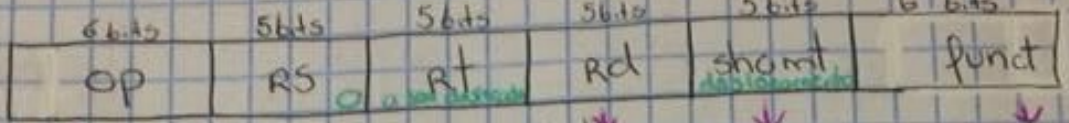
MIPS-32 (64 bits)
- REG-REG
- todas as instruções têm 32 bits
- endereços válidos 2^{30} bytes (2^{30} words)
- RISC → conjunto de instruções Reduzido e Simplificado
-

MIPS

Formato R → operações com 3 operandos (incluindo comparações) e deslocamentos



destino

op → operação codificada

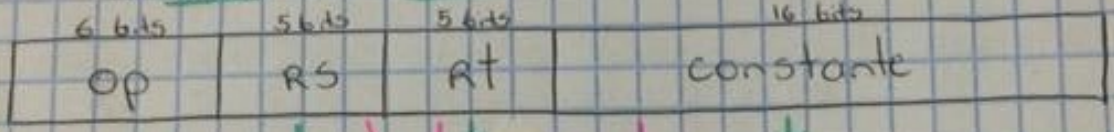
3 operandos
Δ(Rd ← RS Rt)

shamt → só é ≠0 para instrução de deslocamento

funct → distingue operações com mesmo op

Nota: op=0, funct=8, rt=rd=0 → executa [jR] → salta para a posição de endereço RS

Formato I
↳ operações com constantes imediatas → saltos condicionais
↳ leitura e escrita em memória



endereço base da memória

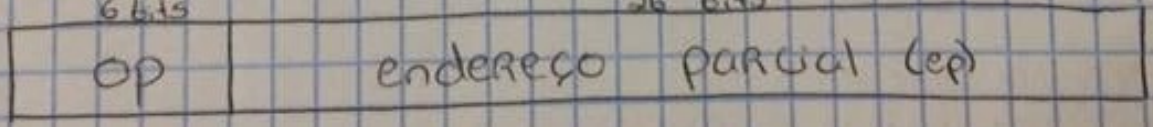
destino

valores a comparar

valor a somar ao endereço base

número de instruções a saltar, a partir da primeira → vai-se para a posição: PC+4+endereço

Formato J → jump



salta para ____ ep 00

4 bits mais significativos do PC

como todos os endereços são multiplos de 4 (por só se usarem palavras)

Binário

Obter o número em binário

inteiro

13 | 2
1 6 | 2
0 3 | 2
1 1 | 2
1 0

1101_2

dividir sucessivamente por 2 até o quociente ser 0

decimal

0,75 × 2 = 1,50
0,50 × 2 = 1,0

0.11_2

multiplicar sucessivamente por 2 até a parte decimal ser zero

ou

decompor em potências de base 2

Sistema Hexadecimal

- 1º converter em binário
- 2º converter em hexadecimal

0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

exemplo:

$10 = 2^4 + 2^3 = 11000_2 = 18_H$

$20B_H = \underbrace{0010}_{2}\ \underbrace{1101}_{D}\ \underbrace{1100}_{B} = 2^9 + 2^3 + 2^6 + 2^5 + 2^3 + 2^2 = 512 + 128 + 64 + 16 + 8 + 4 = 732_{10}$

Soma

```
  1 1
  1101
+ 0100
------
 10001
```

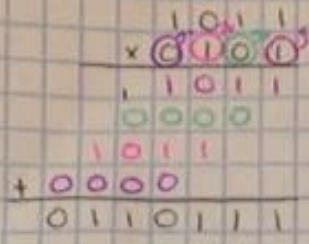
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 0 e vai 1

Subtracção

```
  1101
 -1 -1
- 0110
------
  0111
```

0 - 0 = 0
0 - 1 = 1 e vai 1
1 - 0 = 1
1 - 1 = 0

Multiplicação



caso particular

- multiplica por 2^n_{10}
  - ↳ acrescenta-se n zeros à direita

$(1 \times 1000)_2 = 11_2 \times 2_{10} = 11000_2$

Divisão - caso particular

dividir por 2^n_{10} → retira-se n zeros à direita

$(1101000 \div 100)_2 = 1101000_2 \div 2^2_{10} = 11010_2$ //

Sinal e Grandeza (SG)

N bits → 1 bit de sinal + N-1 bits pl grandeza
(0 ⇒ + 1 ⇒ -)

gama de representação: $[-(2^{N-1}-1) ; 2^{N-1}-1]$

↓
Nota: são contados 2 zeros (+0 e -0)

soma em SG

o sinal é igual?

