Breve-Resumo
Máquinas Multinível

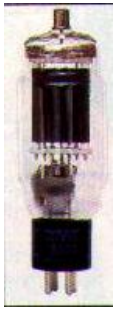
- lucilena.lima@fatec.sp.gov.br
- luma.delima@gmail.com

Arquitetura de Computadores

- Nestes slides...
  - Resumo Histórico
  - Máquina Multinível
  - Componentes do Computador
  - Conjunto (básico) de Barramentos

Quanto à característica de construção

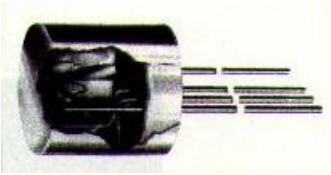
- **1ª GERAÇÃO (...Década de 50):**
  - A válvula é o componente básico
    - Grande
    - Esquentava muito
    - Gastava muita energia elétrica



  - Computadores ocupavam muito espaço físico.
  - Tinham, dispositivos de Entrada/Saída primitivos (através da cartões perfurados).
  - Eram aplicados em campos científicos e militares.
  - Linguagem de programação: linguagem de máquina.
  - Operações internas mediam-se em milissegundos.

Quanto à característica de construção

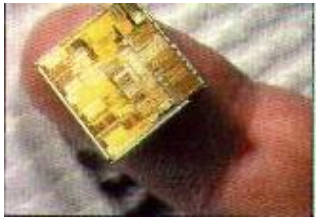
- **2ª GERAÇÃO (Início dos anos 60):**



  - O transistor é o componente básico
    - Tamanho menor que a válvula
    - Esquentava menos
    - Gastava menos energia elétrica
    - Mais durável e confiável
  - As máquinas diminuíram muito em tamanho e suas aplicações passam além da científica e militar a administrativa e gerencial.
  - Surgem as primeiras linguagens de programação.
  - Além dos núcleos de ferrite, fitas e tambores magnéticos passam a ser usados como memória.
  - Operações internas mediam-se em microssegundos.

Quanto à característica de construção

- **3ª GERAÇÃO (meados dos anos 60 até meados dos anos 70):**
  - Marco inicial: surgimento dos C.Is.
  - O LSI passa a ser o componente básico
    - O LSI ficou conhecido como ‘chip’
    - Pequena pastilha de silício de 1 cm^2
    - Composto de milhares de transistores



  - Os computadores diminuíram de tamanho e aumentaram seu desempenho
  - Evolução dos Sistemas Operacionais, surgimento da multiprogramação, *real time* e modo interativo.
  - A memória é feita de semicondutores e discos magnéticos.
  - Operações internas mediam-se em nanossegundos.

Quanto à característica de construção

- **4ª GERAÇÃO (meados dos anos 70 a início dos anos 90):**
  - Tem como marco inicial o surgimento do microprocessador.
  - O VLSI é o componente básico (menor que o LSI), porém com maior integração
  - Houve a miniaturização dos computadores
  - Nesta geração é que surgiram os microcomputadores PC
  - Surgem muitas linguagens de alto-nível e nasce a teleinformática, transmissão de dados entre computadores através de rede.
  - Operações internas mediam-se em picossegundos.

Computador -

- O computador é uma máquina eletrônica capaz de receber informações, submetê-las a um conjunto especificado e pré-determinado de operações lógicas e aritméticas, e fornecer o resultado destas operações.

- Os computadores de hoje são dispositivos eletrônicos que, sob direção e controle de um programa, executam quatro operações básicas:
  - **Entrada,**
  - **Processamento,**
  - **Saída e**
  - **Armazenamento.**

Principais Unidades Funcionais do Computador

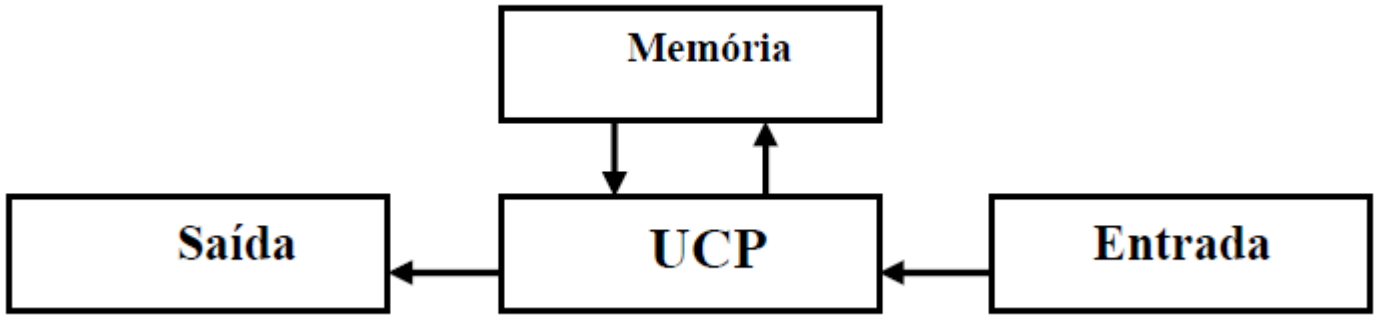
Modelo de von Neumann

Nos anos 40 o matemático *John von Neumann desenvolveu um modelo teórico do* funcionamento dos computadores. Este modelo é usado até hoje com apenas algumas modificações.

Von Neumann propôs construir computadores que:

1. Codificassem instruções que pudessem ser armazenadas na memória e sugeriu que usassem cadeias de uns e zeros (binário) para codificá-las;
2. Armazenassem na memória as instruções e todas as informações que fossem necessárias para a execução da tarefa desejada;
3. Ao processarem o programa, as instruções fossem buscadas na diretamente na memória. Este é o conceito de programa armazenado.

A Figura apresenta um esquema do modelo de von Neumann.



O Processador

Principal unidade do computador (o cérebro);

Executa e gerencia todas as operações do computador com o auxílio dos demais dispositivos (memória, periféricos,…);

Principal unidade a determinar o poder computacional da máquina;

É composto por milhões de transistores.

O Processador é composto por:

- **Unidade de controle (UC):**
  - Faz com que as instruções sejam processadas:
  - Gerenciamento da memória principal;
  - Requisições as unidades/dispositivos que devem colaborar no processo;
- **Unidade Lógica Aritmética (ULA):**
  - Realiza as operações matemáticas (soma, subtração, divisão e multiplicação).
  - Realiza testes lógicos baseados nas instruções de programa (Álgebra Booleana).

O Processador é composto por:

- **Registradores:**
  - Armazena endereços de instruções e dados que estão sendo processados;
  - Memória rápida para guardar informações de controle e resultados intermediários
- **Clock:**
  - Emite pulsos elétricos que se propagam pelos barramentos da placa-mãe;
  - Usado para cronometrar operações de processamento e ditar a velocidade de transferência de dados;

Unidade Funcional

- **MEMÓRIA:**

  Armazenamento de programas e dados;

  Local onde o processador: Busca dados a serem processados; Guarda valores intermediários; Envia resultados finais do processamento;

  Tipos de Memória:

  Principal: RAM, ROM;

  Cache;

  Secundárias:

  Disco Rígico (HD), discos flexíveis, CD etc...

Unidade Funcional: Memória RAM

RAM (Random Access Memory)

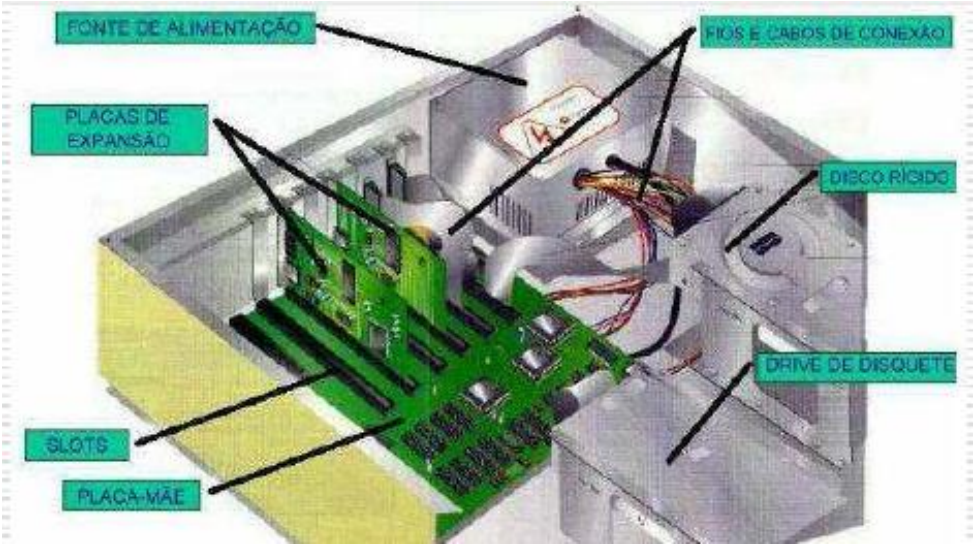
- Acesso aleatório: capacidade de acesso a qualquer posição em qualquer momento;
- Armazena instruções que estão sendo executadas e os dados necessários a sua execução;
- Memória de leitura e escrita e de rápido acesso;
- É volátil (na falta de energia elétrica ou desligamento do computador as informações são perdidas).;

