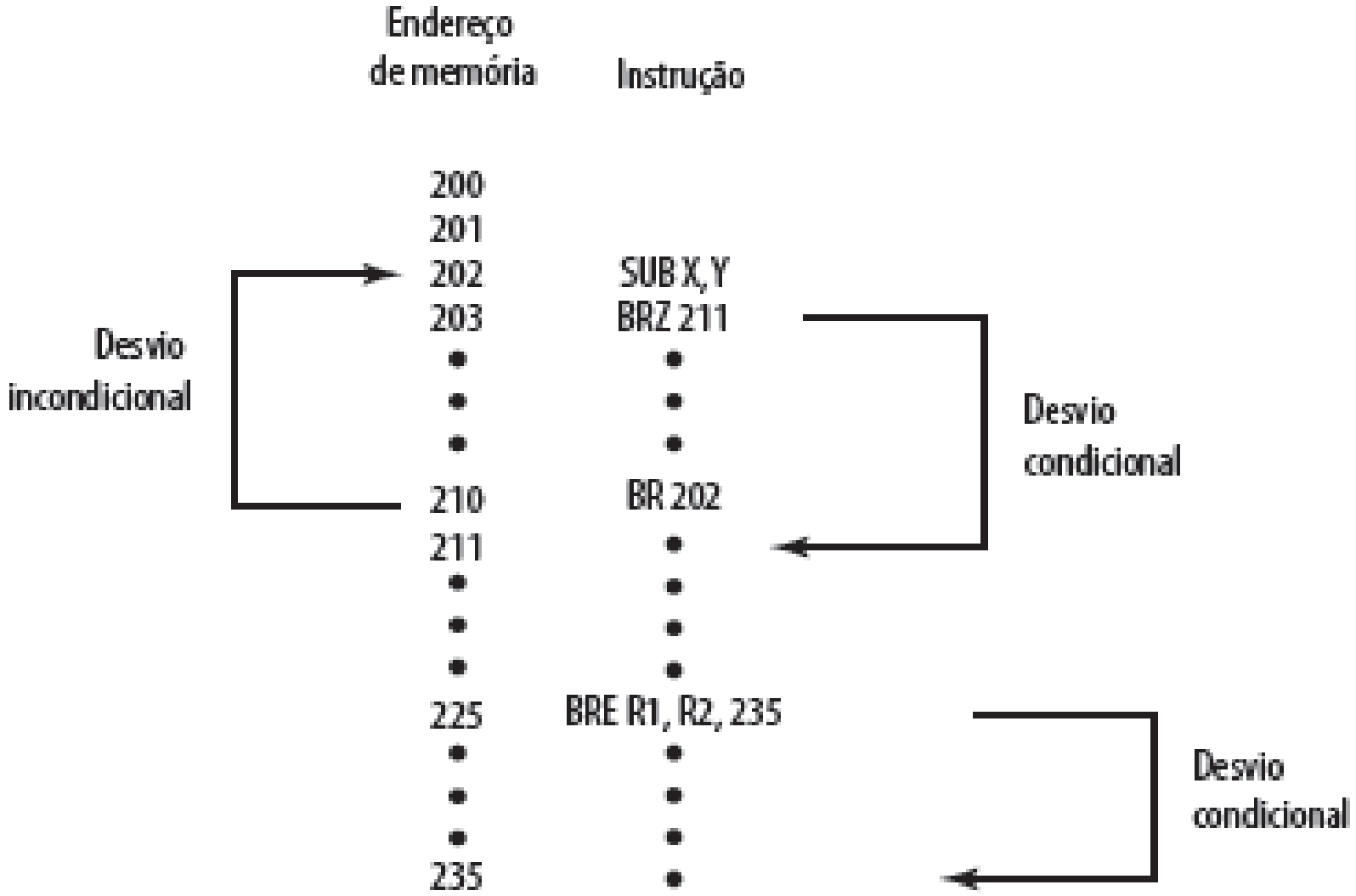
Tipos de operação

- Transferência de dados
- Aritmética
- Lógica
- Conversão
- E/S
- Controle do sistema
- **Transferência de controle**

