

Aula 3

Prof. Daniel Cavalcanti Jeronimo

Fundamentos de Programação

CP41F

Fundamentos de computação.

Linguagens de alto nível.

Compilador/Interpretador. História e
Introdução a C.

Universidade Tecnológica Federal do Paraná (UTFPR)

Engenharia de Computação – 1º Período

2016.1

Plano de Aula

- Fundamentos de computação.
 - Computador
 - Analógico
 - Digital
 - Arquitetura
 - Memória
 - Codificação
 - Processadores
- Linguagens de alto nível.
- Compilador/Interpretador.
- História e Introdução a C.

Fundamentos de Computação

- **Computador:**

Dispositivo que recebe informações e a manipula utilizando sequências de instruções, os programas, para obter um resultado.



Dispositivo de Antikythera:
computador de posições
astronômicas e eclipses
(~200 a.C.)

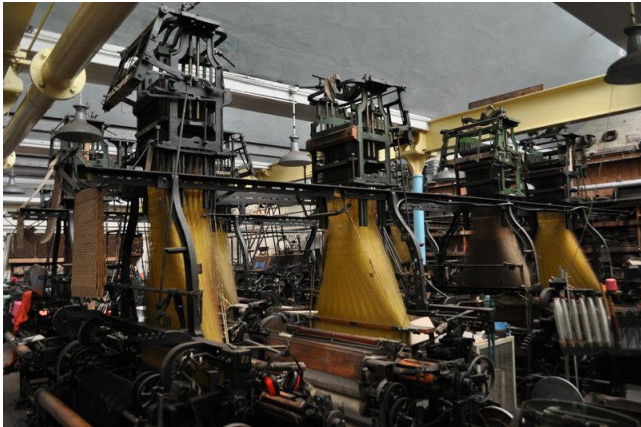
Autômatos de Jaquet-Droz
– **A Escritora:** memória de
quarenta caracteres
(~1700)



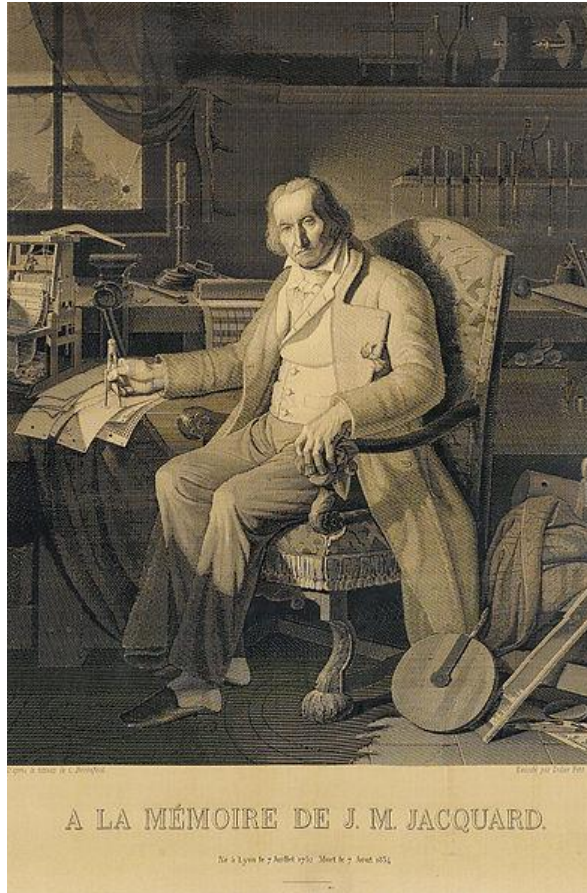
Computador de marés:
previsão da altura das
marés (~1800, Lord Kelvin)

Fundamentos de Computação

- Tear de Jacquard: utilizava cartões perfurados para programação dos padrões (1839).



