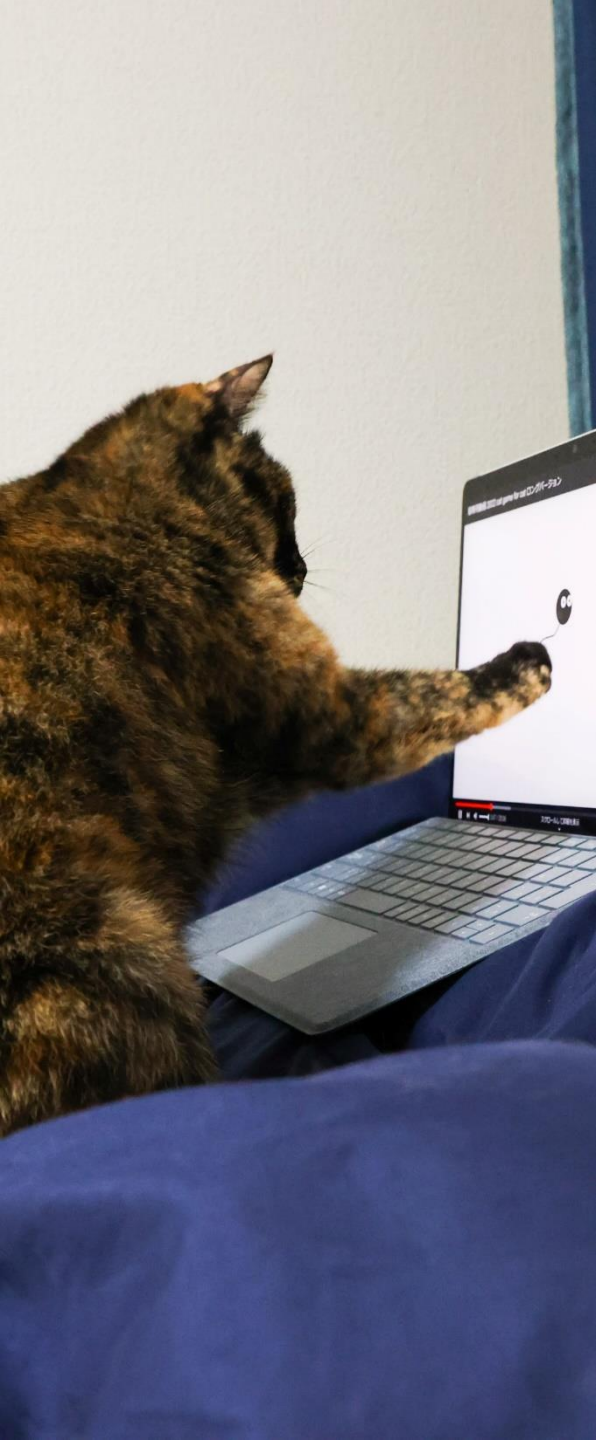
Performance em Sistemas Ciberfísicos

Aula 2 – Arquitetura ISA

Frank Coelho de Alcantara – 2023 -1



Arquitetura de Von Neumann

“Se as pessoas não acreditam que a matemática é simples é porque ainda não entenderam como a vida é complicada.”

John von Neumann

HISTÓRICO

1944: Projeto **EDVAC** (*Electronic Discrete Variable Automatic Computer*) incluindo algoritmos em memória;

1945: John von Neumann escreve um relatório de Pesquisa (*First Draft of a Report on EDVAC*);

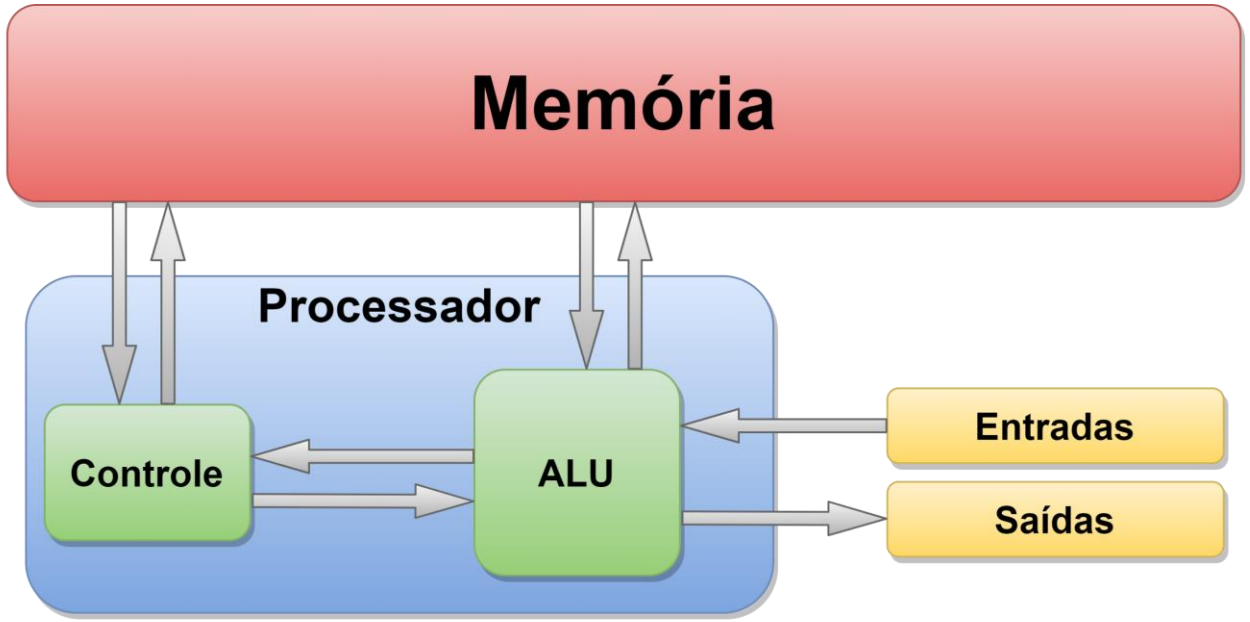
1951: Totalmente operacional com 6,000 válvulas, 12,000 diodos, e dois conjuntos de 64 relés capazes de armazenar 8 caracteres por linha, totalizando 1024 caracteres.

A IDEIA

"A ideia é que instruções (operações) e dados (operandos) de muitos tipos possam ser armazenados em memória como números."

John Von Neumann, 1944

ARQUITETURA DE VON NEUMANN



A Máquina executa um laço infinito de quatro ciclos:

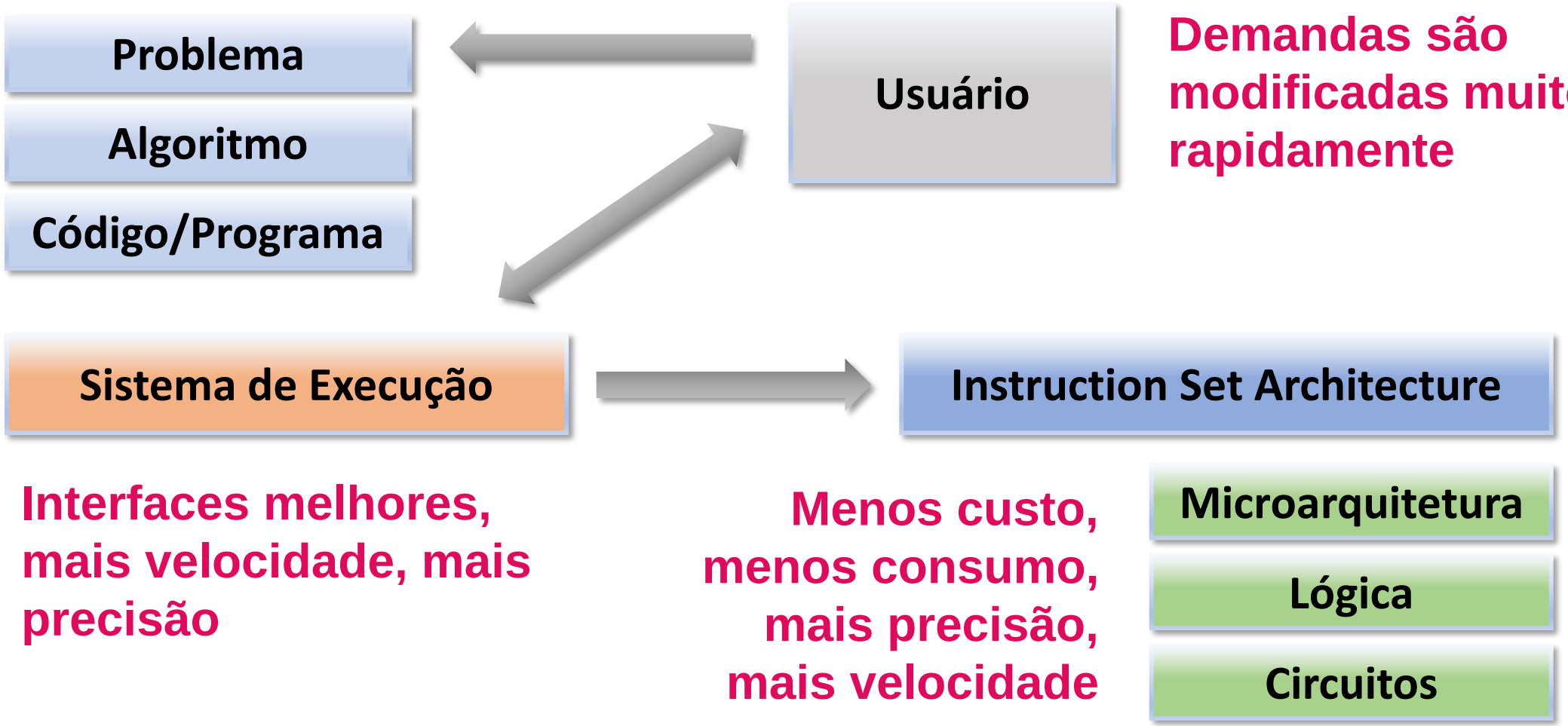
(1) Recuperação da Instrução;

(2) Decodificação da Instrução;

(3) Execução da Instrução;

(4) Registro dos Resultados.

