



ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

Introdução

Profª. Fabiana F F Peres

Apoio: Camile Bordini

Introdução

- Os **problemas computacionais** são resolvidos por computadores através da execução de **programas**
- **Programa** é um conjunto de **instruções/comandos** que descreve a maneira de um computador realizar determinada tarefa.

```
import java.io.*;  
import java.math.*;  
import java.net.*;
```

Introdução

- Também dizemos que uma **sequência de sinais elétricos** corresponde a uma **instrução**
- Porém, muito mais **simples** que escritas nos programas por programadores!
 - Ex: **10001011** - > ordena para que seja feito algo, como uma operação



*Cuidar com a
duplicação de
termos!*

Introdução

- Os circuitos eletrônicos de um computador reconhecem e executam um **conjunto limitado de instruções**
- Exemplos?

Introdução

- Os circuitos eletrônicos de um computador reconhecem e executam um **conjunto limitado de instruções**
- Exemplos?
 - Somar 2 números, comparar um número com 0, copiar um conjunto de dados de uma parte da memória para outra
 - *Em geral, as instruções básicas de um computador não são mais complicadas do que isso!*
- Essas instruções em conjunto formam a **linguagem de máquina (binária)**

Introdução

- Os *projetistas de computadores* então devem decidir quais instruções farão parte de sua linguagem de máquina
- Quais os **requisitos** que estas **instruções** deveriam ter?



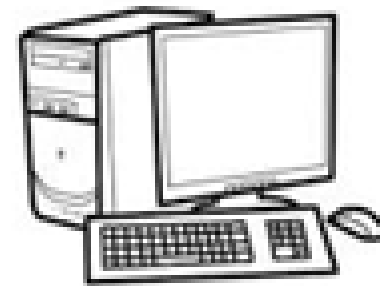
Introdução

- Os *projetistas de computadores* então devem decidir quais instruções farão parte de sua linguagem de máquina
- Quais os **requisitos** que estas **instruções** deveriam ter?
 - Tão **simples** quanto possível
 - compatíveis com o **uso** pretendido da máquina
 - compatíveis com o **desempenho** requerido
 - que o custo e a complexidade da **eletrônica** necessária sejam reduzidos

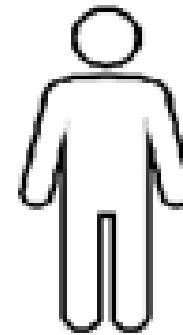


Problema

- Devido às **instruções da linguagem de máquina** serem muito **SIMPLES**...



- ... isto torna-as distantes de uma linguagem natural (humana)



Problema

Complexidade do que as pessoas precisam fazer

versus

simplicidade do conjunto de instruções do computador

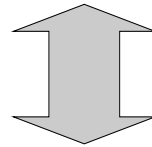
Como resolver?



Solução



Criar uma *hierarquia de abstrações* de níveis mais altos baseados nos níveis mais baixos.



Metodologia da “*organização estruturada de computadores*”

Organização Estrutura de Computadores

Solução

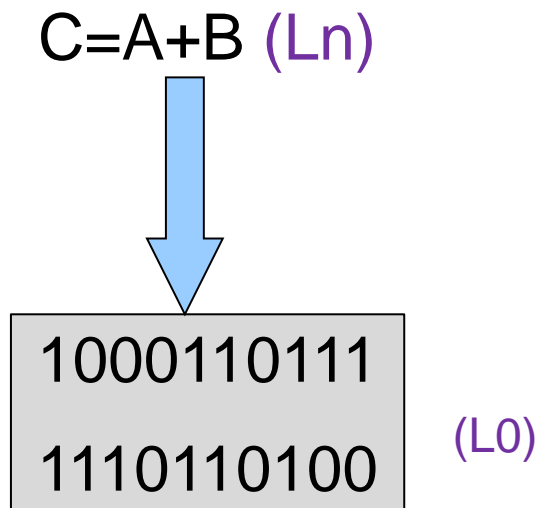
- Como:

As pessoas desejam fazer X,
mas os computadores só podem fazer Y

- Por isso, devem ser especificados:
 - **Níveis** ou **Máquinas Virtuais** (sinônimos para nós)
 - **Linguagens** para cada nível

Solução

- Supondo a existência das seguintes **linguagens**:
 - **Ln**: linguagem mais natural ao usuário (alto nível e complexa)
 - **L0**: linguagem da máquina (baixo nível e simples)
- Como compatibilizar **Ln** com **L0**?



Necessário:

- **Tradutor**
- **Ou Interpretador**
- **Ou Metodologia híbrida**

Tradutor

- É um programa escrito normamente na linguagem L_{n-1}
- Recebe um programa fonte escrito na linguagem L_n
- Para cada instrução do programa fonte o **tradutor** gera um novo conjunto de instruções equivalentes na linguagem alvo (L_{n-1}).

*Obs: o programa fonte pode ser traduzido
uma única vez e executado várias vezes*

Tradutores

- O **montador** e os **compiladores** são tradutores.
- **Montador** são tradutores simples.
- **Compiladores** são tradutores complexos divididos em várias etapas.

