Aula 1
2021-02-26

Introdução

Arquitetura de Computadores

Recuando cerca de 100 anos...

Computar = contar, avaliar, somar → 1º tipo de aplicação – cálculos matemáticos para fins militares

Modelo do computador ? → Homem: resolução de problemas (ex:cálculos matemáticos)

<u>Resolução de problemas</u> (ex. um calculo matemático, 98*7=?)

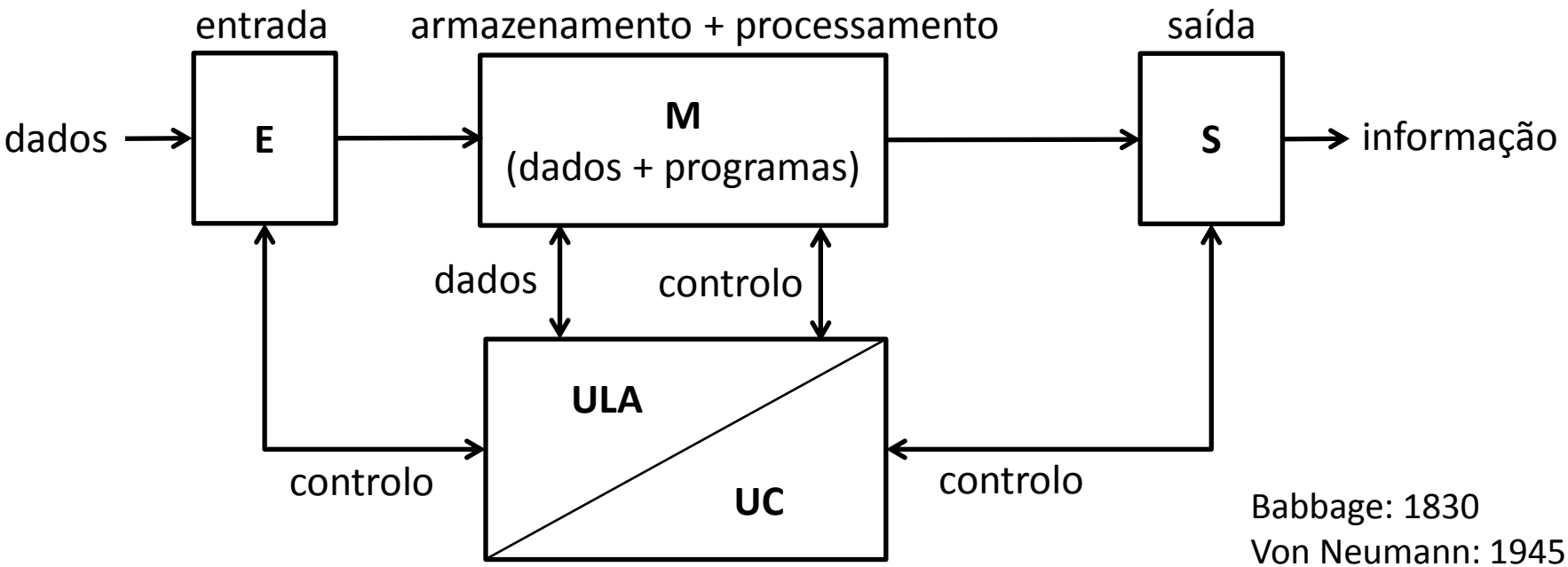
1) aquisição de dados = sentidos → unidade de entrada (E)

2) armazenamento (dados/informação) = memória → unidade de memória (M)

3) processamento = aplicação de regras e de processos operatórios
   → unidade lógica e aritmética (ULA) : regras
   → unidade de controlo (UC) : sequência das acções

4) comunicação de resultados = fala, escrita → unidade de saída (S)

Arquitetura de Computadores



Babbage: 1830
Von Neumann: 1945

https://www.youtube.com/watch?v=35MwtZ5MKjM

E : teclado (interruptores) , rato , scanner, microfone , ...

M: cartões perfurados , sistemas magnéticos , circuitos digitais(Flip-Flops) , condensadores , ...

ULA(Unidade Lógica e Aritmética): rodas dentadas , circuitos digitais (somadores/mux-demux), ...

UC(Unidade de Controlo): rodas dentadas , cartões perfurados , circuitos digitais (tabelas de verdade), ...

S: cartões perfurados , impressoras , ecrans vídeo , ...

Cartões perfurados
https://www.youtube.com/watch?v=kTIeJyEYF7A
https://www.youtube.com/watch?v=J0hDuCa_KlY

Rodas dentadas
https://www.youtube.com/watch?v=35MwtZ5MKjM

Arquitetura de Computadores

na prática de que é feito tudo isto?

lixo + areia da praia + materiais duvidosos...

...além de muita água!

Arquitetura de Computadores

Em grande parte os computadores são feitos de materiais plásticos...
- os plásticos são originados a partir de resinas derivadas do petróleo...
- o petróleo resulta da decomposição de restos orgânicos de animais e de vegetais... → **lixo!**

Principal matéria prima dos processadores... **areia da praia!** Quem o diz é a Intel!

From Sand to Silicon: the Making of a Chip
https://www.youtube.com/watch?v=Q5paWn7bFg4

Materiais duvidosos: borracha, ouro, prata, paládio, cobre, estanho, gálio, índio, boro, rutênio, alumínio, titânio, silício, germânio... podem usar até cerca de 40 elementos da tabela periódica
→ a maior parte deles venenosos e perigosos para o ambiente!

E **água**...muita água! Cerca de 1500Kg por PC...

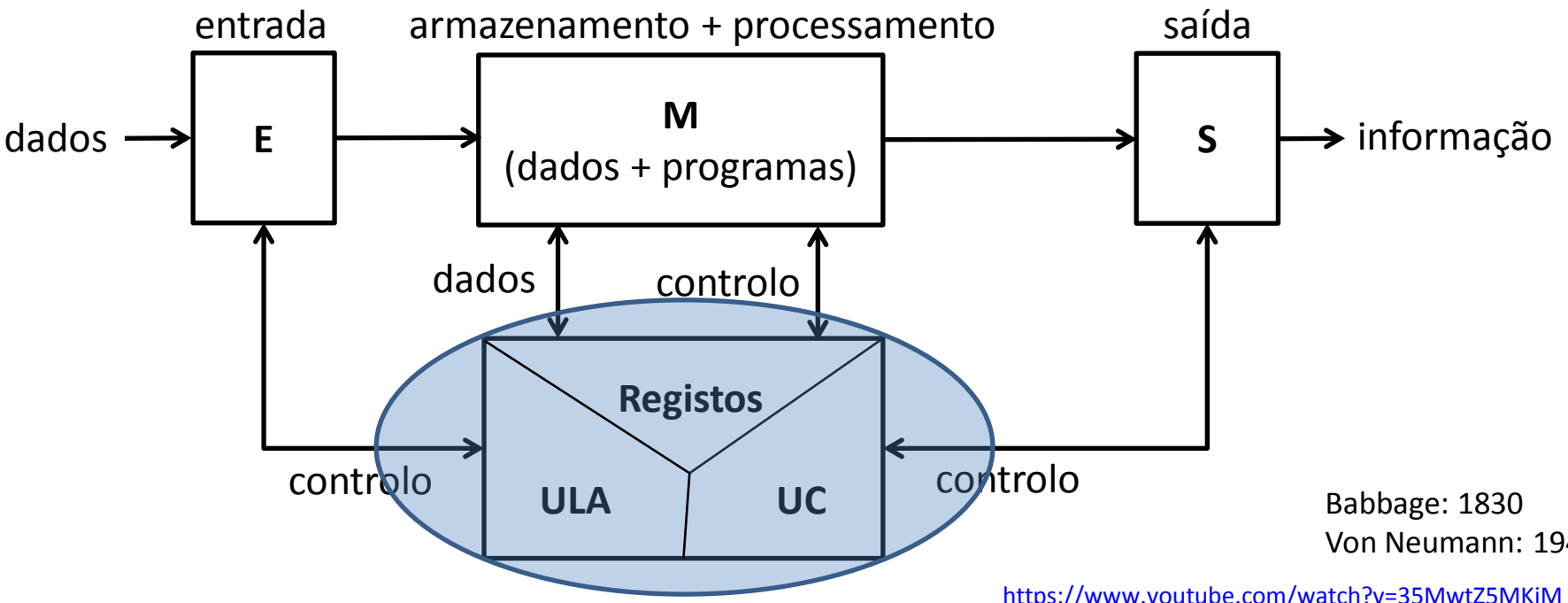
História dos computadores: Htimeline , Historia , The Virtual Museum of Computing (VMoC)

Aula 2
2021-03-05

CPU – Central Processing Unit
Memória

Execução das instruções

Modelo do computador digital



Babbage: 1830
Von Neumann: 1945

https://www.youtube.com/watch?v=35MwtZ5MKjM

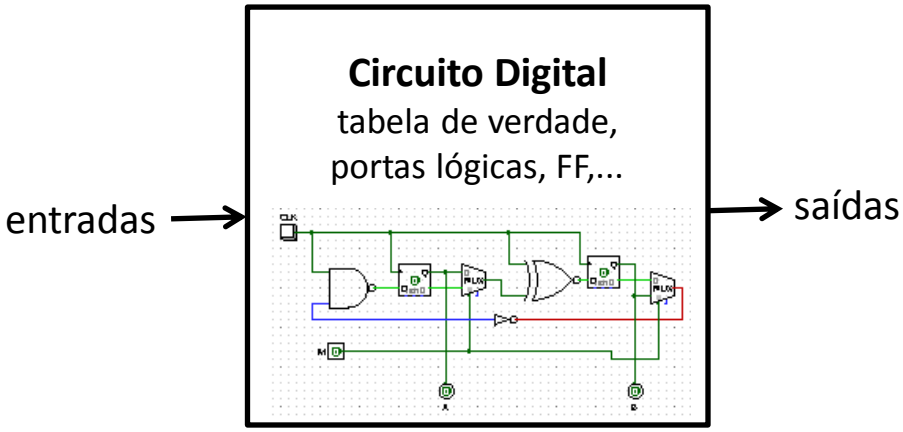
Processador

CPU – Central Processing Unit

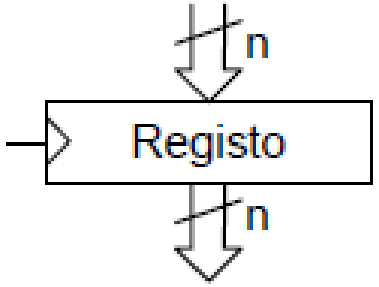
Processador(CPU) : responsável pela execução das instruções e pelo controlo dos restantes dispositivos

- ULA(Unidade Lógica e Aritmética): responsável pelas instruções lógicas (“A>=B ?” , “X=0 ?”) e pelas instruções aritméticas (“+” , ”-” , ”*” , ”/”)
- UC(Unidade de Controlo): responsável pelo controlo do próprio processador e de outros dispositivos
- Registos – zona de memória para dados, resultados de operações e códigos de funções

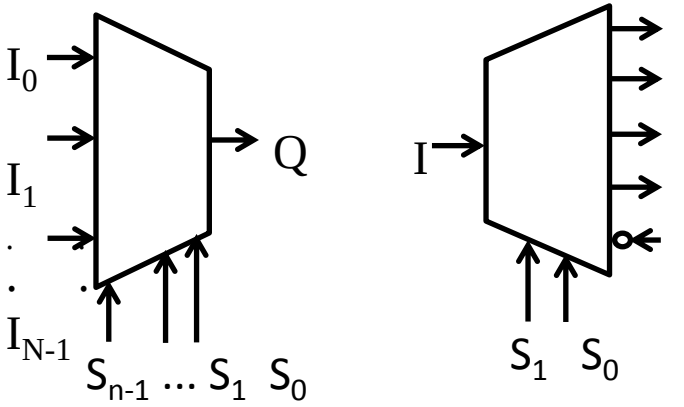
Processador : componentes básicos



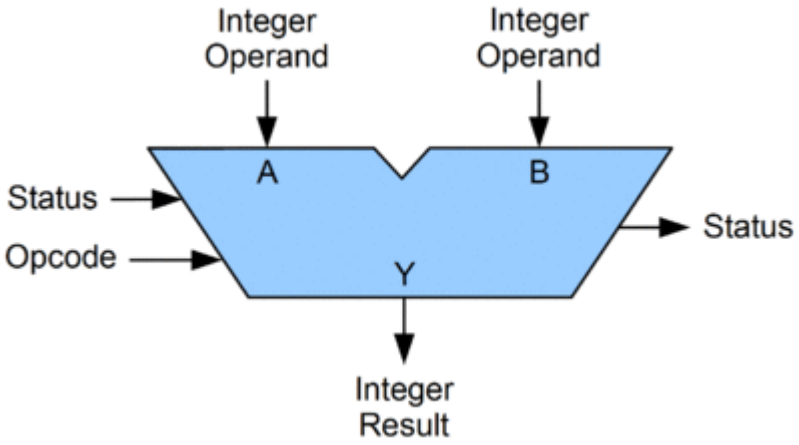
realizam funções lógicas (ex: controlador do LPB)



armazenam conjuntos de bits (ex: 8bit=1byte)



multiplex/demultiplex: encaminham bits (dados)

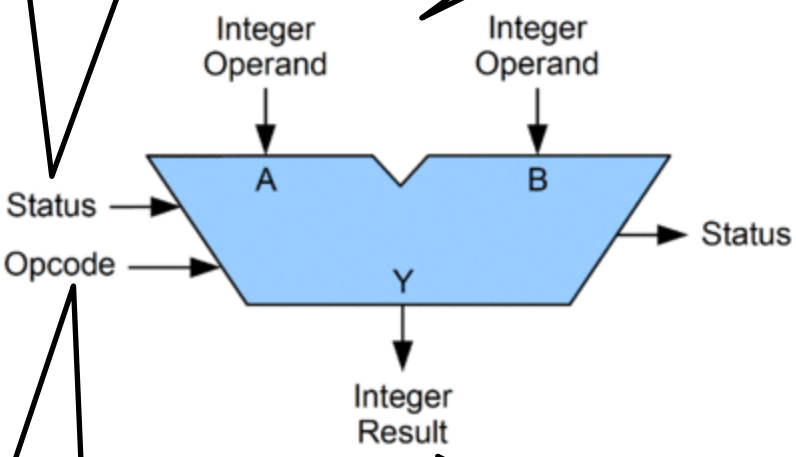


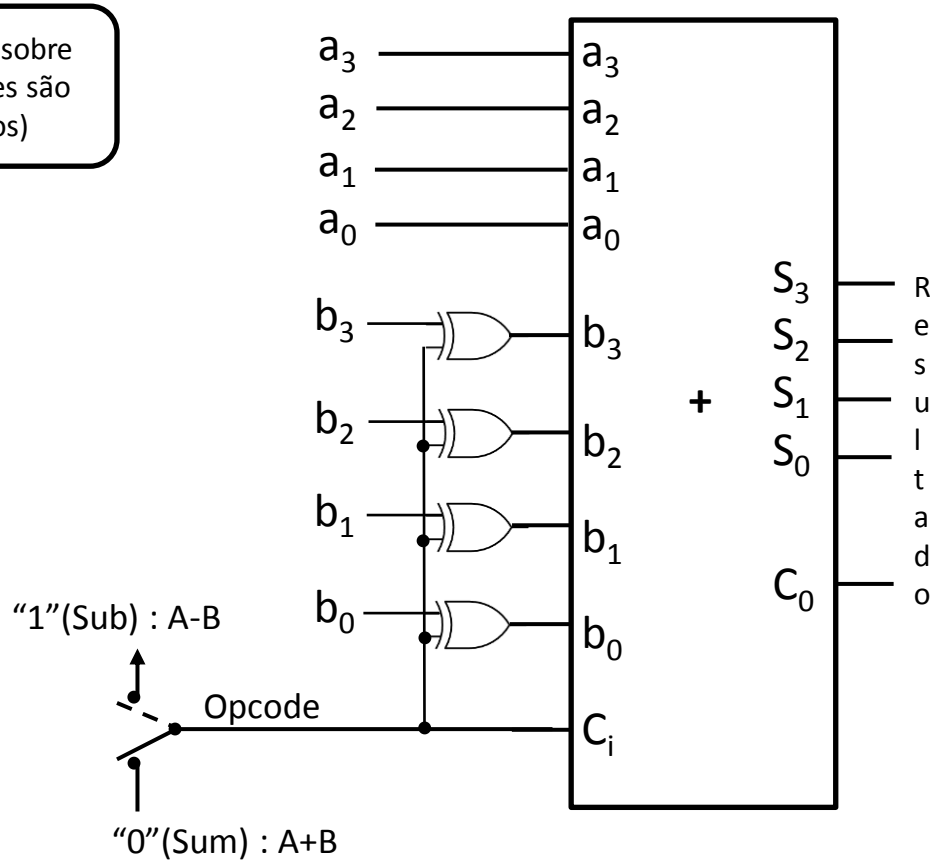
ALU: realizam operações lógicas/aritméticas (ex. somador / subtrator)

ALU : exemplo de operação aritmética → adição / subtração

Status : indicam a existência de determinadas condições, por exemplo se um valor é negativo ou zero, ou se houve um erro (ex:divisão por zero)

Operandos : valores sobre os quais as operações são executadas (dados)



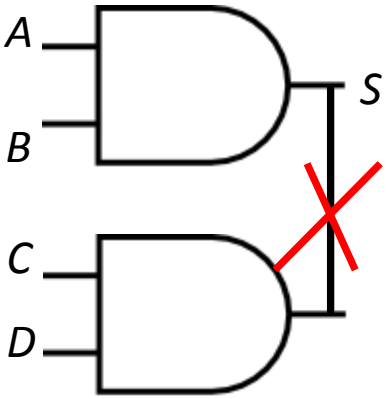


Opcode(operation code) : determina a operação a executar; corresponde às instruções de um programa.

Resultados : valores obtidos após a execução das operações – podem servir como dados para outras operações

Y = operação(A,B)

Z : o terceiro estado digital

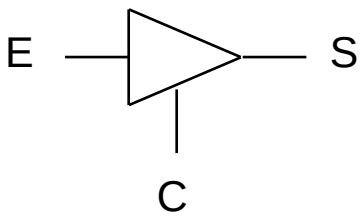


Qual o valor de S para diferentes situações?

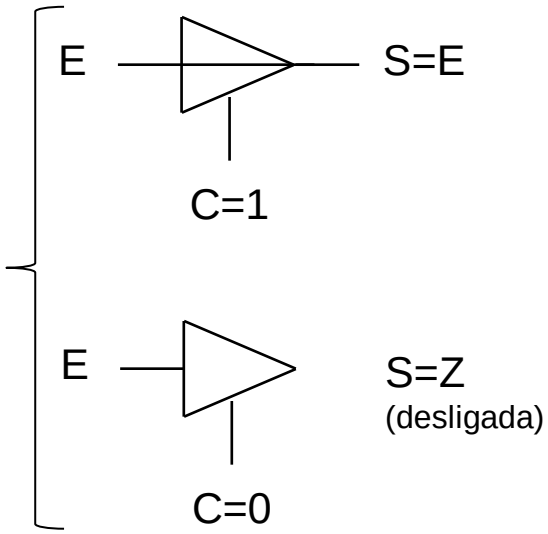
A	B	C	D	S
0	0	0	0	0
1	0	0	1	0
1	1	1	1	1
0	1	1	1	? (conflito)
1	1	0	0	? (conflito)

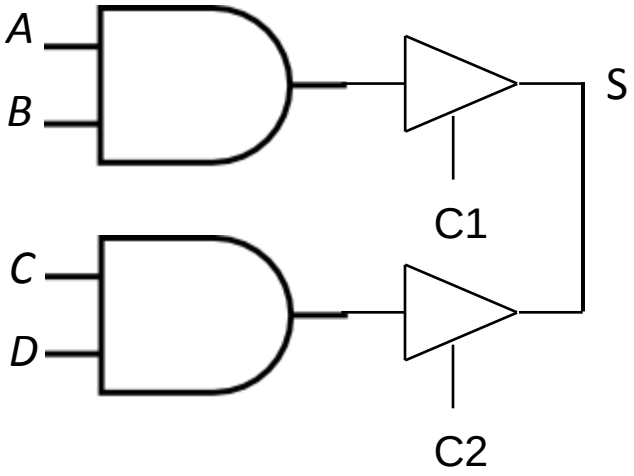
Porta *tri-state*: cria um terceiro estado, designado por Z (alta-impedância), que não é 0 nem 1 (corresponde ao nada, é como se a saída ficasse desligada)

ex: buffer tri-state



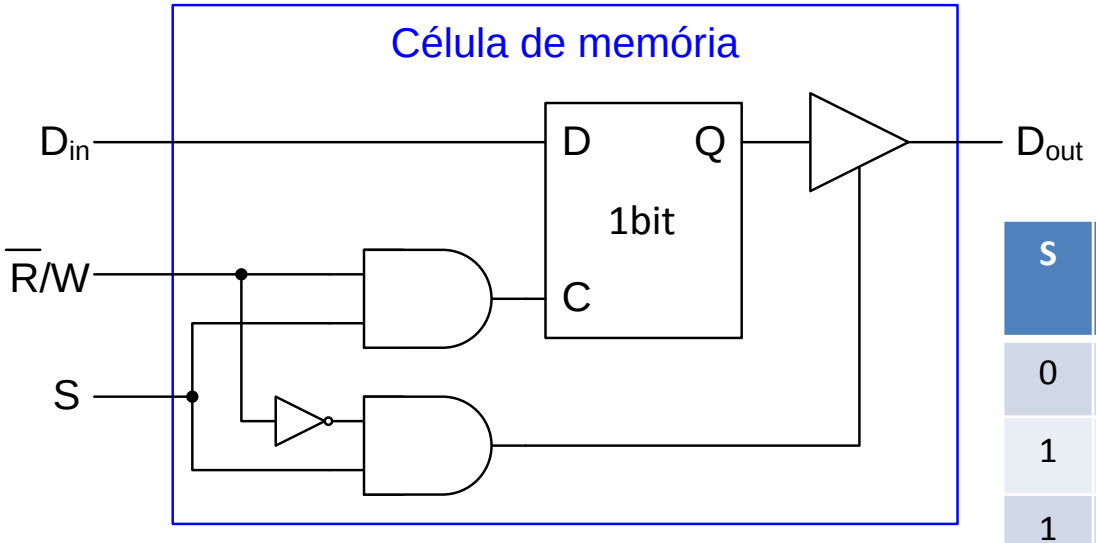
C	E	S
1	0	0
1	1	1
0	-	Z





Desde que C1 e C2 não estejam ambos ativos (1) não haverá conflito

Célula básica de memória de **1** bit



S	$\overline{R}$ / W	Din	Dout	Estado
0	-	-	Z	Inativo
1	0	-	Conteúdo do FF	Read
1	1	0 /1	Z	Write FF captura Din

D_{in} – bit de entrada

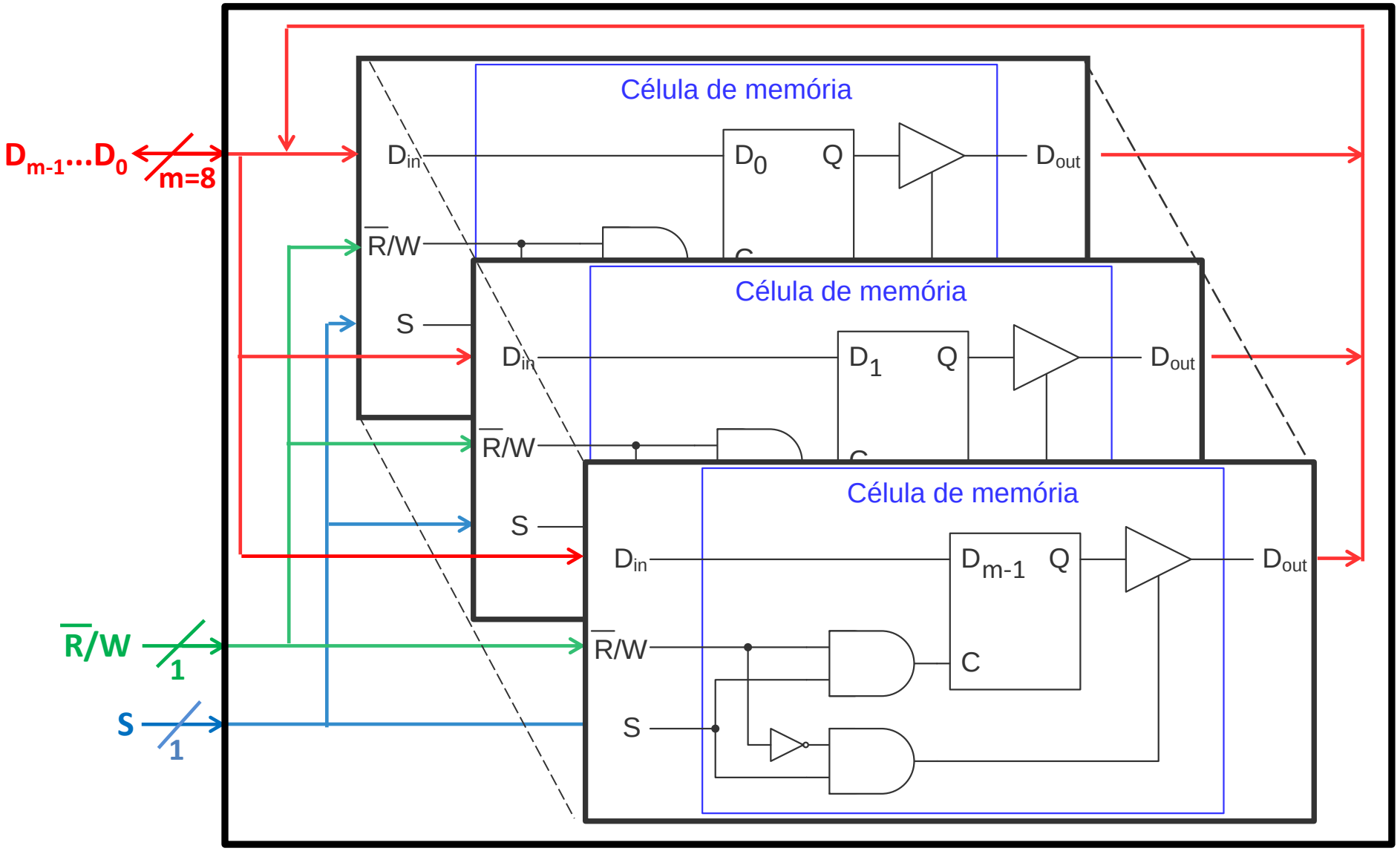
$\overline{R}/W$ – sinal de Read/Write (0=Read , 1=Write)

S – Select(Strobe) : habilita/desabilita a acção de read/write (0=desabilita , 1=habilita)

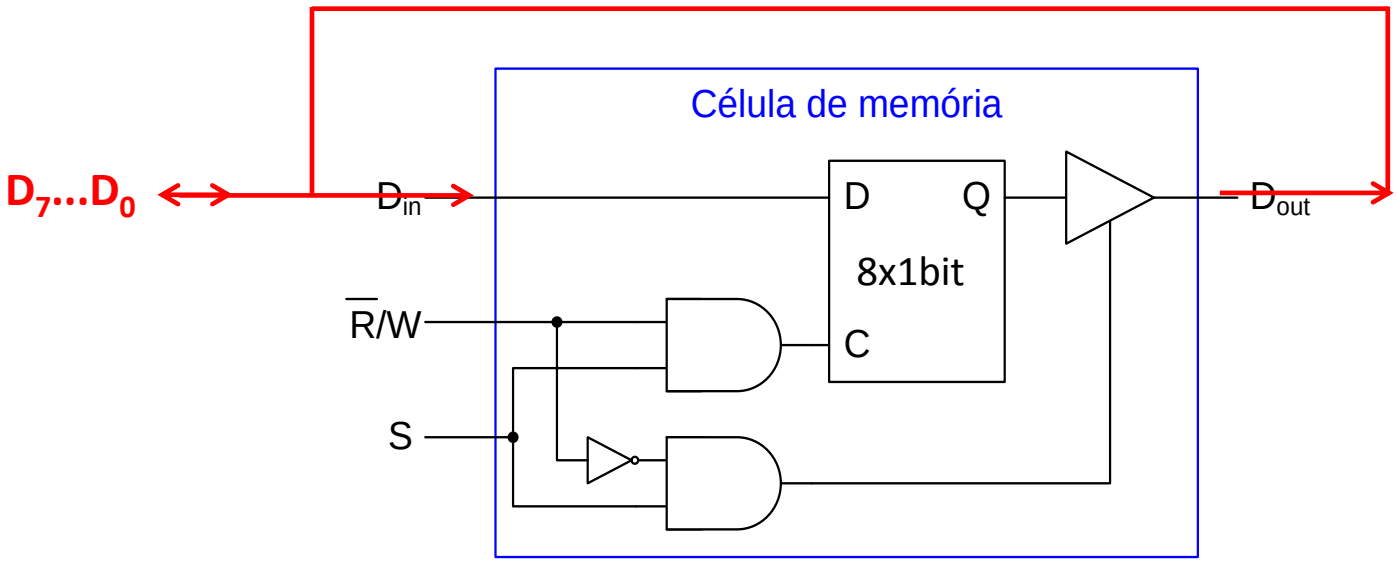
D_{out} – bit de saída

Célula básica de memória : registo de **m** bit

Exemplo: m = 8 bit → registo constituído por 8 células de memória de 1 bit



Célula básica de memória de 8 bit (1 byte)



$D_{7...0}$ – bits de entrada/saída $[D_7\, D_6\, D_5\, D_4\, D_3\, D_2\, D_1\, D_0]$ (m=8)

$\overline{R}/W$ – sinal de Read/Write (0=Read , 1=Write) – só uma acção é possível em cada instante

S – Select(Strobe) : habilita/desabilita a acção de read/write (0=desabilita , 1=habilita)

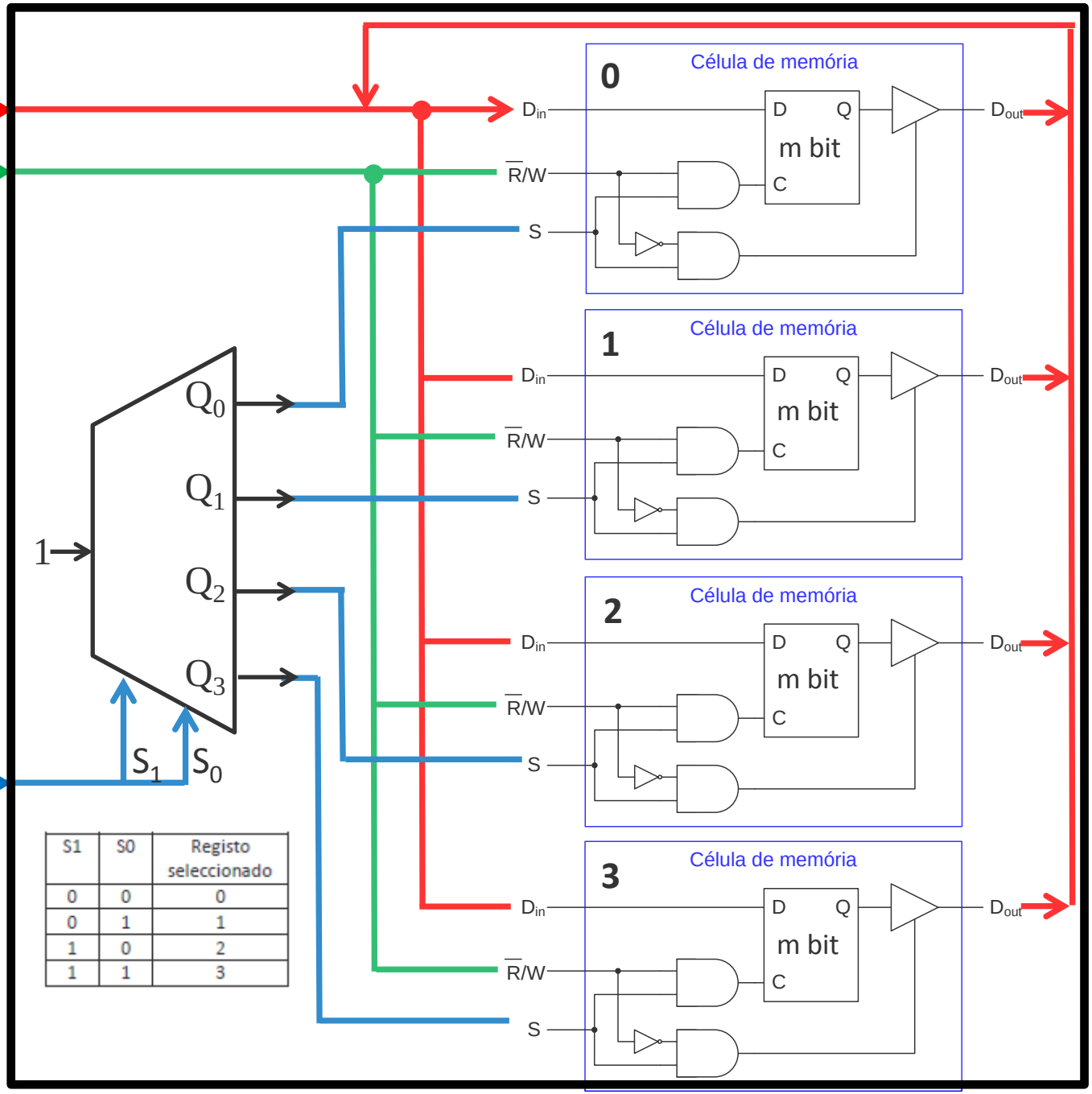
Memória : 2^n registos de **m** bits

dados (Data Bus)
dados a serem lidos ou escritos

controlo (Control Bus)
determina a operação a executar (leitura/escrita)

endereços (Address Bus)
permite seleccionar o registo sobre o qual são feitas as operações de leitura/escrita

Exemplo: $2^n=2^2=$ 4 registos
dois bits de selecção (S_1 , S_0)

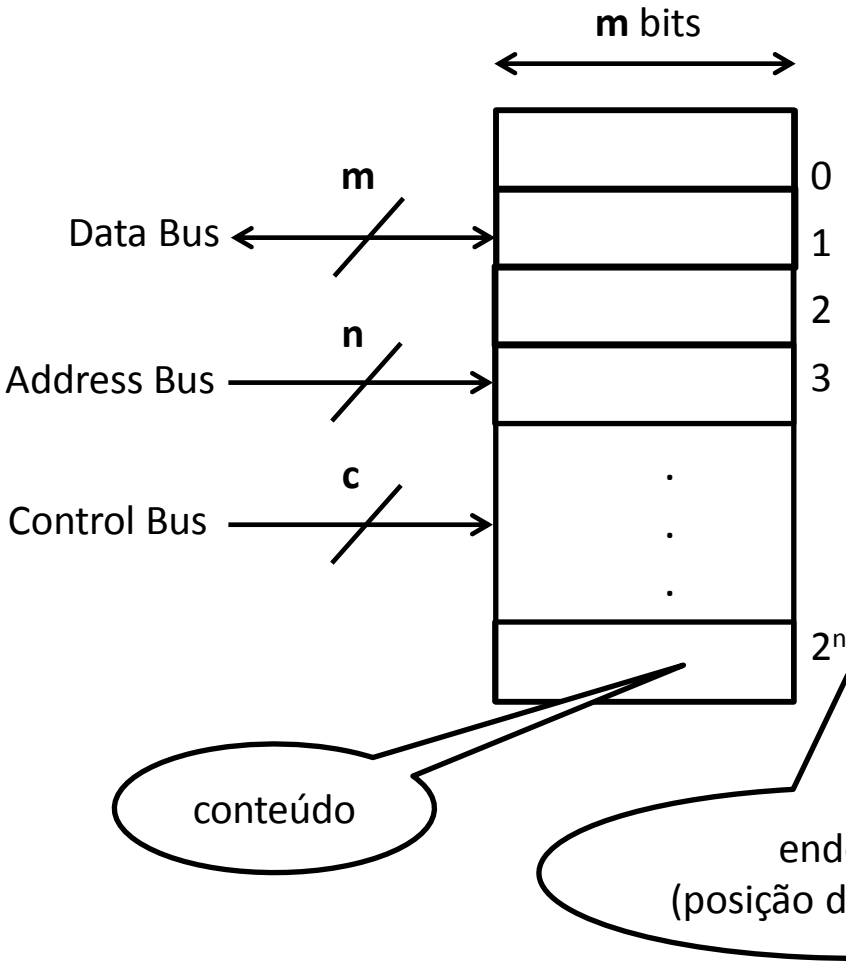


S1	S0	Registo seleccionado
0	0	0
0	1	1
1	0	2
1	1	3

Memória : 2^n registos de **m** bits

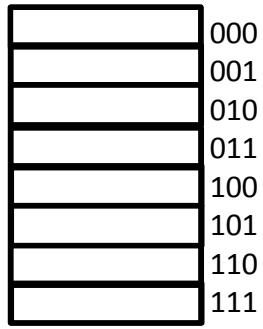
Organização e capacidade da memória (RAM – Random Access Memory)

- palavra : *m* bits (ex. m=8 → 1 byte)
- capacidade : 2^n registos, endereços ou palavras → n bits de selecção
- operações : read(leitura) , write(escrita) , chip select(selecção)



n bits de endereço → 2^n registos (ou posições de memória)

ex: n= 3 → 2^3 = 8 byte
n= 4 → 2^4 = 16 byte
n= 8 → 2^8 = 256 byte
n=10 → 2^{10} = 1024=1Kbyte
n=20 → 2^{20} = 1024*1024=1Mbyte
n=30 → 2^{30} = 1 Gbyte



Estrutura dos Processadores (CPU)

Desafio: construir um dispositivo (processador) capaz de executar uma tarefa descrita por um conjunto de instruções (programa)

1) As tarefas a realizar limitam-se a operações aritméticas sobre valores inteiros, representados em código de complemento para 2 (valores positivos e negativos);

2) As instruções, os dados do problema e respetivos resultados são constituídos por conjuntos de n bits (ex: n = 8 → processador de 8 bits n = 32 → processador de 32 bits)

2.1) codificação das instruções: opcode(operation code)
ex: ADD (soma) → opcode = 00001111
SUB(subtração) → opcode = 11110000

3) As instruções e os dados do problema estão armazenados na memória e os resultados da operação também deverão ficar armazenados na mesma memória;

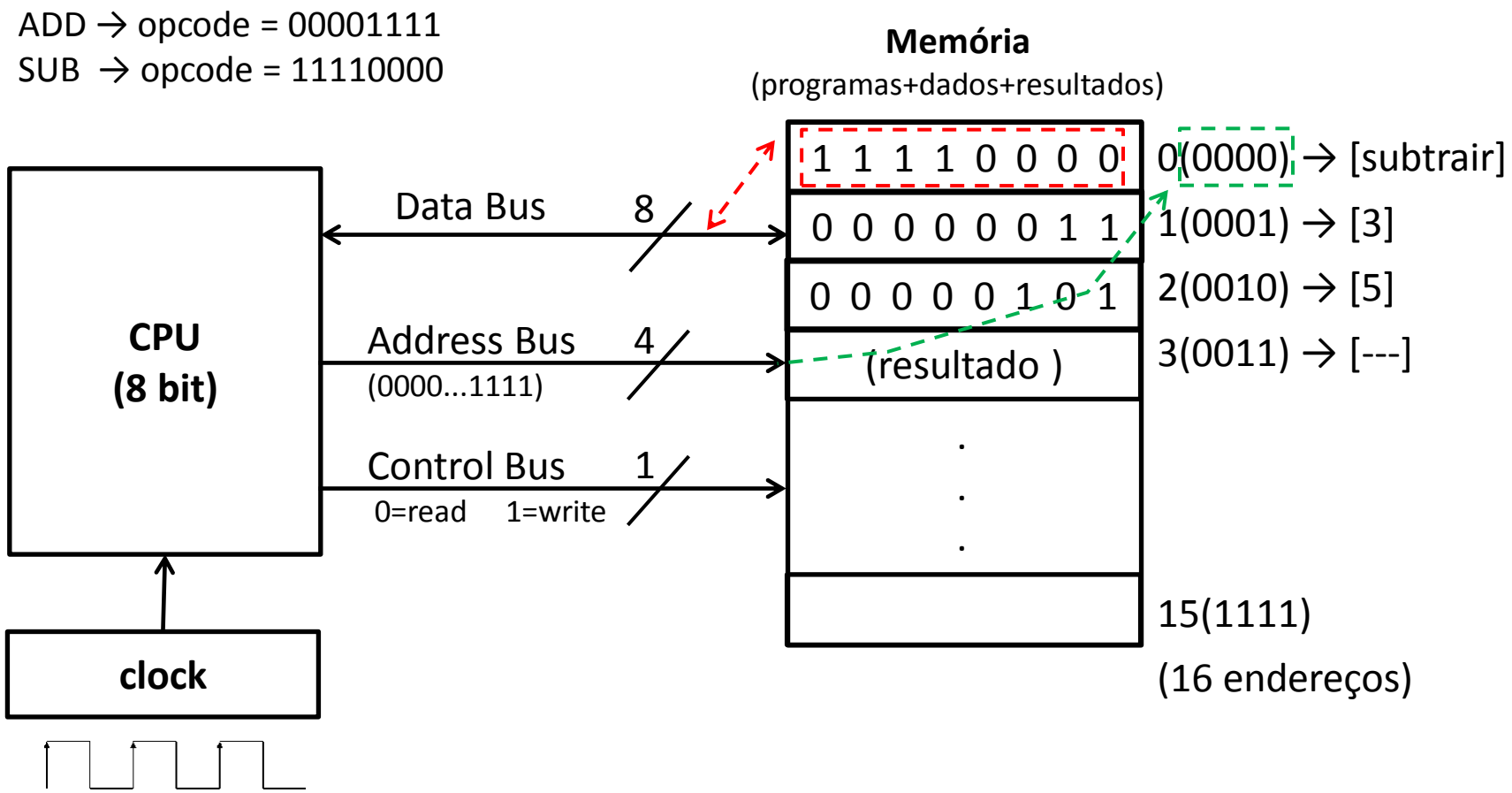
4) As instruções deverão ser executadas sequencialmente pela ordem em que constam na memória, ao ritmo constante de um sinal de controlo (clock);

https://pt.wikipedia.org/wiki/Lista_de_microprocessadores_da_Intel

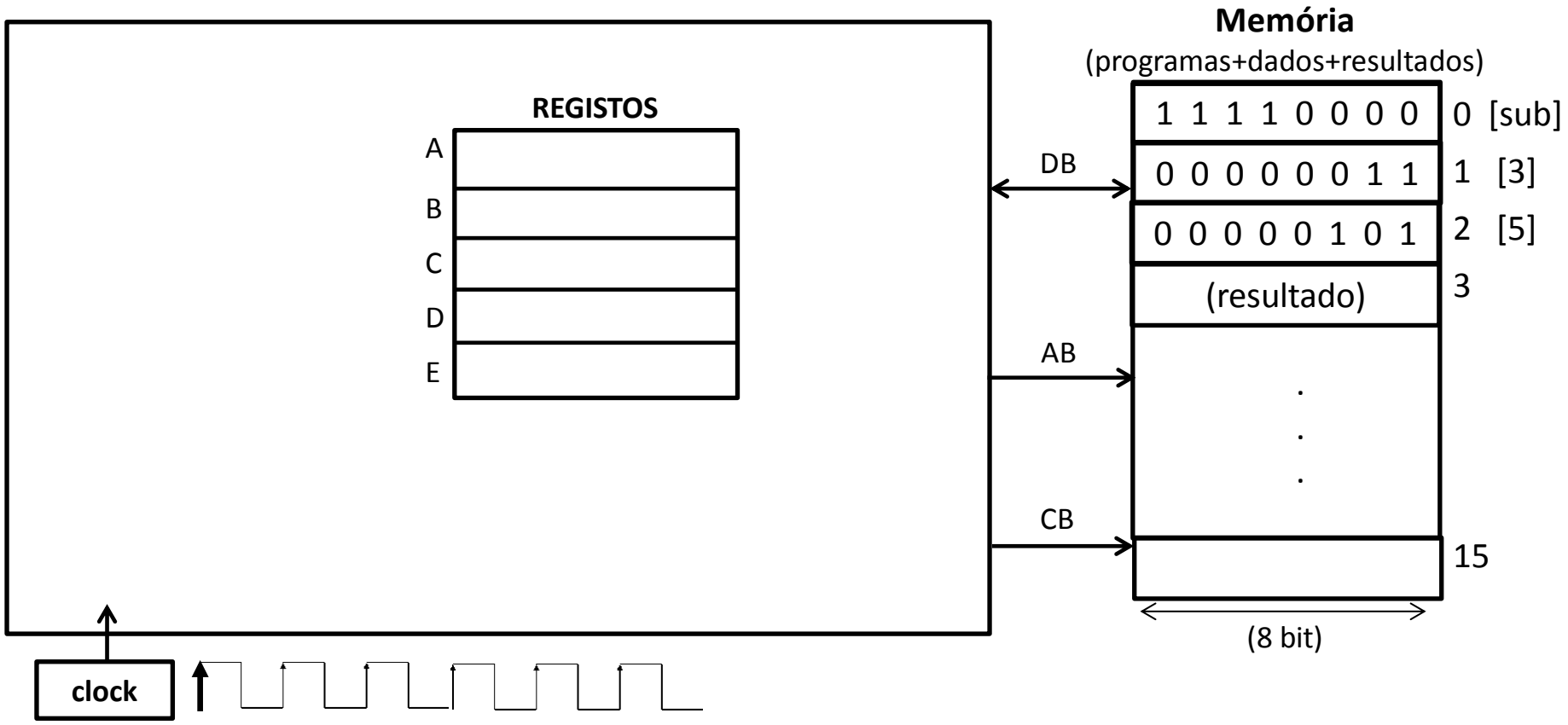
Estrutura dos Processadores (CPU)

Exemplo de tarefa: subtrair o valor 5 do valor 3 e colocar o resultado na memória

subtrair(3,5) = 3 - 5 = -2



Estrutura dos Processadores (CPU)

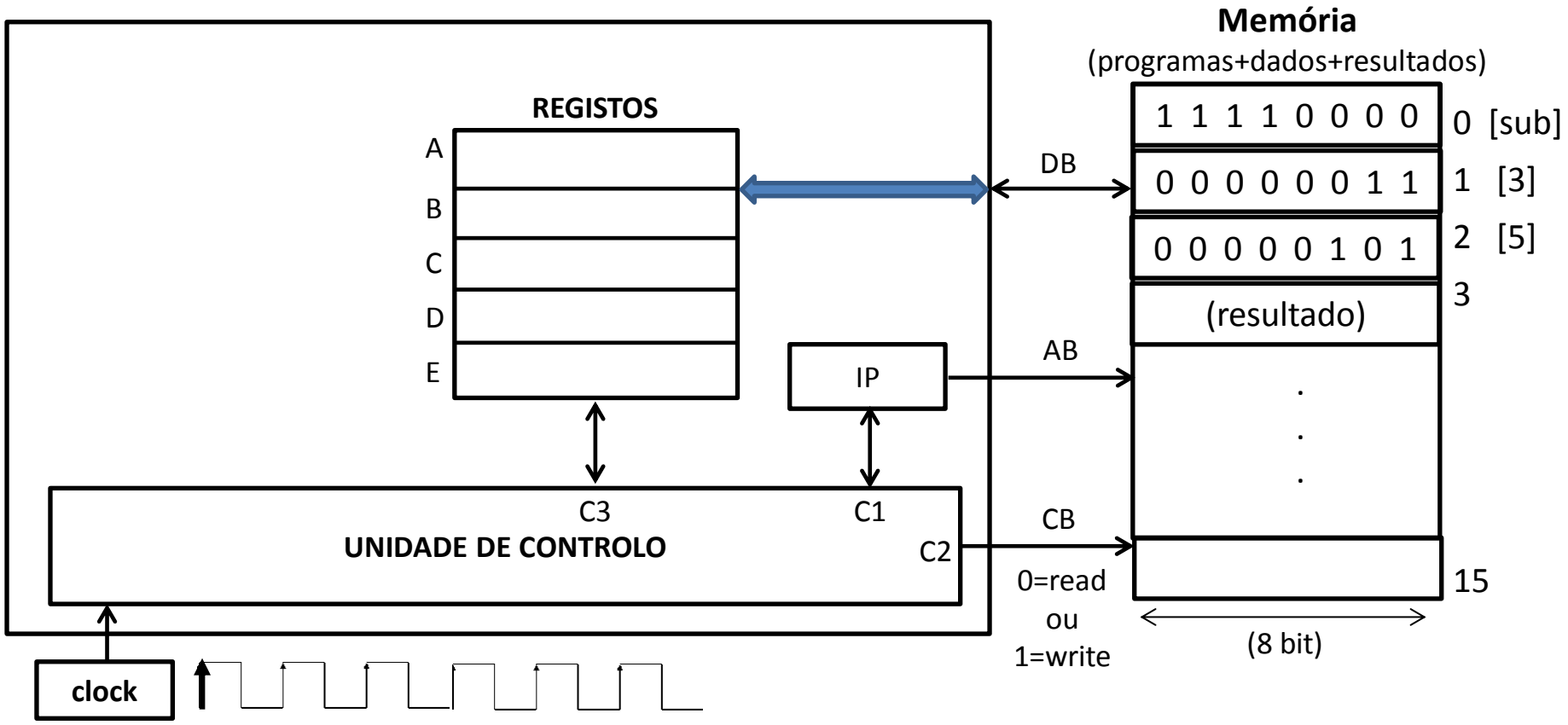


1º) Como indicar ao CPU qual a operação a executar e os correspondentes dados?

