

INTRODUÇÃO AO CURSO

Anotações do material suplementar (apresentações PPT) ao Livro do Hennessy e Patterson e do material do Prof. Celso Alberto Saibel Santos (DI/CT).

PARTE 1 – PLANO DE CURSO

Prof. Rodolfo da Silva Villaça

Departamento de Informática (DI/CT)
Universidade Federal do Espírito Santo (UFES)

rodolfo.villaca@inf.ufes.br / rodolfo.villaca@ufes.br

Ementa

3

Resumidamente:

- Introdução à Arquitetura e Organização de Computadores
- Questões de desempenho e eficiência
- Linguagem de Máquina e Conjunto de Instruções
- Projeto de Processador: monociclo e pipeline
- Sistema de Memória
- ~~• Interconexão e entrada e saída~~

□ Plano de Curso (ver pdf)

Referências

REFERÊNCIA BÁSICA

- ❑ Patterson, David A.; Hennessy, John. L. **Organização e Projeto de Computadores**, 4a Ed, 2012 (Americana). Elsevier.

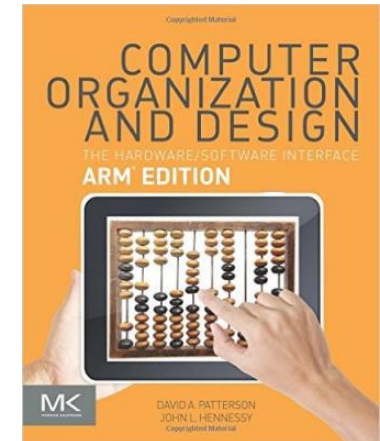
REFERÊNCIAS COMPLEMENTARES

- ❑ Hennessy, John. L.; Patterson, David A. Arquitetura de Computadores: Uma abordagem quantitativa, 4a Edição, 2007. Campus.
- ❑ TANENBAUM, A. Arquitetura de Computadores, 5ª Edição, Prentice Hall do Brasil.
- ❑ STALLINGS, W. Arquitetura de Computadores, 5ª Edição, Prentice Hall do Brasil.

Edições mais recentes

5

- David A. Patterson, John L. Hennessy. Organização e Projeto de Computadores, Tradução da 5a Ed, 2014. Elsevier.
- David A. Patterson, John L. Hennessy. Computer Organization and Design ARM Edition: The Hardware Software Interface – ARM Edition (The Morgan Kaufmann Series in Computer Architecture and Design) 1st Edition. 2017.



PARTE 2 – INTRODUÇÃO ARQUITETURA E PROGRAMA ARMAZENADO



Sistemas Computacionais

- Podem ser vistos como uma estrutura de 3 partes:

1. Hardware e seus Componentes;
2. Software;
3. Dados e sinais.

Sistemas Computacionais

- Os sistemas (computacionais) são normalmente complexos demais para serem estudados em todos seus detalhes:
SOLUÇÃO → Diferentes níveis de **ABSTRAÇÃO**
- **Abstração:** distinção entre as propriedades externas de um componente e os detalhes internos de sua construção
 - Utilizamos dispositivos elétricos sem entender a sua tecnologia;
 - Utilizamos serviços de terceiros sem conhecer com detalhes suas atividades;
 - Tentar fazer o sistema “o mais invisível possível”.

Abstração

Removendo camadas
temos menor abstração

Uma abstração omite detalhes
desnecessários, ajudando a reduzir
a complexidade

High-level
language
program
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

C compiler

Assembly
language
program
(for MIPS)

```
swap:
  muli $2, $5, 4
  add $2, $4, $2
  lw $15, 0($2)
  lw $16, 4($2)
  sw $16, 0($2)
  sw $15, 4($2)
  jr $31
```

Assembler

Binary machine
language
program
(for MIPS)

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
100011001111001000000000000000100
10101100111100100000000000000000
101011000110001000000000000000100
0000001111100000000000000001000
```

hardware

Código em Linguagem Binária

MIPS

```
001001111011110111111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
1010111110100000000000000011000
1010111110100000000000000011100
1000111110101110000000000011100
1000111110111000000000000011000
0000000111001110000000000011001
0010010111001000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
0001010000100000111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
000000111110000000000000001000
00000000000000000001000000100001
```

Código em **Ling. Montagem**

- Assembly;
- Assembler;
- MIPS.

```
addiu    $29, $29, -32
sw       $31, 20($29)
sw       $4, 32($29)
sw       $5, 36($29)
sw       $0, 24($29)
sw       $0, 28($29)
lw       $14, 28($29)
lw       $24, 24($29)
multu    $14, $14
addiu    $8, $14, 1
slti     $1, $8, 101
sw       $8, 28($29)
mflo     $15
addu     $25, $24, $15
bne      $1, $0, -9
sw       $25, 24($29)
lui      $4, 4096
lw       $5, 24($29)
jal      1048812
addiu    $4, $4, 1072
lw       $31, 20($29)
addiu    $29, $29, 32
jr       $31
move     $2, $0
```

Código em Ling. Montagem (labels)

- Assembly;
- Assembler;
- MIPS.

