Laboratório de Pesquisa em Redes e Multimídia

Introdução à Engenharia de Computação

Universidade Federal do Espírito Santo
Departamento de Informática

Tópico: O Computador como uma Máquina Multinível (cont.)

José Gonçalves - LPRM/DI/UFES
Introdução à Engenharia de Computação

Máquina Multinível Moderna

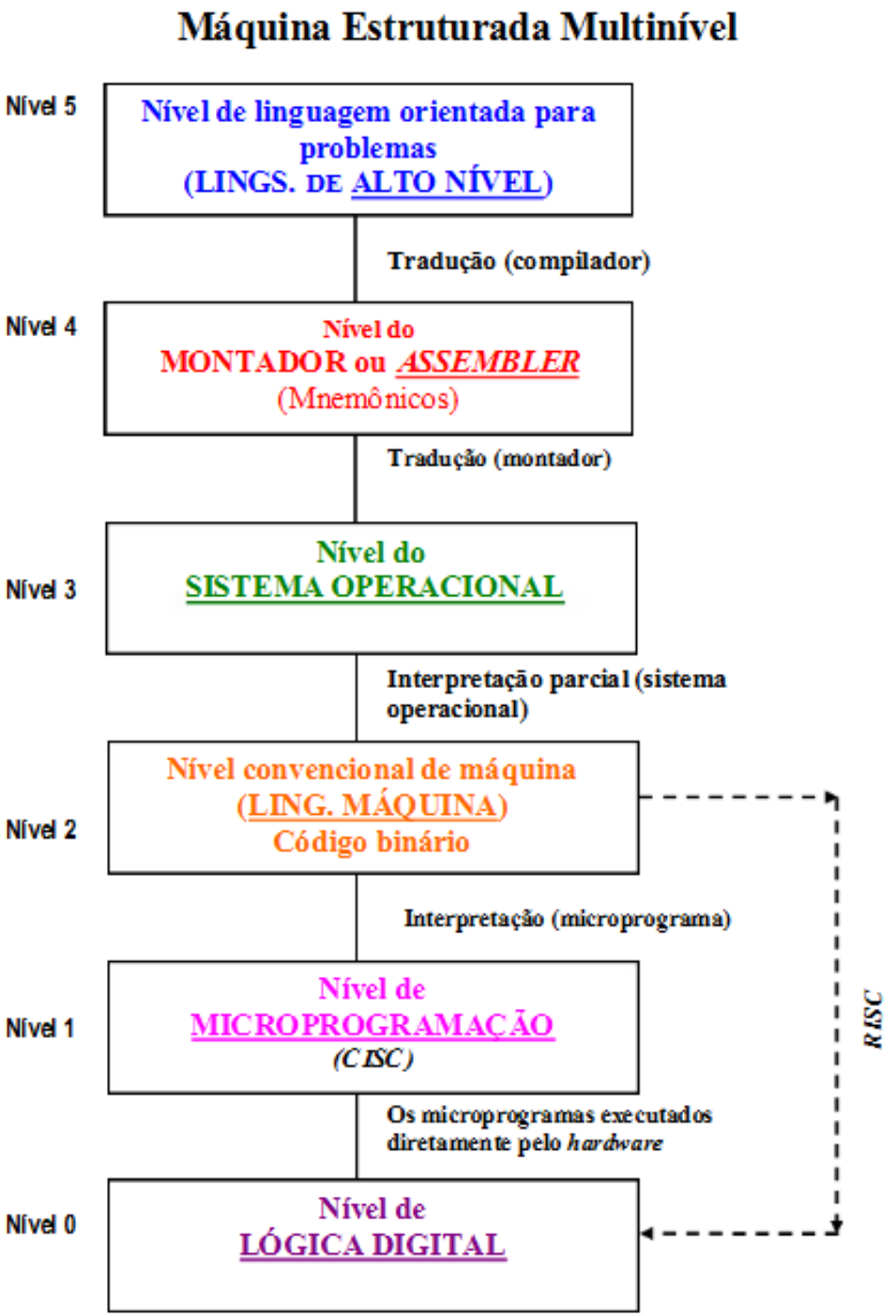
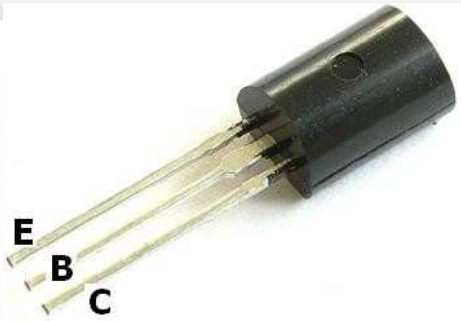
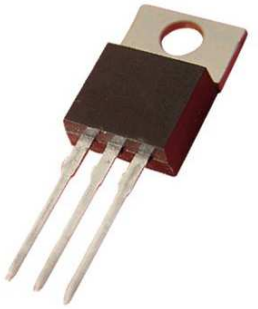


Figura 1 – Máquina de seis níveis. (adaptado de Tanenbaum 1995)