sim:
- mantem o sinal
- soma grandezas
- verifica se ocorreu overflow

não:
- determina o operando de maior grandeza
- o sinal é o do operando de maior grandeza
- o grandeza é igual a |grandeza maior| - |grandeza menor|

subtração → transforma-se a subtração numa soma

Complemento para 2 (C2)

passar para C2:
- nº positivo → mantem-se
- nº negativo →
  - copiamos da direita para a esquerda até encontrar o 1º um (1)
  - copiamos o 1º 1 (um)
  - trocamos os restantes dígitos

exemplo: $-00011 \rightarrow 11101$

Nota:
- nºs positivos → começam sempre por zero (0)
- nºs negativos → começam sempre por um (1)

gama de representação: $[-2^{N-1} ; 2^{N-1}-1]$

↳ mais um número que em SG

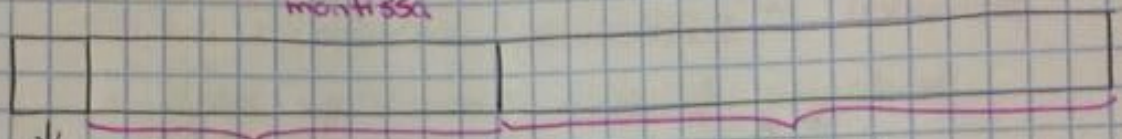
Soma
- Soma normal
- Se:
  - os operandos tem o mesmo sinal e o resultado sinal diferente → ocorreu overflow → não está correto
  - se não → está correto

Subtração
- subtração normal
- Se:
  - os operandos tem sinal diferente e o resultado não tiver o sinal do operando de maior grandeza → ocorreu overflow
  - se não → está correto

Vírgula Flutuante → norma IEEE 754

$$1,11101_2 \times 2^{exp}{}_{10}$$

(1,11101 = significando; mantissa)



sinal (0 → +, 1 → −) → 1 bit

expoente codificado = 127 + exp → 8 bits

significando → 32 bits

expoentes normalizados: [1 ; 254] (codificado) ou [−126 ; 127] (normal)

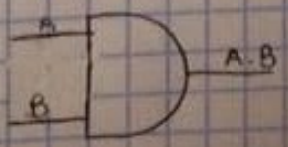
exp cod = 0 → número subnormal
↳ ex: 0 → exp cod e significando = 0

exp cod = 255
- significando = 0 → +∞ ou −∞
- significando ≠ 0 → not a number (NaN)

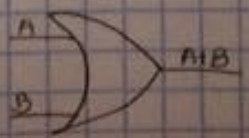
Circuitos Combinatórios
(não tem memória)

Portas Lógicas

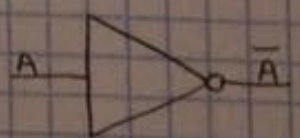
AND



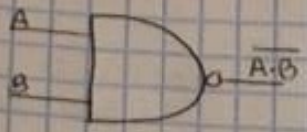
OR



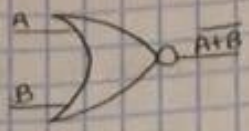
NOT → INVERSOR



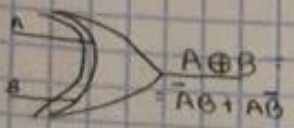
NAND



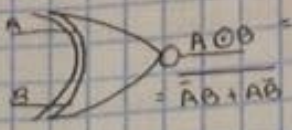
NOR



XOR → ou exclusivo



XNOR



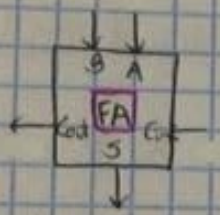
Álgebra de Boole

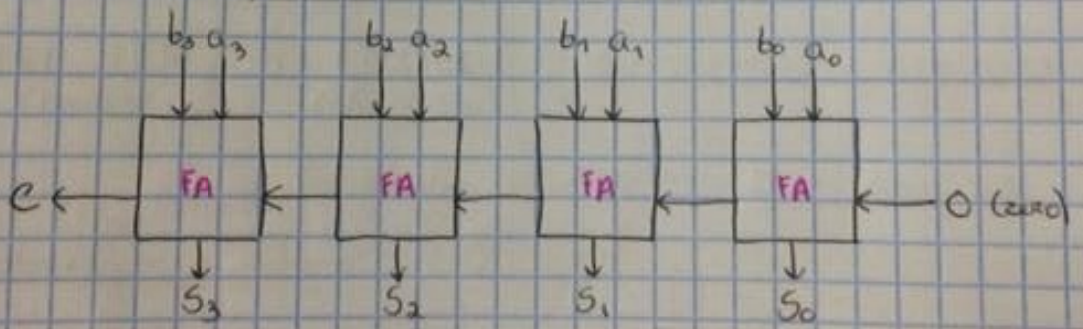
$\underline{0}$ → Falso
$\underline{1}$ → Verdadeiro
$\underline{+}$ → ou ($\cup$)
$\underline{\cdot}$ → e ($\cap$)
$\underline{\overline{A}}$ → negação ($\neg$ ou $\sim$)