```
.text
.align 2
.globl main
main:
    subu    $sp, $sp, 32
    sw      $ra, 20($sp)
    sd      $a0, 32($sp)
    sw      $0, 24($sp)
    sw      $0, 28($sp)
loop:
    lw      $t6, 28($sp)
    mul     $t7, $t6, $t6
    lw      $t8, 24($sp)
    addu    $t9, $t8, $t7
    sw      $t9, 24($sp)
    addu    $t0, $t6, 1
    sw      $t0, 28($sp)
    ble     $t0, 100, loop
    la      $a0, str
    lw      $a1, 24($sp)
    jal     printf
    move    $v0, $0
    lw      $ra, 20($sp)
    addu    $sp, $sp, 32
    jr      $ra

.data
.align 0
str:
.asciiz "The sum from 0 .. 100 is %d\n"
```

- Labels: abstração de endereços de memória;

Código em Linguagem C

```
#include <stdio.h>

int
main (int argc, char *argv[])
{
    int i;
    int sum = 0;

    for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
    printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

Hello world em Assembly x86

```
#include <stdio.h>

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

    printf("hello world\n");
    return 0;

}
```

```
00000000040050c <main>:
40050c: 55                push    %rbp
40050d: 48 89 e5          mov     %rsp,%rbp
400510: 48 83 ec 10       sub     $0x10,%rsp
400514: 89 7d fc          mov     %edi,-0x4(%rbp)
400517: 48 89 75 f0       mov     %rsi,-0x10(%rbp)
40051b: bf dc 05 40 00    mov     $0x4005dc,%edi
400520: e8 bb fe ff ff    callq   4003e0 <puts@plt>
400525: b8 00 00 00 00    mov     $0x0,%eax
40052a: c9               leaveq  %eax
40052b: c3               retq
40052c: 90               nop
40052d: 90               nop
40052e: 90               nop
40052f: 90               nop
```

Hello world em Assembly ARM

```
.data
msg:.ascii      "Hello, ARM!\n"
len = . - msg
.text
.globl _start
_start:
    /* syscall write(int fd, const void *buf, size_t count) */
    mov     %r0, $1          /* fd -> stdout */
    ldr     %r1, =msg        /* buf -> msg */
    ldr     %r2, =len        /* count -> len(msg) */
    mov     %r7, $4          /* write is syscall #4 */
    swi     $0               /* invoke syscall */

    /* syscall exit(int status) */
    mov     %r0, $0          /* status -> 0 */
    mov     %r7, $1          /* exit is syscall #1 */
    swi     $0               /* invoke syscall */
```

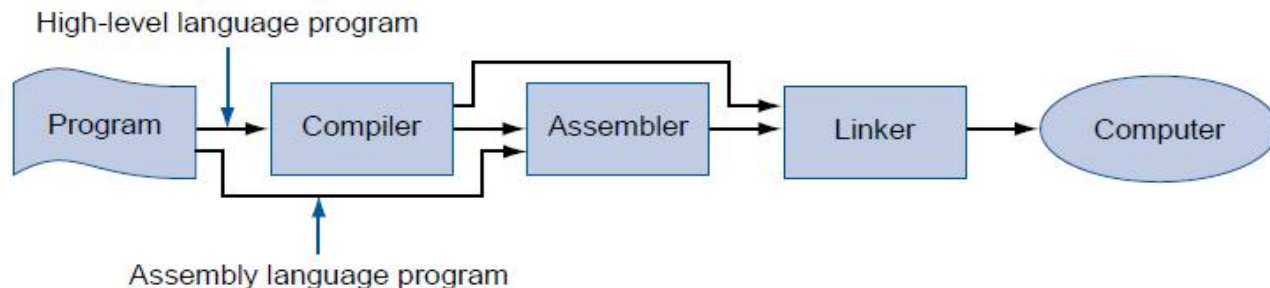
Hello world em Assembly ARM

00008314 <_start>:

```
8314: f04f 0b00 mov.w    fp, #0
8318: f04f 0e00 mov.w    lr, #0
831c: bc02      pop      {r1}
831e: 466a      mov      r2, sp
8320: b404      push     {r2}
8322: b401      push     {r0}
8324: f8df c010 ldr.w    ip, [pc, #16]; 8338 <_start+0x24>
8328: f84d cd04 str.w    ip, [sp, #-4]!
832c: 4803      ldr      r0, [pc, #12]; (833c <_start+0x28>)
832e: 4b04      ldr      r3, [pc, #16]; (8340 <_start+0x2c>)
8330: f7ff efde blx      82f0 <_init+0x2c>
8334: f7ff efe8 blx      8308 <_init+0x44>
8338: 00008449 .word    0x00008449
833c: 000083f1 .word    0x000083f1
8340: 00008409 .word    0x00008409
```