→ colocando os respetivos códigos em locais apropriados dentro do CPU

→ o conteúdo desses locais muda frequentemente, logo deverão ser registos; os registos têm nomes e destinam-se a guardar dados e/ou resultados bem como códigos de operações

Estrutura dos Processadores (CPU)



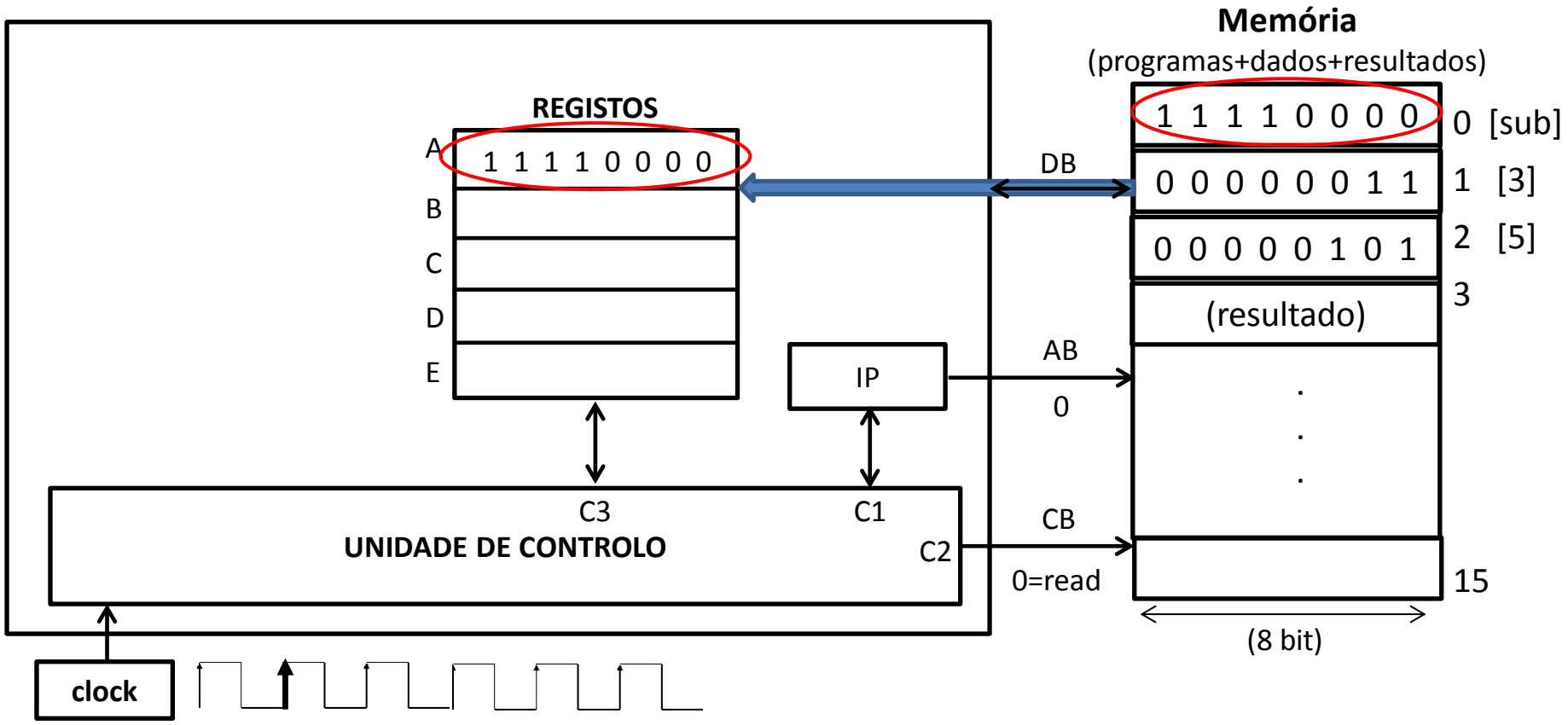
2º) Como pode o CPU aceder às várias células (ou posições) de memória?

- CPU indica qual a posição ou endereço a que pretende aceder → através do AB(Address Bus)

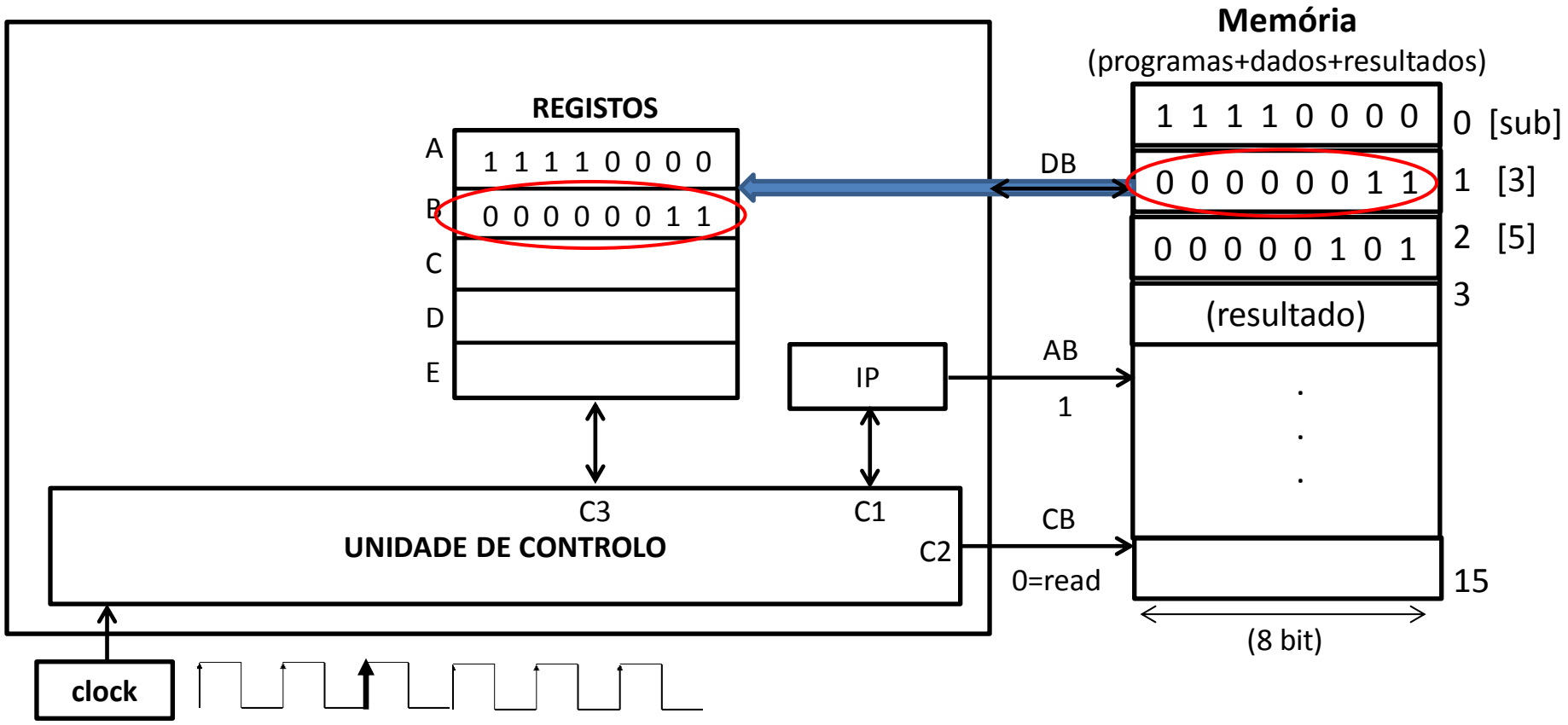
  existe um elemento (IP-Instruction Pointer) que aponta para os endereços e que pode avançar ou recuar consoante o programa vai evoluindo → poderá ser formado por um contador
- CPU indica qual a operação que pretende executar, leitura ou escrita → através do CB(Control Bus)
- CPU recebe ou envia o conteúdo das diversas posições de memória ↔ através do DB(Data Bus)

Esta sequência é controlada pela UC(Unidade de Controlo)

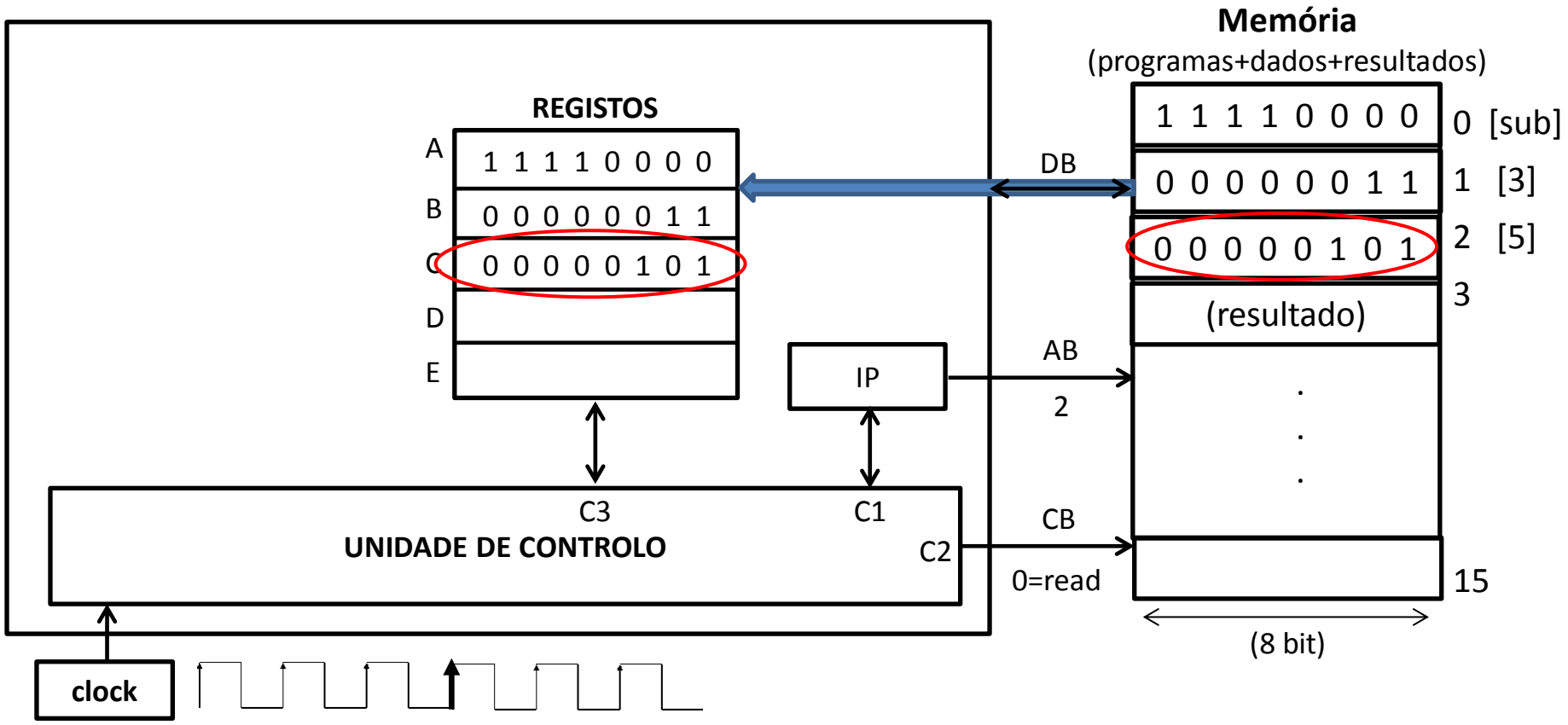
Estrutura dos Processadores (CPU)



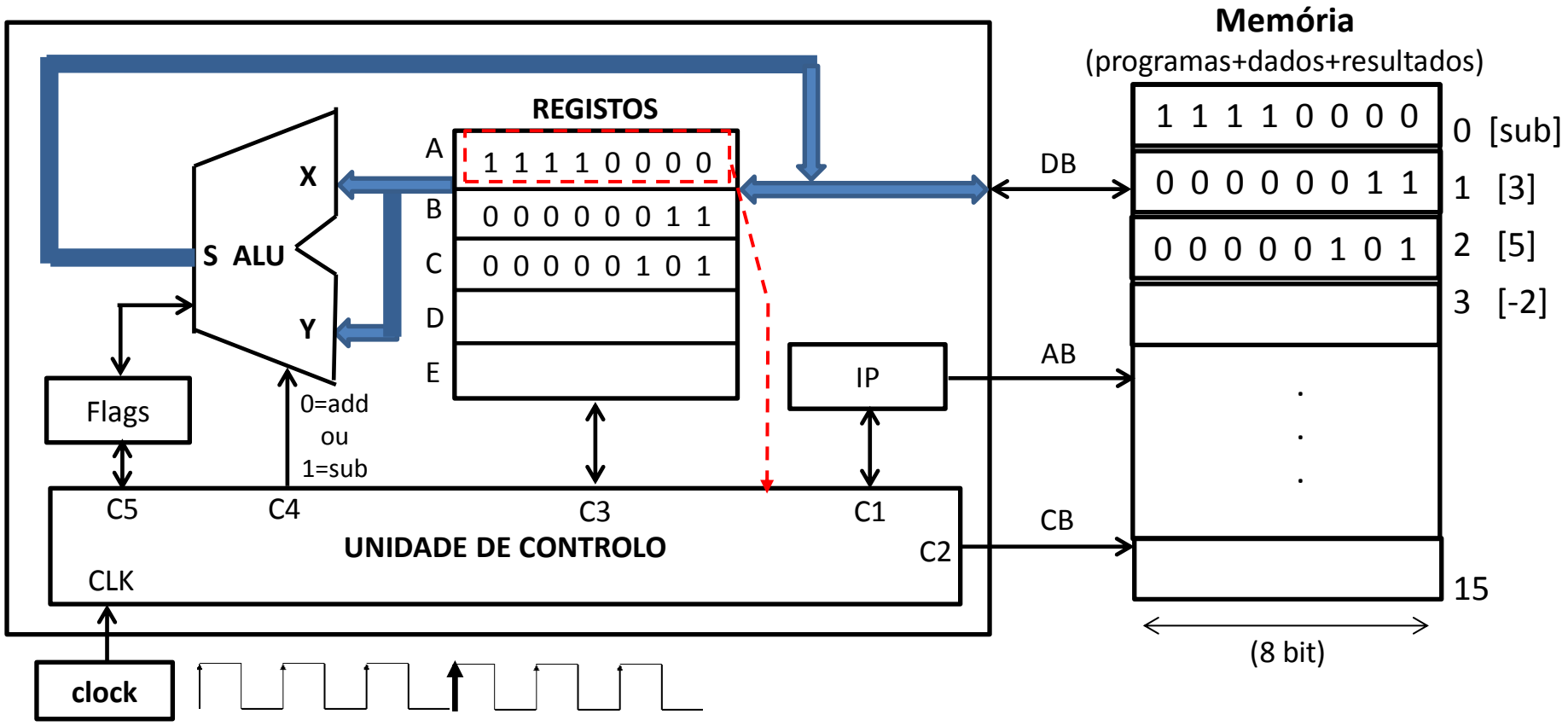
Estrutura dos Processadores (CPU)



Estrutura dos Processadores (CPU)



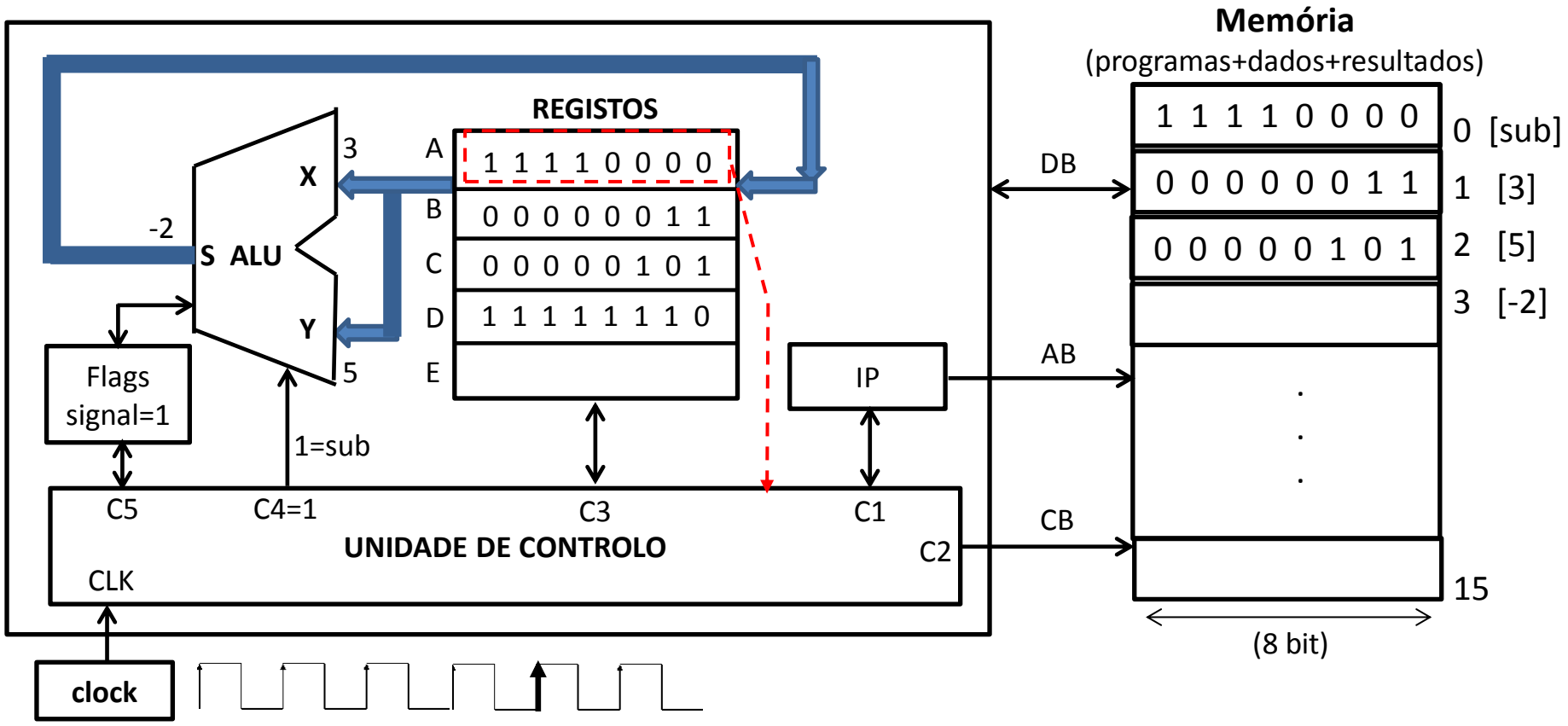
Estrutura dos Processadores (CPU)



3º) Como pode o CPU executar a operação?

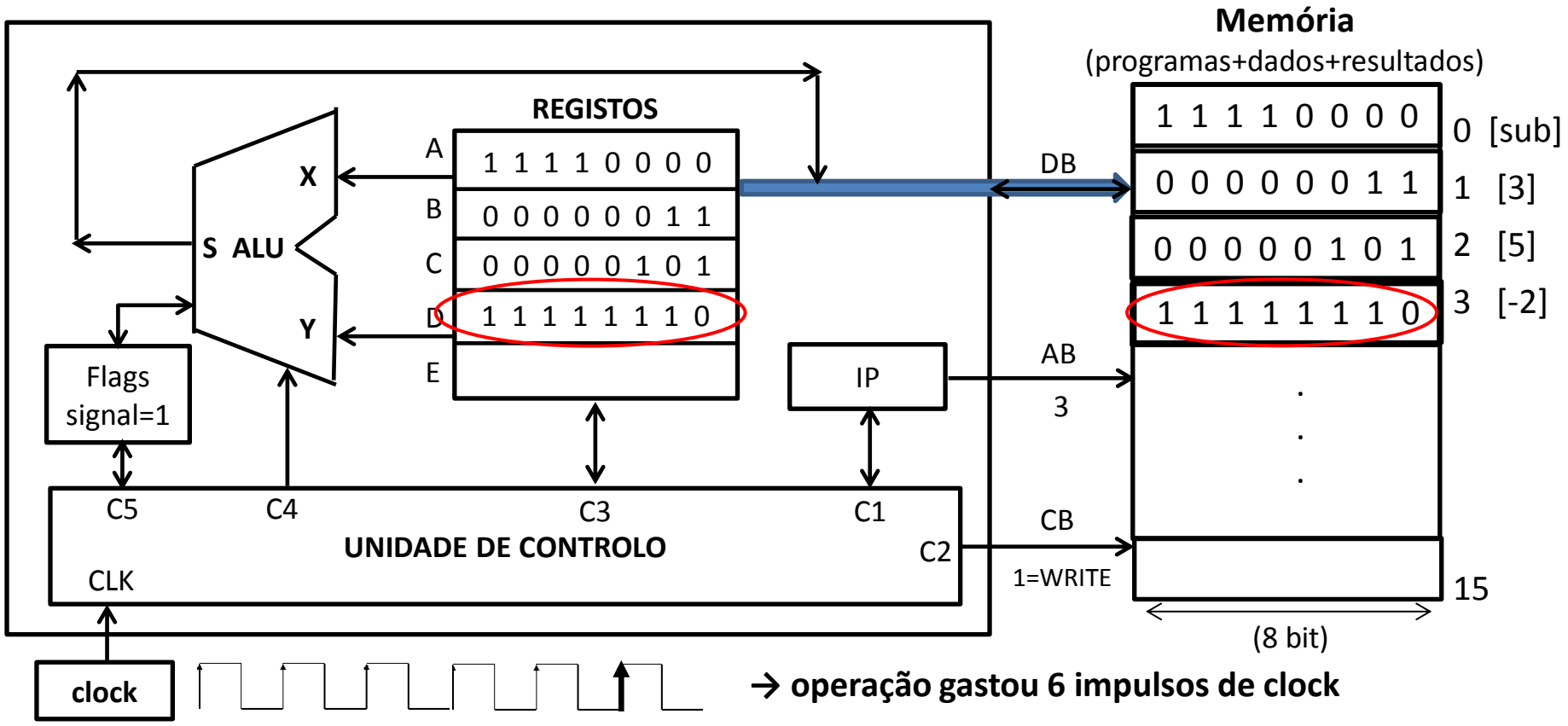
- dispondo de uma ALU a qual realiza operações aritméticas e lógicas – esta é controlada pela UC em função do código da instrução a executar, ou seja, pelo "*opcode*" (o qual neste exemplo está contido no registo A)
- a UC descodifica o *opcode* e em resultado disso gera os sinais de controlo apropriados
- certas situações devem ser registadas para controlarem as próximas acções do CPU - por exemplo se houver uma divisão por zero o processamento poderá parar e ser indicado um erro, ou, como neste exemplo, assinalar que o resultado é negativo → registo de Flags(bandeiras)

Estrutura dos Processadores (CPU)



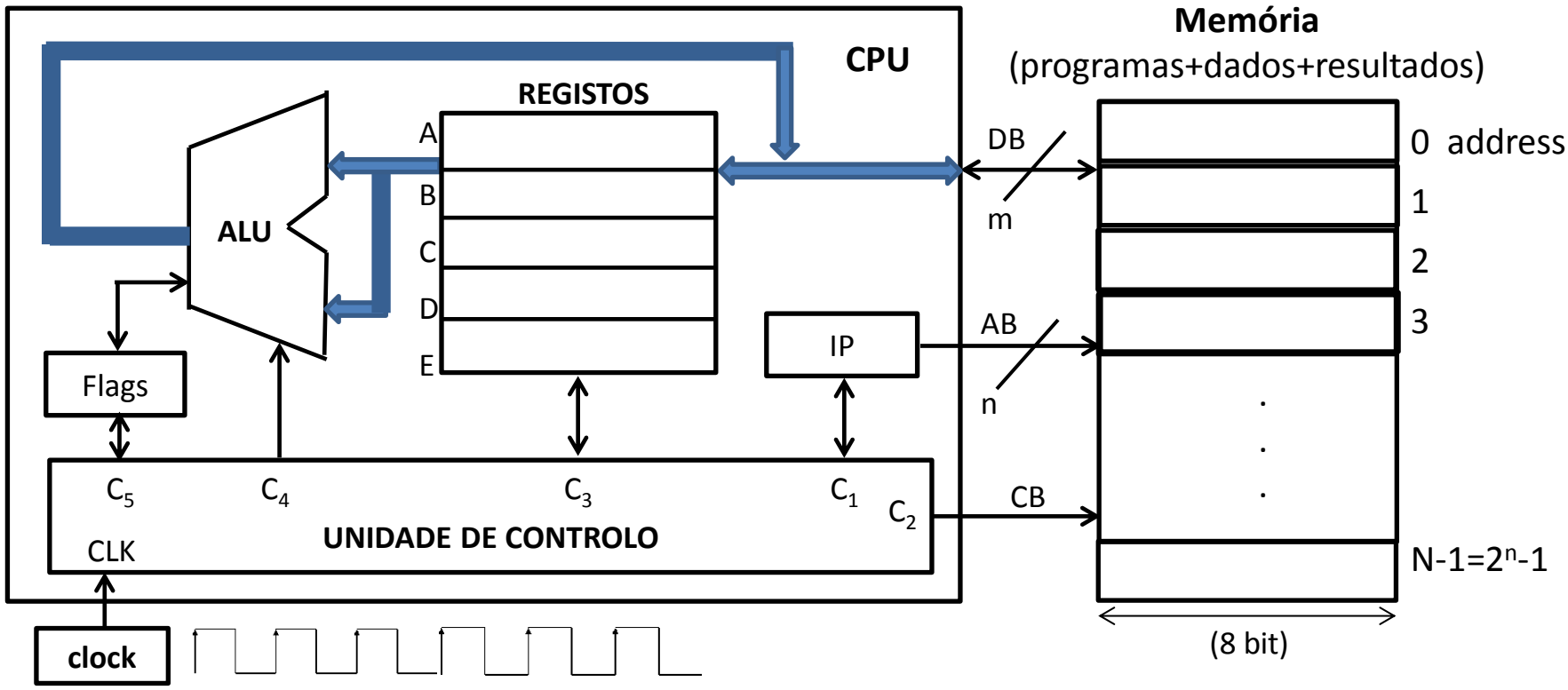
- a operação é executada e o resultado (-2=11111110) é colocado num registo disponível, neste caso o registo D
- a flag *signal* é ativada, indicando que o resultado da operação é negativo

Estrutura dos Processadores (CPU)



- o IP avança para apontar à posição de memória que vai conter o resultado
- a UC canaliza o conteúdo do registo D para o DB
- o sinal no CB muda para indicar que o processador pretende enviar um valor para a memória (Write)
- a tarefa (programa) termina

Estrutura dos Processadores (CPU)



Memória: contém os dados dos problemas e as instruções para operar sobre esses dados(ie,os programas)

- constituída por $N=2^n$ registos de m bits (ex: m=8 , n=20 → 1 MByte), cada um com o seu endereço(address)

CPU: - executa as instruções a partir da memória

- REGISTOS: armazenam os dados/resultados das operações, bem como os códigos das operações a executar
  - IP(Instruction Pointer): aponta na memória qual o endereço da próxima instrução a executar
  - Flags: sinalizam determinadas ocorrências (ex: operação com resultado negativo)
- ALU: executa as operações artiméticas e lógicas
- UC(Unidade de Controlo): coordena o funcionamento dos restantes blocos , gerando sinais de controlo C_n
- Clock: determina a cadência a que as instruções são executadas

Aula 3
2021-03-12

Execução das instruções (cont.)

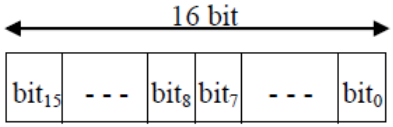
Página da UC

http://www.di.ubi.pt/~paraujo/Cadeiras/ArquiteturaComputadores/2020-2021/AC_2020-2021.html

Registos do processador 8086 (x86-16 bits)

Agrupam-se em: 1)dados 2)endereços 3)segmento 4)controlo

1)DADOS (uso geral e funções do processador) X=16bit H=High(8bit) L=Low(8bit)



Funções típicas

AX	AH	AL	Acumulador: operações lógicas/aritméticas, I/O, funções do processador , etc
BX	BH	BL	Base: modos de endereçamento
CX	CH	CL	Contagem: contagens, ciclos (loops)
DX	DH	DL	Dados: multiplicação/divisão, I/O, extensão do AX

2)ENDEREÇOS

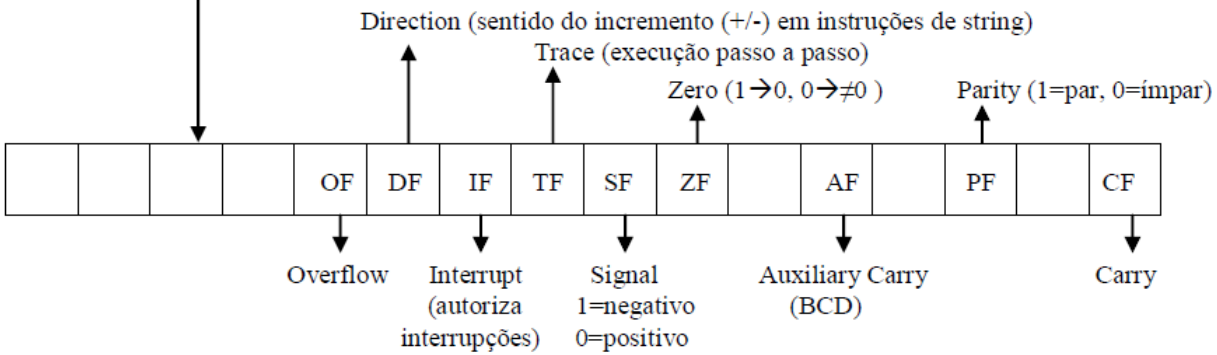
SP	Stack Pointer: controle da pilha (stack)
BP	Base Pointer: registo base de endereçamento
SI	Source Index: indexador de origem (strings)
DI	Destination Index: indexador de destino (strings)

3)SEGMENTO

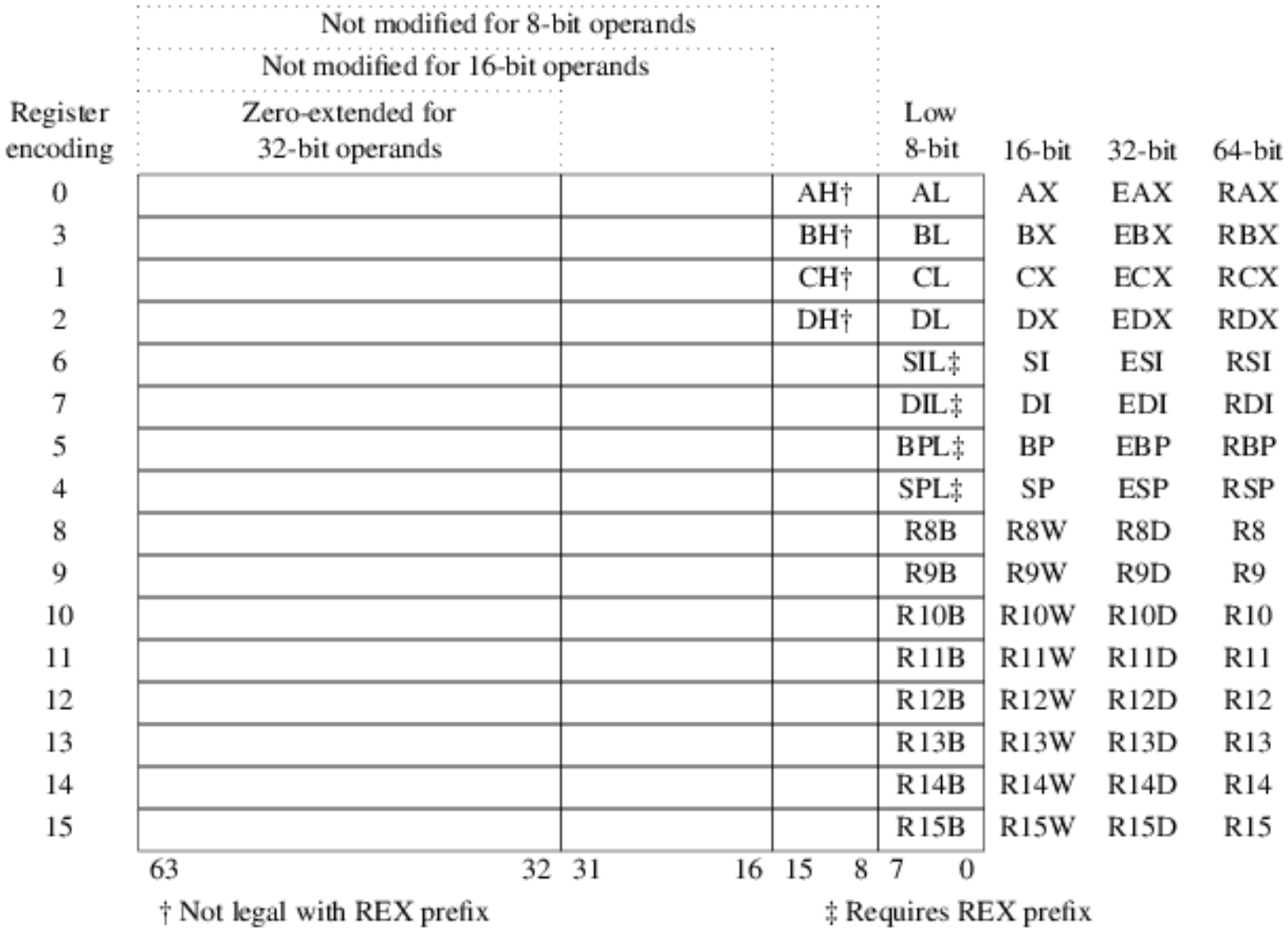
CS	Code Segment: acesso ao código
DS	Data Segment: acesso aos dados
SS	Stack Segment: acesso à pilha
ES	Extra Segment: extensão (strings)

4)CONTROLO

IP	Instruction Pointer: indica a localização das instruções na memória
FLAGS	Indicadores do estado do processador



Conjunto de registos x86 de 64-bit



https://www.tortall.net/projects/yasm/manual/html/arch-x86-registers.html

Processo de programação

Máquina : apenas trabalha com bits (0 e 1) → tudo tem de ser introduzido na máquina nesta forma, incluindo dados e programas(conjuntos de instruções)

- Programa em código binário → programa em linguagem máquina ou em baixo nível

Homem: é difícil trabalhar com extensas listas de bits, é mais fácil trabalhar com linguagens naturais (ex:Inglês)

- Programa em linguagem natural (ex: C, Java, HTML) → programa em alto nível

Processo de tradução: passagem de um programa de uma linguagem de alto nível para uma de baixo nível

ex. printf(“Hello World”) → 10010001001001000100001111100101010...

Compilador: programa que traduz o código fonte de um programa escrito numa linguagem de alto nível, para o código correspondente numa linguagem de programação de baixo nível (por exemplo, Assembly ou código máquina) – todas as instruções do programa são primeiro convertidas e depois o programa é executado como um todo;

Interpretador: a diferença para o compilador é que a tradução do programa fonte é feita instrução a instrução, sendo cada uma delas executada de imediato;

Assembly

Havendo necessidade de programar diretamente em código máquina, isso obrigaria a utilizar códigos de 0 e 1's, difíceis de manipular e memorizar.

O *Assembly (*linguagem de montagem) simplifica este processo ao atribuir nomes (mnemónicas) a conjuntos de bits, nomes esses que permitem usar as instruções nativas do processador sem usar o respetivo código binário.

Em seguida é usado um programa de montagem, designado por Assembler, que irá converter cada mnemónica para o correspondente código binário.

instrução em Assembly = <opcode> , <operand>

Ex: opcode(mnemónica)	operand	binário		ação
mov al	5	10110000	00000101	mover para AL o valor 5
add ax	539Fh	00000101	0101 0011 1001 1111	somar ao registo AX o valor 539Fh

NOTA: a designação da linguagem é *Assembly* e não Assembler (é errado dizer "programar em Assembler"). O Assembler não é uma linguagem, mas sim um programa que efetua a tradução das mnemónicas do *Assembly* para código máquina.

Intel 80x86 Assembly Language OpCodes : http://www.mathemainzel.info/files/x86asmref.html

Assembly

Razões para utilizar Assembly

Rapidez: o código em Assembly pode ser optimizado, resultando em programas mais pequenos, logo mais rápidos. Tomando como exemplo a programação de jogos, estes têm de responder muito rapidamente às ações do utilizador.

Memória: os tradutores automáticos (compiladores, tradutores) geram por vezes código supérfluo o qual pode ocupar memória desnecessária. Programando diretamente em Assembly pode reduzir-se a ocupação da memória.

Eficiência: um programa em Assembly faz uso direto das características do processador a que se destina, o que obriga a um conhecimento aprofundado dessas características para tirar partido da linguagem. Ao usar o Assembly, o programador adquire conhecimentos que lhe permitem inclusivamente escrever código eficiente mesmo ao usar linguagens de alto-nível.

Controlo: por razões de segurança, a maioria dos compiladores/interpretadores dificulta ou bloqueia o acesso a certas componentes do hardware, o que pode ser ultrapassado pelo uso do Assembly.