Nota: se, numa igualdade, substituirmos os 1's, os 0's, os +'s e os ·'s, continuaremos com uma igualdade verdadeira.

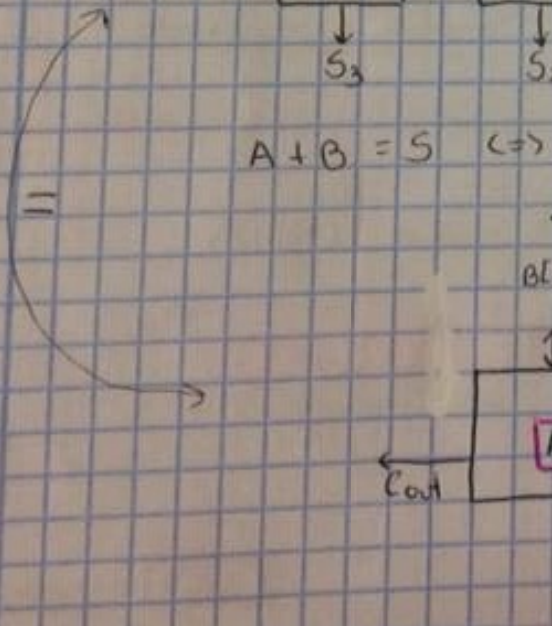
Somador → Ripple-carry

tendo um full-adder → módulo que calcula a soma de 3 bits

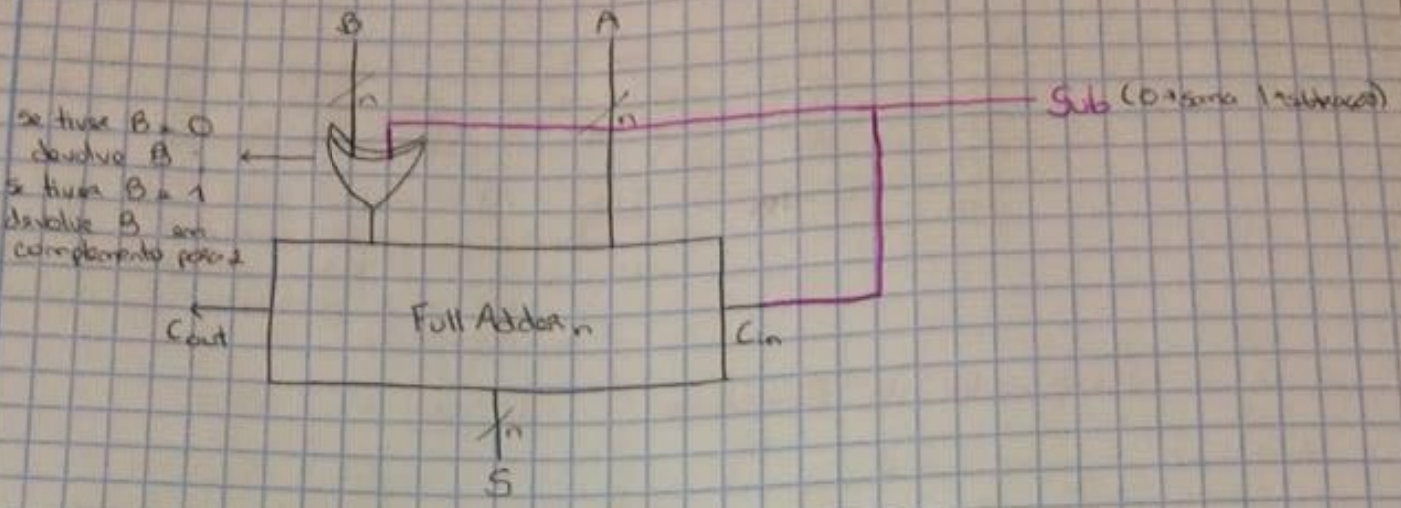




$A + B = S \iff \underbrace{a_3 a_2 a_1 a_0}_{4^{\text{o}}\text{ dígito}} + b_3 b_2 b_1 b_0 = S_3 S_2 S_1 S_0$ (1º dígito)