Geração código em Ling. de Máquina

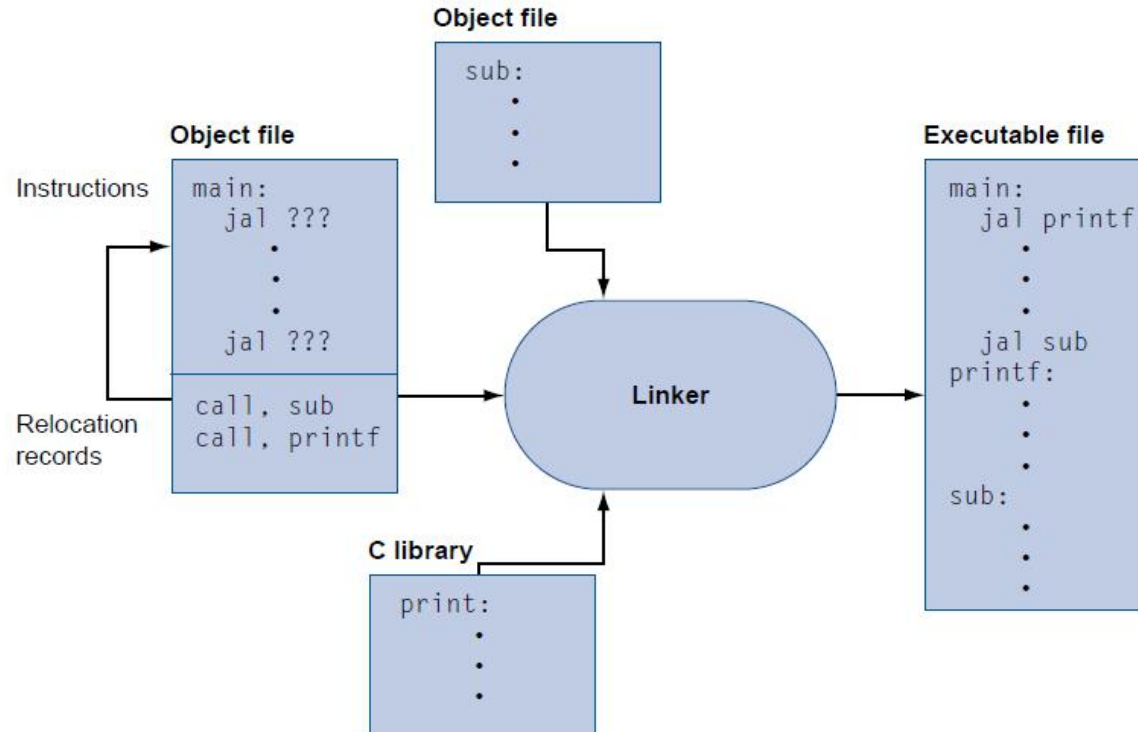
- Um programa em *assembly* pode ser a saída do compilador ou ser escrito por um programador.



Exemplo de “desassemblador” online:

<https://www.onlinedisassembler.com/odaweb/>

Ligador (*linker*)



Níveis de Abstração

Programa em
Linguagem de Alto-Nível

Compilador

Programa em Linguagem
de Montagem

Montador (Assembler)

Programa em
Linguagem de Máquina

Conjunto de Instruções

Interpretação da Máquina (CPU)

Especificação do controle de
sinais e dos caminhos

*Compilador /
Interpretador*

$x = (a+b)$

$x = x - (c+d)$

lw 15, 0(2)

lw 16, 4(2)

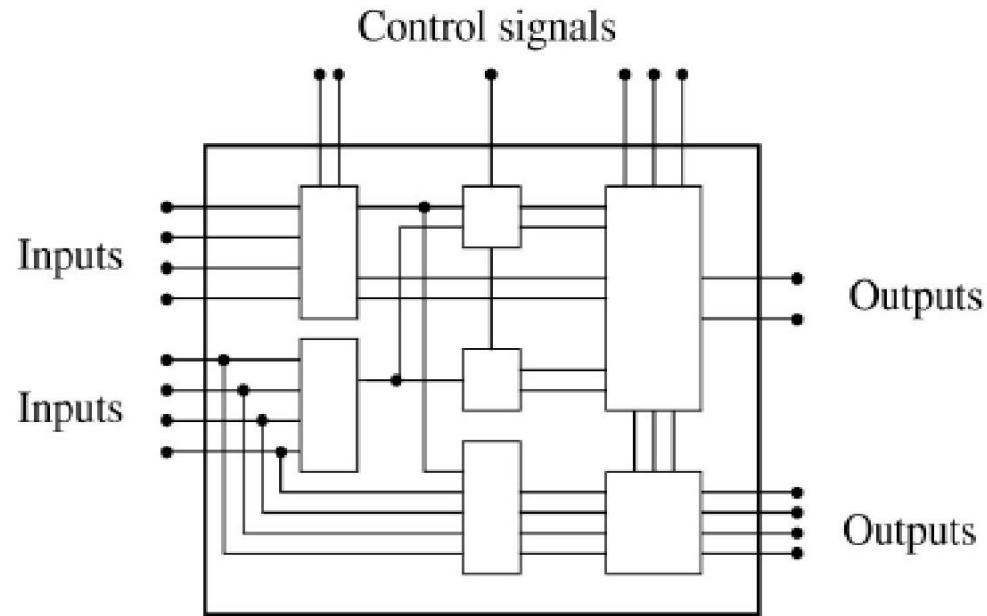
sw 16, 0(2)

sw 15, 4(2)