Arquitetura Hoje



Arquitetura de Von Neumann - Estrutura

- Programas são armazenados em memória (***1024 words***):
  - As instruções e dados estão armazenados eum um conjunto linear de memória (***array***);
  - O significado de uma *word* depende de sinais de controle;
- As instruções são processadas sequencialmente:
  - Uma instrução de cada vez e sempre um ciclo complete;
  - Um contador, *Instruction pointer*, identifica a instrução corrente e é incrementado sequencialmente, no final da instrução;

Fluxo de Dados - *Dataflow*

- **Modelo Von Neumann**: uma instrução é recuperada e executada em um fluxo de controle ordenado pelo *instruction pointer,* de forma sequential;
- **Modelo Dataflow:** Uma instrução é recuperada e executada Segundo a ordem do fluxo de dados:
  - O *instruction pointer* ainda existe mas perdeu valor*;*
  - As instruções são executadas de acordo com as características de cada dado;

Fluxo de Dados - *Dataflow*

- As instruções são executadas quando a operação está pronta:
  - Cada instrução especifica quem deve receber o resultado e onde este resultado será armazenado;
  - Existe um potencial de paralelismo implícito neste conceito, o que permite que várias instruções sejam executadas ao mesmo tempo.

Fluxo de Dados – *Dataflow*

Problema

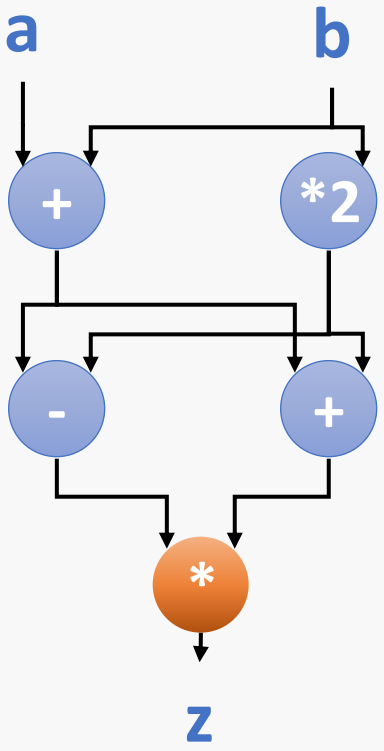
$$(a + b) - (2b) * (a + b) + (2b)$$

Qual das duas soluções é mais natural para você?

Von Neumann

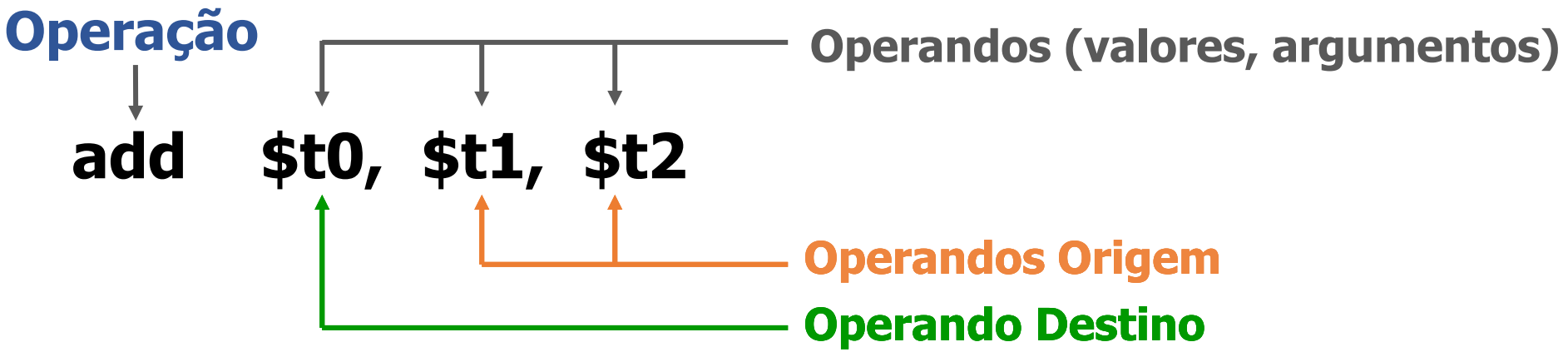
```
v ←a + b;
w←b * 2;
x←v - w
y←v + w
z←x * y
```

Dataflow



Anatomia de uma Instrução em Assembly

- Uma instrução é uma operação primitiva:
  - Uma instrução especifica uma operação e seus operandos;
  - Tipos de operandos incluem: imediato, fonte, destino.



Anatomia de uma Instrução em Assembly

- Uma operação é abreviada. Um *opcode* algo entre 1 e 4 caracteres. Com uma sintaxe regular e muito rígida:
  - Começa com um *opcode,* muitas vezes seguido pelo destino;
  - Geralmente as operações tem apenas três endereços.

add $t0, $t1, $t2  **(int) t0=t1+t2**

As instruções foram criadas para manter uma relação com as regras da matemática até onde for possível.

Entendendo a Máquina

As instruções são sequenciais, uma instrução só é iniciada quando a instrução anterior estiver terminada.

Entendendo a Máquina

Saber o que o programa faz lhe permite atribuir um nome significativo a função, uma *label.*

vezes7

Variáveis

$t0: w 2x 4x 8x

$t1: x 7x

$t2: y

$t3: z

Instruções

```
add $t0, $t1, $t1   1
add $t0, $t0, $t0   2
add $t0, $t0, $t0   3
sub $t1, $t0, $t1   4
```

Entendendo a Máquina

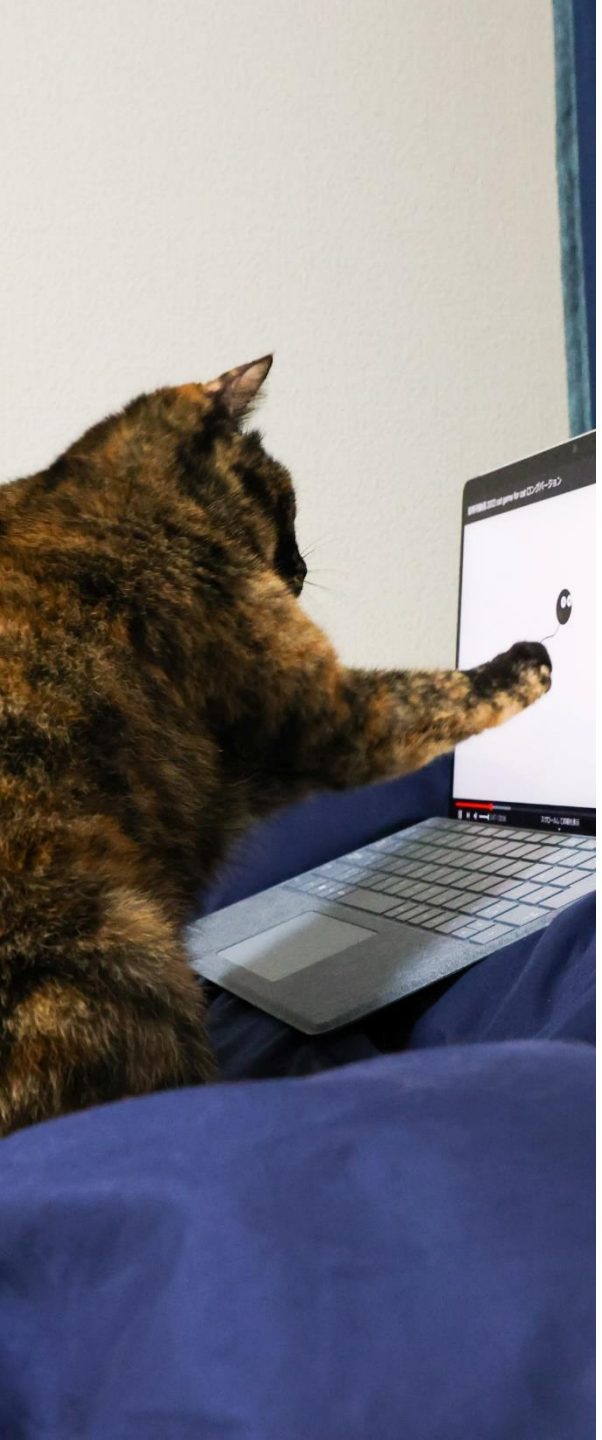
- Podemos controlar o fluxo com saltos, usando *labels*, ou endereços, como referência.
  - Instruções do tipo *goto*, ou *jump*, com condicionais ou não;
  - A *label* é apenas para os programadores, *labels* são endereços em memória.

Instruções

```
vezes7 : add $t0, $t1, $t1
         add $t0, $t0, $t0
         add $t0, $t0, $t0
         sub $t1, $t0, $t1
         JMP vezes7
```