O Nível dos Dispositivos





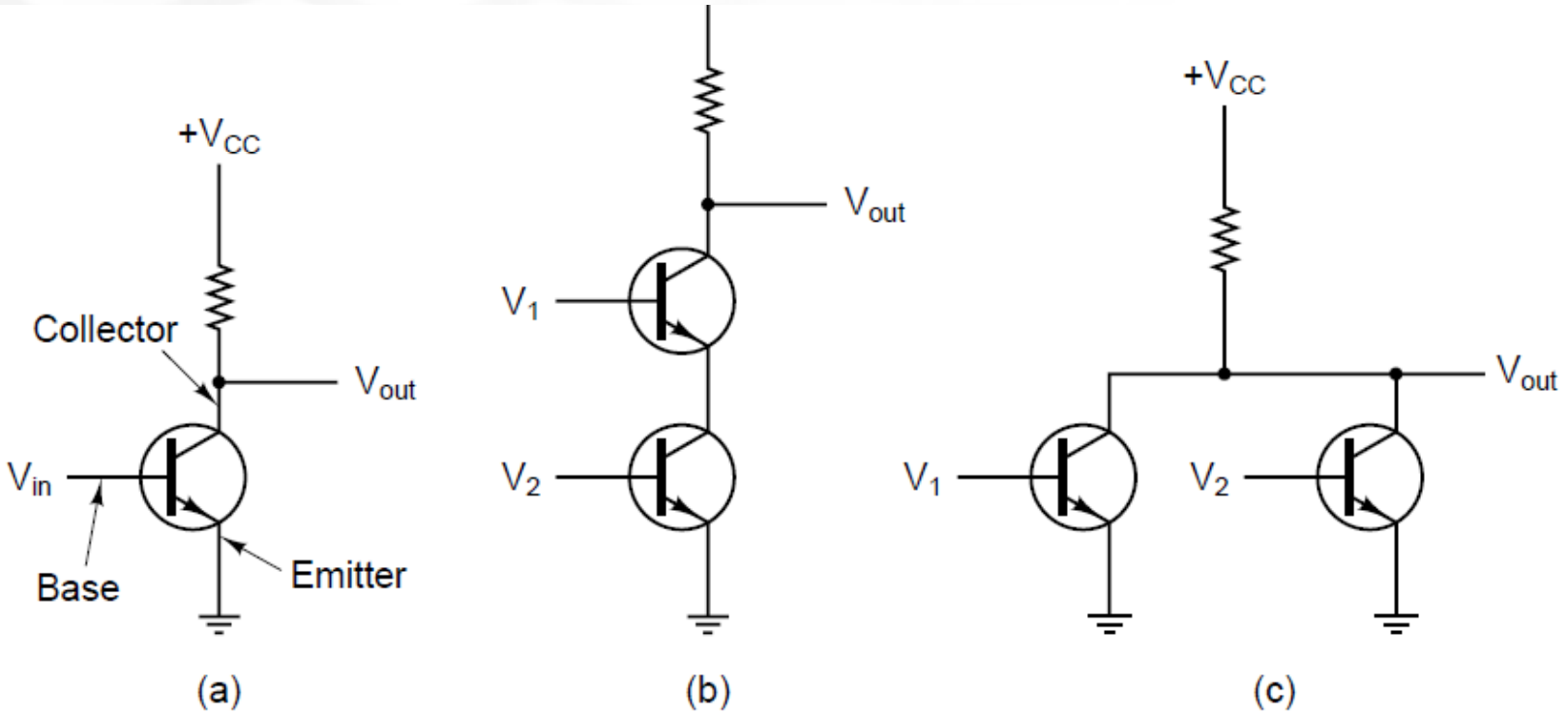
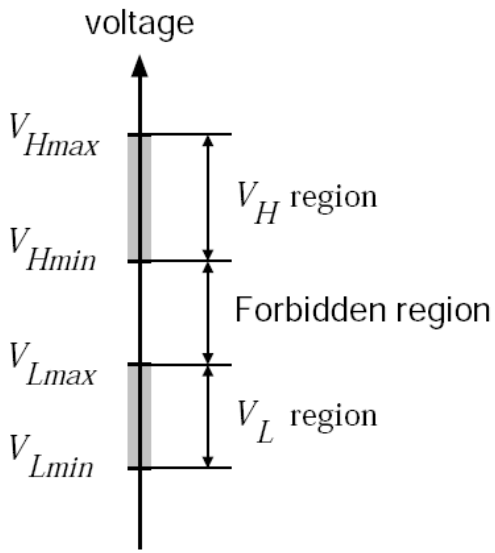
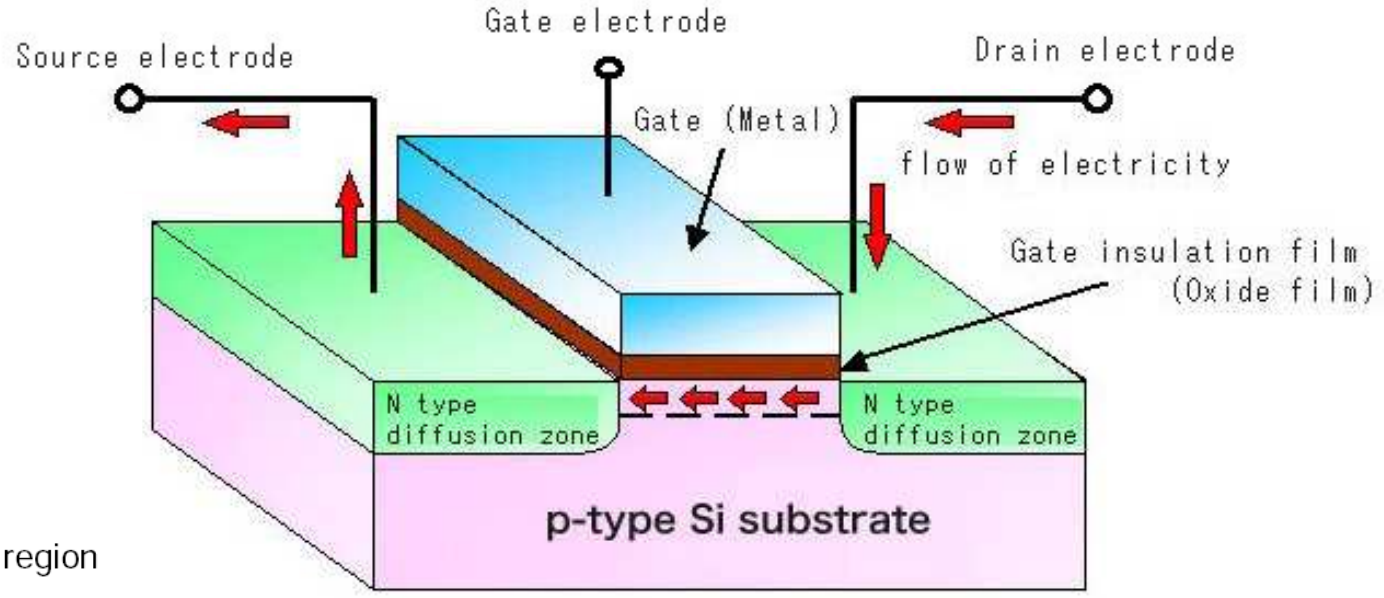


Figure 3-1. (a) A transistor inverter. (b) A NAND gate. (c) A NOR gate.

O Nível dos Dispositivos





Construction of MOSFET

O Nível dos Dispositivos (cont.)

- Situado abaixo do nível 0, não é representado na arquitetura multinível.
- Seus elementos pertencem ao campo da microeletrônica, engenharia elétrica e da física do estado sólido.
  - Transistores individuais, cristal semicondutor (substrato dos circuitos integrados), malha de transistores
- Tecnologias de fabricação de circuitos integrados
  - Bipolar
    - TTL (Transistor-Transistor Logic) e ECL (Emitter-Coupled Logic)
  - MOS - Metal Oxide Semiconductor
    - Portas mais lentas, gastam menos energia e ocupam espaço bem menor.
    - CMOS - transistores mosfet: baixíssimo consumo, altíssima densidade de integração.

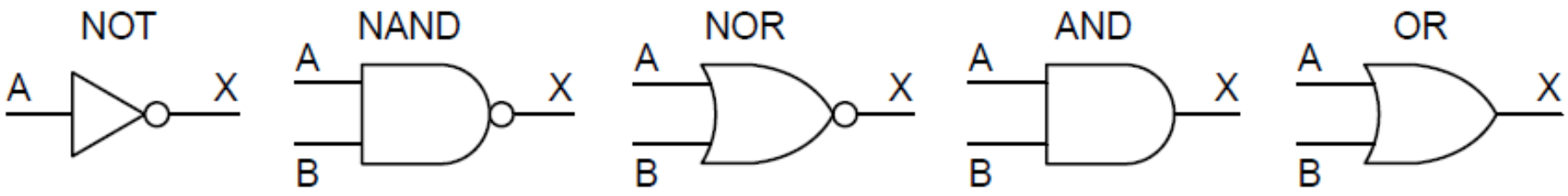
Nível 0 (Nível de Lógica Digital) (cont.)

- Composto por circuitos lógicos digitais. Nesse nível, os objetos de interesse são as **portas lógicas**.
- As portas lógicas são dispositivos digitais construídas a partir de componentes analógicos (ex: transistores). Elas formam os elementos primários dos circuitos lógicos do computador.
  - Portas básicas: NOT, AND, OR, NAND, NOR, XOR
- As portas são usadas para criar desde componentes digitais simples ou complexos, como somadores, deslocadores, decodificadores, memórias (registradores), ULA's e circuitos integrados.
  - Um único tipo de porta, NAND ou NOR, é suficiente para construir qualquer circuito lógico.

Nível 0 (Nível de Lógica Digital) (cont.)

- Os circuitos lógicos são construídos de acordo com a Álgebra de Boole, a partir da combinação das portas lógicas básicas e suas combinações, e integrados em pastilhas (chips) de silício.
- De acordo com a densidade de componentes, os chips são classificados em:
  - Integração em Pequena Escala (SSI): de 1 a 10 portas lógicas.
  - Integração em Média Escala (MSI): de 10 a 100 portas lógicas.
  - Integração em Grande Escala (LSI): de 100 a 100.000 portas lógicas.
  - Integração em Escala Muito Grande (VLSI): acima de 100.000 portas lógicas

Nível 0 (Nível de Lógica Digital)



A	X
0	1
1	0

(a)

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b)

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

(c)