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111

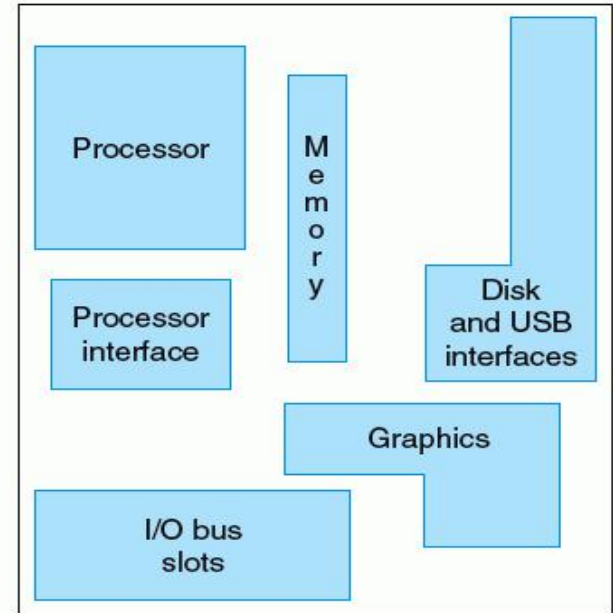
$ALUOP[0:3] \leftarrow InstReg[9:11] \& MASK$

Hardware: Nível “mais baixo”



Hardware “Tradicional”

Motherboard



Placa mãe típica – Fonte Cap1 do Livro Texto

Cubietruck, Raspberry Pi

Sistemas
embarcados
únicos



ARM 32
bits

Era “pós-PC”

23

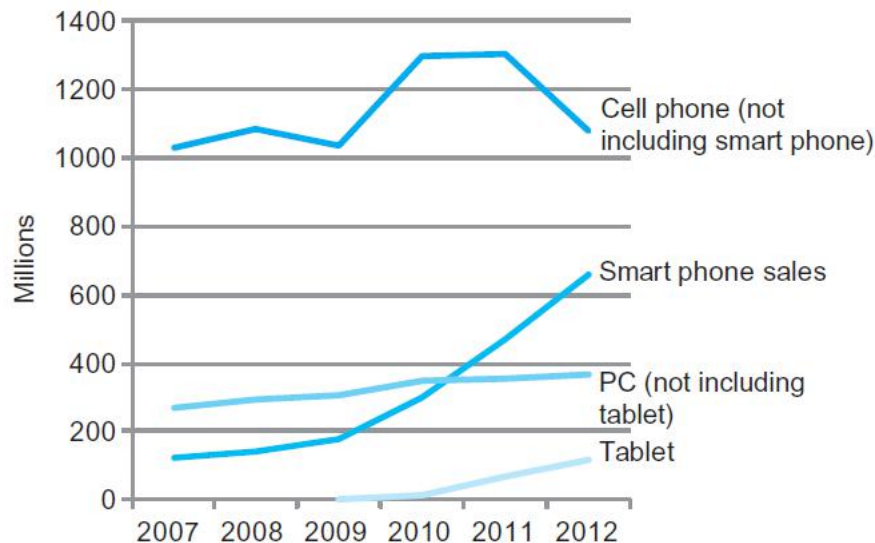


FIGURE 1.2 The number manufactured per year of tablets and smart phones, which reflect the PostPC era, versus personal computers and traditional cell phones. Smart phones represent the recent growth in the cell phone industry, and they passed PCs in 2011. Tablets are the fastest growing category, nearly doubling between 2011 and 2012. Recent PCs and traditional cell phone categories are relatively flat or declining.