Processo de programação

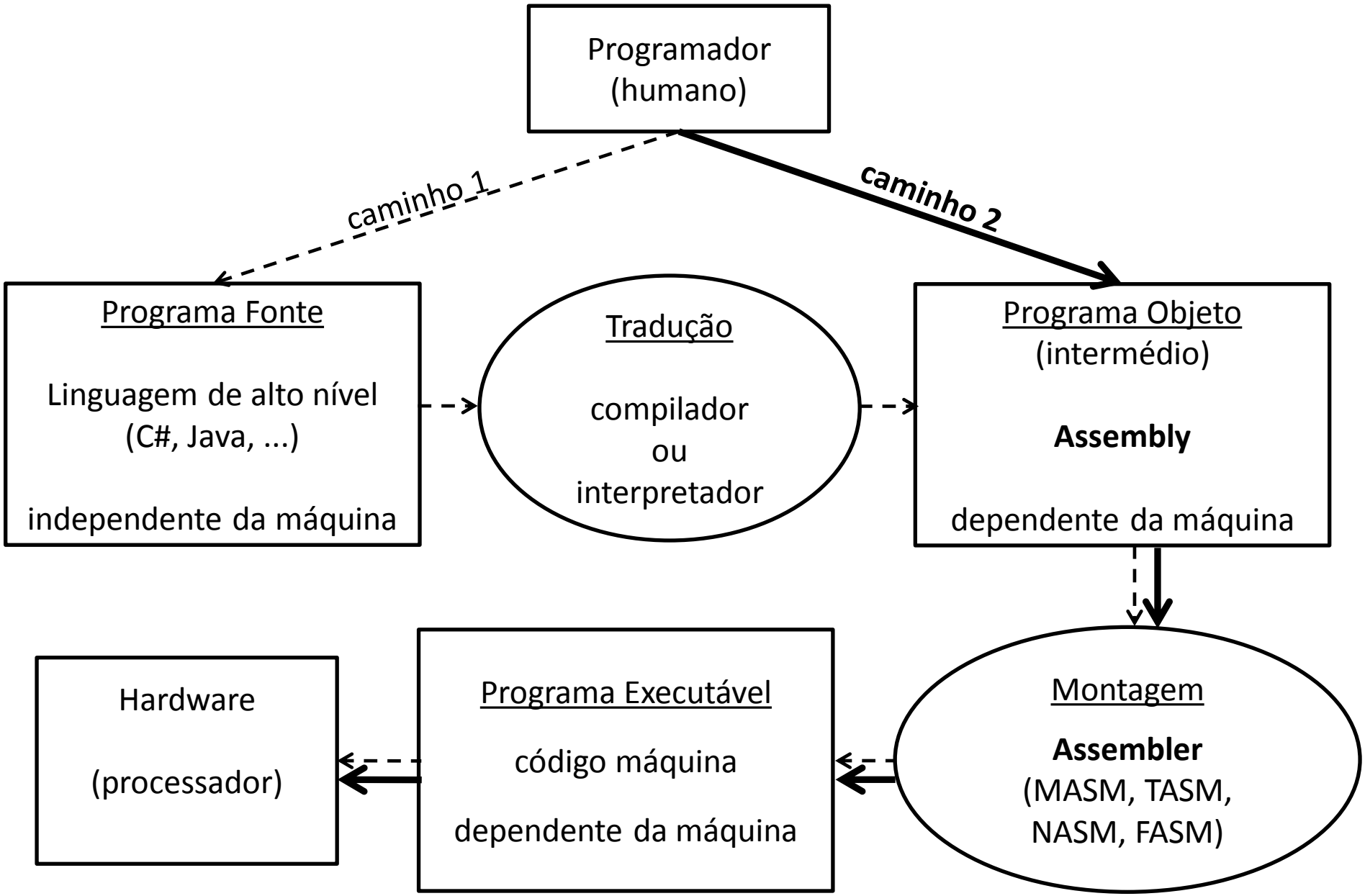


Tabela ASCII - American Standard Code for Information Interchange

Relaciona os carateres com a sua representação numérica (decimal, hexadecimal, binária)

Tabela ASCII (7bits)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00h	^@ Null	32	20h		64	40h	@	96	60h	`
1	01h	☺ ^A SOH-Start of Header	33	21h	!	65	41h	A	97	61h	a
2	02h	☻ ^B STX- Start of Text	34	22h	"	66	42h	B	98	62h	b
3	03h	♥ ^C ETX- End of Text	35	23h	#	67	43h	C	99	63h	c
4	04h	♦ ^D EOT- End of Transmission	36	24h	$	68	44h	D	100	64h	d
5	05h	♣ ^E ENQ- Enquiry	37	25h	%	69	45h	E	101	65h	e
6	06h	♠ ^F ACK- Acknowledgment	38	26h	&	70	46h	F	102	66h	f
7	07h	• ^G BEL- Bell	39	27h	'	71	47h	G	103	67h	g
8	08h	◘ ^H BS- Backspace	40	28h	(	72	48h	H	104	68h	h
9	09h	○ ^I HT-Horizontal Tab	41	29h	)	73	49h	I	105	69h	i
10	0Ah	◙ ^J LF-Line Feed	42	2Ah	*	74	4Ah	J	106	6Ah	j
11	0Bh	♂ ^K VT-Vertical Tab	43	2Bh	+	75	4Bh	K	107	6Bh	k
12	0Ch	♀ ^L FF-Form Feed	44	2Ch	,	76	4Ch	L	108	6Ch	l
13	0Dh	♪ ^M CR-Carriage Return	45	2Dh	-	77	4Dh	M	109	6Dh	m
14	0Eh	♫ ^N SO-Shift Out	46	2Eh	.	78	4Eh	N	110	6Eh	n
15	0Fh	☼ ^O SI- Shift In	47	2Fh	/	79	4Fh	O	111	6Fh	o
16	10h	► ^P DLE- Data Link Escape	48	30h	0	80	50h	P	112	70h	p
17	11h	◄ ^Q DC1- (XON) Device Control1	49	31h	1	81	51h	Q	113	71h	q
18	12h	↕ ^R DC2- Device Control2	50	32h	2	82	52h	R	114	72h	r
19	13h	‼ ^S DC3- (XOFF) Device Control3	51	33h	3	83	53h	S	115	73h	s
20	14h	¶ ^T DC4- Device Control4	52	34h	4	84	54h	T	116	74h	t
21	15h	§ ^U NAK- Negative Acknowledge	53	35h	5	85	55h	U	117	75h	u
22	16h	▬ ^V SYN- Synchronous Idle	54	36h	6	86	56h	V	118	76h	v
23	17h	↨ ^W ETB- End of Trans. Block	55	37h	7	87	57h	W	119	77h	w
24	18h	↑ ^X CAN- Cancel	56	38h	8	88	58h	X	120	78h	x
25	19h	↓ ^Y EM- End of Medium	57	39h	9	89	59h	Y	121	79h	y
26	1Ah	→ ^Z SUB- Substitute	58	3Ah	:	90	5Ah	Z	122	7Ah	z
27	1Bh	← ^[ESC- Escape	59	3Bh	;	91	5Bh	[	123	7Bh	{
28	1Ch	└ ^\ FS- File Separator	60	3Ch	<	92	5Ch	\	124	7Ch	|
29	1Dh	↔ ^] GS- Group Separator	61	3Dh	=	93	5Dh	]	125	7Dh	}
30	1Eh	▲ ^^ RS- Record Separator	62	3Eh	>	94	5Eh	^	126	7Eh	~
31	1Fh	▼ ^_ US- Unit Separator	63	3Fh	?	95	5Fh	_	127	7Fh	⌂

Exemplos

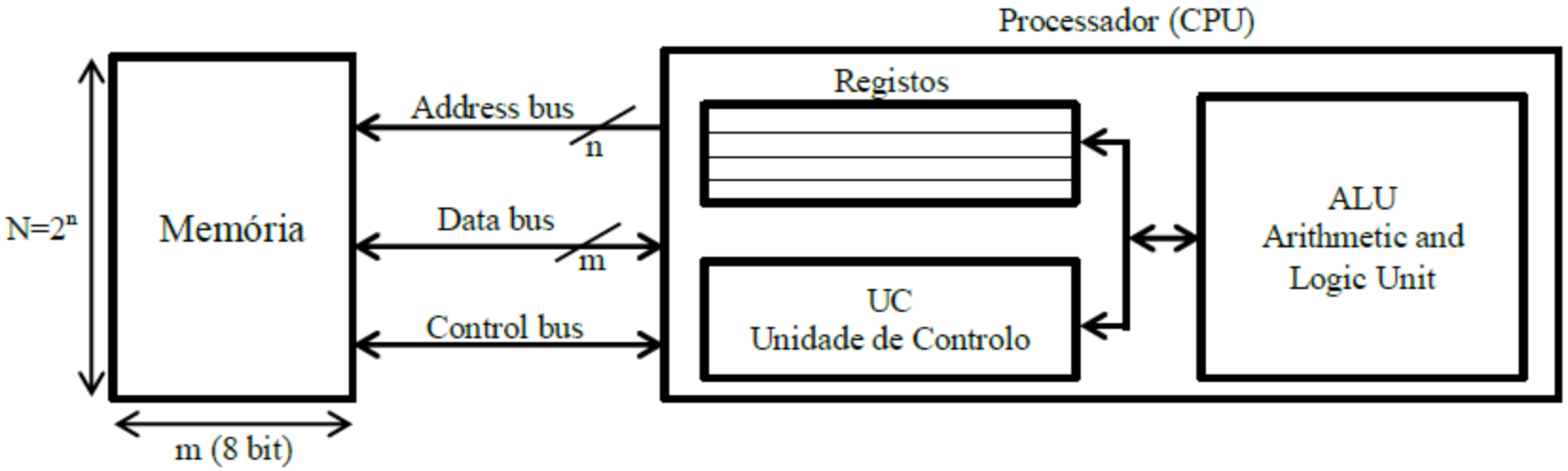
espaço = 32=20h=00100000b

'A' = 65=41h=01000001b

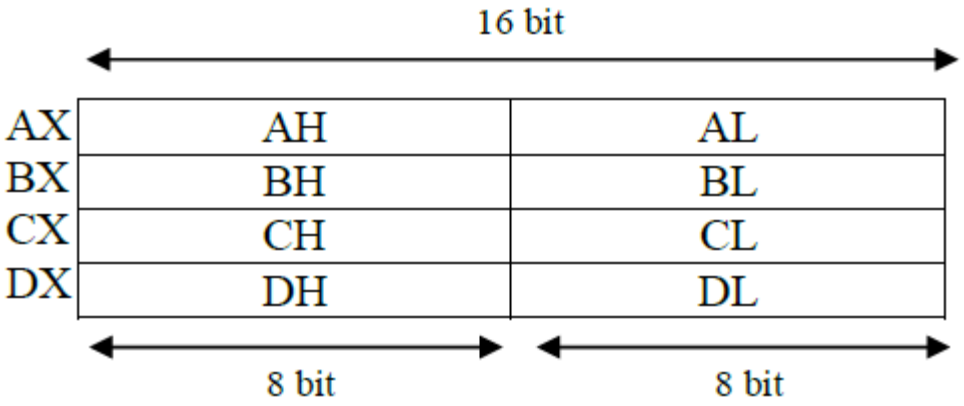
'z' = 122=7Ah=01111010b

Decimal (pesos) 10^1 10^0	Binário (pesos) 2^3 2^2 2^1 2^0 8 4 2 1	Octal (pesos) 8^1 8^0	Hexadecimal (pesos) 16^0
0	0 0 0 0	0	0
1	0 0 0 1	1	1
2	0 0 1 0	2	2
3	0 0 1 1	3	3
4	0 1 0 0	4	4
5	0 1 0 1	5	5
6	0 1 1 0	6	6
7	0 1 1 1	7	7
8	1 0 0 0	10	8
9	1 0 0 1	11	9
10	1 0 1 0	12	A
11	1 0 1 1	13	B
12	1 1 0 0	14	C
13	1 1 0 1	15	D
14	1 1 1 0	16	E
15	1 1 1 1	17	F

Processador básico



Registos principais do CPU (família Intel x86): x = 16



Execução dos programas

1º exemplo: escrever no ecrã a mensagem “Ola UBI” → printf(“Ola UBI”)

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

Programa Assembly

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ;escrita
mov bx, 1     ;ecran
mov cx, 7     ;nº caracteres
mov dx, msg   ;string
int  21h      ;executa

;aguarda tecla
mov ah, 3Fh   ;leitura
int  21h      ;executa

;terminar o programa
mov ah, 4Ch   ;terminar
int  21h      ;executa

msg db 'Ola UBI'
```

Execução dos programas

1º exemplo: escrever no ecrã a mensagem "Ola UBI" → printf("Ola UBI")

Programa Assembly

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ;escrita
mov bx, 1     ;ecran
mov cx, 7     ;nº caracteres
mov dx, msg   ;string
int  21h      ;executa

;aguarda tecla
mov ah, 3Fh   ;leitura
int  21h      ;executa

;terminar o programa
mov ah, 4Ch   ;terminar
int  21h      ;executa

msg db 'Ola UBI'
```

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

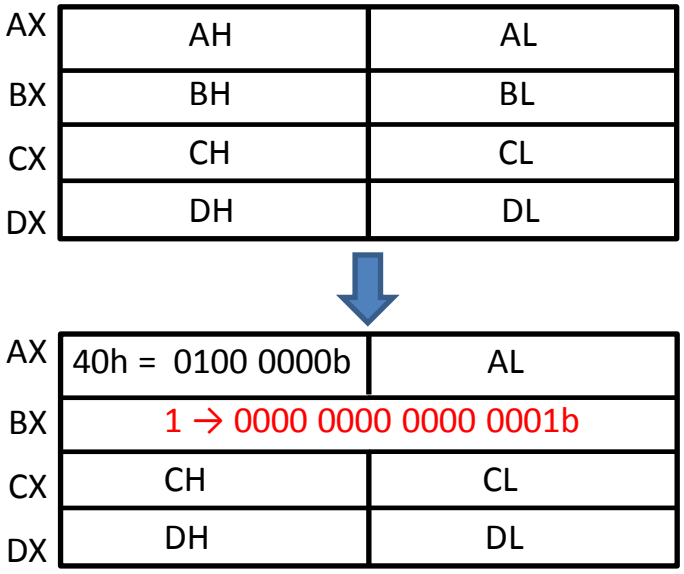
AX	40h = 0100 0000b	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

Execução dos programas

1º exemplo: escrever no ecrã a mensagem "Ola UBI" → printf("Ola UBI")

Programa Assembly

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ;escrita
mov bx, 1     ;ecran
mov cx, 7     ;nº caracteres
mov dx, msg   ;string
int  21h      ;executa

;aguarda tecla
mov ah, 3Fh   ;leitura
int  21h      ;executa

;terminar o programa
mov ah, 4Ch   ;terminar
int  21h      ;executa

msg db 'Ola UBI'
```

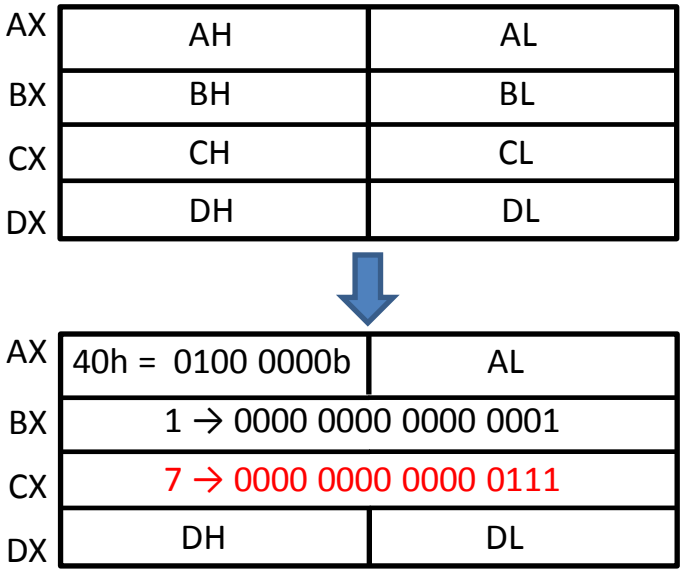


Execução dos programas

1º exemplo: escrever no ecrã a mensagem "Ola UBI" → printf("Ola UBI")

Programa Assembly

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ;escrita
mov bx, 1     ;ecran
mov cx, 7     ;nº caracteres
mov dx, msg   ;string
int  21h      ;executa

;aguarda tecla
mov ah, 3Fh   ;leitura
int  21h      ;executa

;terminar o programa
mov ah, 4Ch   ;terminar
int  21h      ;executa

msg db 'Ola UBI'
```



Execução dos programas

1º exemplo: escrever no ecrã a mensagem "Ola UBI" → printf("Ola UBI")

Programa Assembly

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ;escrita
mov bx, 1     ;ecran
mov cx, 7     ;nº caracteres
mov dx, msg   ;string
int  21h      ;executa

;aguarda tecla
mov ah, 3Fh  ;leitura
int  21h      ;executa

;terminar o programa
mov ah, 4Ch  ;terminar
int  21h      ;executa

msg db 'Ola UBI'
```

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

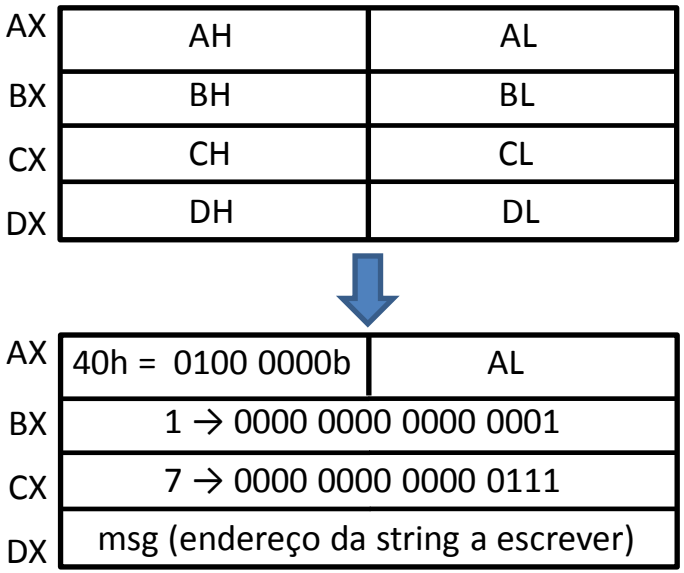
AX	40h = 0100 0000b	AL
BX	1 → 0000 0000 0000 0001	
CX	7 → 0000 0000 0000 0111	
DX	msg (endereço da string a escrever)	

Execução dos programas

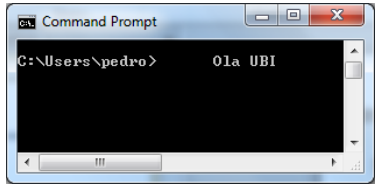
1º exemplo: escrever no ecrã a mensagem “Ola UBI” → printf(“Ola UBI”)

Programa Assembly

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ;escrita
mov bx, 1     ;ecran
mov cx, 7     ;nº caracteres
mov dx, msg   ;string
int  21h      ;executa

;aguarda tecla
mov ah, 3Fh  ;leitura
int  21h     ;executa

;terminar o programa
mov ah, 4Ch  ;terminar
int  21h     ;executa

msg db 'Ola UBI'
```



escreve no ecran

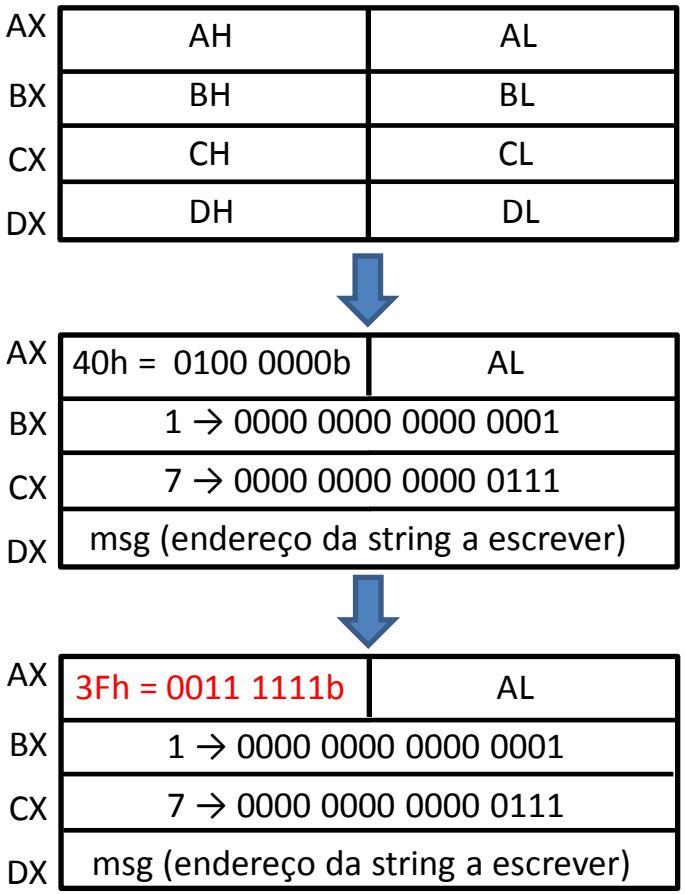


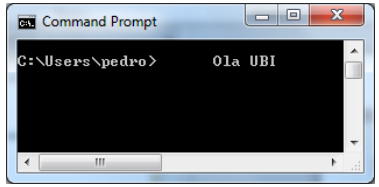
Execução dos programas

1º exemplo: escrever no ecrã a mensagem "Ola UBI" → printf("Ola UBI")

Programa Assembly

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ;escrita
mov bx, 1     ;ecran
mov cx, 7     ;nº caracteres
mov dx, msg   ;string
int  21h      ;executa

;aguarda tecla
mov ah, 3Fh  ;leitura
int  21h      ;executa

;terminar o programa
mov ah, 4Ch  ;terminar
int  21h      ;executa

msg db 'Ola UBI'
```





Execução dos programas

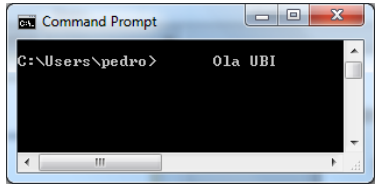
1º exemplo: escrever no ecrã a mensagem "Ola UBI" → printf("Ola UBI")

Programa Assembly

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ;escrita
mov bx, 1     ;ecran
mov cx, 7     ;nº caracteres
mov dx, msg   ;string
int  21h      ;executa

;aguarda tecla
mov ah, 3Fh  ;leitura
int  21h     ;executa

;terminar o programa
mov ah, 4Ch  ;terminar
int  21h     ;executa

msg db 'Ola UBI'
```

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

AX	40h = 0100 0000b	AL
BX	1 → 0000 0000 0000 0001	
CX	7 → 0000 0000 0000 0111	
DX	msg (endereço da string a escrever)	



AX	3Fh = 0011 1111b	AL
BX	1 → 0000 0000 0000 0001	
CX	7 → 0000 0000 0000 0111	
DX	msg (endereço da string a escrever)	

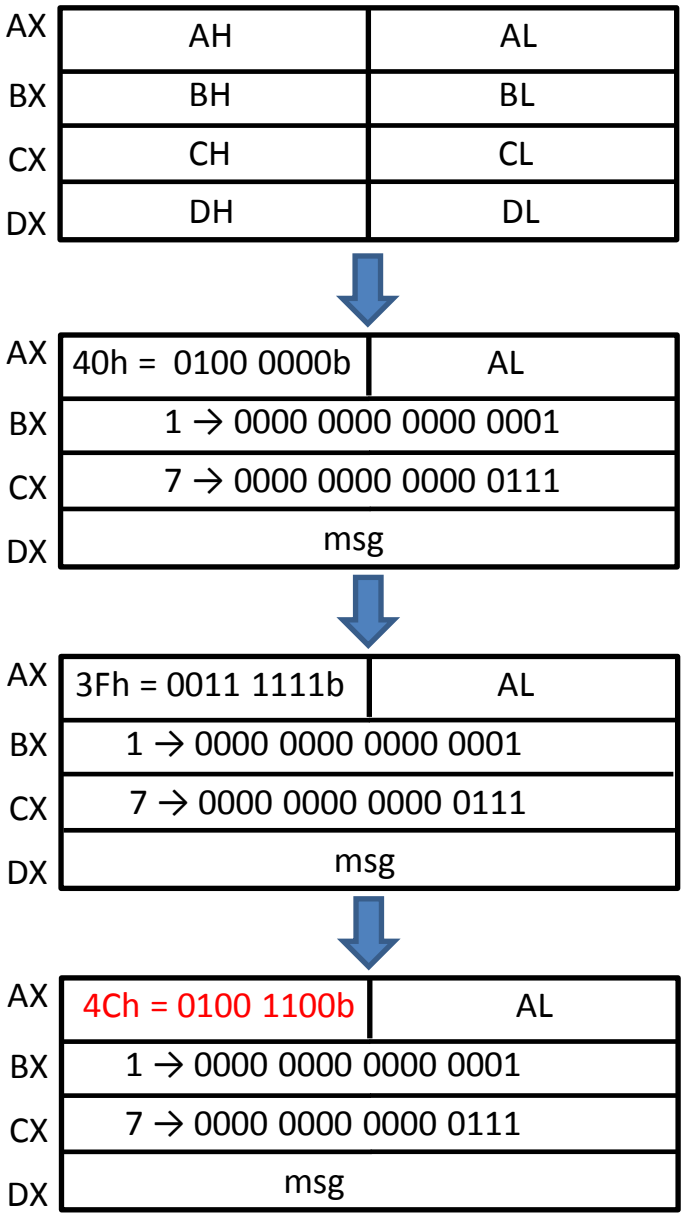
aguarda por tecla

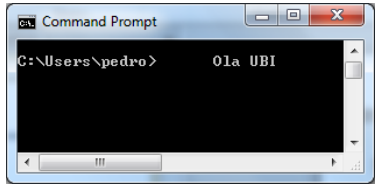
Execução dos programas

1º exemplo: escrever no ecrã a mensagem "Ola UBI" → printf("Ola UBI")

Programa Assembly

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ;escrita
mov bx, 1     ;ecran
mov cx, 7     ;nº caracteres
mov dx, msg   ;string
int  21h      ;executa

;aguarda tecla
mov ah, 3Fh  ;leitura
int  21h     ;executa

;terminar o programa
mov ah, 4Ch  ;terminar
int  21h     ;executa

msg db 'Ola UBI'
```





Execução dos programas

1º exemplo: escrever no ecrã a mensagem "Ola UBI" → printf("Ola UBI")

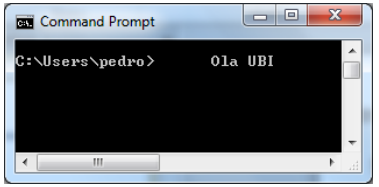
Programa Assembly

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ;escrita
mov bx, 1     ;ecran
mov cx, 7     ;nº caracteres
mov dx, msg   ;string
int  21h      ;executa

;aguarda tecla
mov ah, 3Fh  ;leitura
int  21h     ;executa

;terminar o programa
mov ah, 4Ch  ;terminar
int  21h     ;executa

msg db 'Ola UBI'
```

'Ola UBI' = 4Fh,6Ch,61h,20h,55h,42h,49h
= 01001111b, 01101100b,...

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

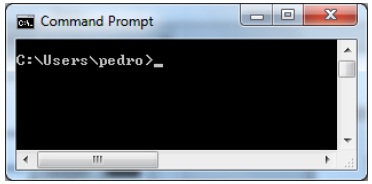
AX	40h = 0100 0000b	AL
BX	1 → 0000 0000 0000 0001	
CX	7 → 0000 0000 0000 0111	
DX	msg	



AX	3Fh = 0011 1111b	AL
BX	1 → 0000 0000 0000 0001	
CX	7 → 0000 0000 0000 0111	
DX	msg	

AX	4Ch = 0100 1100b	AL
BX	1 → 0000 0000 0000 0001	
CX	7 → 0000 0000 0000 0111	
DX	msg	

termina



Aula 4
2021-03-19

Execução das instruções (cont.)

Intel 80x86 Assembly Language OpCodes

http://www.mathemainzel.info/files/x86asmref.html#mov

MOV - Move Byte or Word
Usage: MOV dest,src
Modifies flags: none

Exemplos:

mov ah, 40h ↔ B440=
1011010001000000

mov cx, 07 ↔ B90700

técnica *little-endian* → troca entre o byte de menor peso com o de maior peso

	mnemonics	op xx xx xx xx xx	sw	len	flags
MOV	AL,rmb	A0 d0 d1	B	3	--------
MOV	AX,rmw	A1 d0 d1	W	3	--------
MOV	AL,ib	B0 i0	B	2	--------
MOV	AH,ib	B4 i0	B	2	--------
MOV	AX,iw	B8 i0 i1	W	3	--------
MOV	CL,ib	B1 i0	B	2	--------
MOV	CH,ib	B5 i0	B	2	--------
MOV	CX,iw	B9 i0 i1	W	3	--------
MOV	DL,ib	B2 i0	B	2	--------
MOV	DH,ib	B6 i0	B	2	--------
MOV	DX,iw	BA i0 i1	W	3	--------
MOV	BL,ib	B3 i0	B	2	--------
MOV	BH,ib	B7 i0	B	2	--------
MOV	BX,iw	BB i0 i1	W	3	--------

INT – Interrupt
Usage: INT num
Modifies flags: TF IF

Exemplo:
int 21h ↔ CD21

mnemonics	op xx xx xx xx xx	sw	len	flags
INT 3	CC		1	--00----
INT ib	CD i0		2	--00----

Execução dos programas

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ↔ B440=1011010001000000  ;escrita
mov bx, 1     ↔ BB0100=101110110000000100000000  ;ecran
mov cx, 7     ↔ B90700=101110010000011100000000 ;nº char
mov dx, msg   ↔ BA????=10111010???????????????? ;string
int  21h      ↔ CD21=1100110100100001 ;executa

;aguarda tecla
mov ah, 3Fh ↔ B43F=1011010000111111  ;leitura
int  21h    ↔ CD21=1100110100100001  ;executa

;terminar o programa
mov ah, 4Ch ↔ B44C=1011010001001100  ;terminar
int  21h    ↔ CD21=1100110100100001  ;executa


msg db 'Ola UBI'  ;4Fh,6Ch,61h,20h,55h,42h,49h
                  ;01001111,01101100,01100001,...
```

programa

dados

Memória

address		
1011 0100	0100	B4h
0100 0000	0101	40h
1011 1011	0102	BBh
0000 0001	0103	01h
0000 0000	0104	00h
1011 1001	0105	B9h
0000 0111	0106	07h
0000 0000	0107	00h
1011 1010	0108	BAh
?	0109	?
?	010A	?
1100 1101	010B	CDh
0010 0001	010C	21h
1011 0100	010D	B4h
0011 1111	010E	3Fh
1100 1101	010F	CDh
0010 0001	0110	21h
1011 0100	0111	B4h
0100 1100	0112	4Ch
1100 1101	0113	CDh
0010 0001	0114	21h
0100 1111	0115	'O' ← msg
0110 1100	0116	'l'
0110 0001	0117	'a'
0010 0000	0118	
0101 0101	0119	'U'
0100 0010	011A	'B'
0100 1001	011B	'I'
.....	.....	
	M	

Big-endian: os bytes são armazenados da esquerda(maior byte) para a direita (menor byte)

Little-endian: os bytes são armazenados da direita(menor byte) para a esquerda(maior byte) – INTEL x86

Execução dos programas

```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ↔ B440      ;escrita
mov bx, 1     ↔ BB0100    ;ecran
mov cx, 7     ↔ B90700    ;nº char
mov dx, msg   ↔ BA1501    ;string
int  21h      ↔ CD21      ;executa