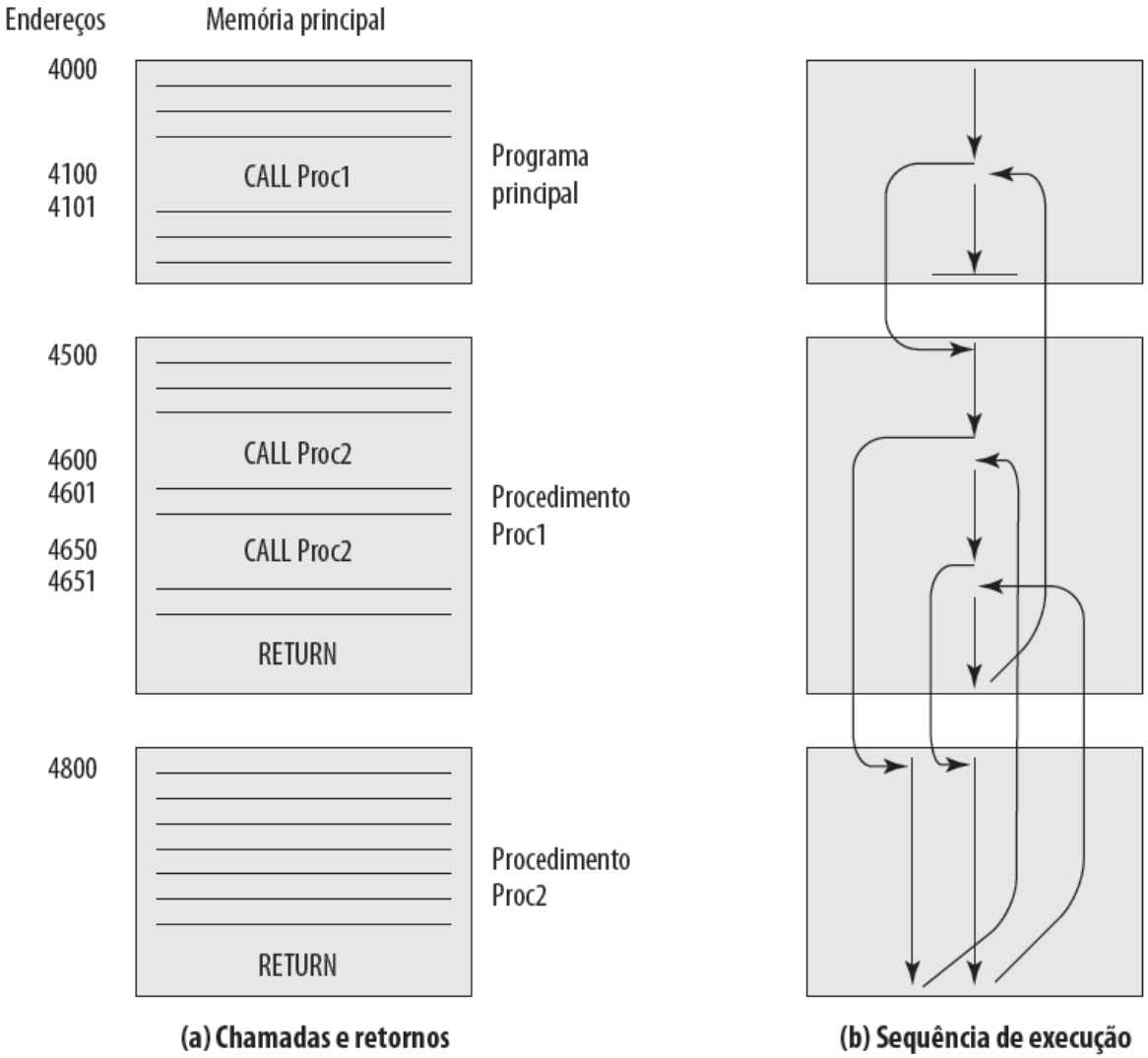
Transferência de controle

- Desvio:
  - E.g., desvio para endereço *x* se resultado for zero
  - E.g., incrementa e salta se for zero
  - Desvia para *x* incondicionalmente
- Chamada de sub-rotina:
  - Chamada de interrupção
  - *CALL*
  - *RETURN*

Instrução de desvio



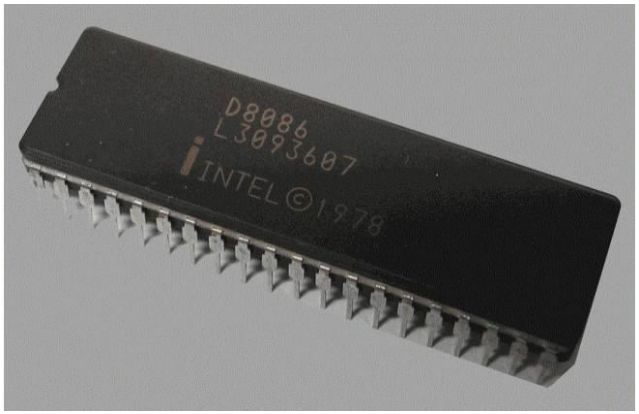
Chamada de subrotina



Objetivos

- ISA – *Instruction Set Architecture*
  - Definição
  - Formato e elementos
  - Endereços e registradores
  - Decisões de projeto
  - Tipos de instrução
  - Tipos de dados
  - Operações e OP Codes
- **Exemplos de instruções**
- Armazenamento (e*ndianess*)

