PARTE 2 – INTRODUÇÃO ARQUITETURA E PROGRAMA ARMAZENADO

Sistemas Computacionais

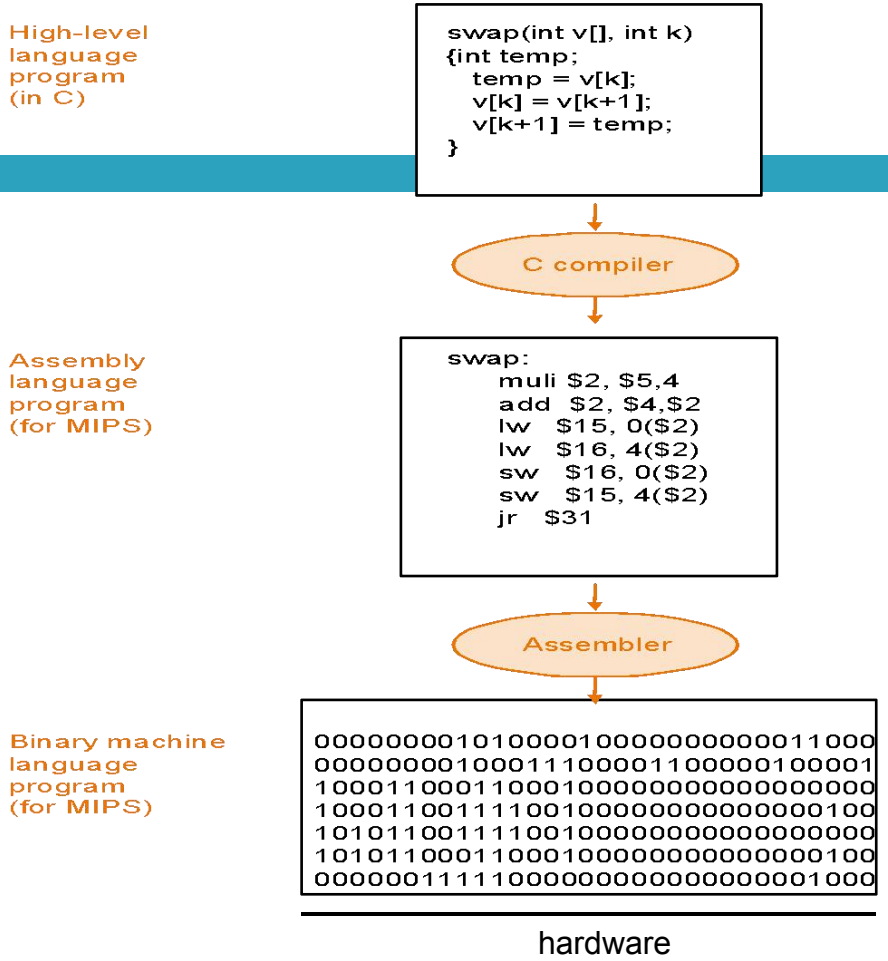
- Podem ser vistos como uma estrutura de 3 partes:

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

Abstração

Removendo camadas temos menor abstração

Uma abstração omite detalhes desnecessários, ajudando a reduzir a complexidade



Código em Linguagem Binária

MIPS

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
```

Hello world em Assembly x86

```c
#include <stdio.h>


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

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

}
```

```
000000000040050c <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
  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
```

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

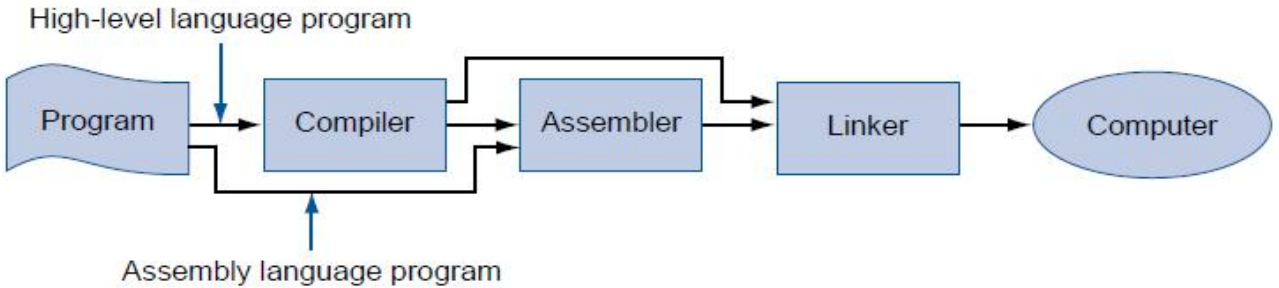
```c
#include <stdio.h>