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

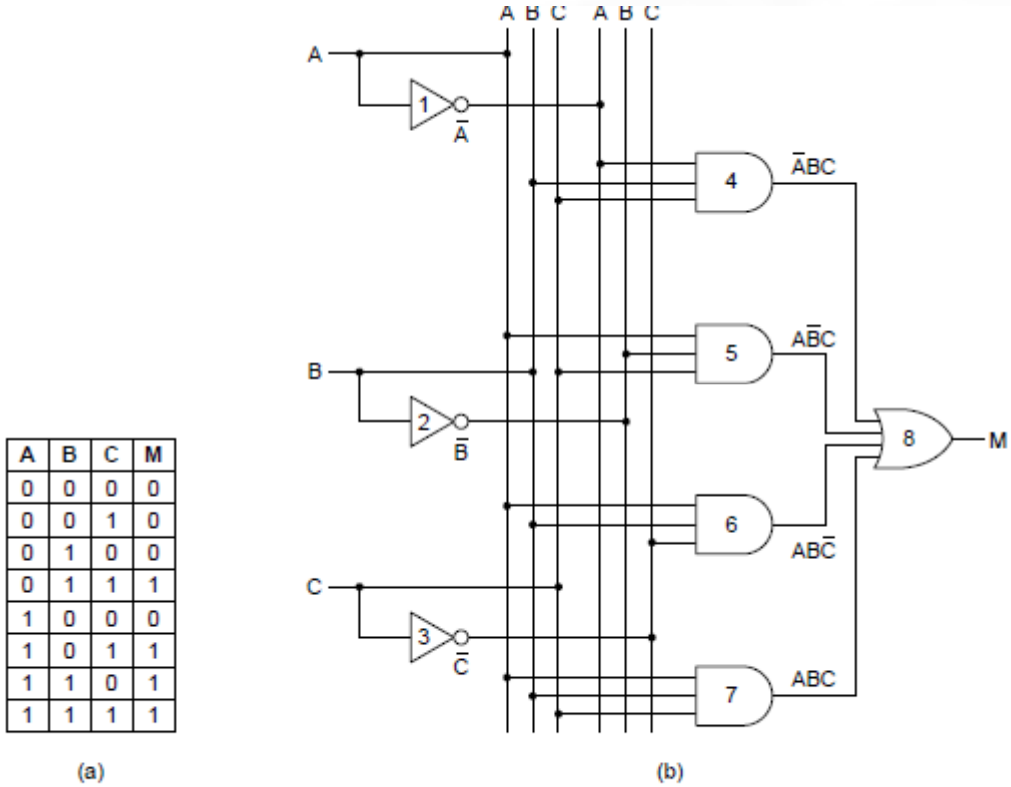
(d)

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

(e)

Figure 3-2. The symbols and functional behavior for the five basic gates.

Nível 0 (Nível de Lógica Digital) (cont.)



A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(a) (b)

Figure 3-3. (a) The truth table for the majority function of three variables. (b) A circuit for (a).

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

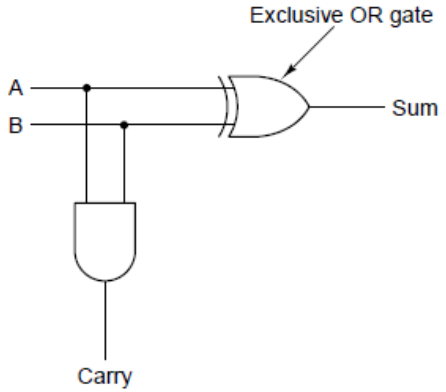


Figure 3-17. (a) Truth table for 1-bit addition. (b) A circuit for a half adder.

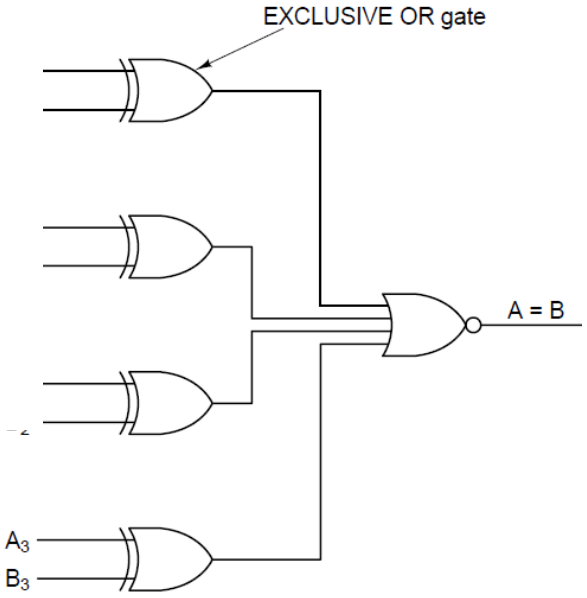
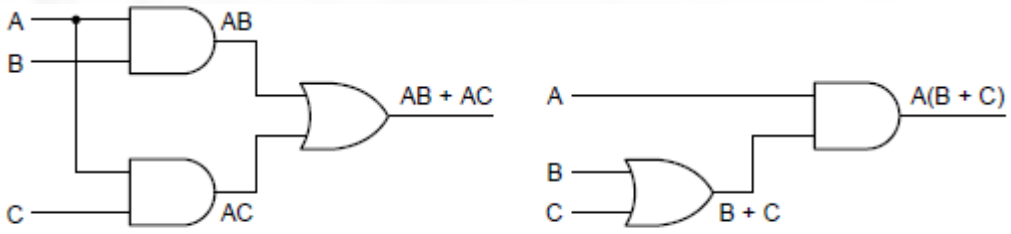


Figure 3-14. A simple 4-bit comparator.

- Alguns circuitos lógicos básicos

Nível 0 (Nível de Lógica Digital) (cont.)



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

(a)

A	B	C	A	B + C	A(B + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

(b)

Figure 3-5. Two equivalent functions. (a) $AB + AC$. (b) $A(B + C)$.

- Equivalência de circuitos lógicos

Nível 0 (Nível de Lógica Digital) (cont.)

- O nível 0 é responsável pela interpretação de instruções do nível superior, ou seja, seus circuitos executam os programas em linguagem de máquina do nível 1:
  - Macroinstruções (nível ISA – Instruction Set Architecture); ou
  - Microinstuções (nível de microprogramação).
- A arquitetura da máquina pode ser entendida como um conjunto de registradores, associados a outros módulos lógicos adicionais para processamento de informação, interconectados de maneira apropriada.
- O seqüenciamento adequado dos sinais de controle destes módulos e registradores produz o fluxo ordenado de informação entre os mesmos, necessário à interpretação de instruções do nível superior.
- Equivalência entre hardware e software
  - funções desempenhadas pelo software podem migrar para o hardware e vice-versa.

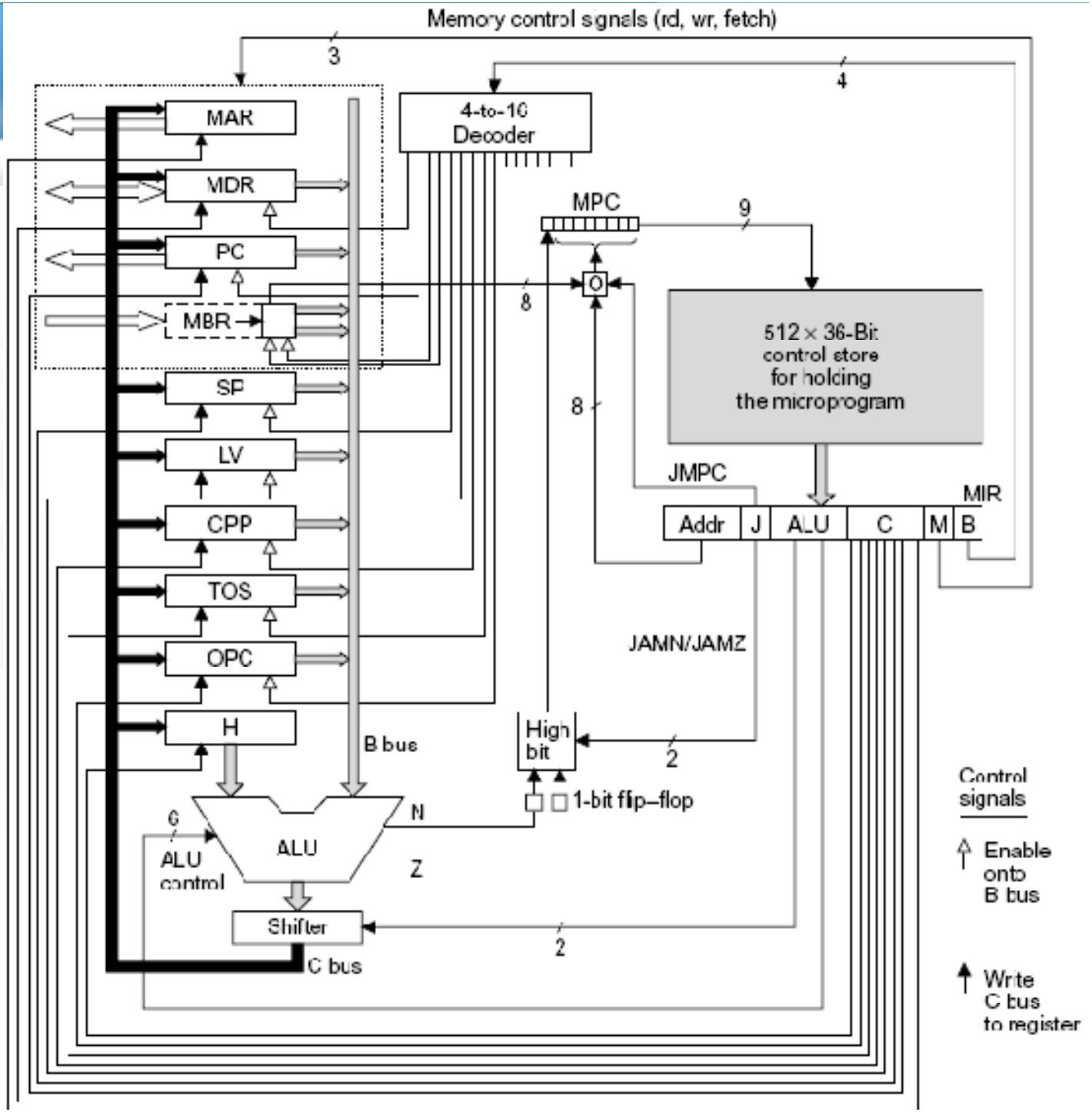
Nível 1 (Nível de Microarquitetura)