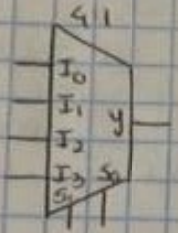


Somador e subtrator



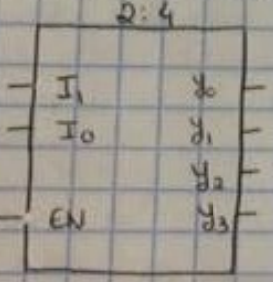
Multiplexer (MUX) → multiplexador $\boxed{2^N : 1}$

↳ seleciona uma entrada entre 2^N
↳ tem n entradas de controlo



→ a saída corresponde ao valor na posição $S_n \dots S_1 S_0$

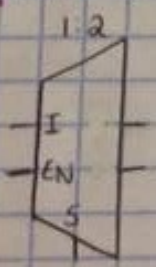
Decoder → descodificador $\boxed{N : 2^N}$



→ põe todas as saídas a zero, exceto a saída cujo índice é $I_n \dots I_1 I_0$

Enable: • 0 → põe todas as saídas a 0
• 1 → o circuito funciona normalmente

Demultiplier (DEMUX) → desmultiplexador $\boxed{1 : 2^N}$



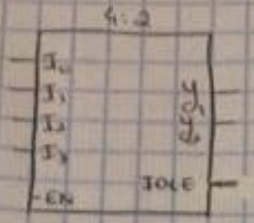
↓
tem N entradas de controlo

→ a saída de índice $S_N \dots S_0$ fica com o valor de I; as outras saídas ficam a 0.

EN	S_0	I	y_1	y_0	
0	X	X	0	0	
1	0	0	0	0	S_0=0 logo y_0 tem o valor de I
1	0	1	0	1	
1	1	0	0	0	S_0=1 logo y_1 tem o valor de I
1	1	1	1	0	

Encoder → codificador

$2^N : N$

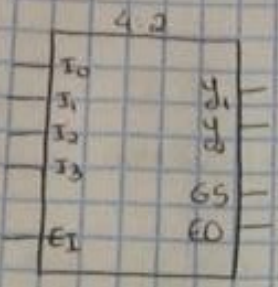


→ y_N $y_1 y_0$ corresponde ao número em binário do índice da única entrada a 1. → oposto do decoder

IOLE: • 1 → enable está a 0 ou os entradas não estão corretas (não está nenhuma a 1 e todas as outras a 0)
• 0 → o programa funcionou corretamente

Codificador de Prioridade

$2^N : N$



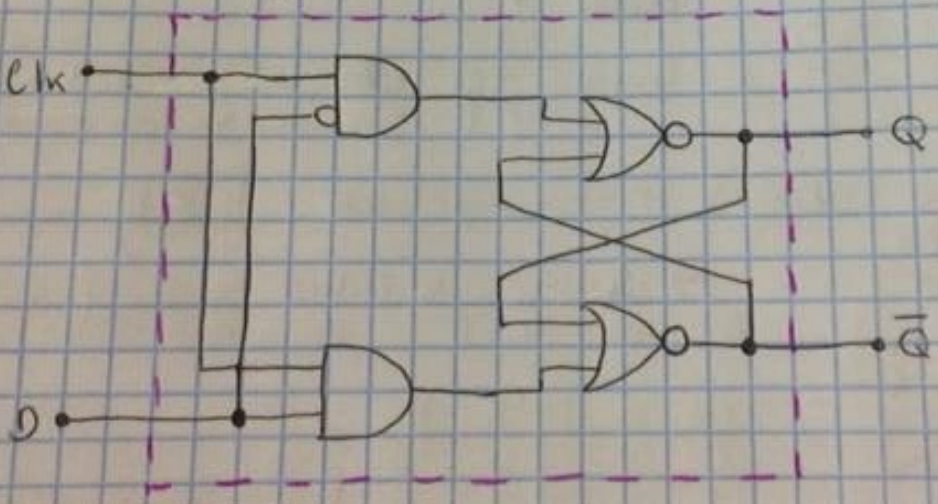
→ y_N $y_1 y_0$ corresponde ao número em binário do índice da 1ª entrada a 1 (3 tem prioridade sobre 2, etc)

EI (enable input) → circuito habilitado
EO (enable output) → • 1 → EI: 1 e GS: 0
• 0 → todas as outras situações
GS (got something) → • 1 → EI: 1 e alguma entrada está a 1 (programa funcionou corretamente)
↳ negação do IOLE