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);
}
```

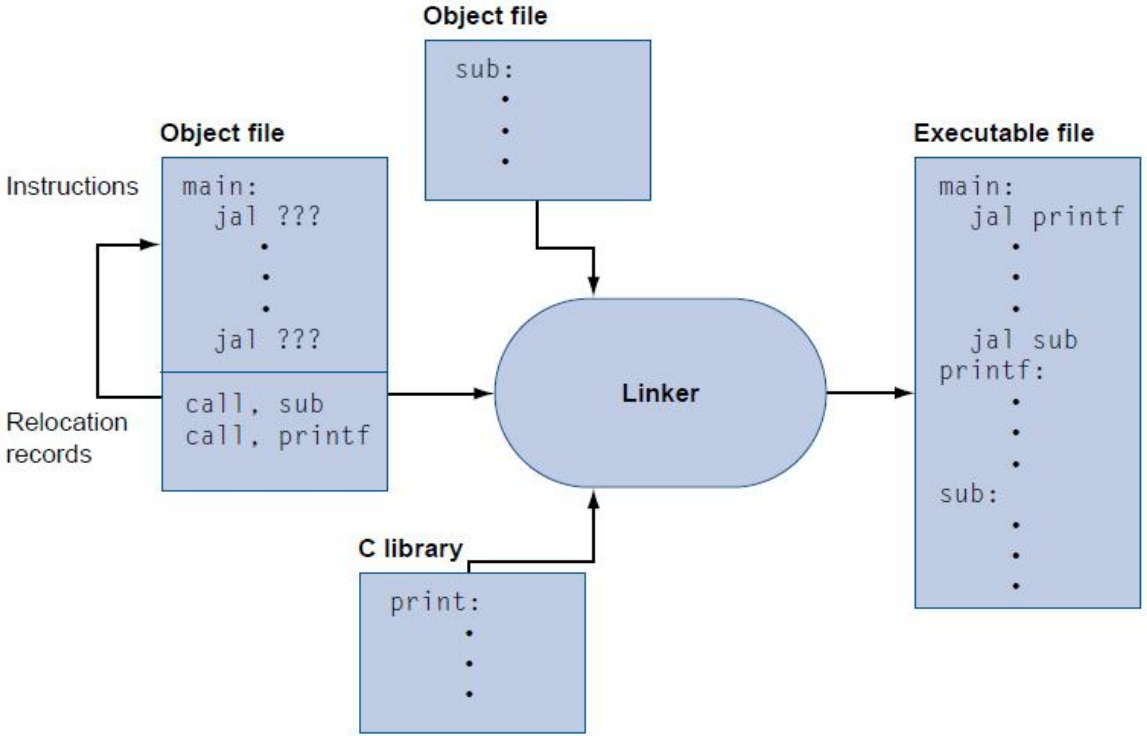
Geração Código

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

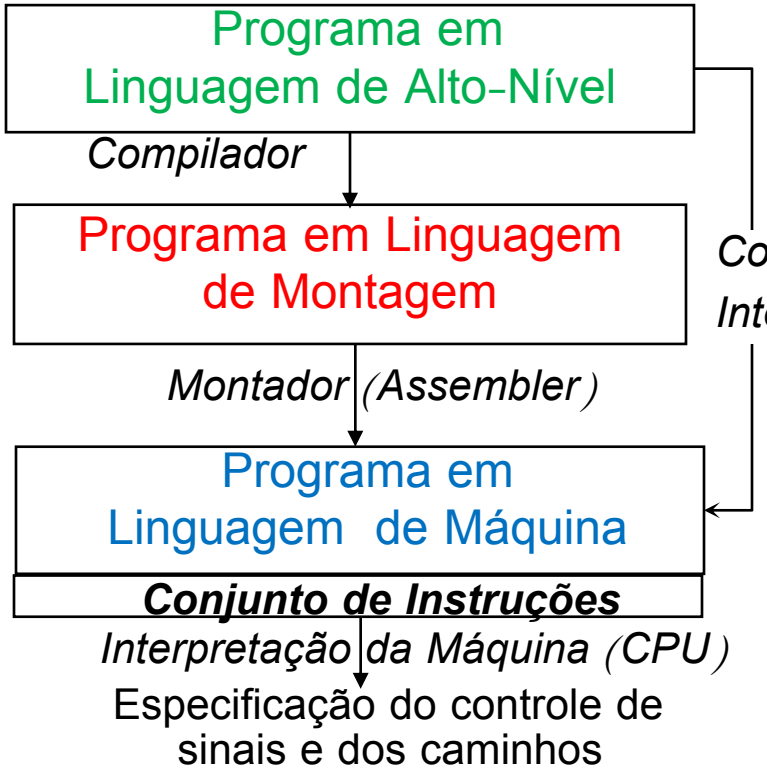