Ex.: Tipos de operação do x86



Ex.: Tipos de operação do x86

Instrução	Descrição
Movimentação de dados	
MOV	Operação de movimentação entre registradores ou entre registrador e memória.
PUSH	Coloca operando na pilha.
PUSHA	Coloca todos os registradores na pilha.
MOVSX	Move byte, palavra, palavra dupla, com extensão de sinal. Move um byte para uma palavra, ou uma palavra para uma palavra dupla, com extensão de sinal para complemento de dois.
LEA	Carrega endereço efetivo. Carrega o deslocamento do operando de origem, em vez do seu valor, ao operando de destino.
XLAT	Tradução de tabela de pesquisa. Substitui um byte em AL por um byte de uma tabela de tradução codificada pelo usuário. Quando XLAT é executada, AL deverá ter um índice sem sinal para a tabela. XLAT muda o conteúdo de AL do índice de tabela para a entrada de tabela.
IN, OUT	Operando de entrada, saída do espaço de E/S.
Aritméticas	
ADD	Adiciona operandos.
SUB	Subtrai operandos.
MUL	Multiplicação de inteiro sem sinal, com operandos de byte, palavra ou palavra dupla, e resultado de palavra, palavra dupla ou quatro palavras.
IDIV	Divisão com sinal.
Lógicas	
AND	AND dos operandos.
BTS	Teste e definição de bit. Opera sobre um operando de campo de bit. A instrução copia o valor atual de um bit para o flag CF e define o bit original como 1.
BSF	Varredura de bit adiante. Varre uma palavra ou palavra dupla para o bit 1 e armazena o número do primeiro bit 1 em um registrador.
SHL/SHR	Deslocamento lógico à esquerda ou à direita.
SAL/SAR	Deslocamento aritmético à esquerda ou à direita.
ROL/ROR	Rotação à esquerda ou à direita.
SETcc	Define um byte como zero ou 1, dependendo de qualquer uma das 16 condições definidas pelos flags de status.

Ex.: Tipos de operação do x86

Transferência de controle	
JMP	Salto incondicional.
CALL	Transfere o controle para outro local. Antes da transferência, o endereço da instrução após o CALL é colocado na pilha.
JE/JZ	Salto se igual/zero.
LOOPE/LOOPZ	Loop se igual/zero. Esse é um salto condicional usando um valor armazenado no registrador ECX. A primeira instrução decrementa ECX antes de testá-lo para a condição de desvio.
INT/INTO	Interrompe/Interrompe se houver *overflow*. Transfere o controle para uma rotina de serviço de interrupção.
Operações de *string*	
MOVS	Move *string* de byte, palavra, palavra dupla. A instrução opera sobre um elemento de uma *string*, indexado pelos registradores ESI e EDI. Após cada operação de *string*, os registradores são automaticamente incrementados ou decrementados para apontarem para o próximo elemento da *string*.
LODS	Carrega byte, palavra ou palavra dupla de *string*.
Suporte a linguagem de alto nível	
ENTER	Cria um frame de pilha que pode ser usado para implementar as regras de uma linguagem de alto nível estruturada em bloco.
LEAVE	Reverte a ação de um ENTER anterior.
BOUND	Verifica limites de *array*. Verifica se o valor no operando 1 está dentro dos limites inferior e superior. Os limites estão em dois locais de memória adjacentes referenciados pelo operando 2. Uma interrupção ocorre se o valor estiver fora dos limites. Essa instrução é usada para verificar um índice de *array*.
Controle de flag	
STC	Define flag de *carry*.
LAHF	Carrega registro AH a partir dos flags. Copia bits SF, ZF, AF, F e CF para o registrador A.