Unidade Funcional: Memória ROM

ROM (Read Only Memory)

- Memória de leitura– gravada pelo fabricante;
- Pouca capacidade de armazenamento;
- Não-volátil;
- Exemplos:
- BIOS(Basic Input Output System): Armazenam informações para iniciar o computador, verificar a memória RAM, iniciar dispositivos e dar início ao processo de boot;
- CMOS (Complementary Metal-Oxide Semicondutor): Armazena as informações do sistema (setup);

Unidade Funcional: Placa Mãe (motherboard)



Unidade Funcional: Placa-mãe (motherboard)

É o meio pelo qual o processador se comunica com os demais dispositivos do computador;

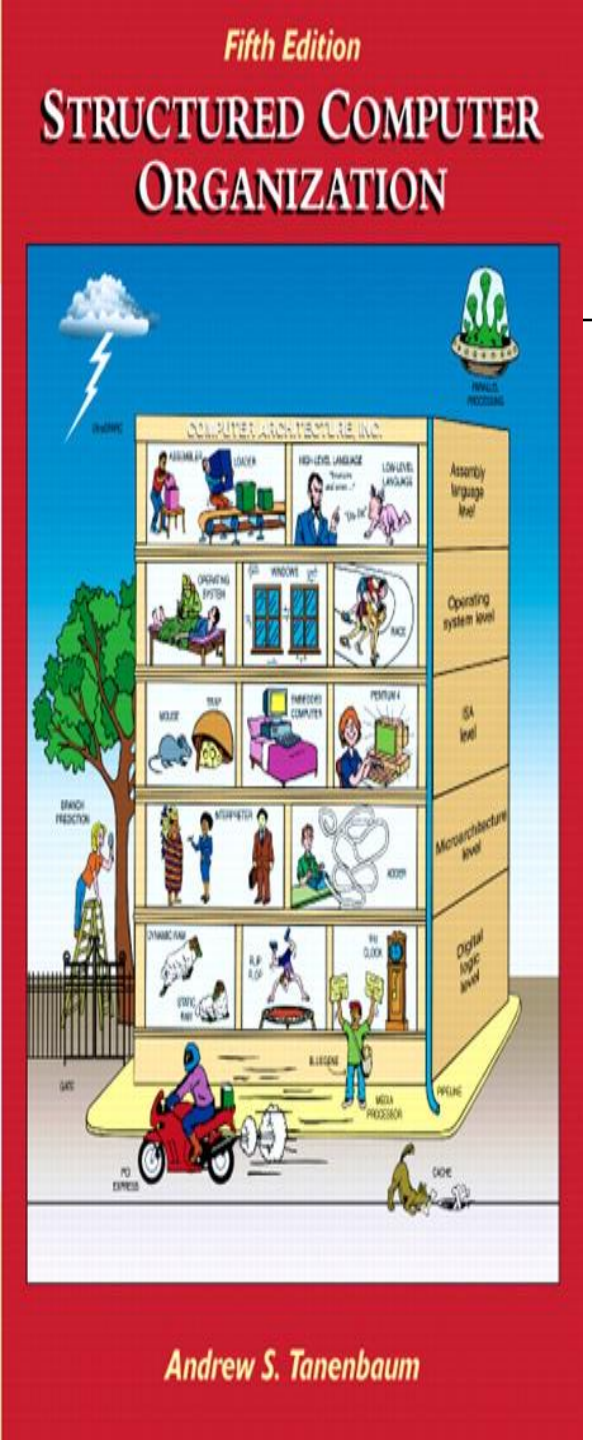
É um circuito impresso composto basicamente por:

- Processador;
- Memórias;
- Barramentos;
- Chipsets;
- Slots;
- Portas;
- Outros...

As placas-mãe podem ser:

On-board – Placas controladores de dispositivos (som, rede, fax e vídeo) são embutidas nela;

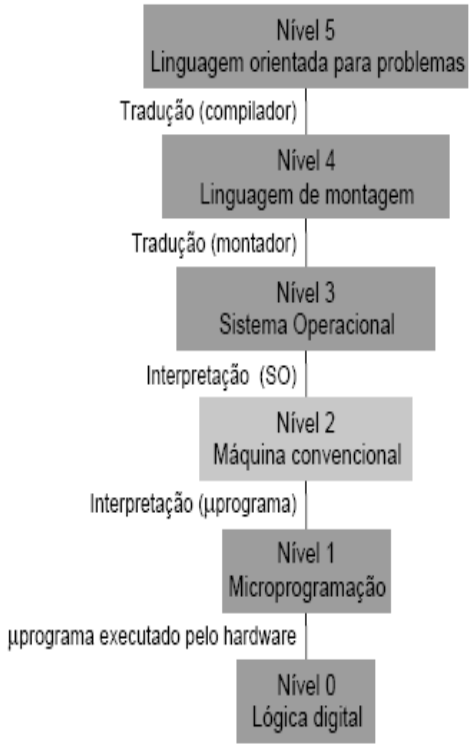
Off-board – Placas individuais, acopladas a placa-mãe através de slots de expansão;



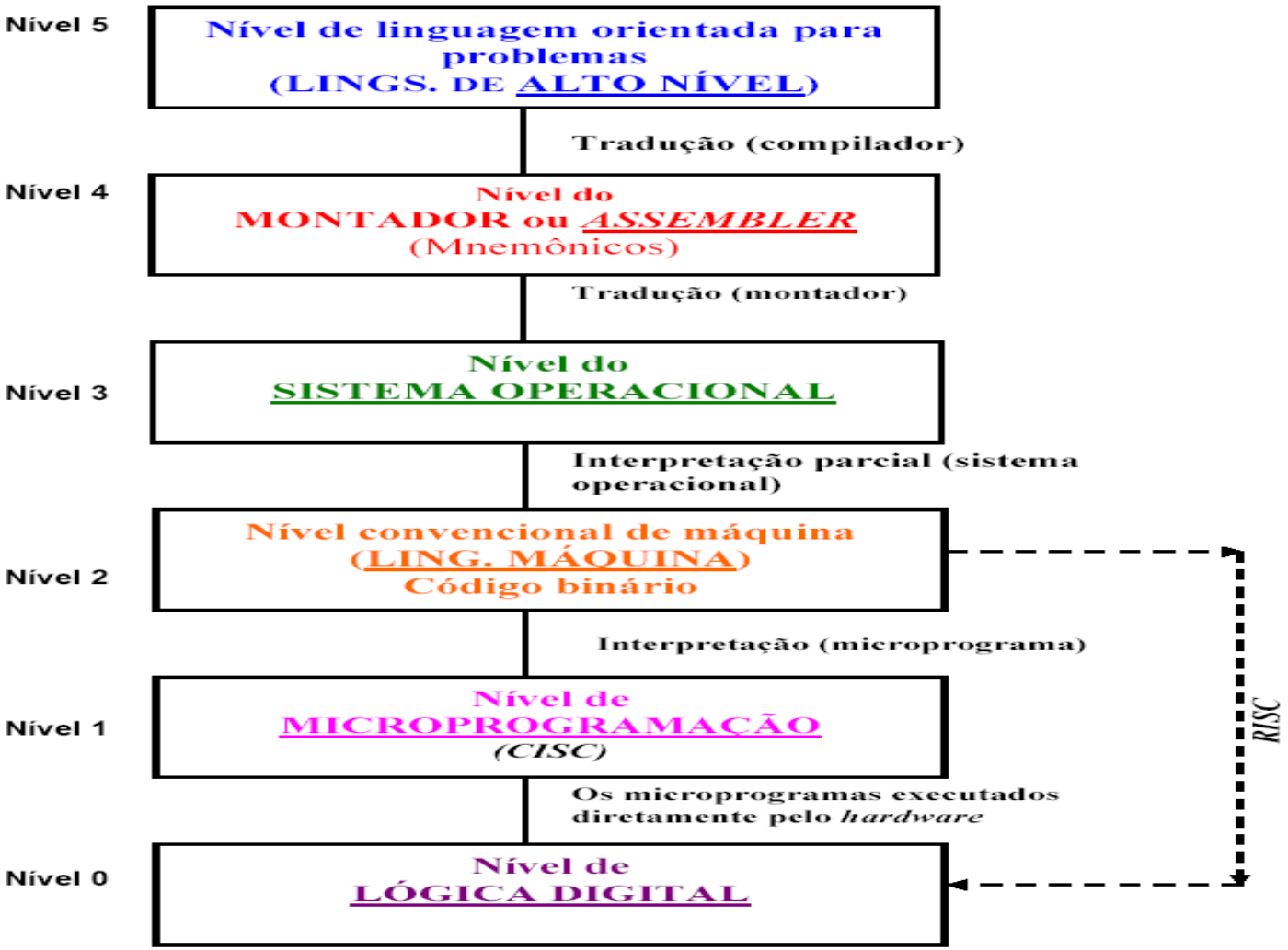
Níveis de Abstração

- O que é um Computador
• Um computador é uma máquina eletrônica/lógica
• Programável – Programa
• Instruções•

Máquina Multinível:
-Representada por uma hierarquia
de níveis de abstração (Tanenbaum)



Máquina Estruturada Multinível



LINGUAGENS e NÍVEIS

- A IMPLEMENTAÇÃO DE NOVAS LINGUAGENS, CADA UMA MAIS CONVENIENTE DO QUE AS ANTECESSORAS PODE PROSSEGUIR INDEFINIDAMENTE ATÉ SE CONSEGUIR A MAIS ADEQUADA AOS NOSSOS PROPÓSITOS.

