FUNDAMENTOS DA COMPUTAÇÃO

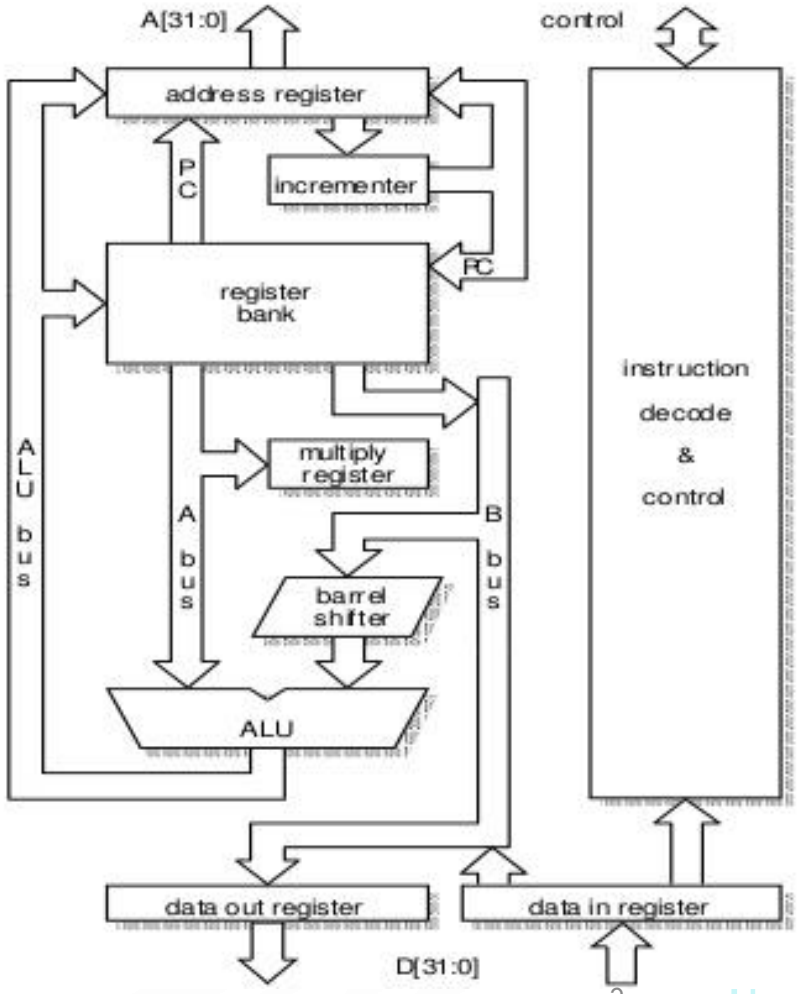
PROF. JOSENALDE OLIVEIRA

josenalde.oliveira@ufrn.br

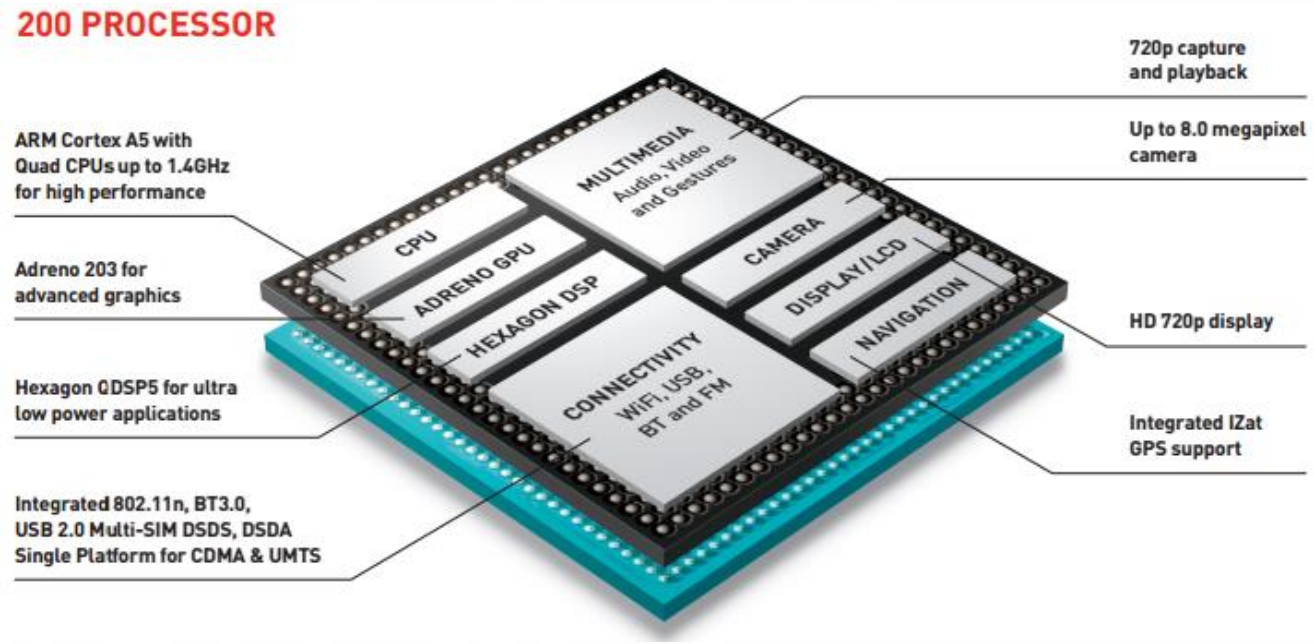
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

Arquitetura ARM (ARM Holdings, Cambridge, Inglaterra, arm.com)

- Família de arquiteturas **RISC** para processadores
  - Set de instruções menor, instruções mais simples
  - Acesso simples à memória, menos ciclos para obter operandos
  - Set de registradores (register bank) para permitir **pipeline**, várias instruções executadas ao mesmo tempo
  - Programas maiores (assembler) ocupam mais memória
- Licenciada para outros fabricantes, podem criar seus próprios Chips, mas usam esta arquitetura como base (Samsung, Apple, Nvidia, Qualcomm, Texas Instruments etc.)
- 37 registradores (30 de propósito geral)
- Instruções de 16 bits (set de instruções thumb)
- Variantes (extensões):
  - NEON (áudio, vídeo), VFP (autotronica, gráfico, 3D, indústria)
  - DSP (sinais digitais), Jazelle (java), big.Little (múltiplos processadores)
  - TrustZone (proteção e segurança digital)



Arquitetura ARM e System on a Chip (SoC)



Cortex A73/A53

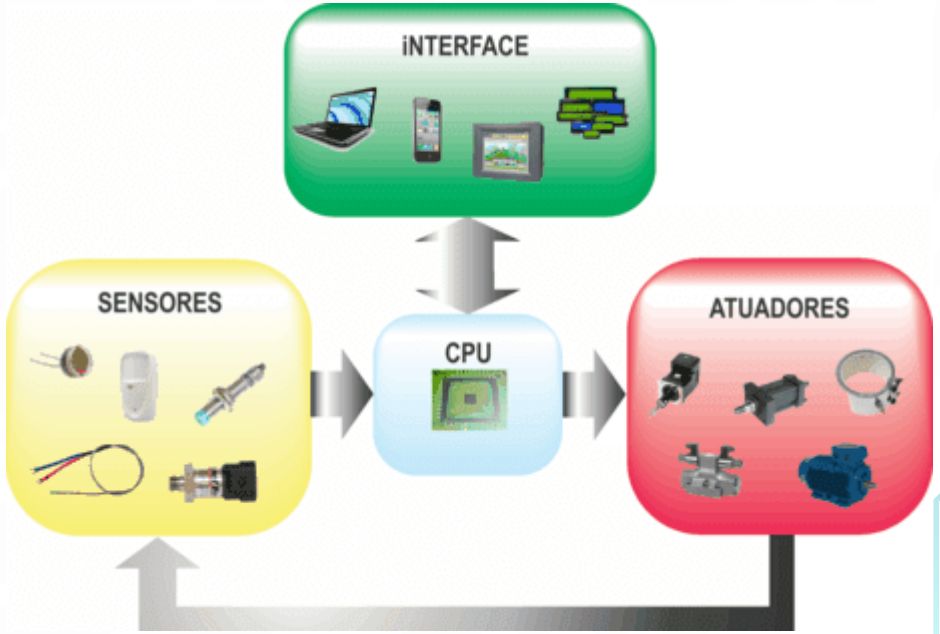
- Integração de componentes num único chip (subsistemas) – CPU, memória, E/S, tratamento de radiofrequência, gerenciamento de bateria, sensores
- Dispositivos menores, com menor consumo de energia
- Ideal para computação móvel, embarcada (ou embutida, **embedded systems**)
- Smartphones, tablets, câmeras digitais, e-book reader, consoles, gps, microcontroladores – em 2017 + 100 bilhões de processadores ARM fabricados