Circuitos Sequenciais

(tem memória → a saída depende das entradas atuais e anteriores)

Trinco tipo D

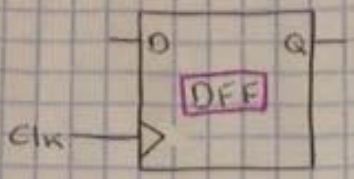


Clk = 1 → modo transparente → $Q = D$ e $\overline{Q} = \overline{D}$

Clk = 0 → modo de retenção → $Q = Q$ anterior
$\overline{Q} = \overline{Q}$ anterior

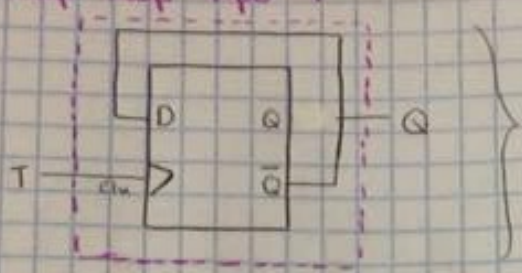
→ as saídas só "atualizacam" quando clk = 1

Flip-Flop tipo D → "edge-triggered flip-flop"



O valor de Q só é "atualizado" quando há um flanco ativo ↑ (Clk muda de 0 para 1). Do resto, Q tem sempre o valor anterior de Q

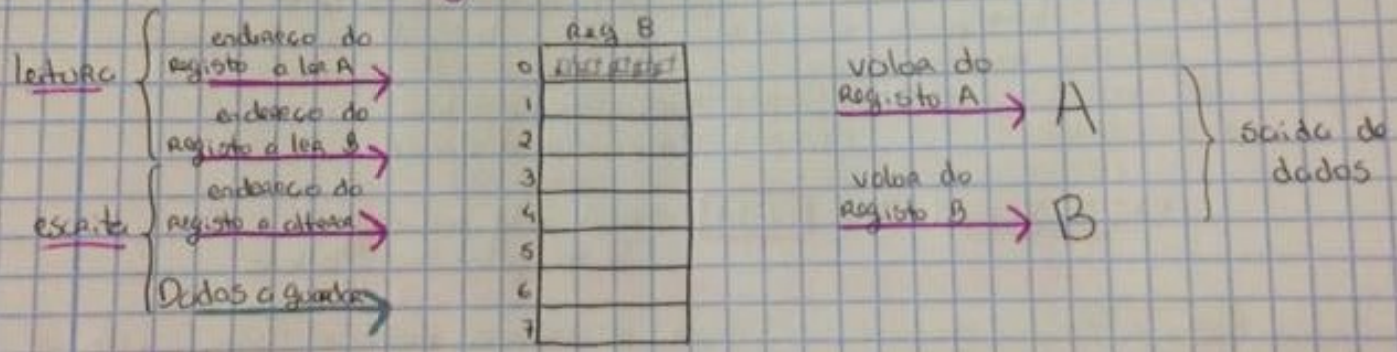
Flip-Flop tipo T



O valor da saída Q muda a cada ciclo de relógio (a cada flanco ativo)

Registo → conjunto de n elementos de memória

Banco de Registos → conjunto de Registos



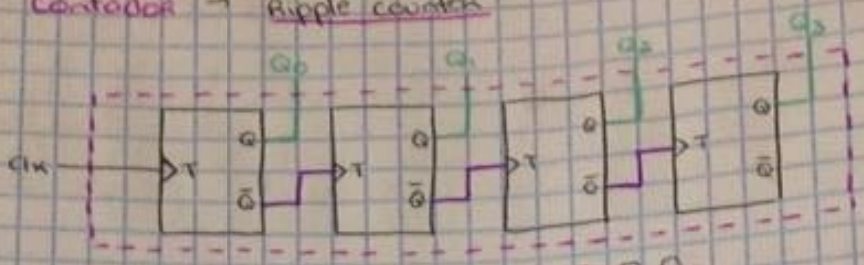
Endereços de leitura ⇒ entradas de controlo de multiplexadores (onde as entradas são os Registos)

Endereços de escrita ⇒ entradas de descodificadores cujas saídas ligam a enables de Registos, sendo que os dados estão ligados a todos os Registos (o descodificador garante que só um Registo atualiza com os novos dados)