Interpretador

- É um programa também escrito em L_{n-1}
- Recebe um programa fonte escrito na linguagem L_n
- Cada instrução do programa fonte é **interpretada** para uma instrução L_{n-1} carregada na memória e executada diretamente (antes mesmo da próxima instrução L_n ser lida)
- Diferente dos **tradutores**, **interpretadores** **NÃO** criam um programa na linguagem L_{n-1}

Interpretação

- O próprio interpretador controla o computador durante a execução do programa na linguagem L_n
- O programa em L_n é tratado como dado de entrada

Obs: o programa fonte deve ser interpretado cada vez que for executado

Cada vez mais comum a metodologia híbrida

Níveis de Abstração

- Voltando aos **níveis de abstração**, junto às traduções ou interpretações, podemos imaginar a existência de:
 - uma **máquina virtual M_n** , cuja linguagem de máquina seja L_n ;
 - outra **máquina virtual M_{n-1}** , com linguagem L_{n-1}
 - ...
 - até chegarmos no nível da **máquina real M_0** , com as **instruções de máquina L_0**

Níveis de Abstração

- *Questão:* seria possível a construção de uma **máquina real** com uma linguagem de máquina em C++ ou COBOL?

Níveis de Abstração

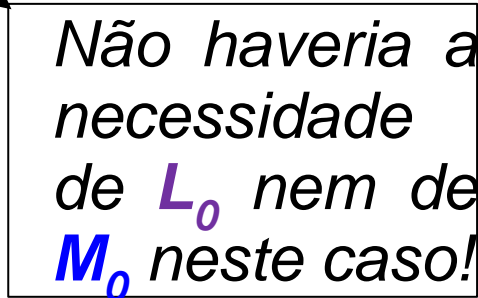
- *Questão:* seria possível a construção de uma **máquina real** com uma linguagem de máquina em C++ ou COBOL?
- *Sim, sua construção física a partir de circuitos eletrônicos seria perfeitamente possível com a tecnologia atual. Porém, há uma boa razão para que não seja feita: a relação custo/benefício não seria favorável*
- *Muito complexa e cara!*

Níveis de Abstração

- Assim, os programas podem ser escritos em L_n , para a máquina M_n e...

... serem executados diretamente em M_n (se esta máquina não fosse complicada nem cara de ser construída)

ou

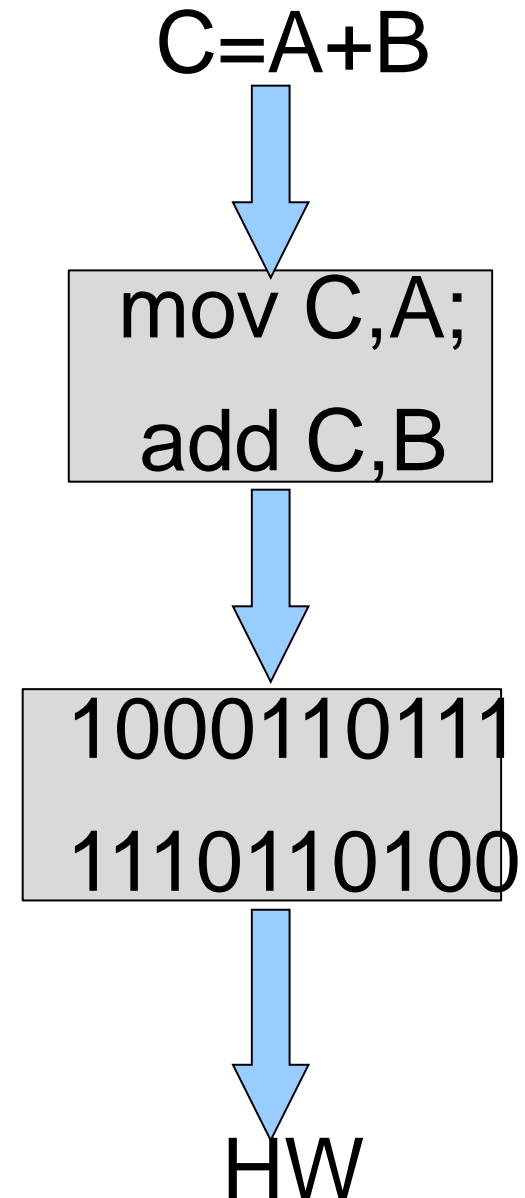


Não haveria a necessidade de L_0 nem de M_0 neste caso!

... serem traduzidos ou interpretados até a linguagem de máquina L_0 e executados em M_0

Níveis de Abstração

- Ou seja, no fim das contas, as pessoas podem escrever programas na linguagem L_n , independente se M_n existe realmente ou virtualmente!