Sistemas Embarcados

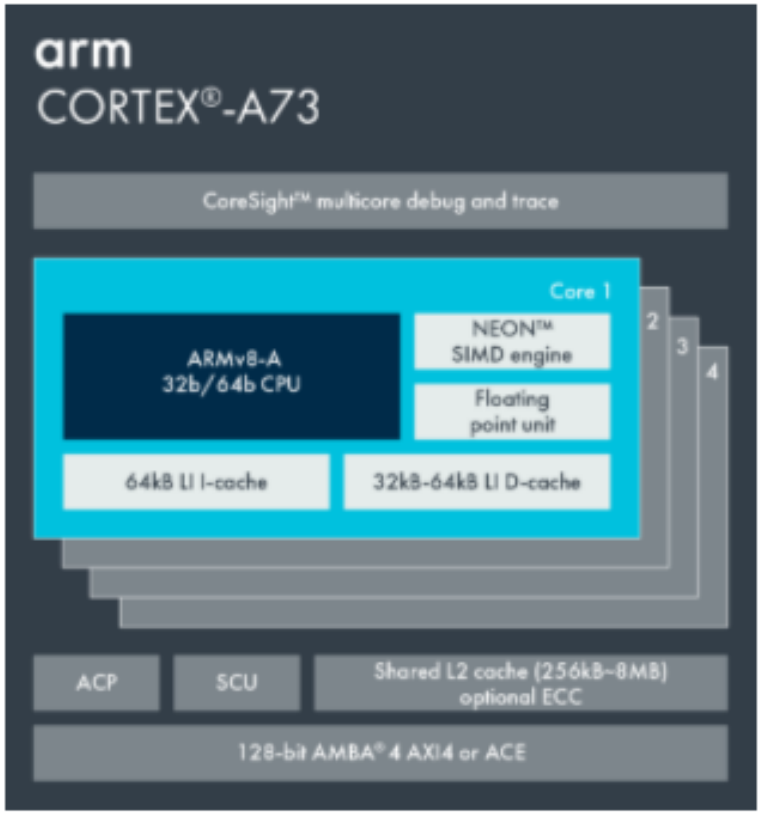
- Eletrônica e software dentro de um produto, ao contrário de um computador de uso geral
- Combinação de hardware e software, com a possibilidade de integração de partes mecânicas e outras, projetada para realizar uma função dedicada; Podem fazer parte de um sistema ou produto maior, como exemplo o sistema de freios ABS de um carro
- Requisitos e restrições variáveis (segurança, confiabilidade, tempo real, flexibilidade, suporte à radiação, vibrações, umidade, tempo de resposta, precisão, etc).

CATEGORIAS

1) **TEMPO REAL: aplicações de armazenamento, automotivas, industriais, redes etc. (Free RTOS etc.)**
2) **PLATAFORMAS: dispositivos com SO abertos, como Linux, Android, Chrome etc.**
3) **SEGURANÇA: smart cards, placas SIM, terminais de pagamento**



Arquitetura ARM e desenvolvimento de software

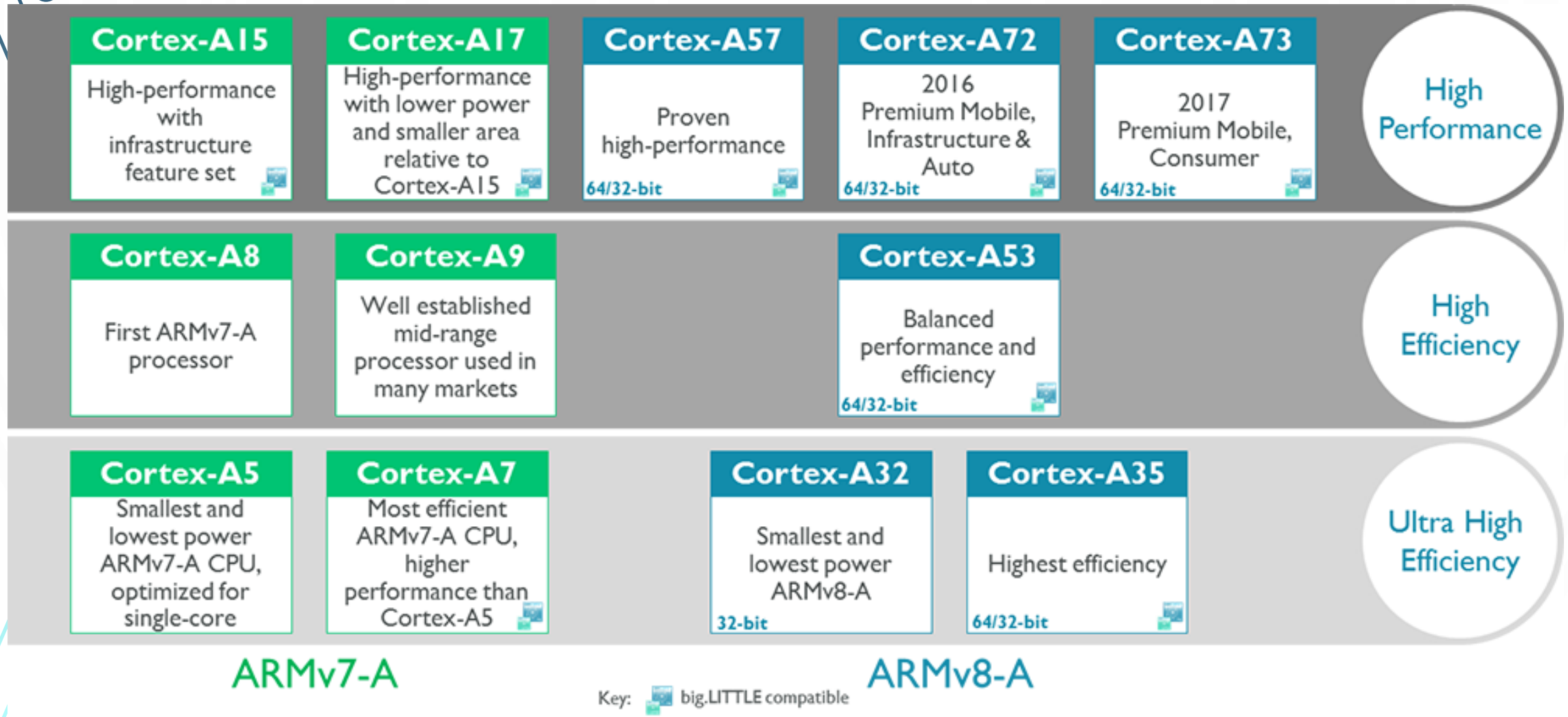


Arm Cortex-A73 CPU

Architecture	Armv8-A
Multicore	1-4x Symmetrical Multiprocessing (SMP) within a single processor cluster, and multiple coherent SMP processor clusters through AMBA 4 ACE technology
ISA Support	• AArch32 for full backward compatibility with Armv7 • AArch64 for 64-bit support and new architectural features • TrustZone security technology • Neon advanced SIMD • DSP & SIMD extensions • VFPv4 floating point • Hardware virtualization support
Debug & Trace	CoreSight SoC-400

https://developer.arm.com/ip-products/processors/cortex-a/cortex-a73

Arquitetura ARM e desenvolvimento de software



Arquitetura ARM e desenvolvimento de software

Raspberry Pi 4 Model B
Broadcom 2711
Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz.