Banco de Registos → estão nos processadores

2^P posições no banco de Registos → P endereços

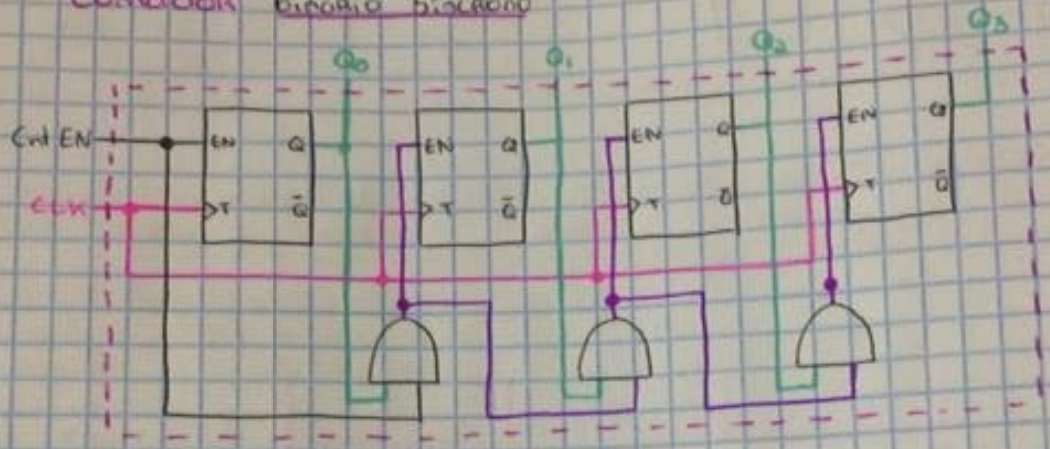
Contador → Ripple counter



saída: $Q_3 Q_2 Q_1 Q_0$

problema → muito lento

Contador binário síncrono



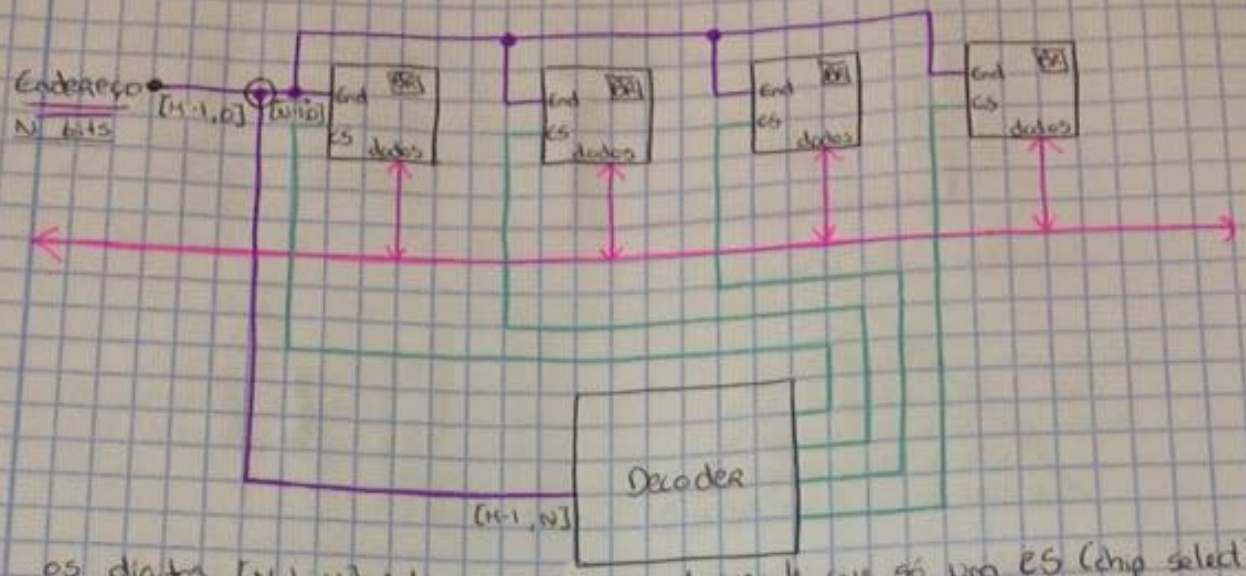
Tipos de Memória

- Volátil ⇒ RAM → random access memory
  - ↳ permitem leitura e escrita
  - ↳ perde os dados quando é desligada a alimentação
    - SRAM → memória estática
    - DRAM → memória dinâmica
- Não Volátil ⇒ ROM → read only memory
  - ↳ só permitem leitura
  - ↳ onde se indica memória flash, CD's, etc.