Exemplo de “desassemblador” online:
https://www.onlinedisassembler.com/odaweb/

Ligador (*linker*)



Níveis de Abstração



x = (a+b)

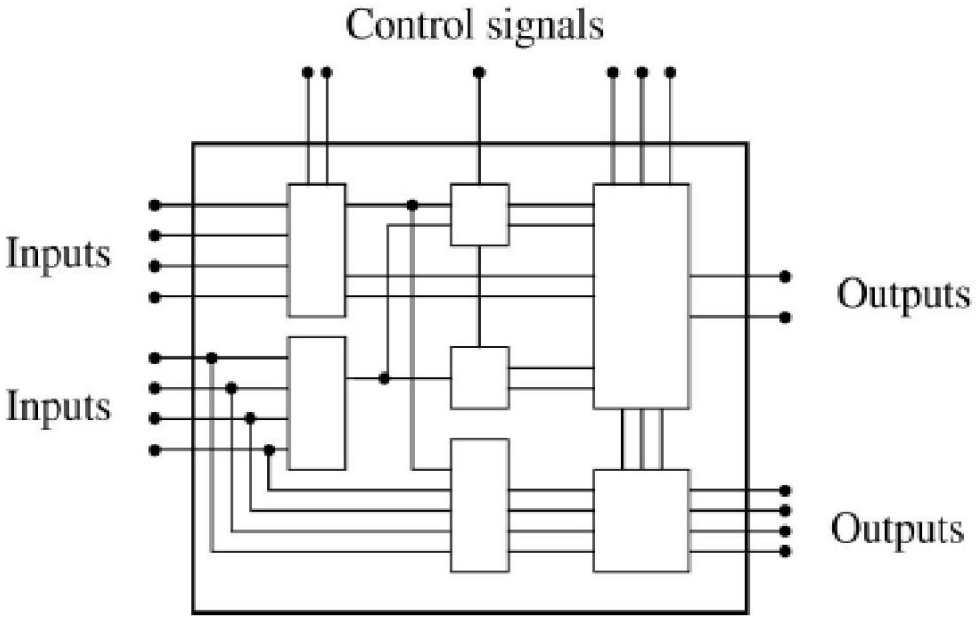
x = x – (c+d)