- É nesse nível que se inicia o conceito de programa como uma sequência de instruções a serem executadas diretamente pelos circuitos eletrônicos.
- Enxerga-se nesse nível um conjunto de 8 a 32 **registradores** (PC, MAR, MDR, SP, etc.) que formam uma memória local e uma **ULA (Unidade Lógica e Aritmética)**, capaz de realizar operações aritméticas simples.
- Os registradores e a ULA são conectados para formar o **caminho de dados (*Data Path*)**, estrutura sobre a qual os dados fluem.
- As operações são controladas por um **microprograma** ou diretamente por hardware

Nível 1 (Nível da Microarquitetura) (cont.)

- Controle por software
  - Microprograma = Interpretador
  - O microprograma busca, decodifica e executa as instruções vindas do nível 2, uma a uma, usando o caminho de dados para a realização da tarefa.
  - Exemplo: execução da instrução ADD
    - A instrução é buscada na memória, seus operandos são localizados e trazidos para os registradores, a soma é realizada na ULA, e o resultado é encaminhado para o lugar apropriado
- Controle por hardware
  - Os mesmos passos acima são executados, mas sem que haja um programa armazenado para controlar a interpretação das instruções.

Nível 1 (Nível de Microarquitetura) (cont.)



- Microprograma exemplo:

- O microprograma é um interpretador de macroinstruções: implementa o ciclo de busca, decodificação e execução das macroinstruções.
- O ciclo de busca, decodificação, execução é executado em laço, que começa na linha 0 do microprograma. Ao final da execução de uma macroinstrução, sempre existe uma microinstrução de desvio incondicional para 0, de modo a repetir o laço.

- Busca da macroinstrução: (linhas 0, 1 e 2 do microprograma):
  - Na linha 0, o cp é carregado no REM para endereçar a memória principal. Simultaneamente, o sinal RD é ativado para requisitar leitura.
  - Na linha 1, o sinal RD é mantido ativo, para carregar a instrução endereçada no RDM. Simultaneamente, o CP é incrementado para apontar para a próxima instrução.
  - Na linha 2, a instrução armazenada no RDM é transferida para o RI, concluindo a busca.
- Decodificação da macroinstrução:
  - Realizada analisando os bits do código de operação um a um
  - Cada bit é analisado através de sucessivos deslocamentos à esquerda da macroinstrução, os quais são associados ao teste do MSB correspondente usando o bit N de *status* da ULA.
  - A cada deslocamento, a instrução deslocada é armazenada temporariamente no RT.
  - De acordo com os testes dos bits do código de operação, o fluxo do microprograma é desviado numa busca em árvore até chegar no código responsável pela execução da macroinstrução
  - A decodificação começa já na linha 2 do microprograma, paralelamente ao fim do processo de busca. O primeiro bit do código de operação é testado ao passar pela ULA, na sua transferência do RDM para o IR.
  - A decodificação em árvore é realizada nas linhas 2, 3, 4, 5, 14, 17, 20, 22, 28, 29, 30, 33, 41, 43 (ver figura).
- Execução da macroinstrução:
  - Após o processo de decodificação, o microprograma executa mais algumas microinstruções responsáveis pela execução da macroinstrução. Algumas macroinstruções semelhantes compartilham trechos de código, por questão de eficiência.
  - No final da execução, sempre se executa um desvio incondicional para a linha 0, de modo a recomeçar o ciclo e interpretar a próxima macroinstrução.

Microintrução	Comentário	Microinstrução	Comentário
		34: pp:=pp-1;	1010 CALL
0: rem:=cp; rd;	Laço principal	35: rem:=pp;	Endereça topo da pilha
1: cp:=cp+1; rd;	Incremente cp	36: rdm:=cp;wr;	Disponibiliza cp
2: ri:=rdm; if n then goto 28;	Salva rdm/0xxx ou 1xxx?	37: cp:=(ri∧masc); wr; goto 0;	Empilha cp e desvia
3: rt:=(<<ri); if n then goto 17;	00xx ou 01xx?		
4: rt:=(<<rt); if n then goto 12;	000x ou 001x?	38: rem:=pp; rd;	1011 RETN
5: rt:=(<<rt); if n then goto 9;	0000 ou 0001?	39: pp:=pp+1; rd;	Lê cp do topo da pilha
		40: cp:=rdm; goto 0	Retorna
6: rem:=(ri∧masc); rd;	0000 = LOAD		
7: rd;	Transfere dado	41: rt:=(<<rt); if n then goto 49;	110x ou 111x?
8: ac:= rdm; goto 0;	Carrega dado		
		42: pp:=pp-1;	Abre espaço na pilha
9: rem:=(ri∧masc);	0001 = STOR	43: rt:=(<<rt); if n then goto 47;	1100 ou 1101?
10: rdm:=ac; wr;	Disponibiliza dado		
11: wr; goto 0;	Transfere dado	44: rdm:=ac;	1100 PUSH
		45: rem:=pp; wr;	Endereça pilha / disponibiliza dado
12: rem:=ri;rd;	Endereça operando	46: wr; goto 0;	Empilha
13: rd;	Busca operando		
14: rt:=(<<rt); if n then goto 16;	0010 ou 0011?	47: rem:=(ac∧masc); rd;	1101 PSHI
		48: rd; goto 45	Lê operando indireto
15: ac:=ac+rdm; goto 0;	0010 ADDD		
		49: rem:=pp;	Endereça pilha
16: ac:=ac-rdm; goto 0;	0011 SUBD	50: pp:=pp+1; rd;	Atualiza ponteiro de pilha / sinaliza leitura
		51: rd;	Desempilha
17: rt:=(<<rt); if n then goto 22;	010x ou 011x?		
		52: rt:=(<<rt); if n then goto b0=1;	1110 ou 1111?
18: rem:=(ri∧masc);	Mascara constante		
19: rem:=rem+pp; rd.	Calcula endereço	53: ac:=rdm; goto 0;	1110 POP
20: rt:=(<<rt); if n then goto 16; rd;	Busca operando/0100 ou 0101 (SUBL)?		
		54: rem:=(ac∧masc); wr; goto 11	1111 POPI
21: goto 15;	0100 ADDL		
22: rt:=(<<rt); if n then goto 26;	0110 ou 0111?		
23: ac:=ac; if z then goto 25;	0110 JZER		
24: goto 0;	Desvio não realizado		
25: cp:=(ri∧masc); goto 0;	Desvia		
26: ac:=ac; if n then goto 25;	0111 JNEG		
27: goto 0;	Desvio não realizado		
28: rt:=(<<ri); if n then goto 41;	10xx ou 11xx?		
29: rt:=(<<rt); if n then goto 33;	100x ou 101x?		
30: rt:=(<<rt); if n then goto 32;	1000 ou 1001?		
31: cp:=(ri∧masc); goto 0;	1000 JUMP		
32: ac:=(ri∧masc); goto 0;	1001 LOCO		
33: rt:=(<<rt); if n then goto 38;	1010 ou 1011?		