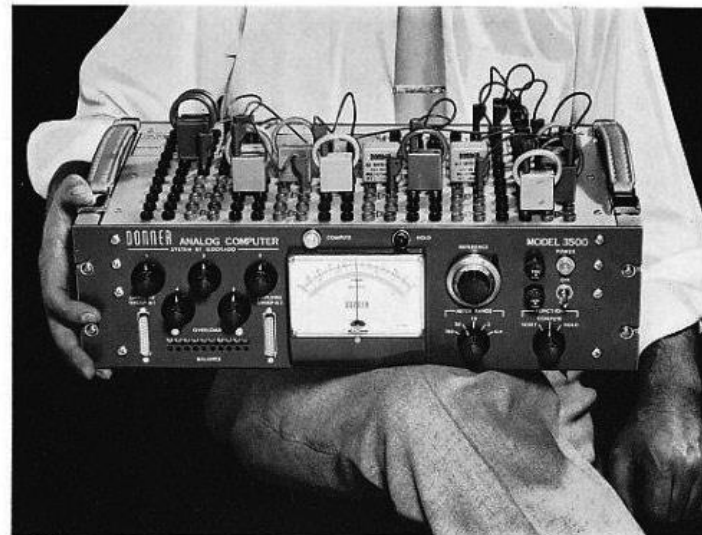
O retrato de Jacquard foi tecido em seda usando uma de suas máquinas, necessitando de 24.000 cartões perfurados.



Fundamentos de Computação

- **Computador analógico (eletrônico):**

No lugar de eixos mecânicos estes computadores empregavam conexões elétricas (~1900).



This is the one computer you can take with you
Announcing the Donner 3500
SMALLEST ANALOG COMPUTER EVER MADE

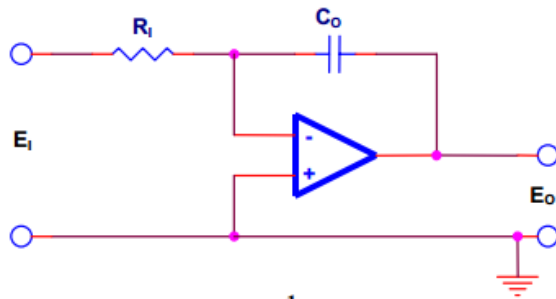


Fundamentos de Computação

- Computador analógico (eletrônico):

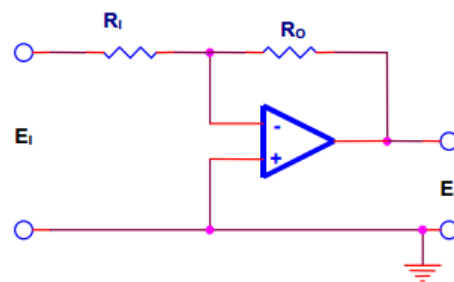
Utilizando amplificadores operacionais.

Integrator



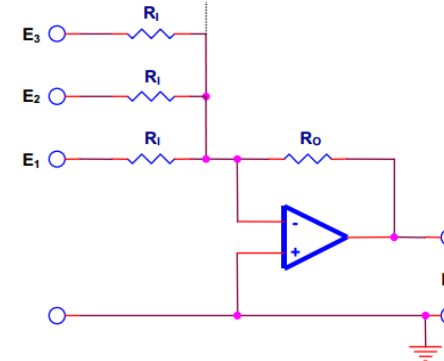
$$E_o = \frac{-1}{R_i C_o} \int E_i dt$$

INVERTING AMPLIFIER



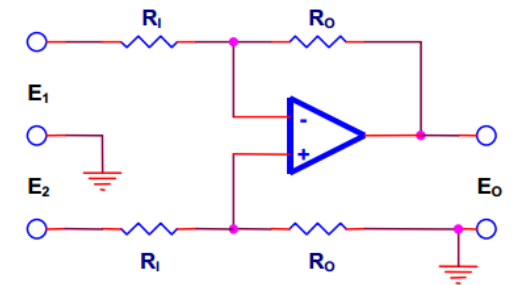
$$\frac{E_o}{E_i} = -\frac{R_o}{R_i}$$

Voltage Adder



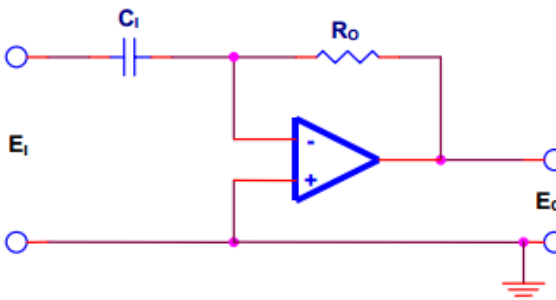
$$E_o = -\frac{R_o}{R_i} (E_1 + E_2 + E_3 + \dots)$$