HARDWARE: IMPLEMENTAÇÃO DA LÓGICA BINÁRIA



Desempenho e Integração

Célula
Complexa

Lógica de portas
CMOS

Transistores

Ligações

Produzindo uma CPU

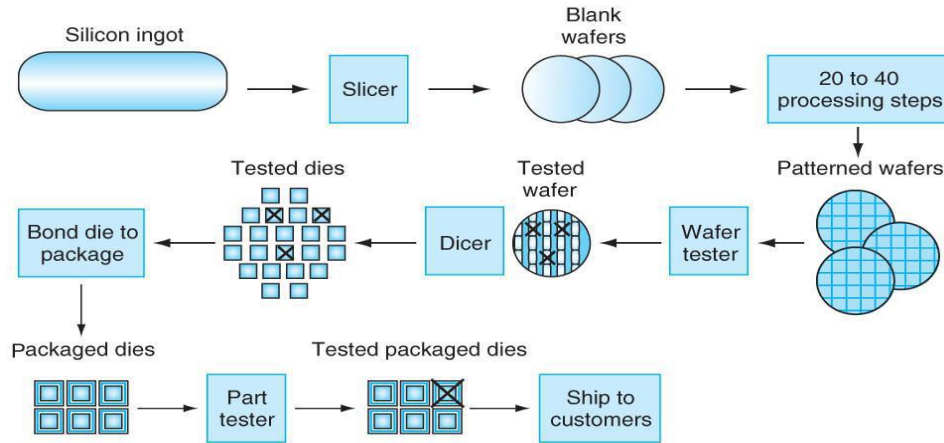


Fig 1.18 Processo de produção de um chip. Após fatiar um pedaço de silício, *wafers* “em branco” passam por 20 a 40 passos para criar *wafers* com os padrões desejados (mostrados no próximo slide). Estes *wafers* “gravados” são então testados com um testador de *wafers* e um mapeamento das partes boas do *wafer* é feito. Então, os *wafers* são divididos em *dies* (próximo slide). Nesta figura, um wafer gerou 20 *dies*, 17 dos quais passaram nos testes (X indica falha). A taxa de dies bons nesta produção é de 17/20, ou 85%. Os *dies* bons são então empacotados em um chip e novamente testados antes de serem enviados aos clientes. No exemplo, 1 pacote ruim foi encontrado nessa última fase.

Copyright © 2009 Elsevier, Inc. All rights reserved.

Pastilha com várias CPUs

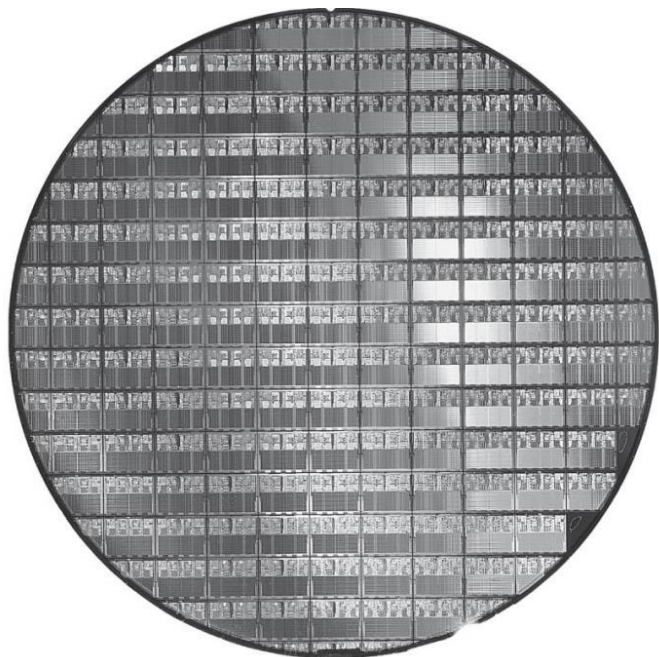
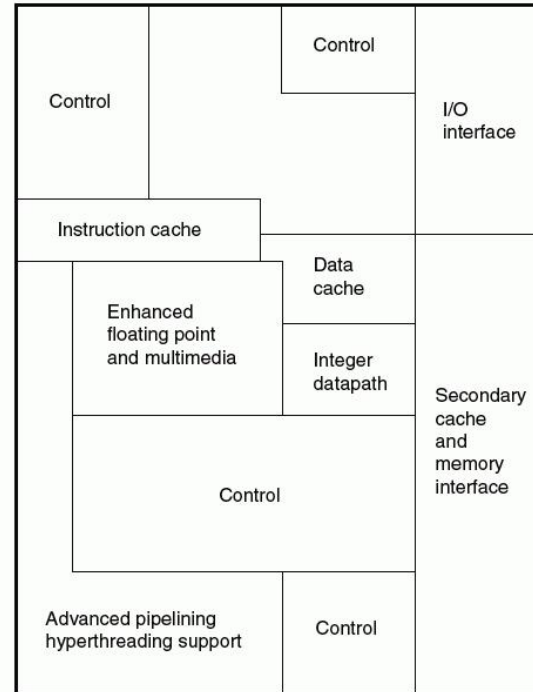


Fig 1.19 Um wafer de 300mm de chips AMD Opteron X2, o predecessor dos chips Opteron X4 (Cortesia AMD). O número de *dies* por wafer com taxa de 100% de sucesso é 117. Este *die* usa tecnologia de **90-nanometros**, o que significa que os menores transistores usados na fabricação tem tamanho aproximado de **90 nm**.

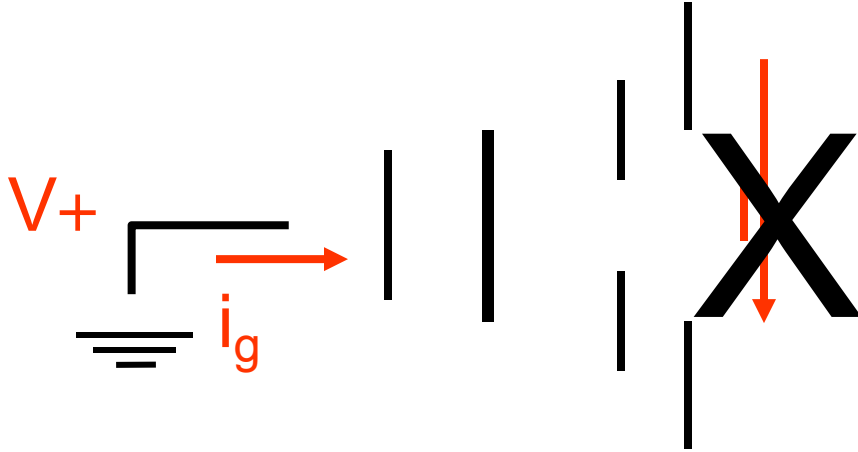
Copyright © 2009 Elsevier, Inc. All rights reserved.