- CADA LINGUAGEM UTILIZA A ANTECESSORA COMO BASE. DESSA FORMA UM COMPUTADOR QUE USE ESSA TÉCNICA PODE SER VISTO COMO UM CONJUNTO DE CAMADAS E NÍVEIS.

LINGUAGENS e NÍVEIS

INTRODUÇÃO

- PROGRAMA → SEQUÊNCIA DE INSTRUÇÕES PARA EXECUTAR UMA DETERMINADA TAREFA.

- LINGUAGEM DE MÁQUINA →SÃO INSTRUÇÕES PRIMITIVAS PARA INTERAÇÃO ENTRE USUÁRIO E MÁQUINA.
- A L.M É DIFÍCIL DE SER USADA EM VIRTUDE DA PROXIMIDADE COM O NÍVEL DE HARDWARE DA MÁQUINA. ASSIM USA-SE UM CONJUNTO DE INSTRUÇÕES QUE SEJA MAIS CONVENIENTE PARA AS PESSOAS.

LINGUAGENS DE BAIXO-NÍVEL

- A arquitetura proposta por Von Neumann (1945) utiliza instruções de máquina. Este tipo de instrução não é simples para ser usada por programadores.
- A linguagem *assembly* (desenvolvida nos anos 50). Utiliza códigos mnemônicos (ADD, SUB, ...), mais fáceis de aprender e memorizar que os códigos numéricos.
- Linguagem *assembly* exige o uso de montadores: programas que traduzem a linguagem *assembly* em linguagem de máquina.

LINGUAGENS DE BAIXO-NÍVEL

- Como cada processador tem seu próprio conjunto de instruções, também tem seu próprio montador.
- Isto significa que um programa em linguagem *assembly* só pode ser escrito para um tipo particular de máquina.
- Linguagem de máquina e linguagem *assembly* são chamadas linguagens de baixo-nível.

LINGUAGENS DE ALTO-NÍVEL

•As linguagens de alto nível surgiram para resolver os problemas (principalmente de entendimento) das linguagens *Assembly*.

•As linguagens de ato nível, são mais naturais para o ser humano, sendo composta de palavras mais próximas do seu vocabulário e são independentes de máquina.

•Estas facilidades permitem ao programador se preocupar apenas com o problema em particular e não como traduzí-lo para o nível de compreensão da máquina

LINGUAGENS DE ALTO-NÍVEL

Por exemplo, escrever

A = B + C

é mais fácil do que escrever

MOV @C, R1;

ADD @B, R1;

MOV R1, @A;

LINGUAGENS DE ALTO-NÍVEL

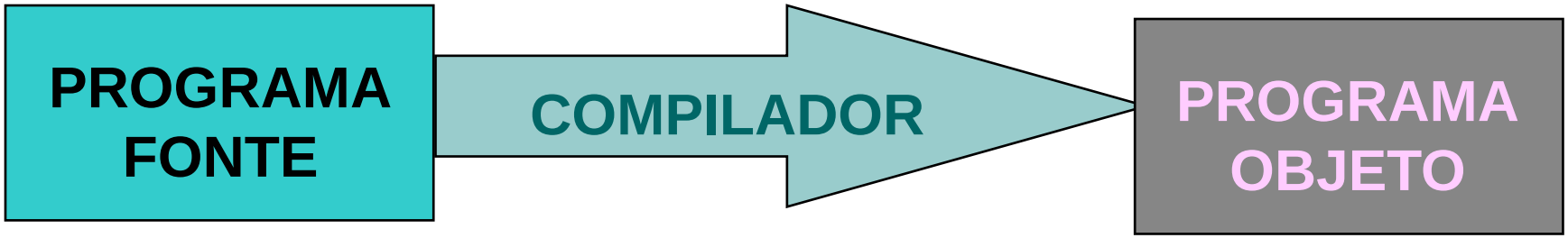
- As linguagens de alto-nível foram desenvolvidas nos meados dos anos 50. A primeira delas foi FORTRAN, seguida pelo ALGOL e LISP.
- As linguagens de alto-nível modernas incluem Pascal, C, C++, Java, ...
- Estas linguagens requerem o uso de compiladores (programas que traduzem o código fonte de alto-nível na linguagem de máquina).
- Cada instrução do código fonte pode corresponder a várias instruções da linguagem de máquina.

LINGUAGENS e NÍVEIS

TRADUÇÃO/COMPILAÇÃO

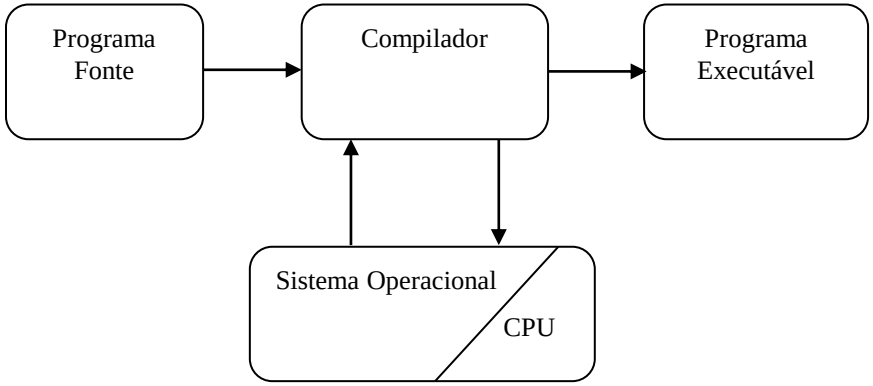
- TRADUÇÃO → CONSISTE EM SUBSTITUIR CADA INSTRUÇÃO Do progama em alto nível POR UMA SEQUÊNCIA EQUIVALENTE DE INSTRUÇÕES EM Linguagem de baixo nível.

- **Programa Fonte**
  - **Instruções de alto nível**
    - **IF/THEN/ELSE**
    - **A:=B+C+D**
- **Programa Objeto**
  - **Instruções básicas**

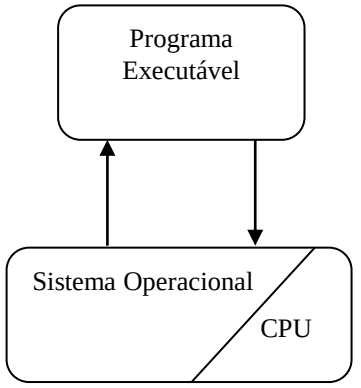


LINGUAGENS e NÍVEIS

Compilador: conversor de programas escritos em uma linguagem de programação (alto nível) para programas em linguagem de máquina (baixo nível). Uma vez que o programa foi convertido para código de máquina, este pode ser executado independente do compilador e do programa original.



Geração do Programa Executável



Execução do Programa

LINGUAGENS e NÍVEIS

Vimos que: O compilador traduz o programa de alto nível em uma sequência de **instruções de processador**. O resultado desta tradução é o programa em **Linguagem de montagem** ou **linguagem de máquina (***assembly language***)**.

A linguagem de montagem é uma forma de representar textualmente as instruções oferecidas pela arquitetura. Cada arquitetura possui uma linguagem de montagem particular. No programa em linguagem de montagem, as instruções são representadas através de mnemônicos, que associam o nome da instrução à sua função, por exemplo, ADD ou SUB, isto é soma e subtração, respectivamente.

O programa em linguagem de montagem é convertido para um programa em **código objeto** pelo **montador** (*assembler*). O montador traduz diretamente uma instrução da forma textual para a forma de código binário. É sob a forma binária que a instrução é carregada na memória e interpretada pelo processador.