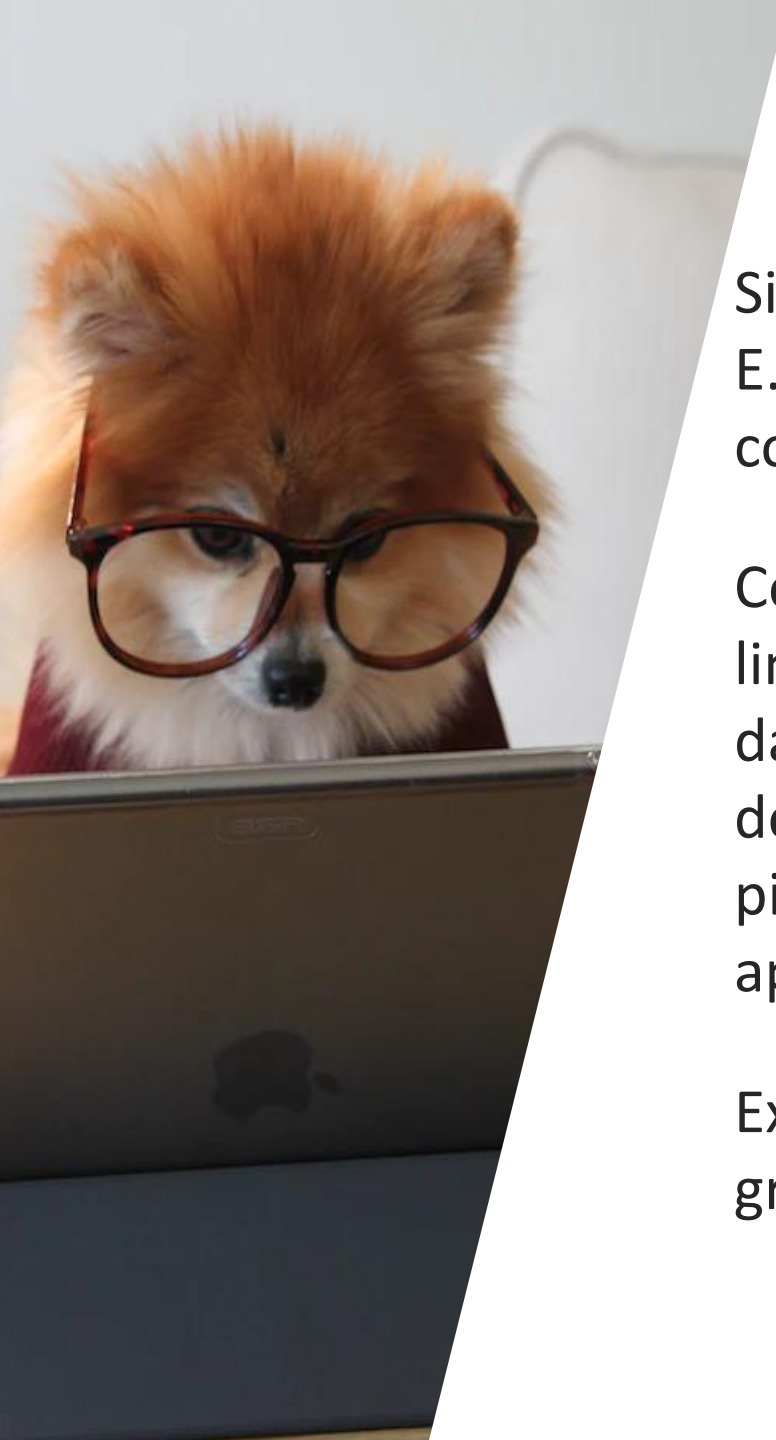
Considerações importantes

- Onde as instruções são armazenadas?
- Onde as variáveis são armazenadas?
- Como as *labels* são associadas a uma instrução?
- Como funcionam tipos compostos e complexos:?
  - Arrrays? Structures? Objetos?
- Como um programa começa?
- Como um programa para?



Atividade em sala 1

“A Linguagem C combina todo o poder da Linguagem Assembly com a faciliade de uso da Linguagem Assembly.”
***Mark Pearce**.*

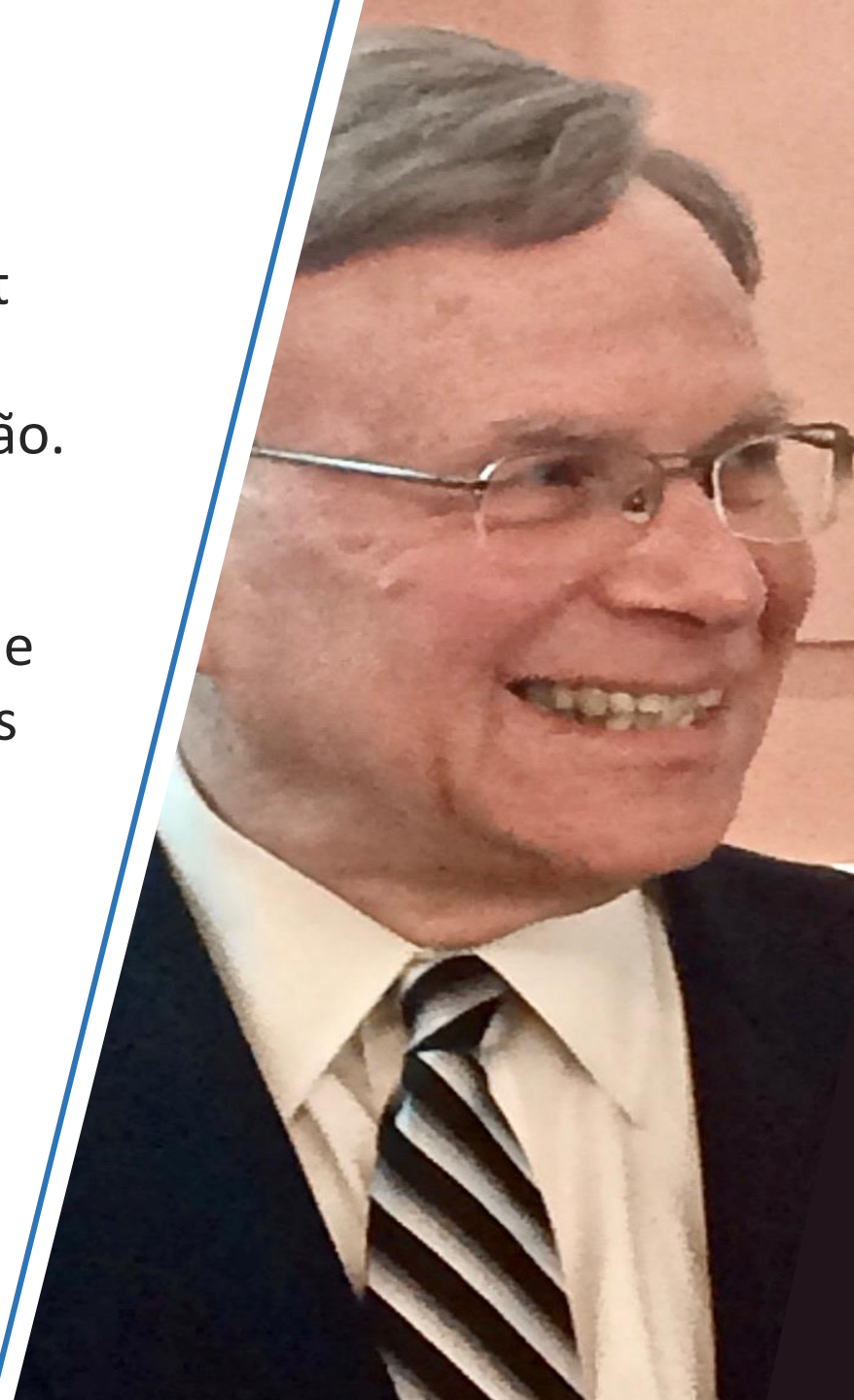


LITTLE MAN COMPUTER

Simulador criado em 1965 por Stuart E. Madnick para o ensino dos conceitos de algoritmos e computação.

Consiste, na verdade, de uma linguagem Assembly, muito simples, e das regras para o uso dos operadores desta linguagem. Uma máquina de pilha simples e efetiva para aprendizado.

Existem dezenas destes simuladores gratuitos na internet.



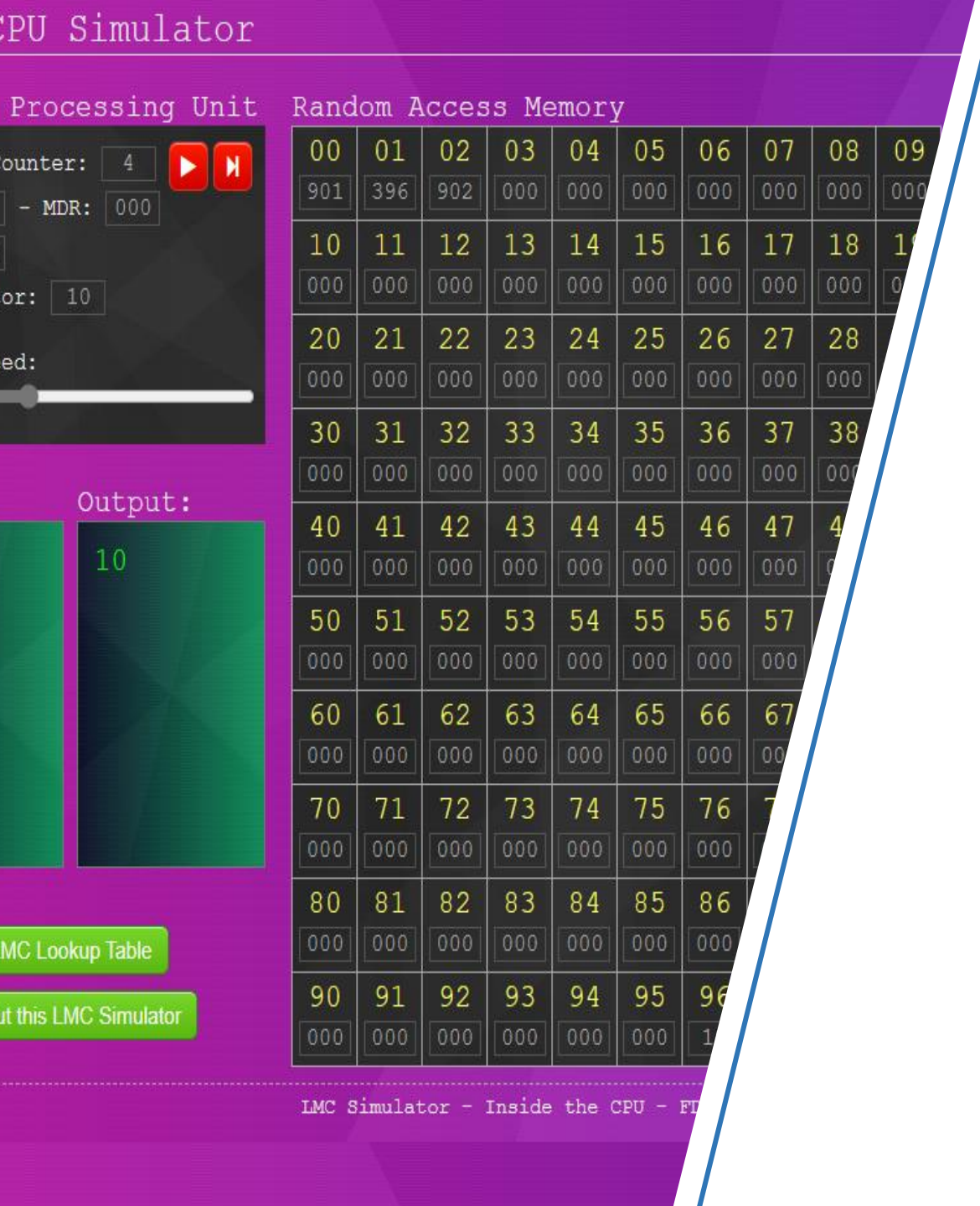
Nossa Linguagem Especial Para o LMC

MNEMONICO	Nome	Descrição	OP CODE
INP	INPUT	Recebe uma entrada do operador e armazena no Acumulador	901
OUT	OUTPUT	Devolve o resultado armazenado no Acumulador	902
LDA	LOAD	Carrega o Acumulador com o valor de um endereco de memória	5xx
STA	STORE	Armazena o Acumulador em um endereco de memória	3xx
ADD	ADD	Soma o conteúdo de um endereço de memória e armazena o resultado no Acumulador	1xx
SUB	SUBTRACT	Subtrai o conteúdo de um endereco de memória do Acumulador	2xx
BRP	BRANCH IF POSITIVE	Branch/Jump faz um *goto* se o conteúdo do Acumulador for positivo.	8xx
BRZ	BRANCH IF ZERO	Branch/Jump faz um *goto* se o conteúdo for zero.	7xx
BRA	BRANCH ALWAYS	Branch/Jump sempre faz um *goto*.	6xx
HLT	HALT	Para o código	000
DAT	DATA LOCATION	Usado para associar uma *label* a um endereco de memória. Um operador opcional pode ser usado para armazenar um valor no endereco de memória.	