;aguarda tecla
mov ah, 3Fh ↔ B43F  ;leitura
int  21h    ↔ CD21  ;executa

;terminar o programa
mov ah, 4Ch ↔ B44C  ;terminar
int  21h    ↔ CD21  ;executa

msg db 'Ola UBI'  ;[4F,6C,61,20,55,42,49]
                    códigos ASCII
```

programa

dados

Memória

	address		
little-endian	1011 0100	0100	B4h
	0100 0000	0101	40h
	1011 1011	0102	BBh
	0000 0001	0103	01h
	0000 0000	0104	00h
little-endian	1011 1001	0105	B9h
	0000 0111	0106	07h
	0000 0000	0107	00h
little-endian	1011 1010	0108	BAh
	0001 1001	0109	15h
	0000 0001	010A	01h
	1100 1101	010B	CDh
	0010 0001	010C	21h
	1011 0100	010D	B4h
	0011 1111	010E	3Fh
	1100 1101	010F	CDh
	0010 0001	0110	21h
	1011 0100	0111	B4h
	0100 1100	0112	4Ch
	1100 1101	0113	CDh
	0010 0001	0114	21h
	0100 1111	0115	'O' ← msg
	0110 1100	0116	'l'
	0110 0001	0117	'a'
	0010 0000	0118	
	0101 0101	0119	'U'
	0100 0010	011A	'B'
	0100 1001	011B	'I'
	.....	.....	
		M	

Big-endian: os bytes são armazenados da esquerda(maior byte) para a direita (menor byte)

Little-endian: os bytes são armazenados da direita(menor byte) para a esquerda(maior byte) – INTEL x86

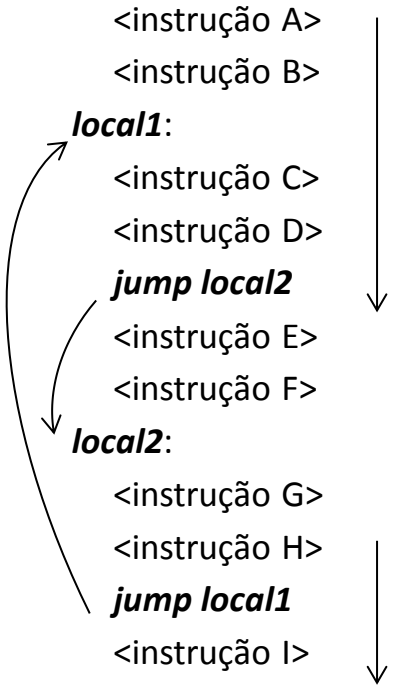
Execução dos programas

Instruções de salto (jump) : alteram a sequência de execução das instruções

- permitem programar ciclos e tomada de decisões em função de determinadas condições

jump incondicional – não depende de nenhuma condição para saltar

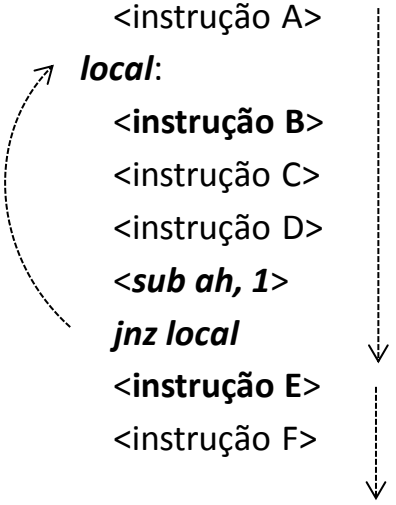
ex: *jump local* → ao encontrar esta instrução o programa continua a partir de "*local*"



jump condicional – depende de uma condição para saltar

ex: *sub ah, 1* ;subtrai 1 ao registo AH

jnz local → (jump if not zero) : se AH≠0 o programa continua a partir de "*local*"(instrução B), senão continua na instrução seguinte *(*instrução E*)*

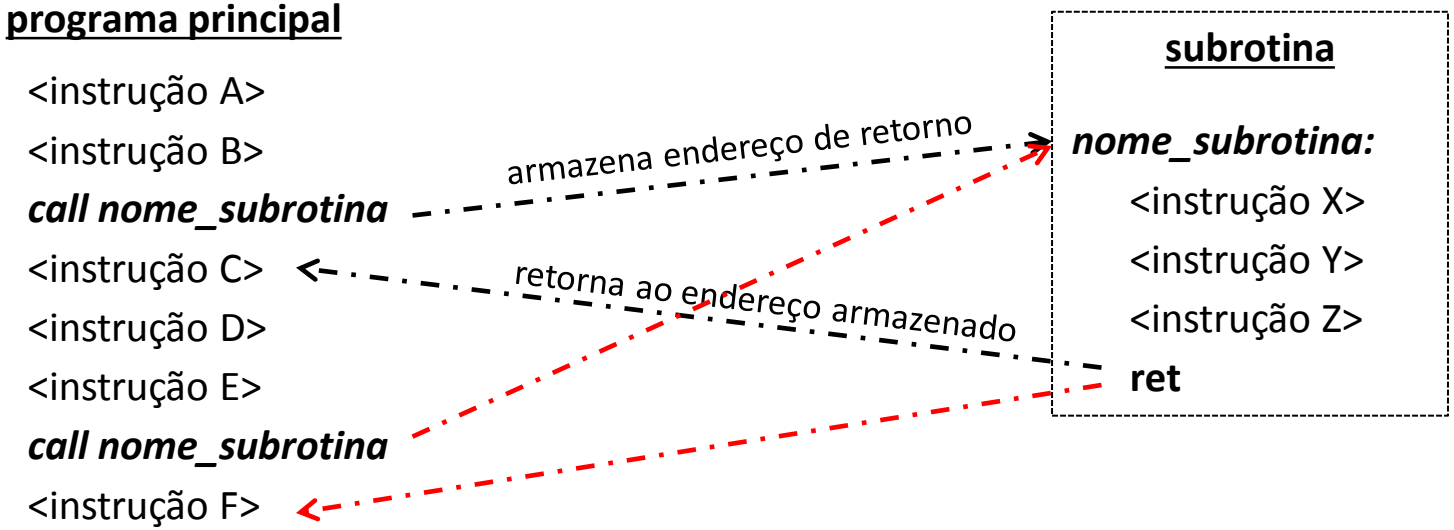


Sub-rotinas

Os ciclos permitem repetir um bloco de instruções num certo local do programa. Mas por vezes esse bloco precisa de ser usado várias vezes e em diferentes locais do programa, para isso:

- podem usar-se *subrotinas* que funcionam como pequenos programas dentro do programa principal;
- as subrotinas são chamadas (call) a partir de qualquer local do programa principal, executam a sua função e retornam (ret) devolvendo o controlo ao programa principal;
- torna-se necessário armazenar o local de onde a subrotina é chamada para que quando esta terminar o programa retome a execução a partir daí;

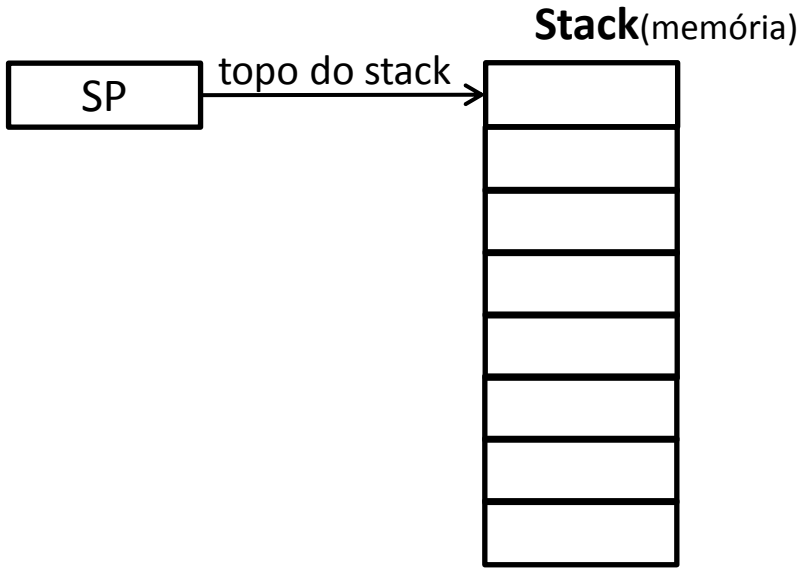
ex: na sequência abaixo, ao executar a primeira instrução *call nome_subrotina*, o programa armazena o endereço de <instrução C> que representa o endereço de retorno da subrotina e em seguida executa o corpo desta; a instrução *ret* faz o programa retomar a execução a partir do endereço que foi previamente armazenado.
Na segunda chamada (a vermelho)o endereço de retorno é o da <instrução F> .



Uso do stack (pilha)

O stack é uma estrutura de dados que serve para armazenar valores de forma temporária.

É constituído por uma zona reservada de memória, cujos endereços são referenciados através de um registo especial designado SP-*stack pointer*.



O registo SP aponta para o próximo endereço livre, designado por *topo do stack*, o qual habitualmente vai diminuindo (descendo) à medida que a pilha vai contendo mais dados.

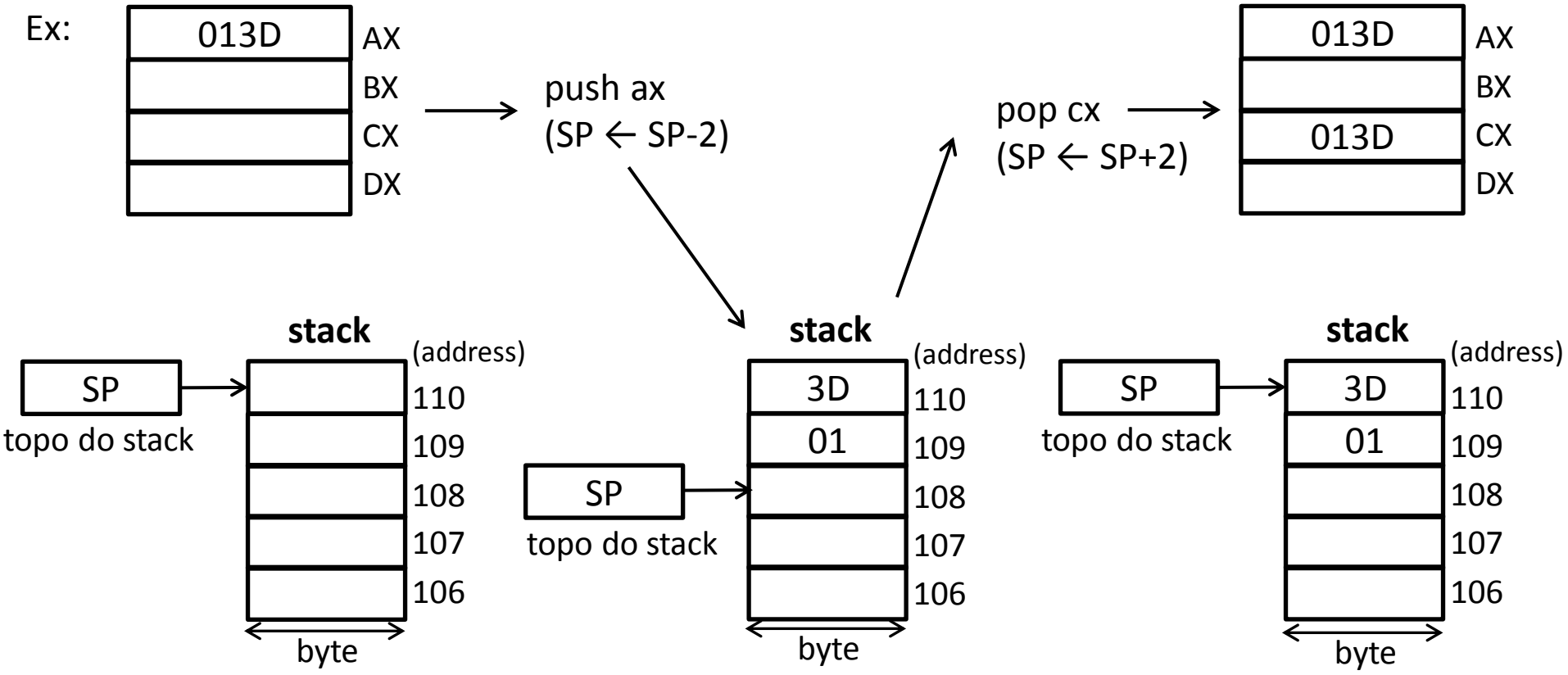
Além de permitir armazenar variáveis o stack também é usado para armazenar o endereço de retorno das sub-rotinas.

Uso do stack

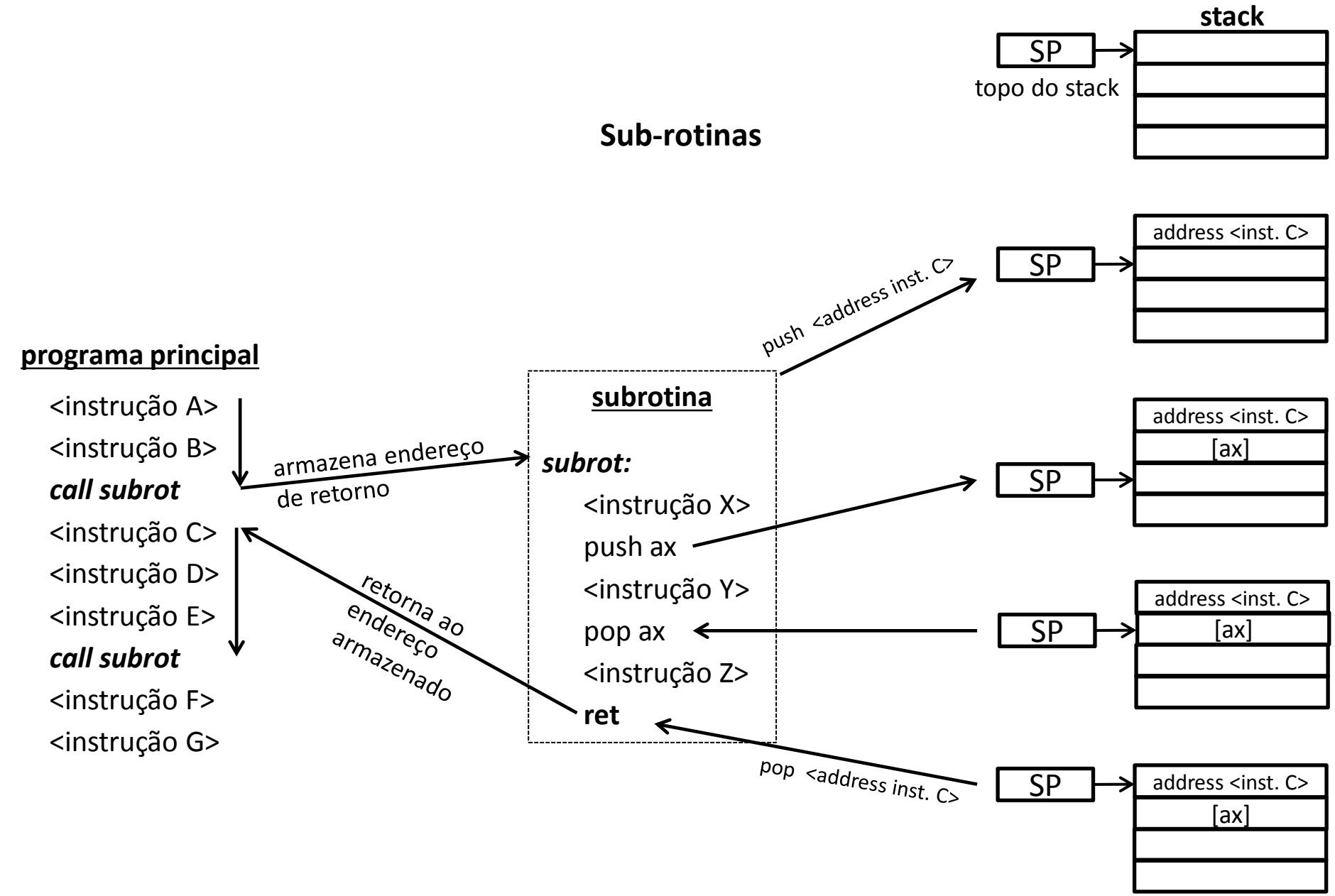
Instruções de manipulação do stack (16 bit):

- *push* <val> : insere um valor no topo do stack
  <val> pode ser um valor imediato ou o conteúdo de um registo;

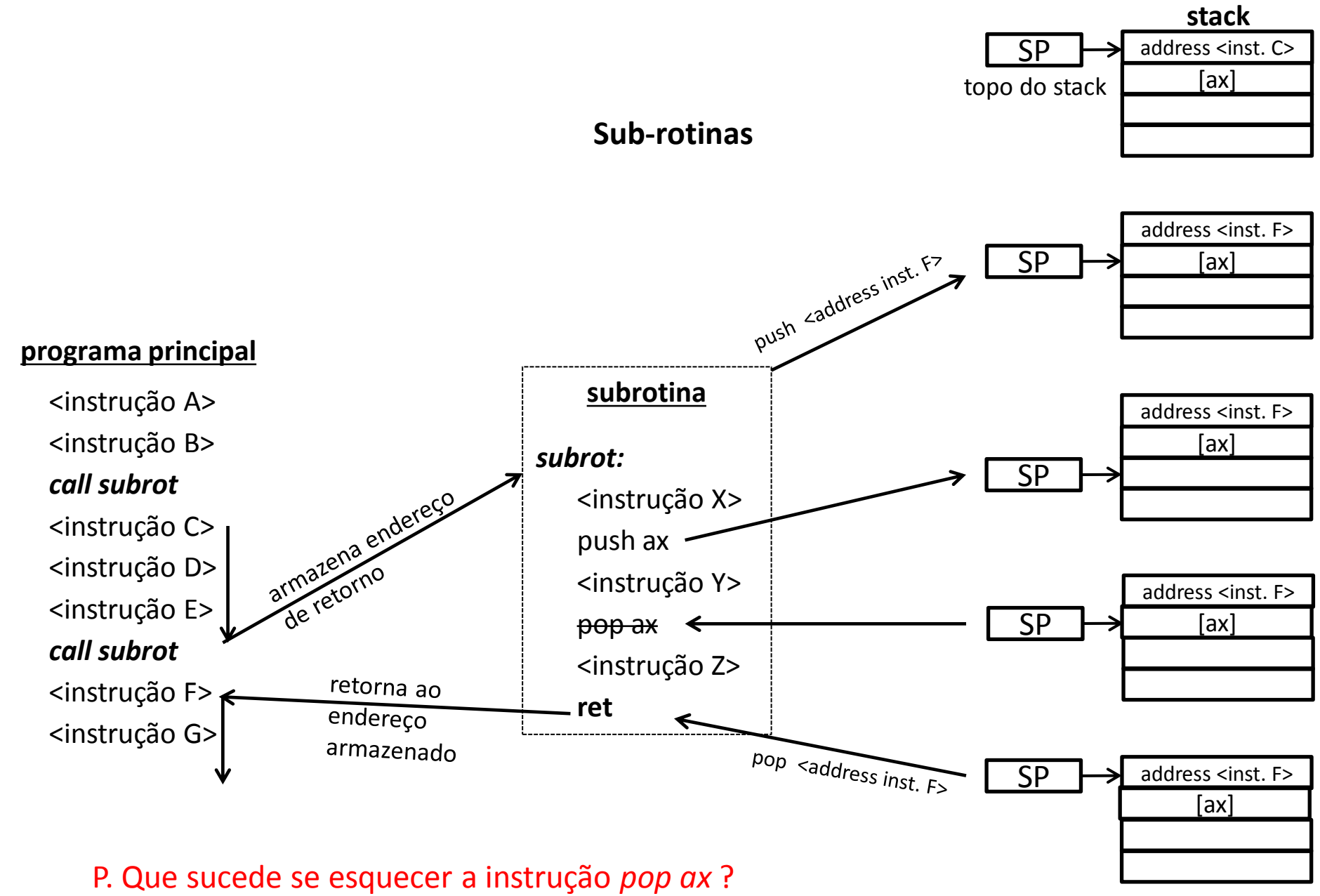
- *pop* <local> : retira o valor que está no topo do stack e coloca-o em <local> , este pode ser uma variável ou um registo;



Uso do stack


stack
SP
topo do stack
Sub-rotinas
programa principal
<instrução A>
<instrução B>
call subrot
<instrução C>
<instrução D>
<instrução E>
call subrot
<instrução F>
<instrução G>
armazena endereço de retorno
subrotina
subrot:
<instrução X>
push ax
<instrução Y>
pop ax
<instrução Z>
ret
retorna ao endereço armazenado
push <address inst. C>
pop <address inst. C>
address <inst. C>
[ax]

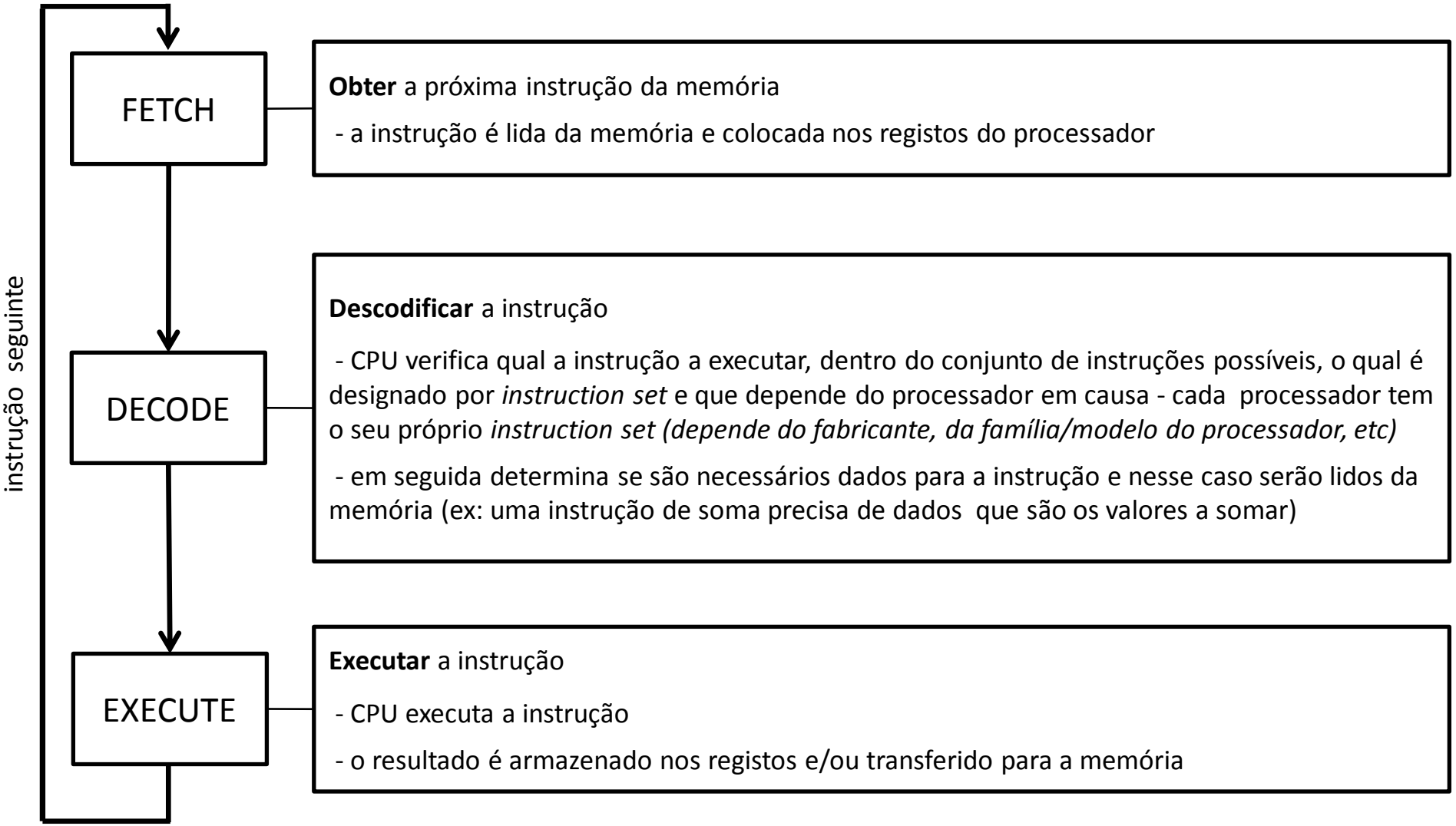
Uso do stack



P. Que sucede se esquecer a instrução *pop ax* ?

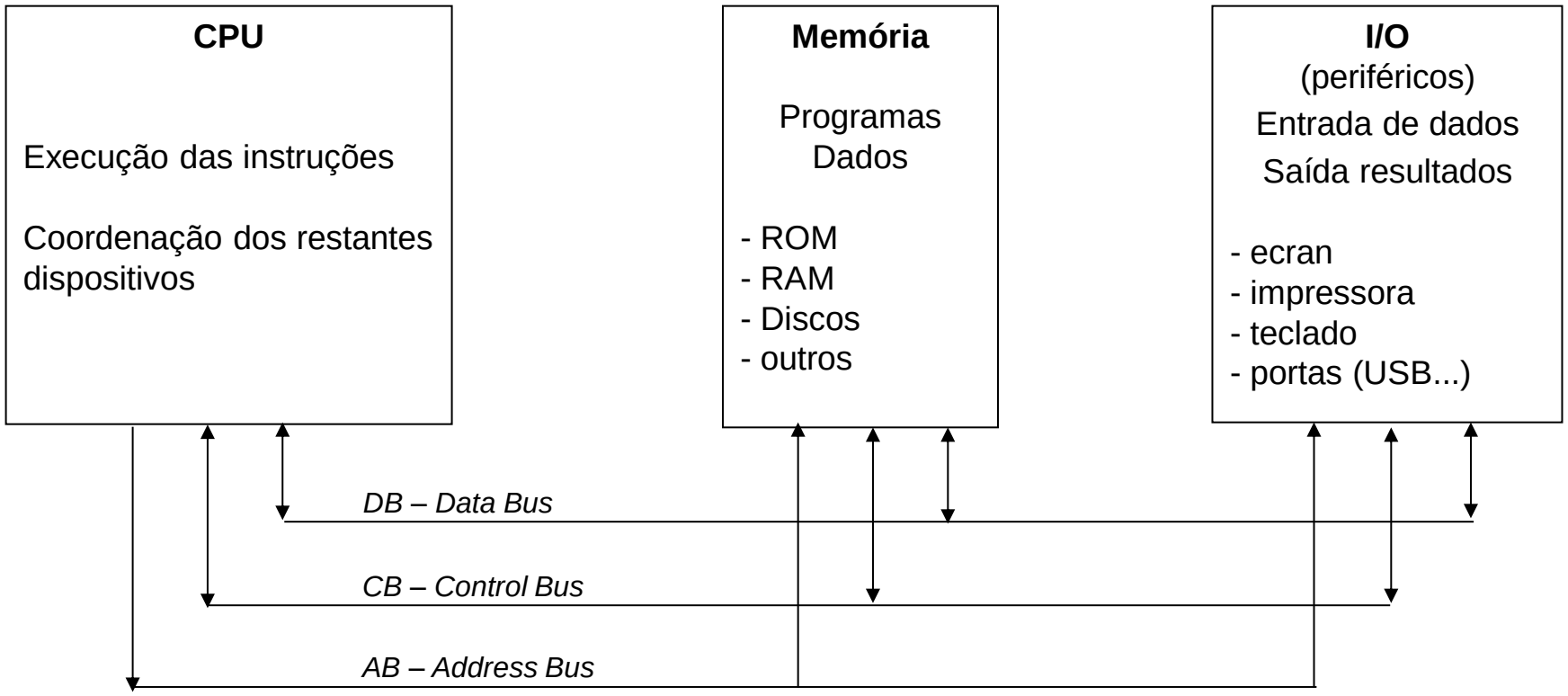
Ciclo: FETCH – DECODE - EXECUTE

O CPU está permanentemente a executar instruções que podem provir de aplicações ou do sistema operativo (Windows, ...). As instruções são executadas segundo a sequência FETCH-DECODE-EXECUTE (embora cada uma destas componentes possa ainda ser decomposta em outras mais simples)



CPU – MEMÓRIA – PERIFÉRICOS(Input/Output)

A interligação entre os vários componentes faz-se através de três canais de dados, designados por *buses*



Bus (barramento) : conjunto de linhas de comunicação que interligam os vários componentes de um sistema de computação. Principais características: largura(nº de bits), velocidade de transmissão(bps-bits por segundo)

DB(Data Bus) – caminho dos dados, bidireccional. (ex: Pentium IV: 64/128 bits externos, 32/64 internos; 3.2GB/s)

CB(Control Bus) – bidireccional, sinais de controlo. (ex: Read, Write, Reset)

AB(Address Bus) – unidireccional, sinais de endereço (ex: Pentium IV: 32 bits/4GB MEM ; 36bits/64GB)

Endereçamento

Como selecionar um de entre vários dispositivos (ex: ler da memória ou de um periférico) ?

Através do mecanismo de endereçamento → só o periférico endereçado fica ativo

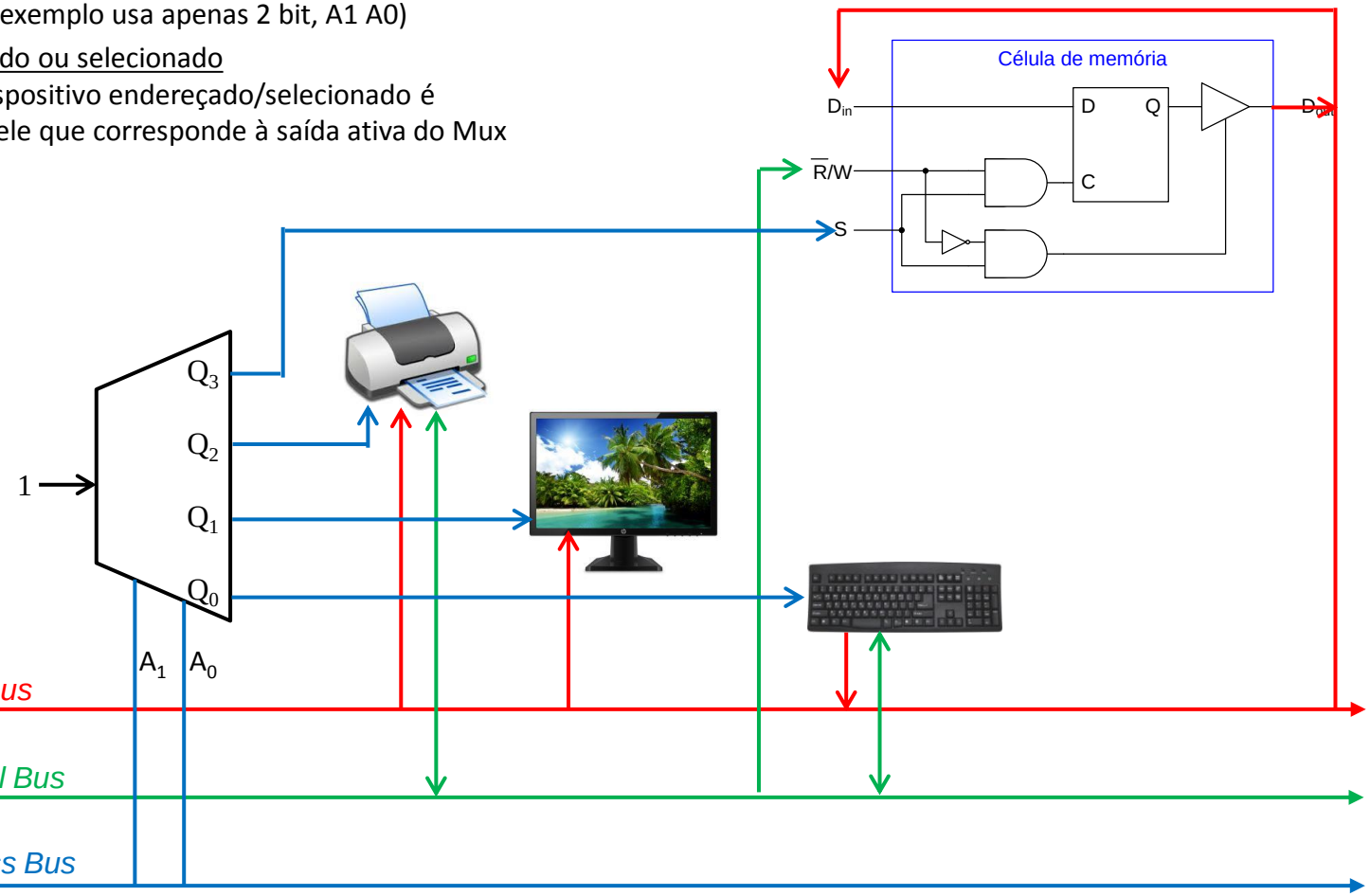
O AB (Address Bus) é usado para enviar o endereço do dispositivo a selecionar

Ex: imaginemos um CPU capaz de aceder apenas a 4 dispositivos (teclado , ecran, impressora, memória)

Address Bus (neste exemplo usa apenas 2 bit, A1 A0)

A1	A0	Dispositivo endereçado ou selecionado
0	0	Teclado
0	1	Ecran
1	0	Impressora
1	1	Memória

o dispositivo endereçado/selecionado é aquele que corresponde à saída ativa do Mux



Aula 5
2021-03-26

Métrica binária

Agrupamentos de n bits

n	gama=2^n	designação	nº de bytes
1	2 : (0 , 1)	bit	-
4	16 : (0 .. 15)	nibble	-
8	256 : (0 .. 255)	byte (octeto)	1
10	1024 : (0 .. 1023)	Kbit	-
16	65536 : (0 .. 65535)	Palavra 16 bits (word)	2
32	2^{32} : (0 .. 2^{32}-1)	Palavra 32 bits	4
64	2^{64} : (0 .. 2^{64}-1)	Palavra 64 bits	8

bit = **b**inary digi**t**

byte = **b**inar**y** **te**rm

K=2^{10}=1024

NOTAS:

- em decimal o prefixo K significa 1000(=10^3); em binário o prefixo K significa 1024(=2^{10})
- a definição de "word" varia consoante a máquina usada, máquinas com registos de 16 bit →word=16 bit; máquinas com registos de 32 bit → word=32 bit

Agrupamentos de bytes

binário	gama	unidade	decimal aproximado
2^{10}	1024 byte	K (Kilo) - Kbyte	10^3
2^{20}	1024 * K = 1 048 576 byte	M (Mega) - Mbyte	10^6
2^{30}	1024 * M = 1 073 741 824 byte	G (Giga) - Gbyte	10^9
2^{40}	1024 * G = 1 099 511 627 776 byte	T (Tera) - Tbyte	10^{12}

K=2^{10}=1024

Métrica binária

Correspondência de números em diversas bases

Decimal (pesos) 10^1 10^0	**Binário** (pesos) 2^3 2^2 2^1 2^0 8 4 2 1	**Hexadecimal** (pesos) 16^0
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
1 0	1 0 1 0	A
1 1	1 0 1 1	B
1 2	1 1 0 0	C
1 3	1 1 0 1	D
1 4	1 1 1 0	E
1 5	1 1 1 1	F

nibble(4 bit)

1 byte = 2 nibble = 2 digitos hexadecimais ex: E4h = 1110 0100b

Métrica Binária

A norma IEC 80000-13: *Quantities and units – Part 13: Information science and technology* , publicada em 2008 define os seguintes prefixos binários:

Nome	Símbolo	Potência = valor
kibi	Ki	$2^{10} = 1024$
mebi	Mi	$2^{20} = 1\,048\,576$
gibi	Gi	$2^{30} = 1\,073\,741\,824$
tebi	Ti	$2^{40} = 1\,099\,511\,627\,776$
pebi	Pi	$2^{50} = 1\,125\,899\,906\,842\,624$
exbi	Ei	$2^{60} = 1\,152\,921\,504\,606\,846\,976$
zebi	Zi	$2^{70} = 1\,180\,591\,620\,717\,411\,303\,424$
yobi	Yi	$2^{80} = 1\,208\,925\,819\,614\,629\,174\,706\,176$

Prefixos binários segundo a norma IEC 80000-13 (2008).

Consultar: Prefixes for binary multiples - http://physics.nist.gov/cuu/Units/binary.html

Exemplo: 1 KByte = 10^3 Byte = 1000 Byte (Kilo Byte, decimal)
1 KiByte = 2^{10} Byte = 1024 Byte (Kibi Byte, binário)

Tabela ASCII - American Standard Code for Information Interchange

Relaciona os carateres com a sua representação numérica (decimal, hexadecimal, binária)

Tabela ASCII (7bits)

Dec	Hex	Char		Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00h	^@	Null	32	20h		64	40h	@	96	60h	`
1	01h	☺	^A SOH-Start of Header	33	21h	!	65	41h	A	97	61h	a
2	02h	☻	^B STX- Start of Text	34	22h	"	66	42h	B	98	62h	b
3	03h	♥	^C ETX- End of Text	35	23h	#	67	43h	C	99	63h	c
4	04h	♦	^D EOT- End of Transmission	36	24h	$	68	44h	D	100	64h	d
5	05h	♣	^E ENQ- Enquiry	37	25h	%	69	45h	E	101	65h	e
6	06h	♠	^F ACK- Acknowledgment	38	26h	&	70	46h	F	102	66h	f
7	07h	•	^G BEL- Bell	39	27h	'	71	47h	G	103	67h	g
8	08h	◘	^H BS- Backspace	40	28h	(	72	48h	H	104	68h	h
9	09h	○	^I HT-Horizontal Tab	41	29h	)	73	49h	I	105	69h	i
10	0Ah	◙	^J LF-Line Feed	42	2Ah	*	74	4Ah	J	106	6Ah	j
11	0Bh	♂	^K VT-Vertical Tab	43	2Bh	+	75	4Bh	K	107	6Bh	k
12	0Ch	♀	^L FF-Form Feed	44	2Ch	,	76	4Ch	L	108	6Ch	l
13	0Dh	♪	^M CR-Carriage Return	45	2Dh	-	77	4Dh	M	109	6Dh	m
14	0Eh	♫	^N SO-Shift Out	46	2Eh	.	78	4Eh	N	110	6Eh	n
15	0Fh	☼	^O SI- Shift In	47	2Fh	/	79	4Fh	O	111	6Fh	o
16	10h	►	^P DLE- Data Link Escape	48	30h	0	80	50h	P	112	70h	p
17	11h	◄	^Q DC1- (XON) Device Control1	49	31h	1	81	51h	Q	113	71h	q
18	12h	↕	^R DC2- Device Control2	50	32h	2	82	52h	R	114	72h	r
19	13h	‼	^S DC3- (XOFF) Device Control3	51	33h	3	83	53h	S	115	73h	s
20	14h	¶	^T DC4- Device Control4	52	34h	4	84	54h	T	116	74h	t
21	15h	§	^U NAK- Negative Acknowledge	53	35h	5	85	55h	U	117	75h	u
22	16h	▬	^V SYN- Synchronous Idle	54	36h	6	86	56h	V	118	76h	v
23	17h	↨	^W ETB- End of Trans. Block	55	37h	7	87	57h	W	119	77h	w
24	18h	↑	^X CAN- Cancel	56	38h	8	88	58h	X	120	78h	x
25	19h	↓	^Y EM- End of Medium	57	39h	9	89	59h	Y	121	79h	y
26	1Ah	→	^Z SUB- Substitute	58	3Ah	:	90	5Ah	Z	122	7Ah	z
27	1Bh	←	^[ESC- Escape	59	3Bh	;	91	5Bh	[	123	7Bh	{
28	1Ch	└	^\ FS- File Separator	60	3Ch	<	92	5Ch	\	124	7Ch	|
29	1Dh	↔	^] GS- Group Separator	61	3Dh	=	93	5Dh	]	125	7Dh	}
30	1Eh	▲	^^ RS- Record Separator	62	3Eh	>	94	5Eh	^	126	7Eh	~
31	1Fh	▼	^_ US- Unit Separator	63	3Fh	?	95	5Fh	_	127	7Fh	⌂

Exemplos

espaço = 32=20h=00100000b

'A' = 65=41h=01000001b

'z' = 122=7Ah=01111010b

Decimal (pesos) 10^1 10^0	**Binário** (pesos) 2^3 2^2 2^1 2^0 8 4 2 1	**Hexadecimal** (pesos) 16^0
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
10	1 0 1 0	A
11	1 0 1 1	B
12	1 1 0 0	C
13	1 1 0 1	D
14	1 1 1 0	E
15	1 1 1 1	F

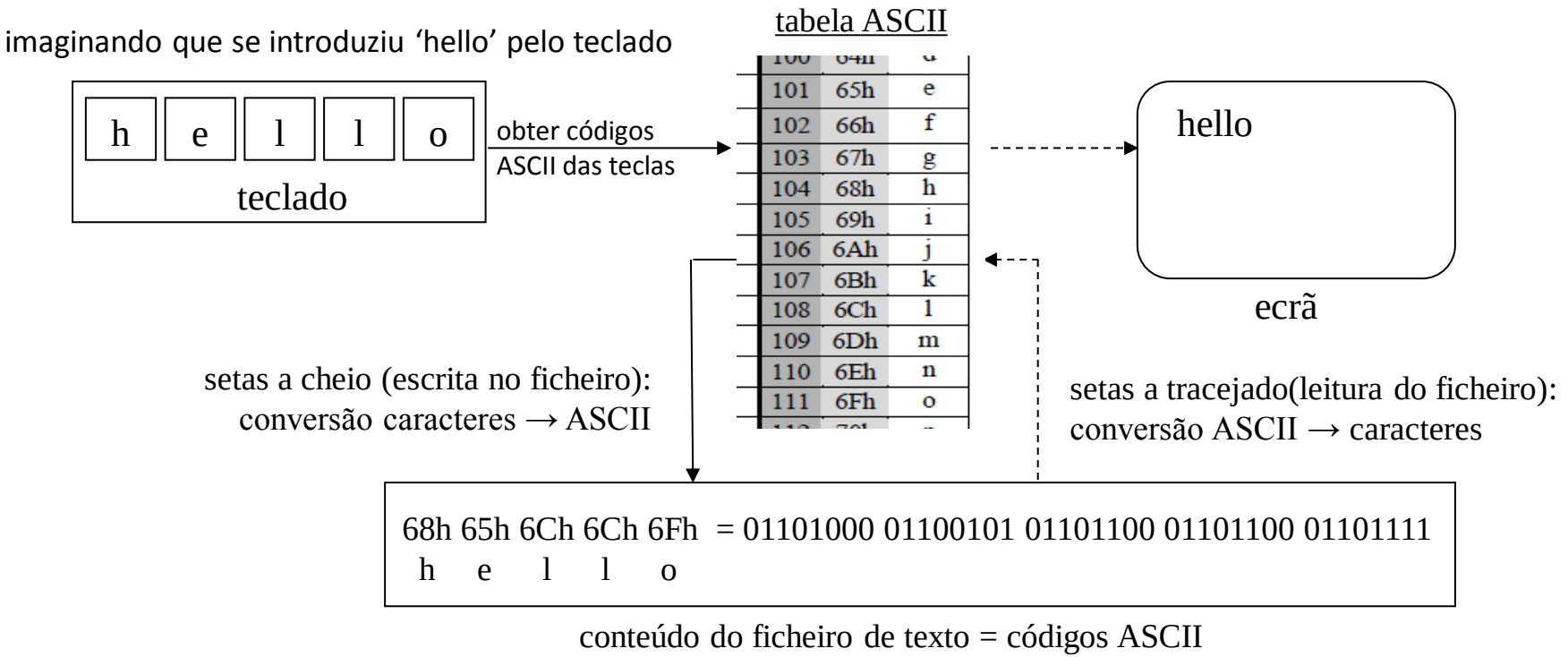
Armazenamento da Informação

(formato de **texto** , formato de números)

Texto

Escrita → para cada símbolo alfanumérico, sinal de pontuação, etc, é obtido o respetivo código da tabela ASCII, o qual é armazenado na memória ou ficheiro;

Leitura → para cada código ASCII lido da memória ou ficheiro, é obtido o respetivo símbolo alfanumérico o qual é apresentado no ecrã;



Armazenamento da Informação

Nota

Um editor de texto como o Notepad ou o Word trata os ficheiros como sendo sequências de caracteres ASCII, apresentando no ecran cada carácter e não o seu código.

Por exemplo: se usarmos o Notepad para escrever a letra 'A' dentro de um ficheiro de texto, o que fica efetivamente gravado no ficheiro é o código 41h (ver a tabela ASCII).

Quando abrimos o ficheiro o que o Notepad mostra é o 'A' e não o código 41h.

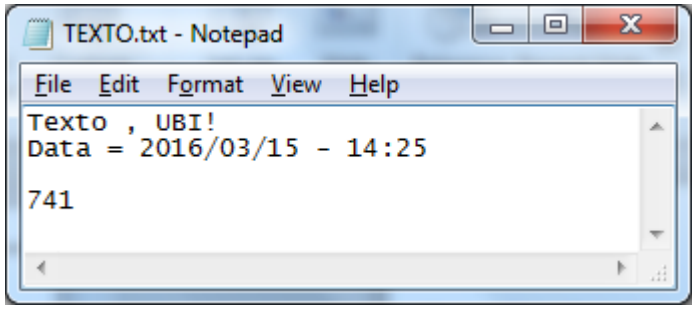
Se pretendermos ver estes códigos podemos usar um editor hexadecimal como o MiTeC HexEdit disponível em http://www.mitec.cz/hex.html (site com muitas outras ferramentas)

Nada como experimentar...

Armazenamento da Informação

Texto (letras, algarismos, sinais de pontuação)

usando o editor de texto Notepad → ficheiro TEXTO.txt



usando o editor hexadecimal "MiTeC- HexEdit" → ficheiro TEXTO.txt

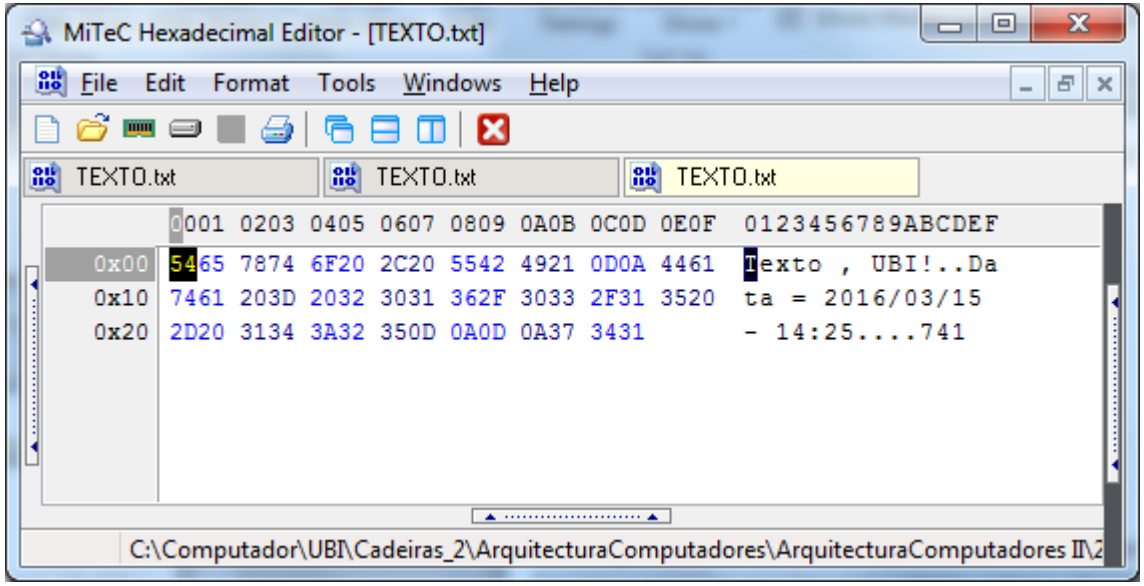


Tabela ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
32	20h		64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_	127	7Fh	⌂

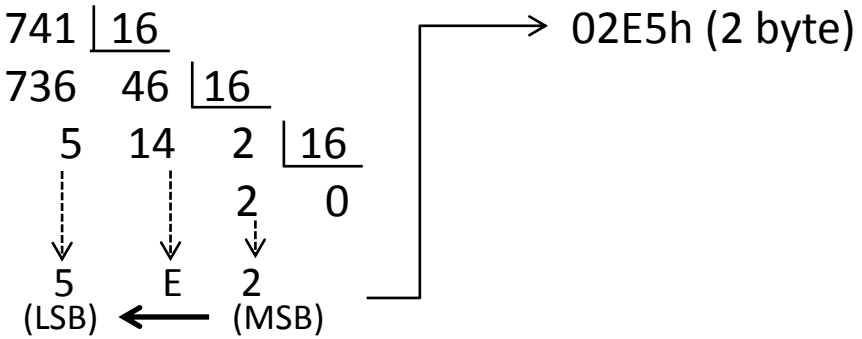
Armazenamento da Informação

(formato de texto , formato de números)

Valores numéricos (inteiros)

Como é armazenado o valor decimal 741?

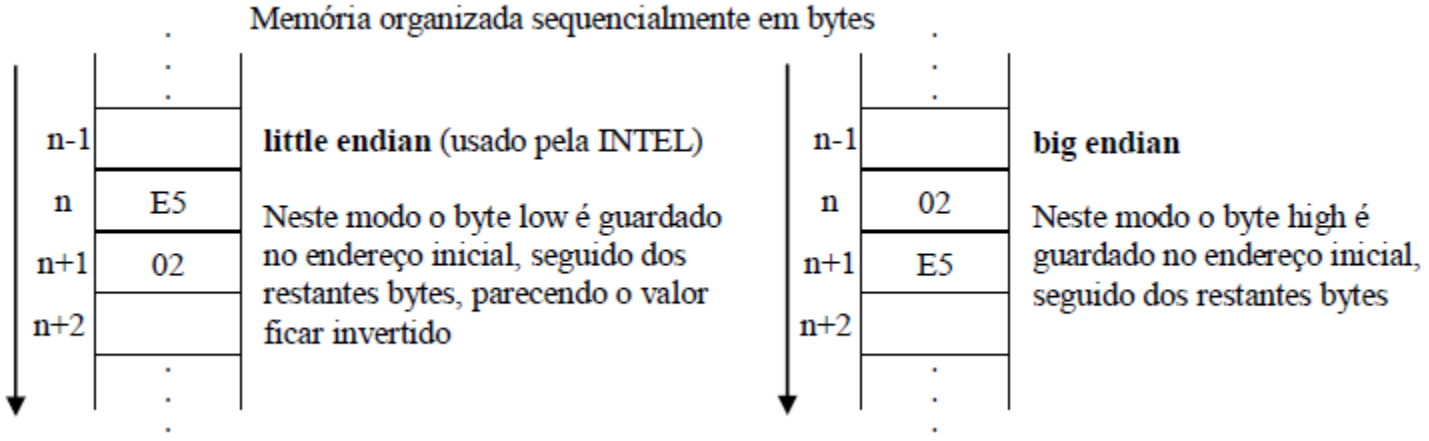
conversão para binário (hexadecimal) →



memória → conjunto de bytes organizados sequencialmente

armazenamento de conjuntos de n bytes → dois modos possíveis:

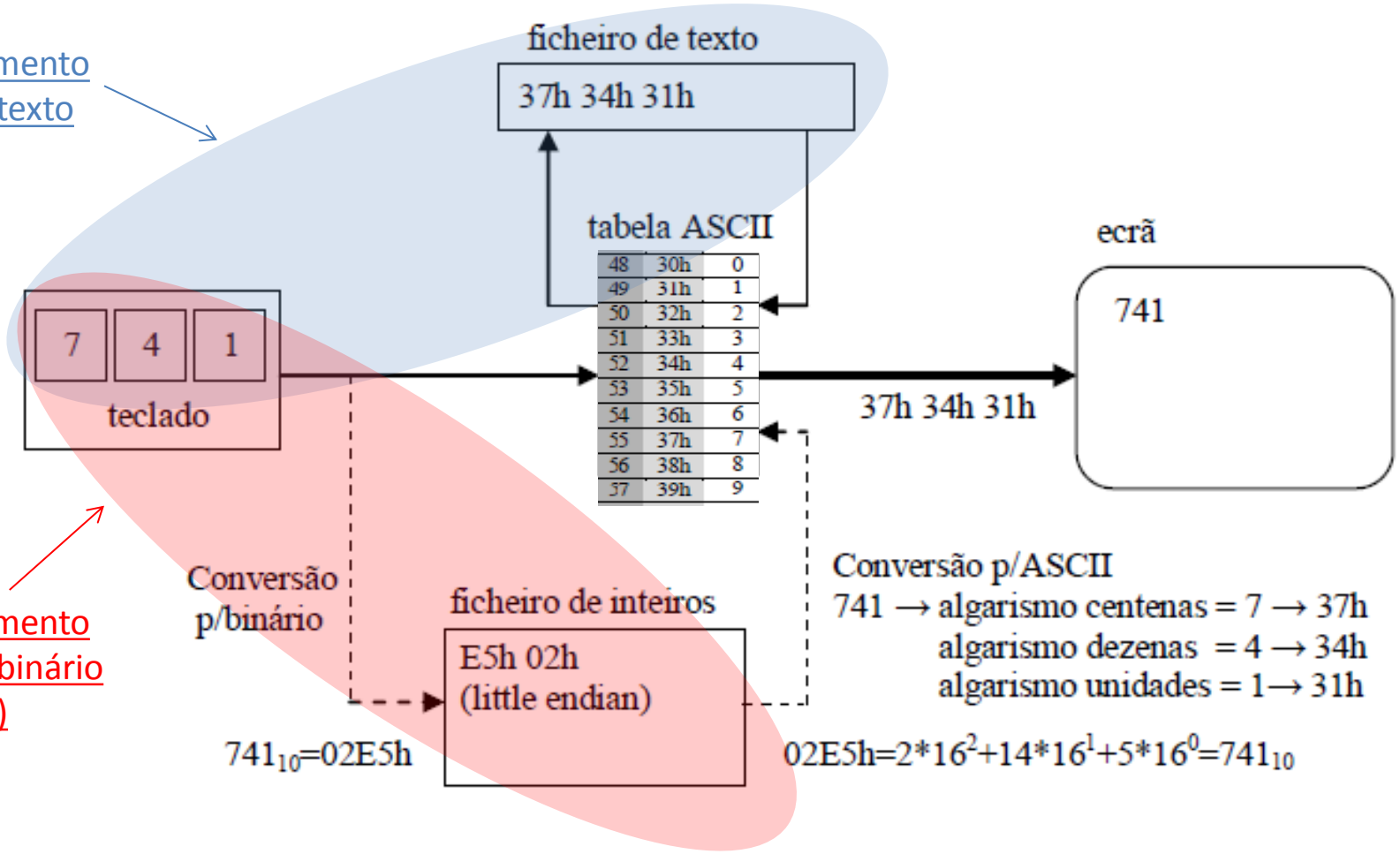
- *little endian* : byte de menor peso primeiro (usado pela Intel, linguagem C, C#, etc)
- *big endian* : byte de maior peso primeiro (usado pelo Java)



O termo *endian* tem origem no livro "As viagens de Gulliver" e refere-se à questão de qual dos lados os ovos cozidos devem ser quebrados.

Armazenamento da Informação
(texto , números)

Diferença entre armazenamento em modo texto (ASCII) ou em modo binário (hexadecimal)



setas a cheio: tratamento de texto (cadeias de caracteres ASCII) → os programas de processamento de texto (NotePad, Word,...) interpretam dados em código ASCII;

setas a tracejado: tratamento de valores numéricos em binário (inteiros);

Armazenamento da Informação

1) <u>Valores numéricos</u>: ficheiro de inteiros com o valor 741

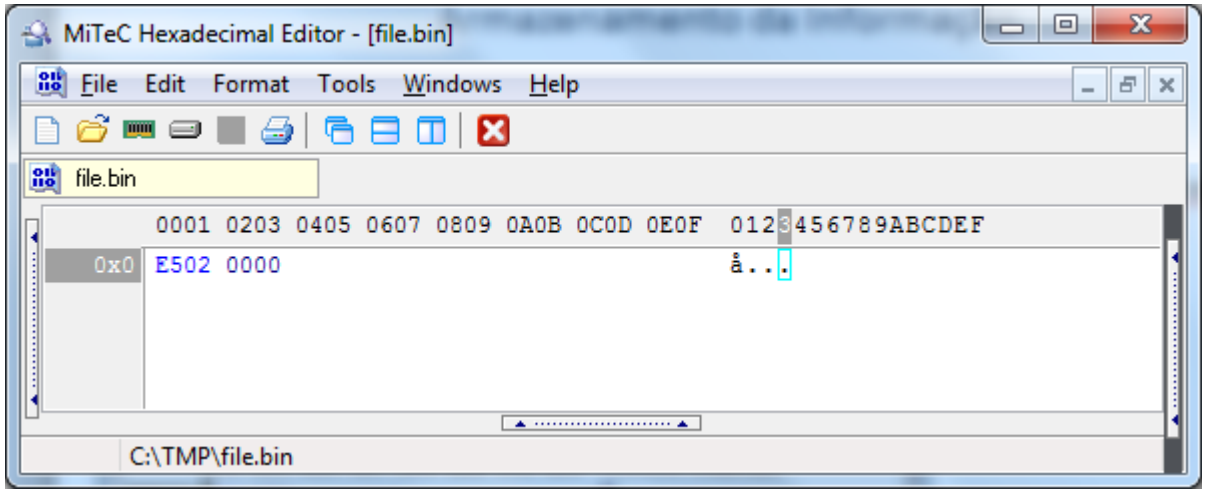
C# → using (BinaryWriter b = new BinaryWriter(File.Open("file.bin", FileMode.Create))) { b.Write(741); }

int → 32 bits(4 bytes) com sinal , -2.147.483.648 ↔ 2.147.483.647

741_{10} = 00 00 02 E5h (4 byte=32bit)

MSB LSB

MSB-Most Significant Byte LSB-Least Significant Byte



C# → little-endian
primeiro é guardado o byte de menor peso (LSB)

https://betterexplained.com/articles/understanding-big-and-little-endian-byte-order/

Armazenamento da Informação

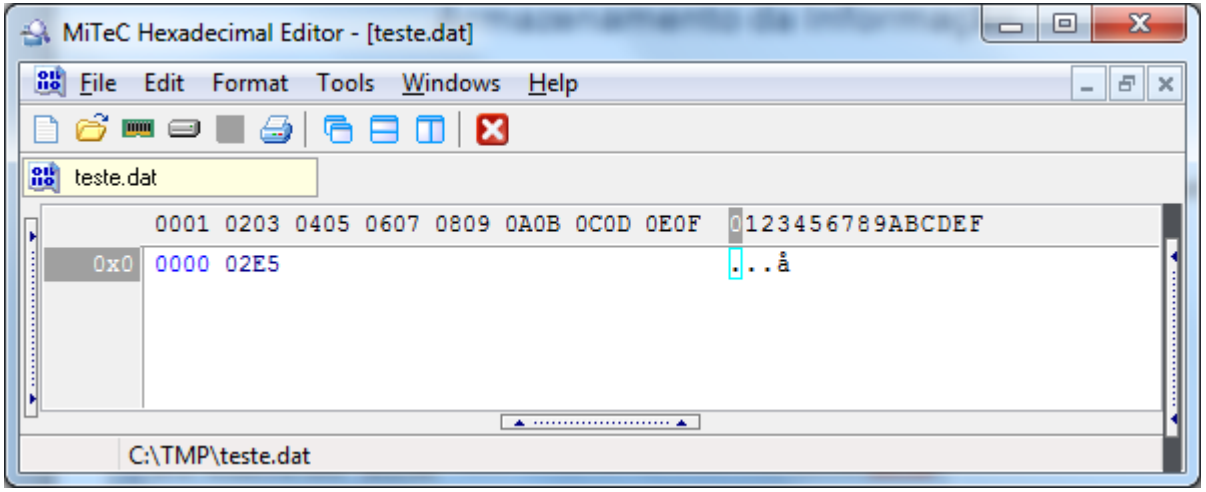
2) <u>Valores numéricos</u>: ficheiro de inteiros com o valor 741

Java → FileOutputStream os = new FileOutputStream (new File ("teste.dat"), true);
DataOutputStream dos = new DataOutputStream (os);
dos.writeInt(741);

int → 32 bits(4 bytes) com sinal , -2.147.483.648 ↔ 2.147.483.647

741_{10} = 00 00 02 E5h (4 byte=32bit)
MSB LSB

MSB-Most Significant Byte LSB-Least Significant Byte



Java → big-endian
primeiro é guardado o byte de maior peso (MSB)

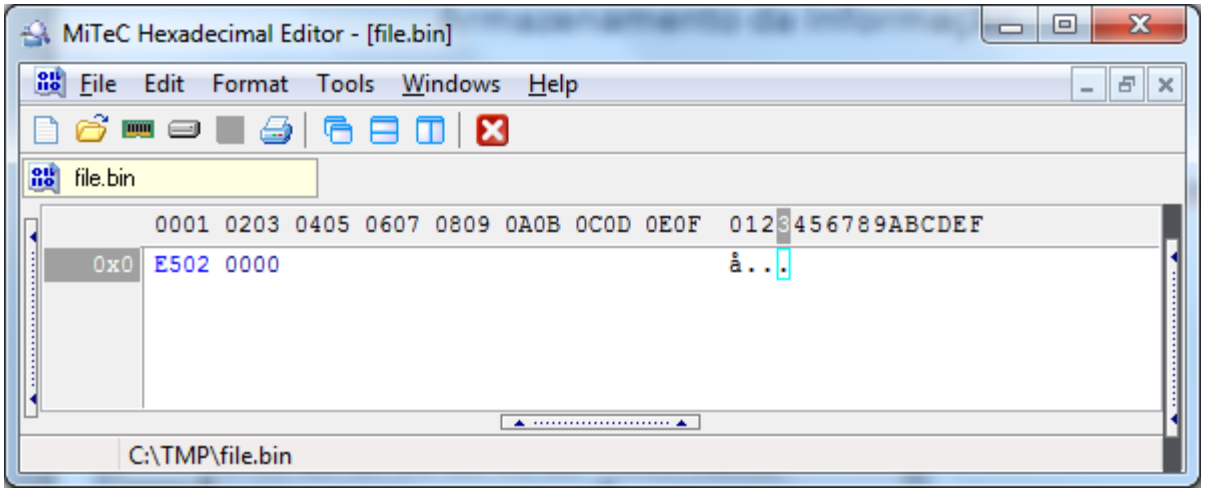
http://mindprod.com/jgloss/endian.html
http://howtodoinjava.com/core-java/basics/little-endian-and-big-endian-in-java/

Valores numéricos: ficheiro de inteiros com o valor 741

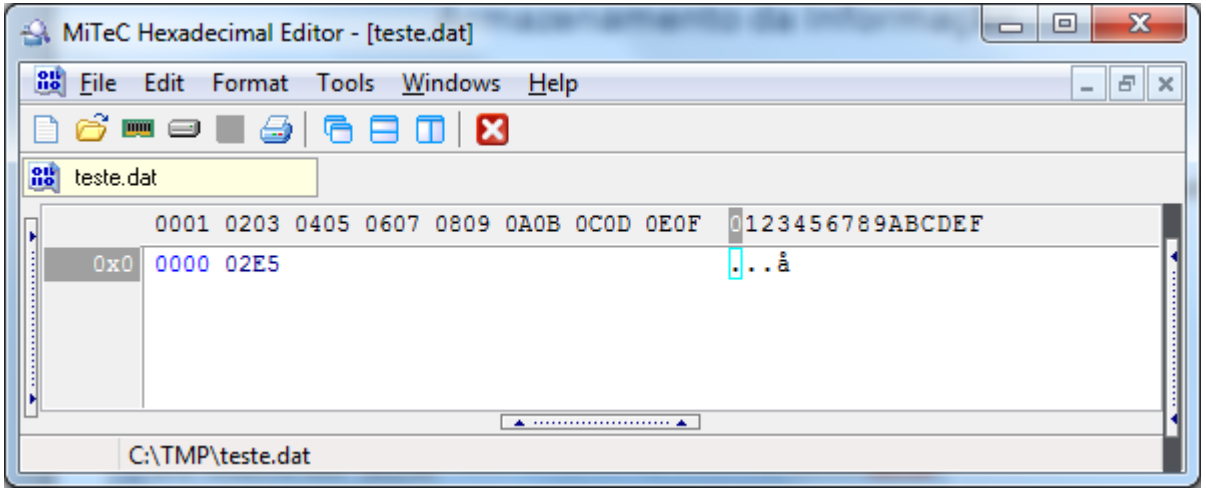
741_{10} = 00 00 02 E5h (4 byte=32bit)

MSB LSB

MSB-Most Significant Byte LSB-Least Significant Byte



C# → little-endian
primeiro é guardado o byte de menor peso (LSB)

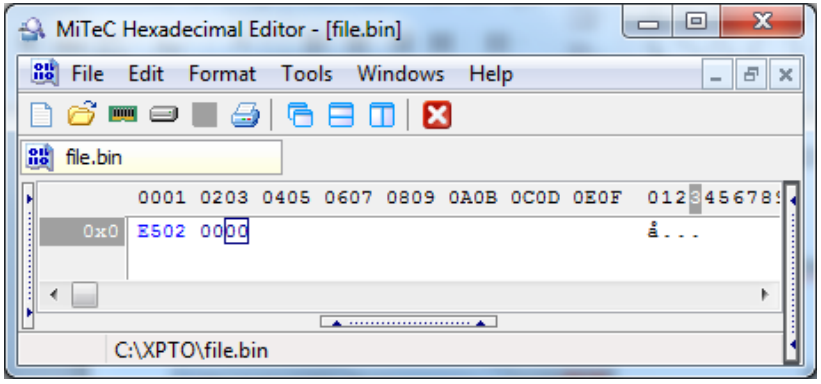


Java → big-endian
primeiro é guardado o byte de maior peso (MSB)

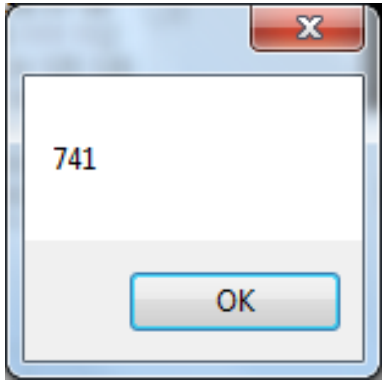
Armazenamento da Informação

1) Ler com C# o ficheiro escrito em C# contendo o valor inteiro 741 ?

C# → using (BinaryReader b = new BinaryReader(File.Open("file.bin", FileMode.Open)))
int v = b.ReadInt32();
MessageBox.Show(v.ToString());

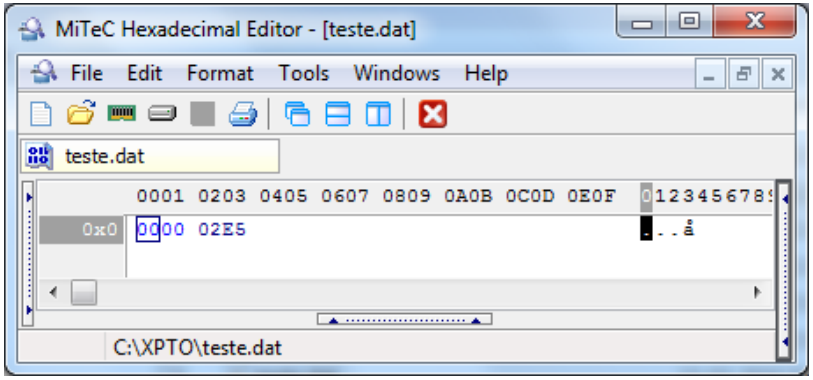


OK!

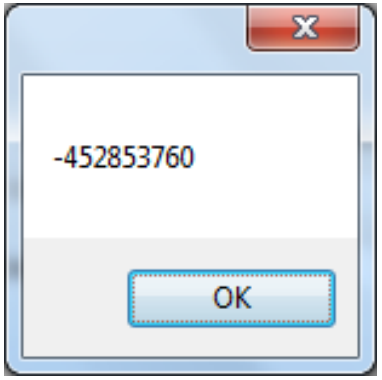


2) Ler com C# o ficheiro escrito em Java contendo o valor inteiro 741 ?

C# → using (BinaryReader b = new BinaryReader(File.Open("teste.dat", FileMode.Open)))
int v = b.ReadInt32();
MessageBox.Show(v.ToString());

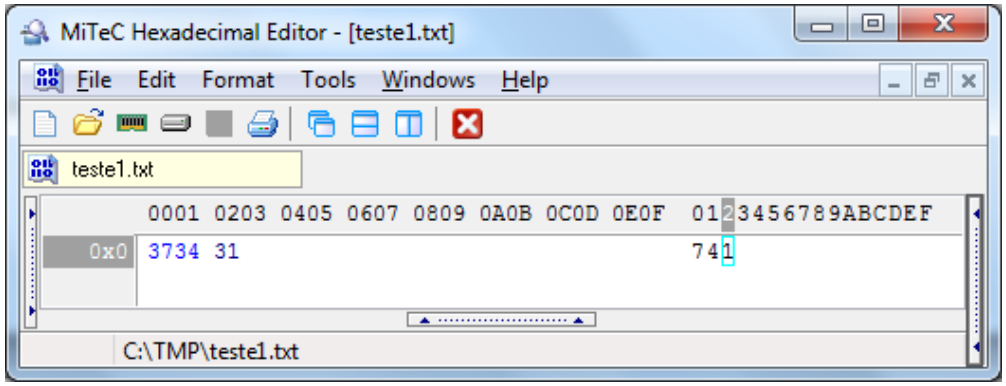


???



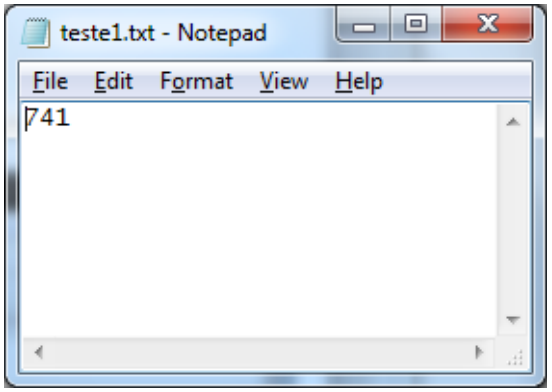
Armazenamento da Informação

1) Ler com um processador de texto o ficheiro escrito em C# contendo a string (conjunto de caracteres) “741” ?



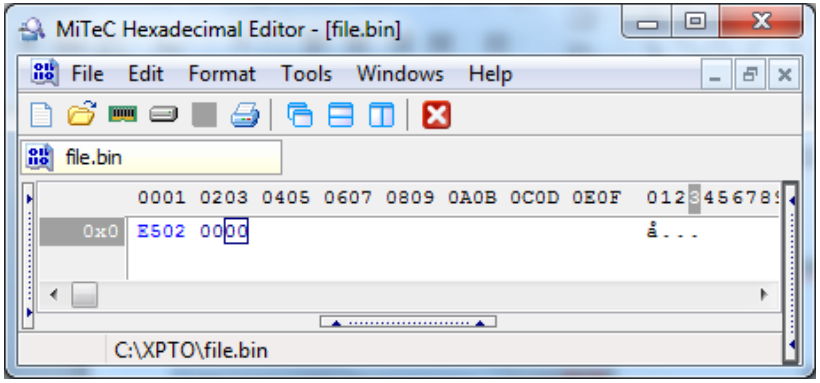
ASCII

OK!



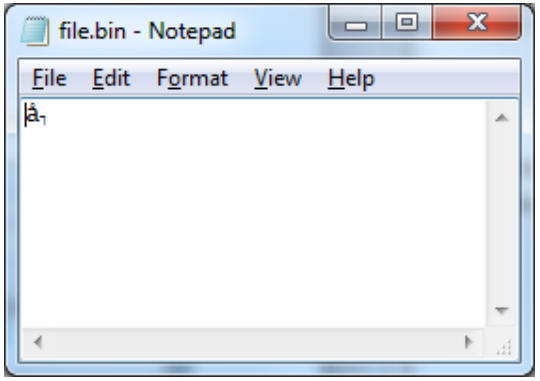
2) Ler com um processador de texto o ficheiro escrito em C# ou Java contendo o inteiro (em binário) 741 ?

C#

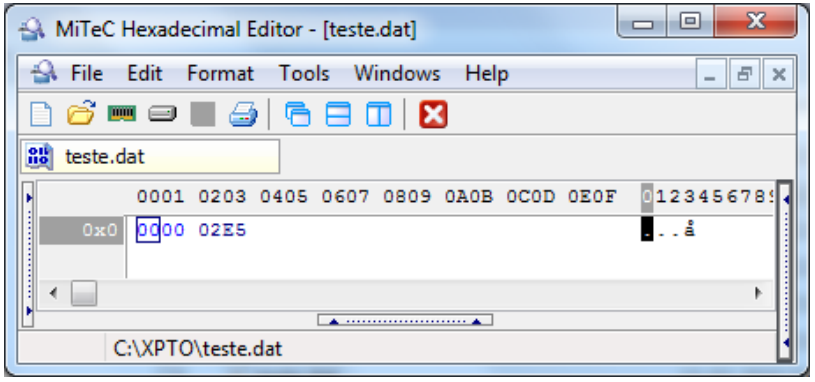


ASCII

???

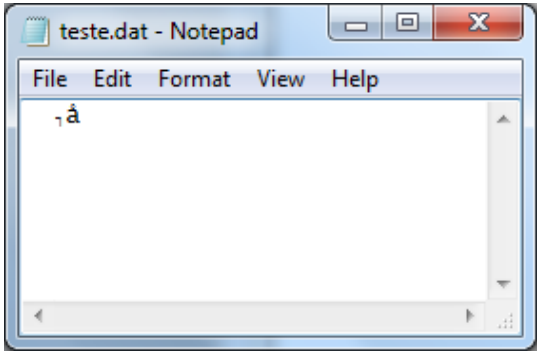


Java



ASCII

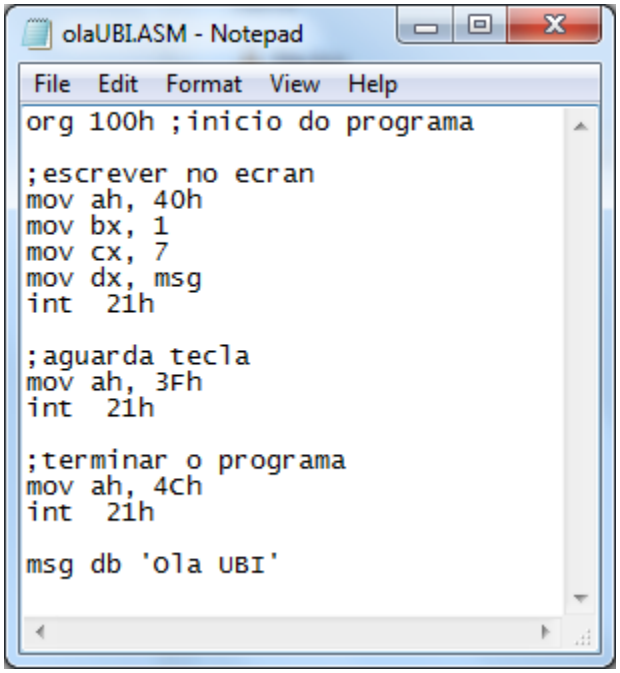
???



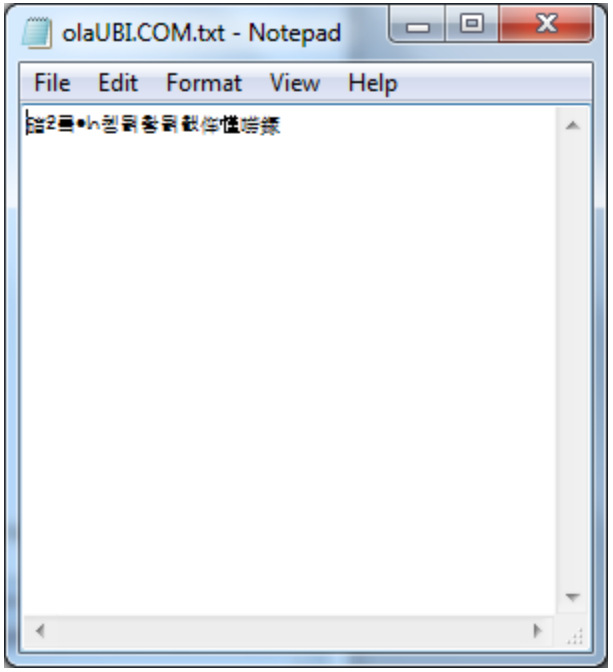
Armazenamento da Informação

Ler com um processador de texto o conteúdo do ficheiro “olaUBI.ASM” e “olaUBI.COM”

OK



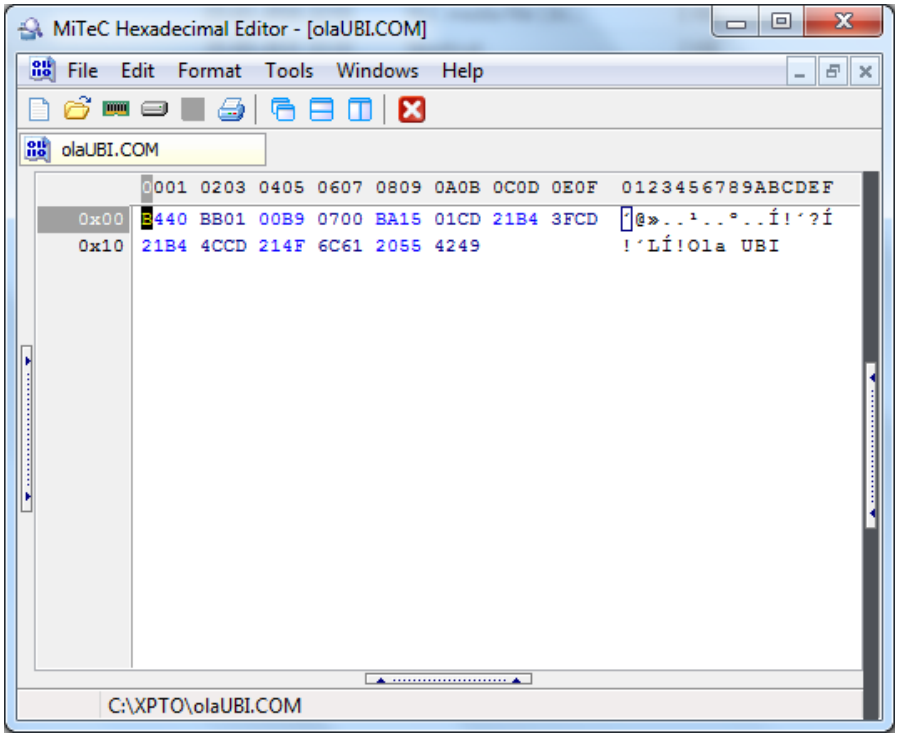
???



Armazenamento da Informação

Ler com um editor hexadecimal o conteúdo do ficheiro “olaUBI.COM”

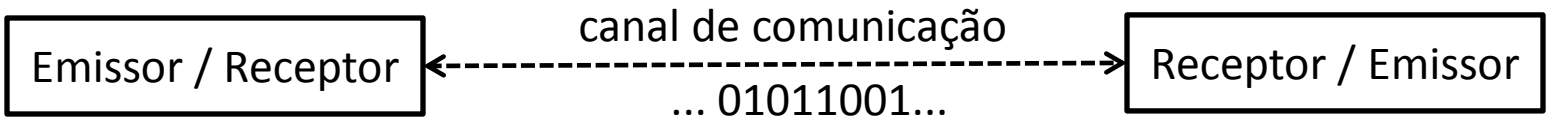
```
org 100h ;inicio do programa