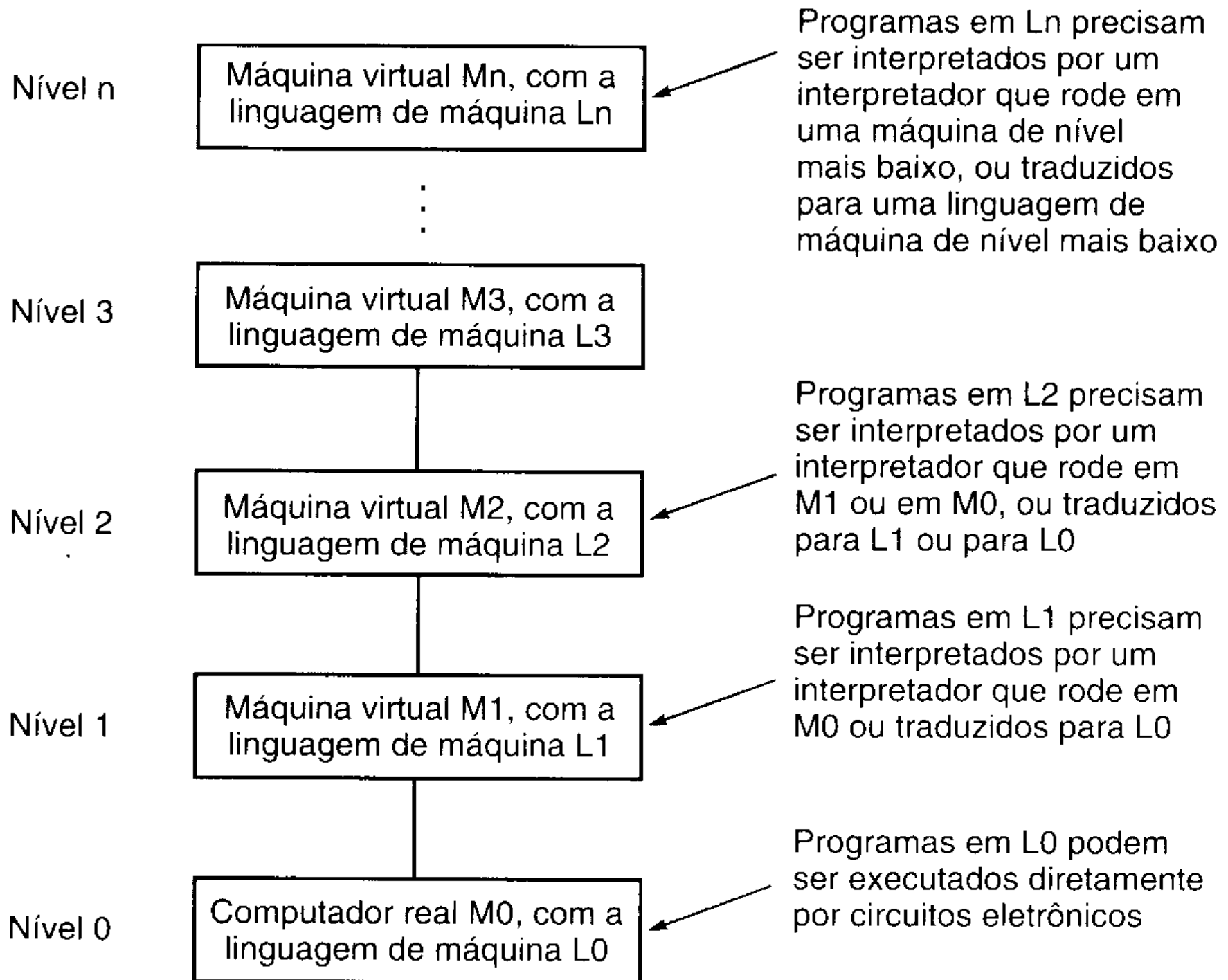


Níveis de Abstração

- Considere a máquina M_1 , ou seja, a imediatamente acima de M_0
- Para tornar a **tradução** ou **interpretação** uma tarefa prática, **as linguagens L_1 e L_0 não podem ser “muito” diferentes**
- Assim, apesar de L_1 ser melhor que L_0 , ainda está longe do ideal para a maioria das aplicações
- **Uma solução óbvia:** inventar um outro conjunto de instruções mais próximo das pessoas: L_2

Níveis de Abstração

- De maneira análoga ao raciocínio anterior, as pessoas poderiam escrever programas em L_2 como se a máquina M_2 realmente existisse
- A criação de uma série de linguagens, cada uma mais conveniente para os humanos do que suas antecessoras, pode seguir até uma que sirva aos nossos propósitos
- Assim, um **computador** que use essa técnica pode ser visto como um **conjunto de camadas ou níveis**



Níveis de Abstração

- *Questão:* Uma pessoa que escreve programas para a máquina virtual no nível ***n*** precisa saber se seu programa será executado por um conjunto de **interpretadores/tradutores**, ou se serão executados **diretamente pelos circuitos eletrônicos** da máquina?

Níveis de Abstração

- *Questão:* Uma pessoa que escreve programas para a máquina virtual no nível ***n*** precisa saber se seu programa será executado por um conjunto de **interpretadores/tradutores**, ou se serão executados **diretamente pelos circuitos eletrônicos** da máquina?
- *Não! O mesmo resultado é obtido em ambos os casos: seus programas são executados.*
- Porém! Pessoas interessadas em **projetar novos computadores** ou **níveis** para computadores existentes, precisam conhecer os demais níveis!