Download the Arduino IDE

ARDUINO 1.8.13

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for Installation instructions.

Windows Installer, for W
Windows ZIP file for no

Windows app Requires
Get

Mac OS X 10.10 or newer

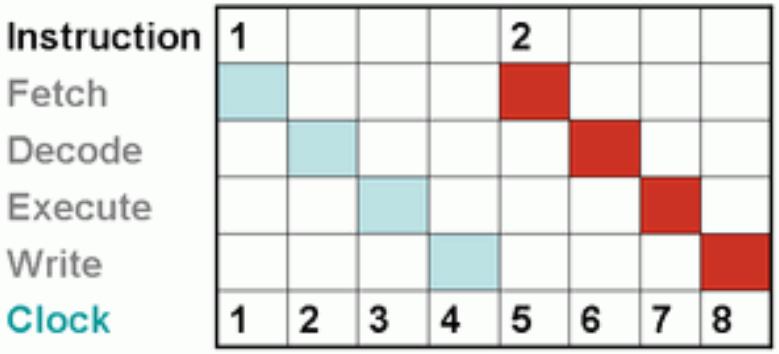
Linux 32 bits
Linux 64 bits
Linux ARM 32 bits
Linux ARM 64 bits

Release Notes
Source Code
Checksums (sha512)

https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/README.md

Noções sobre multiprocessamento (múltipla execução simultânea)

- Ideia base: pipelining



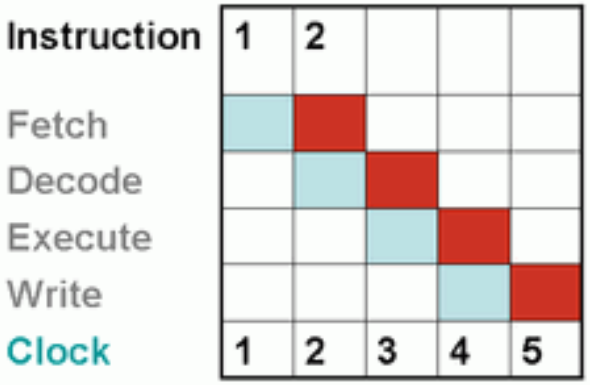
Sem pipeline
Non-Pipelined

Número de estágios do PIPELINE influencia VELOCIDADE
Porém maior complexidade do processador e necessidade de mecanismo de controle de dependência de dados

M=A+B
T=C+D

A+B	C+D
Estágio 1	Estágio 2

Estágios executados simultaneamente



M=A+B
T=M+C

Com pipeline
Pipelined

A+B	Vazio
M+C	Vazio com C
Estágio 1	Estágio 2

Problema: pipeline Stall
Solução: Wide Dynamic Execution (Intel)

https://www.quora.com/What-is-pipelining-super-pipelining-and-super-scalar-in-computer-architecture-and-organization

Noções sobre multiprocessamento (múltipla execução simultânea)

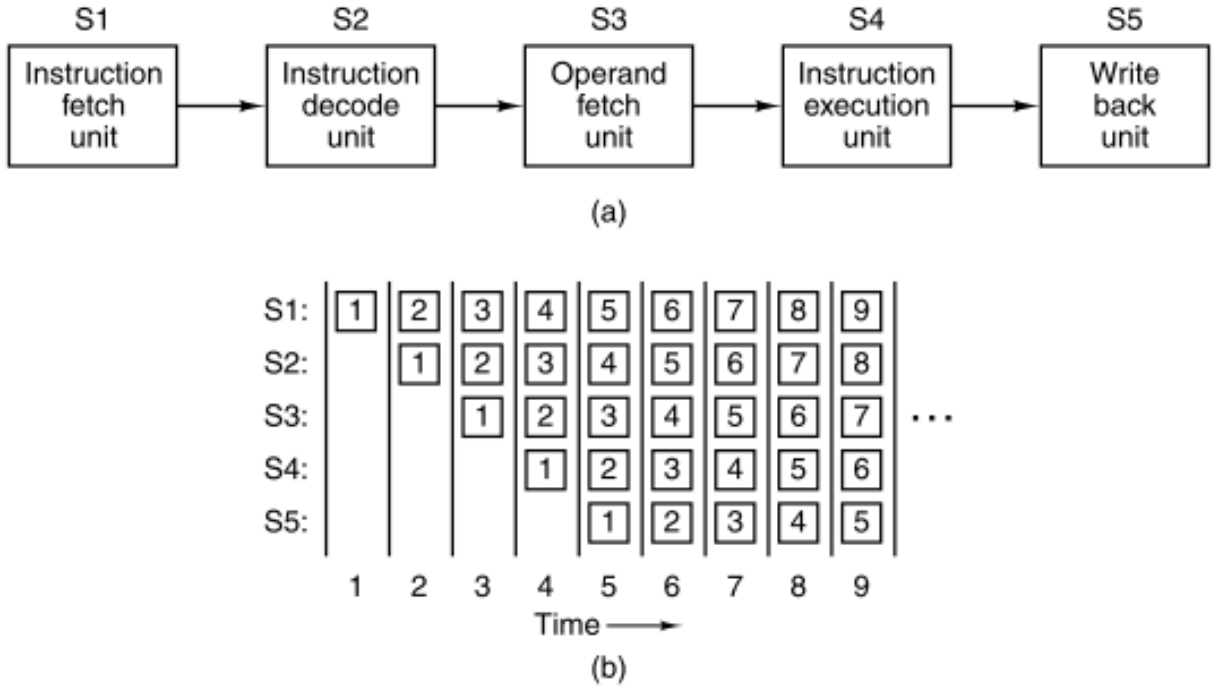


Figure 2-4. (a) A five-stage pipeline. (b) The state of each stage as a function of time. Nine clock cycles are illustrated.

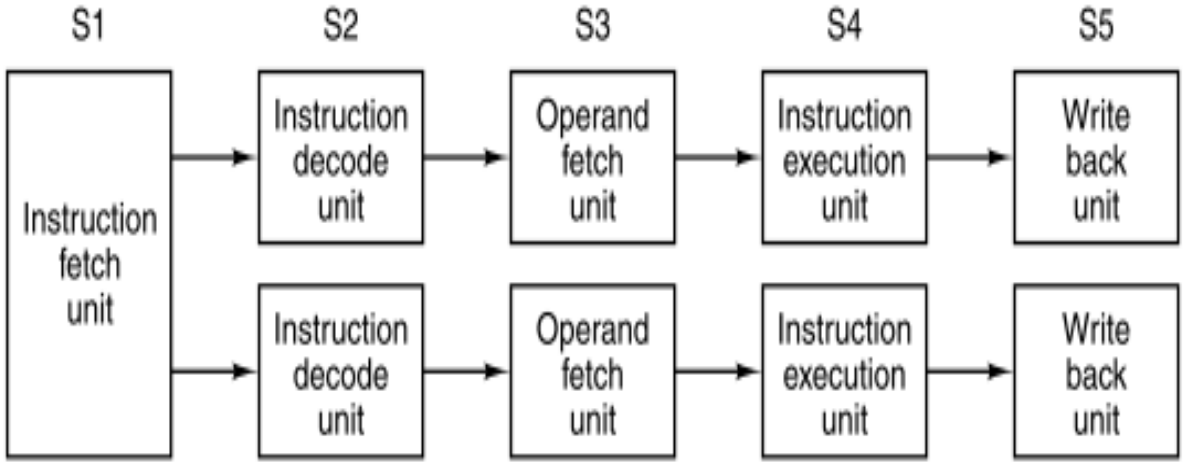


Figure 2-5. Dual five-stage pipelines with a common instruction fetch unit.

Noções sobre multiprocessamento (múltipla execução simultânea)

Problema: ficar ocioso aguardando o resultado da comparação?