;escrever no ecran
mov ah, 40h   ↔ B440      ;escrita
mov bx, 1     ↔ BB0100    ;ecran
mov cx, 7     ↔ B90700    ;nº char
mov dx, msg   ↔ BA1501    ;string
int  21h      ↔ CD21      ;executa

;aguarda tecla
mov ah, 3Fh ↔ B43F  ;leitura
int  21h    ↔ CD21  ;executa

;terminar o programa
mov ah, 4Ch ↔ B44C  ;terminar
int  21h    ↔ CD21  ;executa


msg db 'Ola UBI'  ;[4F,6C,61,20,55,42,49]
```



Aula 6
2021-04-09

Tratamento de erros

Tratamento de erros



A transmissão de informação ao longo de um canal de comunicação pode sofrer interferências dando origem a erros, exemplos:

- transferência de dados entre registos dentro do CPU;
- entre o CPU e a memória;
- nas entradas/saídas de dispositivos periféricos (teclado, impresssora);
- em comunicações em rede (cabos Ethernet, fibra óptica);

A existência de ruído pode provocar com que um bit mude de estado: 1 passa a 0 ou 0 passa a 1

ex: 01011001 → 01001001 ou 01011001 → 01011101 (num erro em rajada pode até mudar mais de um bit em simultâneo)

O ruído (interferência) pode ter várias causas: interferência eletromagnética , falha de energia, deficiência nos circuitos(ex. maus contactos elétricos), ...

Abordagens possíveis

Deteção de erros: detetar que ocorreu um erro – em seguida a transmissão pode ser repetida;

Correção de erros: detetar a ocorrência do erro e efetuar a sua correção (evita a retransmissão);

Todos os métodos se baseiam na utilização de bits que são adicionados à mensagem original, designados por bits de paridade ou de verificação.

Deteção de erros

Princípio: adicionar bits extra a cada bloco de dados a transmitir de modo que quando o bloco é recebido os bits extra são verificados para constatar se ocorreu ou não um erro.

Bit de paridade → conta-se o nº de bits "1" do bloco a transmitir e acrescenta-se um bit adicional, tal que:

- para paridade par (EVEN) → nº total de bits "1" seja par
- para paridade ímpar (ODD) → nº total de bits "1" seja ímpar

Exemplo (a negrito está o bit de paridade)

Dados binários	Paridade Par	Ímpar
0000	**0**0000	**1**0000
0001	**1**0001	**0**0002
0101	**0**0101	**1**0101
0110	**0**0110	**1**0110
1000	**1**1000	**0**1000
1011	**0**1011	**0**1011
1101	**1**1101	**0**1101
1111	**0**1111	**1**1111

Como obter o bit de paridade?

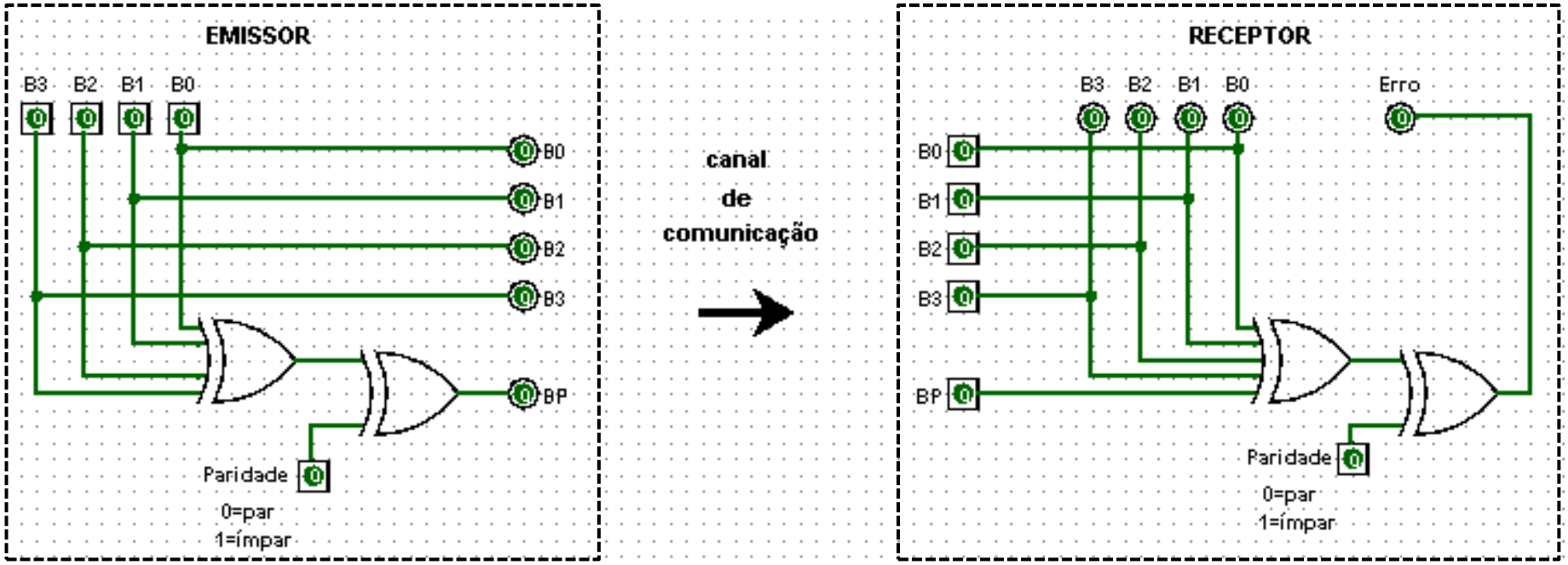
Definição da funcão XOR

XOR = 1 → quando há um nº ímpar de 1's nas entradas
= 0 → quando há um nº par de 1's nas entradas

paridade	bit de paridade
par	XOR entre os bits de dados
ímpar	$\overline{\text{XOR entre os bits de dados}}$

Deteção de erros – circuito do Logisim

Circuito gerador(emissor)/detetor(recetor) do bit de paridade para palavras de 4 bits



- Para que o sistema possa funcionar, emissor e recetor têm de estar a usar o mesmo tipo de paridade (par ou ímpar)
- Se durante a transmissão, um número ímpar de bits for alterado (incluindo o próprio bit de paridade), a paridade altera-se e o erro será detetado
- Se o número de bits alterados for par, a paridade não sofre alteração e o erro não é detetado
- Este circuito deteta erro num bit mas não indica qual o bit errado → quando há erro os dados devem ser descartados e retransmitidos novamente.

Deteção de erros

O bit de paridade encontra-se muito associado às comunicações série, tal como nas portas COM dos PCs

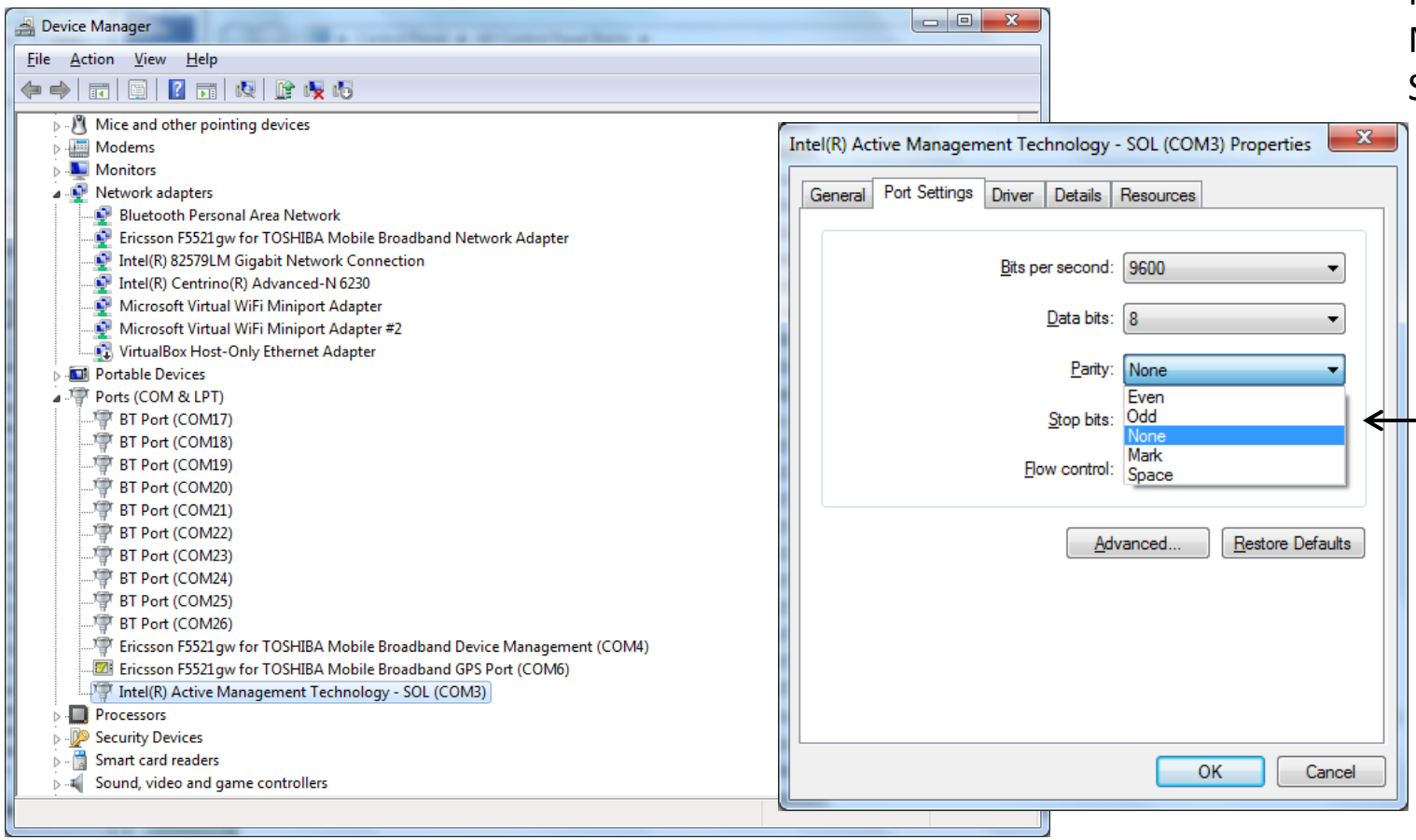
COM → bits por segundo (baud rate) + Data bits + Parity

Bps: 4800, 9600, 19200...

Data Bits: 4 , 5 , 6 , 7 , 8

Parity: EVEN, ODD
NONE – nenhuma
MARK – sempre 1
SPACE – sempre 0

Windows > Control Panel > Device Manager



Deteção e Correção de erros

- bit de paridade → permite detetar erros num certo grupo de bits;
  → não permite corrigir o erro pois não se sabe qual o bit que o originou;

- Para detetar, identificar e corrigir o erro (em qual bit ou bits) terá de adicionar-se mais informação, ou seja, bits de paridade/verificação adicionais;

- Códigos de correção possibilitam recuperar o dado original a partir do código com erros;

- Consistem na inclusão de informação adicional que deteta situações inválidas mas que mantém a identidade do dado original;

Código de Hamming

- faz uso do conceito de bit de paridade, mas além da deteção de um erro indica qual foi o bit no qual ele ocorreu, permitindo assim corrigir esse erro.

O código de paridade do código de Hamming é obtido a partir da palavra de dados, inserindo pontos de controlo, denominados bits de paridade.

Em cada palavra de dados, de comprimento **n** bits, são inseridos um número fixo **k**, de bits de paridade, ficando a palavra de código com um comprimento **N = n + k** , sendo:

$N = 2^K-1$, $n = N – k$, $n = 2^k-1-k$

Ex: k = 3 bits de paridade → N=7 bits no total , $n = 2^3-1-3 = 4$ bits de dados

n bits de dados	k bits de paridade	N = n + k total de bits
1	2	3
4	3	7
11	4	15
26	5	31
57	6	63
120	7	127
247	8	255

Nota: determinar k a partir de n → $2^k \geq k + n + 1$

Código de Hamming

Construção do código de Hamming

1) Numerar os bits a transmitir a partir da esquerda → posição do bit : 1, 2, 3, 4, 5, 6, 7, 8, ...
2) Todas as posições que correspondem a potências de dois são bits de paridade (p_n)
3) Todas as outras posições são os bits de dados (d_n)

Regra para construção do código

		2^0	2^1		2^2				2^3								2^4					
Posição do bit		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
bits codificados		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
bits de paridade	p1	X		X		X		X		X		X		X		X		X		X		...
	p2		X	X			X	X			X	X			X	X			X	X		
	p4				X	X	X	X					X	X	X	X					X	
	p8								X	X	X	X	X	X	X	X						
	p16																X	X	X	X	X	

cada bit de paridade

→ é calculado a partir da função XOR entre os bits marcados na tabela

→ na verificação, os bits de paridade são recalculados e codificam em binário a posição do bit errado

- se, $p_1 = p_2 = p_4 = p_8 = ... = 0$ → todos os bits de paridade são zero → não há nenhum bit errado
- se, $p_n \neq 0$ (algum bit de paridade não é zero) → a posição do bit errado é dada pelo código $p_8\, p_4\, p_2\, p_1$

  (ex: $p_8\, p_4\, p_2\, p_1 = 0011$ → bit3[d1] → errado)

Código de Hamming

<u>Emissor</u>: geração do código de Hamming

Exemplo : n = 4 bits de dados (d1,d2,d3,d4) , k = 3 bits de paridade (p1,p2,p4) , N = 7 bits total

palavra a codificar: **1011**

		2^0	2^1		2^2			
Posição do bit		1	2	3	4	5	6	7
bits codificados		p1	p2	d1	p4	d2	d3	d4
bits de paridade	p1	X		X		X		X
	p2		X	X			X	X
	p4				X	X	X	X
	p8							
	p16							

<u>bits de dados</u>

d1 d2 d3 d4 = **1011**

		2^0	2^1		2^2			
Posição do bit		1	2	3	4	5	6	7
bits codificados		p1	p2	**1**	p4	**0**	**1**	**1**
bits de paridade	p1	X		X		X		X
	p2		X	X			X	X
	p4				X	X	X	X
	p8							
	p16							

<u>bits de paridade</u>

p1 → bit3 ⊕ bit5 ⊕ bit7 = 1 ⊕ 0 ⊕ 1 = **0**

p2 → bit3 ⊕ bit6 ⊕ bit7 = 1 ⊕ 1 ⊕ 1 = **1**

p4 → bit5 ⊕ bit6 ⊕ bit7 = 0 ⊕ 1 ⊕ 1 = **0**

		2^0	2^1		2^2			
Posição do bit		1	2	3	4	5	6	7
bits codificados		**0**	**1**	1	**0**	0	1	1
bits de paridade	p1	X		X		X		X
	p2		X	X			X	X
	p4				X	X	X	X
	p8							
	p16							

palavra original : **1011** → palavra codificada : **0110011**

Código de Hamming

Recetor: verificação do código de Hamming

O recetor vai recalcular os bits de paridade - para tal usa os próprios bits de paridade que foram recebidos, além de bits de dados:

- Se todos os bits de paridade recalculados forem 0 (zero) então não houve erros na comunicação
- Se algum bit de paridade recalculado for 1 (um) então houve algum erro – nesse caso os bits de paridade codificam em binário a posição do bit errado

Exemplo1 : palavra recebida pelo canal = 0110011, conterá erros?

		2^0	2^1		2^2			
Posição do bit		1	2	3	4	5	6	7
bits codificados		**p1 0**	**p2 1**	**d1 1**	**p4 0**	**d2 0**	**d3 1**	**d4 1**
bits de paridade	p1	X		X		X		X
	p2		X	X			X	X
	p4				X	X	X	X
	p8							
	p16							

bits de dados
d1 d2 d3 d4 = **1011**

bits de paridade

p1 → bit1 ⊕ bit3 ⊕ bit5 ⊕ bit7 = 0 ⊕ 1 ⊕ 0 ⊕ 1

p2 → bit2 ⊕ bit3 ⊕ bit6 ⊕ bit7 = 1 ⊕ 1 ⊕ 1 ⊕ 1

p4 → bit4 ⊕ bit5 ⊕ bit6 ⊕ bit7 = 0 ⊕ 0 ⊕ 1 ⊕ 1

0 0 0

bits de paridade todos zero → nenhum bit errado → dados são aceites !

Código de Hamming

Recetor: verificação do código de Hamming

Exemplo2 : palavra recebida pelo canal = 0110001 , conterá erros?

		2^0	2^1		2^2			
Posição do bit		1	2	3	4	5	6	7
bits codificados		p1 0	p2 1	d1 1	p4 0	d2 0	d3 0	d4 1
bits de paridade	p1	X		X		X		X
	p2		X	X			X	X
	p4				X	X	X	X
	p8							
	p16							

bits de dados

d1 d2 d3 d4 = **1001**

↓ inverter

1011

bits de paridade

p1 → bit1 ⊕ bit3 ⊕ bit5 ⊕ bit7 = 0 ⊕ 1 ⊕ 0 ⊕ 1

p2 → bit2 ⊕ bit3 ⊕ bit6 ⊕ bit7 = 1 ⊕ 1 ⊕ 0 ⊕ 1

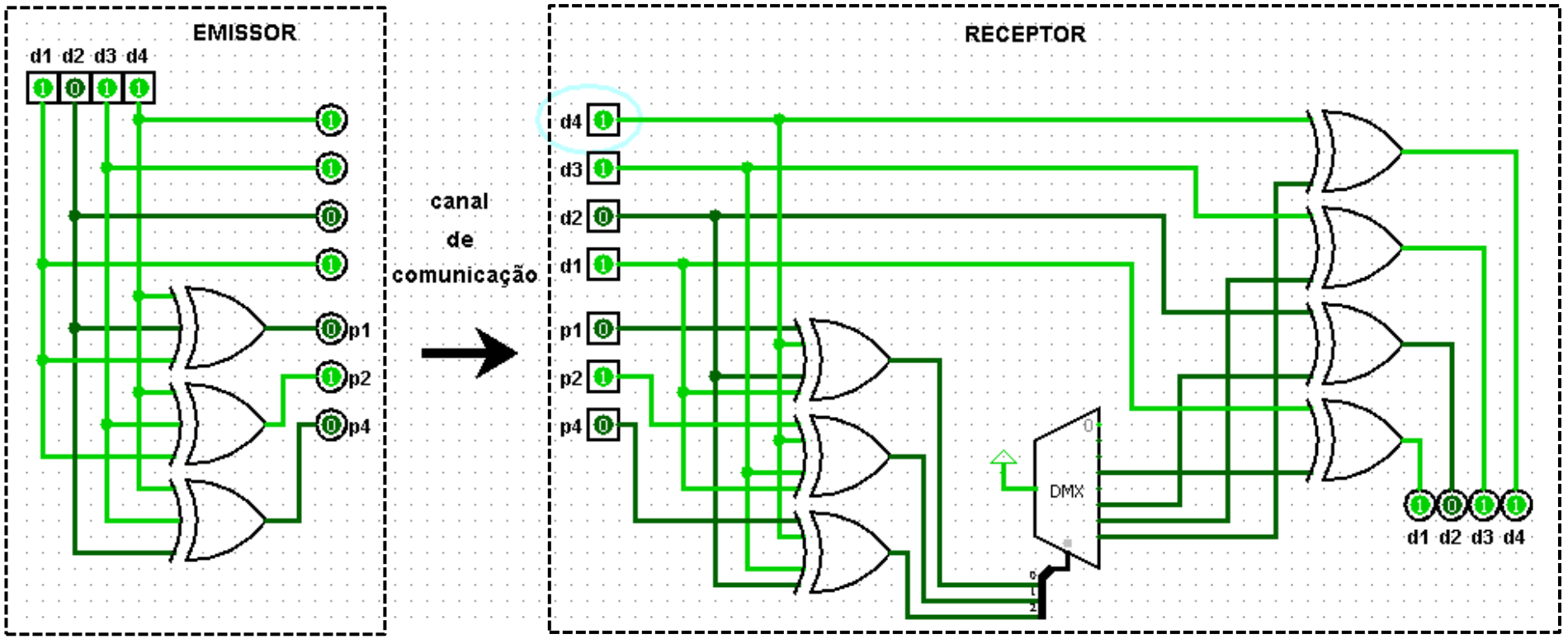
p4 → bit4 ⊕ bit5 ⊕ bit6 ⊕ bit7 = 0 ⊕ 0 ⊕ 0 ⊕ 1

1 1 0 (=6) →posição 6 = bit d3 →errado!

sabendo-se que um bit está errado (trocado), então é só invertê-lo

Código de Hamming

Circuito gerador + detetor /corretor do código de Hamming para palavras de 4 bits:
detecta e corrige um erro em <u>um</u> bit → assim não obriga a retransmissão



<u>bits de paridade</u>

p1 → d1 ⊕ d2 ⊕ d4
p2 → d1 ⊕ d3 ⊕ d4
p4 → d2 ⊕ d3 ⊕ d4

<u>bits de paridade</u>

p1 → p1 ⊕ d1 ⊕ d2 ⊕ d4
p2 → p2 ⊕ d1 ⊕ d3 ⊕ d4
p4 → p4 ⊕ d2 ⊕ d3 ⊕ d4

bits paridade p4 p2 p1	bit errado
0 0 0	-
0 0 1	p1
0 1 0	p2
0 1 1	d1
1 0 0	p4
1 0 1	d2
1 1 0	d3
1 1 1	d4

Aula 7
2021-04-16

Avaliação do desempenho

Avaliação do Desempenho

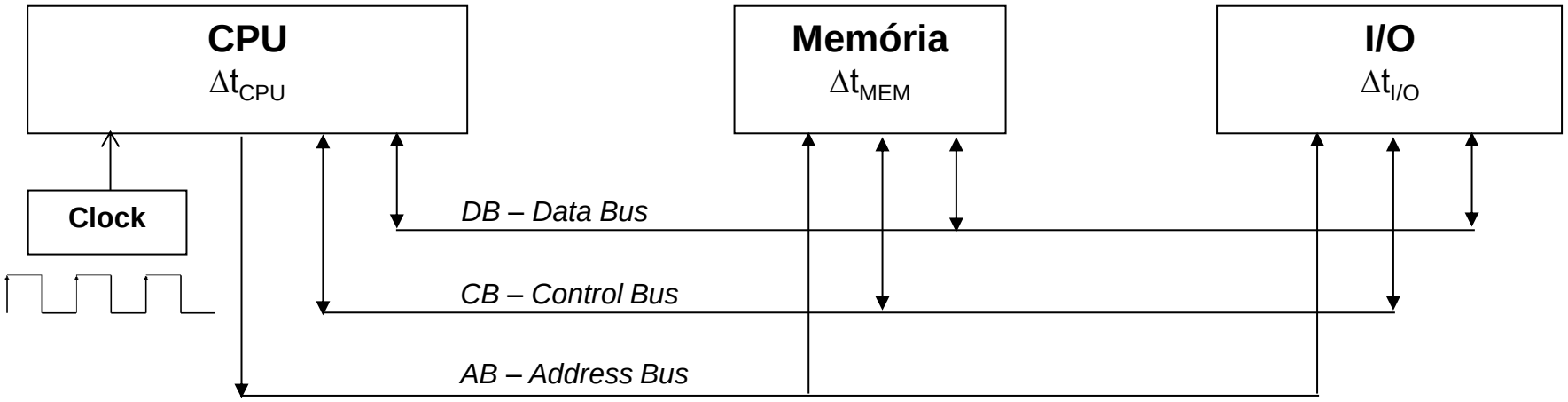
Desempenho – como veremos, significa no fundo a rapidez com que as máquinas executam as suas funções

Com a avaliação do desempenho pretende-se:

1) Estabelecer métricas para medir/avaliar o desempenho de uma máquina;
2) Comparar entre si o desempenho de várias máquinas;
3) Perceber alguns dos aspetos que limitam o desempenho;
4) Introduzir melhorias na arquitetura das máquinas com vista à melhoria do desempenho;

Os diagramas a seguir mostram os tempos envolvidos nas diversas fases de execução das instruções

Avaliação do Desempenho



Δt_{CPU} : tempo que o CPU leva para executar uma instrução (ex.mover dados entre registos, usar a ALU, etc);

Δt_{MEM} : tempo de acesso memória, correspondente à leitura ou escrita dos dados;

$\Delta t_{I/O}$: tempo de acesso a periféricos,como seja o teclado, ecran, impressora;

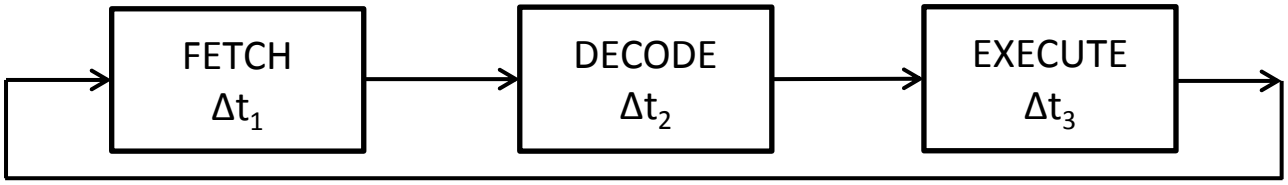
(há também o tempo de trânsito dos dados nos diversos bus mas que é muito pequeno e será ignorado)

Daqui resulta → tempo total para executar uma instrução : $\Delta T = \Delta t_{CPU} + \Delta t_{MEM} + \Delta t_{I/O}$

Neste estudo iremos concentrar-nos apenas na análise de Δt_{CPU}

Avaliação do Desempenho

CPU → ciclo : Fetch-Decode-Execute



O ciclo Fetch-Decode-Execute representa o conjunto de etapas cumpridas permanentemente pelo CPU ao executar as instruções (instruções essas vindas de qualquer programa, incluindo do sistema operativo):

Δt_1 – busca da instrução: tempo para obter uma instrução da memória, relacionado com o tempo de resposta da memória (Δt_{MEM}) e daí a importância da memória no desempenho das máquinas;

Δt_2 – descodificação: tempo que o CPU precisa para "perceber" qual é a instrução e o que é preciso para a executar – ex. numa instrução de soma é preciso ir à memória buscar os valores a somar;

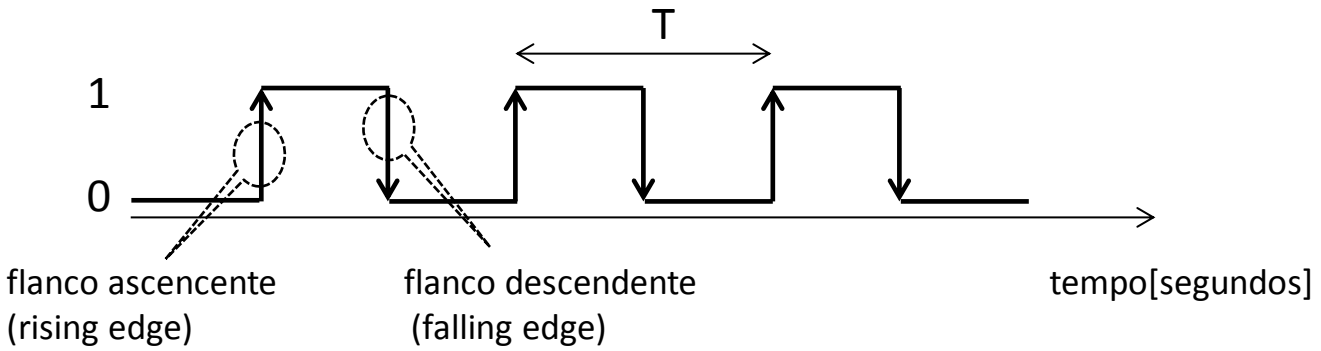
Δt_3 – execução: tempo para executar realmente a instrução – ex.numa instrução de soma, implica usar a ALU, guardar o resultado, etc;

Tempo gasto pelo CPU: $\Delta t_{CPU} = \Delta t_1 + \Delta t_2 + \Delta t_3$

QUESTÃO: como avaliar (e depois diminuir) Δt_{CPU} ?

Avaliação do Desempenho

Sinal de relógio (clock) : sinal muito importante pois controla o tempo e os momentos em que as instruções são executadas – o CPU pode executar as instruções quando acontece uma transição ascendente do clock, mas também na transição descendente ou em ambas.



T = período ou ciclo(clock cycle) [s] : tempo que decorre entre dois acontecimentos iguais: unidade = segundo

F = frequência(clock rate) [Hz] : nº de repetições de períodos por unidade de tempo: unidade = Hertz

F = 1 / T (frequência é o inverso do período) , T = 1 / F (período é o inverso da frequência)

1Hz = 1 período / segundo

F - frequência		T - tempo	
[Hz]	Designação	[s]	Designação
1 Hz	Hertz = 1 período por segundo	1 s	segundo
1KHz = 10^3Hz	Kilo Hertz = 10^3 periodos por segundo	10^{-3}s = 0.001s = 1ms	mili segundo
1MHz = 10^6Hz	Mega Hertz = 10^6 periodos por segundo	10^{-6}s = 0.000001s = 1µs	micro segundo
1GHz = 10^9Hz	Giga Hertz = 10^9 periodos por segundo	10^{-9}s = 0.000000001s = 1ns	nano segundo

(ex: rede eléctrica doméstica → F=50Hz, T=20ms)

Avaliação do Desempenho

Sinal de relógio

Cada instrução gasta um certo número de períodos T de clock; se for possível diminuir esse período T, então o desempenho irá melhorar:

$\Delta t_{CPU} = f(T_{clock})$: $\downarrow T \rightarrow \downarrow \Delta t_{CPU}$ (diminuindo T diminui Δt_{CPU}, logo melhora o desempenho)

ou, dito de outro modo:

$\Delta t_{CPU} = f(1/F_{clock})$: $\uparrow F \rightarrow \downarrow \Delta t_{CPU}$ (aumentando F diminui Δt_{CPU}, logo melhora o desempenho)

No entanto, $\downarrow T$ (ou o equivalente $\uparrow F$) :

- diminui o tempo para executar as instruções, podendo não ser possível de as executar;
- pode obrigar a mais períodos de clock para executar as mesmas instruções;
- obriga a tornar as instruções mais eficientes (demorarem menos tempo) ;
- aumentar a frequência está condicionado por questões técnicas, como tempos de comutação das portas lógicas (transições 0→1 e 1→0), aumento da temperatura, etc;

Os fabricantes têm vindo a conseguir aumentar sucessivamente o valor de F:

1981 : IBM PC original….. 4.77 MHz

1995 : Pentium…….. 100 MHz

2000 : AMD …….. 1 GHz

2002 : Pentium 4…….. 3 GHz

(a partir daqui a evolução tem sido mais lenta)

Avaliação do Desempenho

Métricas (de má qualidade) para avaliação do desempenho

1) F - frequência : maior F não implica necessariamente melhor desempenho, pois um CPU com menor F_{clock} pode ser mais eficiente que um com F_{clock} mais elevada, podendo realizar mais operações em menos tempo.

ex: CPU_A: F=1GHz , 2 ciclos por instrução (média) CPU_B: F=1.5GHz , 3 ciclos por instrução (média)

Qual CPU tem melhor desempenho?

$CPU_A \rightarrow T = 1/F = 1/10^9 = 10^{-9}s = 1ns$
t(instrução) = 2 * T = 2 * 1 = 2ns

$CPU_B \rightarrow T = 1/F = 1/(1.5*10^9) = 0.67*10^{-9}s = 0.67ns$
t(instrução) = 3 * T = 3 * 0.67 = 2.01ns

Conclusão: CPU_A é mais rápido que CPU_B , apesar de ter menor F_{clock}

2) MIPS - Millions of Instructions Per Second : normalmente é o tempo de execução de certas instruções (ex: NOP). No entanto os programas são constituídos por diversas outras instruções, pelo que devem ser usadas médias ponderadas.