Evolução das máquinas com vários níveis

- Primeiros computadores:
 - Fronteira entre o hardware e o software era muito clara.
- Atualmente:
 - Muito difícil separar o hardware do software.

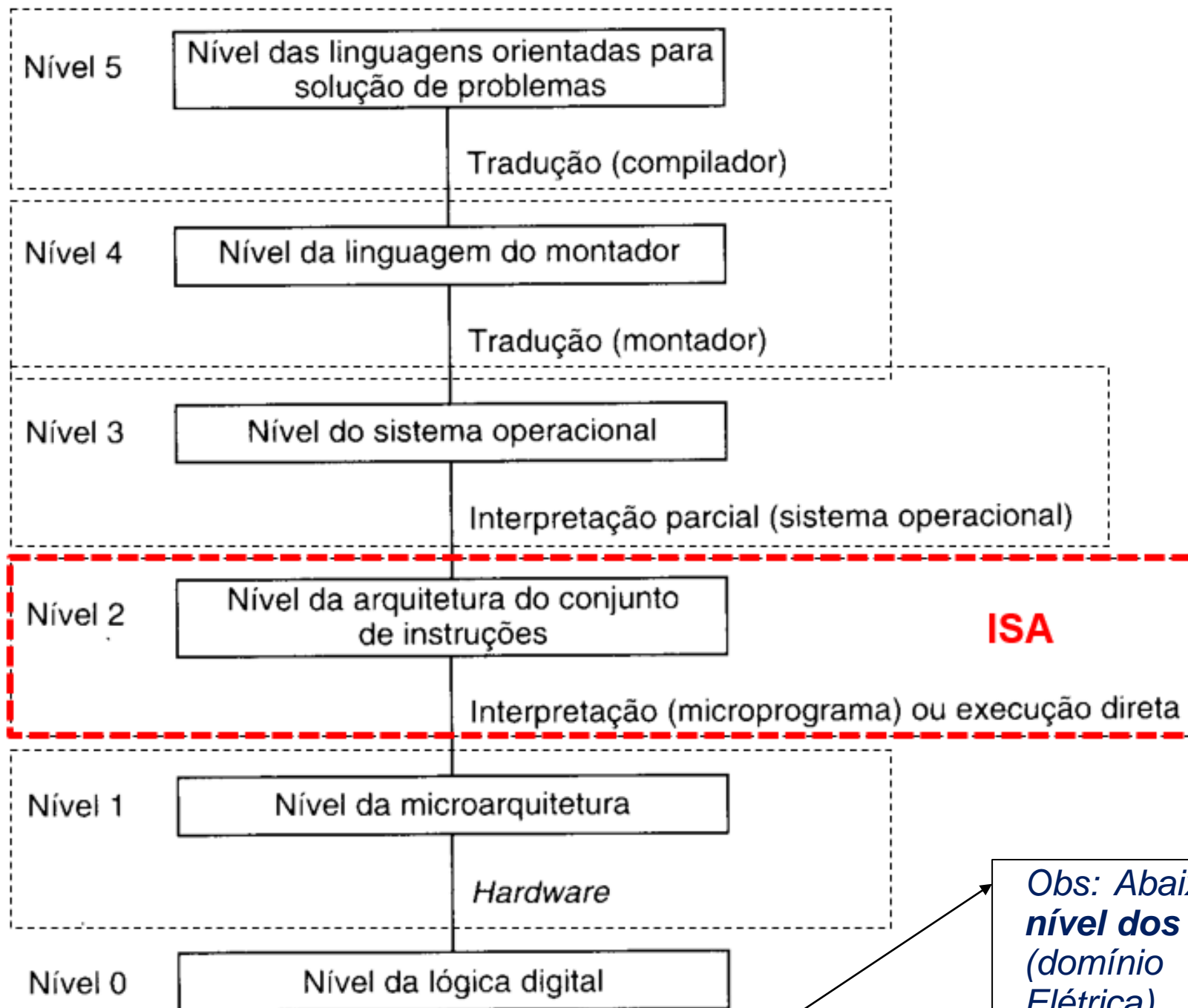
Evolução das máquinas com vários níveis

*Hardware e software são
equivalentes logicamente.*

Qualquer operação realizada por software pode ser realizada diretamente por hardware.

Qualquer instrução executada por hardware pode ser simulada em software.