Figura 5.10. Exemplo de Microprograma.

- Exemplo de um microprograma

Nível 1 (Nível de Microarquitetura (cont.)

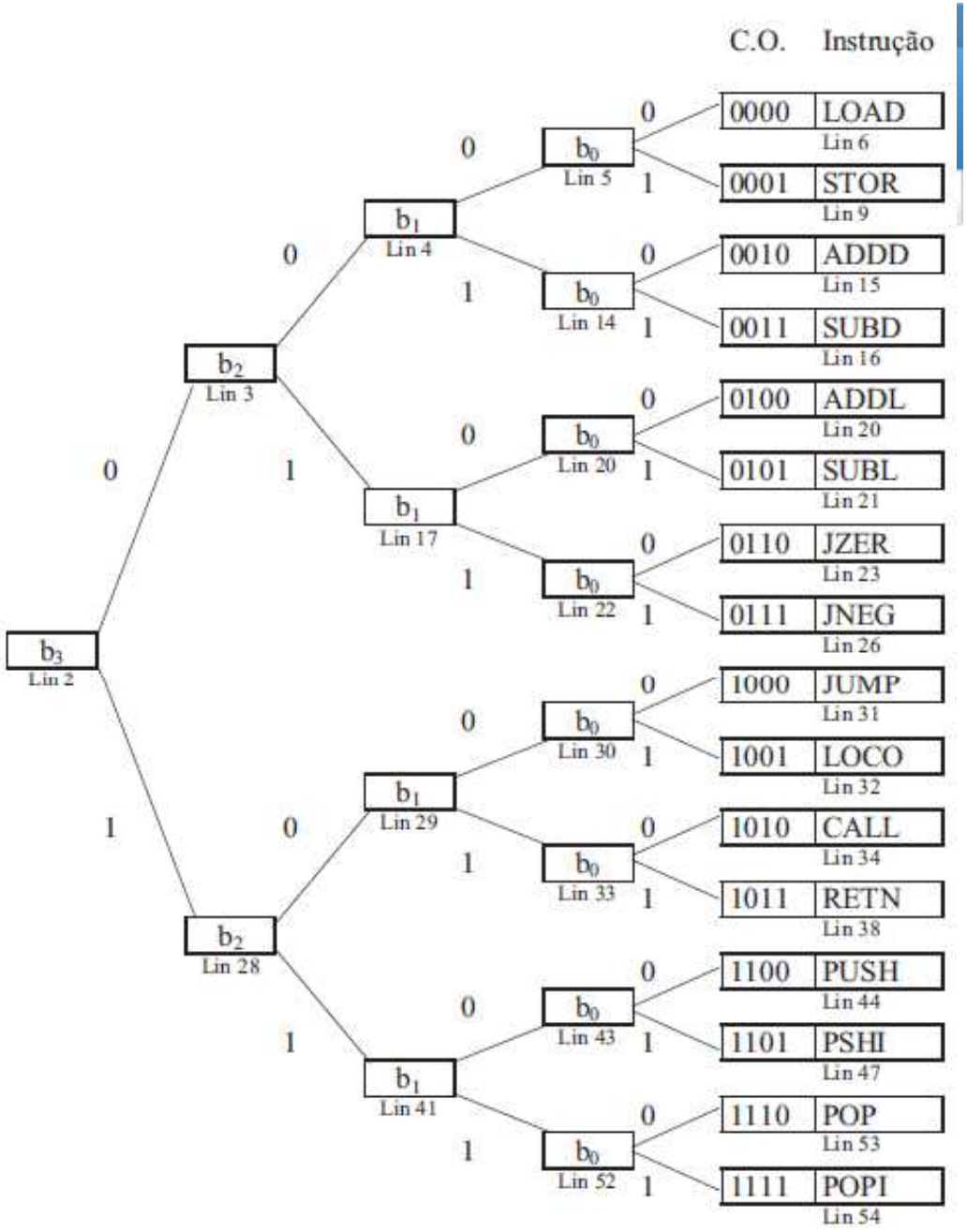


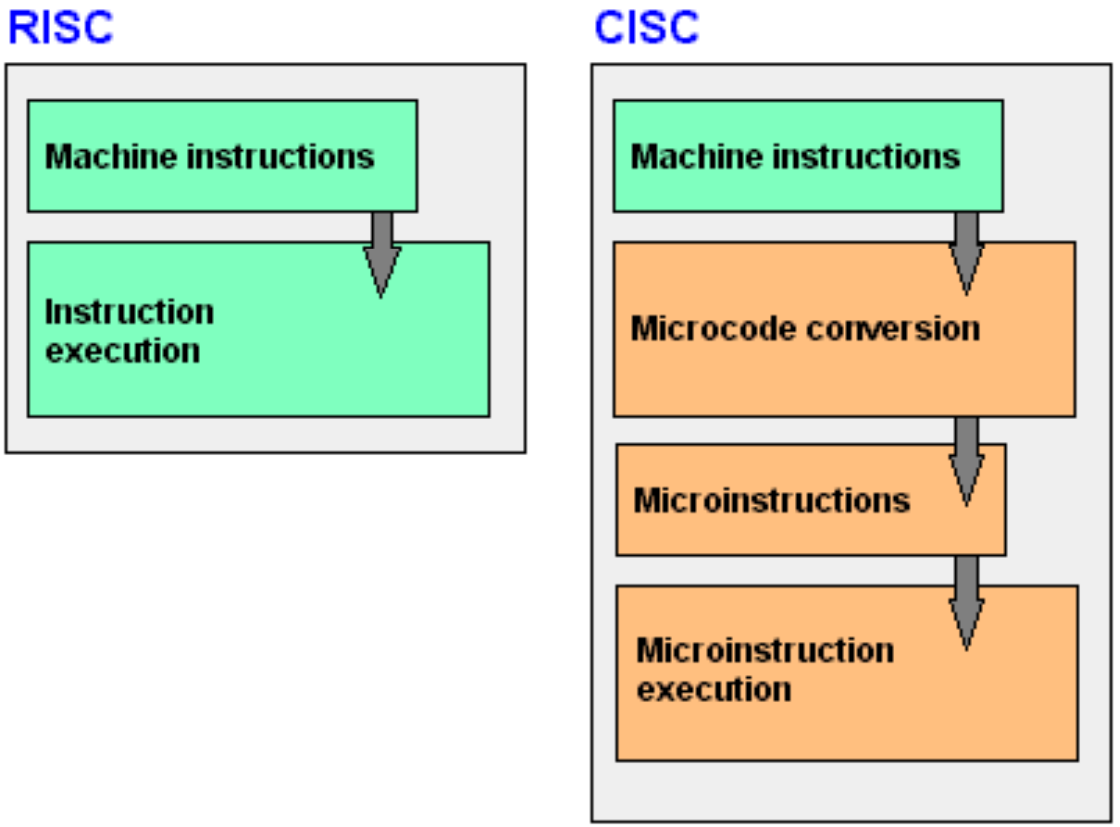
Figura 5.11. Busca em arvore para decodificação do Código de Operação da Instrução (C.O. = $b_3b_2b_1b_0$).

Nível 1 (Nível da Microarquitetura) (cont.)

- Vantagens da microprogramação
  - Facilita o projeto e a construção dos circuitos digitais (implementação de parte da lógica digital dentro do firmware).
  - Possibilita o desenvolvimento de instruções mais potentes no nível de máquina convencional.
  - Permite alterar o ISA escrevendo outro microprograma. Facilita a concepção de famílias de microprocessadores.