3) MFLOPS-Millions of Floating-Point Operations Per Second : número máximo de instruções de vírgula flutuante (não inteiros) por segundo. No entanto, a maioria dos programas não faz uso intensivo destas instruções.

Portanto nenhuma destas medidas serve realmente para fazer a avaliação do desempenho

Avaliação do Desempenho

Tempo de execução

- única medida completa e de confiança para avaliar o desempenho
- a máquina que executa o mesmo trabalho em menos tempo é a mais rápida
- desempenho define-se como sendo o inverso do tempo de execução

Para um certo programa executado numa máquina X

$$\text{DesempenhoX} = \frac{1}{\text{Tempo}_{\text{execuçãoX}}} \qquad (\downarrow \text{Tempoexecu}_{\text{çãoX}} \rightarrow \uparrow \text{DesempenhoX})$$

Para duas máquinas X e Y

$$\text{DesempenhoX} > \text{DesempenhoY} \rightarrow \frac{1}{\text{Tempo}_{\text{execuçãoX}}} > \frac{1}{\text{Tempo}_{\text{execuçãoY}}} \rightarrow \text{Tempo}_{\text{execuçãoX}} < \text{Tempo}_{\text{execuçãoY}}$$

Se a máquina X é **N** vezes mais rápida que a máquina Y

$$\text{DesempenhoX} = N * \text{DesempenhoY} \rightarrow \frac{\text{DesempenhoX}}{\text{DesempenhoY}} = \frac{\text{Tempo}_{\text{execuçãoY}}}{\text{Tempo}_{\text{execuçãoX}}} = N \rightarrow \text{Tempo}_{\text{execuçãoX}} = \frac{\text{Tempo}_{\text{execuçãoY}}}{N}$$

Ex: máquina X → $\text{Tempo}_{\text{execuçãoX}}$, para uma certa tarefa = 10s

máquina Y → $\text{Tempo}_{\text{execuçãoY}}$, para a mesma tarefa = 15s

$$N = \frac{\text{DesempenhoX}}{\text{DesempenhoY}} = \frac{\text{Tempo}_{\text{execuçãoY}}}{\text{Tempo}_{\text{execuçãoX}}} = \frac{15}{10} = 1.5$$

$$\text{DesempenhoX} = 1.5 * \text{DesempenhoY} \quad , \quad \text{Tempo}_{\text{execuçãoX}} = \frac{\text{Tempo}_{\text{execuçãoY}}}{1.5}$$

→a máquina X é 1.5 vezes mais rápida que máquina Y

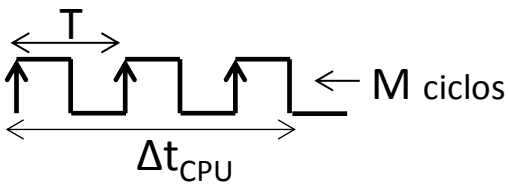
Avaliação do Desempenho

Tempo total para executar uma tarefa: $\Delta T = \Delta t_{CPU} + \Delta t_{MEM} + \Delta t_{I/O}$, considerando apenas a parte do CPU:

Δt_{CPU} = $Tempo_{execuçãoCPU}$ = tempo que o CPU leva para executar um programa - para executar um programa o CPU precisa de um certo nº , M, de ciclos de relógio, cada um de duração T, logo:

$Tempo_{execuçãoCPU}$ = M(ciclos clock) * T_{clock}(clock cycle)

$Tempo_{execuçãoCPU}$ = M(ciclos clock) / F_{clock}(clock rate)



F_{clock} → é conhecida (é o relógio da máquina)

M = ?? (como saber quantos ciclos de clock correspondem a um certo programa ?)

Para um certo programa

- N = nº de instruções totais do programa (mov, add, jmp,...)
- cada instrução precisa de um certo valor médio de períodos de clock para ser executada, esse valor é o CPI (Clocks Per Instruction) = nº médio ponderado de períodos de clock para cada instrução, que é um valor indicado pelo fabricante do CPU

então: M = N * CPI → $Tempo_{execução}$ = N * CPI / F_{clock} → 3 factores condicionantes

1) N : depende do programador e do compilador, quanto melhor programador/compilador menor N
2) CPI : depende da arquitetura do processador, necessita de mais ou menos clocks por instrução
3) F : depende do hardware, a eletrónica permite uma frequência de relógio maior ou menor

Avaliação do Desempenho

Exemplo

Dispondo de duas máquinas diferentes mas da mesma arquitetura (Intel x86):

CPU_A : F_A=500MHz , CPI_A=1.2 CPU_B : F_B=800MHz , CPI_B=2

vefifica-se que : clock da máquina A (F_A) < clock da máquina B (F_B)

$CPI_A < CPI_B$ → a máquina A é mais eficiente que a máquina B (aproveita melhor o clock)

Para um mesmo programa, qual a máquina que o executa mais rapidamente?

$Tempo_{execução} = N * CPI / F_{clock}$ Programa → N instruções

$Tempo_{execuçãoA} = N * CPI_A / F_A = N * 1.2 / 500*10^6$

$Tempo_{execuçãoB} = N * CPI_B / F_B = N * 2 / 800*10^6$

$DesempenhoA / DesempenhoB = Tempo_{execuçãoB} / Tempo_{execuçãoA} = n$

$$Tempo_{execuçãoB} / Tempo_{execuçãoA} = \frac{N * 2 / 800*106}{N * 1.2 / 500*106}$$ → n = 1.04 → máquina A é mais rápida que B (apesar do menor clock)

Avaliação do Desempenho

Agora que dispomos de uma forma de calcular o desempenho, iremos introduzir aperfeiçoamentos ou melhorias na arquitetura do CPU de modo a melhorar esse desempenho.

De que modo esses aperfeiçoamentos ou melhorias, afetam o desempenho?

- Cada aperfeiçoamento introduzido vai ser responsável por melhorar apenas um aspeto do funcionamento do CPU

  ex: uma unidade de multiplicação por hardware acelera as multiplicações (e talvez as divisões) mas em nada afeta os acessos à memória, pois não interfere nesse processo;

- As preocupações com as melhorias devem centrar-se nos casos mais comuns (mais frequentes) tornando mais rápida (mais optimizada) a sua execução, ou seja, deve ser melhorado aquilo que ocorre mais frequentemente.

Lei de Amdahl - *"O ganho de desempenho que pode ser obtido melhorando uma determinada parte do sistema é limitado pela fração de tempo em que essa parte é utilizada durante a operação"*

$Tempo_{antigo}$: tempo anterior à introdução de um aperfeiçoamento (ou melhoramento)

⇩ introdução de um aperfeiçoamento

$Tempo_{novo} = Tempo_{melhorado} + Tempo_{inalterado}$: tempo após a introdução do aperfeiçoamento

$Tempo_{melhorado}$: tempo afetado pela introdução do aperfeiçoamento

$Tempo_{inalterado}$: tempo não afetado pela introdução do aperfeiçoamento

Avaliação do Desempenho

Lei de Amdahl

$$\text{Aceleração}_{global}\ (\text{speedup}) = \text{Desempenho}_{novo} / \text{Desempenho}_{antigo} = \text{Tempo}_{antigo} / \text{Tempo}_{novo}$$

$$= \frac{1}{(1 - \text{Fração}_{melhorada}) + \frac{\text{Fração}_{melhorada}}{\text{Aceleração}_{melhorada}}}$$

$$\text{Tempo}_{novo} = \text{Tempo}_{antigo} * \left((1 - \text{Fração}_{melhorada}) + \frac{\text{Fração}_{melhorada}}{\text{Aceleração}_{melhorada}} \right)$$

$\text{Aceleração}_{global}$ (speedup) : aceleração final do sistema

$\text{Desempenho}_{antigo}$: desempenho anterior à introdução do aperfeiçoamento

Desempenho_{novo} : desempenho após a introdução do aperfeiçoamento

$\text{Fração}_{melhorada}$: fração de tempo em que o aperfeiçoamento é usado

$\text{Aceleração}_{melhorada}$: aceleração obtida quando o aperfeiçoamento é usado (seria a aceleração global se esse aperfeiçoamento fosse usado o tempo todo)

Avaliação do Desempenho

Exemplo

Uma arquitetura não tem suporte hardware para multiplicações fazendo-as por software usando adições sucessivas (ex: 3*2 = 2+2+2, o que implica executar várias vezes o algoritmo da soma). Com multiplicação por hardware o CPU é capaz de executar o correspondente algoritmo de forma muito eficiente.

Admitindo que:
- uma multiplicação por software consome M=200 ciclos de relógio
- uma multiplicação por hardware consome M=4 ciclos de relógio

Calcule a aceleração produzida pela introdução de uma unidade de multiplicação por hardware, nos casos:
a) se um programa gasta 10% do seu tempo em multiplicações
b) se um programa gasta 40% do seu tempo em multiplicações

Resolução

a) $\text{Aceleração}_{melhorada} = \text{Tempo}_{antigo} / \text{Tempo}_{novo} = 200 / 4 = 50$ (aceleração apenas da parte melhorada)

$\text{Fração}_{melhorada} = 10\% = 0.1$

$$\text{Aceleração}_{global} = \frac{1}{(1 - \text{Fração}_{melhorada}) + \frac{\text{Fração}_{melhorada}}{\text{Aceleração}_{melhorada}}} = \frac{1}{(1-0.1) + 0.1/50} = 1.11 \rightarrow 11\%$$

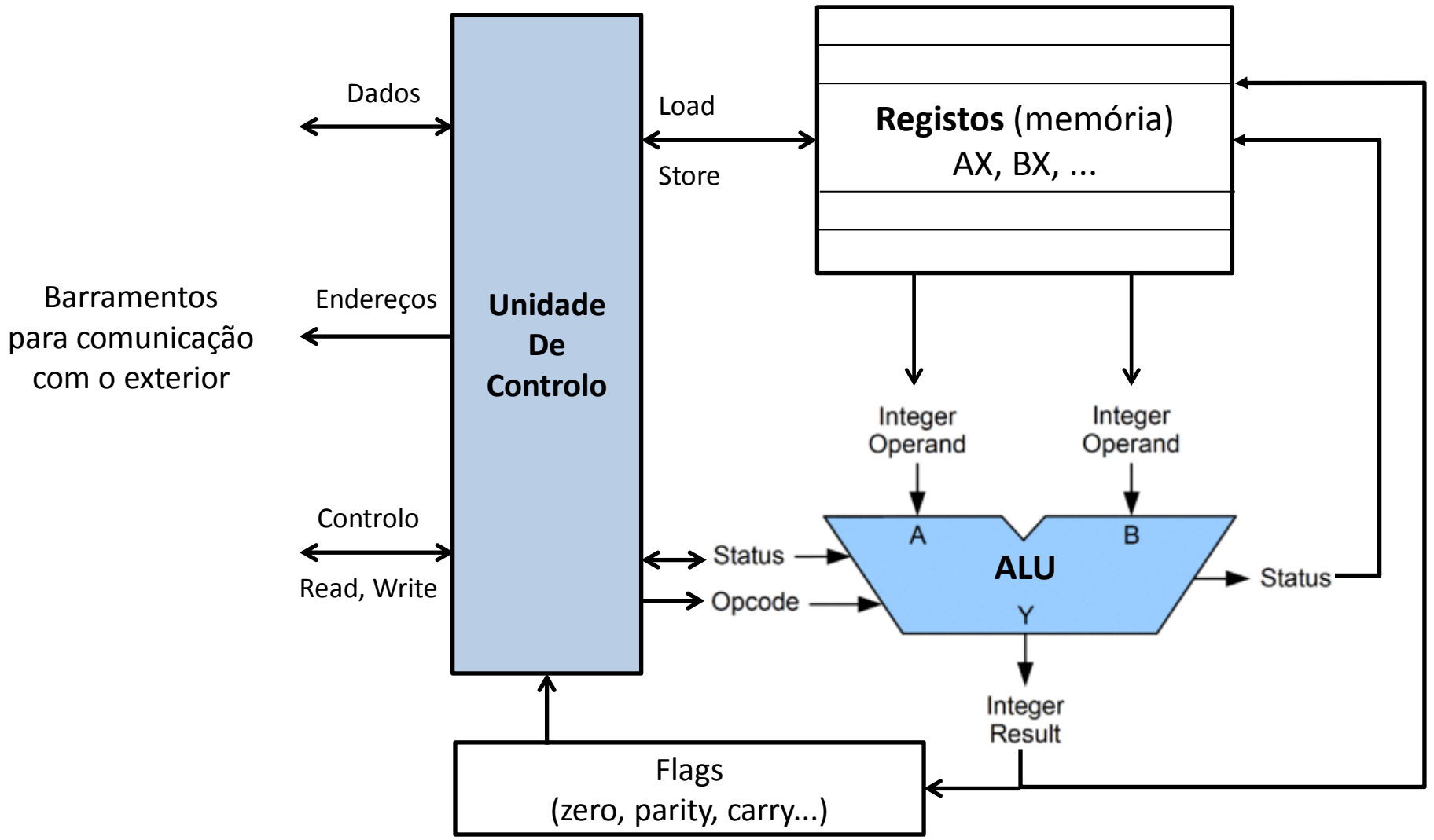
b) $\text{Aceleração}_{melhorada} = \text{Tempo}_{antigo} / \text{Tempo}_{novo} = 200 / 4 = 50$ (aceleração apenas da parte melhorada)

$\text{Fração}_{melhorada} = 40\% = 0.4$

$$\text{Aceleração}_{global} = \frac{1}{(1 - \text{Fração}_{melhorada}) + \frac{\text{Fração}_{melhorada}}{\text{Aceleração}_{melhorada}}} = \frac{1}{(1-0.4) + 0.4/50} = 1.64 \rightarrow 64\% \quad (>> 11\%)$$

Estrutura dos Processadores (CPU)

Dado o diagrama abaixo que representa a arquitetura computacional vista até agora, pretende-se estudar quais os aspetos a melhorar para aumentar o desempenho, ou seja, diminuir o tempo de execução das tarefas.



Melhoria do desempenho

Como melhorar o desempenho?

equação do desempenho: $Tempo_{CPU} = N * CPI / F$

- N = nº de instruções do programa ;
- CPI (Clocks Per Instruction) = nº médio de ciclos de relógio gastos por instrução ;
- F = Frequência do clock(Hz);

<u>Alternativas</u>

1) Aumentar F : está limitada pela eletrónica, nomeadamente pela velocidade de comutação dos transistores e por questões de dissipação de temperatura (aumentar F aumenta a Tª);
2) Diminuir N : depende da qualidade do programador e do compilador, quanto melhor estes forem menor será o número de instruções para a mesma tarefa;
3) Diminuir CPI : diminuir o nº médio de ciclos de relógio gastos por instrução implica que o CPU deve ser mais eficiente em gerir o tempo, ou seja, executar o maior número de tarefas por unidade de tempo. Isso pode conseguir-se à custa de melhorias na forma como as instruções são executadas, p.ex., executando várias em simultâneo ou recorrendo a formas de execução mais “inteligentes”;

A seguir serão analisadas algumas técnicas que conduzem a melhorias no desempenho.

Aula 8
2021-04-23

Avaliação do desempenho (cont.)
Técnicas de melhoria do desempenho

Técnicas de melhoria do desempenho : Unidade de Controlo

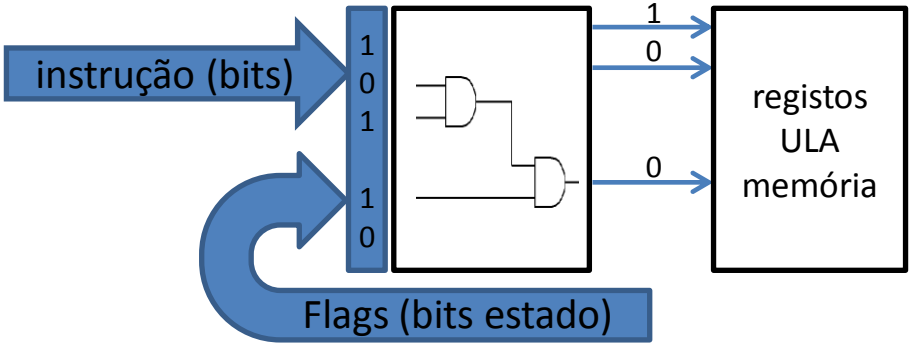
Unidade de controlo - unidade muito complexa, responsável por:

- obter e descodificar as instruções;
- gerar os sinais que indicam aos restantes blocos as ações a executar, em função das instruções do programa e do estado do processador (flags);
- a sua estrutura condiciona fortemente o desempenho do CPU;

Técnicas de construção da unidade de controlo

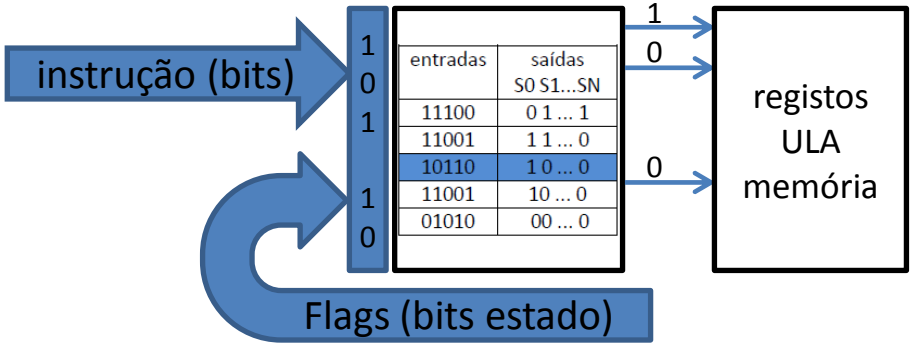
1) Hardwired (por hardware): os diversos sinais de controlo são gerados por circuitos lógicos (portas lógicas AND, OR, NOT, etc) segundo uma certa tabela de verdade e em função da instrução a executar; caraterísticas:

- rápida
- custo elevado
- alterações na arquitetura implicam alterações no hardware
- usada em máquinas RISC



2) Microcoded (microprogramado): existe uma tabela em memória (micromemória) interna ao processador, que contém os sinais a gerar em função da instrução a executar (palavra de controlo); características:

- mais lentas, pois dependem da rapidez da micromemória
- alterações na arquitetura apenas alteram o conteúdo da micromemória
- usada em máquinas CISC



Técnicas de melhoria do desempenho : CISC-RISC

Inicialmente:

- não havia linguagens de programação nem compiladores evoluídos;
- computadores eram programados quase exclusivamente em linguagem *Assembly* (linguagem máquina);
- memórias eram lentas e caras;
- processadores incluiam um grande número de funcionalidades na expetativa de que os compiladores as iriam utilizar;

John Cocke (IBM, 1974) & D. Patterson, descobriram que:

- a maior parte dos programas envolve poucas instruções, relativamente ao total de instruções disponíveis no processador ;
  regra dos 80/20 : 20% das instruções fazem 80% do trabalho(um conjunto pequeno de instruções realiza a maior parte do trabalho);
- instruções mais simples são as mais frequentes (ex: mov a,b);
- as instruções mais complexas pouco ou nunca eram geradas pelos compiladores;
- as instruções complexas complicam toda a arquitetura, obrigando a diminuir a frequência do relógio (levando a que as instruções mais simples ficassem mais lentas);

Frequência das instruções
- mov: 33%
- salto(jump): 20%
- aritméticas/logicas: 16%
- outras: 0.1% - 10%

Objetivo: minimizar o tempo de execução do conjunto de instruções mais frequentes (20%) e mesmo substituir as restantes(80%) por combinações mais rápidas daquelas, reduzindo e simplificando o total de instruções do CPU
→ isto conduziu à filosofia RISC.

Técnicas de melhoria do desempenho : CISC-RISC

Principais arquiteturas de processadores

CISC (*Complex Instruction Set Computer)*

- Grande número de instruções (~ 200 a 300) capazes de efetuar ações complexas, embora relativamente lentas, exigindo múltiplos ciclos de clock para serem executadas;
- Muitos modos de acesso à memória (direto, indireto, imediato,...);
- Instruções de tamanho variável de acordo com o modo de endereçamento;
- Unidade de controlo baseada em "Microcoded" (lento);
- Exs: *Intel 80x86 , Motorola 68K;*

RISC (*Reduced Instruction Set Computer)*

- Princípios : hardware mais simples e optimização do caso mais frequente;
- Unidade de controlo baseada em "*Hardwired*" (rápido);
- Conjunto reduzido de instruções (~50) simples e rápidas de executar;
- Permitem uma maior frequência de relógio (clock);
- Poucos modos de acesso à memória (baseado em LOAD/STORE);
- Cada instrução CISC dá origem a várias instruções RISC;
- Exs: MIPS(*Microprocessor without Interlocking Pipe Stages*), microcontroladores PIC (Microhip), Motorola PowerPC (*Performance Optimization With Enhanced RISC – Performance Computing*);

Actualmente muitos processadores adoptam uma estrutura híbrida CISC/RISC (ex.Pentium):

instruções frequentes ----------> RISC : *hardwired*
instruções menos frequentes ---> CISC : *microcoded*

Permitem um compromisso em termos de desempenho mantendo a compatibilidade com sistemas antigos

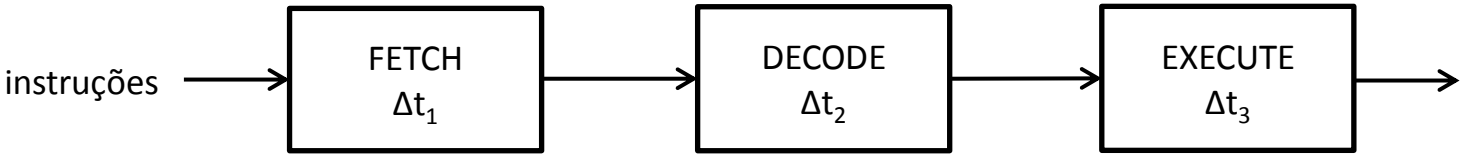
Técnicas de melhoria do desempenho : paralelismo

Paralelismo

- **A) nível das instruções** : execução de várias instruções em paralelo para aumentar o nº de instruções por segundo
- **B) nível do processador** : múltiplos CPUs trabalhando em conjunto

A) Paralelismo ao nível das instruções

Tendo em atenção a sequência FETCH-DECODE-EXECUTE que o CPU executa continuamente:



Se uma instrução fosse executada apenas quando a anterior terminou isso levaria a um mau aproveitamento, pois:

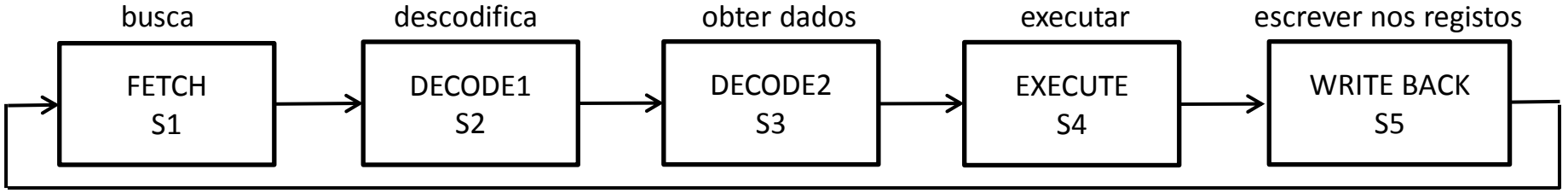
- cada instrução tem de passar pelas três fases,Fetch-Decode-Execute, cada uma dessas fases é executada por um determinado bloco do CPU, demorando tempos diferentes;
- se uma instrução está por exemplo na fase Execute, então apenas esse bloco estaria a trabalhar, estando parados os blocos correspondentes às fases Fetch e Decode;

O que se pretende é manter todos os blocos em funcionamento contínuo (tal como numa linha de fabrico em que cada operário executa uma operação passando o seu trabalho ao operário seguinte e iniciando de imediato uma nova operação)

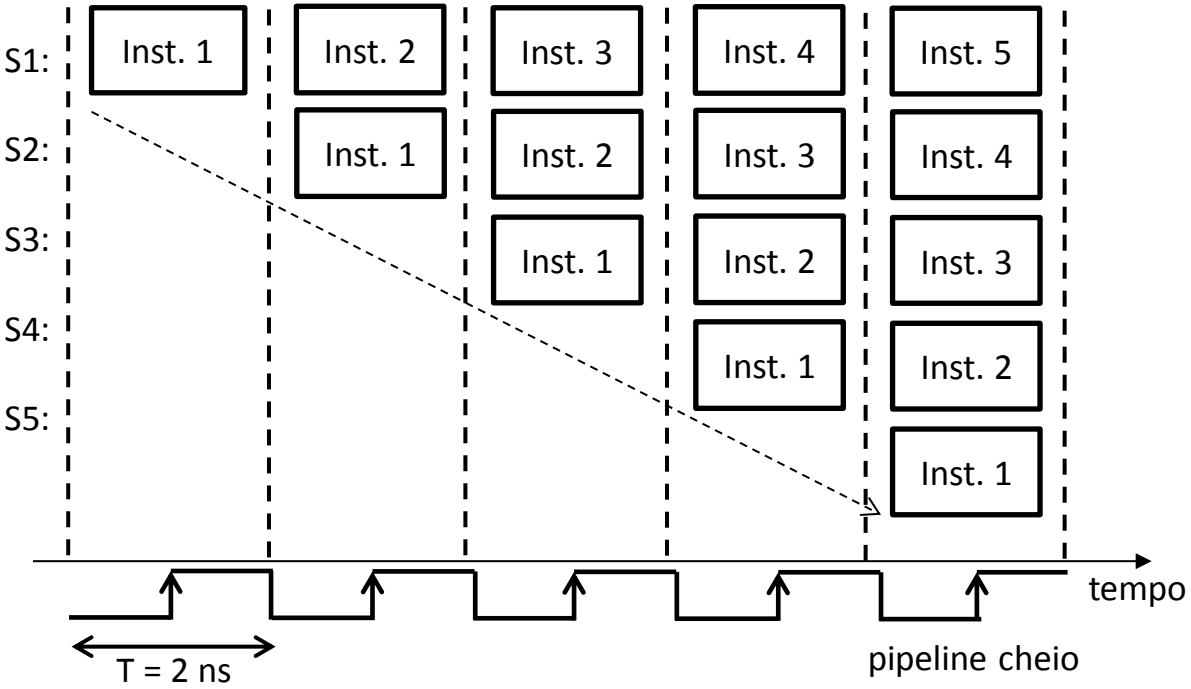
Técnicas de melhoria do desempenho : pipeline

Pipeline: vai-se dividir cada bloco anterior em blocos mais pequenos e especializados, de modo a permitir o processamento simultâneo de múltiplas instruções, cada uma em estágios de processamento diferentes;

exs: Pentium: pipeline de 5 estágios P IV: pipeline de 20 estágios



À medida que as instruções vão entrando elas vão percorrendo o pipeline ; estando sempre a entrar instruções novas a dada altura cada bloco está ocupado com sua instrução.