Rodando programas no Simulador

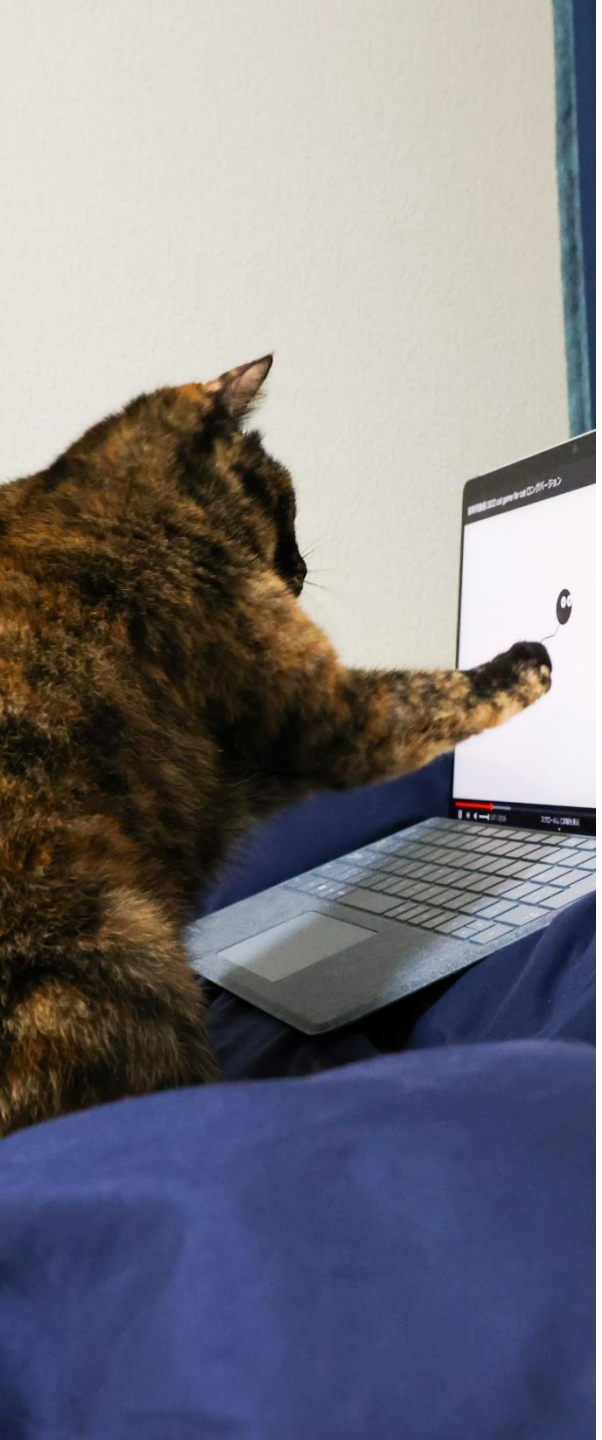
- Sua tarefa será fazer dois programas em um simulador do *Litte Man Computer, online:*
  1. Um programa para receber dois inteiros do usuário e imprimir o resultado da soma destes inteiros;
  2. Um programa que fornece o maior entre dois inteiros digitados pelo usuário;
  3. Um programa que recebe três inteiros e coloca eles em ordem na saída.

Estes exercícios não têm peso na média da disciplina, contudo são indispensáveis para aprovação. Este conhecimento será útil nas atividade avaliativas.



TEREMOS QUE USAR A WEB

Little Man Computer



Instruction Set Archtecture

“A Natureza está escrita na linguagem da Matemática.”
Galileo Galilei

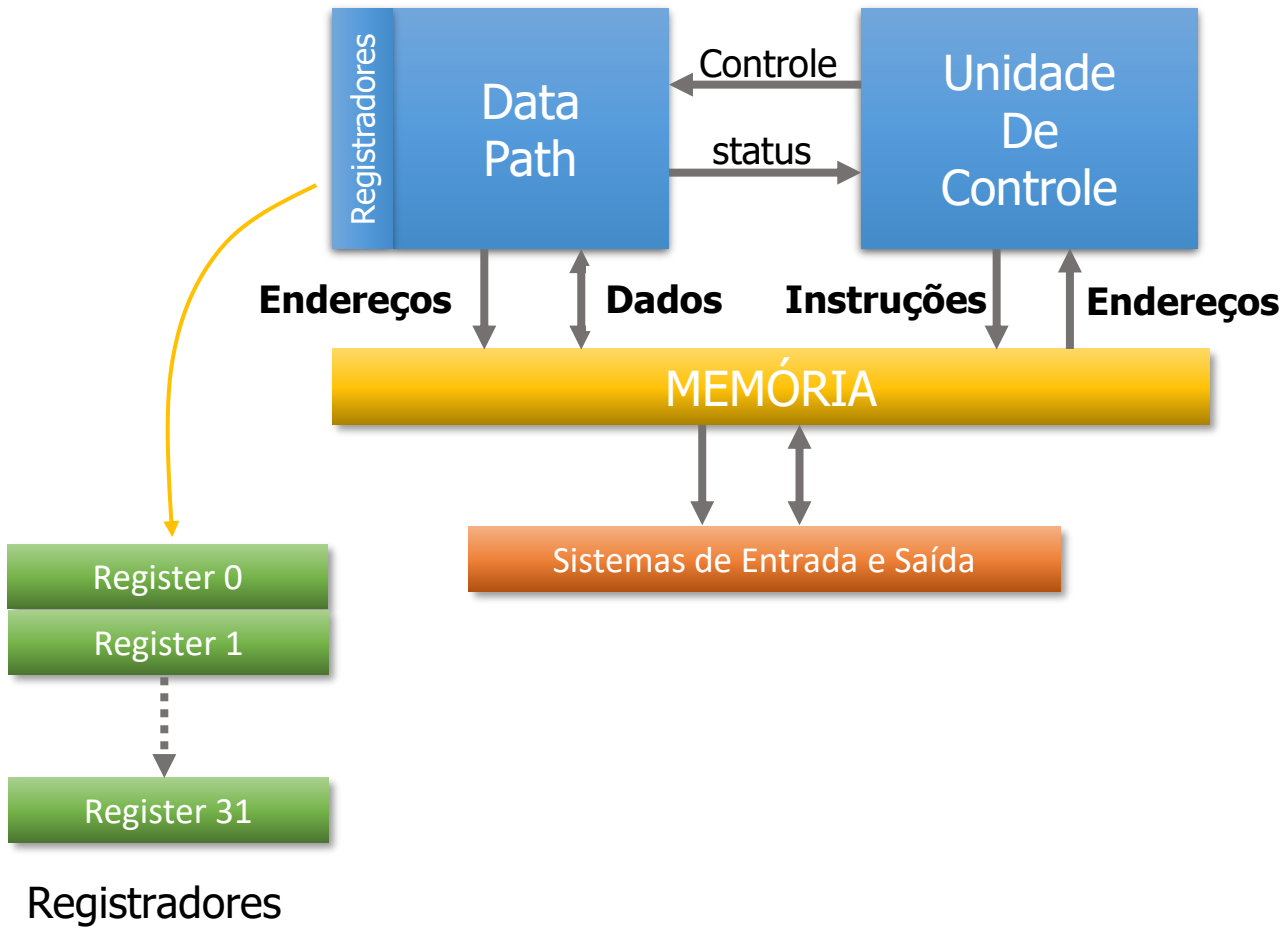
Instruction Set Architecture (ISA): definição

- A parte da arquitetura relacionada com a programação. Inclui os tipos de dados nativos, as instruções, registradores, modos de endereçamento, estrutura de memória, interrupções, gerenciamento de erros e interfaces de entrada e saída.
- Notadamente, a especificação do conjunto de *opcodes*, característico de um determinado processador.

Instruction Set Architecture (ISA): considerações

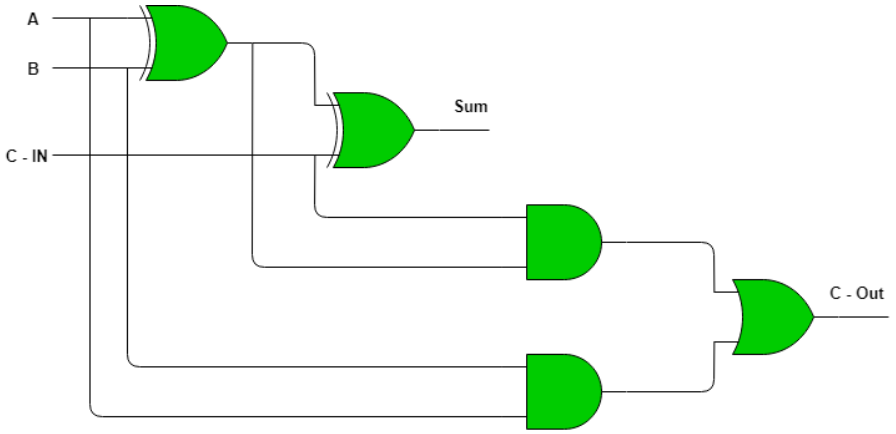
- Desempenho, tamanho, metáfora e compreensão;
- Complexidade:
  - Quantas instruções? A que nível (arrays? Multimídia?)?
  - RISC *(Reduced Instruction Set)*: instruções simples;
- Uniformidade: todas as instruções devem ter o mesmo tamanho? Levar o mesmo tempo?