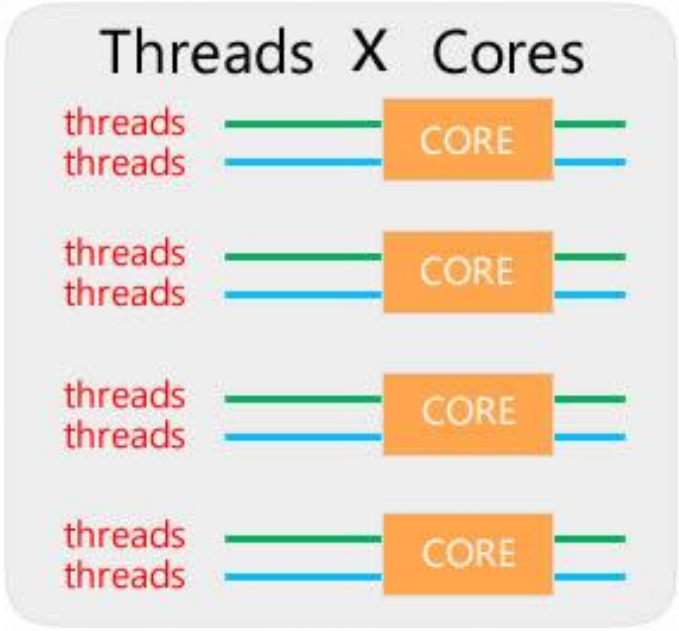
```
Se A=B então
   D = C + 1
Senão
   D = C – 1
Fim Se
```

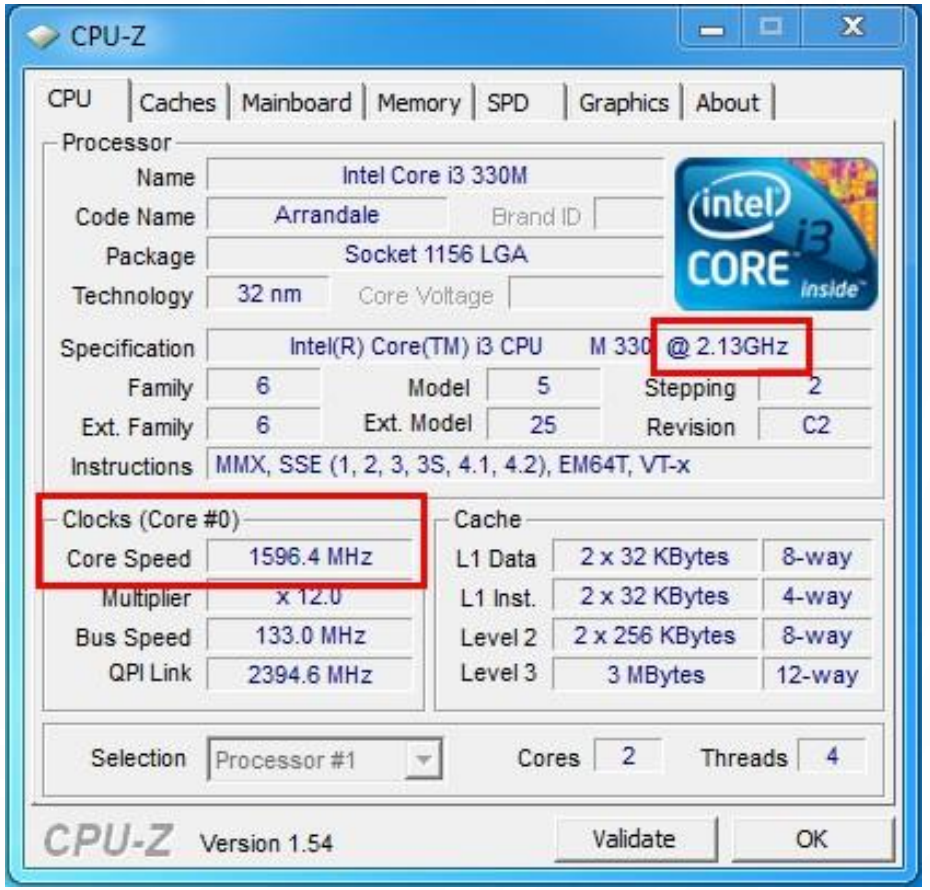
Desvio condicional:
Se A for igual a B, então o processador deve armazenar Em D o resultado de C+1; caso contrário o resultado de C -1

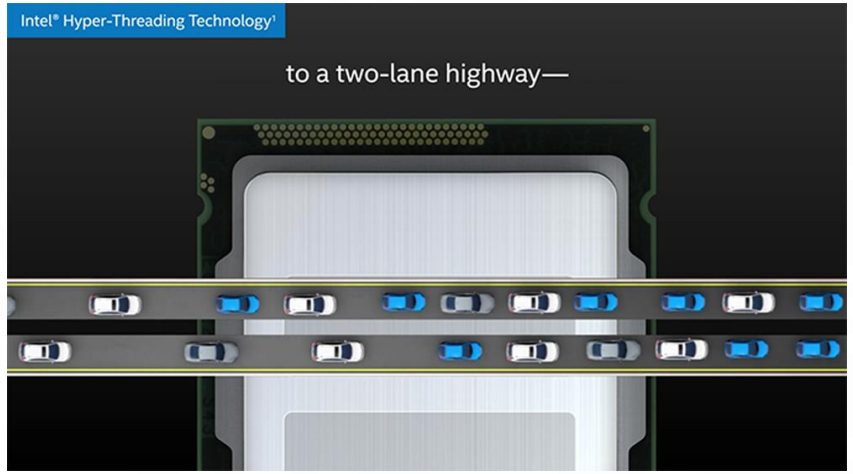
Solução 1: execução especulativa (speculative execution)

- Previsão de desvio (branch prediction) – procurar resultado mais provável; se errar, executar correta, o que gasta tempo
  - **Branch history buffer** (bhb): previsão entre 82% e 99% Armazena bits para tentar criar um histórico de decisões em cada instrução de desvio
  - **Branch target buffer** (btb): permite o processador saber se a próxima instrução é de desvio. Se for, passa a executar técnicas de previsão, antecipando ainda mais.
- Execução fora de ordem (out-of-order execution): instruções completadas foras da ordem, e depois são remontadas na ordem correta – uso de buffers de escrita e instruções longas (vliw) para definir que partes de programas podem ser executadas em paralelo

- Noções multiprocessamento (threads)







- Execução simultânea de sequências diferentes de instruções
- Cada núcleo é uma linha de execução, mas a aplicação precisa ser programada para explorar isto (multi-thread, hyper-thread INTEL)
- Cada núcleo processando 2 threads (fluxos paralelos de código)

DICA: programação paralela C++: <thread>, Open MPI

Noções multiprocessamento (threads)

Processador executando partes diferentes de um mesmo programa, ao mesmo tempo

Para o multiprocessamento ser possível

- Os circuitos de apoio da CPU (chipset) devem suportar
- O sistema operacional deve ser capaz de utilizar vários processadores simultaneamente
- O próprio processador deve poder ser usado num sistema multiprocessado
- Os programas devem ser projetados para tirar proveito do multiprocessamento