<u>Sem pipeline</u>

1 instrução → 5*T = 5*2ns = 10ns = $10*10^{-9}$ s

F = $1/10^{-8} = 10^{8}=100*10^{6}$ =100 MIPS

CPI(Clocks Per Instruction) = 5 → sai uma instrução completa a cada 10nS

<u>Com pipeline</u> (cheio)

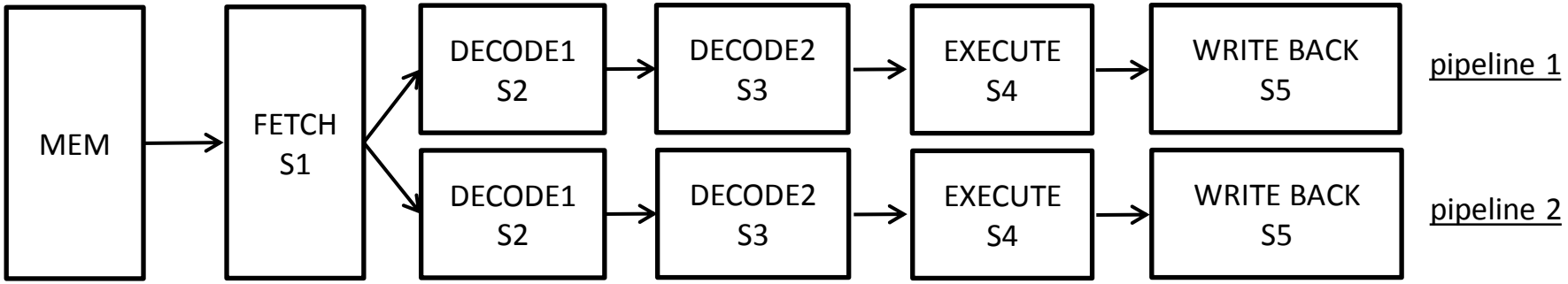
1 instrução a cada T = 2ns = $2*10^{-9}$ s

F = $1/(0.2*10^{-8})$=500 MIPS (5 vezes mais)

CPI(Clocks Per Instruction) = 1 → sai uma instrução completa a cada 2ns (embora cada instrução demore à mesma 10ns)

Técnicas de melhoria do desempenho : estruturas superescalares

Superescalar: corresponde a um pipeline duplo(se um pipeline é bom dois é melhor...): duas instruções executadas de cada vez, cada uma em seu pipeline - o hardware permite extrair paralelismo ao nível das instruções em programas sequenciais



O bloco Fetch retira as instruções da memória e distribui-as ora por um pipeline ora pelo outro (segundo certos critérios)

Esta técnica pode originar problemas (hazards) como sejam:

1) Dependência de dados : quando uma instrução precisa de dados de outra instrução tendo de parar à espera dessa outra:
   pipeline 1 → mul bl ; ax = ah * bl
   pipeline 2 → add [var], ax ; [var] = [var] + ax : tem de esperar que a multiplicação termine antes de poder usar o ax

2) Dependência de recursos : duas instruções acessam à mesma posição de memória ou precisam de usar o mesmo registo:
   pipeline 1 → mul bl ; ax = ah * bl
   pipeline 2 → mov ah, 2 ; ah = 2 : muito rápida , se terminar primeiro irá afetar a instrução de multiplicação anterior

3) Instruções de salto(não sequencialidade) - uma instrução de salto pode invalidar outras instruções que estão no pipeline:
   pipeline 1 → jz label
   pipeline 2 → sub ax, 2
   label: add ax, 2

   se ocorrer salto para label, a instrução sub (que está a ser executada no pipeline 2) terá de ser descartada e substituída pelo add

Técnicas de melhoria do desempenho

Execução fora de ordem (out-of-order) : as instruções são executadas por ordem diferente daquela em que aparecem no programa, desde que o resultado não seja alterado → exige um hardware complexo.

ex: pipeline 1 → div al ;lenta
pipeline 2 → mov bl, 2 ;rápida
mov bh, 3 ;rápida

reordenar (instruções independentes) →

pipeline 1 → mov bl , 2 ;rápida
pipeline 2 → mov bh , 3 ;rápida
div al

(pipelines desiquilibrados,um lento outro rápido) (pipelines equilibrados)

Execução especulativa : as instruções são executadas em sequência mesmo que não venham a ser usadas por ação de um desvio.

ex: pipeline 1 → jz label
pipeline 2 → mov al, 2
label: mov al, 3

o CPU "aposta" na execução da instrução mov al,2 embora por ação do desvio dado por jz label essa instrução possa ter de ser descartada e substituída por mov al,3 (ou seja, a instrução que está no pipeline 2 pode ser mov al,2 ou mov al,3 dependendo do jump

(os CPUs implementam mecanismos de previsão se o desvio vai acontecer ou não)

Técnicas de compilação(paralelismo por software) : os compiladores também são responsáveis por explorar as melhores características do paralelismo do hardware.

ex(em C): desfazer ciclos →aumentar o nº de operações independentes dentro de uma iteração

for (i=0; i<100; i++) → ciclo executa 100 vezes
a[i] = b[i] + c[i]; → pipeline 1 : só um pipeline é usado

⟹

for (i=0; i<100; i+=2) → ciclo executa 50 vezes
a[i] = b[i] + c[i]; → pipeline 1
a[i+1] = b[i+1] + c[i+1]; → pipeline 2

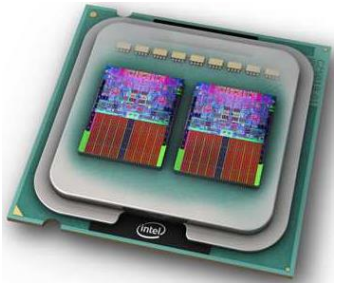
Técnicas de melhoria do desempenho : paralelismo

B) Paralelismo ao nível do processador

Máquinas com múltiplos CPUs. Existem duas possibilidades, conforme a proximidade entre eles:

• CPUs fortemente acoplados – quando estão contidos no mesmo hardware, como no caso de sistemas multicore os quais possuem mais de um núcleo;

• CPUs fracamente acoplados – é o caso da computação em grelha (grid) onde temos vários computadores independentes que são interligados por uma ou mais redes e que trabalham em colaboração para atingir um certo objetivo.



Multicore Processor Intel

Multiprocessadores

- sistema composto de vários processadores independentes;
- compartilham uma mesma memória por um barramento principal , podendo cada um ter a sua própria memória local;

Multicomputadores

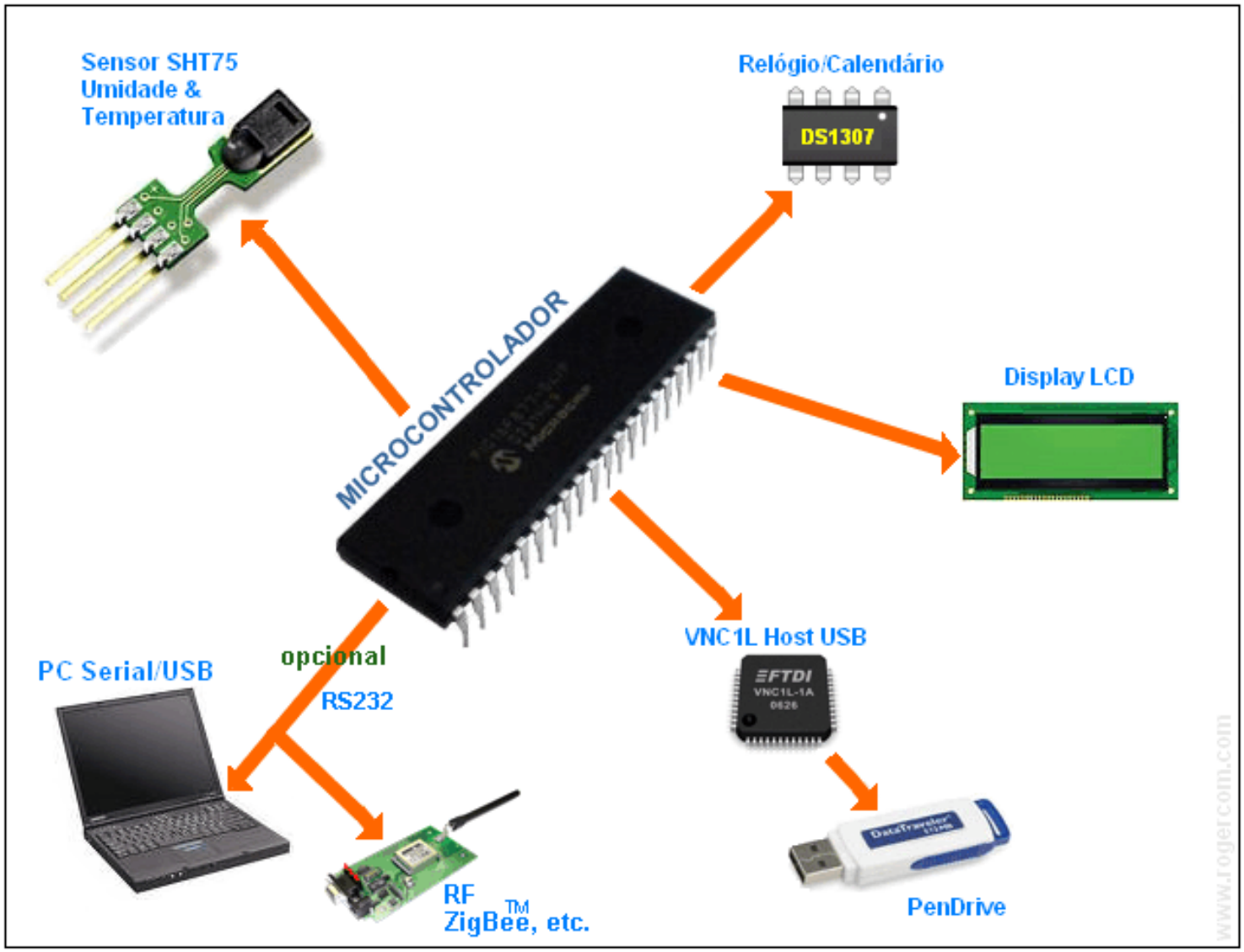
- sistemas com um grande número de computadores interconectados
- comunicação entre computadores é feita através de troca de mensagens
- existem em operação sistemas multicomputadores com cerca de 10000 computadores

É uma técnica que faz uso de clusterização de servidores para implementar o processamento paralelo. Um cluster é um conjunto de computadores que são conetados em rede e que comunicam através de um sistema operativo distribuído, trabalhando como se fossem uma máquina única.

Aula 9
2021-04-30

Microprocessadores vs Microcontroladores

Microprocessadores vs Microcontroladores



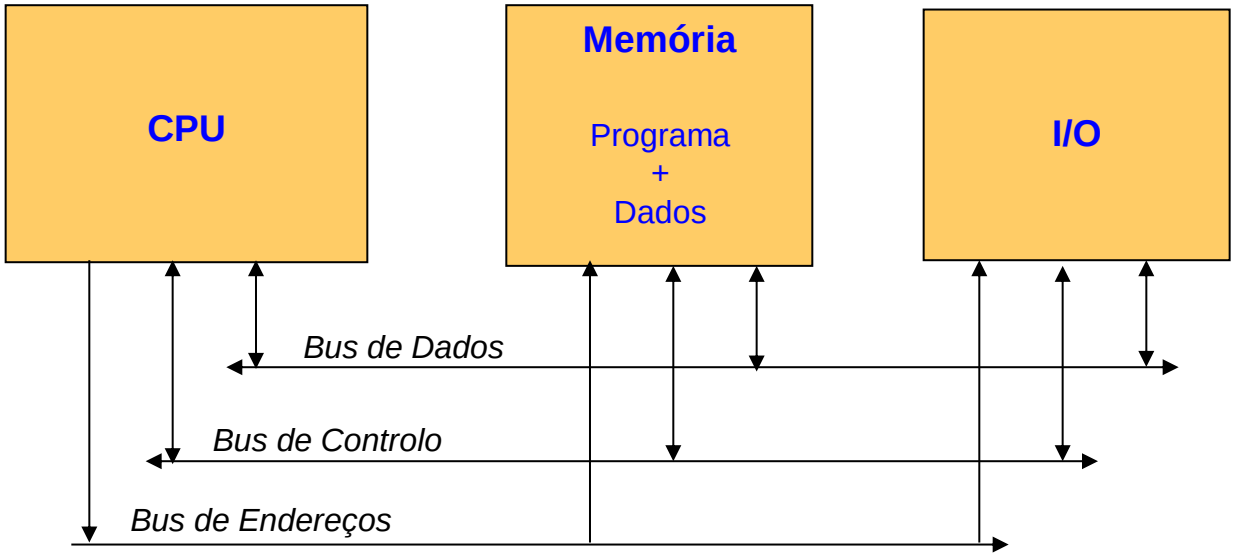
Microprocessadores - *Arquitetura de Von Neuman*

John von Neumann

- instruções e dados compartilham a mesma unidade física de memoria
- CISC – Complex Instruction Set Computer (mas também RISC)

- A vantagem é a simplicidade de acesso à memória - possui um barramento único para aceder à memória (endereços, dados e controlo)

- O grande inconveniente é o facto da memória do programa e dos dados ser comum, pois impede que se possa aceder ao programa e aos dados simultaneamente e muitas vezes o tamanho dos dados é diferente do tamanho das instruções

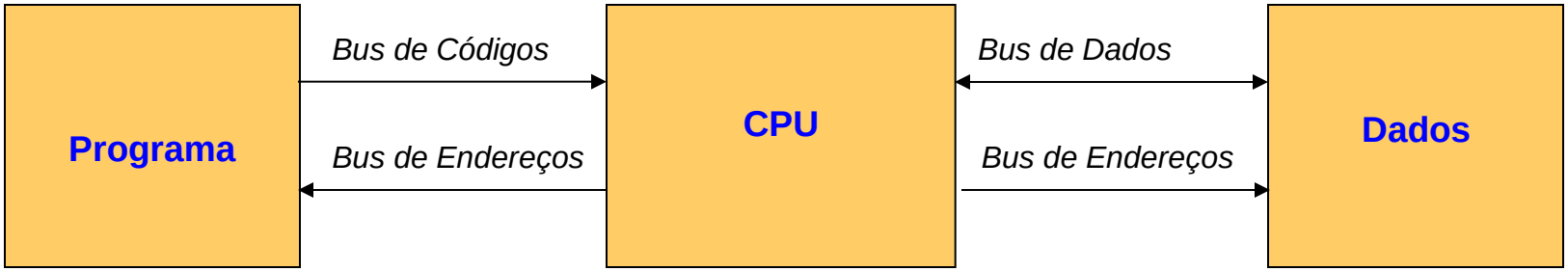


Utilização genérica

Microcontroladores - *Arquitetura de Harvard*

- instruções e dados são armazenados em memórias diferentes
- RISC – Reduced Instruction Set Computer

- vantagens: instruções mais rápidas → instruções e dados podem ser acedidos simultaneamente → aumento do desempenho!

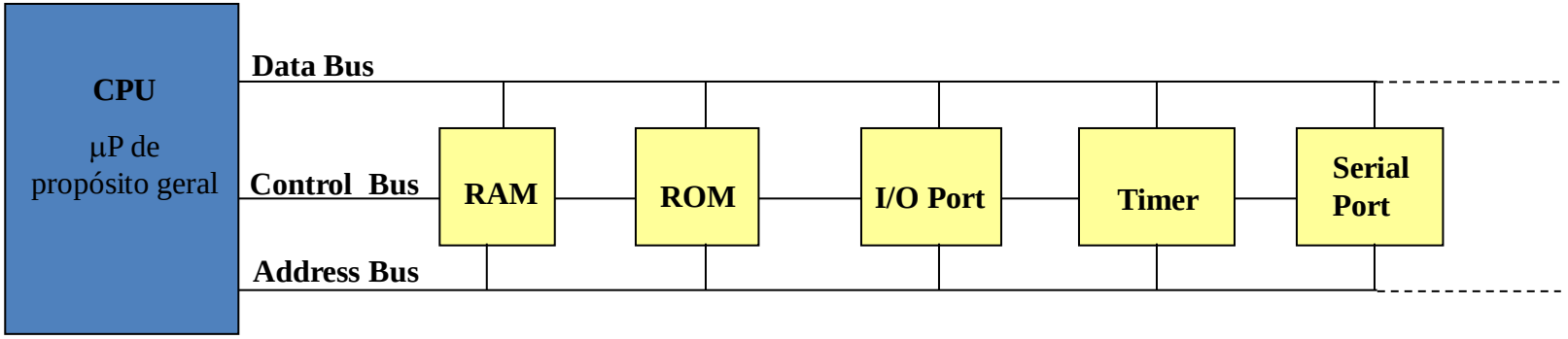


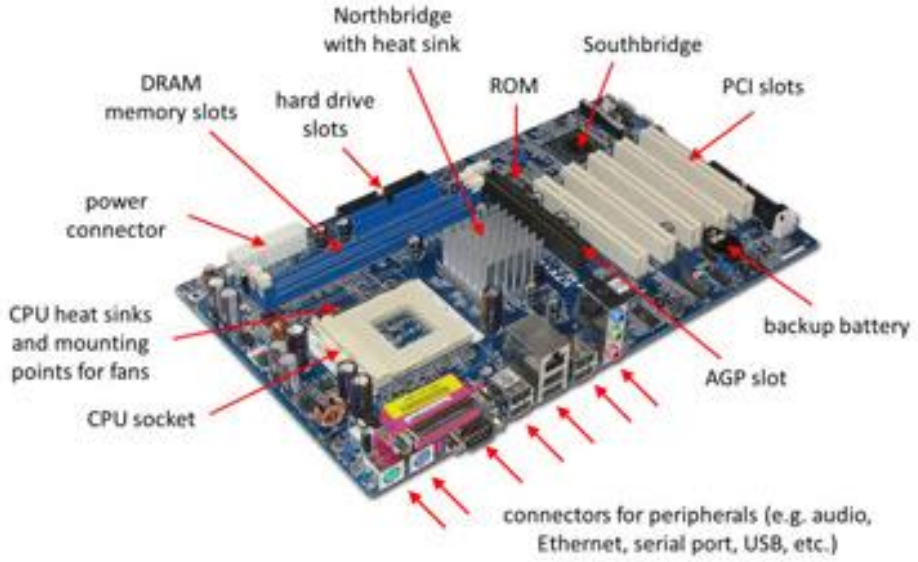
Utilização específica

Microprocessadores

Sistema microprocessador de propósito geral...

- CPU para computadores de propósito geral
- RAM-ROM, I/O, Portas, Timers, A/D & D/A... são exteriores ao CPU
- exemplo：Intel x86(Pentium), Motorola 680x0





Diversos chips na motherboard

Microcontroladores

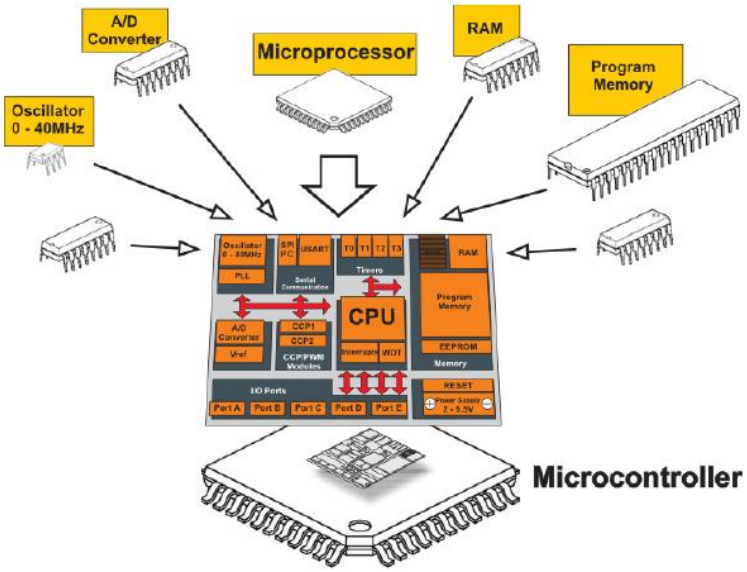
Sistema microcontrolador de uso específico...

- um computador em um único circuito integrado (chip) [SoC - System on Chip]
- RAM-ROM, I/O ports, A/D & D/A...etc. embutidos no chip
- exemplos：Motorola 6811, Intel 8051, Zilog Z8, PICs, AVRs, …

chip único

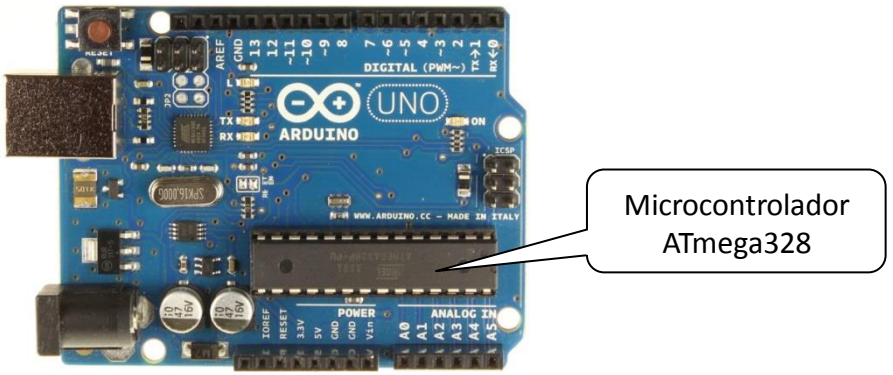
CPU	RAM	ROM
I/O Port	Timer	Serial Port

O microcontrolador integra num único componente os três elementos principais da arquitetura de um computador: CPU, memória e I/O



http://microduvida.blogspot.pt/p/microprocessadores-vs.html

Um só chip na motherboard



Microcontroladores

- podem ser vistos como dispositivos de propósito(objetivo) específico

- usados em tarefas “simples” sem grandes requisitos de processamento, a nível de rapidez e de tipo de instruções

- integram num único circuito integrado (CI , chip):
  - processador;
  - memória;
  - portas de I/O;
  - contadores (contam impulsos);
  - timers (temporizadores, contam tempo);
  - conversores A/D (analógico→digital) e D/A(digital→analógico)
  - …

- tornam-se assim mais baratos e compactos que os circuitos com microprocessador e outros integrados associados (memória, controladores, etc)

Microprocessadores *vs* Microcontroladores

Microprocessador

- CPU => *stand-alone*
  RAM, ROM, I/O, timers... separados
- projetista pode decidir a quantidade de ROM, RAM e *ports* de I/O;
- expansível
- versatilidade, uso geral

Microcontrolador

- CPU, RAM, ROM, I/O, timer... estão integrados em um só *chip*
- quantidade fixa de elementos *on-chip (*ROM, RAM, I/O *ports*)
- para aplicações onde custo, potência e espaço são fatores críticos;
- uso específico

Microcontroladores - exemplos

8051 (INTEL)

- 8 bits
- um dos mais utilizados na prática
- conjunto reduzido de instruções (RISC)
- usado numa grande diversidade de equipamentos (ex:máquinas de costura)

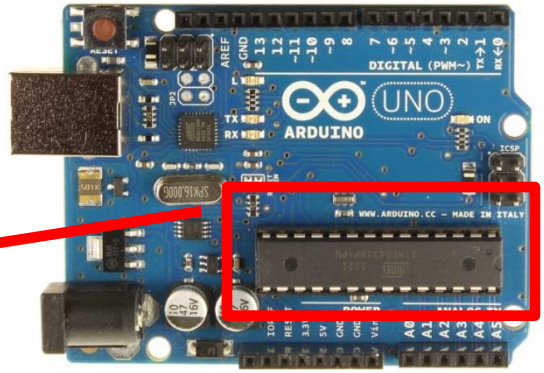
PIC (Microchip Technology - https://www.microchip.com/)

- melhor desempenho
- possui um conjunto de instruções e funções mais elaborados
- baratos (há versões que custam menos de 1€)
- a Microchip fabrica uma família de PIC de 8, 16, 24 e 32 bits
- fáceis de utilizar:
  - a nível de programação
  - a nível de integração com outros componentes eletrónicos

Microcontroladores

ARDUINO

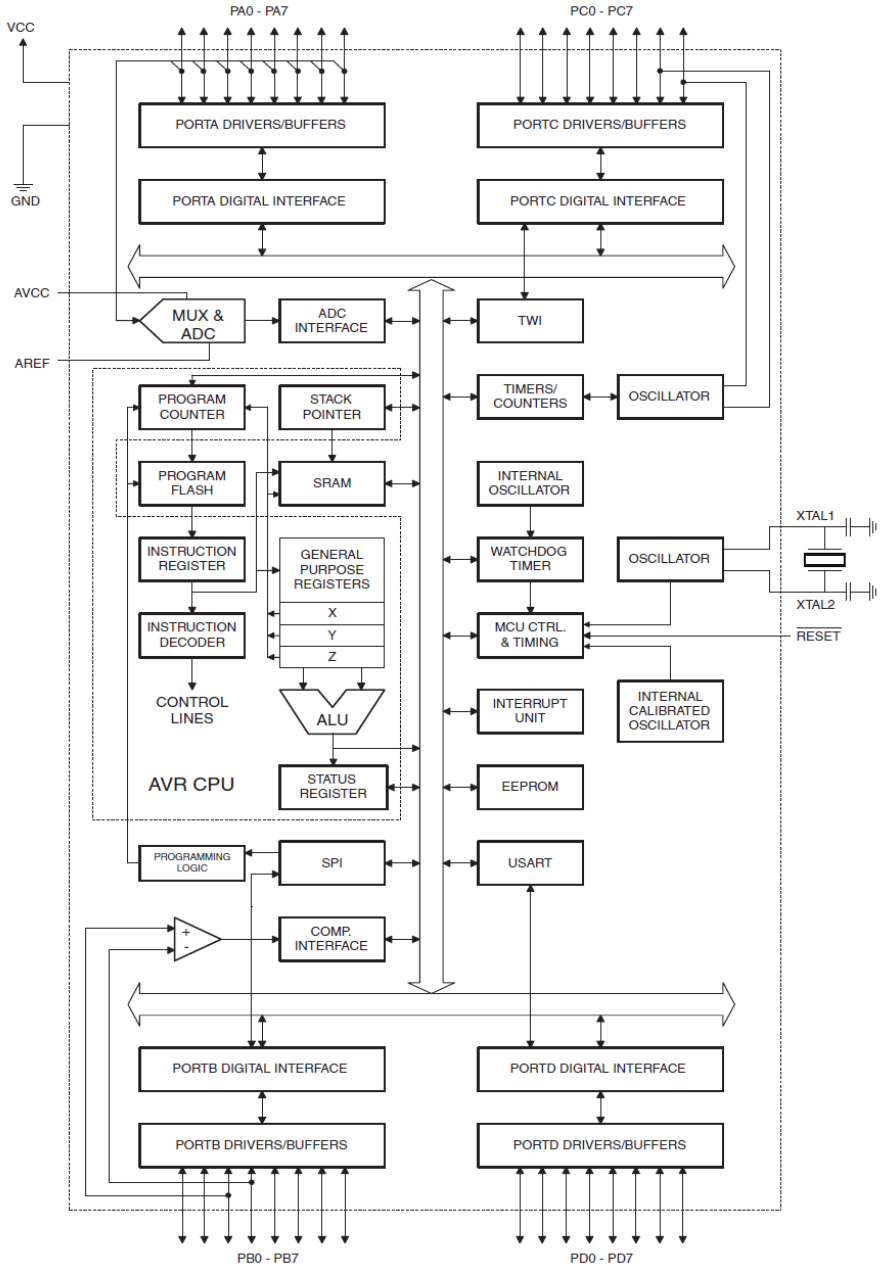
- microcontrolador ATMEL ATMEGA328 : família AVR(Microchip), 8 bits, arquitetura RISC
- 32 KB de Flash , 2 KB de RAM e 1 KB de EEPROM
- Portas I²C, série, I/O digital e analógico, A/D e D/A
- Clock de 16 MHz (máx=20MHz)



ATMEL ATMEGA328

Arduino UNO : Microcontrolador ATMEL ATMEGA328P (Microchip)

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf



ATmega328/P is a low-power CMOS 8-bit microcontroller based on the AVR® enhanced RISC architecture

Advanced RISC Architecture

- 131 Powerful Instructions
- Most Single Clock Cycle Execution
- 32 x 8 General Purpose Working Registers
- 32KBytes Flash program Memory
- 1KBytes EEPROM
- 2KBytes Internal SRAM
- Data Retention: 20 years at 85°C/100 years at 25°C(1)
- 23 Programmable I/O Lines
- One Programmable Serial USART
- Two 8-bit Timer/Counters + One 16-bit Timer/Counter
- 6-channel 10-bit ADC

Processadores Digitais de Sinais (DSPs)

Características:

- diferem dos microprocessadores na arquitetura de hardware, software e no conjunto de instruções, o qual é optimizado para o tratamento digital de sinais
- são empregues em aplicações que exigem processamento de sinais em tempo real

Usos:

- telecomunicações (filtros, compressão, multiplexação e cancelamento de eco);
- processamento de áudio (gravação em estúdio, sintetizadores, mixers, filtros e reconhecimento de voz);
- processamento de imagem (principalmente na área médica);
- instrumentação e controlo (precisão das medidas e controlo industrial).

Processamento de áudio digital

Texas Instruments *TMS320*

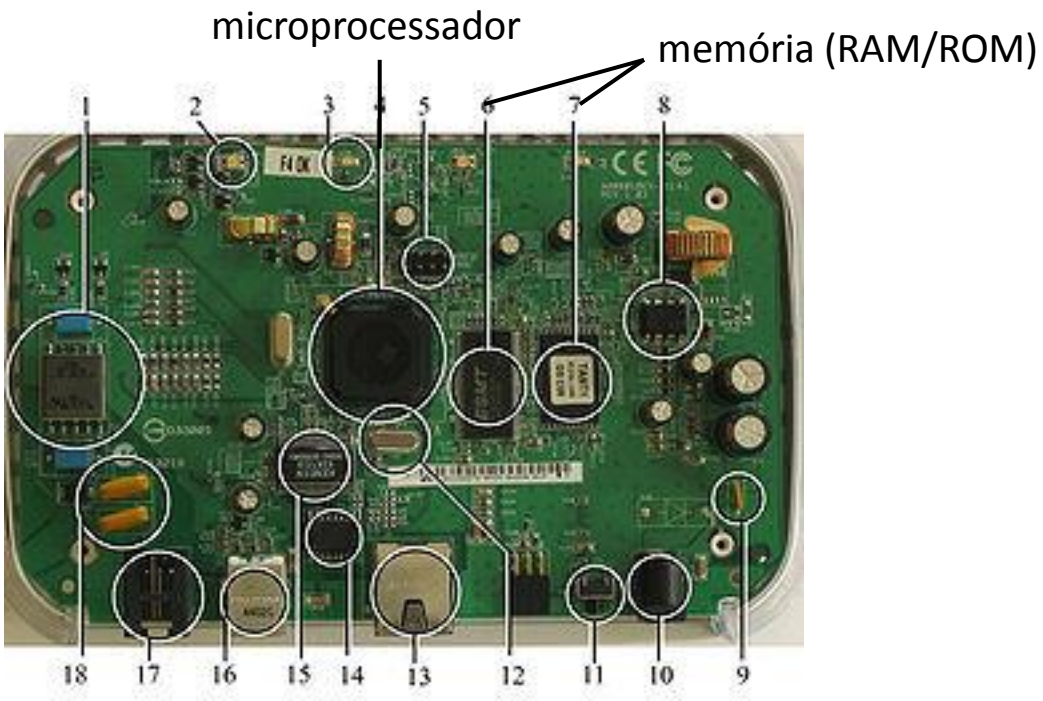
Consola Nintendo

Sistemas embebidos/embutidos (embedded systems)

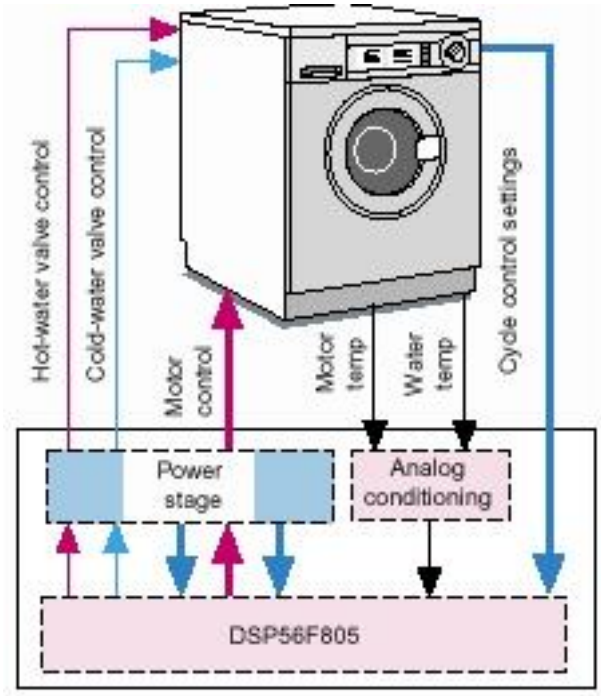
Características:

- sistema embebido significa que o processador está embutido na aplicação;
- um produto embebido utiliza um microprocessador/microcontrolador (ou DSP) para fazer uma ou poucas tarefas dedicadas;
- existe somente uma aplicação de software que normalmente está gravada em ROM (firmware);
- normalmente existe a interação com o meio ambiente ou com o operador;
- exemplos: impressora, teclado, consola jogos, telemóvel, modem…

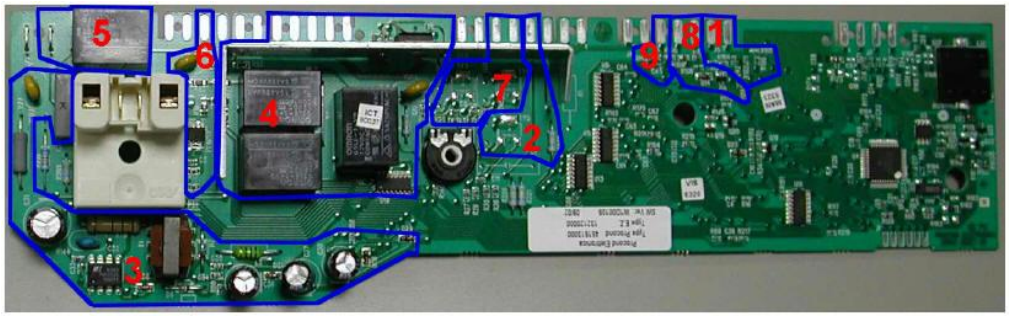
sistema embebido
modem ADSL



Controlo de uma máquina de lavar roupa



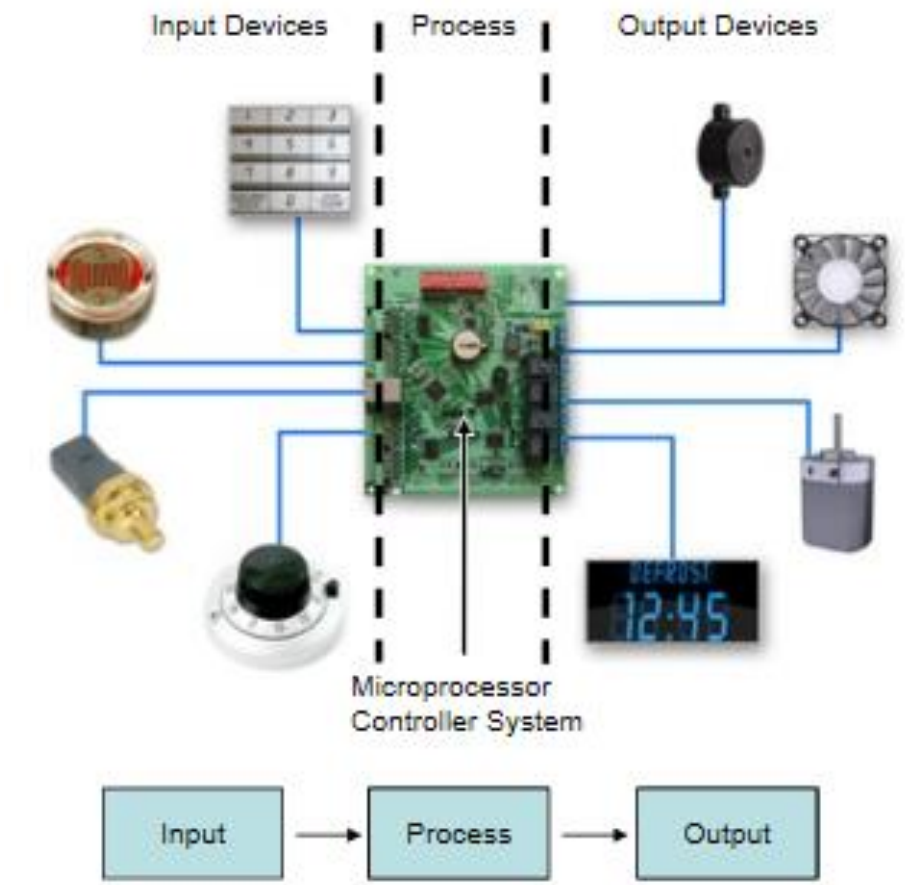
DSP56F800 - 16-bit Digital Signal Controllers



1. NTC washing temperature sensor 2. Drain pump 3. Power supply 4. Motor	5. Heating element 6. Door safety interlock 7. Water fill solenoids 8. Tachymetric generator (motor) 9. Drum positioning system (top-loaders)

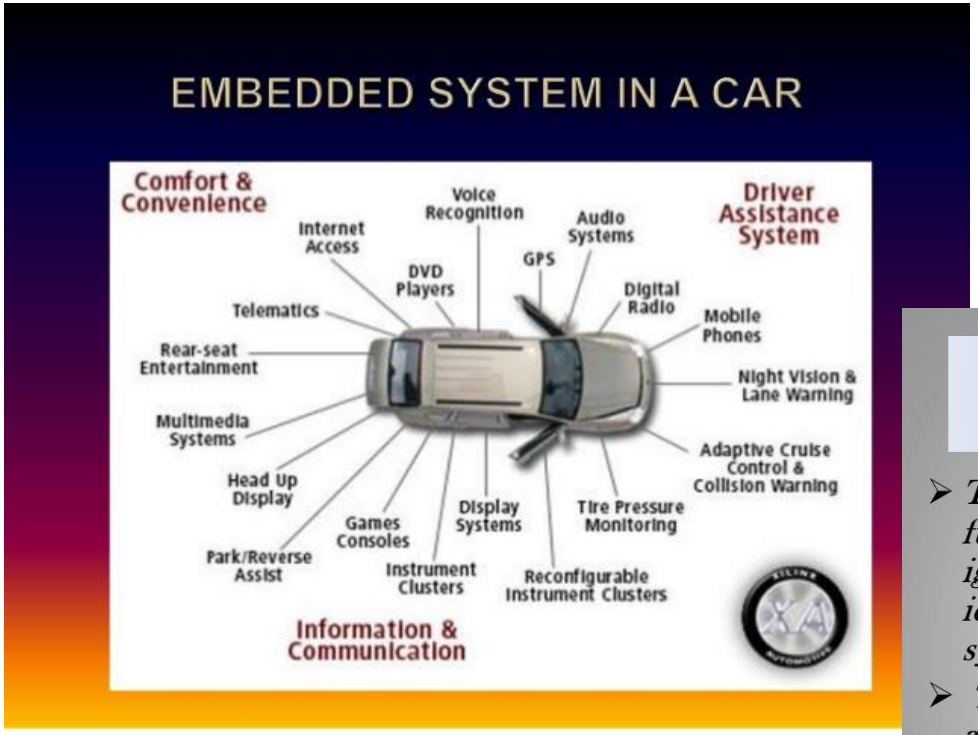
Sistemas embebidos/embutidos - exemplos

Controlo de um forno microondas



Sistemas embebidos/embutidos - exemplos

Aplicações em automóveis



ECU / centralina

Arduino : carta controladora programável

Arduino → plataforma de prototipagem de código aberto (hardware e software) criada em 2005 pelo italiano Massimo Banzi.

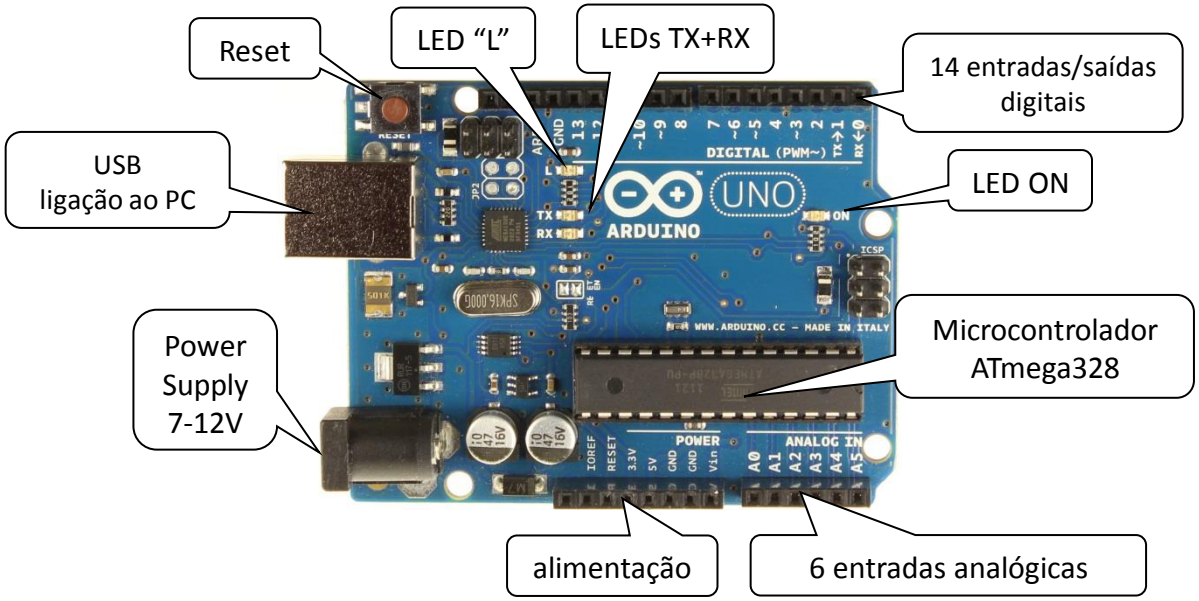
É destinado a artistas, designers, hobbistas, professores e qualquer pessoa interessada em criar objetos ou ambientes interativos.

Objetivo principal : criar uma plataforma hardware de baixo custo apoiada por um sistema de programação de código aberto, para possibilitar o desenvolvimento de protótipos de baixo custo.

Através de sinais provenientes de sensores, o Arduino pode detetar o estado do ambiente que o rodeia e após processamento desses sinais é capaz de controlar indicadores luminosos ou motores e uma diversidade de outros atuadores.

Existe uma enorme comunidade de utilizadores, com inúmeras propostas de ideias e projetos.

Arduino : modelo UNO

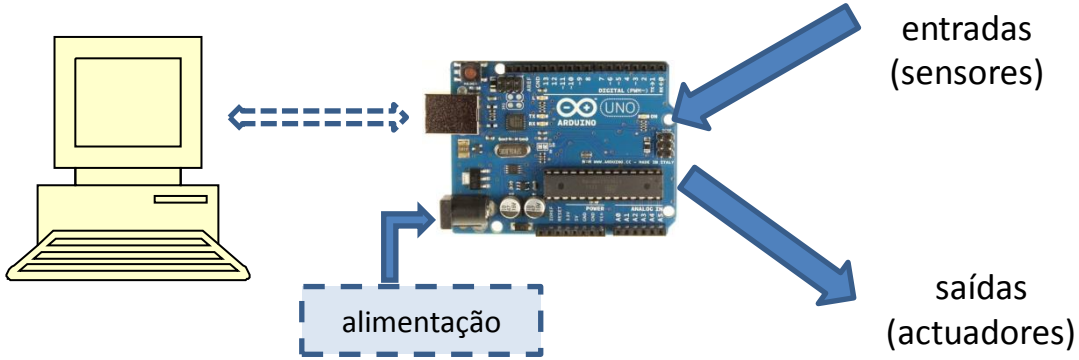


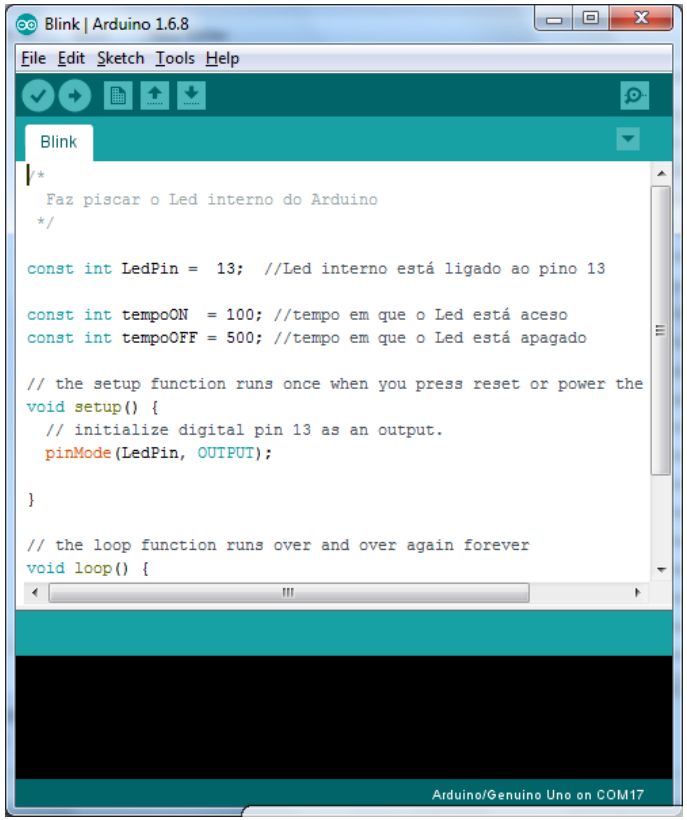
Características Principais

- Baixo custo (≈25€)
- Microcontrolador: ATmega328, 16MHz
- Tensão de alimentação: USB ou fonte externa
- Memória: Flash(código)=32KB, SRAM(variáveis)=2KB, EEPROM=1KB
- Entradas/saídas digitais: 14, entradas analógicas: 6
- Ligação ao PC: USB
- Comunicaões: UART & I^2C
- Conector de expansão
- Diversos módulos externos (*Shields*): controlo motores, comunicações s/fio, …
- Existem diversas variantes: Due, Uno, Duemilenove, Mega, ADK, Lillypad, Nano,...
- Existem “clones” com funções melhoradas (ex: chipKIT)
- Fornecedores nacionais: PT Robotics(http://www.ptrobotics.com/), SAR (http://www.botnroll.com/), ElectroFun(https://www.electrofun.pt/arduino)

Arduino : carta controladora programável

Depois de programado pode funcionar autonomamente ou ligado a um sistema externo (PC)





Ambiente de desenvolvimento(Java): ling. tipo C/C++, gratuito, muitas bibliotecas existentes: Ethernet, LCD, DateTime, …

KIT disponível no LTC: Arduino Physical Computing Kit

IDE: http://arduino.cc/en/Main/Software

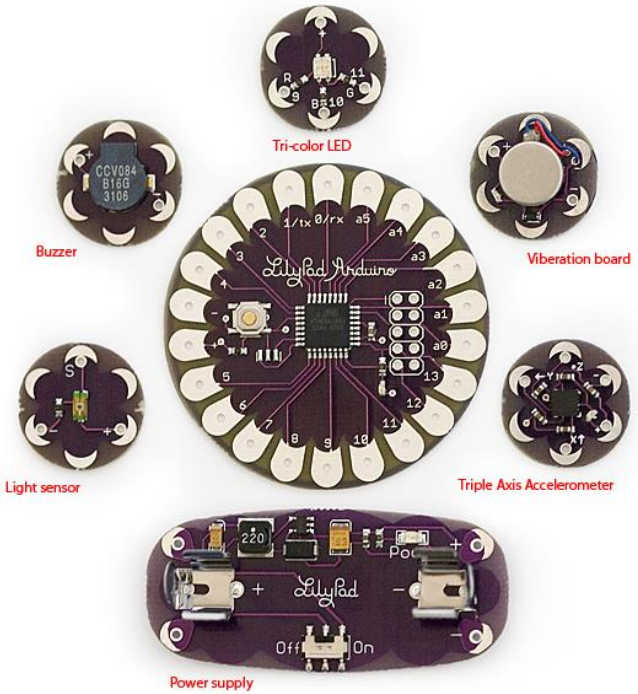
Arduino - Variantes

Uno

Mini

Chipki

Due



Arduino **Lilypad**: versão para aplicação no vestuário

Arduino – Shields (ampliam as funções da placa base)

Motor

GSM

GPS

WiFi

LCD+keyboard

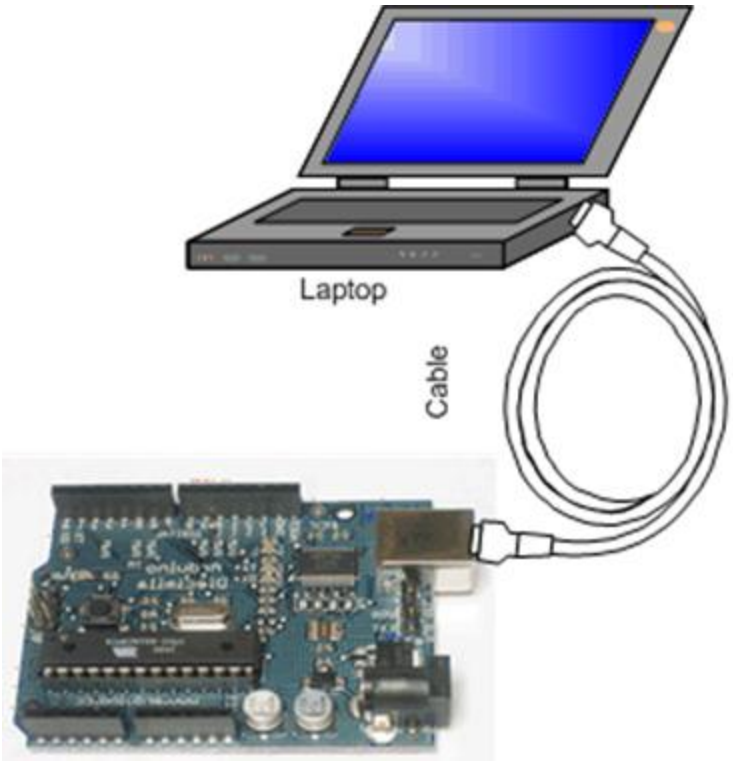
Ethernet

Arduino - Variantes

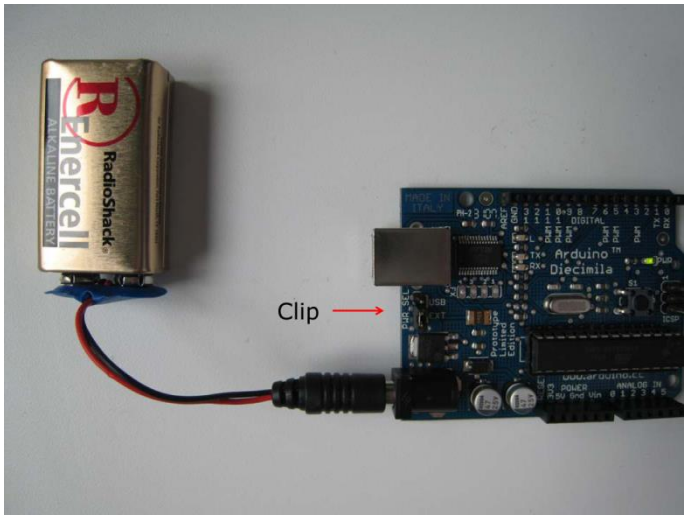
Plataforma de e-Health: Kit para Arduino (e Raspberry Pi)

http://www.cooking-hacks.com/index.php/ehealth-sensors-complete-kit-biometric-medical-arduino-raspberry-pi.html

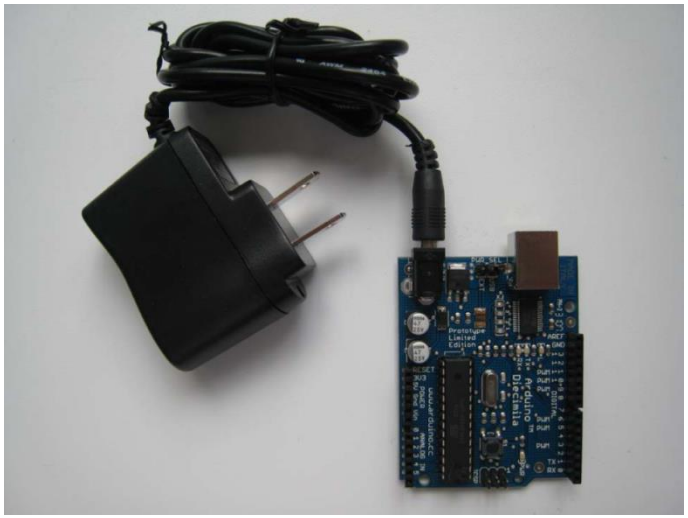
Arduino – alimentação eléctrica



cabo USB

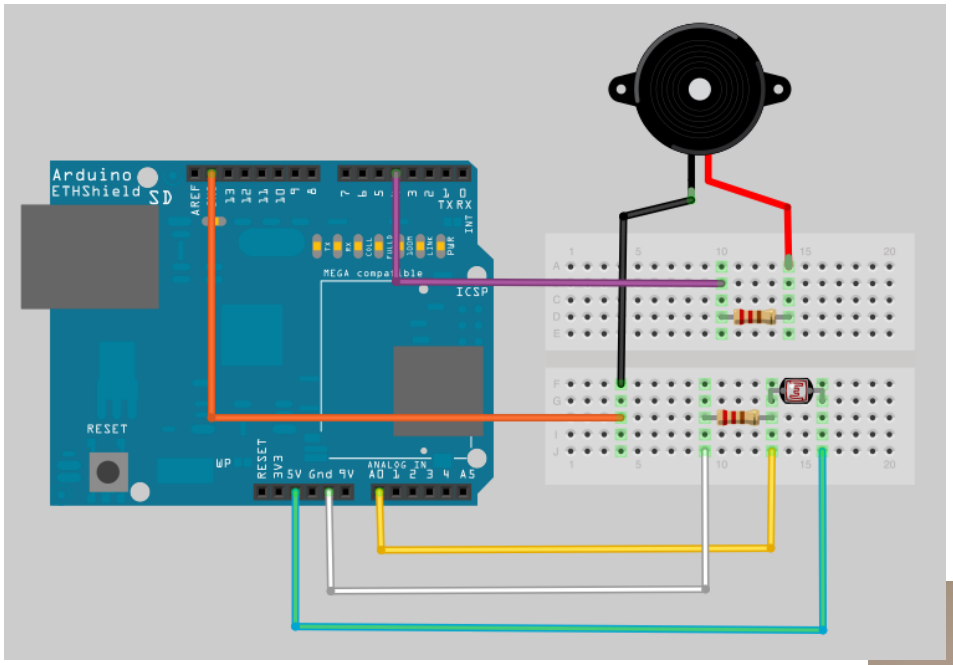


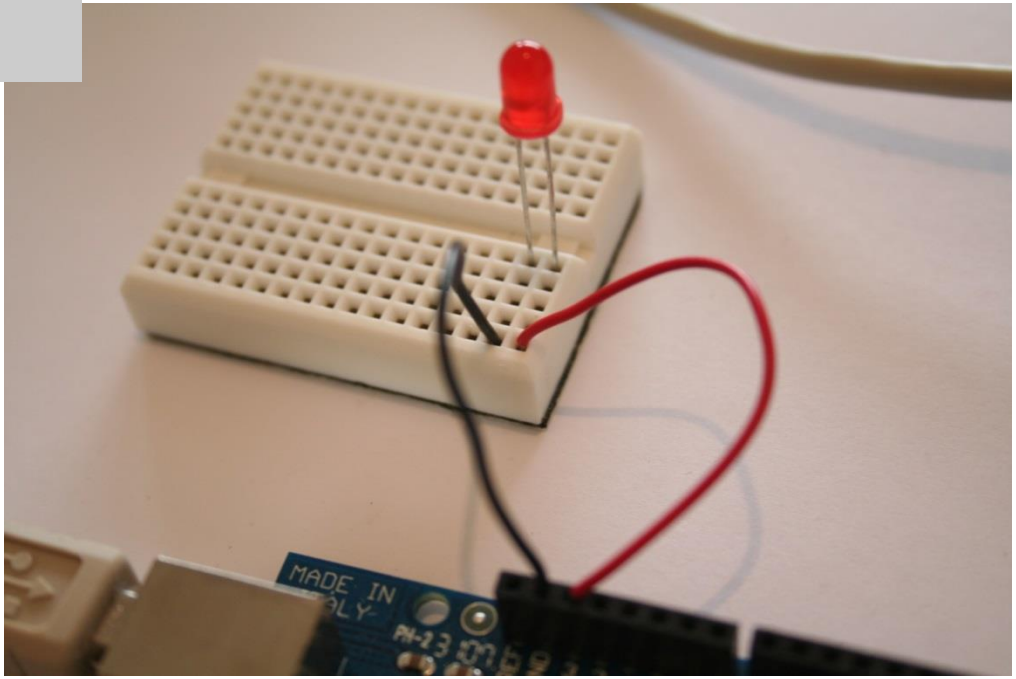
pilha / bateria



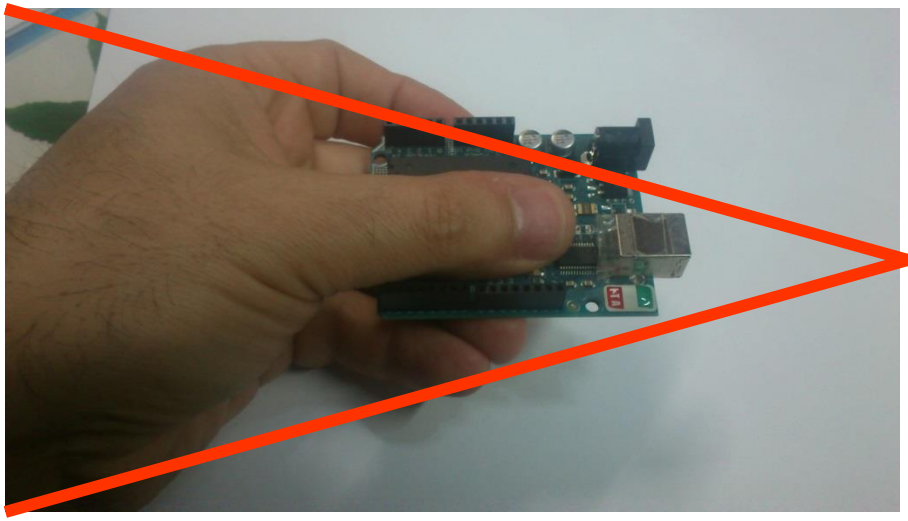
fonte de alimentação (power supply)

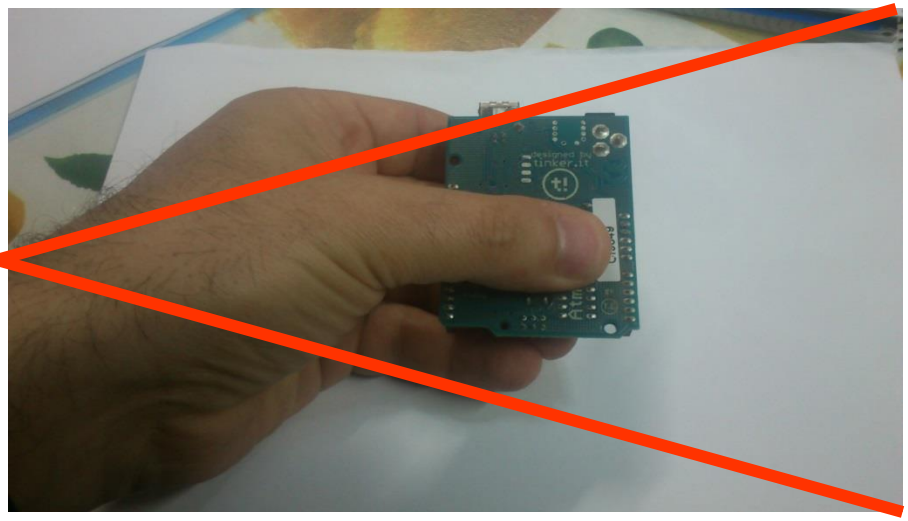
Arduino - ligações

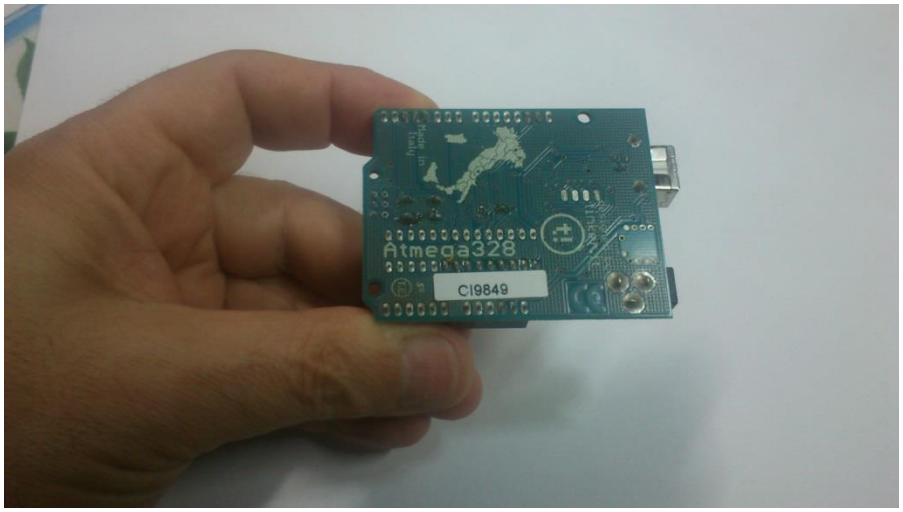




Arduino - manipulação

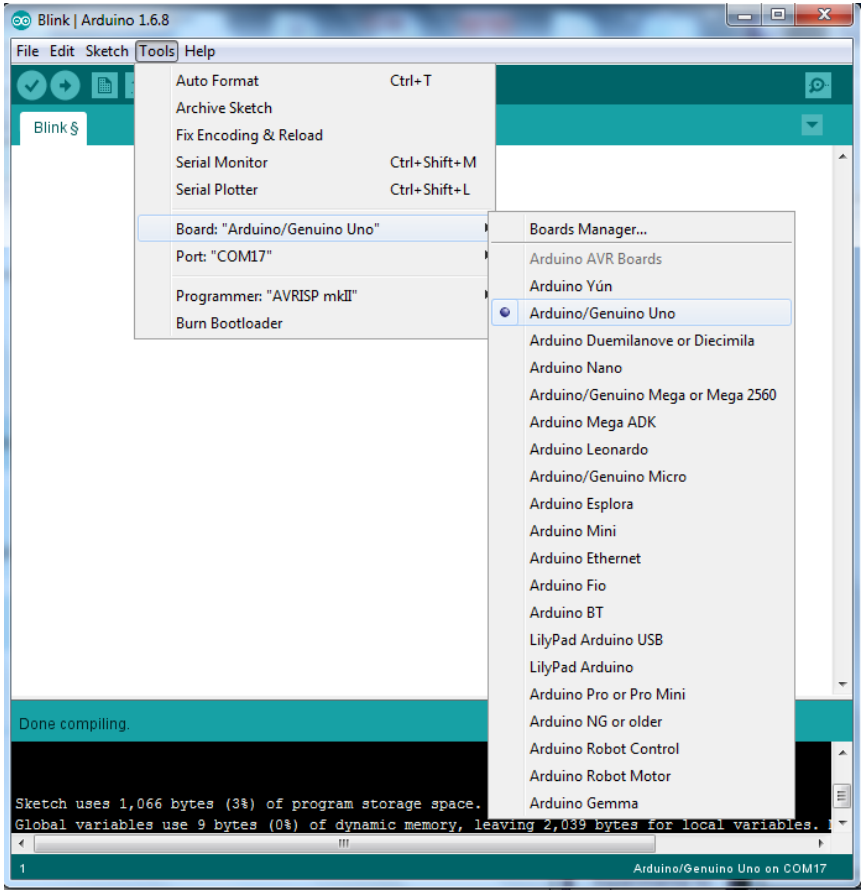




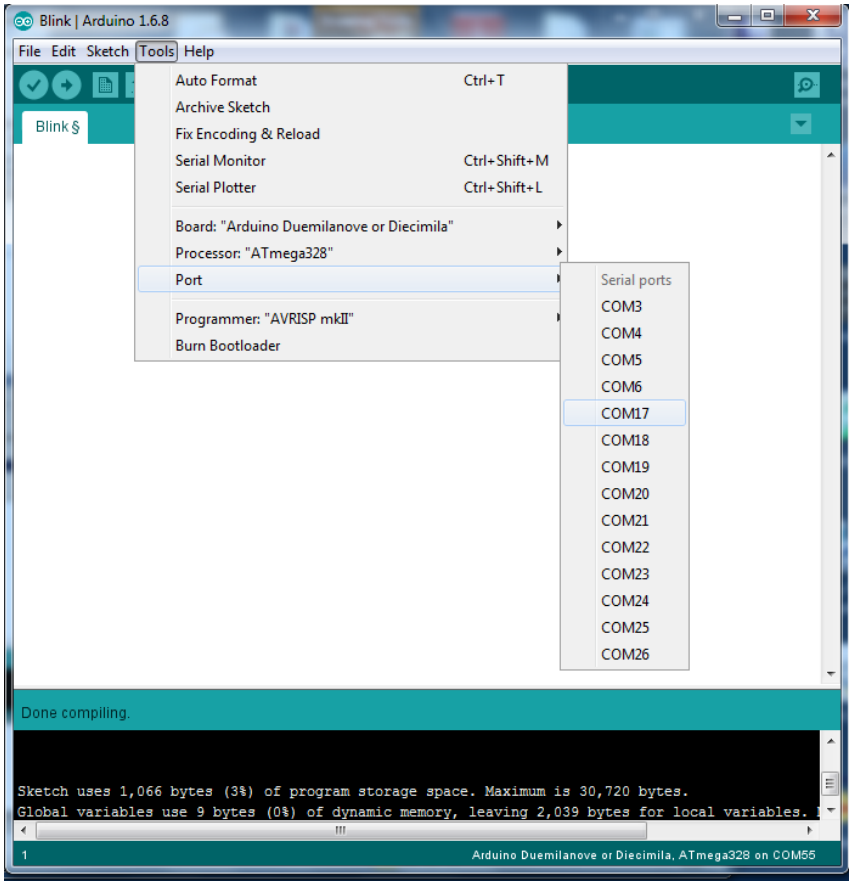


Arduino - configuração

Selecção do modelo da placa Arduino



Porta para comunicação Arduino-PC



Arduino - programação

Programa para Arduino = “SKETCH”

Estrutura de um sketch

<declarações> : declaração de constantes, variáveis, tipos, etc (OPCIONAL)

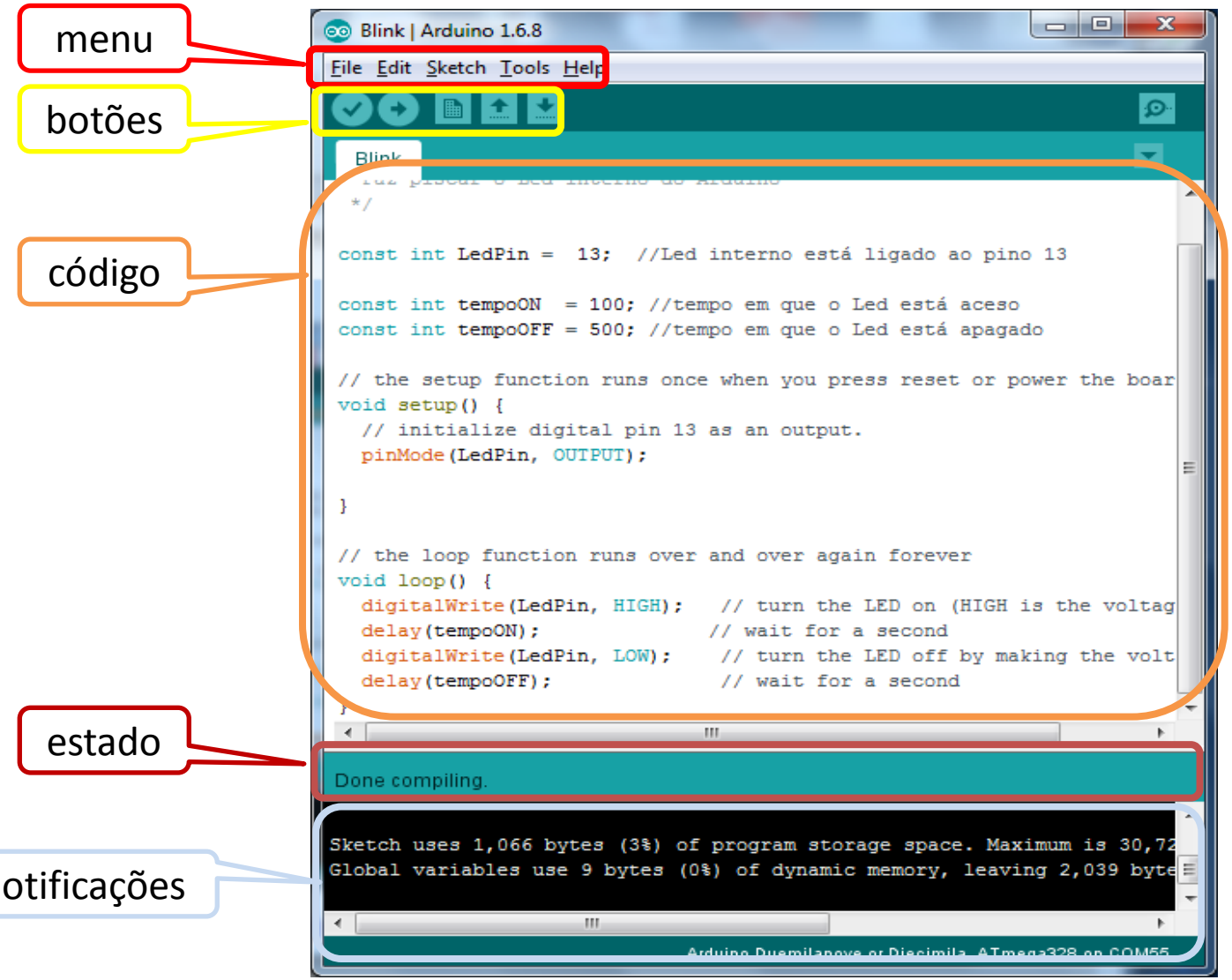
```
void setup ( )
{
  codigo executado uma só vez; serve principalmente para efectuar inicializações
}
```