LINGUAGENS e NÍVEIS

Comando de alto nível: `A = A + 1;`

Código *assembly*:

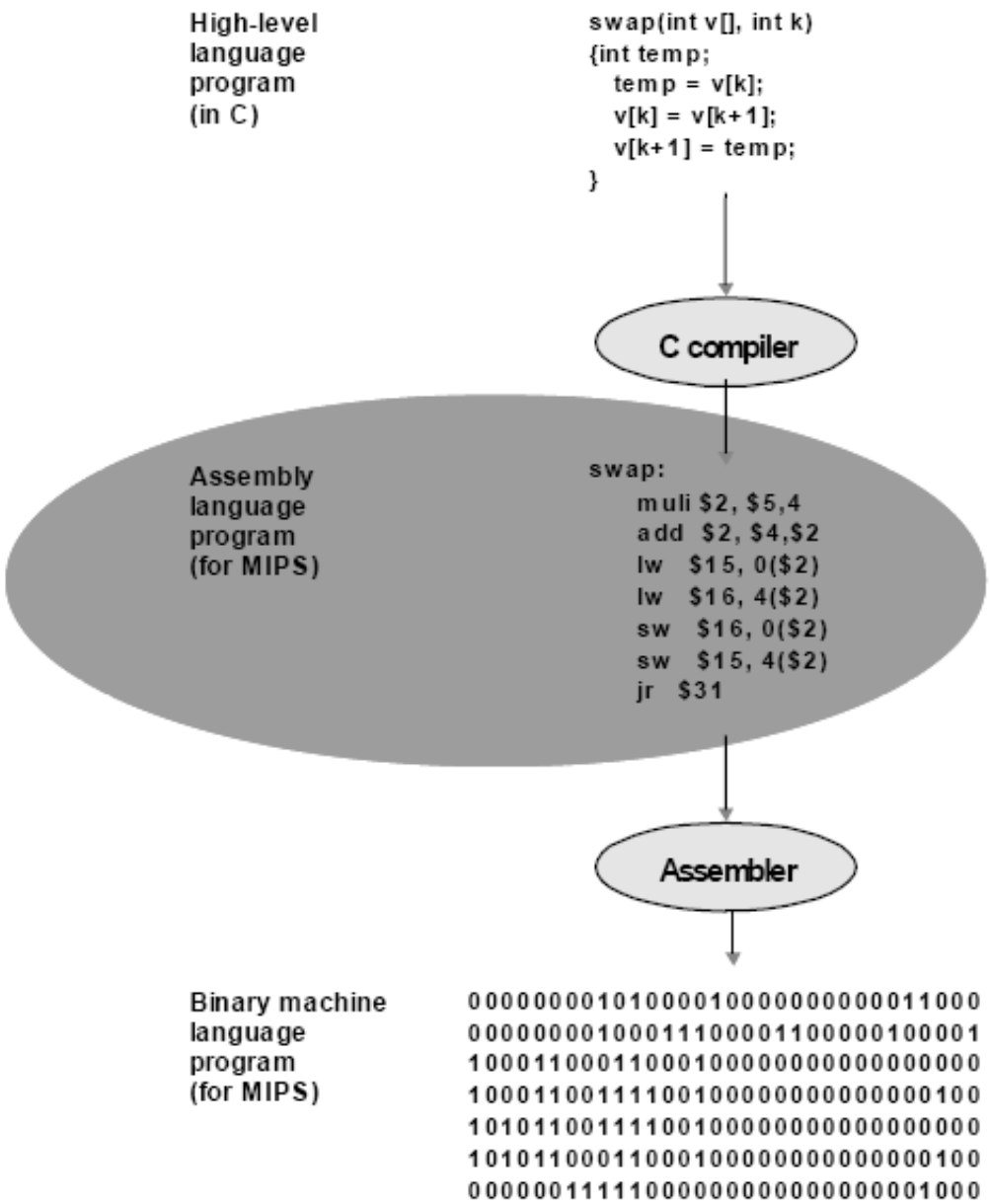
```
LD   A      ; ACC← A
ADDI 1      ; ACC←ACC+1
STO  A      ; A  ←ACC
```

Comando de alto nível: `A = A + B - 3;`

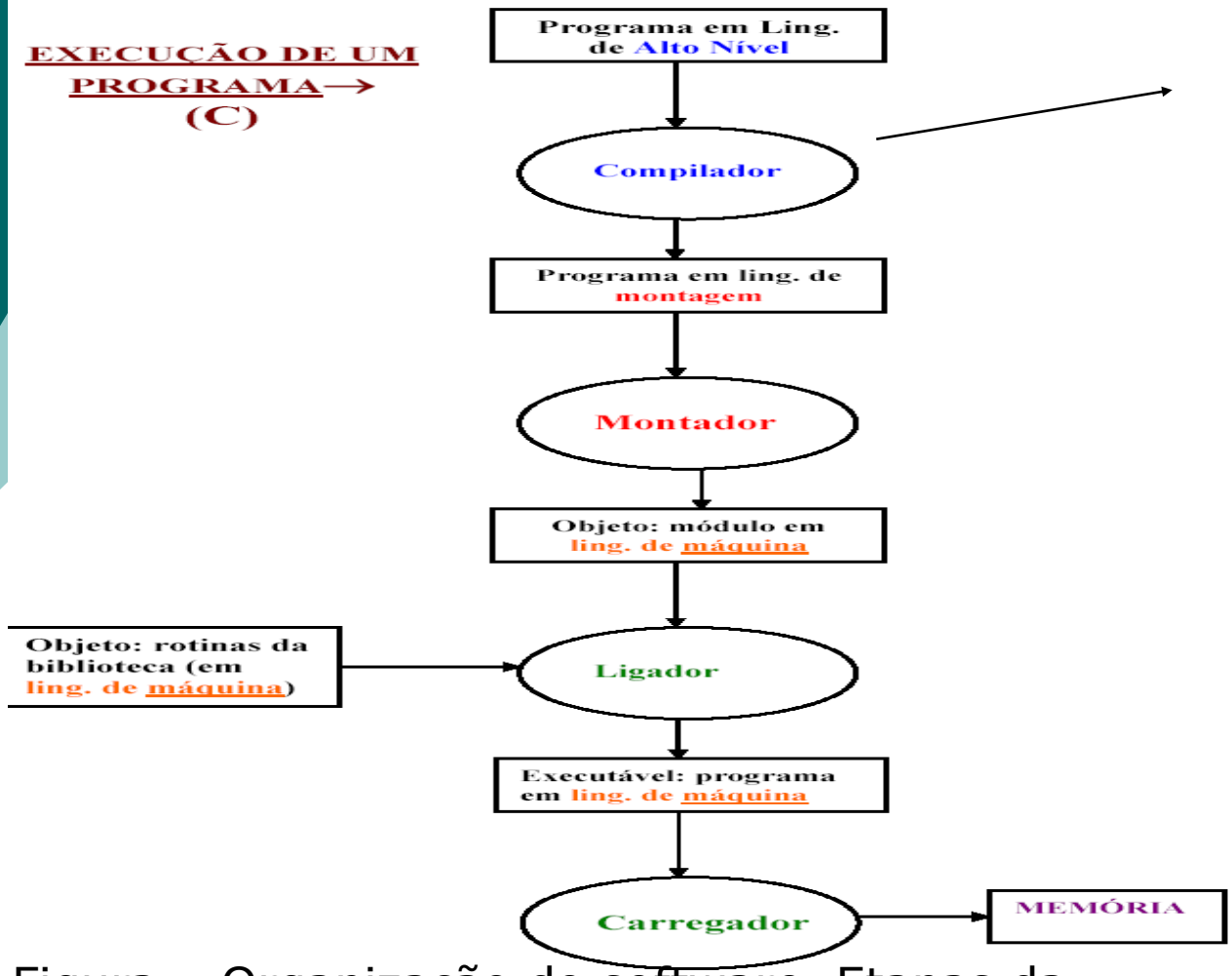
Código *assembly*:

```
LD   A   ; ACC←A
ADD  B   ; ACC←ACC+B
SUBI 3   ; ACC←ACC-3
STO  A   ; A  ←ACC
```