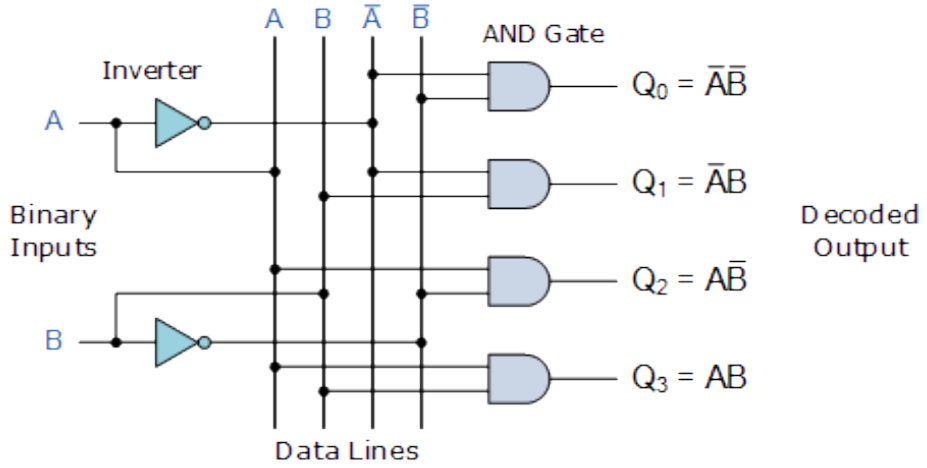
O que chamamos de Arquitetura de Von Neumann

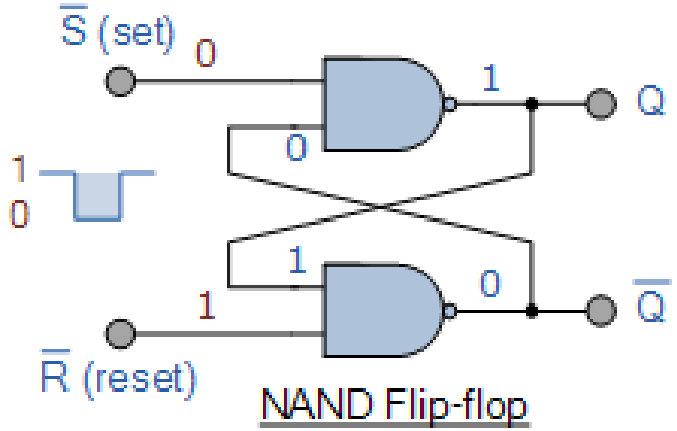


- Instruções são codificadas em binário;
- Existe um Program Counter (PC) com o endereco da próxima instrução;
- A Unidade de Controle possui os circuitos que vão traduzir instruções em sinais de controle para o *Data Path.*

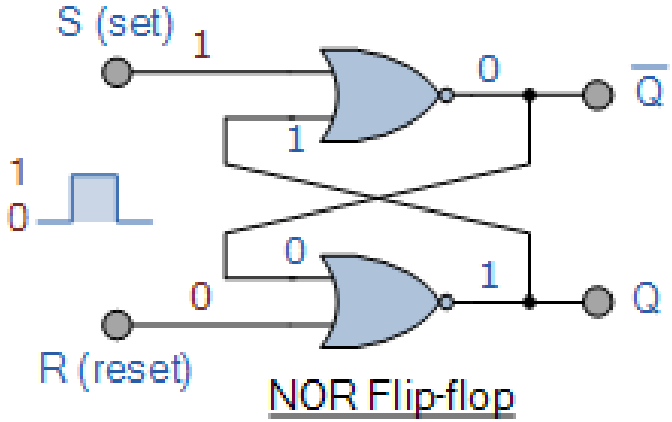
Circuitos Lógicos Exemplos







NAND Flip-flop



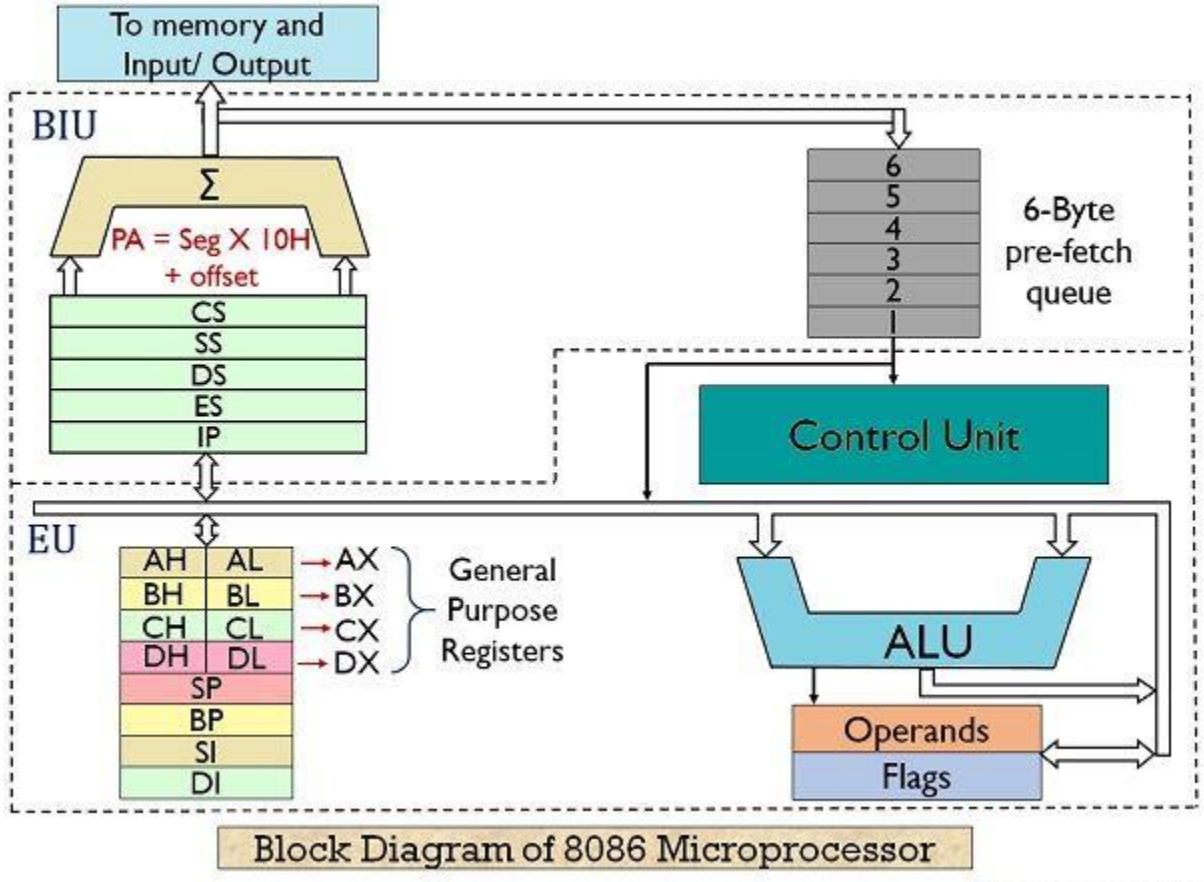
NOR Flip-flop

Tipos de ISA

- Stack (pilha)
  - Operandos ordenados implícitamente (topo da pilha)
  - Uso em ambientes que requerem muitos acessos a memória;
  - Poucos operandos e pouca Liberdade de controle de fluxo.
- Accumuladores
  - Registro de operando implícito (“accumulator”);
  - Uma, e somente uma, memória de operando (LMC);
  - Pouca memória para cálculo.

Tipos de ISA

- General-purpose register
  - Múltiplos registradores;
  - Liberdade no uso de operandos;
  - Memória de cáculo flexível.



Arquiteturas de Registro de Uso Geral - GPR

- Memory-memory (CISC)
  - Permite que operandos estejam nos registradores.
- Register-memory (CISC)
  - x86 – um operandos em registro - registro ou registro - memória.
- Register-register
  - Muito moderno, muitos registradores, muito rápido.
  - Hardware mais caro, mas mais simples. Muita flexibilidade para o compilador