Ex.: Tipos de operação do x86

Registrador de segmento	
LDS	Carrega ponteiro para DS e para outro registrador.
Controle do sistema	
HLT	Termina a execução do programa.
LOCK	Ativa uma suspensão na memória compartilhada de modo que o Pentium tenha uso exclusivo dela durante a instrução que vem imediatamente após o LOCK.
ESC	Escape para uma extensão do processador. Um código de escape que indica que as instruções seguintes devem ser executadas por um processador numérico que admite cálculos com inteiro e ponto flutuante de alta precisão.
WAIT	Espera até BUSY# negado. Suspende a execução do programa no Pentium até que o processador detecte que o pino BUSY esteja inativo, indicando que o coprocessador numérico terminou a execução.
Proteção	
SGDT	Armazena tabela de descritor global.
LSL	Carrega limite de segmento. Carrega um registrador especificado pelo usuário com um limite de segmento.
VERR/VERW	Verifica segmento para leitura/gravação.
Gerenciamento de cache	
INVD	Esvazia a memória cache interna.
WBINVD	Esvazia a memória cache interna após gravar linhas modificadas na memória.
INVLPG	Invalida uma entrada no *translation lookaside buffer* (TLB).

Objetivos

- ISA – *Instruction Set Architecture*
  - Definição
  - Formato e elementos
  - Endereços e registradores
  - Decisões de projeto
  - Tipos de instrução
  - **Tipos de dados**
  - Operações e OP Codes
- Exemplos de instruções
- Armazenamento (e*ndianess*)

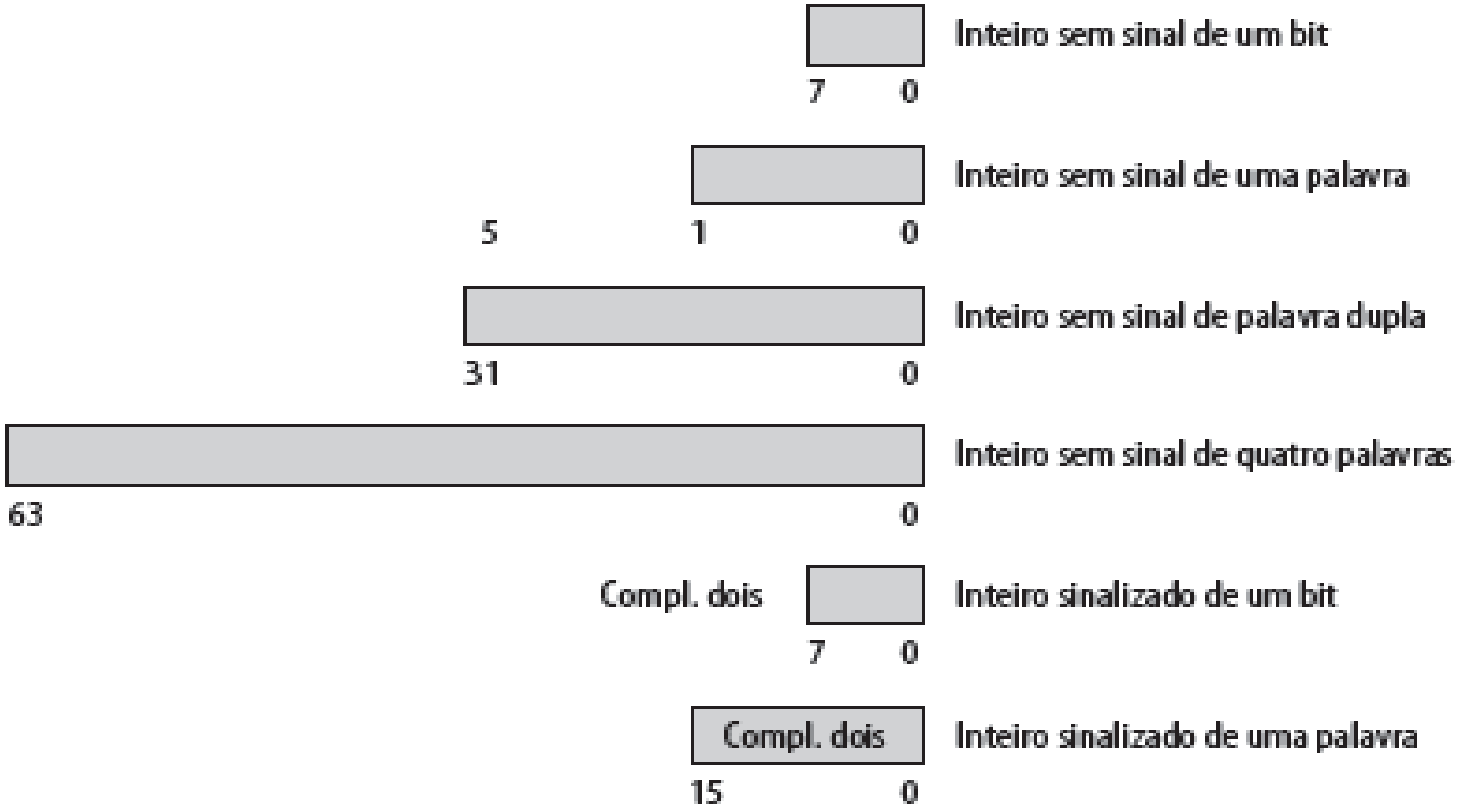
Tipos de operando

- Endereços
- Números:
  - Inteiro/ponto flutuante
- Caracteres:
  - ASCII, etc.
- Dados lógicos:
  - Bits ou flags