```
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
```

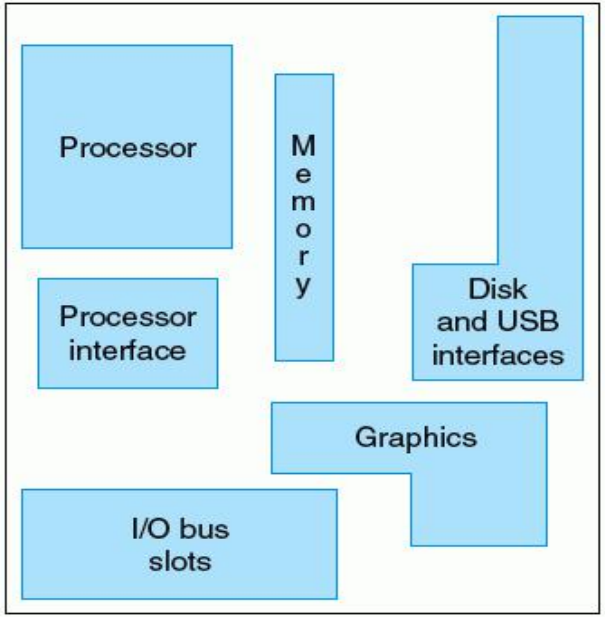
ALUOP[0:3] <= InstReg[9:11] & MASK

Hardware: Nível “mais baixo”



Hardware “Tradicional”

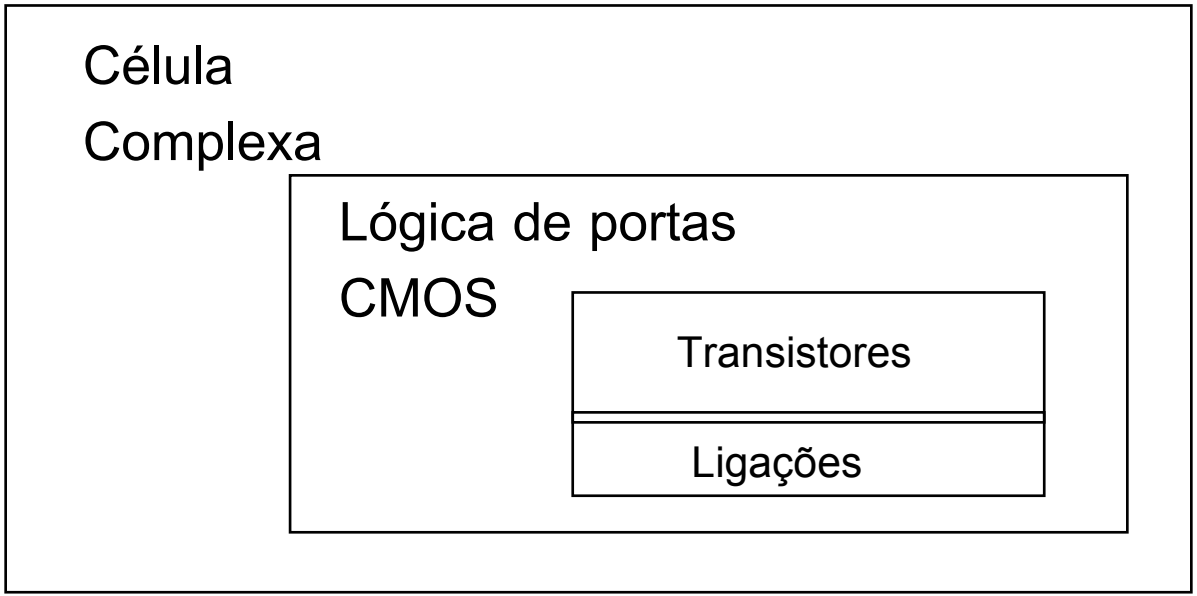
Motherboard



Placa mãe típica – Fonte Cap1 do Livro Texto

HARDWARE: IMPLEMENTAÇÃO DA LÓGICA BINÁRIA

Desempenho e Integração



Produzindo uma CPU

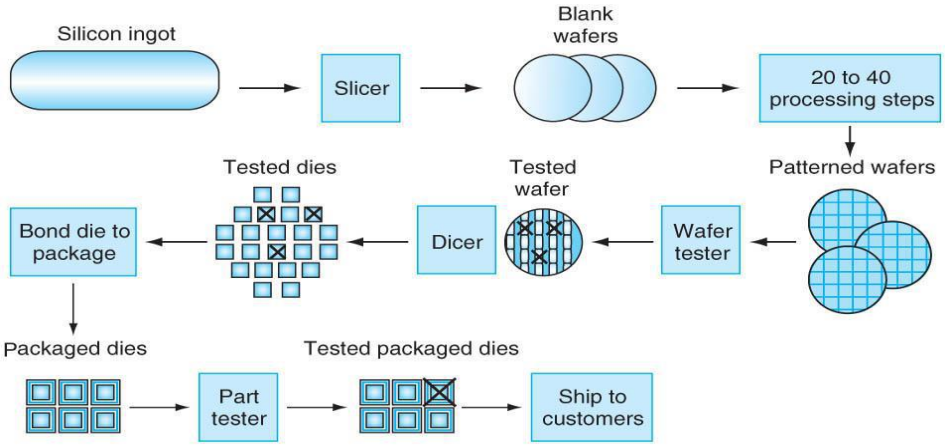


Fig 1.18 Processo de produção de um chip. Após fatiar um pedaço de silício, *wafers* "em branco" passam por 20 a 40 passos para criar *wafers* com os padrões desejados (mostrados no próximo slide). Estes *wafers* "gravados" são então testados com um testador de *wafers* e um mapeamento das partes boas 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.

Pastilha com várias CPUs

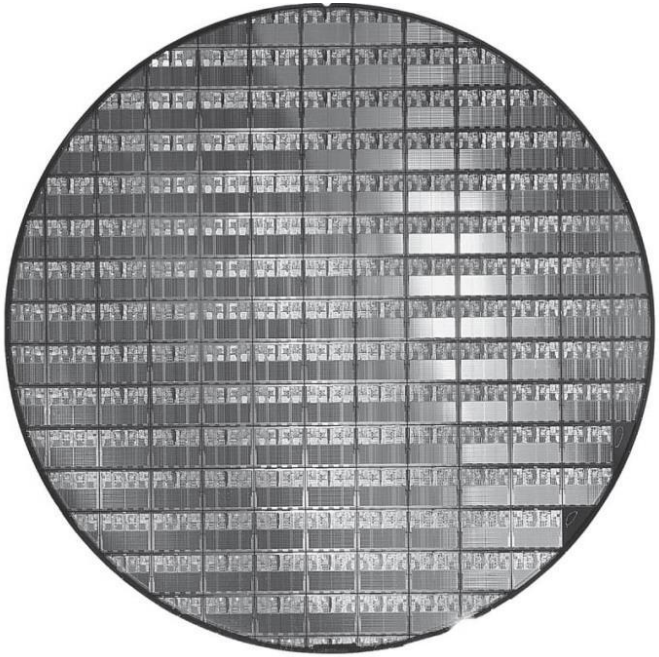
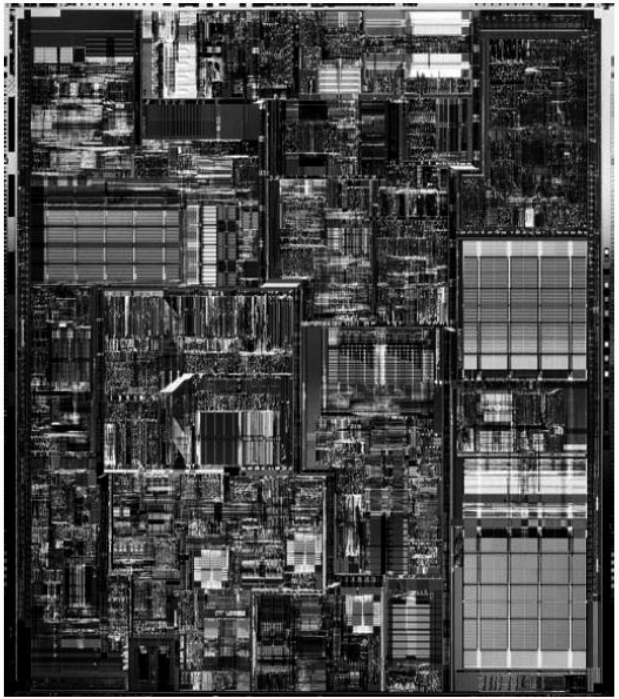
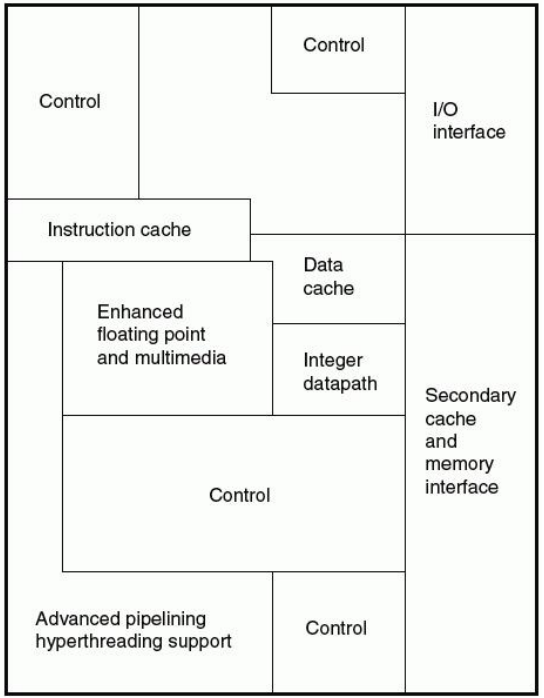