Máquinas de vários
níveis modernas



Obs: Abaixo do nível 0:
nível dos dispositivos!
(domínio da eng.
Elétrica)

Nível 0

a. Nível 0: nível da Lógica Digital

- Nível mais baixo da estrutura
- Objetos de interesse são conhecidos como **portas lógicas**
- Cada porta lógica possui uma ou mais entradas digitais (sinais 0 ou 1) e calculam funções lógicas simples sobre essas entradas. Ex.: AND, OR, XOR
- Cada porta lógica é construída a partir de vários transistores



Nível 0

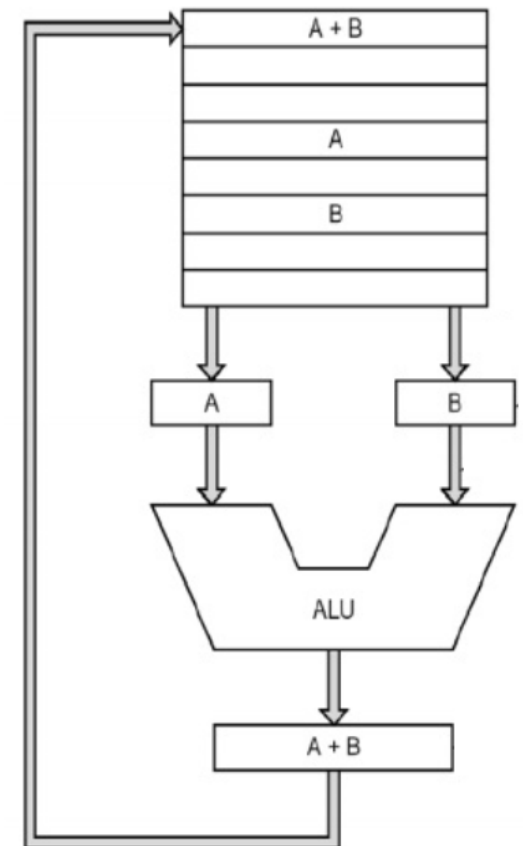
- (cont.)
- Portas lógicas são combinadas para formar, por exemplo:
 - 1 bit de memória (que combinados, podem formar **registradores** de 16, 32 ou 64 bits, por exemplo)
 - Assim, portas lógicas combinadas podem formar o principal dispositivo de computação: o **processador**

Nível 1

b. Nível 1: nível da Microarquitetura

- **Registradores:** memória local dentro do processador
- **ULA:** *Unidade Lógica Aritmética*, que realiza operações lógicas e aritméticas simples;

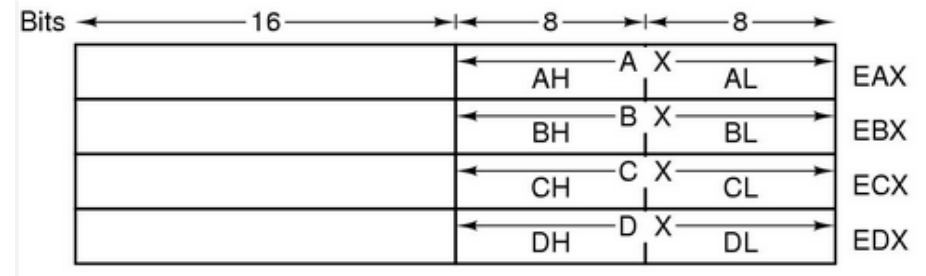
**Registradores + ULA =
caminho de dados** (estrutura
por onde os dados fluem)



Nível 1

- (cont.)
- **Operação básica do caminho de dados:** selecionar registradores para que a ULA opere sobre eles
- Em algumas máquinas:
 - Isso é controlado por **microprogramas** (**interpretador** para as instruções do **nível 2**): busca, decodifica e executa as instruções, uma a uma usando o caminho de dados para a realização desta tarefa
- Em outros tipos de máquinas:
 - O caminho de dados é controlado **diretamente pelo hardware**

Nível 2



c. Nível 2: nível da Arquitetura do Conjunto de Instruções (Nível ISA – *Instruction Set Architecture*)

- Definida pelo fabricante: disponibiliza “*Manual de Referência da Linguagem de Máquina*”
- Descrevem como as instruções devem ser executadas...
 - ... **interpretativamente** pelo **microprograma**,
ou
 - ... como são executadas **diretamente pelo hardware**
- Não trata nada de níveis mais baixos

Nível 3



d. Nível 3: nível do Sistema Operacional

- Geralmente um **nível híbrido**:
 - Com instruções próprias deste nível: interpretadas pelo **S.O.**
 - Mas também pode conter instruções do nível [ISA](#): **interpretadas** diretamente pelo **microprograma** ou pelo **hardware**
- Além disso, também suporta:
 - uma organização diferente da memória
 - a execução de dois ou mais programas simultaneamente

Nível 4

e. Nível 4: nível da linguagem do montador ou de montagem (*Assembly language*)

- **Linguagem de montagem:** forma simbólica de representação das linguagens dos níveis mais baixos
- São, inicialmente, **traduzidos** para as linguagens dos níveis 1, 2 e 3 e depois **interpretados** pela **máquina virtual** apropriada ou pela própria **máquina real**

- Programa que realiza a tradução:
montador

```
mov ax,cs
mov ds,ax
mov ah,9
mov dx, offset Hello
int 21h
xor ax,ax
int 21h
```

Nível 5

f. Nível 5: nível das linguagens orientadas para solução dos problemas

- Conhecidas como **linguagens de alto nível**
 - Ex: Basic, C, C++, Pascal, Java, LISP, Prolog...
- Programas dos níveis 4 e 5 são normalmente **traduzidos** por **compiladores** (em alguns casos, **interpretados**)

```
1  #include <stdio.h>
2  /* Um Primeiro Programa */
3
4  int main()
5  {
6      printf("Hello, World!\n");
7      return(0);
8  }
```