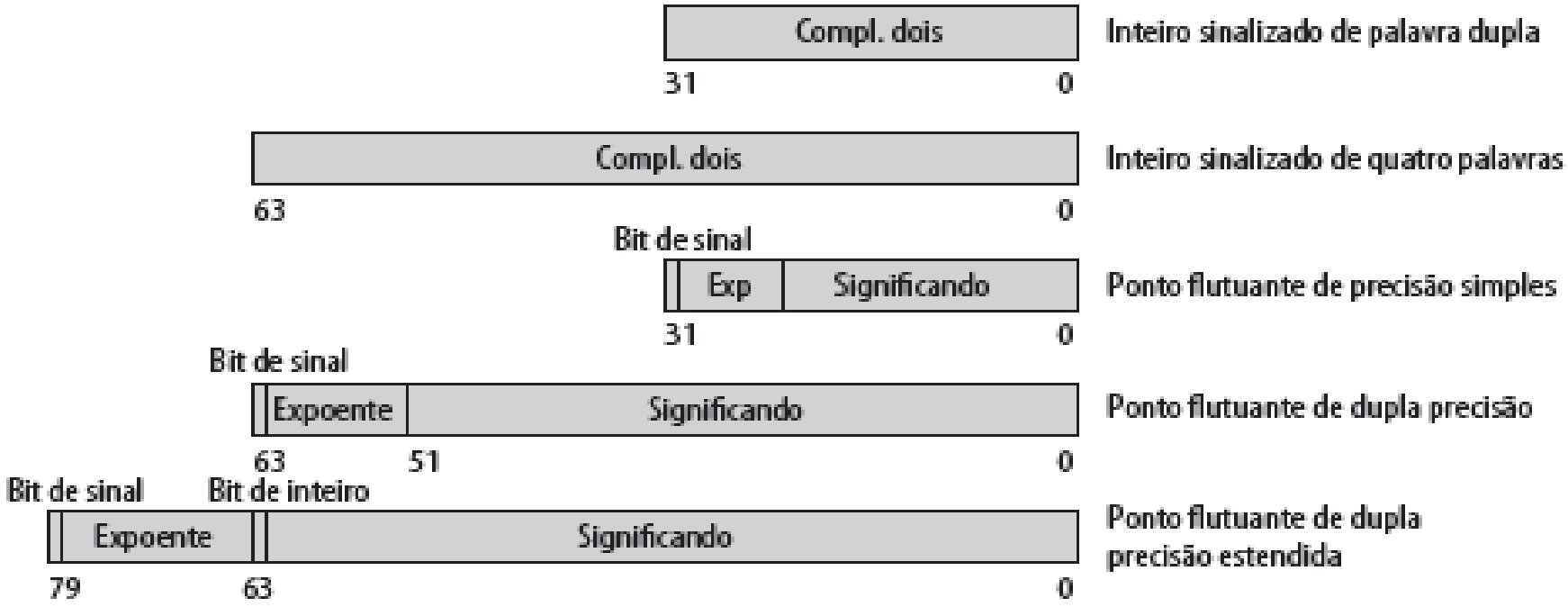
Ex.: Dados do x86

- Byte de 8 bits
- Word (palavra) de 16 bits
- Doubleword de 32 bits
- Quadword de 64 bits
- Palavras duplas de 128 bits