Fig 1.19 Um wafer de 300mm de chips AMD Opteron X2, o predecessor dos chips Opteron X4 (Cortesia AMD). O número de *dies* por wafer com taxa de 100% de sucesso é 117. Este *die* usa tecnologia de **90-nanometros**, o que significa que os menores transistores usados na fabricação tem tamanho aproximado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

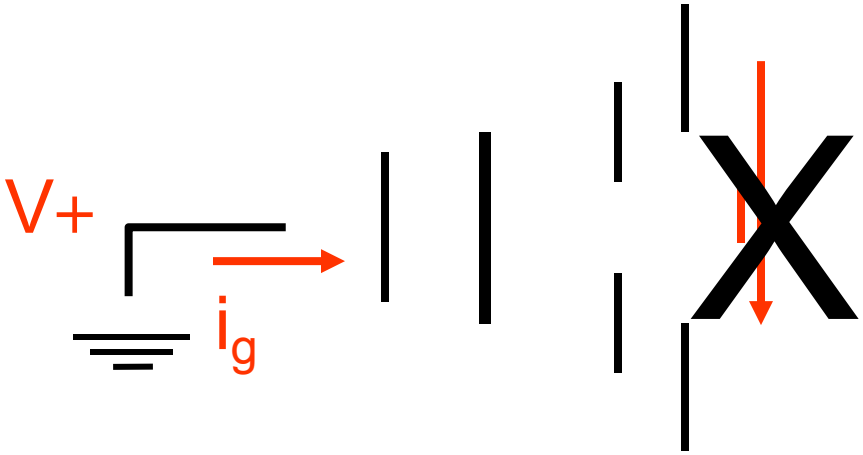
Hardware de uma CPU

Chip Pentium IV – Fonte: Cap1, Livro Texto





Transistor MOS: Base de Construção



Porta Lógica NOT ou Inversora

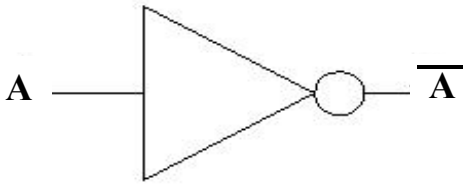
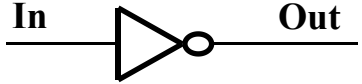
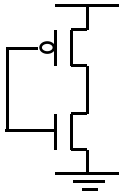
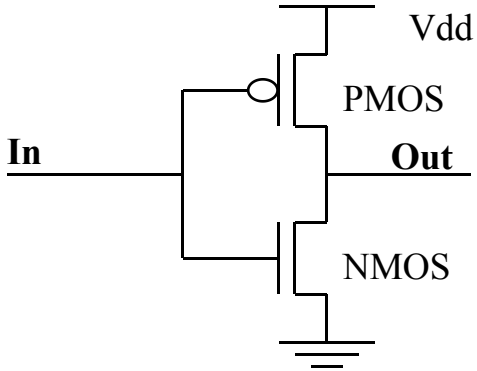


Tabela da Verdade

A	F = A’
0	1
1	0

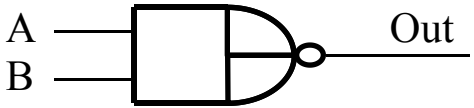




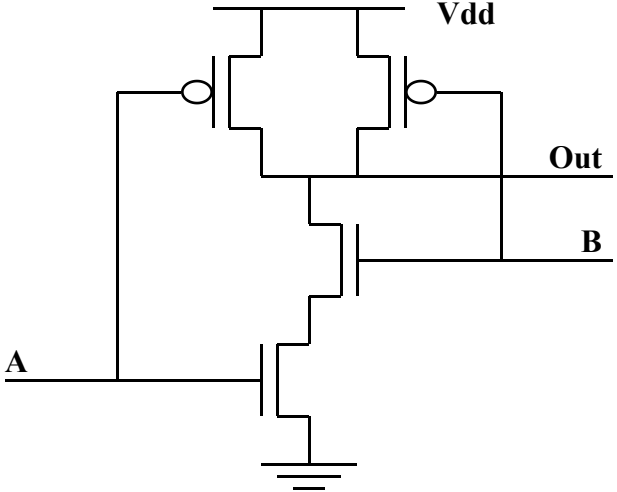


Portas CMOS

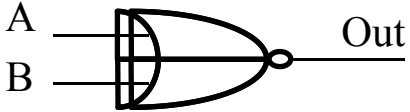
Porta NAND



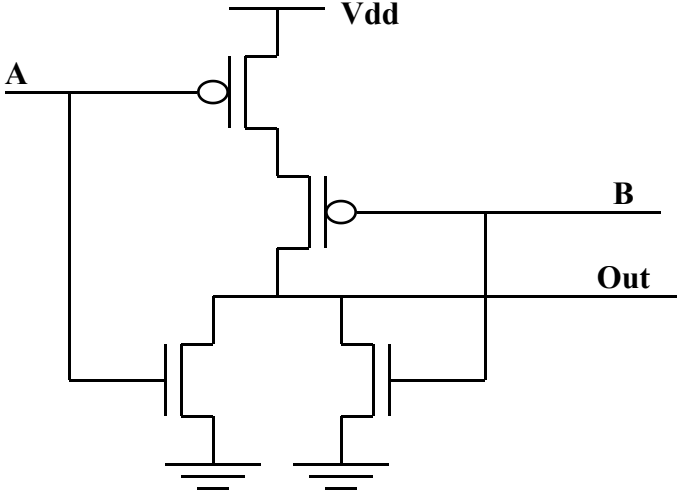
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0



Porta NOR



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

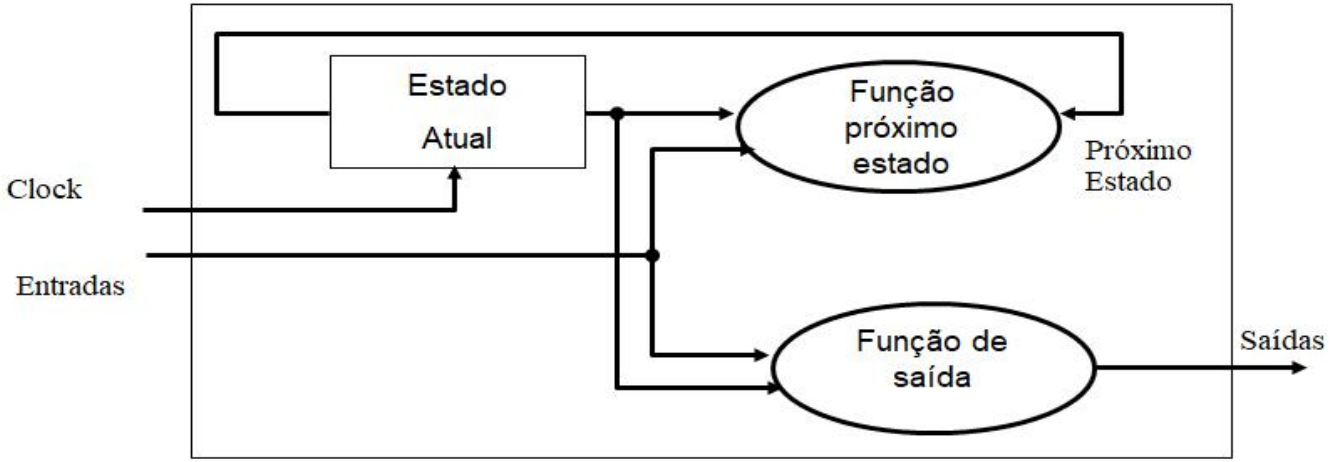


Circuitos Lógicos

- 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