De quantas formas Podemos fazer C = A + B

Stack

```
PUSH A
PUSH B
ADD
POP C
```

Accum

```
LOAD A
ADD B
STORE C
```

Reg-Reg

```
LOAD R1,A
LOAD R2,B
ADD
R3,R1,R2
STORE R3,C
```

Mem-Mem

```
ADD C,A,B
```

Reg-Men

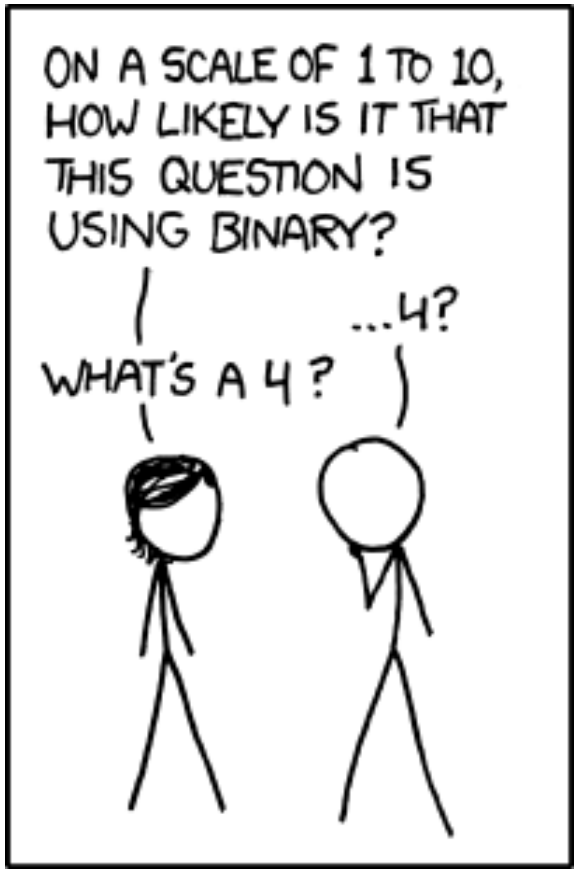
```
LOAD R1,A
ADD R1,B
STORE R1,C
```

Estruturas
Alto Desempenho

Máquinas
Modernas

Endereçamento

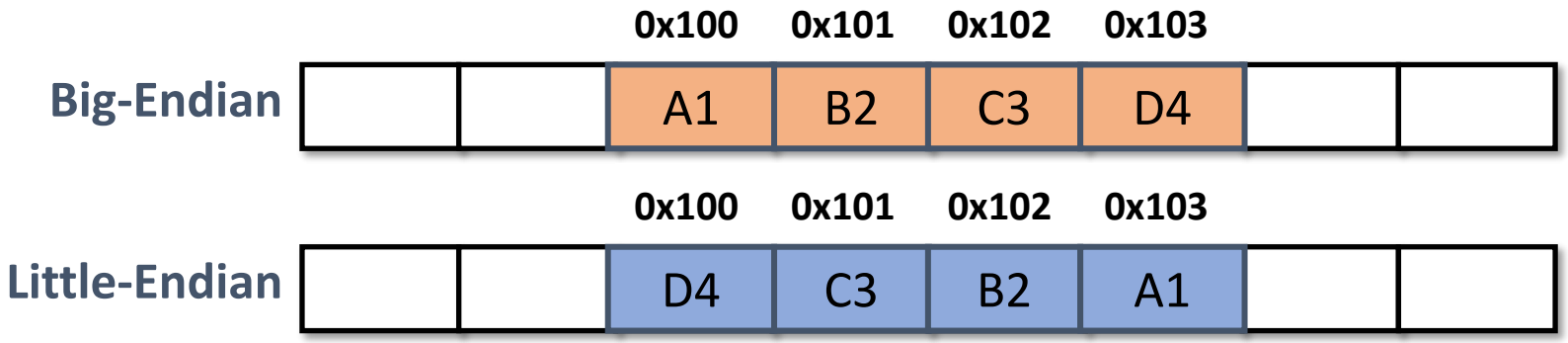
- Começamos com *Word Addressing*, como no LMC.
- A IBM criou os *bytes* de oito *bits* no IBM 360.
- Já tentamos endereçamento por bit. Não deu certo.
- Dados menores que um *byte* são transparentes para o programador, o processador resolve os problemas de endereçamento.



xkcd: 1 to 10

Ordem dos Bytes em memória

- Como os bytes de uma determinada *word* devem ser colocados na memória*?* Sem dúvida queremos que os bytes consecutivos fiquem em endereços consecutivos. Por convenção a ordem dos bytes de uma *word* em memória é chamada *endianness.* As duas opções são *big-endian* e *little-endian*: considere 0xA1B2C3D4 uma palavra de 4-bytes



Ordem dos Bytes em memória

- Big-endian (SPARC, z/Architecture)
  - O byte menos significativo tem o maior endereço.
- Little-endian (x86, x86-64)
  - O byte menos significante tem o menor endereço.
- Bi-endian (ARM, PowerPC)
  - Você pode especificar.

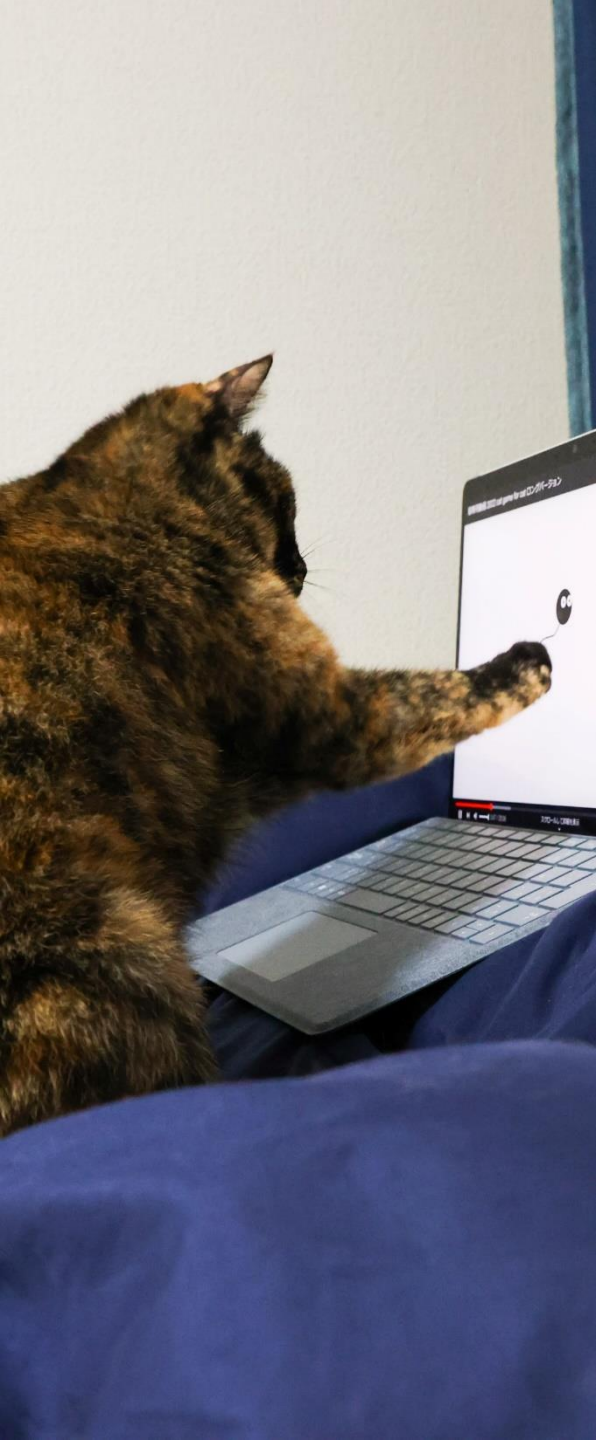
xkcd: Formatting Meeting

Endian-ness

- O nome é “Endian”, not “Indian”. Referencia ao Livro As viagens de Guliver.
- Little-Endian foi inventado pela Digital Equip. para o PDP-11:
  - Muito bom para matemática, muito ruim para humanos:
    - Deveria ter sido banido da face da terra. Não foi.
  - Algumas arquiteturas podem trocar de Endian-ness:
    - Uma ideia muito mais estúpida que a original.

Endian-ness

- A maior parte do tempo você pode ignorar a Endian-ness. A exceção são os sistemas de alto desempenho:
  - O hardware coloca os bytes no lugar certo e o compilador gera o comportamento esperado.
- A expressão “A maior parte do tempo” é sua inimiga:
  - Problemas quando acessamos arquivos de tipos diferentes.
  - Precisamos *debugar* alguma coisa ou Precisamos traduzir para código de máquina na mão.



Exercícios Sugeridos

“A Natureza está escrita na linguagem da Matemática.”
Galileo Galilei

Conhecendo sua Linguagem de Programação 1

Todas as linguagens de programação dignas deste nome permitem a manipulação de bits. Chamamos a estas operações de *bitwise operations.* O Python não é diferente.

1. Crie uma função que recebe dois inteiros como argumento, *valor* e *n* e devolve *valor* com o enésimo bit invertido.
2. Crie uma função que troque o valor de duas variáveis sem usar uma variável temporária ou operações aritméticas.

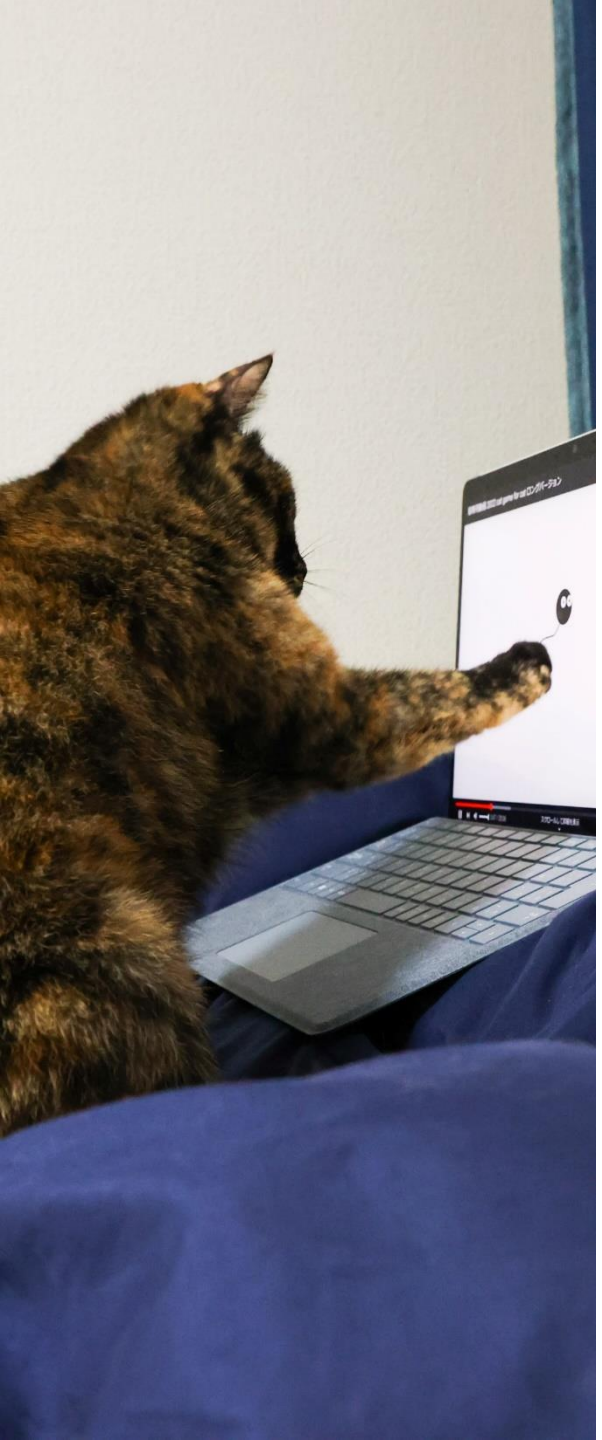
Estes exercícios não têm peso na média da disciplina, contudo são indispensáveis para aprovação. Este conhecimento será útil nas atividade avaliativas.

Conhecendo sua Linguagem de Programação 2

Sistemas diferentes usam padrões diferentes de Endian-ness. Pior que isso, informações trocadas via rede, entre sistemas e protocolos diferentes, podem estar sendo manipuladas por sistemas diferentes. Sua tarefa será descobrir os comandos em Python que permitem determinar a Endian-ness do seu computador e estudar como o módulo *struct* permite trocar a Endian-ness do buffer definido por:

```
buf = b'\x01\x00\x00\x00\x02\x00\x00\x00\x02\x00\x00\x00\x02\x00\x00\x00'
```

Estes exercícios não têm peso na média da disciplina, contudo são indispensáveis para aprovação. Este conhecimento será útil nas atividade avaliativas.