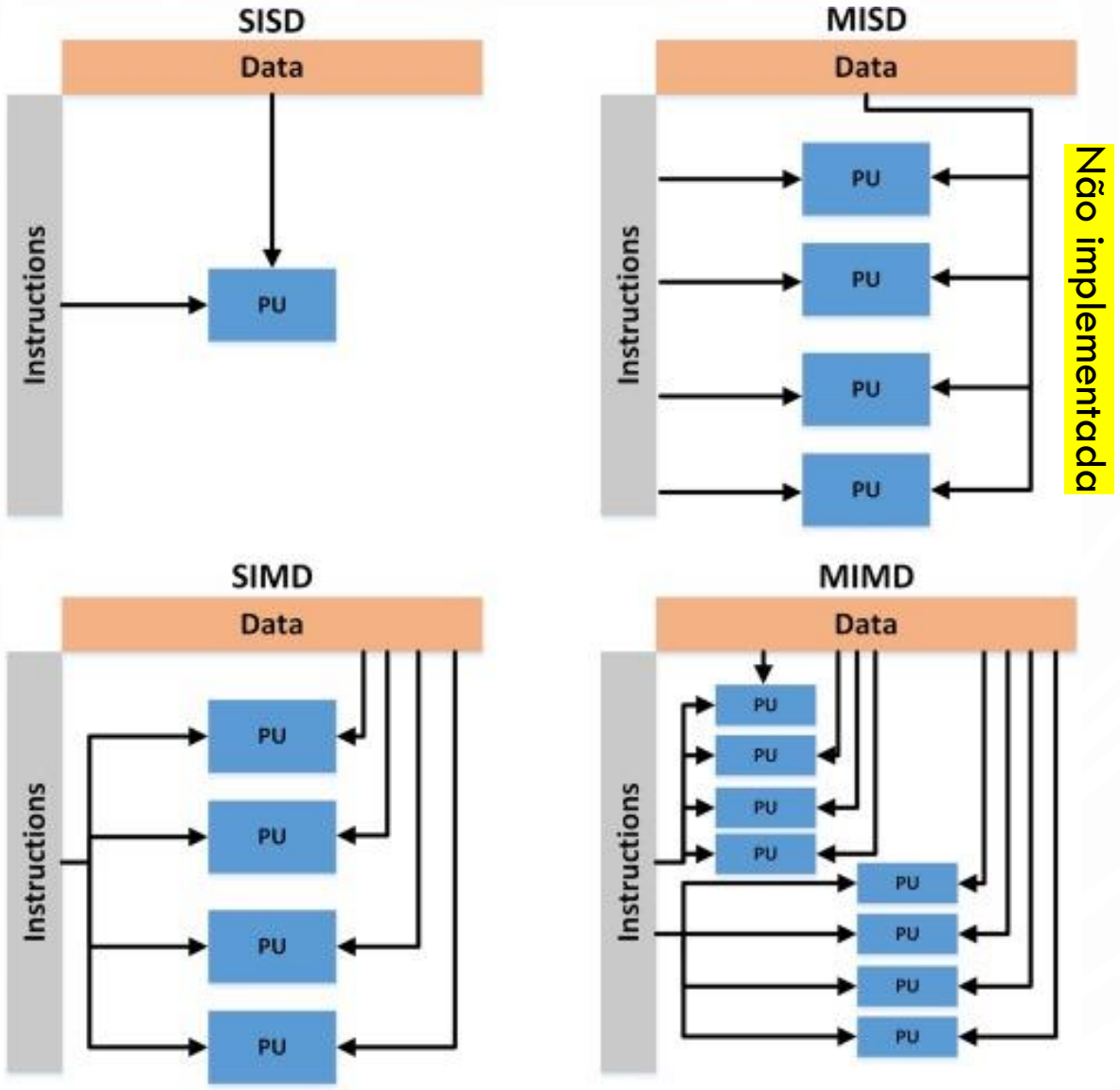
Tipos: **MIMD** (multiple instruction-stream, multiple data-stream): programas diferentes com dados diferentes, computadores de grande porte, SMP, cluster

SIMD: mesma sequência de instruções sobre diferentes conjuntos de dados. Executar a mesma operação com uma grande quantidade de dados diferentes. Cada elemento de processamento possui uma memória de dados associada, então cada instrução é executada num conjunto diferente de dados por processadores diferentes.

MISD: diferentes sequencias de instruções sobre um único conjunto de dados (não implementada comercialmente)

A carga precisa ser distribuída entre os processadores (load balance)

- Noções multiprocessamento (arquiteturas) – classificação de Flynn

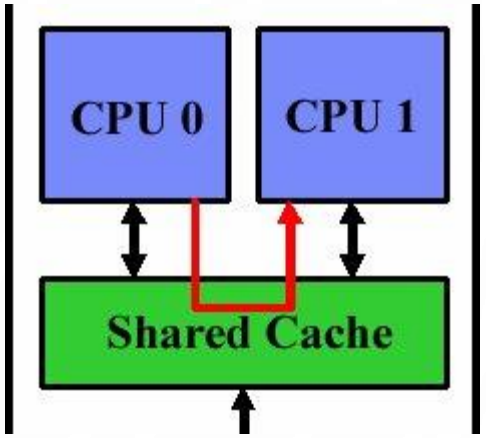


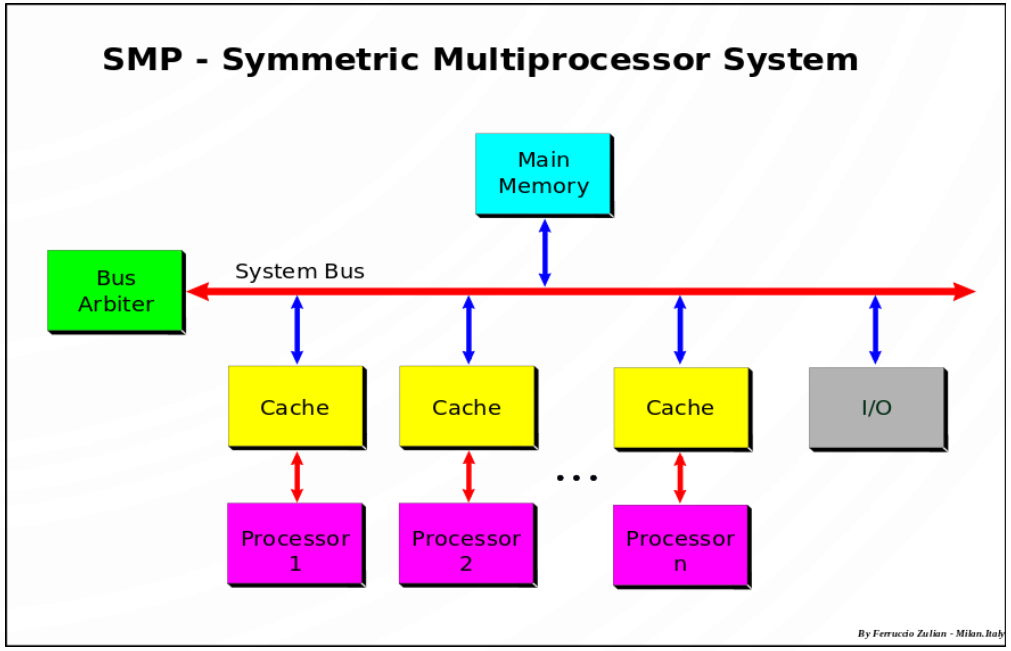
• Noções multiprocessamento (arquiteturas MIMD)

Processadores são de uso geral
Podem haver meios de comunicação do processador diferentes

Solução 1: memória compartilhada: cada processador acessa programas e dados armazenados na memória compartilhada e os processadores se comunicam uns com os outros por meio dessa memória SMP – multiprocessador simétrico – múltiplos processadores compartilham uma única memória ou um pool de memória por um barramento compartilhado; o tempo de acesso é aproximadamente o mesmo para cada processador, embora na arquitetura variante **NUMA** o tempo de acesso possa ser diferenciado (não uniforme).

Solução 2: memória distribuída:





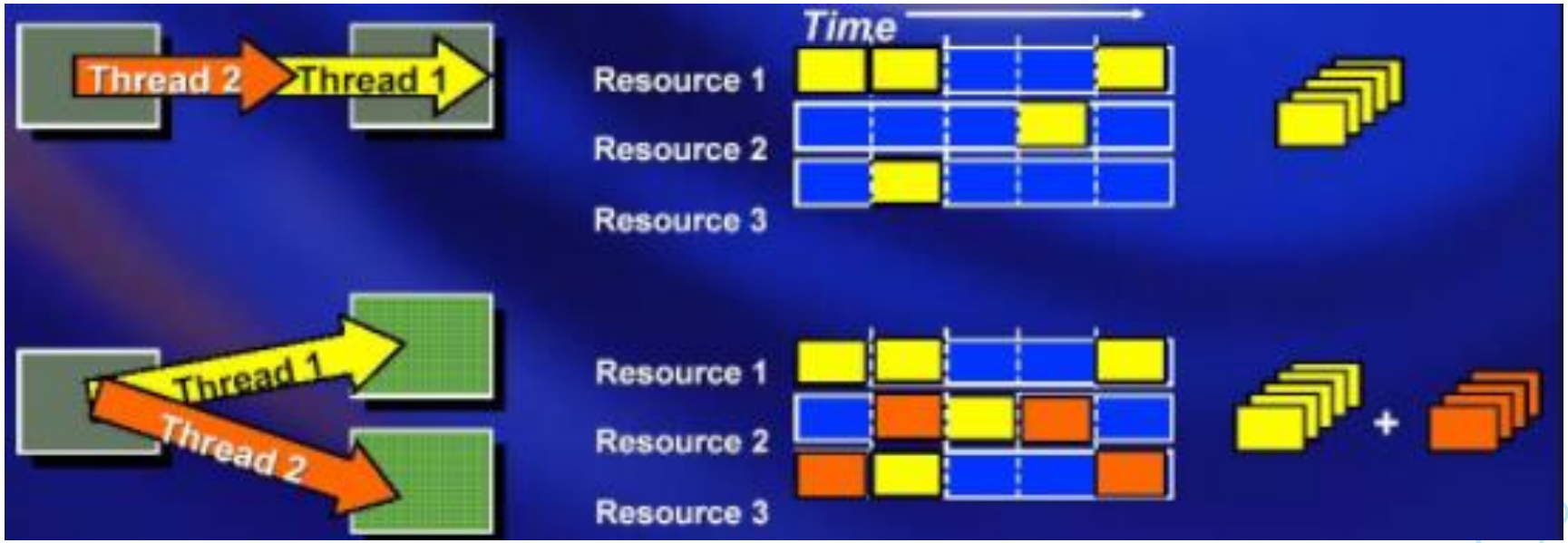
Hyperthreading Intel: ideia é usar recursos ociosos

Várias threads (partes) de um mesmo programa (ou de vários programas) usam recursos do processador ao mesmo tempo

Um processador físico com várias unidades lógicas

Thread 2 precisa esperar Thread 1 finalizar, mesmo com recursos disponíveis

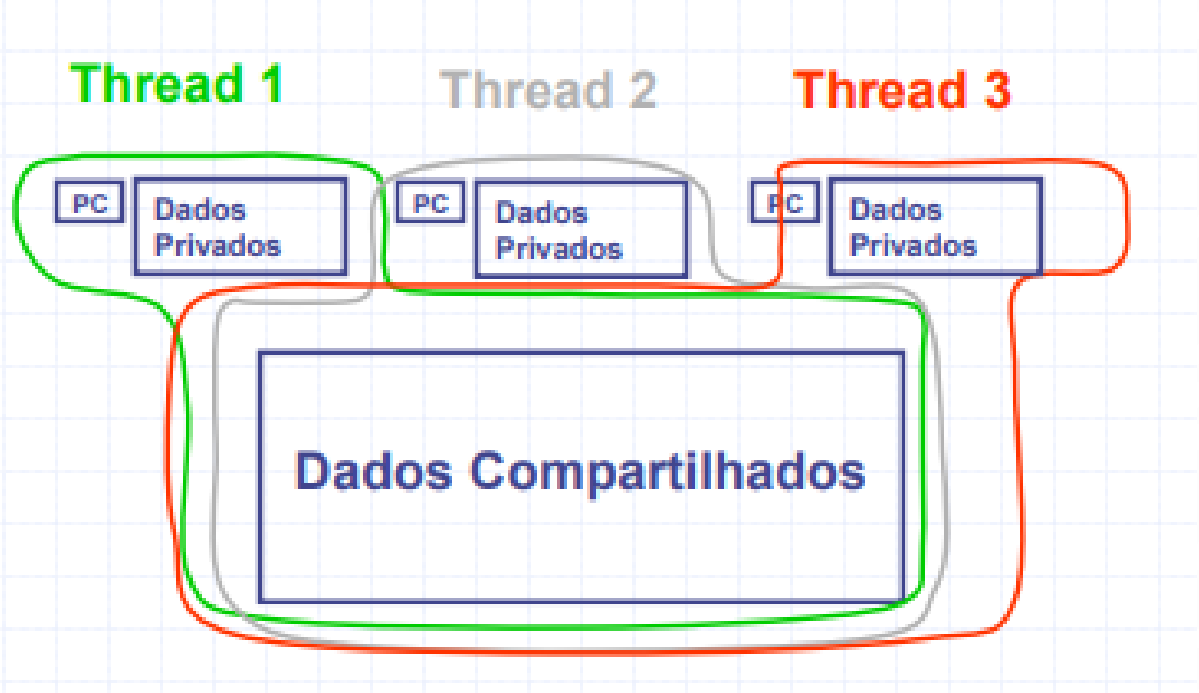
Já aqui, os recursos são utilizados por ambas



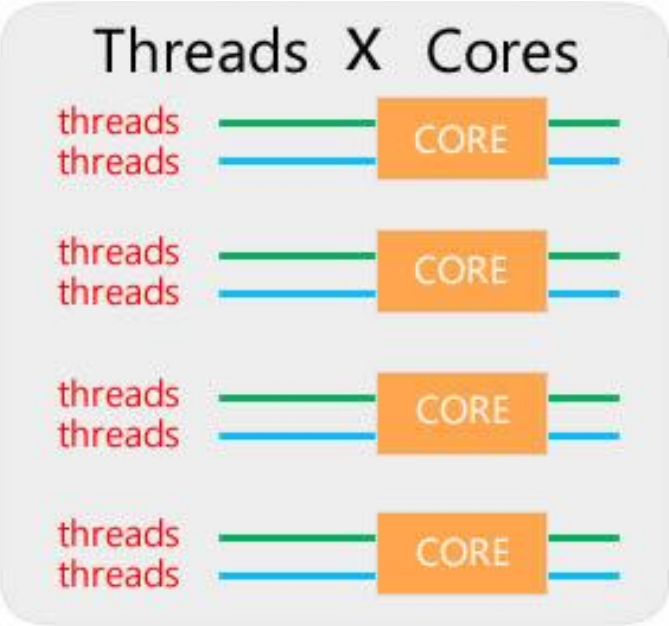
Hyperthreading Intel: ideia é usar recursos ociosos

Várias threads (partes) de um mesmo programa (ou de vários programas) usam recursos do processador ao mesmo tempo

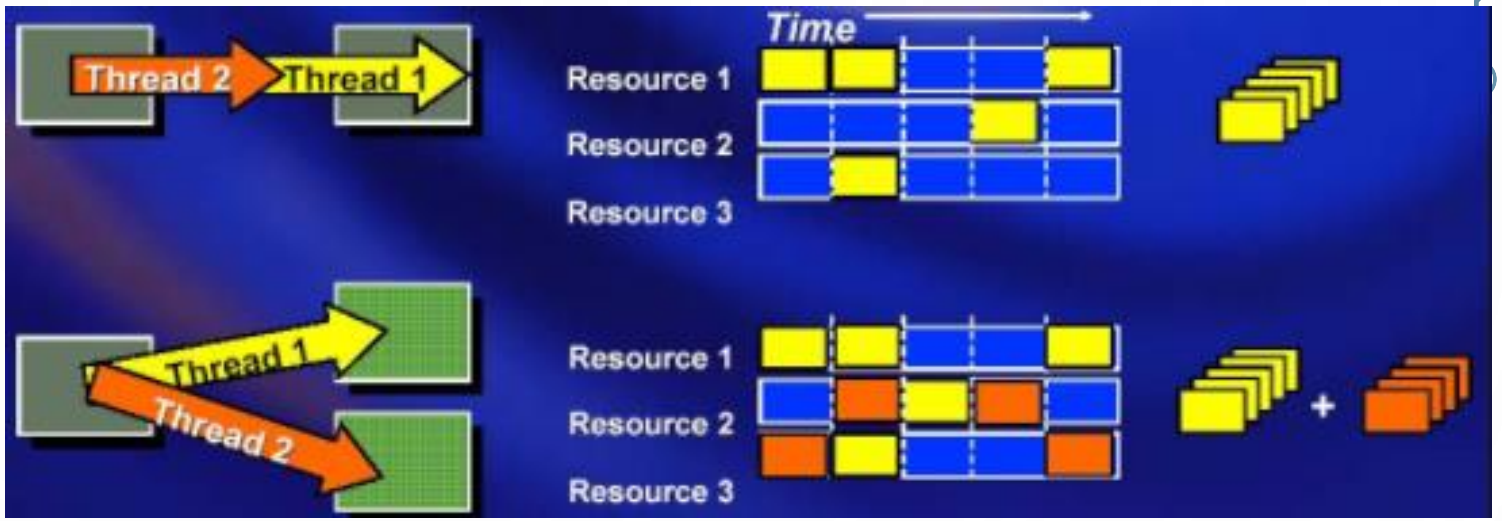
Um processador físico com várias unidades lógicas



Núcleos múltiplos (multicore)



+



Múltiplos núcleos num processador: vários núcleos executando as instruções. Cada núcleo possui sua Unidade de Controle, Unidade de Execução etc.