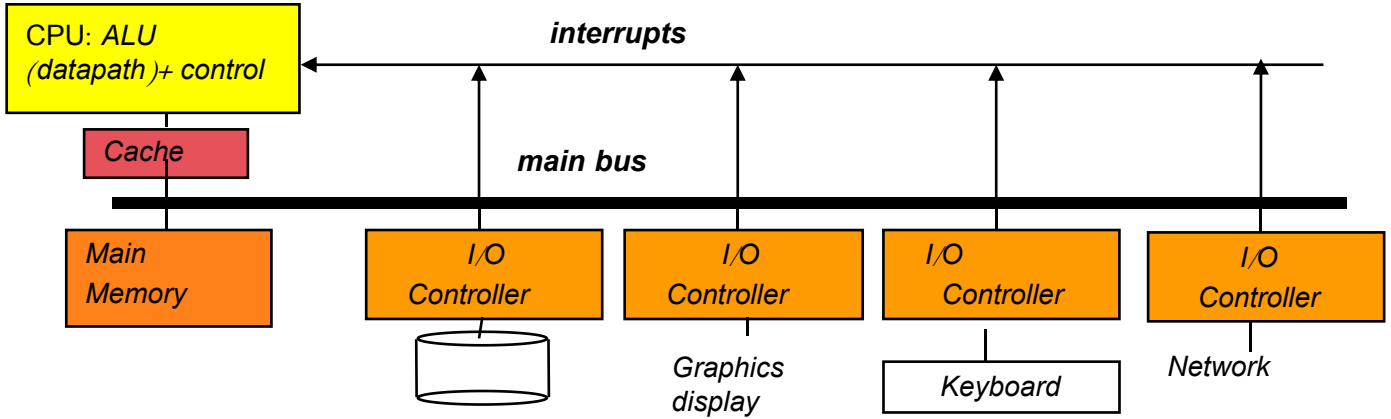
Maioria das máquinas atuais

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

- **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**

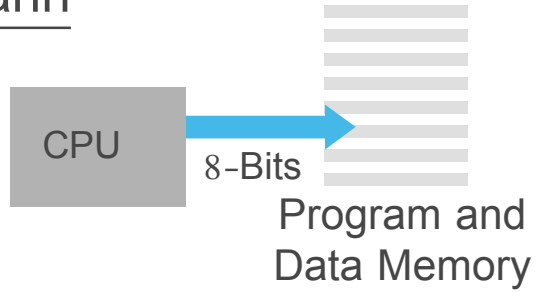
Arquitetura von Neumann

- Processador
- Memória principal
- Dispositivos de E/S
- Estruturas de interconexão



Von Neumann x Harvard

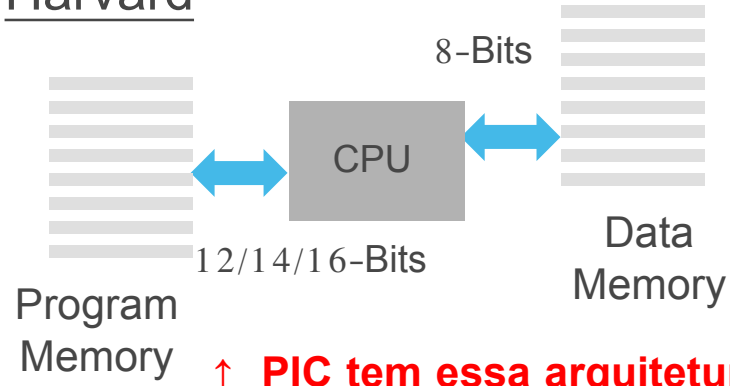
Von Neumann



Busca instruções e dados em uma memória única.

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

Harvard



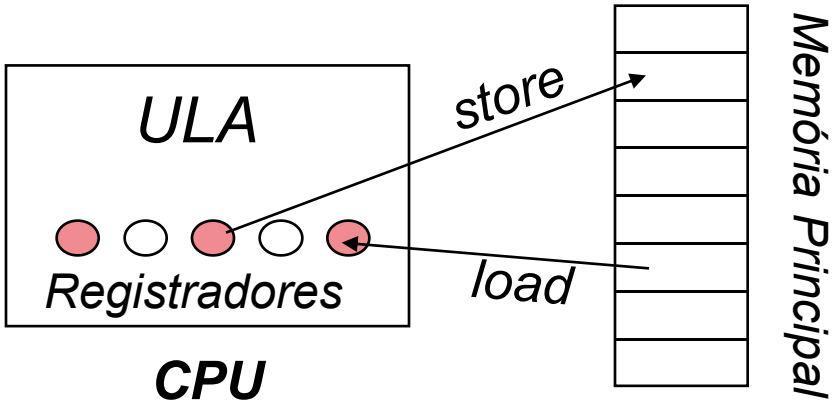
↑ **PIC tem essa arquitetura!**

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.

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.

HARDWARE: MAIS ALGUNS DETALHES...

Memória no MIPS (Patterson)

- No MIPS: vetor de palavras de memória
  - Largura da memória física: 1 Byte (8 bits)
  - cada **palavra** contém 32 bits (isto é, 4 **bytes** de 8 bits)
- Endereços de memória referem-se a **palavras** ao invés de **Bytes**