Hardware de uma CPU

Chip Pentium IV – Fonte: Cap1, Livro Texto

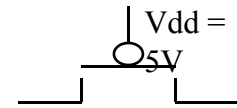
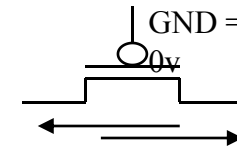
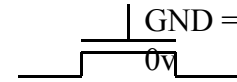
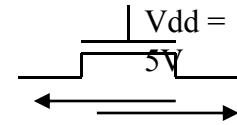


Transistor MOS : Base de Construção



Tecnologia básica: CMOS

- CMOS: Complementary Metal Oxide Semiconductor
 - Transistors NMOS (N-Type Metal Oxide Semiconductor)
 - Transistors PMOS (P-Type Metal Oxide Semiconductor)
- Transistor NMOS
 - Uma tensão alta (HIGH = V_{dd}) no gate “fecha” o circuito
 - Uma tensão baixa (LOW = GND) no gate “abre” o circuito
- Transistor PMOS
 - Uma tensão baixa (LOW = GND) no gate “fecha” o circuito
 - Uma tensão alta (HIGH = V_{dd}) no gate “abre” o circuito



Porta Lógica NOT ou Inversora

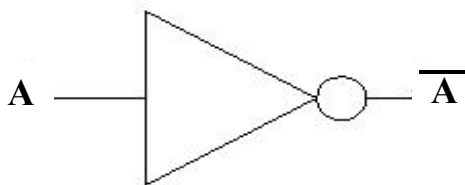
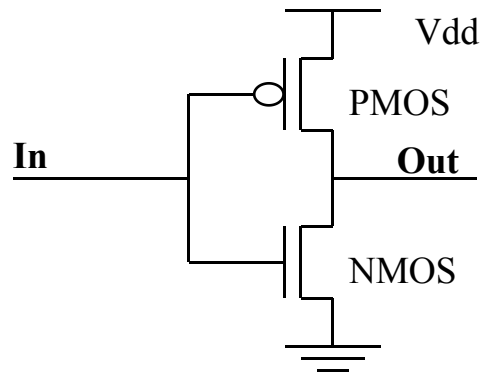
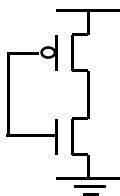
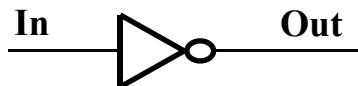
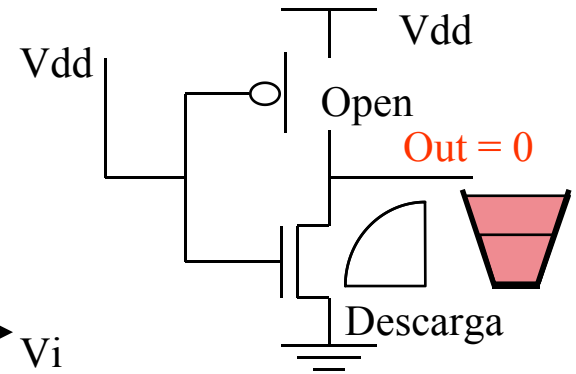
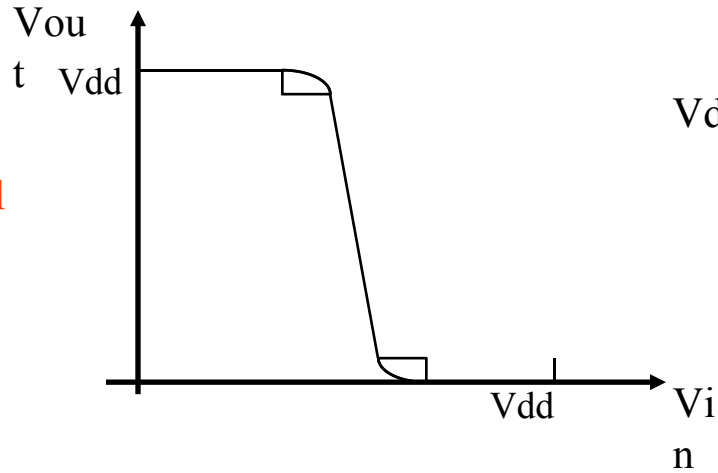
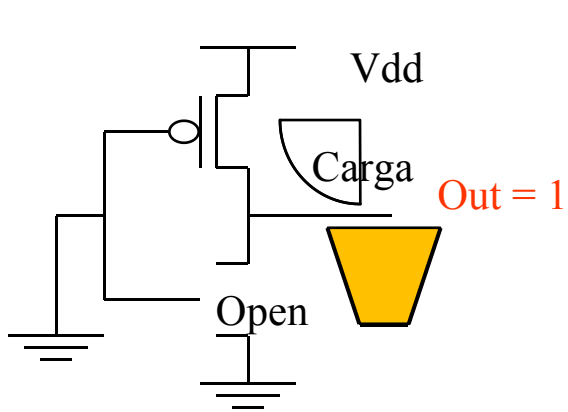