Compilador/Interpretador



Execução de Programas



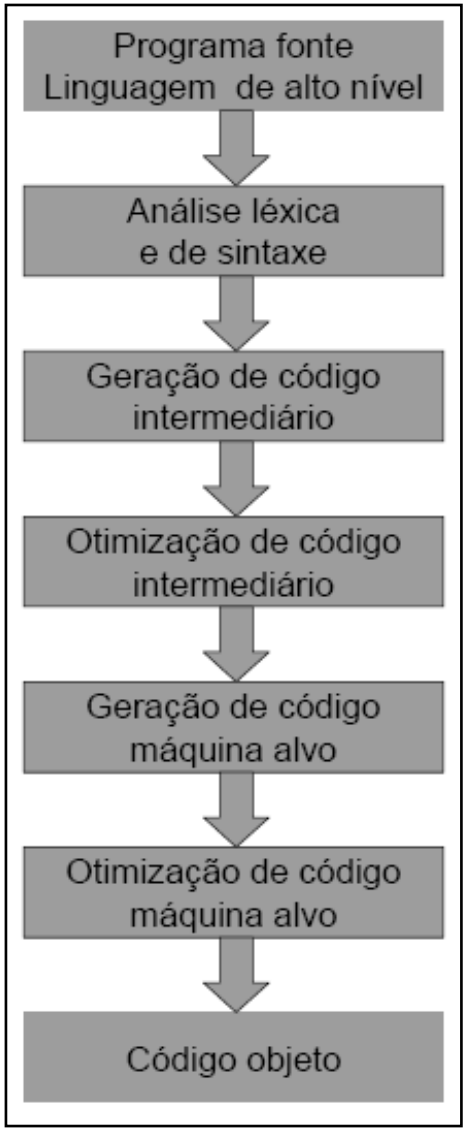
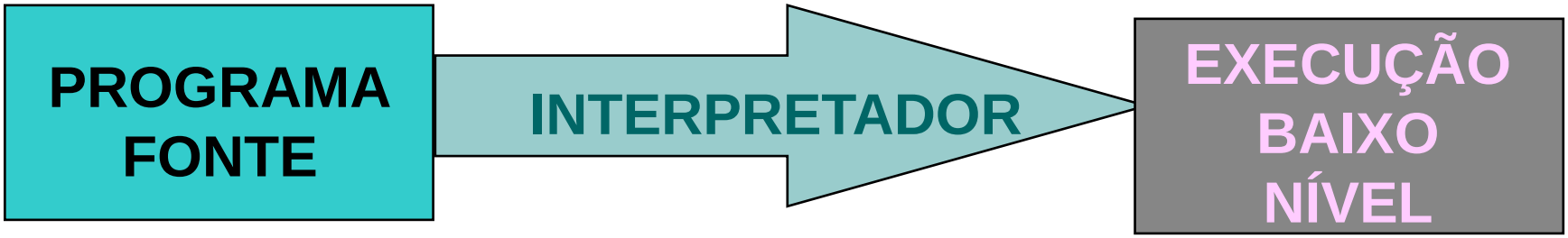


Figura – Organização do software. Etapas da execução de um programa (adaptado de Patterson e Hennessy, 3ª ed., 2005, pág.78).

LINGUAGENS e NÍVEIS

INTERPRETAÇÃO:

- Recebem um programa fonte escrito em uma linguagem DE ALTO NÍVEL, e o executam IMEDIATAMENTE.
- Lê, analisa e executa as instruções do programa fonte, ***uma de cada vez***.



LINGUAGENS e NÍVEIS

•A TRADUÇÃO E INTERPRETAÇÃO SÃO SIMILARES → em ambos os métodos as instruções em L1 são executadas pelas sequências equivalentes de instruções em L0. Ambos os métodos são utilizados.

A DIFERENÇA:

=>NA TRADUÇÃO(quando há compilação), o programa em L1 primeiro é convertido para L0 (“descarta-se”, então, o programa em L1) e o programa em L0 é carregado para a memória e executado.

=>NA INTERPRETAÇÃO, uma instrução em linguagem L1 é executada imediatamente após ter sido analisada e reconhecida pelo interpretador. Neste caso não há geração de um novo programa.

LINGUAGENS e NÍVEIS

	Vantagens	Desvantagens
Compiladores	Execução mais rápida	Várias etapas de tradução
	Permite estruturas de programação mais completas	Programação final é maior, necessitando mais memória para a sua execução
	Permite a otimização do código fonte	Processo de correção de erros e depuração é mais demorado
Interpretadores	Depuração do programa é mais simples	Execução do programa é mais lenta
	Consome menos memória	Estruturas de dados demasiado simples
	Resultado imediato do programa ou rotina desenvolvida	Necessário fornecer o programa fonte ao utilizador

Instruções de Máquina

RISC: abreviação de *Reduce Instruction Set Computer*, computador com conjunto reduzido de instruções.

A arquitetura RISC é constituída por um pequeno conjunto de instruções simples que são executadas diretamente pelo hardware, sem a intervenção de um interpretador (microcódigo), ou seja, as instruções são executadas em apenas uma microinstrução.

As máquinas RISC só se tornaram viáveis devido aos avanços de software no aparecimento de compiladores otimizados .

Características RISC

Características da arquitetura RISC	
Características	**Considerações**
Menor quantidade de instruções que as máquinas CISC	Simplifica o processamento de cada instrução e torna este item mais eficaz.
Execução otimizada de chamadas de funções	Embora o processadores RS/6000 possua 184 instruções, ainda assim é bem menos que as 303 instruções do sistemas VAX-11. Além disso, a maioria das instruções é realizada em 1 ciclo de relógio, o que é considerado o objetivo maior dessa arquitetura.
Menor quantidade de modos	As instruções RISC utilizam os registradores da UCP (em maior quantidade que os processadores CISC) para armazenar parâmetros e variáveis em chamadas de rotinas e funções. Os processadores CISC usam mais a memória para a tarefa.
Utilização em larga escala de *pipelining*	Um dos fatores principais que permite aos processadores RISC atingir seu objetivo de completar a execução de uma instrução pelo menos a cada ciclo de relógio é o emprego de *pipelining* em larga escala

Arquitetura CISC

CISC: *Complex Instruction Set Computer*, computador com conjunto complexo de instruções.

Vários processadores, até o Pentium, utilizam a tecnologia denominada CISC.

Esta classe de processadores possui um conjunto de instruções grande e uma área denominada micro-código, responsável por armazenar como o processador deve manipular cada instrução individualmente.

À medida em que novas instruções eram acrescidas, o decodificador de instruções do processador tinha que ficar mais complexo, o que o tornava mais lento.

O micro-código ficava maior, o que acarretava, além da lentidão, um processador fisicamente maior e mais difícil de ser construído. Isto quer dizer que, paradoxalmente, quanto mais "poderoso" fosse o processador, mais lento e difícil de ser construído ele ficaria.

Arquitetura CISC

Curiosidade … Processadores Intel:

Para driblar o problema, de lentidão e do tamanho, a Intel inovava seus processadores com características específicas de aumento de performance, como o cache de memória interno e arquitetura superescalar (o Pentium funciona como se fosse dois processadores trabalhando em paralelo; ele é capaz de executar duas instruções por pulso de clock).

Instruções CISC

Amostra das instruções internas do Pentium II

Moves

MOV DST,SRC	Move SRC to DST
PUSH SRC	Push SRC onto the stack
POP DST	Pop a word from the stack to DST
XCHG DS1,DS2	Exchange DS1 and DS2
LEA DST,SRC	Load effective addr of SRC into DST
CMOV DST,SRC	Conditional move

Arithmetic

ADD DST,SRC	Add SRC to DST
SUB DST,SRC	Subtract DST from SRC
MUL SRC	Multiply EAX by SRC (unsigned)
IMUL SRC	Multiply EAX by SRC (signed)
DIV SRC	Divide EDX:EAX by SRC (unsigned)
IDIV SRC	Divide EDX:EAX by SRC (signed)
ADC DST,SRC	Add SRC to DST, then add carry bit
SBB DST,SRC	Subtract DST & carry from SRC
INC DST	Add 1 to DST
DEC DST	Subtract 1 from DST
NEG DST	Negate DST (subtract it from 0)

Binary coded decimal

DAA	Decimal adjust
DAS	Decimal adjust for subtraction
AAA	ASCII adjust for addition
AAS	ASCII adjust for subtraction
AAM	ASCII adjust for multiplication
AAD	ASCII adjust for division

Boolean

AND DST,SRC	Boolean AND SRC into DST
OR DST,SRC	Boolean OR SRC into DST
XOR DST,SRC	Boolean Exclusive OR SRC to DST
NOT DST	Replace DST with 1's complement

Shift/rotate

SAL/SAR DST,#	Shift DST left/right # bits
SHL/SHR DST,#	Logical shift DST left/right # bits
ROL/ROR DST,#	Rotate DST left/right # bits
RCL/RCR DST,#	Rotate DST through carry # bits

Test/compare

TST SRC1,SRC2	Boolean AND operands, set flags
CMP SRC1,SRC2	Set flags based on SRC1 - SRC2

Transfer of control

JMP ADDR	Jump to ADDR
Jxx ADDR	Conditional jumps based on flags
CALL ADDR	Call procedure at ADDR
RET	Return from procedure
IRET	Return from interrupt
LOOPxx	Loop until condition met
INT ADDR	Initiate a software interrupt
INTO	Interrupt if overflow bit is set

Strings

LODS	Load string
STOS	Store string
MOVS	Move string
CMPS	Compare two strings
SCAS	Scan Strings

Condition codes

STC	Set carry bit in EFLAGS register
CLC	Clear carry bit in EFLAGS register
CMC	Complement carry bit in EFLAGS
STD	Set direction bit in EFLAGS register
CLD	Clear direction bit in EFLAGS reg
STI	Set interrupt bit in EFLAGS register
CLI	Clear interrupt bit in EFLAGS reg
PUSHFD	Push EFLAGS register onto stack
POPFD	Pop EFLAGS register from stack
LAHF	Load AH from EFLAGS register
SAHF	Store AH in EFLAGS register