- Os endereços podem ser armazenados em palavras
- Cada posição endereçada conterá 32 bits
  - Em geral, bits não são acessados diretamente, mas em blocos ou conjuntos de bits
  - O tamanho dos blocos variam de acordo com os níveis de memória endereçados...
  - Manipulação posterior para acessar pedaços da palavra buscada

Representação binária

- Uma palavra 32 bits de memória de dados contém o seguinte:
  - Hexadecimal: 41424344
  - Binário: 0100 0001 0100 0010 0100 0011 0100 0100

O que representa?

Poderia ser uma string de caracteres codificados em ASCII (ANSI):

'A'=65 (0x41) 'B'=66 'C'=67, ...

'0'=48 '1'=49, ...

Hex Char	Hex Char	Hex Char	Hex Char
00 nul	20 sp	40 @	60 `
01 soh	21 !	41 A	61 a
02 stx	22 "	42 B	62 b
. . .	. . .	. . .	. . .
0A lf	2A *	4A J	6A j
. . .	. . .	. . .	. . .
1E rs	3E >	5E ^	7E ~
1F us	3F ?	5F _	7F del

Big ou little endian

- Depende da forma como o endereçamento aos bytes é organizado:
  - 'ABCD' se byte + significativo está no < endereço: ***little-endian***
  - 'DCBA' se byte + significativo está no ı endereço: ***big-endian***