RISC x CISC

From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.



Nível 2 (Nível ISA - Instruction Set Architecture)

- Denominado de "Nível Convencional", define a interface entre o hardware e o software.
- Determina o conjunto de instruções de máquina executáveis pelo processador ("**linguagem de máquina**").
  - O nível 2 corresponde à linguagem de máquina ou ao código binário interpretado e executado pelo hardware.
- Cada processador tem a sua própria linguagem de máquina, documentada em manuais específicos de cada fabricante.
- Além dos formatos e tipos de instrução, define-se tipos de dados, modelos de memória, modos de endereçamento, etc.
- O nível 2 é interpretado para o nível de microprogramação (máquinas CISC) ou é executado diretamente pelo hardware (máquinas RISC). Manuais descrevem como as instruções são executadas interpretativamente pelo microprograma ou como elas são executadas diretamente pelo hardware.

Nível 2 (Nível ISA) (cont.)

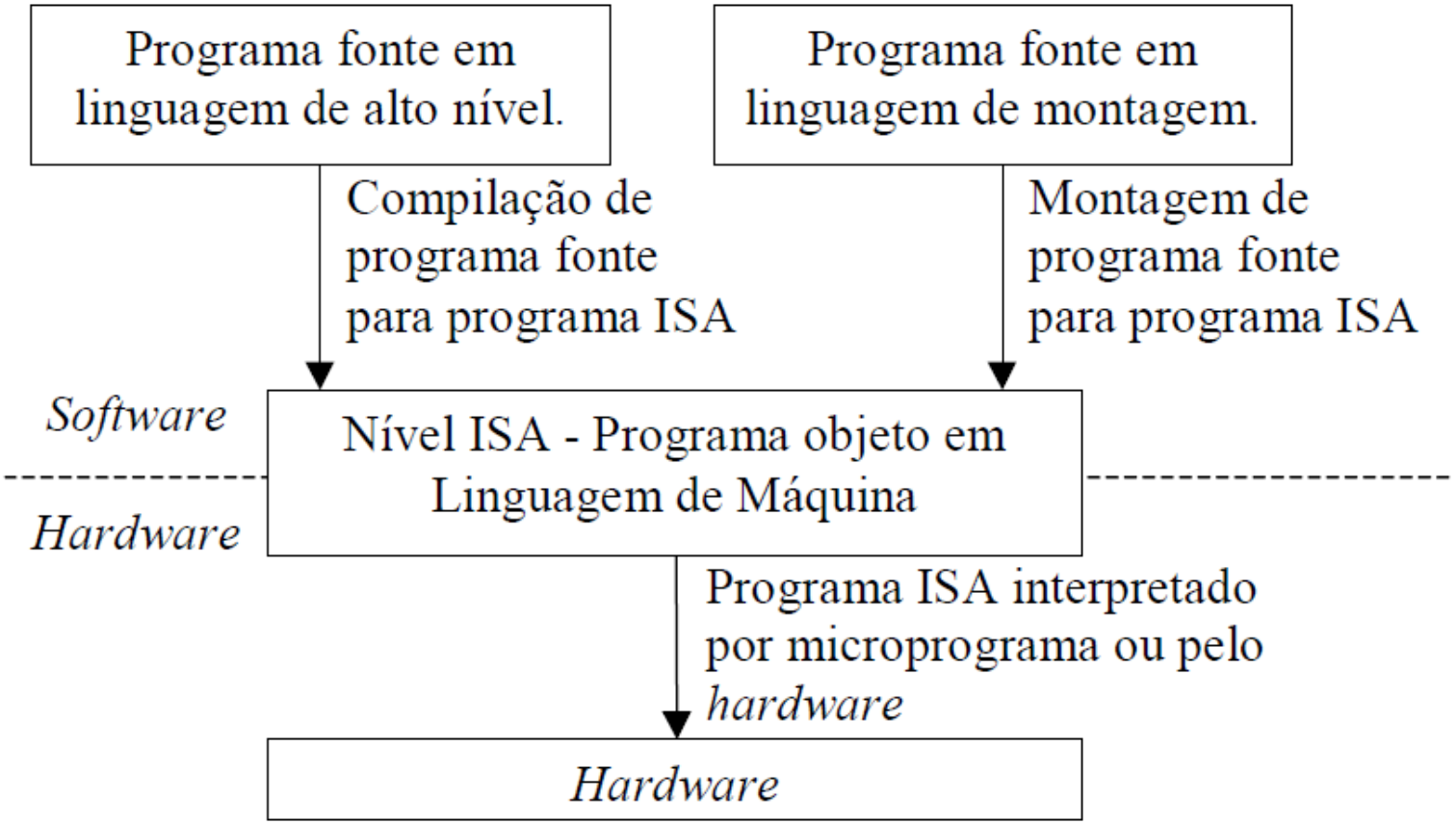


Figura 2.1. Nível ISA - interface *software* / *hardware*.

Nível 2 (Nível ISA)
(cont.)

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

Figure 5-33. A selection of the Pentium II integer instructions.

Nível 2 (Nível ISA) (cont.)

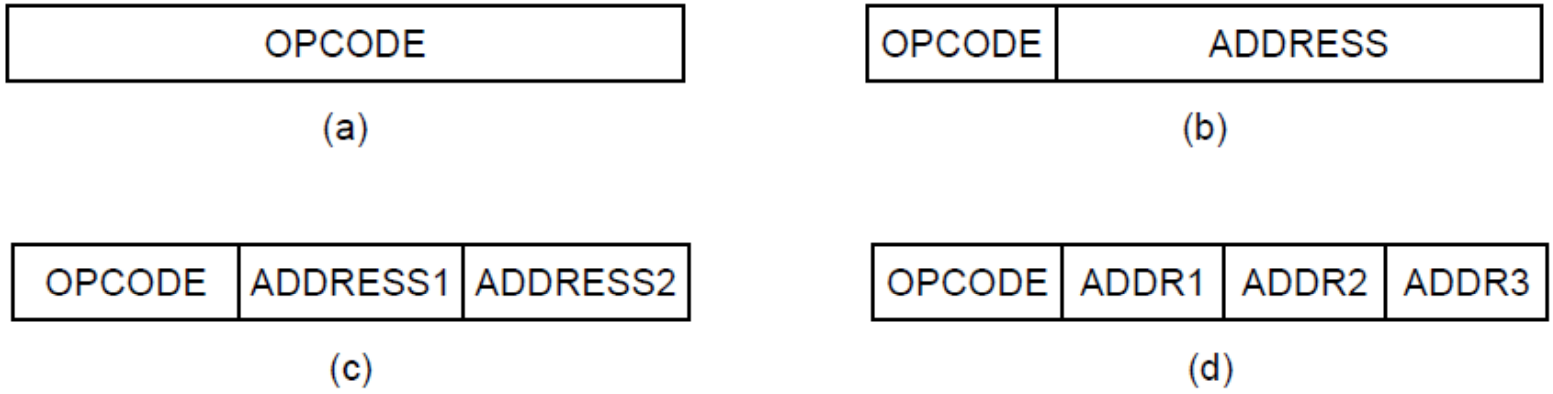


Figure 5-9. Four common instruction formats: (a) Zero-address instruction. (b) One-address instruction (c) Two-address instruction. (d) Three-address instruction.

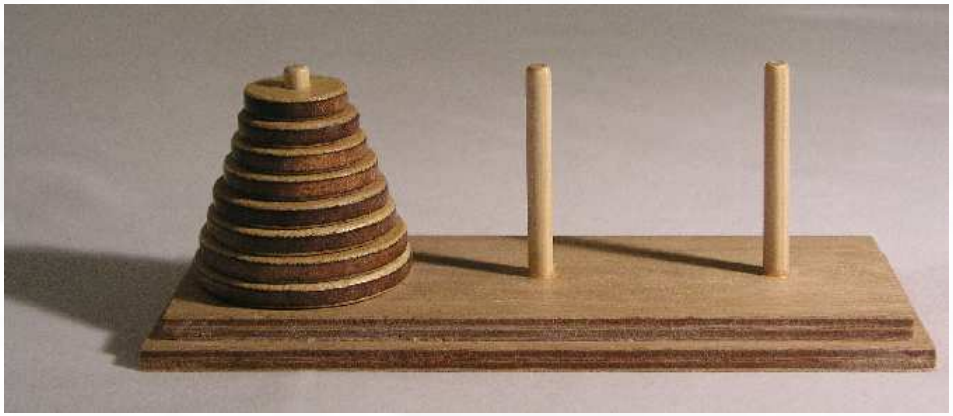
Figure 5-16. An immediate instruction for loading 4 into register 1.