Estática	≠	Dinâmica
• utiliza flip-flops		• utiliza condensadores
↑ • + rápida		• - rápida ↓
↓ • - capacidade (- dados)		• + capacidade (+ dados) ↑
↓ • + consumo energético		• - consumo energético ↑
↓ • - económica (+ cara)		• + económica ↑

↑ + utilizada

Organização da Memória



os dígitos [M-1, N] → dizem qual o banco de Registos

↳ garante que só um ES (chip select) está ativado, ou seja, só um dos bancos está a Receber/transmitir dados

os dígitos [N-1, 0] → dizem qual a posição no banco de registos, ou seja, o registo

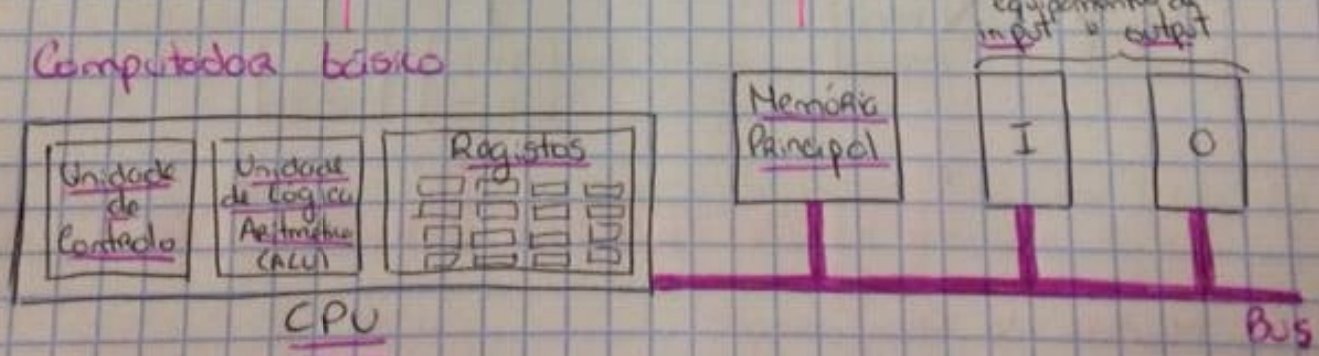
espaço de endereçamento = $2^{nº\ posições}$ x nº bits

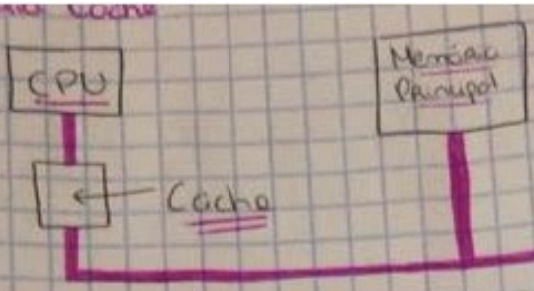
Descodificação:

- total: a 1 endereço corresponde 1 posição
  todos os bits do endereço são utilizados
- parcial: a n endereços corresponde a mesma posição
  a gama de representação é > que o espaço de endereçamento
  (os valores vão estar repetidos n vezes)

Arquitetura de Computadores

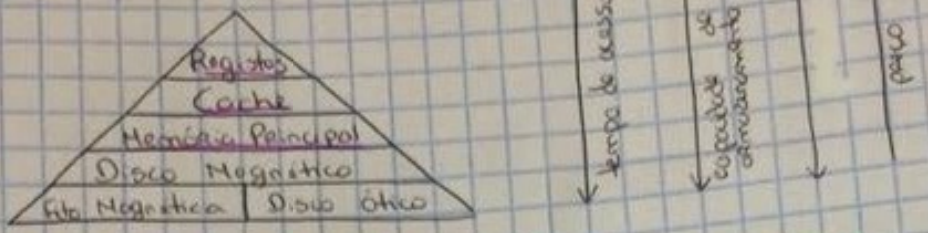
Computador básico



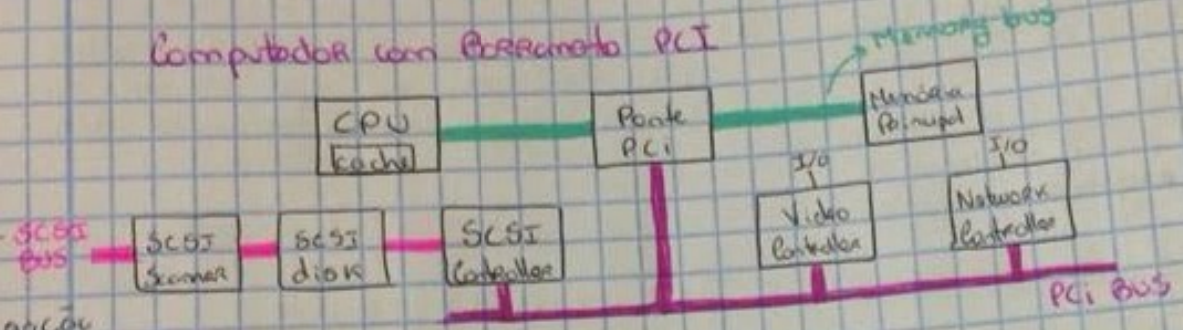


Cache → memória rápida e pequena responsável por armazenar transparentemente e temporariamente instruções e dados muito usados