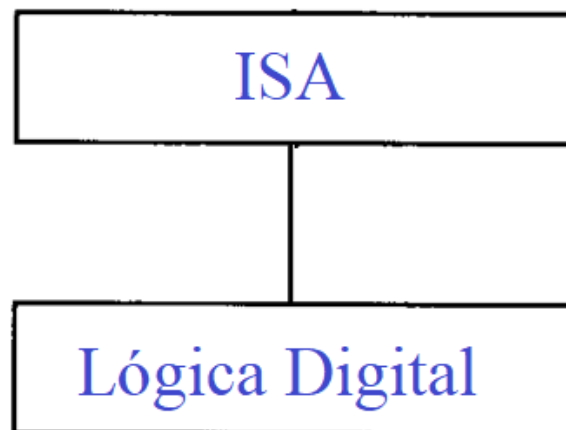
Comparações

	Linguagem	Programadores
Nível 1	Linguagens numéricas, desconfortáveis para o ser humano	Programadores de sistemas
Nível 2		
Nível 3		
Nível 4	Linguagens contém palavras e abreviações, convenientes para o ser humano	Programadores de aplicação
Nível 5		

A invenção da Microprogramação

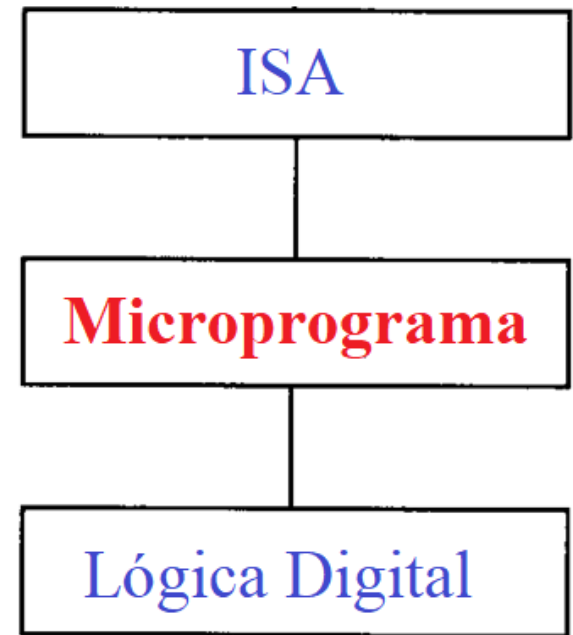
A invenção da **microprogramação**

- Os primeiros computadores digitais (~1940) tinham **somente 2 níveis**:
 - **Nível ISA**, onde toda a programação era feita
 - **Nível da lógica digital**, onde os programas eram executados (circuitos complicados e pouco confiáveis)



- Em **1951** (Maurice Wilkes): ideia de construir um computador com 3 níveis para simplificar o hardware:

- **Necessário um interpretador (microprograma)** para executar programas no **nível ISA**



- Portanto: **hardware** passa a executar somente **microprogramas** (*que possuem conjunto de instruções bastante limitado*), **ao invés de programas no nível ISA** (*com muito mais instruções!*)
- **Necessário bem menos circuitos eletrônicos!**

menos válvulas = maior confiabilidade

A invenção da **microprogramação**

- Algumas dessas **máquinas de 3 níveis** foram construídas ao longo dos **anos 1950**
- **Década de 1960**: quantidade de máquinas produzidas dessa forma aumentou bastante
- **Década de 1970**: tornou-se prática comum ter um **nível ISA** interpretado por um **microprograma**, em vez de ser executado diretamente por circuitos eletrônicos

A invenção da **microprogramação**

- Projetistas logo observaram que poderiam acrescentar novas instruções ao **conjunto de instruções** do processador simplesmente **expandindo o microprograma!**
- **Explosão no conjunto de instruções!**
- Projetistas disputando na produção de conjuntos maiores e melhores!



A invenção da **microprogramação**

- Porém, a maioria dessas novas instruções não eram essenciais
- Seu efeito poderia ser muito bem alcançado por instruções já existentes, por exemplo:
 - **ADD**: soma genérica (instrução já existente)
 - **INC** (INCremento): adicionava 1 unidade a um número (instrução nova, só um pouco mais rápida que ADD)



A invenção da **microprogramação**

- Foram adicionadas **várias outras instruções** ao **conjunto de instruções** por meio do **microprograma**:
 - para multiplicação e divisão de números inteiros
 - para aritmética em ponto flutuante
 - para chamada e retorno de procedimentos
 - para acelerar a execução de loops
 - para manipular *strings* de caracteres

0011111	01011	01010	001	01000	1100111
imm[12:6]	rs2	rs1	funct3	imm[5:1]	opcode

A invenção da **microprogramação**

- Os projetistas também logo perceberam que era muito fácil incorporar também novas **funcionalidades** às máquinas:
 - Acelerar o processamento de programas
 - Permitir que os programas pudessem ser deslocados de posição na memória após o início da execução
 - Sistemas de interrupção que avisam o processador assim que uma operação de entrada ou saída termina
 - Suspender a execução de um programa e iniciar outro, usando um número muito pequeno de instruções (troca de contextos entre processos)

A invenção da **microprogramação**

- Consequências da “**era de ouro da microprogramação**” entre décadas de 60 e 70?