Formatos de dados numéricos do x86



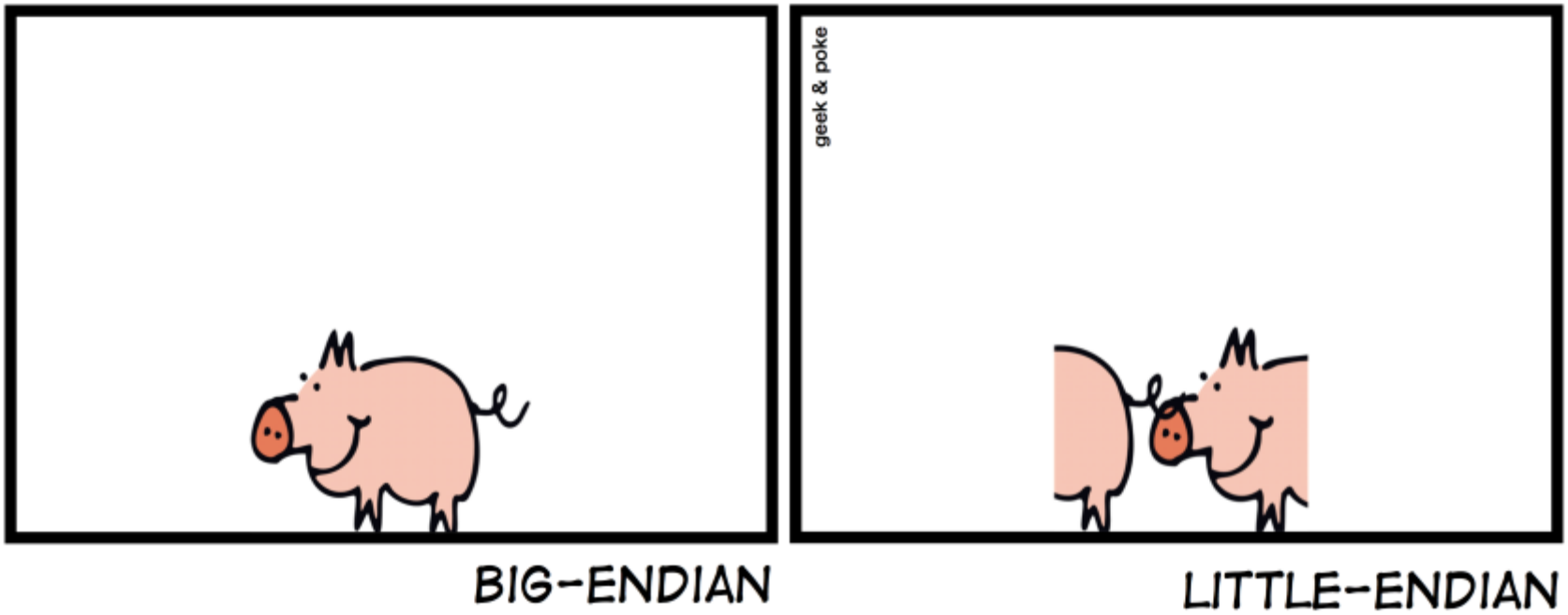
Formatos de dados numéricos do x86



Objetivos

- ISA – *Instruction Set Architecture*
  - Definição
  - Formato e elementos
  - Endereços e registradores
  - Decisões de projeto
  - Tipos de instrução
  - Tipos de dados
  - Operações e OP Codes
- Exemplos de instruções
- **Armazenamento (e*ndianess*)**

Ordem de bytes e endereços



Ordem de bytes e endereços

- Em que ordem lemos números que ocupam mais de um byte?
  - —"Endereços definem UM byte"
- Ex.: (números em hexa para facilitar a leitura)
  - —0x12345678 pode ser armazenado em 4 locais de 8 bits, das formas a seguir:

Ordem do byte (exemplo): 0 x 12345678

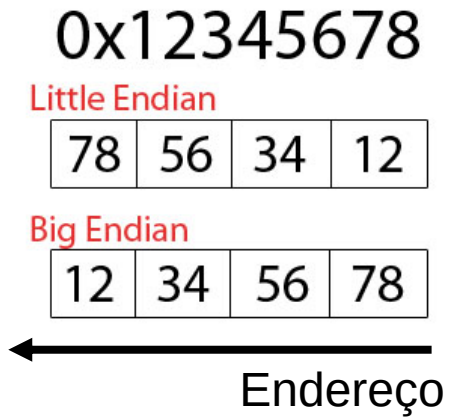
- Endereço Valor (1)
- 187 12
- 186 34
- 185 56
- 184 78

Ordem do byte (exemplo): 0 x 12345678

Endereço	Valor (1)	Valor (2)
187	12	78
186	34	56
185	56	34
184	78	12

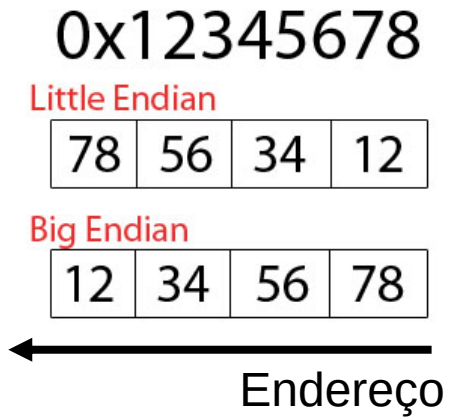
- Ou seja, escrever/ler de cima para baixo ou de baixo para cima?