```
void loop ( )
{
  código executado de modo contínuo (em ciclo) até que a alimentação seja desligada (ou reset).
}
```

Comentários

```
//linha de comentário

/*
  texto de comentário
*/
```

Arduino IDE(Integrated Development Environment)

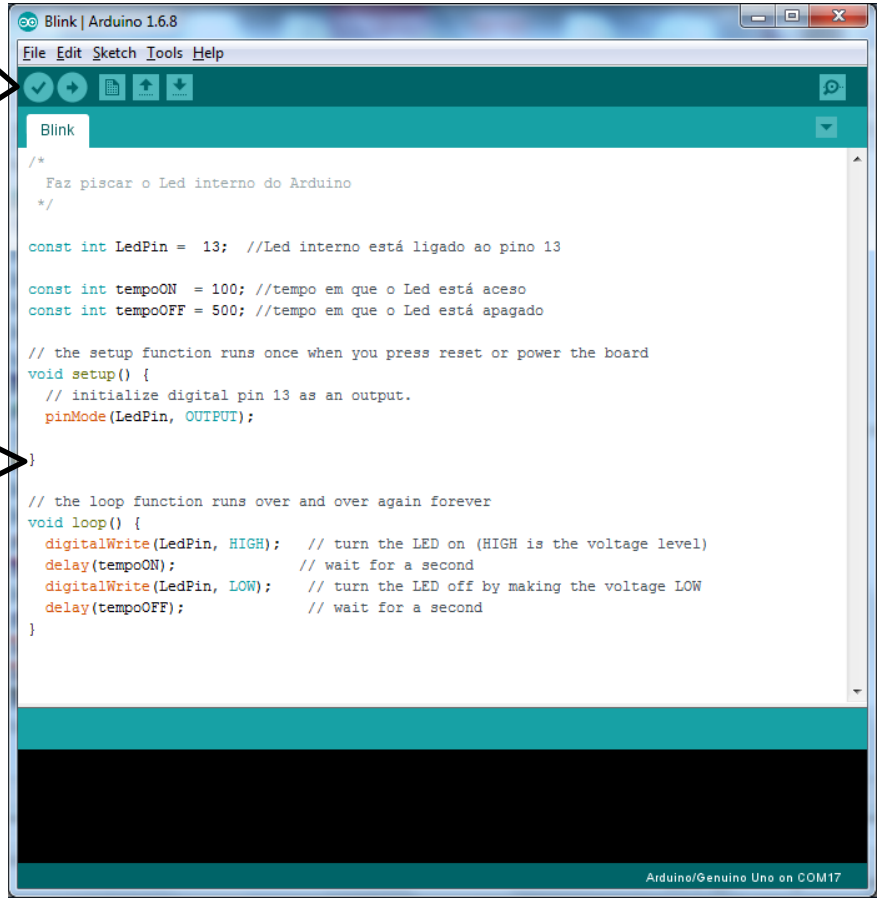


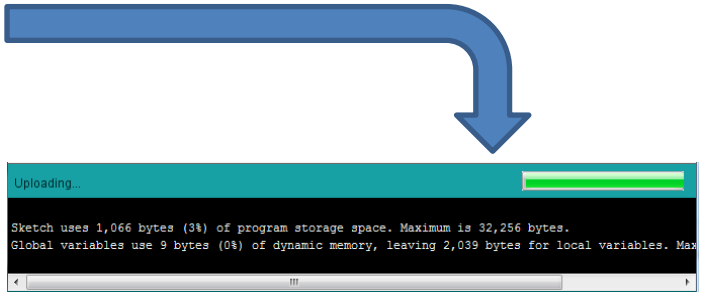
Ver: http://arduino.cc/en/Guide/Environment

Arduino – utilização

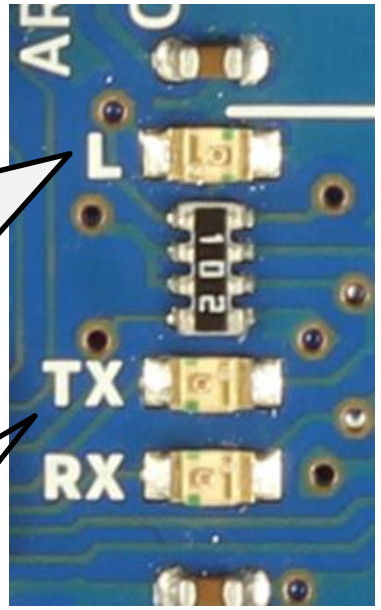
(2) carregar código (verificar)

(1) escrever código





(4) Led L pisca segundo o código do sketch



(3) TX/RX cintilam

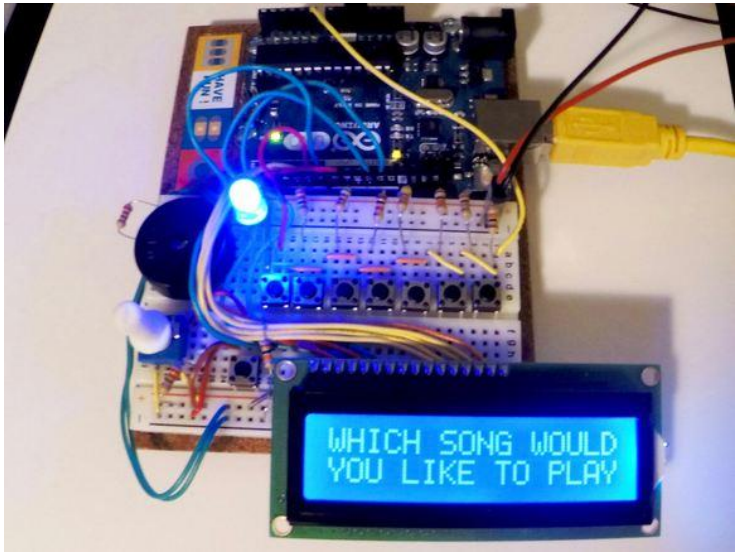
Sketch = “Blink” (Led L a piscar)

Arduino : projetos diversos

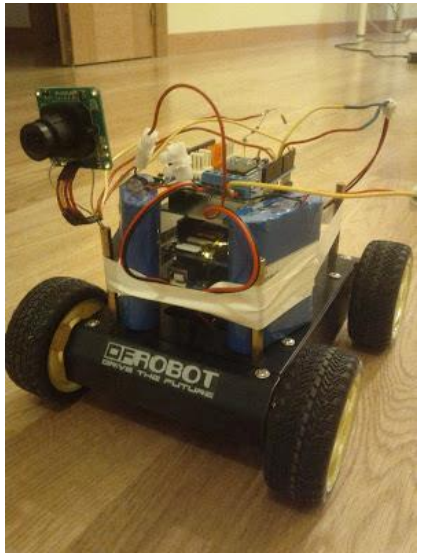
http://playground.arduino.cc/Projects/Ideas

http://runtimeprojects.com/

Electronic Piano



Internet controlled Arduino car



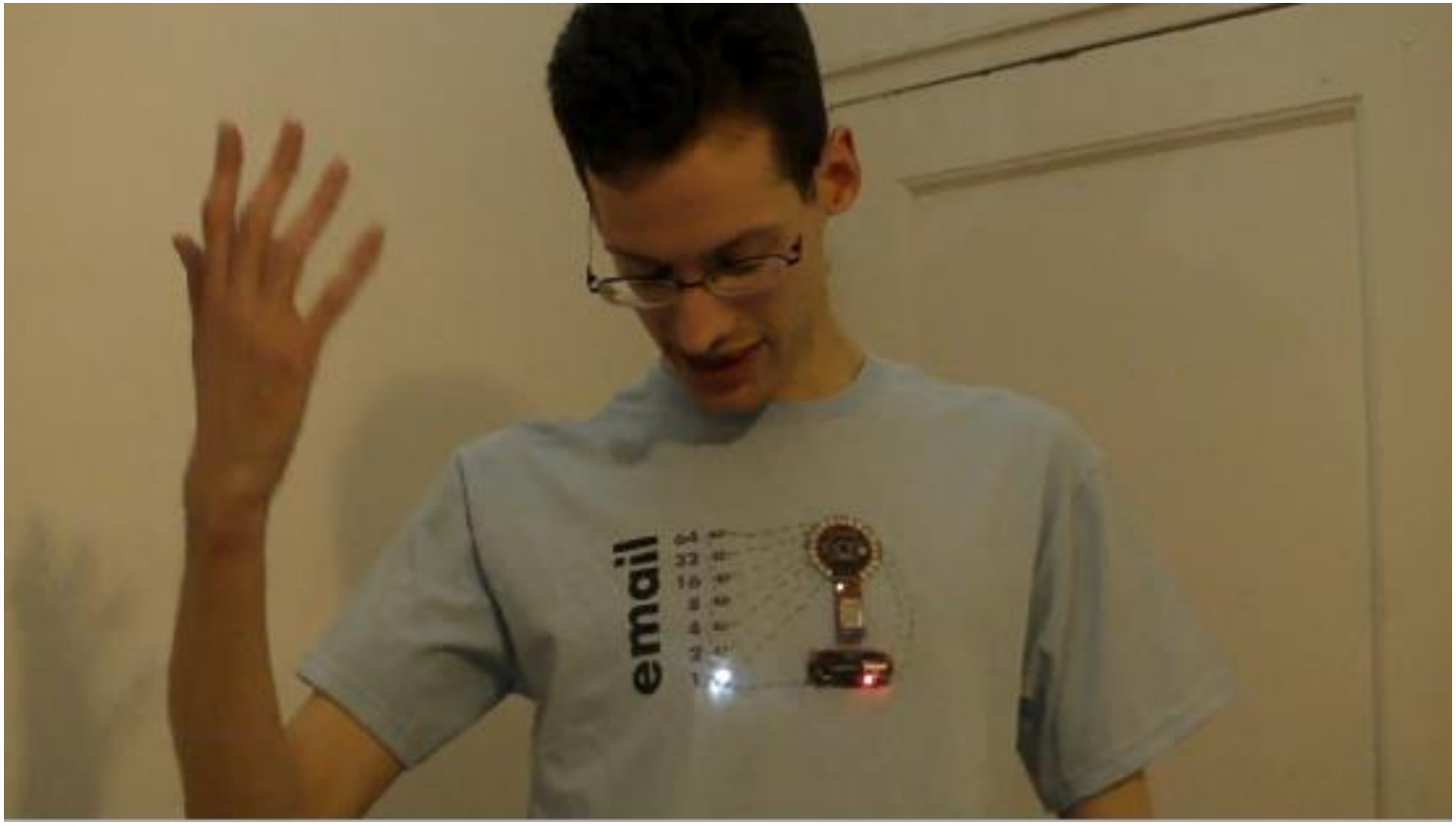
Arduino : projetos diversos

LilyPad Example: LED Biking Jacket

http://www.trendhunter.com/trends/lilypad-arduino-diy-programmable-fashion

Arduino : projetos diversos

T-shirt modded to let you know when you have new emails



http://www.engadget.com/2010/03/30/t-shirt-modded-to-let-you-know-when-you-have-new-emails/

Arduino : resumo de comandos (cheat-sheet)

arduino-cheat-sheet

ARDUINO CHEAT SHEET

JEROEN DOGGEN, AP UNIVERSITY COLLEGE ANTWERP

Structure

```
void setup()
void loop()
```

Control Structures

```
if(x<5){}
for(int i = 0; i < 255; i++ ){}
while((x < 6 ){}
```

Further Syntax

```
//        Single line comment
/*..*/    Multi line comment
#define ANSWER 42
#include <myLib.h>
```

General Operators

```
=     assignment
+, -  addition, substraction
*, /  multiplication, division
%     modulo
==    equal to
!=    not equal to
<     less than
<=    less than or equal to
```

Pointer Access

```
&  reference operator
*  dereference operator
```

Bitwise Operators

```
&  bitwise AND
|  bitwise OR
^  bitwise XOR
~  bitwise NOT
```

Compound Operators

```
++  Increment
--  Decrement
+=  Compound addition
&=  Compound bitwise AND
```

Constants

```
HIGH, LOW
INPUT, OUTPUT
true, false
```

53 : Decimal
B11010101 : Binary
0x5BA4 : Hexadecimal

Data Types

void	
boolean	0, 1, false, true
char	e.g. 'a' -128 → 127
unsigned char	0 → 255
int	-32.768 → 32.767
unsigned int	0 → 65535
long	-2.147.483.648 → 2.147.483.647
float	-3,4028235E+38 → 3.402835E+38
sizeof (myint)	returns 2 bytes

Arrays

```
int myInts[6];
int myPins[]=2,4,8,5,6;
int myVals[6]=2,-4,9,3,5;
```

Strings

```
char S1[15];
char S2[8]='A','r','d','u','i','n','o';
char S3[8]='A','r','d','u','i','n','o','\0';
char S4[]="Arduino";
char S5[8] = "Arduino";
char S6[15] = "Arduino";
```

Conversion

char() **int()** **long()**
byte() **word()** **float()**

Qualifiers

static	Persist between calls
volatile	Use RAM (nice for ISR)
const	Mark read-only
PROGMEM	Use flash memory

Interrupts

attachInterrupt(interrupt, function, type)
detachInterrupt(interrupt)
boolean(interrupt)
interrupts()
noInterrupts()

Advanced I/O

tone(pin, freqhz)
tone(pin, freqhz, duration_ms)
noTone(pin)
shiftOut (dataPin, clockPin, how, value)
unsigned long **pulseIn**(pin, [HIGH,LOW])

Time

unsigned long **millis()** 50 days overflow
unsigned long **micros()** 70 min overflow
delay(ms)
delayMicroseconds(us)

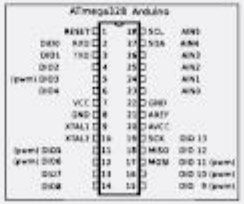
Math

min(x,y) **max**(x,y) **abs**(x)
sin(rad) **cos**(rad) **tan**(rad)
pow(base, exponent)
map(val, fromL, fromH, toL, toH)
constrain(val, fromL, toH)

Pseudo Random Numbers

randomSeed(seed)
long **random**(max)
long **random**(min, max)

ATmega328 Pinout



I/O Pins

	Uno	Mega
# of IO	14 + 6	54 + 11
Serial Pins 3	0 - RX, 1 - TX	RX1 → RX4
Interrupts	2,3	2,3,18,19,20,21
PWM Pins	5,6 - 9,10 - 3,11	0 → 13
SPI (SS, MOSI, MISO, SCK)	10 → 13	50 → 53
I2C (SDA, SCK)	A4, A5	20,21

Analog I/O

```
analogReference(EXTERNAL, INTERNAL)
analogRead(pin)
analogWrite(pin, value)
```

Digital I/O

```
pinMode(pin, [INPUT, OUTPUT])
digitalRead(pin)
digitalWrite(pin, value)
```

Serial Communication

```
Serial.begin(speed)
Serial.print("Text")
Serial.println("Text")
```

Websites

```
forum.arduino.cc
playground.arduino.cc
arduino.cc/en/Reference
```

Arduino Uno Board

http://jeroendoggen.github.io/latex/2013/06/24/arduino-cheat-sheet.html (2016)

Arduino : links úteis

- **Site oficial Arduino** : http://www.arduino.cc

Comandos da linguagem Arduino: http://www.arduino.cc/en/Reference/HomePage

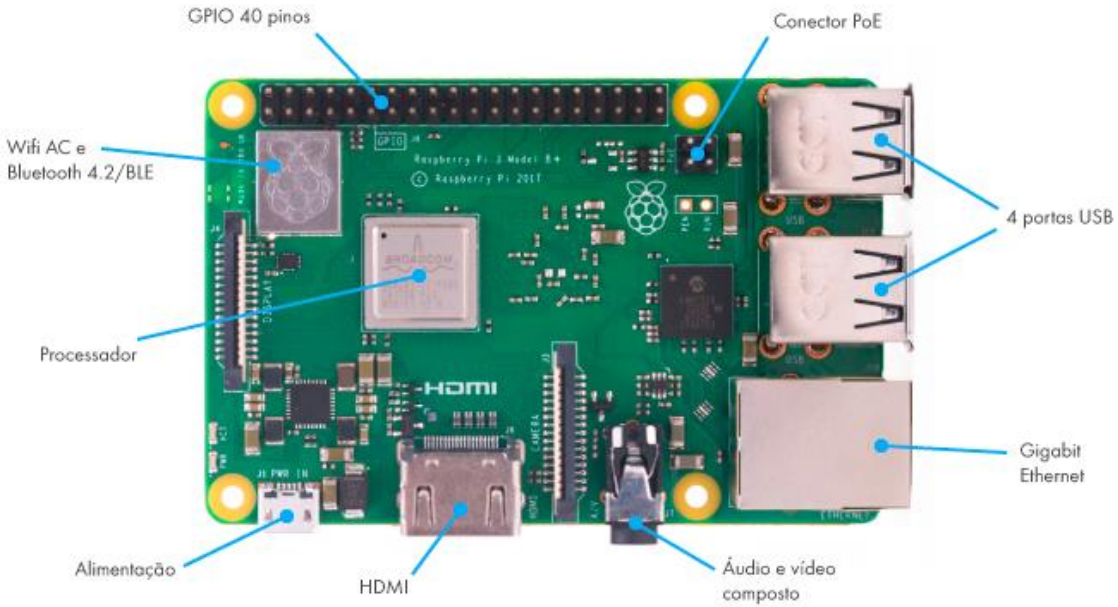
- **Simuladores**

  Tinkercad : https://www.tinkercad.com/

- **Fritzing** - para desenhar esquemas elétricos: www.fritzing.org

- **Processing** -linguagem de programação usada para escrever programas com interface gráfica:
  https://playground.arduino.cc/Interfacing/Processing

Raspberry Pi :https://pt.wikipedia.org/wiki/Raspberry_Pi



Raspberry Pi é um computador do tamanho de um cartão de crédito, que se conecta a um monitor de computador ou TV, e usa um teclado e um mouse padrão; foi desenvolvido no Reino Unido pela *Fundação Raspberry Pi*.

Todo o hardware é integrado numa única placa.

O Raspberry Pi 3 B+ é baseado em um SoC(*system on a chip)* Broadcom BCM2837(64 bit, quad core) a 1.4GHz, 1 GB de RAM, Bluetooth 4.2.

O projeto não inclui uma memória não-volátil - como um disco rígido - mas possui uma entrada de cartão SD para armazenamento de dados.

Pode correr o Linux (Snappy Ubuntu Core) ou o Microsoft Windows 10 IoT edition.

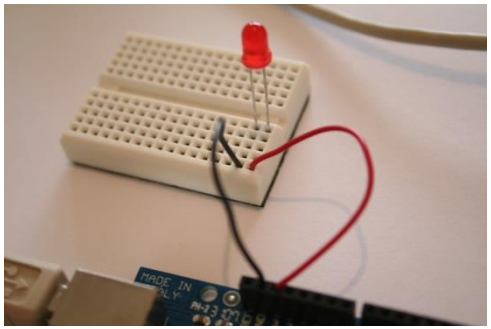
www.raspberrypi.org

Aula 10
2021-05-07

Arduino - TinkerCad

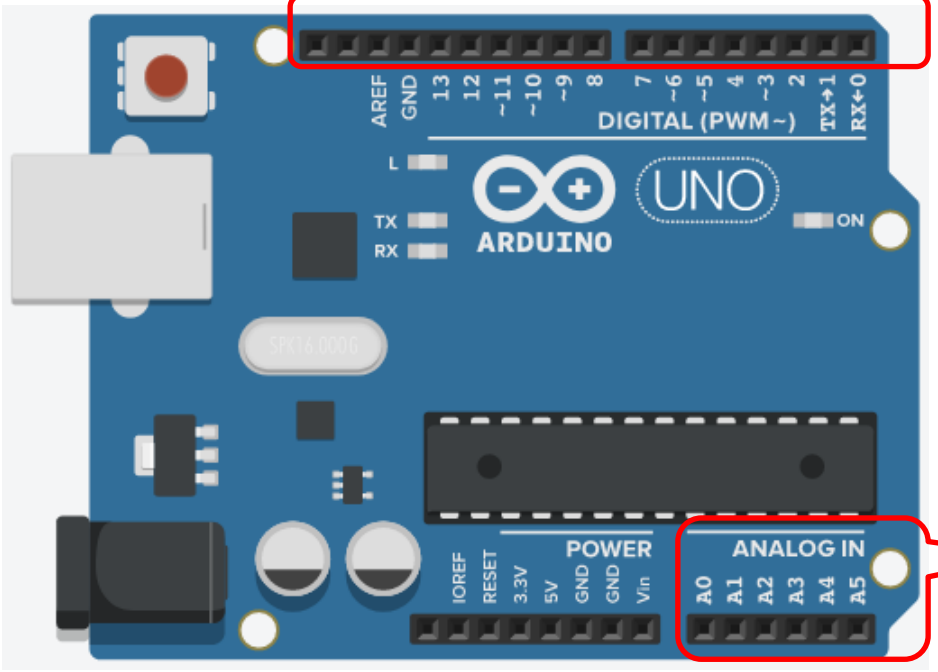
Arduino - entradas/saídas digitais & entradas analógicas

Arduino real



entradas/saídas digitais aceitam valores “0” (0V) ou “1” (5V)

Arduino Tinkercad



Conversão A/D (analógico/digital)

6 entradas analógicas

resolução 10 bits (0...1023)

aceitam valores entre 0 e 5V

Entrada(analógica)	Saída(digital)
0V	0
...	
2.5V	512
...	
5V	1023

Arduino - programação

Programa para Arduino = “SKETCH”

Estrutura de um sketch

<declarações> : declaração de constantes, variáveis, tipos, etc (OPCIONAL)

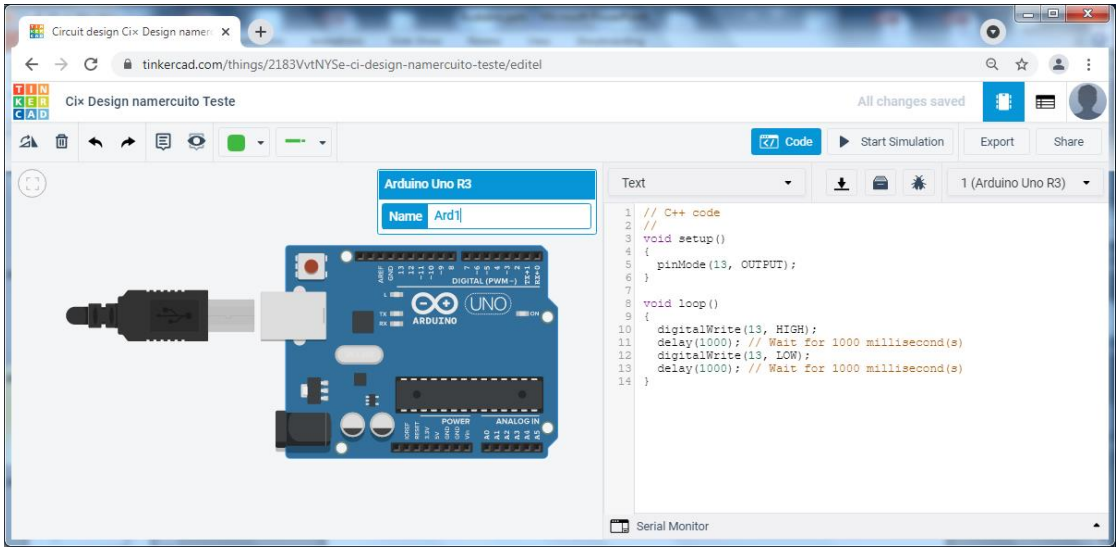
```
void setup ( )
{
  codigo executado uma só vez; serve principalmente para efectuar inicializações
}
```

```
void loop ( )
{
  código executado de modo contínuo (em ciclo) até que a alimentação seja desligada (ou reset).
}
```

Comentários

```
//linha de comentário

/*
  texto de comentário
*/
```

1º exemplo: piscar o LED L (pino 13) interno ao Arduino



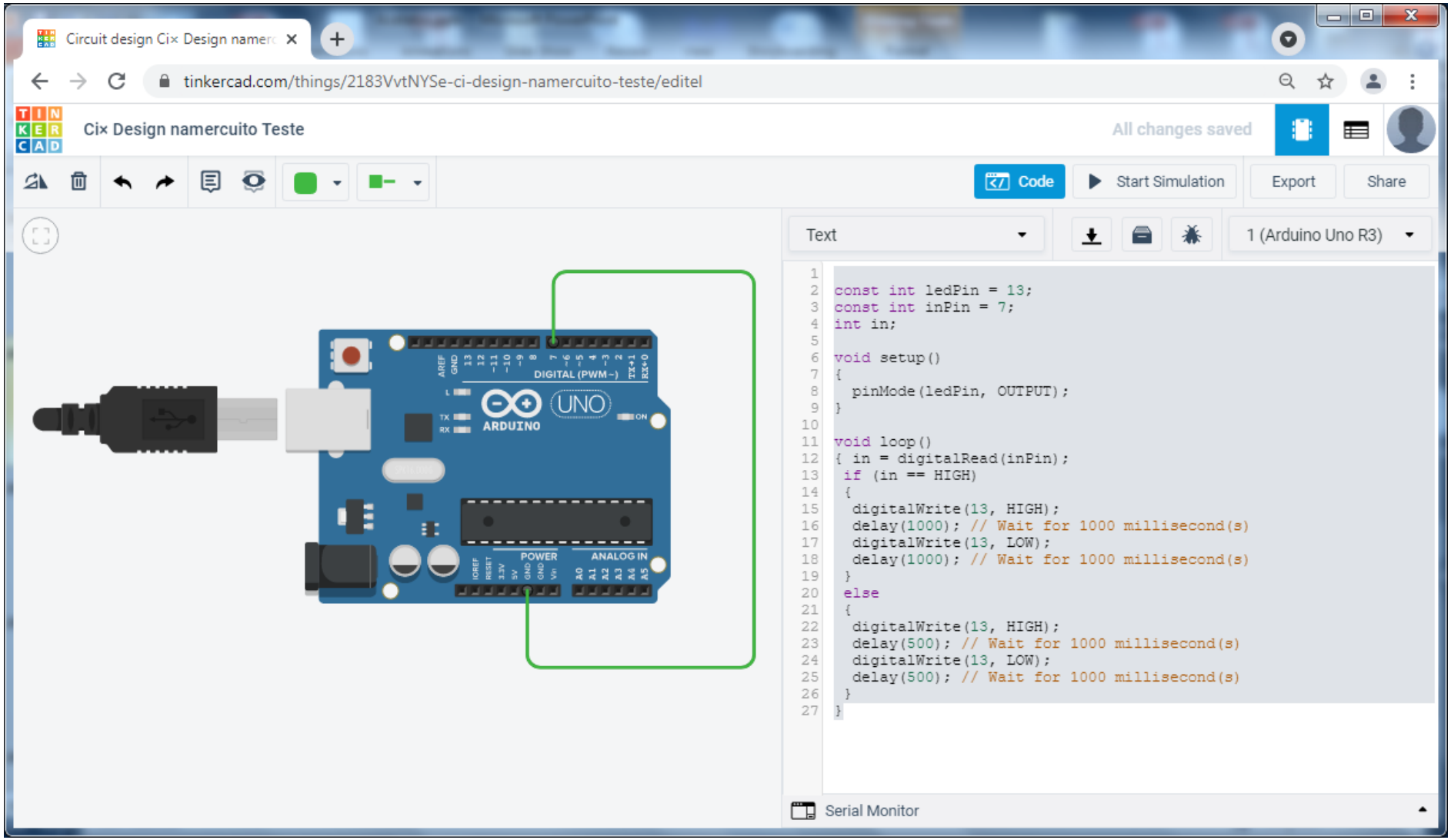
Nota

LED L - ligado internamente ao pino 13 (em princípio, este pino não deve ser usado para mais nada)

```
//não há declarações

void setup()  //executada uma só vez
{
 pinMode(13, OUTPUT); //define o pino 13 como de saída
}

void loop()  //executada em ciclo
{
 digitalWrite(13, HIGH);  //coloca pino 13 a “1”
 delay(1000);             // Wait for 1000 millisecond(s)
 digitalWrite(13, LOW);   //coloca pino 13 a “0”
 delay(1000);             // Wait for 1000 millisecond(s)
}
```

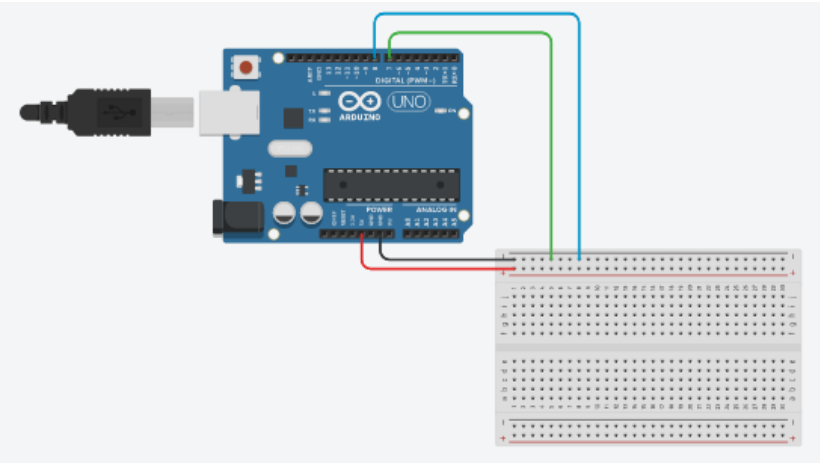
```
const ledPin = 13; //declaração de uma constante

void setup()
{
 pinMode(ledPin, OUTPUT);
}

void loop()
{
 digitalWrite(ledPin, HIGH);
 delay(1000);
 digitalWrite(ledPin, LOW);
 delay(1000);
}
```

2º exemplo: piscar o LED L interno ao Arduino em função de uma entrada digital



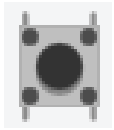
3º exemplo: piscar o LED L interno ao Arduino em função de duas entradas digitais

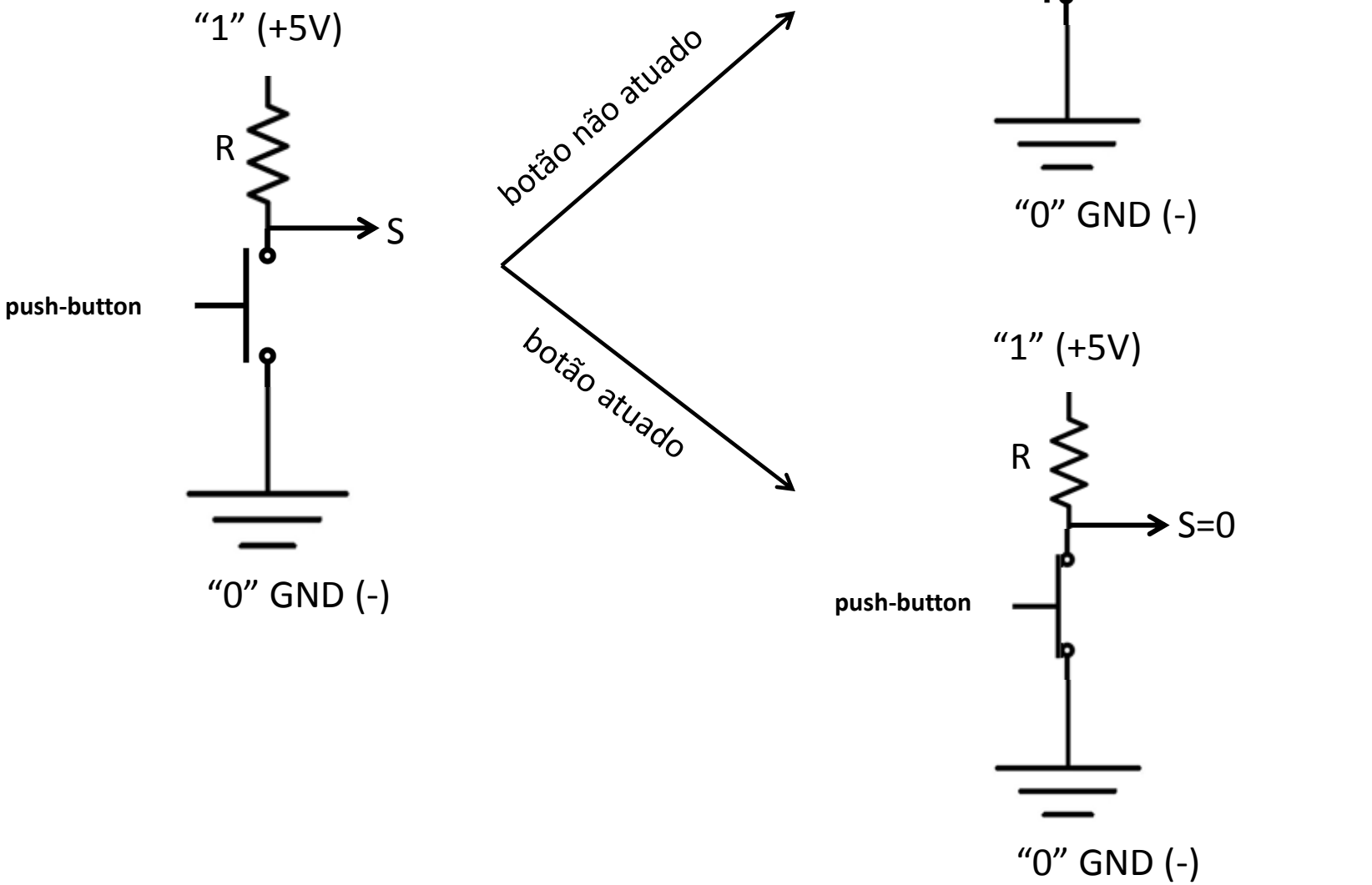
In1	In2	LED L
0	0	apagado
0	1	piscar lento
1	0	piscar rápido
1	1	sempre aceso



```
const int ledPin = 13;
const int inPin1 = 7;
const int inPin2 = 8;
int in1;
int in2;

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
{ in1 = digitalRead(inPin1);
  in2 = digitalRead(inPin2);

 if ((in1==LOW) and (in2==LOW))
 {
  digitalWrite(13, LOW);
 }
 if ((in1==LOW) and (in2==HIGH))
 {
  digitalWrite(13, HIGH);
  delay(1500); // Wait for 1000 millisecond(s)
  digitalWrite(13, LOW);
  delay(1500); // Wait for 1000 millisecond(s)
 }
 if ((in1==HIGH) and (in2==LOW))
 {
  digitalWrite(13, HIGH);
  delay(500); // Wait for 1000 millisecond(s)
  digitalWrite(13, LOW);
  delay(500); // Wait for 1000 millisecond(s)
 }
 if ((in1==HIGH) and (in2==HIGH))
 {
  digitalWrite(13, HIGH);
 }

}
```

Push Button




"1" (+5V)
R
S
push-button
"0" GND (-)
botão não atuado
botão atuado
"1" (+5V)
R
S = 1 (≈+5V)
push-button
"0" GND (-)
"1" (+5V)
R
S=0
push-button
"0" GND (-)

4º exemplo: piscar o LED L interno ao Arduino em função de um botão de pressão (push-button)