• Os **microprogramas** cresceram muito e tornaram-se lentos!

- Pesquisadores começaram a estudar os efeitos de projetar máquinas SEM usar microprogramação!

A eliminação da **microprogramação**

eliminando os **microprogramas**

+

reduzindo o **conjunto de instruções**

+

as **instruções remanescentes** sendo executadas diretamente (caminho de dados controlado pelo hardware)

=

Máquinas poderiam ter um ganho significativo no tempo de execução de instruções

A invenção do Sistema Operacional

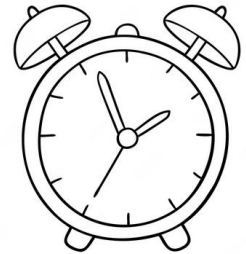
A invenção do Sistema Operacional

- Nos primórdios da computação eletrônica, a maioria dos computadores era operada pelo próprio **programador**
- O programador deveria ir até a sala específica onde eram “rodados” os programas
- Obs: a máquina ficava ociosa enquanto as pessoas carregavam cartões de um lado para outro ou tentando descobrir um erro



A invenção do Sistema Operacional

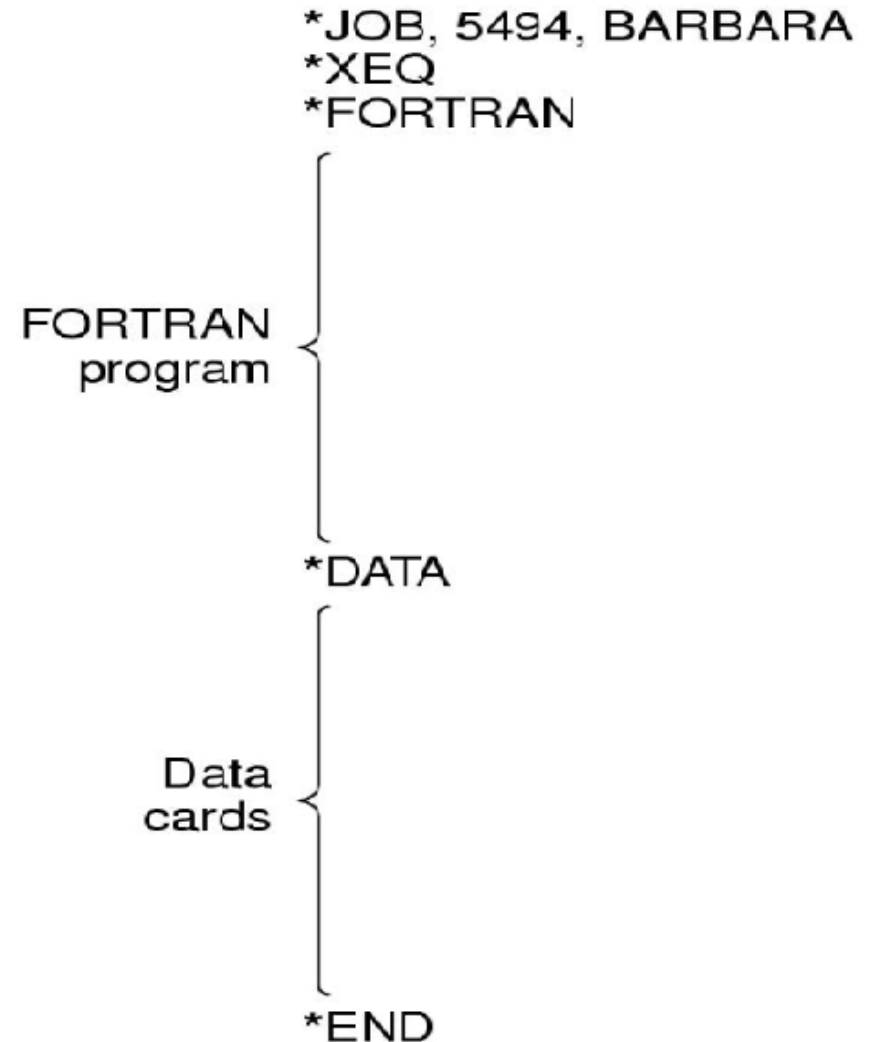
- **1960:** “*Como reduzir o tempo gasto na execução de programas?*”



- Automatizando os trabalhos de operação da máquina
- Um programa, chamado **sistema operacional**, foi projetado para ser mantido na memória durante todo o tempo que ele estivesse sendo executado
- O programador devia preparar **cartões de controle**, que faziam a leitura e execução de seus programas, sob controle do **sistema operacional**

Ex: Cartões de controle (com *)