Differential Input Amplifier



$$E_o = \frac{R_o}{R_i} (E_2 - E_1)$$

Differentiator

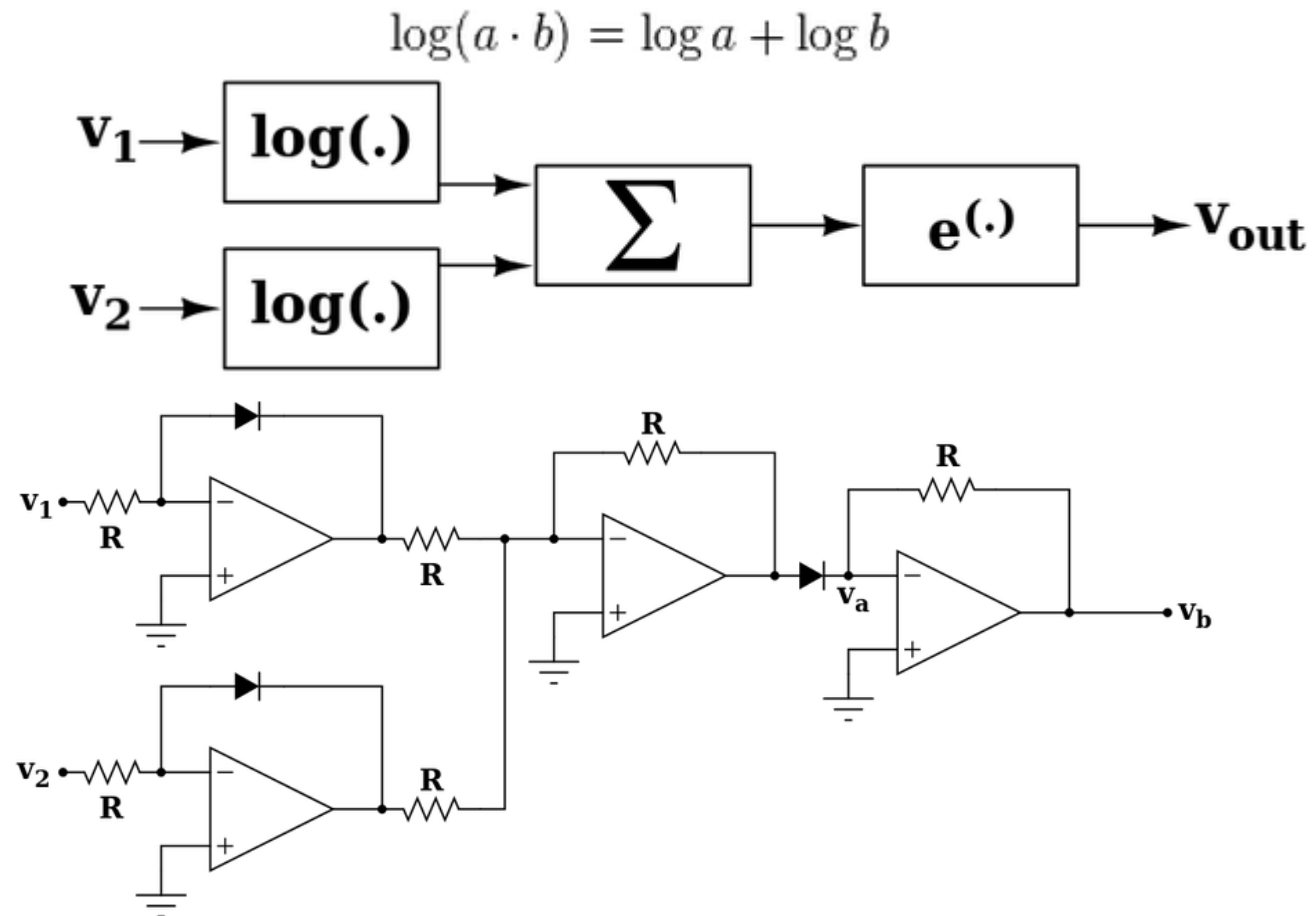


$$E_o = -R_o C_i \frac{dE_i}{dt}$$

Fundamentos de Computação

- Computador analógico (eletrônico):

Multiplicação usando a cabeça...