Nível 2 (Nível ISA) (cont.)

```
        MOV R1,#0      ; accumulate the sum in R1, initially 0
        MOV R2,#A      ; R2 = address of the array A
        MOV R3,#A+1024; R3 = address if the first word beyond A
LOOP:                  ADD R1,(R2); register indirect through R2 to get operand
        ADD R2,#4      ; increment R2 by one word (4 bytes)
        CMP R2,R3      ; are we done yet?
        BLT LOOP       ; if R2 < R3, we are not done, so continue
```

Figure 5-17. A generic assembly program for computing the sum of the elements of an array.

Nível 2 (Nível ISA) (cont.)



```
        .586                    ; compile for Pentium (as opposed to 8088 etc.)
.MODEL FLAT
PUBLIC _towers                  ; export 'towers'
EXTERN _printf:NEAR             ; import printf
.CODE
_towers:                        PUSH EBP; save EBP (frame pointer)
        MOV EBP, ESP            ; set new frame pointer above ESP
        CMP [EBP+8], 1          ; if (n == 1)
        JNE L1                  ; branch if n is not 1
        MOV EAX, [EBP+16]       ; printf(" ...", i, j);
        PUSH EAX                ; note that parameters i, j and the format
        MOV EAX, [EBP+12]       ; string are pushed onto the stack
        PUSH EAX                ; in reverse order. This is the C calling convention
        PUSH OFFSET FLAT:format; offset flat means the address of format
        CALL _printf            ; call printf
        ADD ESP, 12             ; remove params from the stack
        JMP Done                ; we are finished
L1:     MOV EAX, 6              ; start k = 6 – i – j
        SUB EAX, [EBP+12]       ; EAX = 6 – i
        SUB EAX, [EBP+16]       ; EAX = 6 – i – j
        MOV [EBP+20], EAX       ; k = EAX
        PUSH EAX                ; start towers(n – 1, i, k)
        MOV EAX, [EBP+12]       ; EAX = i
        PUSH EAX                ; push i
        MOV EAX, [EBP+8]        ; EAX = n
        DEC EAX                 ; EAX = n – 1
        PUSH EAX                ; push n – 1
        CALL _towers            ; call towers(n – 1, i, 6 – i – j)
        ADD ESP, 12             ; remove params from the stack
        MOV EAX, [EBP+16]       ; start towers(1, i, j)
        PUSH EAX                ; push j
        MOV EAX, [EBP+12]       ; EAX = i
        PUSH EAX                ; push i
        PUSH 1                  ; push 1
        CALL _towers            ; call towers(1, i, j)
        ADD ESP, 12             ; remove params from the stack
        MOV EAX, [EBP+12]       ; start towers(n – 1, 6 – i – j, i)
        PUSH EAX                ; push i
        MOV EAX, [EBP+20]       ; push 20
        PUSH EAX                ; push k
        MOV EAX, [EBP+8]        ; EAX = n
        DEC EAX                 ; EAX = n–1
        PUSH EAX                ; push n – 1
        CALL _towers            ; call towers(n – 1, 6 – i – j, i)
        ADD ESP, 12             ; adjust stack pointer
Done:   LEAVE                   ; prepare to exit
        RET 0                   ; return to the caller
.DATA
format  DB "Move disk from %d to %d\n"; format string
END
```

Figure 5-45. The Towers of Hanoi for the Pentium II.

Nível 3 ou Nível do Sistema Operacional (SO)

- Esse nível suporta um conjunto de novas instruções, uma organização diferente da memória, a capacidade de rodar dois ou mais programas de forma simultânea, e outros.
- Fornece serviços básicos para os níveis acima
  - Interface (gráfica ou linha de comando) com o usuário
  - Gerenciamento de memória
  - Escalonamento de processos
  - Acionamento de dispositivos de entrada e saída de dados, etc.
- Este nível é geralmente desenvolvido de forma híbrida, ou seja, parte em uma linguagem de alto nível (como C ou C++, por exemplo), e parte em linguagem de montagem.

Nível 3 ou Nível do Sistema Operacional (SO) (cont.)

- Níveis abaixo: dirigidos aos **programadores de sistema**
  - Níveis projetados para rodar interpretadores e tradutores (ex. compiladores)
  - Os programadores de sistema são especialistas em projetar e implementar novas máquinas virtuais/interpretadores/tradutores.
  - Predominância de Interpretação.
  - Linguagens frequentemente numéricas, bom para as máquinas, mas ruim para as pessoas.
- Níveis acima: dirigidos aos **programadores de aplicação**
  - Programadores com problemas a serem solucionados.
  - Predominância de tradução (mas nem sempre).
  - Linguagens contendo palavras e abreviações (significativas para as pessoas).

Nível 4 ou Nível de Linguagem de Montagem

- Uma forma simbólica de representação das linguagens dos níveis mais baixos.
- Provê um método para as pessoas escreverem programas para os níveis 1, 2, e 3 de uma maneira não tão desconfortável.
  - Mnemônicos para as instruções de máquina.
  - Linguagem Assembly
- Os programas escritos em linguagem de montagem são primeiramente traduzidos para a linguagem dos níveis 1, 2 ou 3, e depois interpretados.
- **Montador (Assembler)**: programa que executa a tradução dos programas em linguagem de montagem para uma linguagem do nível 1, 2 ou 3.

Nível 4 ou Nível de Linguagem de Montagem

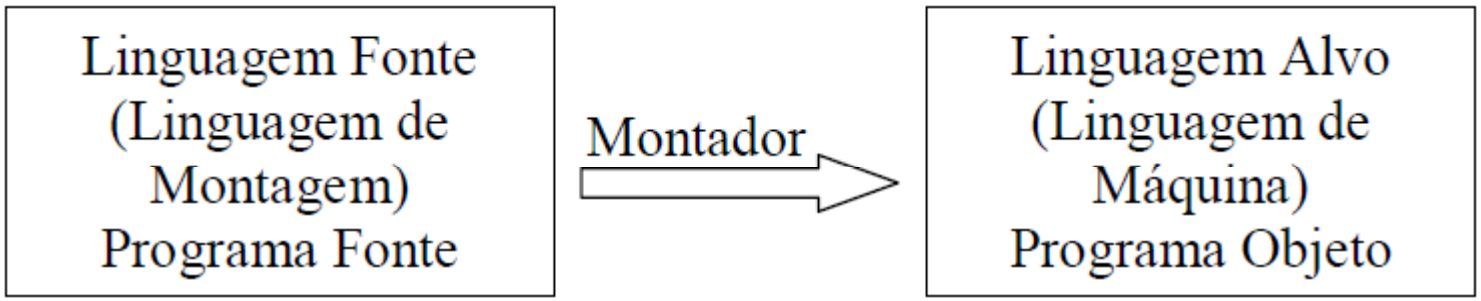


Figura 7.1. Processo de montagem.

Nível 4 ou Nível de Linguagem de Montagem

- Razões para uso da linguagem de montagem:
  - Melhorar o desempenho.
  - Algumas máquinas podem não ter um compilador disponível.
  - Cada comando em linguagem resulta em um comando em linguagem de máquina: uma linha de programa fonte º uma linha de programa objeto.
  - Representação simbólica da linguagem de máquina: Códigos de operação e operandos (números) substituídos por mnemônicos e rótulos.
  - Ao contrário da programação em alto nível, o programador de linguagem de montagem dispõe de todos os recursos do nível ISA.
  - Programa não portável (válido apenas para uma mesma família de processadores).