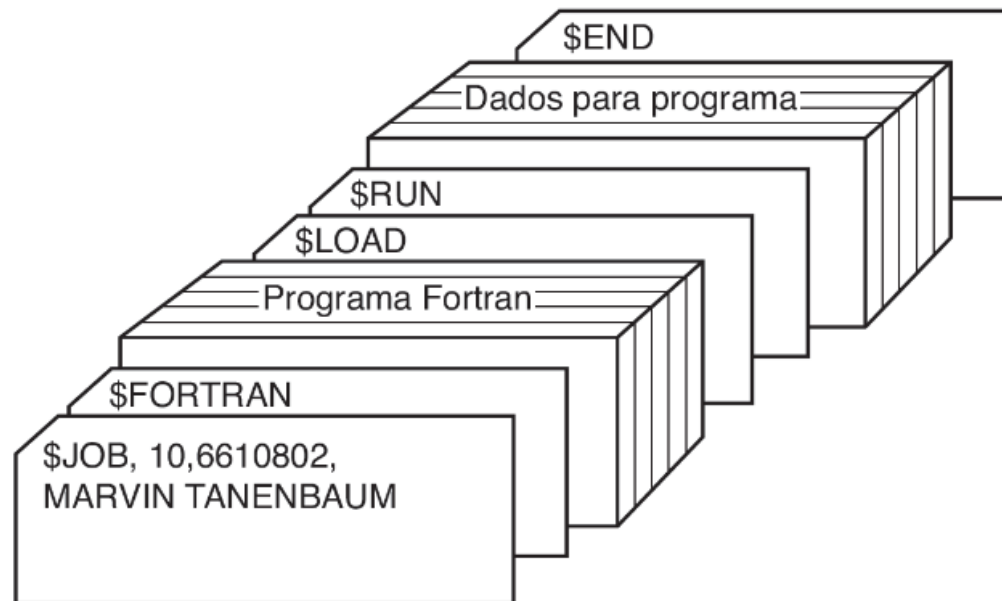
- ***JOB**: identifica o usuário,
- ***FORTRAN**: instrução para carregar o compilador Fortran da fita magnética para a memória
- Compilador lia e compilava o programa em FORTRAN
- ***DATA**: instrução para que o programa traduzido fosse executado, usando os cartões que o seguem como dados



FMS (*FORTRAN Monitor System*), um dos primeiros e mais utilizados S.O.'s

A invenção do Sistema Operacional

- Apesar de ter sido projetado para automatizar o trabalho do operador (daí seu nome), o **sistema operacional** foi também o primeiro passo para o desenvolvimento de uma **nova máquina**, seguindo o conceito de **máquina virtual**



Conclusões

Conclusões

- Computadores são projetados como uma série de níveis
- Cada nível representa uma abstração distinta
- A abstração permite ignorar, abstrair temporariamente detalhes irrelevantes, de níveis mais baixos, reduzindo questões complexas a algo fácil de ser entendido

Conclusões

- O conjunto de **tipos de dados**, **operações** e **características** de cada um dos níveis é conhecido como **arquitetura** do nível.
- A **arquitetura** trata dos **aspectos visíveis** aos programadores de um determinado **nível**, (como por exemplo, disponibilidade de memória)

Obs: *Detalhes de implementação*, como o tipo de chip usado para implementar a memória, **NÃO** são parte da *arquitetura*!

Conclusões

- **Fronteira** entre o hardware e o software pode ser arbitrária e está sempre mudando.
- O que hoje está implementado em **software** poderá em breve ser implementado em **hardware**, e vice-versa.
- A fronteira entre os demais níveis também não estão muito bem definidas e mudam constantemente

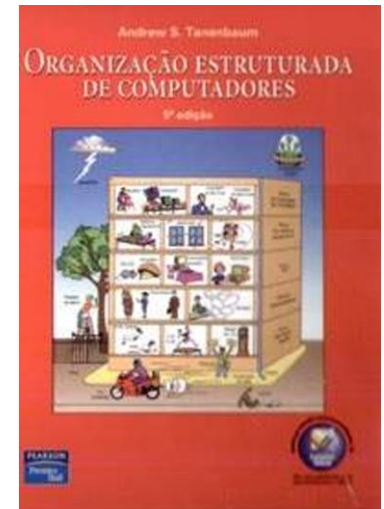
Conclusões

- Para um programador no nível ISA, por exemplo, não tem muita importância a maneira como uma instrução é realmente implementada (em hardware ou em microprogramação?)
 - Obs: exceto se a implementação influir na velocidade de execução da instrução
- Programador de um nível, em geral, não deve se preocupar com implementações de níveis inferiores.



Leitura Indicada

- Capitulo 1 (Patterson, 2018): página 40 a 66
- Capitulo 1 (Tanenbaum): seção 1.1



- Leitura Complementar (Patterson, 2018):
página 67 a 91

Referência Bibliográfica

Patterson, David A. & Hennesy, John L.
**“Computer Organization & Design Interface
RISC V edition”**., Morgan Kaufmann Publishers
Inc., San Francisco, CA, 2018;

Tanenbaum, A S **“Organização Estruturada de
Computadores”** – Prentice Hall do Brasil 5ª
edição, 2006.