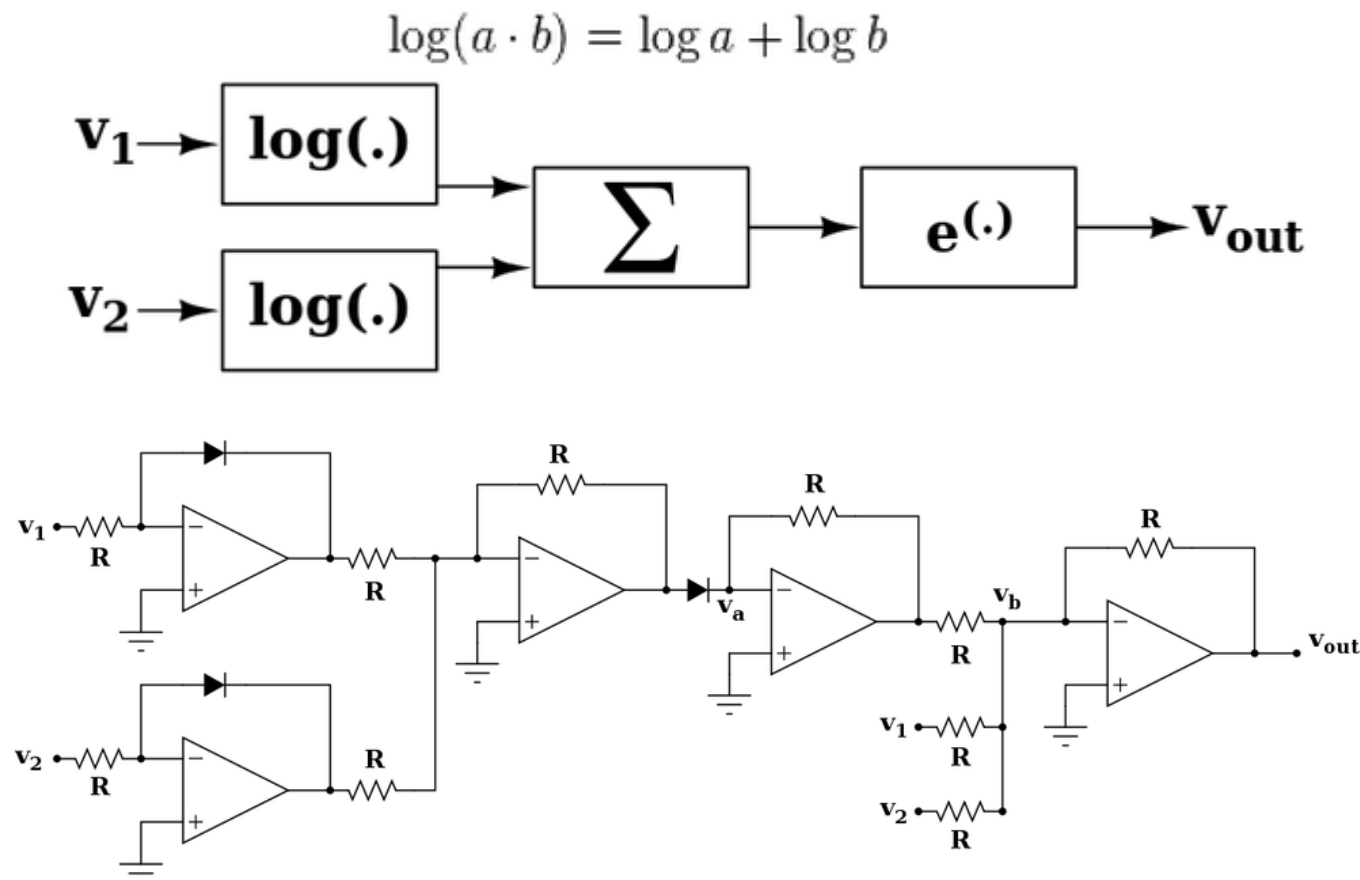


$$-\frac{v_1 \cdot v_2}{RI_s} - (v_1 + v_2)$$

Fundamentos de Computação

- Computador analógico (eletrônico):

Multiplicação usando a cabeça... e adicionando um termo.