Nível 5 ou Nível de Linguagens de Alto Nível

- Linguagens projetadas para serem utilizadas por programadores de aplicação com problemas a serem resolvidos.
- Ex. de linguagens de alto nível: C, C++, Python, Java...
- Os programas escritos nessas linguagens são geralmente traduzidos para o nível 3 ou nível 4 por tradutores conhecidos como **compiladores**, embora às vezes sejam interpretados (como no caso de Java e Python).

Mais Níveis ???

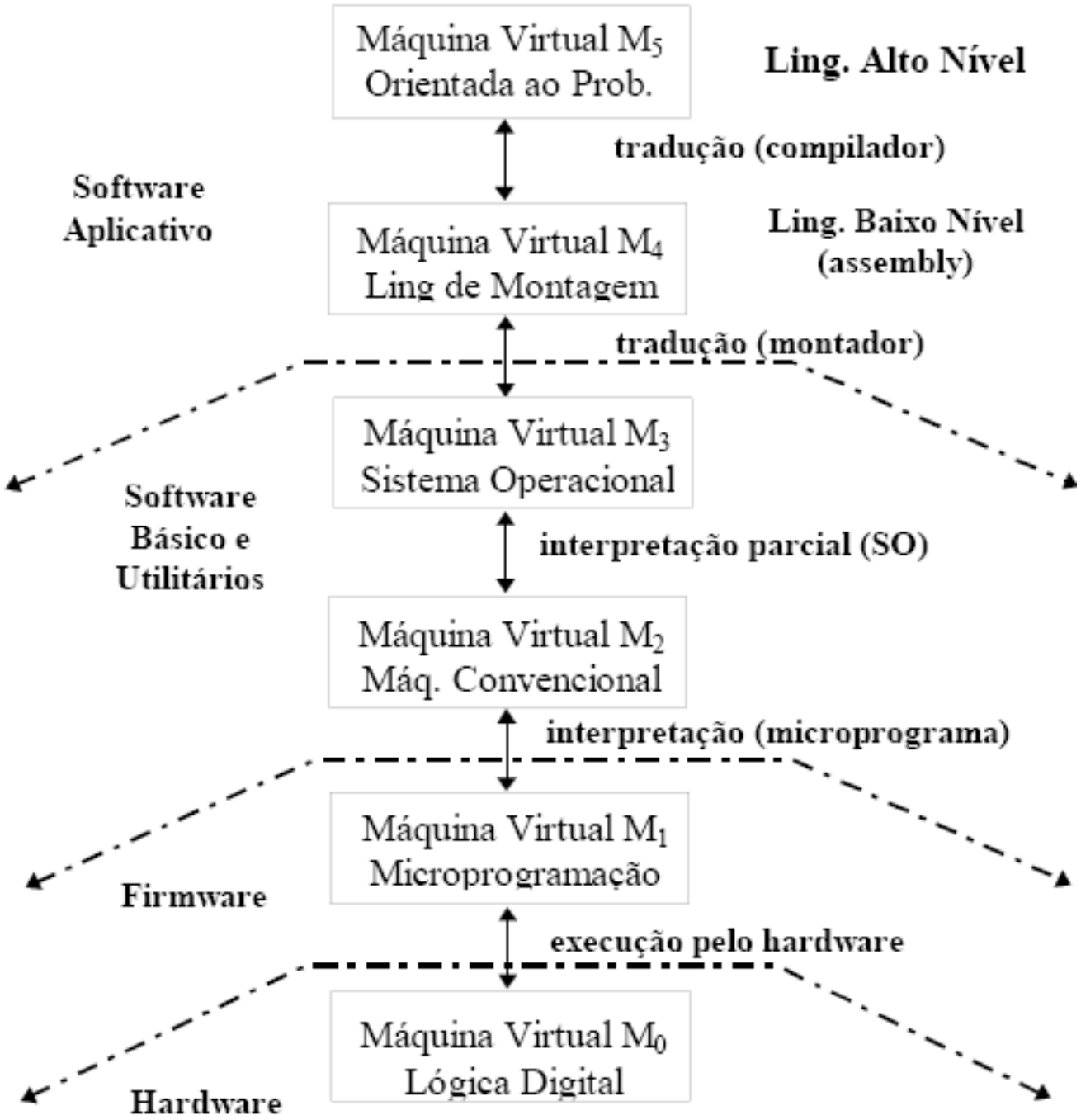
- Acima do nível 5 encontram-se coleções de programas projetados para criar máquinas especialmente adequadas para certas aplicações (ou domínios), contendo grandes quantidades de informação acerca da aplicação.
- Máquinas virtuais voltadas às aplicações
  - Administração, educação, projeto de computadores, realidade virtual, etc.
- Dependendo do projeto da arquitetura, os níveis podem variar

Observações Finais

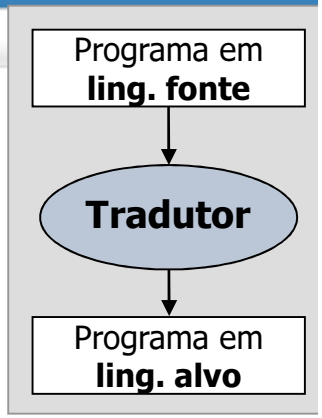
- Pontos fundamentais:
  - Computadores são projetados como uma série de níveis
  - Cada nível é construído em cima de seus precursores.
- Cada nível representa uma abstração distinta, com diferentes objetos e operações presentes
- A abstração permite ignorar, "abstrair", temporariamente detalhes irrelevantes, de níveis mais baixos, reduzindo uma questão complexa a algo muito mais fácil de ser entendido.
  - **Arquitetura do Nível**: conjunto de tipos de dados, instruções e características

Arquitetura de Computadores:
é o estudo de como projetar as partes de um sistema de computador visíveis aos programadores.

Hardware, Software e Firmware (2)



Execução de um Programa



Compiladores, Montadores, Ligadores e Carregadores (1)

- **Compiladores**
  - São programas que recebem como entrada arquivos texto contendo módulos escritos em linguagem de alto nível e geram como saída programa em linguagem de montagem (ou diretamente arquivos objeto) correspondentes a cada módulo.
  - Se todas as bibliotecas ou módulos são apresentados como entrada, geram um programa executável diretamente.
- **Montadores** *(Assemblers)*
  - Montam um programa em linguagem de máquina a partir de sua versão em linguagem de montagem.
  - Geram um **arquivo objeto.** Em geral, não pode ser executado diretamente pela máquina, por conter referências a sub-rotinas e dados especificados em outros arquivos (bibliotecas).

Ambos são TRADUTORES!

Compiladores, Montadores, Ligadores e Carregadores (2)

- **Ligadores (Linkers)**
  - São programas especiais que recebem como entrada arquivos objetos e geram como saída o programa final em linguagem de máquina.
  - Gera um programa executável a partir de um ou mais arquivos objeto.
- **Carregadores (Loaders)**
  - Para executar um programa, um loader deve ser utilizado.
  - O carregador é, em geral, parte do sistema operacional.

Exercícios de Revisão

- Qual a diferença entre interpretação e tradução?

- Considere um computador multinível no qual todos os níveis são diferentes. Cada nível tem instruções que são *m* vezes mais poderosas do que as do nível abaixo dele; isto é uma instrução de nível *r* pode fazer o trabalho de *m* instruções de nível *r-1*. Se um programa de nível 1 requer *k* segundos para executar, quanto tempo levariam programas equivalentes nos níveis 2, 3 e 4 admitindo que são requeridas *n* instruções de nível *r* para interpretar uma única instrução do nível *r+1*?

- Algumas instruções no nível do sistema operacional da máquina são idênticas a instruções em linguagem ISA. Elas são executadas diretamente pelo microprograma, e não pelo sistema operacional. Considerando sua resposta ao problema anterior, porque você acha que isso acontece?

Referências

- Andrew S. Tanenbaum, ***Organização Estruturada de Computadores***, 5ª edição, Prentice-Hall do Brasil, 2001.
  - Capítulo 1
- Lúcia Helena M. Pacheco, **Visão Geral de Organização Estruturada de Computadores e Linguagem de Montagem.** Universidade Federal de Santa Catarina. Centro Tecnológico, Departamento de Informática e de Estatística.
  - http://www.scribd.com/doc/2575596/1OrganizacaoEstruturada1