Tabela da Verdade

A	$F = A'$
0	1
1	0

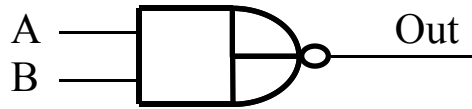


Inversor CMOS

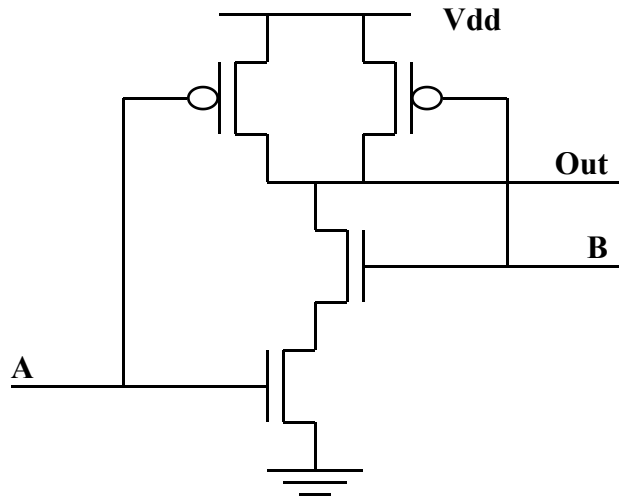


Portas CMOS

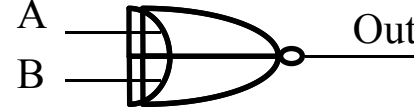
Porta NAND



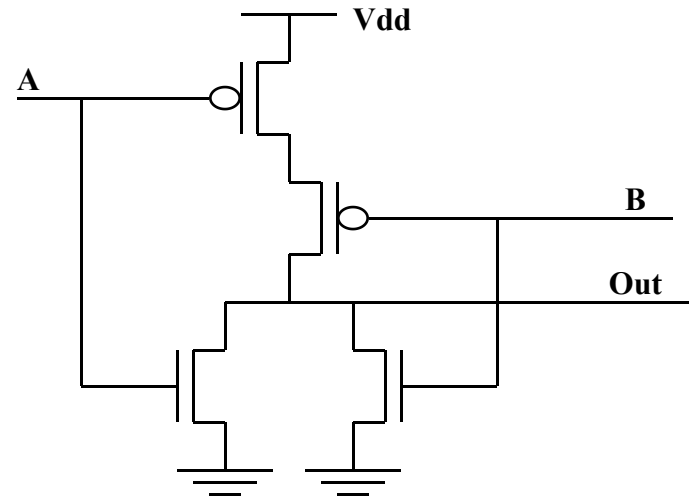
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0



Porta NOR



A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

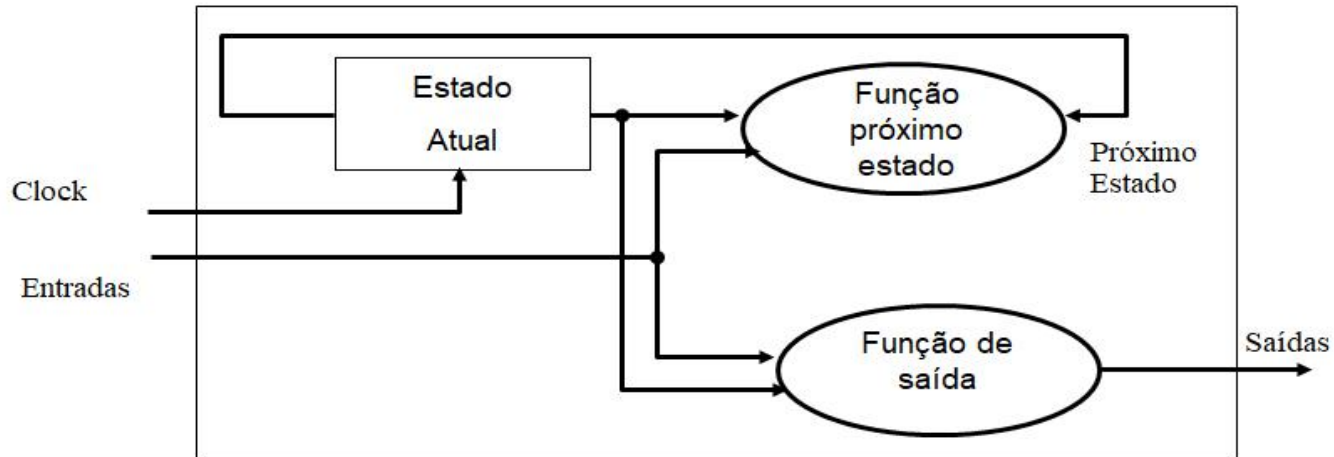


Circuitos Lógicos

- A arquitetura de uma máquina é construída sobre circuitos com lógica **combinacional** e **sequencial**
 - **Circuito Combinacional**: saídas refletem o estado instantâneo das entradas (ex. somador)
 - **Circuito Seqüencial**: progressão de estado em estado na medida que ocorrem mudanças nas entradas (ex. contador)
- Características:
 - Armazenam estado parcial do circuito (têm memória)
 - Usam sinais para promover a transição de um estado a outro (relógios/clocks).