Miscellaneous

SWAP DST	Change endianness of DST
CWQ	Extend EAX to EDX:EAX for division
CWDE	Extend 16-bit number in AX to EAX
ENTER SIZE,LV	Create stack frame with SIZE bytes
LEAVE	Undo stack frame built by ENTER
NOP	No operation
HLT	Halt
IN AL,PORT	Input a byte from PORT to AL
OUT PORT,AL	Output a byte from AL to PORT
WAIT	Wait for an interrupt

SRC = source
DST = destination

# = shift/rotate count
LV = # locals

Instruções RISC

As principais instruções inteiras do UltraSPARC II

Loads

LDSB ADDR,DST	Load signed byte (8 bits)
LDUB ADDR,DST	Load unsigned byte (8 bits)
LDSH ADDR,DST	Load signed halfword (16 bits)
LDUH ADDR,DST	Load unsigned halfword (16)
LDSW ADDR,DST	Load signed word (32 bits)
LDUW ADDR,DST	Load unsigned word (32 bits)
LDX ADDR,DST	Load extended (64-bits)

Stores

STB SRC,ADDR	Store byte (8 bits)
STH SRC,ADDR	Store halfword (16 bits)
STW SRC,ADDR	Store word (32 bits)
STX SRC,ADDR	Store extended (64 btis)

Arithmetic

ADD R1,S2,DST	Add
ADDCC "	Add and set icc
ADDC "	Add with carry
ADDCCC "	Add with carry and set icc
SUB R1,S2,DST	Subtract
SUBCC "	Subtract and set icc
SUBC "	Subtract with carry
SUBCCC "	Subtract with carry and set icc
MULX R1,S2,DST	Multiply
SDIVX R1,S2,DST	Signed divide
UDIVX R1,S2,DST	Unsigned divide
TADCC R1,S2,DST	Tagged add

Shifts/rotates

SLL R1,S2,DST	Shift left logical (64 bits)
SLLX R1,S2,DST	Shift left logical extended (64)
SRL R1,S2,DST	Shift right logical (32 bits)
SRLX R1,S2,DST	Shift right logical extended (64)
SRA R1,S2,DST	Shift right arithmetic (32 bits)
SRAX R1,S2,DST	Shift right arithmetic ext. (64)

Boolean

AND R1,S2,DST	Boolean AND
ANDCC "	Boolean AND and set icc
ANDN "	Boolean NAND
ANDNCC "	Boolean NAND and set icc
OR R1,S2,DST	Boolean OR
ORCC "	Boolean OR and set icc
ORN "	Boolean NOR
ORNCC "	Boolean NOR and set icc
XOR R1,S2,DST	Boolean XOR
XORCC "	Boolean XOR and set icc
XNOR "	Boolean EXCLUSIVE NOR
XNORCC "	Boolean EXCL. NOR and set icc

Transfer of control

BPcc ADDR	Branch with prediction
BPr SRC,ADDR	Branch on register
CALL ADDR	Call procedure
RETURN ADDR	Return from procedure
JMPL ADDR,DST	Jump and Link
SAVE R1,S2,DST	Advance register windows
RESTORE "	Restore register windows
Tcc CC,TRAP#	Trap on condition
PREFETCH FCN	Prefetch data from memory
LDSTUB ADDR,R	Atomic load/store
MEMBAR MASK	Memory barrier

Miscellaneous

SETHI CON,DST	Set bits 10 to 31
MOVcc CC,S2,DST	Move on condition
MOVr R1,S2,DST	Move on register
NOP	No operation
POPC S1,DST	Population count
RDCCR V,DST	Read condition code register
WRCCR R1,S2,V	Write condition code register
RDPC V,DST	Read program counter

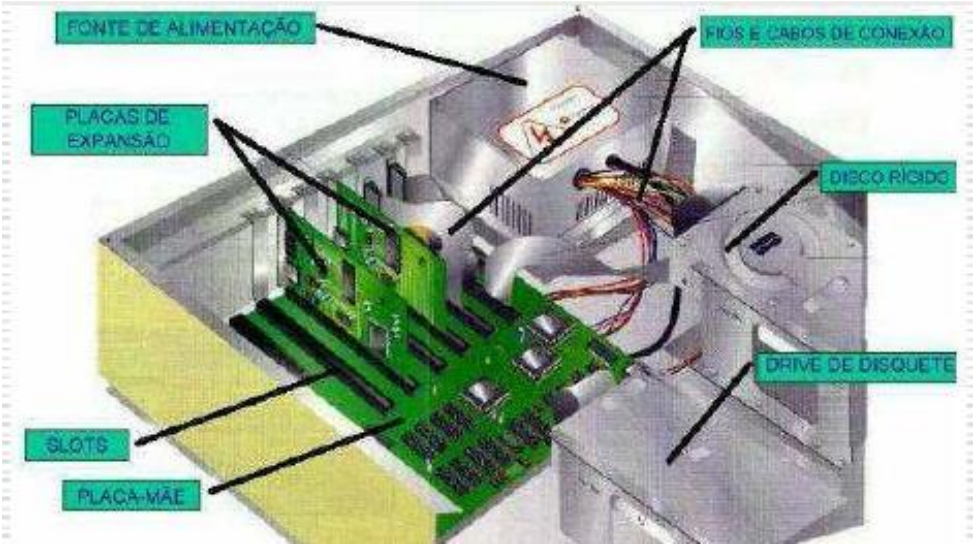
SRC = source register
DST = destination register
R1 = source register
S2 = source: register or immediate
ADDR = memory address

TRAP# = trap number
FCN = function code
MASK = operation type
CON = constant
V = register designator

CC = condition code set
R =destination register

cc =condition
r = LZ,LEZ,Z,NZ,GZ,GEZ

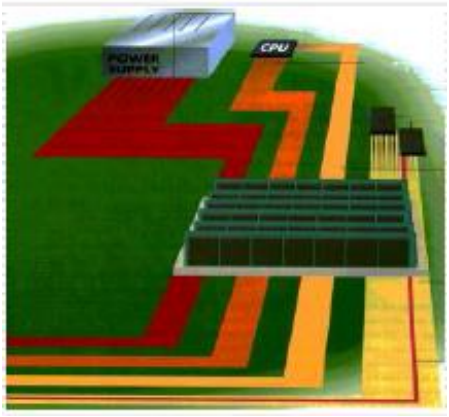
Comunicação entre os dispositivos do Computador - Barramentos



Barramentos

Conjunto de condutores elétricos que interligam os diversos componentes de um computador e de circuitos eletrônicos que controlam o fluxo dos bits;

Pelo barramento trafegam dados, endereços de memória, sinais de controle e energia;



Classificação de Barramentos

- Em função da **número de fios** que transportam a informação. São classificados em barramentos **Seriais** e **Paralelo**s.

- Em função do tipo de **informação transferida** existem, por exemplo, os barramentos de **Dados**, de **Endereço**, de **Controle** e **Multiplexados**.

- Em função da **localização física**, podem ser citados os **barramentos internos** às placas, os **barramentos globais** e os barramentos de **Entrada e Saída**.