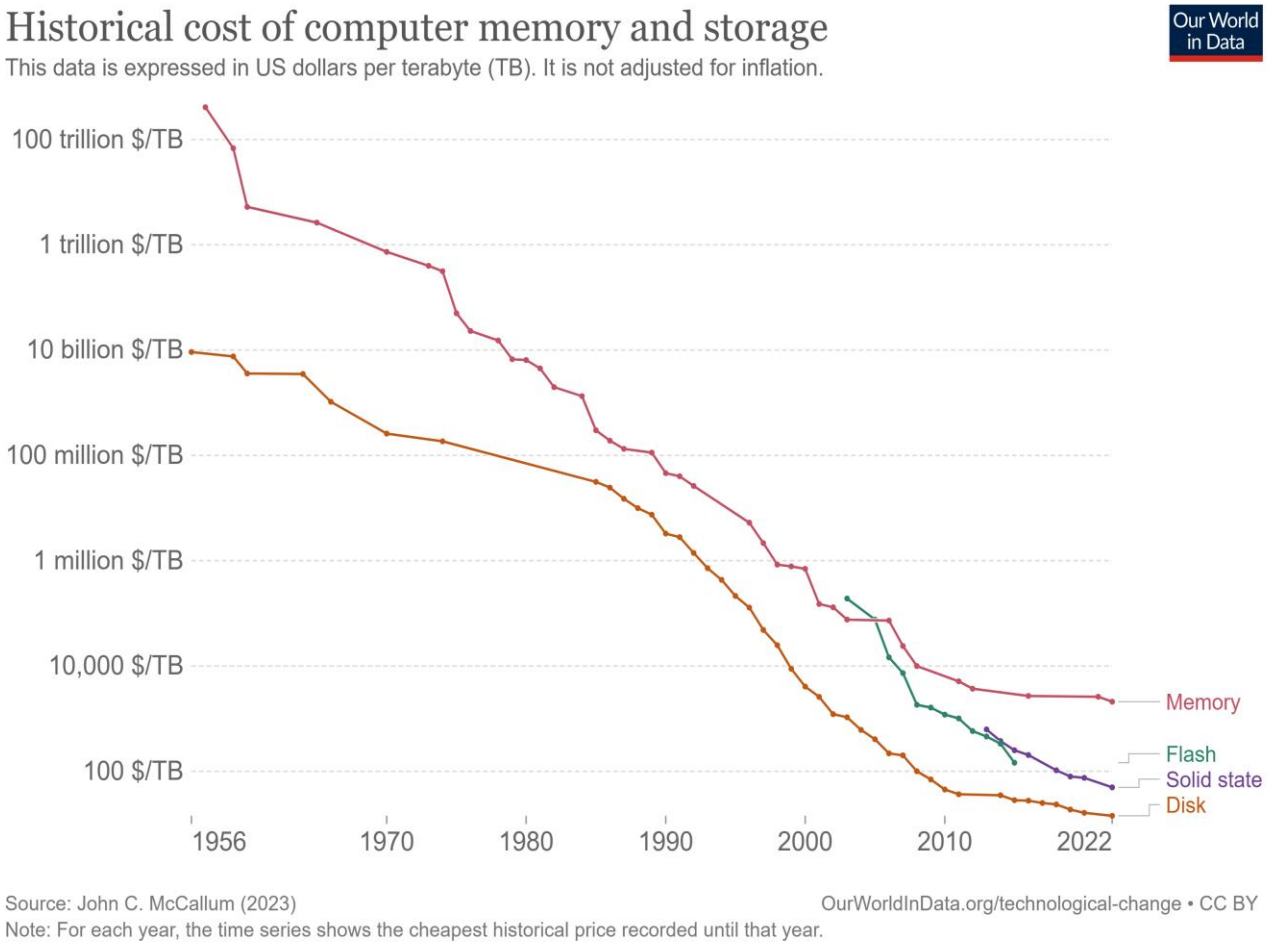
Hierarquia de Memória

“Ninguém nunca irá precisar de mais que 637kb de memória em um computador pessoal.”
Bill Gates

Memória Ideal

- Tempo de acesso nulo: não existe nenhum atraso entre a necessidade do dado e a sua disponibilidade;
- Capacidade infinita: não existem limites para a quantidade de dados, ou instruções, que possa ser armazenado na nossa memória;
- Custo zero: os dispositivos, ou soluções de armazenamento, não devem ter impacto no custo da solução que estamos desenvolvendo;
- Bandwidth Infinita: ser capaz de suportar um número infinito de acessos em paralelo.

Hierarquia de Memória



xkcd: Pointers

Conceitos importantes

- **Registradores:** são espaços limitados de memória, geralmente capazes de armazenar apenas uma unidade de informação que estão localizadas fisicamente muito próximas a CPU e graças a isso são muito rápidas, com tempos de reposta na ordem dos picosegundos e muito caras. Estas memórias são limitadas de acordo com cada arquitetura. Por exemplo: as cpus ARM de 64 bits possuem 31 registradores, a arquitetura **Intel Itanium** possui 128 e a arquitetura CUDA permite a configuração de até 255 registradores por thread .

Conceitos importantes

- **Memória:** a partir deste ponto, os termos memória e DRAM serão utilizados livremente para indicar a mesma área de armazenamento. Uma área de armazenamento externa a cpu, em estado sólido, com capacidade que varia dos 4 G Bytes aos 256 G Bytes, com tempos de resposta que variam entre 50 e 100 nanosegundos.
- Alguns autores, como Tanenbaum se referem a esta área de armazenamento como memória principal.

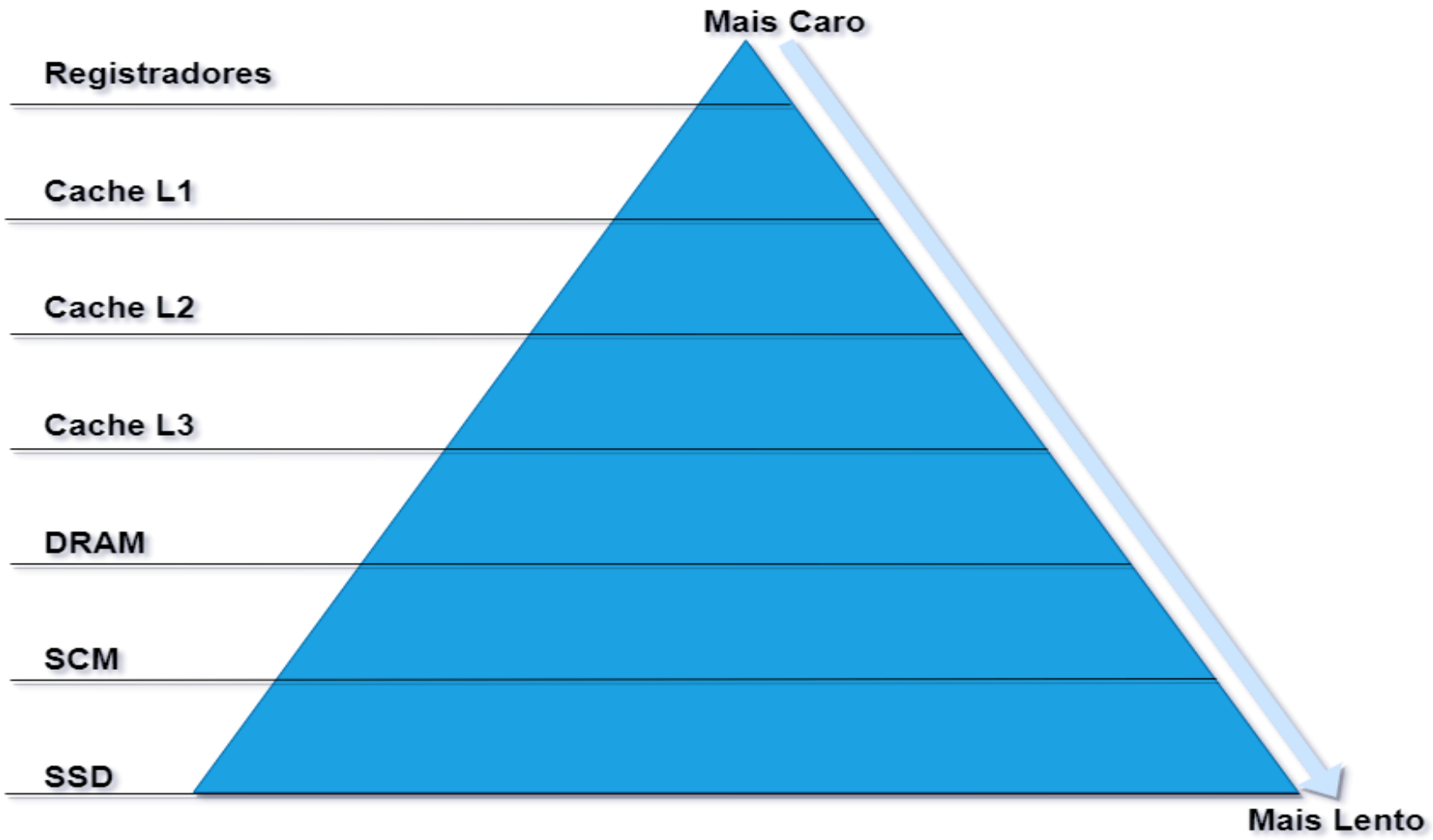
Conceitos importantes

- **Hard Drivers:** ou disco rígido, são dispositivos eletromagnéticos para o armazenamento de grandes quantidades de dados a custo baixo. Alguns autores se referem a esta área de armazenamento como memória secundária. Tipicamente os discos rígidos armazenam entre 1 e 128 T Bytes com tempos de acesso variando entre 5 e 10ms.
- **SSD:** dispositivos que armazenamento que utilizam a tecnologia Flash – RAM. São considerados como sendo memória secundária com tempos de acesso entre 100 e 200 microsegundos.