Máquina de Estados

- ❑ Máquina de estados: progressão do sistema → estado atual e função próximo estado
- ❑ Controle de operações em uma CPU
- ❑ Mudança de estado guiada pelo sinal de *clock*: **síncronas**



HARDWARE: SUBSISTEMAS E BLOCOS DA ARQUITETURA



Maioria das máquinas atuais

Máquina de von Neumann (1946 - base da maioria das arquiteturas)

- Máquina composta por:

- **Memória (MEM)**

- Armazena dados, sendo alguns destes instruções de um programa a ser executado pela CPU
- Característica fundamental: Dados e programas estão armazenados numa mesma memória

- **Unidade Central de Processamento (CPU)**

- Unidade de controle (UC)
- Unidade de cálculos aritméticos e lógicos (UAL)
- Registrador de instrução (IR)
- Ponteiro de instrução (IP)
- Registrador de uso geral (acumulador - ACC)

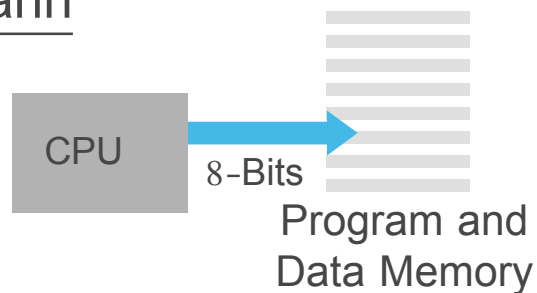
- **Dispositivos de Entrada e Saída**

Como construir um PC ?

<http://lifehacker.com/5828747/how-to-build-a-computer-from-scratch-the-complete-guide>

Harvard

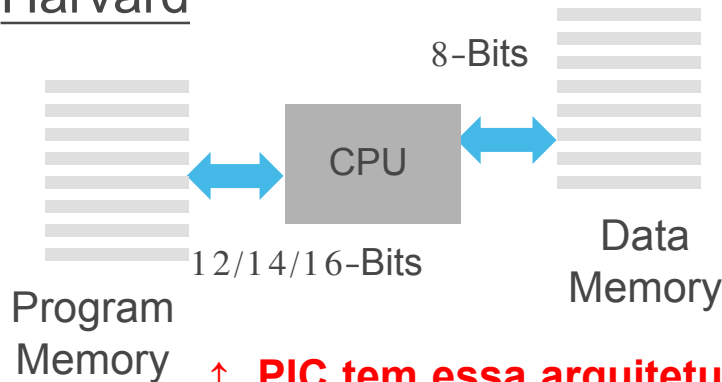
Von Neumann



Busca instruções e dados em uma memória única.

- Limita a banda passante;
- Gestão mais eficiente de memória;

Harvard



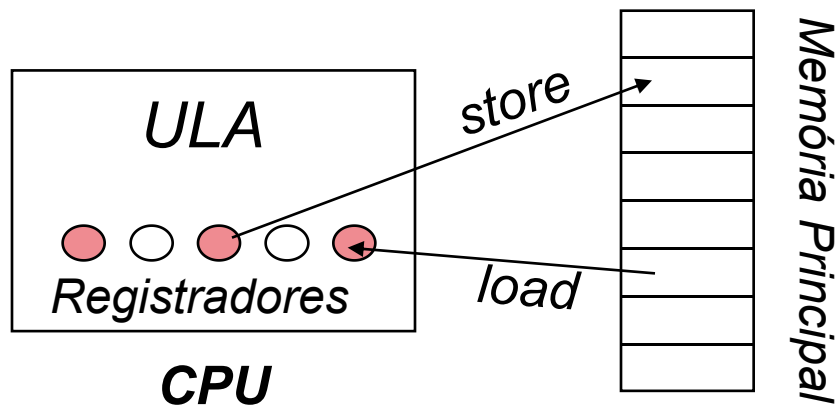
Dois espaços de memória diferentes e endereços para instruções e dados.

- Aumenta a vazão;
- Largura de bandas diferentes para dados e instruções;
- Facilita o pipeline de instruções.

↑ **PIC tem essa arquitetura!**

Armazenando um Programa

- Instruções armazenadas como valores binários.
- Dados também armazenados como valores binários;
- Arquitetura *von Neumann*: Códigos dos programas e seus dados estão armazenados “numa mesma memória”;



Dados e instruções armazenados na mesma memória ...

... então, quando eles são carregados para a CPU, os dados vão para os registradores e as instruções são executadas.