Hierarquia da Memória

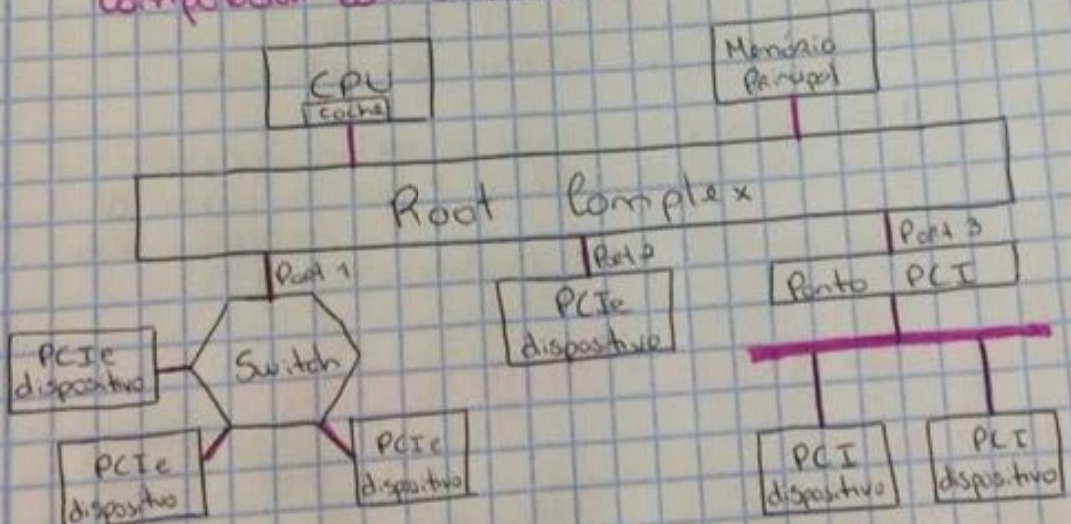


Computador com Barramento PCI



gação
e alto desempenho

Computador com Barramento PCIe



Programação de Processadores

Modelo de Programação

- modelo de execução
- conjunto de instruções:
  - tipos de instruções
  - modos de endereçamento
- Registos
  - de uso geral

Modelo de Execução

1 - iniciar PC = program counter
2 - obter instrução da posição PC da memória
3 - executar instrução e atualizar PC
4 - repetir a partir de 2

Tipos de instruções

- operações aritméticas (+, -, ×, ÷)
- operações lógicas (AND, OR, NOR, deslocamentos [x2^n])
- transferências de dados (leitura e escrita na memória princ.)
- alteração do fluxo sequencial de execução:
  - saltos condicionais e comparações
  - saltos incondicionais
  - subrotinas

Modos de Endereçamento

R2 ← MEM[R1] + MEM[30] + 20

- R2 → **Registo** - o valor está num registo
- MEM[R1] → **indireto** (via registo) - o valor está na posição R1 (registo) de memória
- MEM[30] → **direto** - o valor está na posição 30 da memória
- 20 → **imediato** - o valor é a constante

↳ caso particular: **indireto com deslocamento constante**
escrever: 30 MEM[R1] → corresponde a: MEM[R1+30]

Relativo ao PC - index constante + somar a PC

Tipos de Arquiteturas

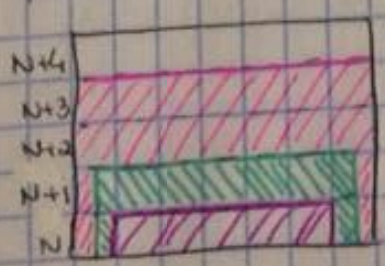
- Reg-Reg → os 3 (máx.) operandos estão nos registos
  os únicos operações com mem são load e store
- Reg-mem → operandos estão nos registos e na memória
- mem-mem → os 3 operandos estão na memória

Tipos de Operandos:

Inteiros:
- palavra / word - 4 bytes (32 bits)
- meia palavra / half-word - 2 bytes (16 bits)
- 1 byte (8 bits)

Vírgula Flutuante
- 4 bytes → precisão simples
- 8 bytes → " dupla

Nota: como cada posição de memória só tem 1 byte de espaço, o endereço é a posição do primeiro byte



← endereço em todos os casos