Denominações da ordem de byte



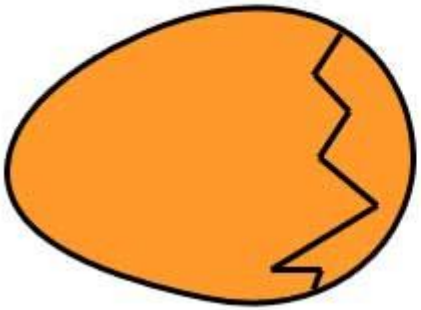
- O problema se chama *Endianness*
- O sistema à esquerda tem o byte **menos significativo** no endereço **mais baixo**
  - Isso é chamado de **big-endian**

Denominações da ordem de byte

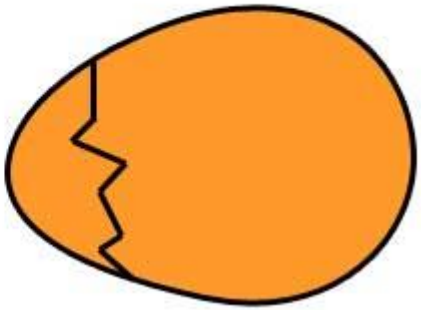


- O problema se chama *Endianness*
- O sistema à esquerda tem o byte **menos significativo** no endereço **mais baixo**
  - Isso é chamado de **big-endian**
- O sistema à direita tem o byte **menos significativo** no endereço **mais alto**
  - Isso é chamado de **little-endian**