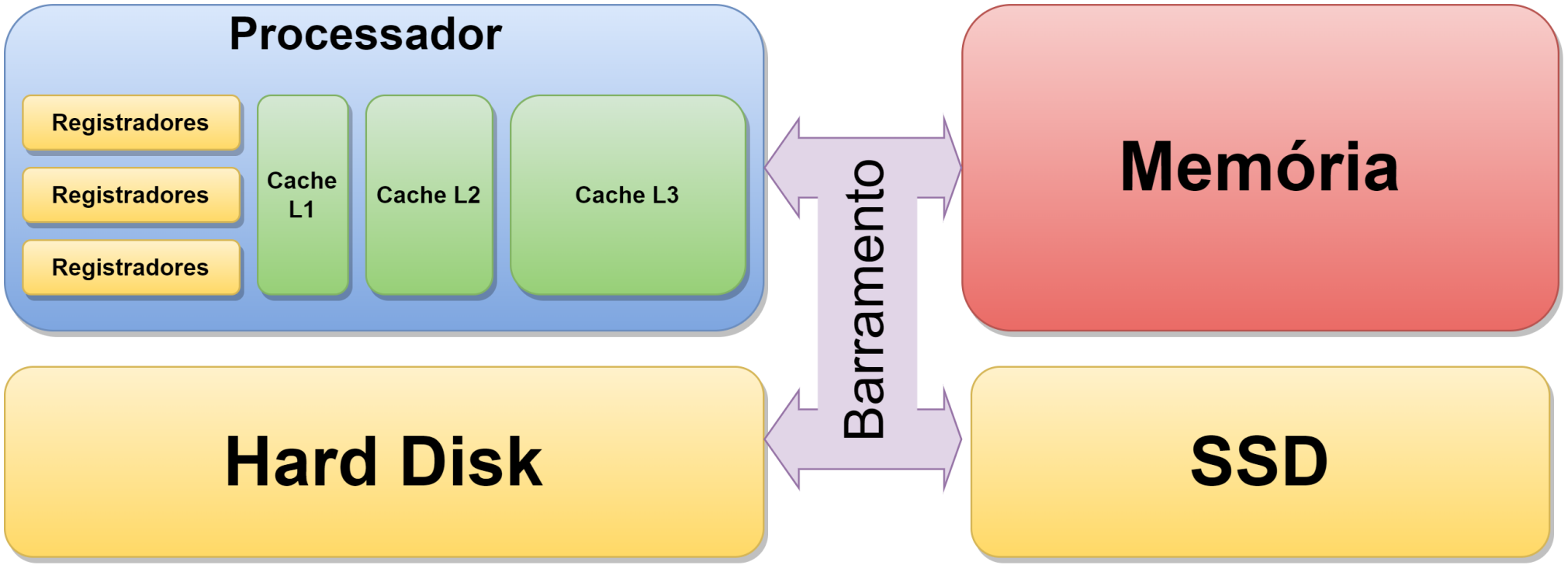
Conceitos importantes

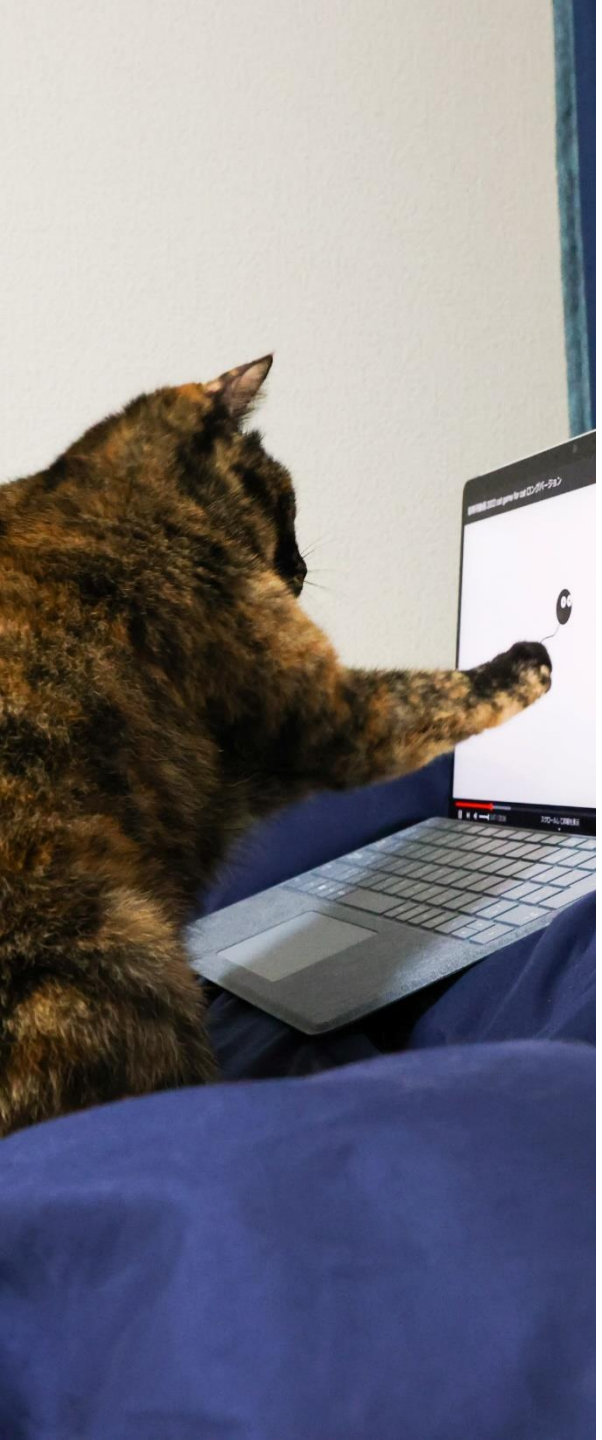
- **Cache:** usamos a palavra cache para definir uma estrutura de hardware, ou software, que vai armazenar dados, ou instruções, que possam ser acessados de forma rápida.
- Nas arquiteturas modernas existem até três níveis de:
  - **caches L1** permitem o armazenamento de até 64K Bytes com velocidades próximas a 1 nanosegundo;
  - **caches L2** até 256 K Bytes tempos de resposta entre 3 e 10ns.
  - **caches L3** armazenam entre 16 e 64 M Bytes com tempos de resposta entre 10 e 20 ns.

Hierarquia de Memória



Cache


Processador
Registradores
Registradores
Registradores
Cache L1
Cache L2
Cache L3
Barramento
Memória
Hard Disk
SSD



Exercício Preparatório

“Estar prepardo é a metade da vitória.”
Miguel de Cervantes

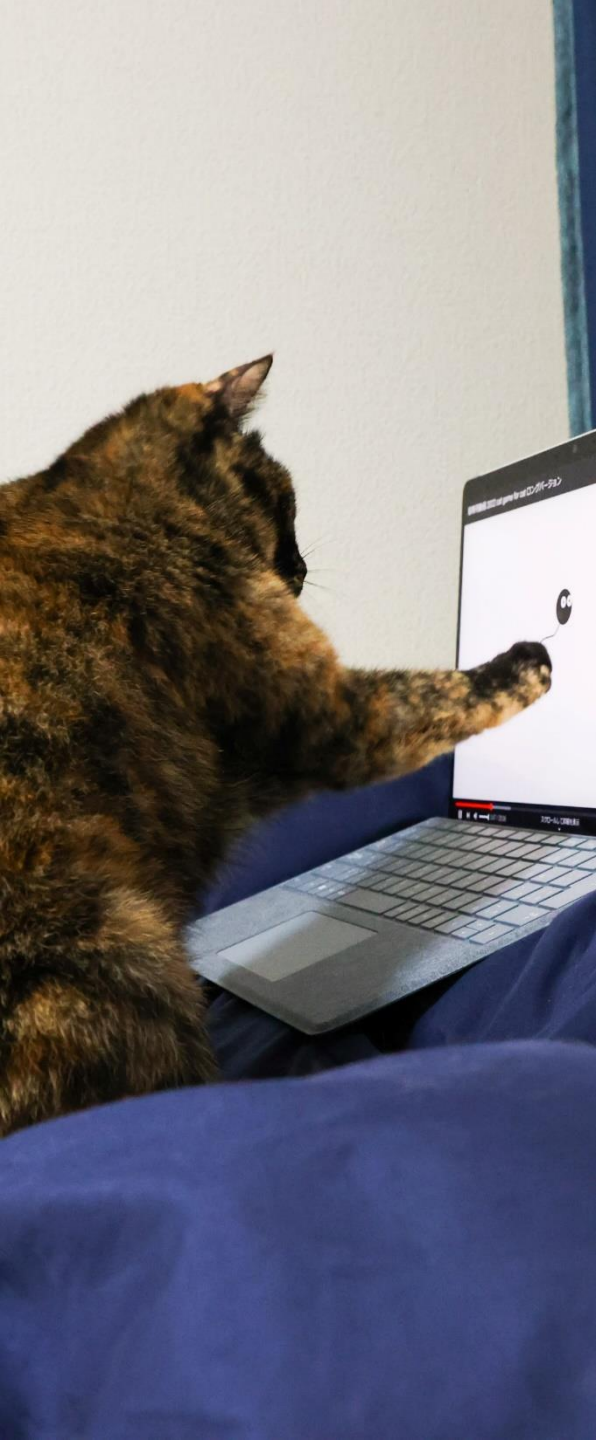
Dados

Escolha um site de livros gratuitos na internet e realize o seguinte exercício: baixe um livro que tenha mais de cem páginas, usando uma requisição *https* e a linguagem Python. Este arquivo deve ser baixado para a sua máquina ou para qualquer ambiente online. Uma vez que o arquivo esteja disponível você deverá dividir este arquivo em arquivos com no máximo 1000 palavras e salvar cada um dos arquivos resultantes da divisão na mesma pasta onde salvou o arquivo original. Por fim, você deverá carregar uma dúzia dos arquivos de 1000 palavras em uma estrutura de dados em memória. Você pode usar uma lista de *strings*.

Estes exercícios não têm peso na média da disciplina, contudo são indispensáveis para aprovação. Este conhecimento será útil nas atividade avaliativas.

Você pode mudar o mundo entendendo:

- Os paradigmas de ontem, e de hoje, que determinam como o mundo funciona;
- As vantagens e os custos destes paradigmas;
- O que muda, e o que permanece constante, entre as gerações;
- Quais as melhores técnicas para entender e solucionar os problemas que estes paradigmas causam ou não resolvem.



Obrigada!

Frank de Alcantara
frank.alcantara@pucpr.br