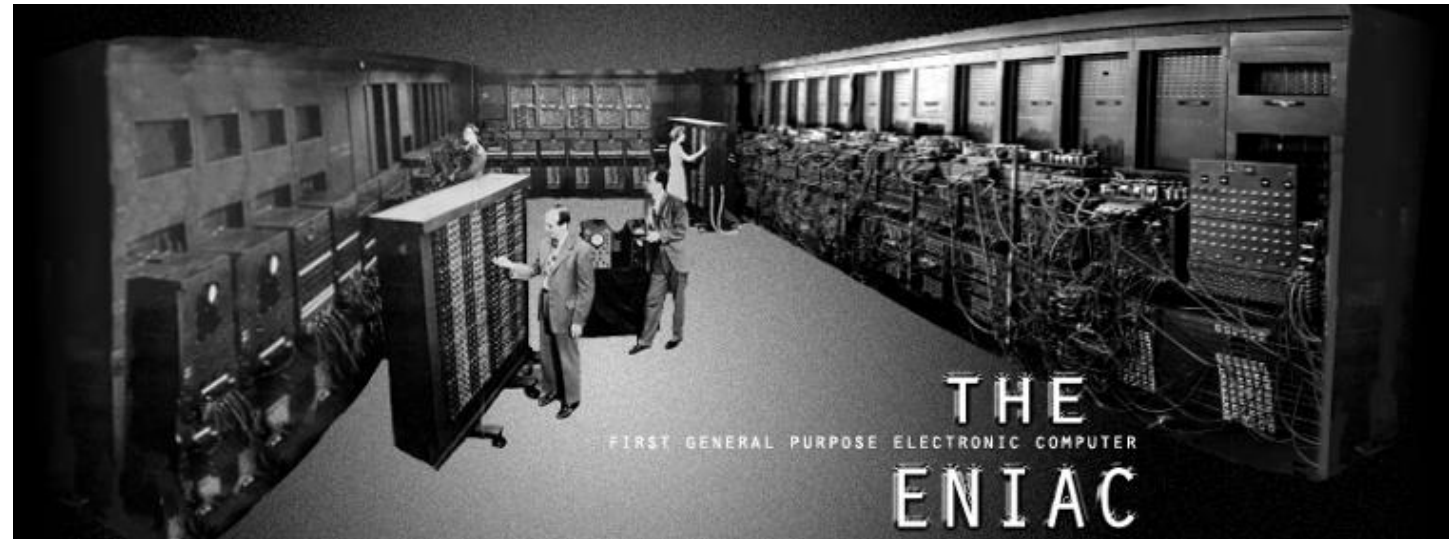
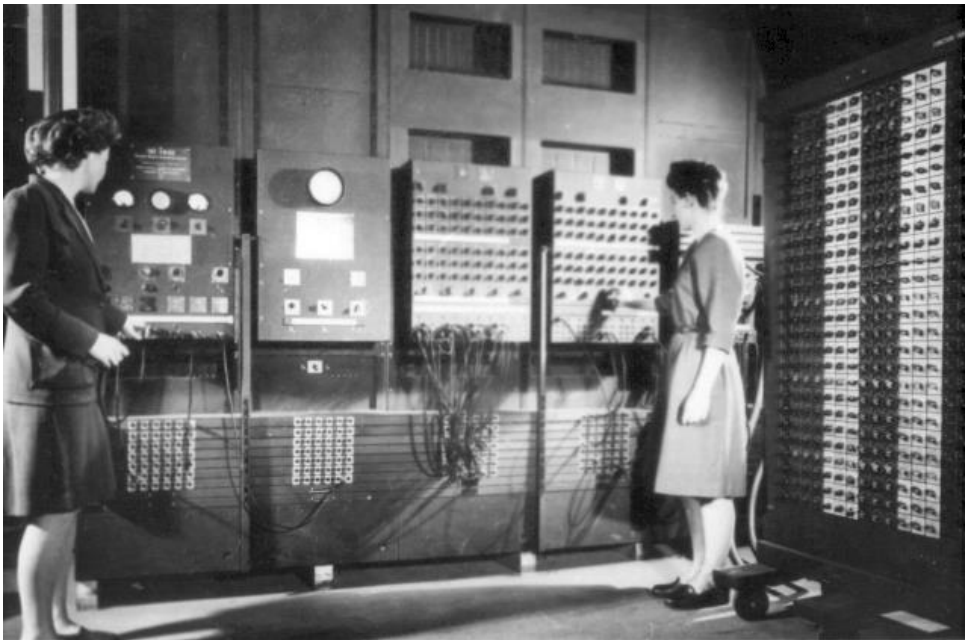
Como fazer divisão?
Considere o logaritmo!

$$\frac{v_1 \cdot v_2}{RI_s}$$

Fundamentos de Computação

- **Computador digital:**

Circuitos substituídos por válvulas...