- Algumas Arquiteturas:
  - MIPS – configura-se o hardware para *big* ou *little-endian*
  - SPARC – *big-endian* (compatibilidade com a linha 68K usada pela SUN)
  - IBM – *big-endian*
  - Intel 80x86 – *little-endian*

MIPS: *little-endian*

Armazenando os caracteres ‘Help‘ (pleH = `0x706c6548`) e o inteiro `32766` (`0x00007ffe`) na memória de dados

0x10000000	0x48 ("H")
0x10000001	0x65 ("e")
0x10000002	0x6c ("l")
0x10000003	0x70 ("p")
0x10000004	0xfe
0x10000005	0x7f
0x10000006	0x00
0x10000007	0x00

Conjunto de Instruções

- Conjunto de instruções = linguagem de máquina
- Formato da Instrução: *opcode* [*operandos*]
- Tipos de instrução:
  - Instruções aritméticas e lógicas (add, sub, AND, OR)
  - Referência à memória ou *Load&store*
  - Desvios Condicionais (*if*) e Incondicionais (*goto* / *jumps*)
- Conjunto de instruções complexo: **CISC**
  - Lógica de controle complexa, microprogramas
- Conjunto de instruções reduzido: **RISC**
  - Lógica de controle mais simples, muitos registradores, endereçamentos simples