Multicore

- Ou chip multiprocessador, combina dois ou mais processadores num único chip de computador
- O uso de chips com processador único (singlecore) cada vez mais complexo atingiu o limite por conta do desempenho do hardware, limites físicos, limites no paralelismo em nível de instruções e limitação de energia
- Por outro lado, a arquitetura multicore oferece desafios para desenvolvedores de software **para explorar a capacidade de multithreading por meio de vários núcleos**
- As principais variáveis são: o número de processadores no chip, o número de níveis de memória cache e a extensão em que a cache é compartilhada
- Decisão de projeto multicore: núcleos individuais superescalares ou SMT (multithreading simultâneo)

Multicore

- Objetivo inicial: paralelismo em nível de instruções, mais trabalho feito por ciclo de clock
- EVOLUÇÃO
  - Pipeline
  - Superescalar: vários pipelines são construídos pela replicação de recursos da execução – execução paralela de instruções em pipelines paralelos
  - **Multithreading simultâneo** (SMT): bancos de registradores são replicados para que várias threads possam compartilhar o uso dos recursos do pipeline

Uso mais eficiente da cache: é improvável que uma única thread de execução possa usar efetivamente toda a memória. Um número de threads ou processos relativamente independentes tem uma oportunidade maior de tirar vantagem da memória cache

Multicore

- Desafio para o desenvolvedor: identificar partes do código ‘sequencial’ que podem ser ‘paralelizados’

$$\text{Aumento de velocidade} = \frac{tempo\ para\ executar\ o\ programa\ em\ single\ core}{tempo\ para\ executar\ o\ programa\ em\ N\ processadores} = \frac{1}{(1-f)+\frac{f}{N}}$$

Lei de Amdahl: a lei supõe um programa no qual uma fração (1-f) do tempo de execução envolve código inerentemente serial e uma fração f que envolve código infinitamente paralelizável

Contudo, numa abordagem direta, não significa aumento de desempenho significativo, pois a aumento de sobrecarga na comunicação e distribuição de trabalho entre vários processadores e sobrecarga de coerência de cache. A curva de melhoria apresenta picos, mas depois passa a degradar. Por exemplo, se f = 0,9 (apenas 10% de serial), com N = 8, a melhoria é de 4,7. Contudo melhorias para reduzir a fração serial foram incorporadas ao longo do tempo e *sistemas operacionais, bancos de dados e software para servidores usam intensamente a organização paralela*

Multicore

- Desafio para o desenvolvedor: identificar partes do código 'sequencial' que podem ser 'paralelizados'
- Threads são de baixo custo computacional, mais baixo nível e gerencia mais cuidadosa – Linux (pthreads); Windows (WinThreads)
- APIs (Application Interfaces) como OpenMP e MPI são modelos de alto nível

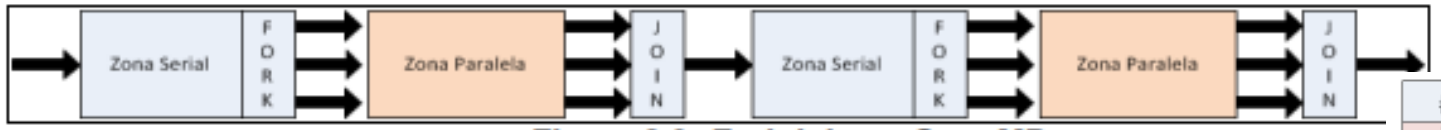


Figura 2.3. *Fork-join* no OpenMP

```
//soma e imprime a soma do vetor
void somaVetor(double *vetor, int tamanho)
{
   int i =0;
   double soma=0;
   #pragma omp parallel for reduction(+:soma)
   for(i=0; i<tamanho; i++)
   {
      soma+=vetor[i];
   }
   printf("Resultado da soma %3.f\n", soma);
}
```

Figura 2.4. OpenMP: a diretiva #pragma omp parallel for informa ao compilador que a região deve ser paralelizada.

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[]) {

  MPI_Init(&argc, &argv);

  // Instruções MPI e a implementação do seu código

  MPI_Finalize();