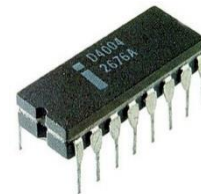
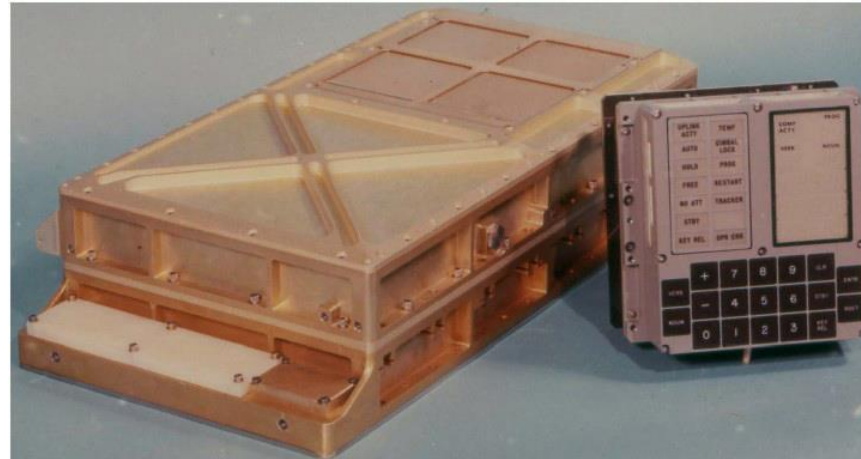
ENIAC - Electronic Numerical Integrator And Computer: Computador para cálculo de trajetórias balísticas. (~1945)

Fundamentos de Computação

- **Computador digital:**

Circuitos substituídos por válvulas... e posteriormente transistores.

IBM 608 – Calculadora Transistorada (1955)



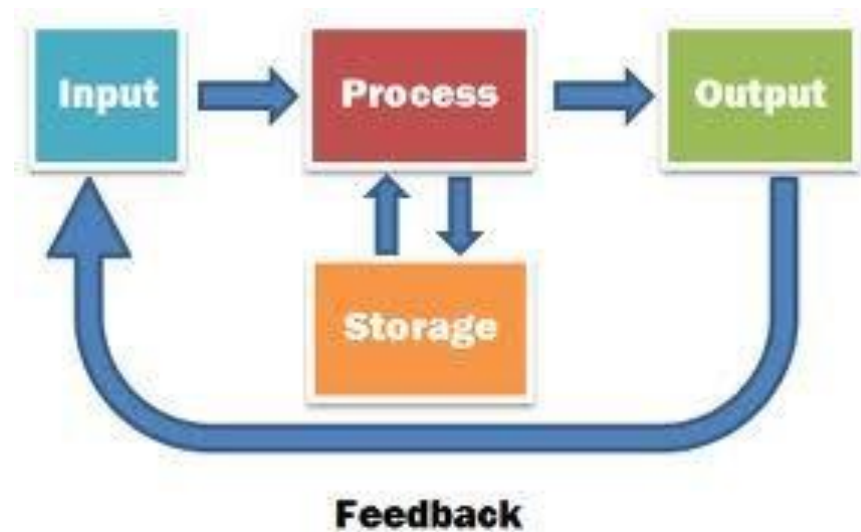
Intel 4004 (1971): primeiro processador, após o chute inicial dos programas Apollo e Minuteman na indústria de circuitos integrados.