Origem: Lilliput - Gulliver



BIG-Endian

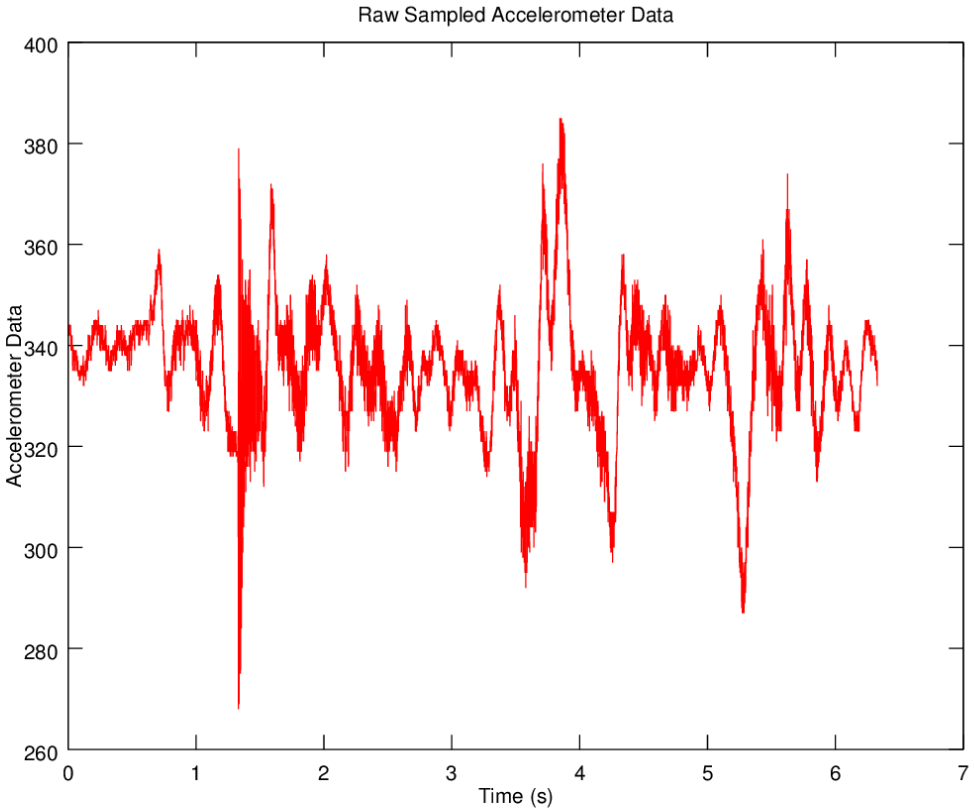


LITTLE-Endian

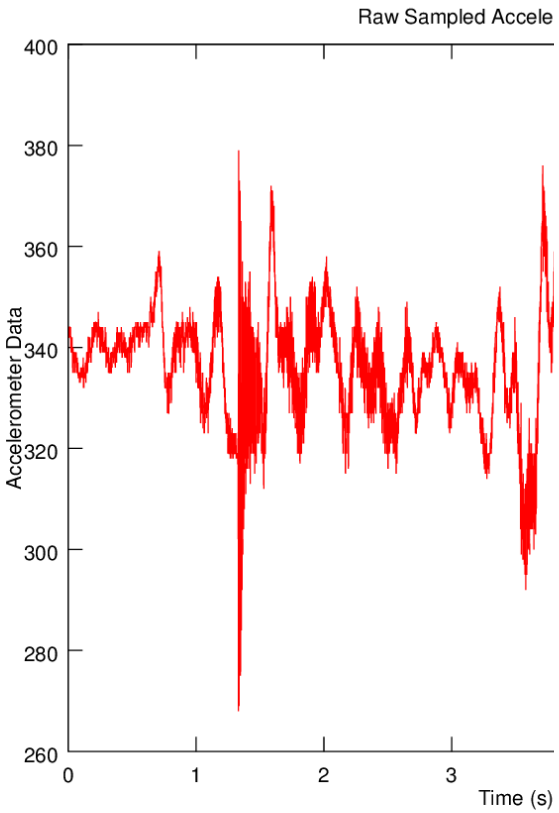
Padrão... Que padrão?

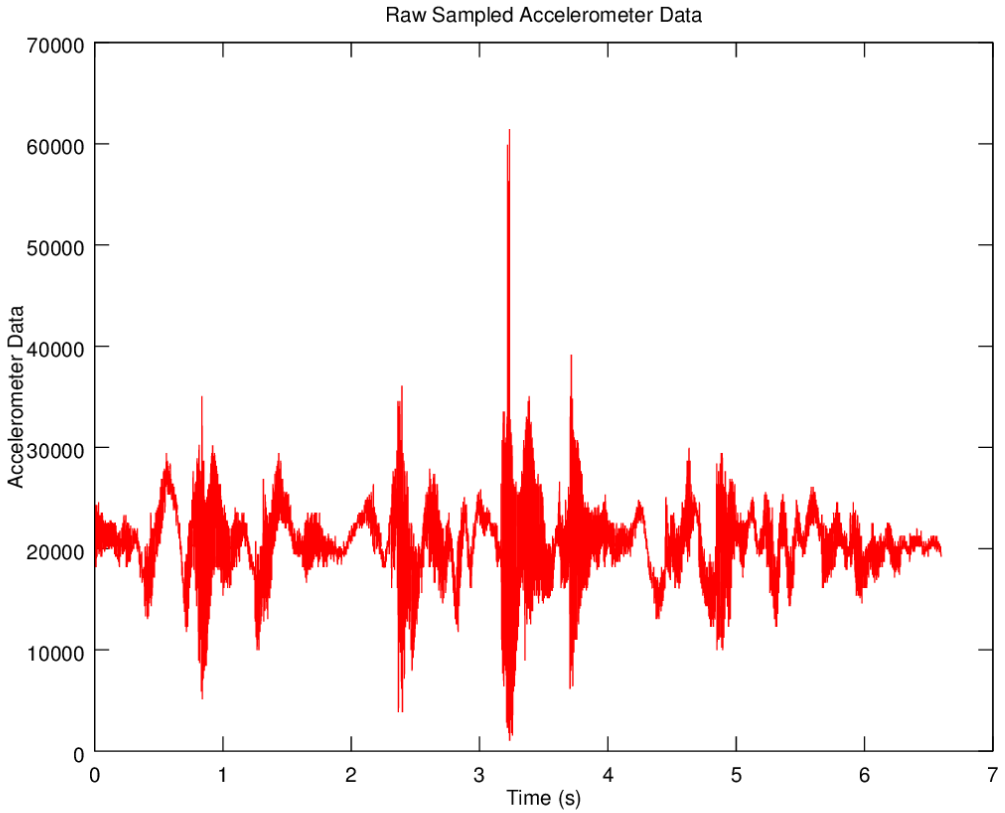
- Pentium (x86), VAX são little-endian
- IBM 370, Motorola 680x0 (Mac) e a maioria dos RISCs são big-endian
- Internet é big-endian
  - Torna a escrita de programas para Internet no PC mais desajeitada!

Problema de transmissão de dados entre sistemas



Problema de transmissão de dados entre sistemas





Suporte a endian no ARM

- Bit E no registrador de controle do sistema
- Sob controle do programa