 return 0;
}
```

Região paralela do código

Figura 2.6. Estrutura de um código em C/MPI.

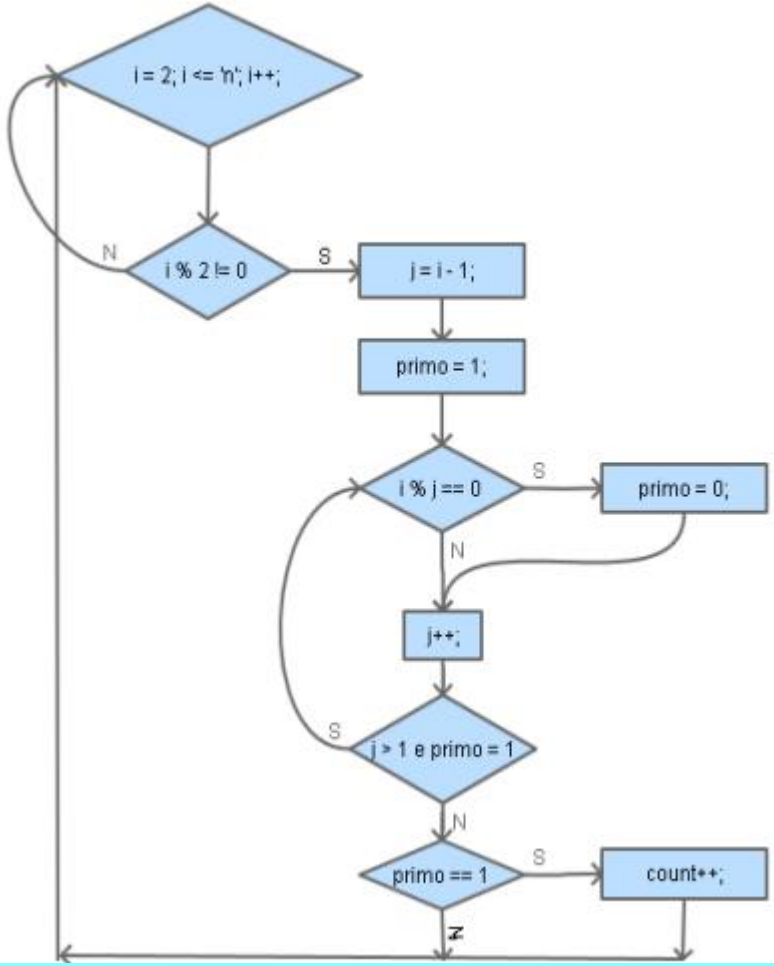
Silva, Cezar G. A.; Asenjo, Maurício N. *Programação paralela – uma introdução ao paralelismo com OPEN MPI*. In: 13. Congresso Nacional de Iniciação Científica, 2013. Disponível em: <http://conic-semesp.org.br/anais/files/2013/trabalho-1000015028.pdf>

Com as aplicações rodando em um computador equipado com o processador Intel Core i3 M350x4 em ambiente Linux, foram obtidos os seguintes resultados:

Quadro 2 – Representação dos resultados obtidos em um processador Intel Core i3.

Quantidade de Números	Sem Paralelismo	Paralelo - 2 Processos	Paralelo - 4 Processos
100	0,000028s	1,175111s	1,039873s
1.000	0,002574s	1,027346s	1,047605s
10.000	0,110456s	1,106495s	1,111911s
100.000	10,531448s	8,884402s	7,512204s
1.000.000	1050,243978s	786,315063s	579,89856s

A presença do paralelismo é evidente nos maiores números e também quando levamos o paralelismo aos limites do processador. Durante a execução da aplicação sem paralelismo foi observado que o uso de CPU não ultrapassou os 30%, porém no algoritmo paralelo dividido em quatro processos o uso de CPU atingiu a marca de 100%, ou seja, o processador estava trabalhando com carga máxima durante o paralelismo.



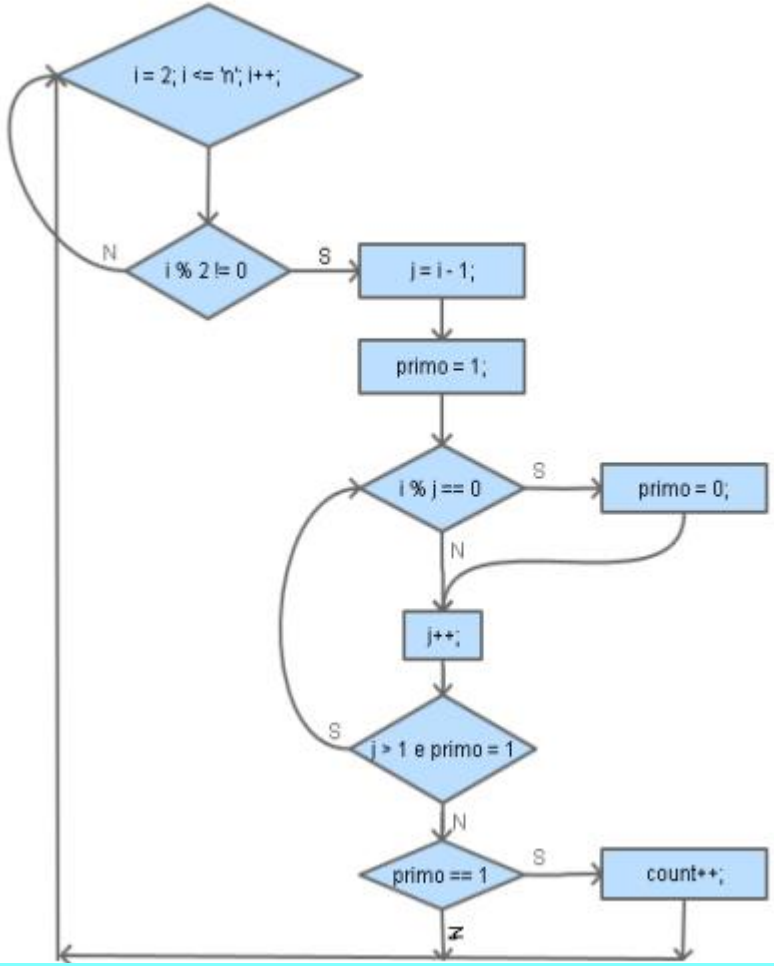
Encontrar o número de primos numa determinada faixa

Silva, Cezar G. A.; Asenjo, Maurício N. *Programação paralela – uma introdução ao paralelismo com OPEN MPI*. In: 13. Congresso Nacional de Iniciação Científica, 2013. Disponível em: <http://conic-semesp.org.br/anais/files/2013/trabalho-1000015028.pdf>

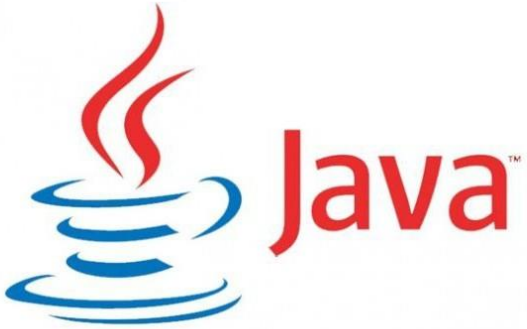
Para ter uma melhor comparação o teste foi realizado também sobre outra configuração, um computador equipado com um processador Inter Core i7x8 em um ambiente Linux rodando em uma máquina virtual. E os resultados obtidos foram:

Quadro 3 – Representação dos resultados obtidos em um processador Intel Core i7.

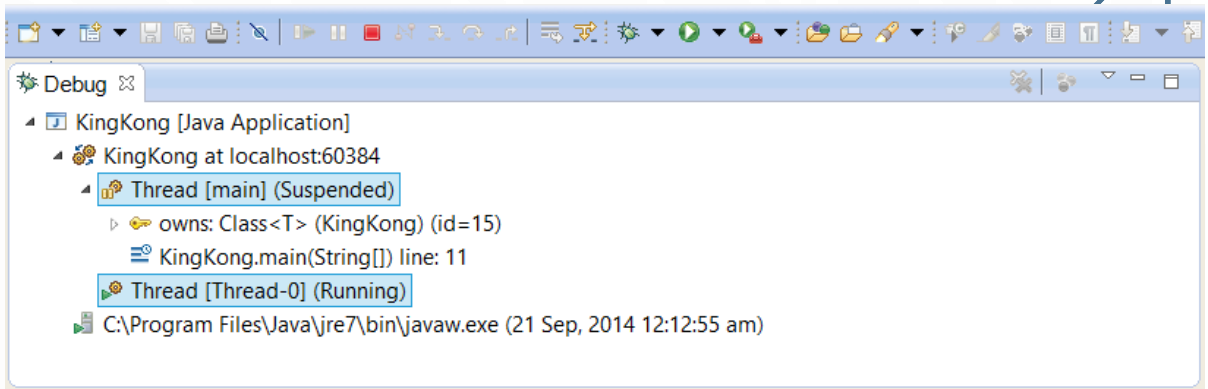
Quantidade de Números	Sem Paralelismo	Paralelo - 4 Processos	Paralelo - 8 Processos
100	0,000012s	1,060125s	1,142675s
1.000	0,000467s	1,065693s	1,089045s
10.000	0,106573s	1,083212s	1,867748s
100.000	9,743917s	5,705782s	4,066512s
1.000.000	959,056274s	496,121704s	373,259491s



Encontrar o número de primos numa determinada faixa

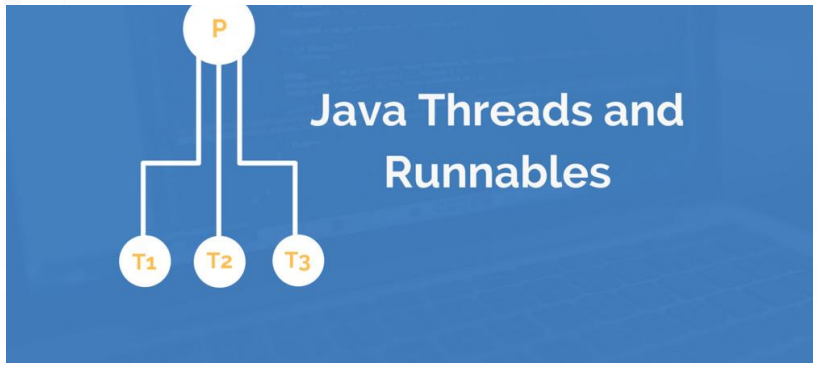


Observação: aplicações Java abraçam threads de uma maneira fundamental. Não apenas a linguagem Java facilita aplicações multithread, mas a Java Virtual Machine é um processo multithread que provê agendamento de memória para aplicações Java. Aplicações Java podem, então, se beneficiar diretamente dos recursos multicore incluindo servidores de aplicação como Java Application Server da Sun, Weblogic da BEA, Websphere da IBM e servidor TOMCAT.



KingKong.java

```
1  class KingKong {
2
3      public static synchronized void main(String[] args) {
4          Thread t = new Thread() {
5              public void run() {
6                  kong();
7              }
8          };
9
10         t.start();
11         System.out.print("King");
12     }
13
14     public static synchronized void kong() {
15         System.out.print("Kong");
16     }
17 }
```



Alguns exemplos com threads e execuções simultâneas

async.cpp

Thread1

Thread 2

Thread 3

Openmp1

Openmp2