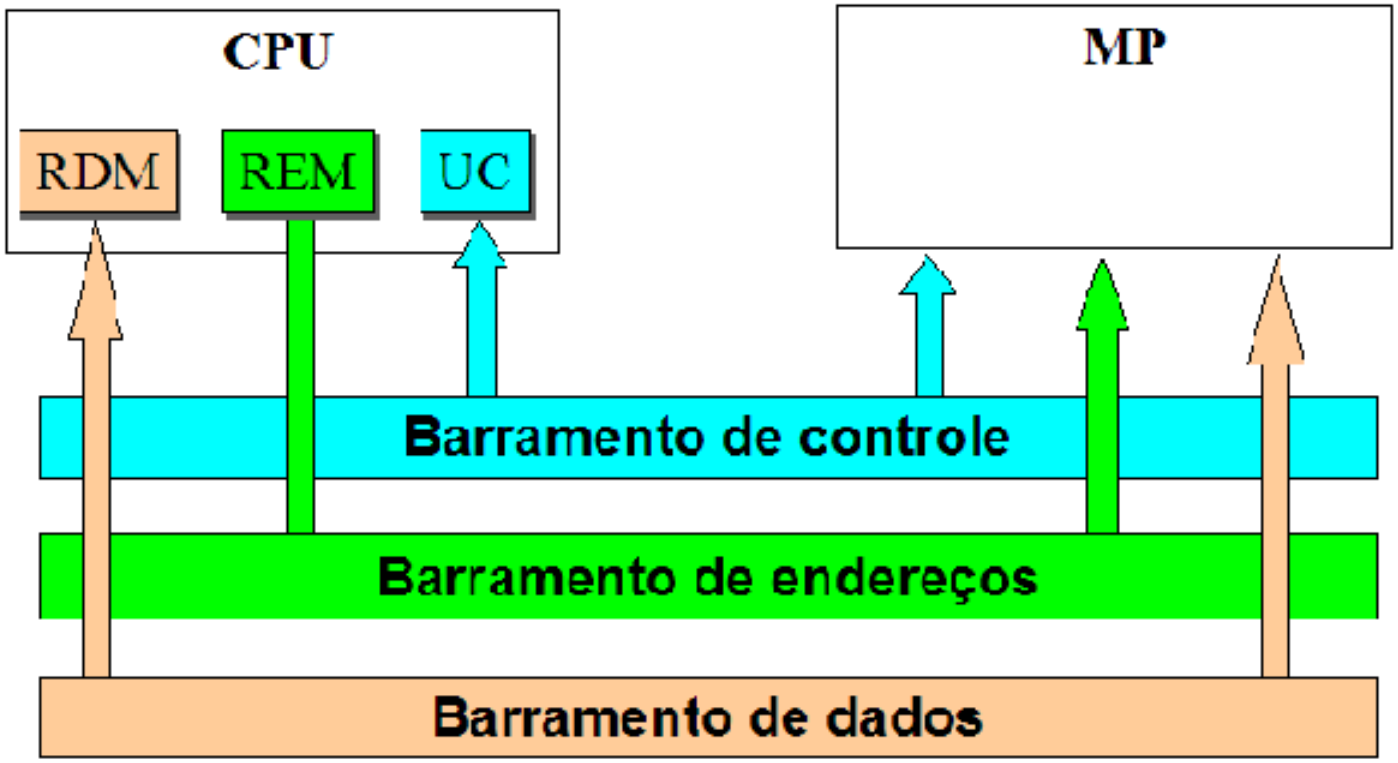
Tipos de Barramentos

Barramento Local: Principal barramento. Conecta o processador, a memória RAM, as caches e os chipsets;

Barramento X: Barramento que conecta os periféricos integrados na placa-mãe (placa de som, vídeo, rede, etc) ao barramento local;

Barramento de Expansão: conectam as placas controladoras dos periféricos ao barramento local.

Tipos de Barramentos – Internos Processador



Tipos de Barramentos

Barramento de dados. Possui, em geral, uma linha (fio) para cada bit de dados. Num microcomputador a largura do barramento de dados (*bus width*) é o fator de especificação de barramento mais lembrado. Se diz por exemplo: "O microcomputador PC tem um barramento de 8bits". Ou então: "O microprocessador 8086 tem um barramento de 16 bits e o microprocessador 80386 tem um barramento de 32 bits".

Tipos de Barramentos

Barramento de Endereçamento. Possui uma linha (fio) para cada bit de endereçamento. A largura do barramento de endereçamento é importante para determinar a capacidade máxima de posições de memória acessíveis. A quantidade de posições de memória endereçáveis *q* é igual a 2^n , onde *n* é o número de bits do barramento de endereçamento. Por exemplo, barramento de endereçamento com:

n bits	**q = 2^n**	
16 bits	**65 536**	**64 kilo posições**
20 bits	**1 048 576**	**1 mega posições**
24 bits	**16 777 216**	**16 mega posições**
32 bits	**4 294 967 296**	**4 giga posições**

Tipos de Barramentos

Barramento de Controle. Agrupa todos os sinais necessários ao controle da transferência de informação entre as unidades do sistema.

O barramento de controle é constituído de inúmeras linhas pela quais fluem sinais específicos da linhas pela quais fluem sinais específicos da programação do sistema.

Barramento de Controle

Sinais de controle comumente empregados nos comumente empregados nos barramentos de controle são:

- **Leitura de dados** ("Memory read")- sinaliza para o controlador de memória decodificar o endereço colocado no barramento de endereços e transferir o conteúdo do barramento de dados para as células para o barramento de dados.
- **Escrita de dados** ("Memory write")- sinaliza para o controlador de memória decodificar colocado no barramento de endereços e transferir o conteúdo do barramento de dados para as células especificadas.

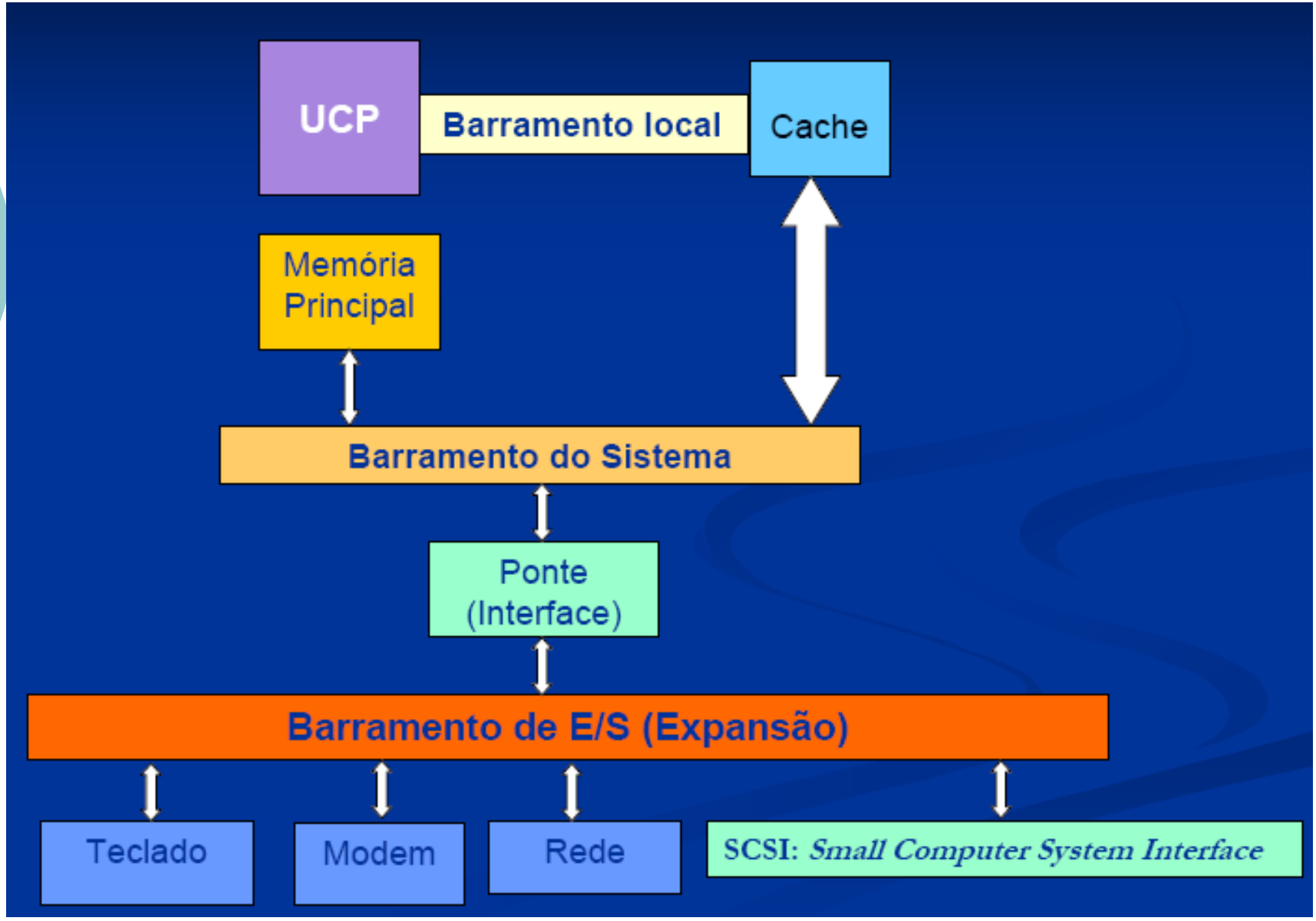
Barramento de Controle

- **Leitura de E/S (“I/O read”)** – processo semelhante ao de leitura de dados na memória.
- **Escrita de E/S (“I/O” write)** – processo semelhante ao de leitura de dados na memória.
- **Certificação de transferência de dados (“transfer ACK”)** – o dispositivo acusa o término da transferência para a UCP
- **Pedido de interrupção (“Interrupt request”)** – Indica ocorrência de uma interrupção.
- **Relógio (“clok”)** – por onde passam os pulsos de sincronização dos eventos durante o funcionamento do sistema.

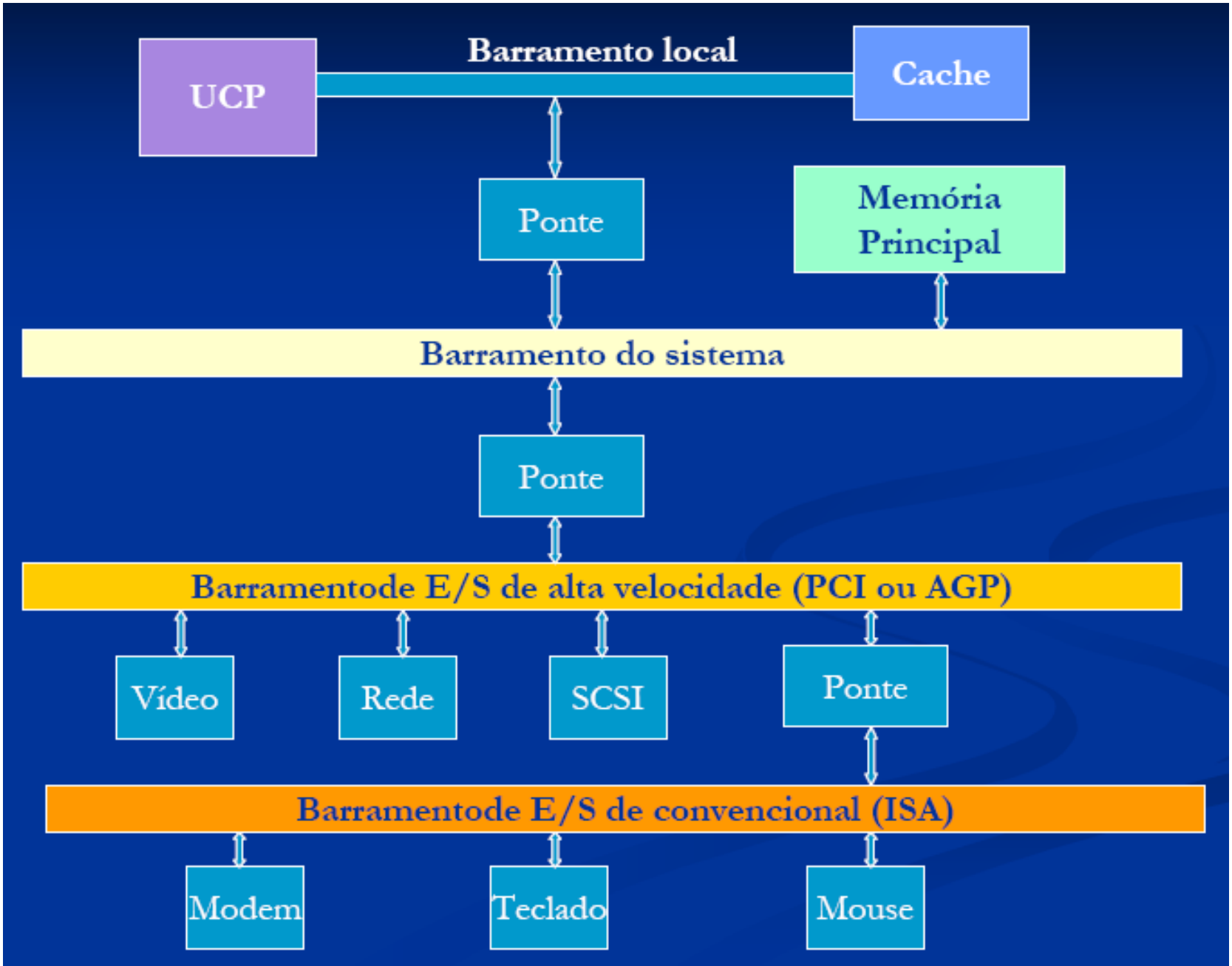
Tipos de Barramento

Atualmente os modelos de organização de sistemas de computação adotados pelos fabricantes possuem diferentes tipos de barramento:

Tipos de Barramentos



Tipos de Barramentos



Tipos de Barramentos

O Barramento de Entrada e Saída (ou E/S) é um conjunto de circuitos e linhas de comunicação que possibilitam a ligação dos periféricos com a parte interna do computador (UCP e *chipset* – placa mãe). Este barramento normalmente segue padrões internacionais e as frequências de transferência são mais baixas. São exemplos de barramentos de entrada e saída: **VESA, ISA, MCA, EISA, VLB, PCI, AGP, PCI Express.**

Padrões de Barramentos de Expansão

ISA (Industry Standard Architecture): O primeiro barramento de expansão (8 e 16 bits); Ainda usado para periféricos mais lentos como placa de som e fax modem;

MCA (Microchannel Architecture): Barramento proprietário IBM (32 bits);

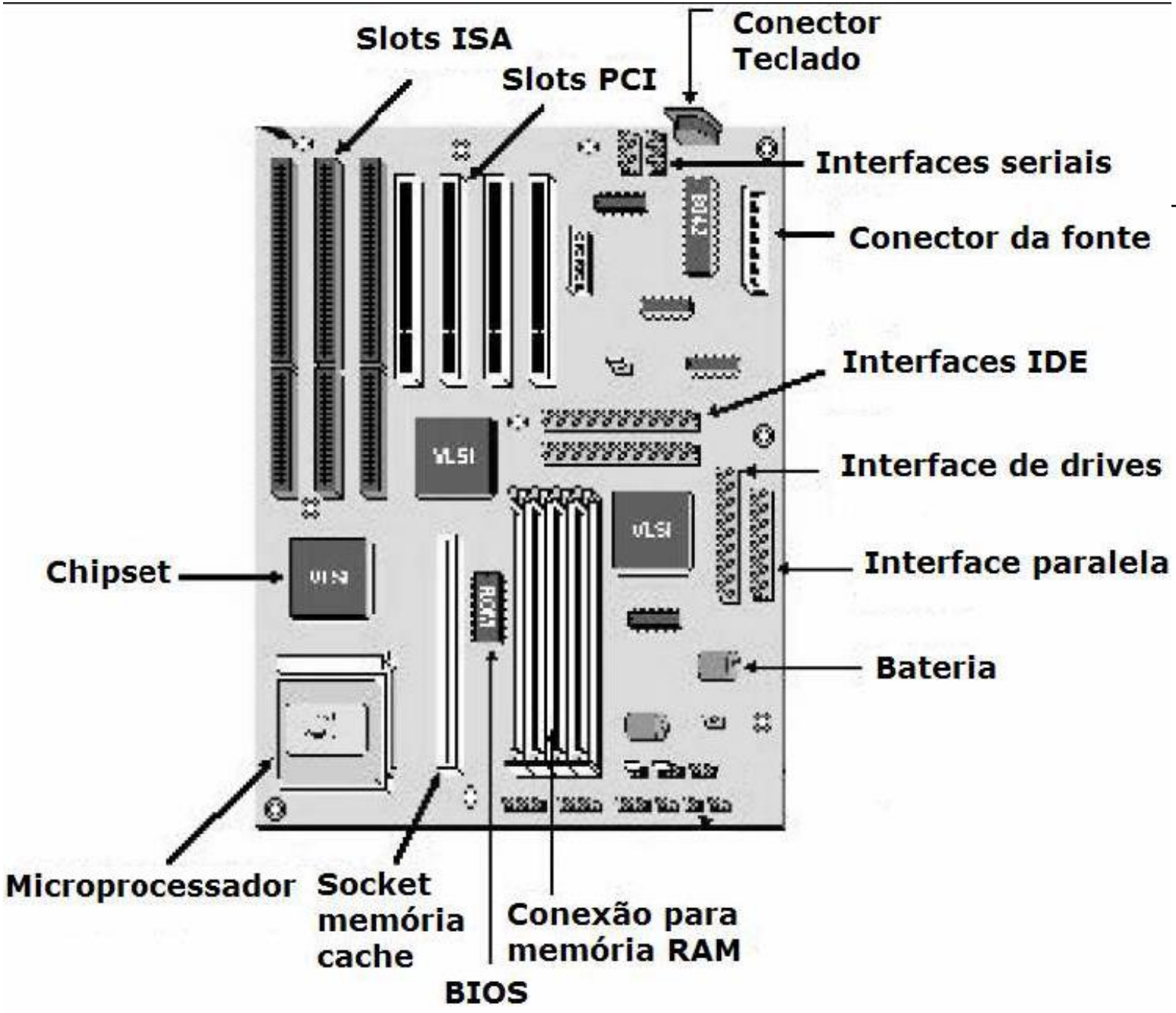
EISA (Extended Industry Standard Architecture): 32 bits e suporte a barramento ISA;

VLB (Vesa Local Bus): 32 bits. Criado pelos fabricantes de interface de vídeo;

PCI (Peripheral Component Interconnect): 32 a 64 bits, suporte a barramento ISA;

AGP (Accelerated Graphics Port):Desenvolvido para as placas de vídeo mais modernas (3D) e processadores Pentium II; 2 vezes mais rápido que o PCI;

USB (Universal Serial Bus): Padrão para a conexão de periféricos externos; Facilidade de uso; Possibilidade de conectar vários periféricos a uma única porta USB;


Slots ISA
Slots PCI
Conector Teclado
Interfaces seriais
Conector da fonte
Interfaces IDE
Interface de drives
Interface paralela
Bateria
Chipset
Microprocessador
Socket memória cache
Conexão para memória RAM
BIOS