Fundamentos de Computação

- **Computador digital (moderno):**

Ciclo IPOS (*Input, Processing, Output, Storage*)

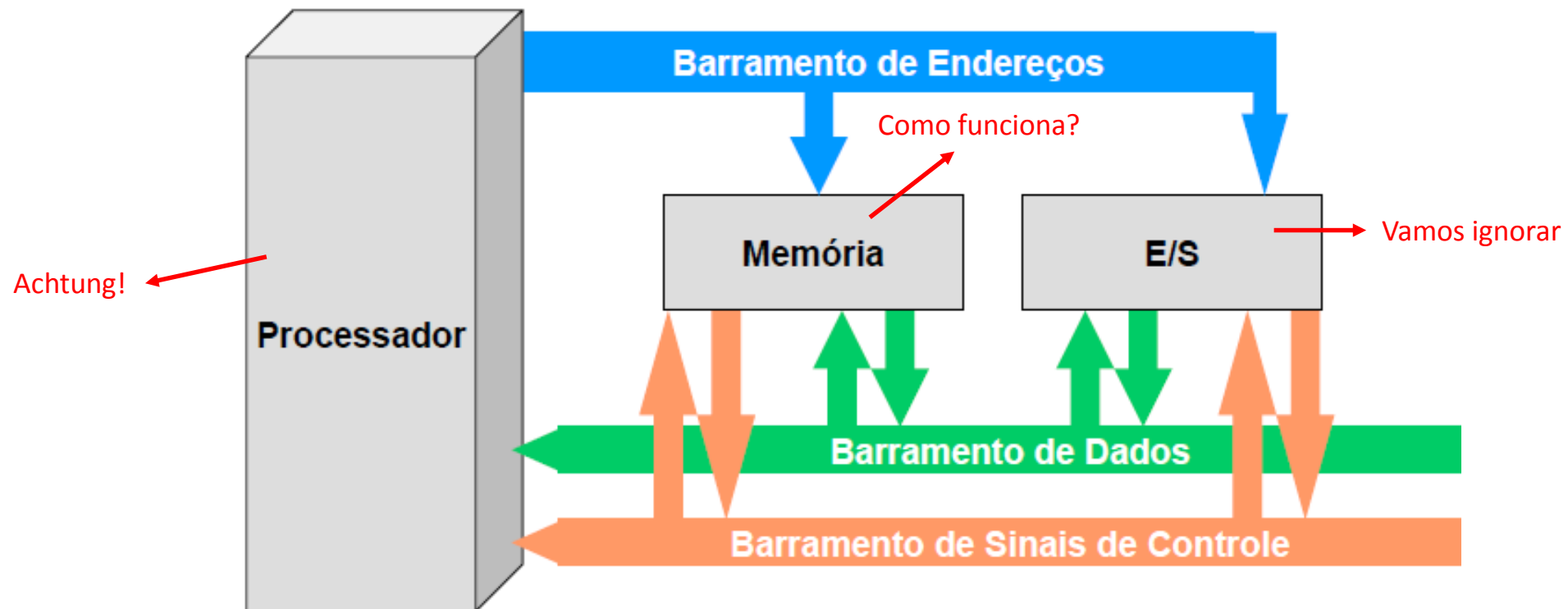


Sistema de Computação: integração de componentes numa única entidade. *peopleware + hardware + software*

componentes: humana + física + lógica

Fundamentos de Computação

- Visão geral da arquitetura de computadores:



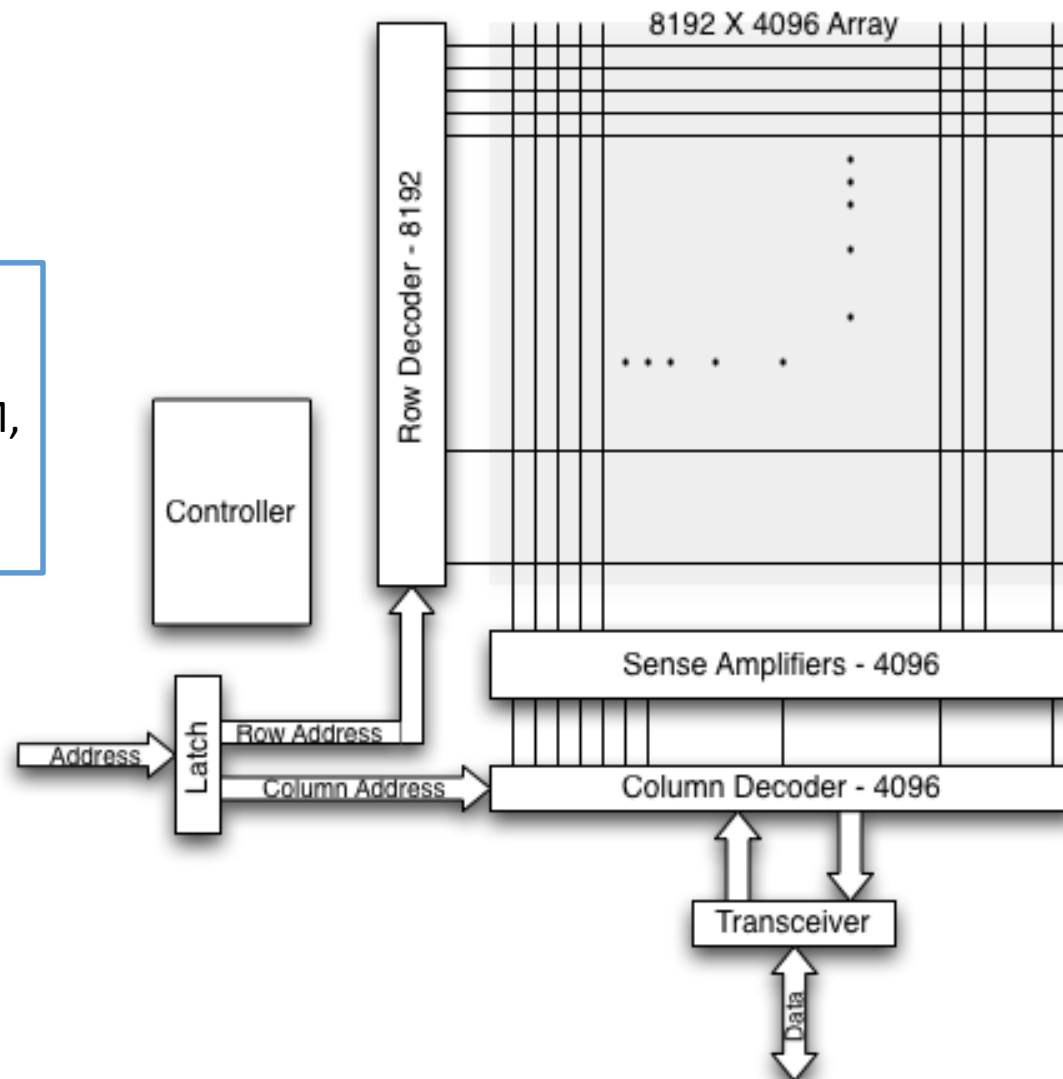
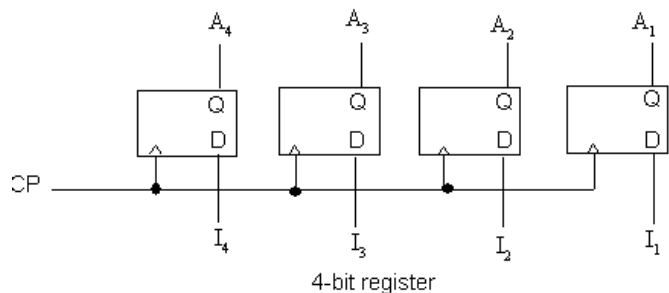
Fundamentos de Computação

- Visão geral da memória:

Matriz de memória

Cada intersecção liga uma célula de memória, há diferentes compromissos entre diferentes tipos de células, SRAM, DRAM, ROM, etc (densidade, velocidade e volatilidade)

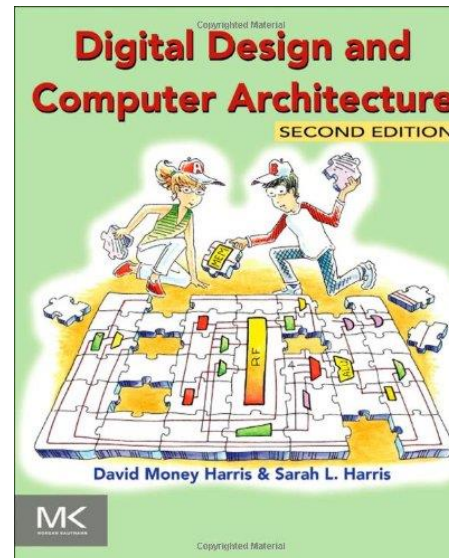
Registrador – o primo isolado



Fundamentos de Computação

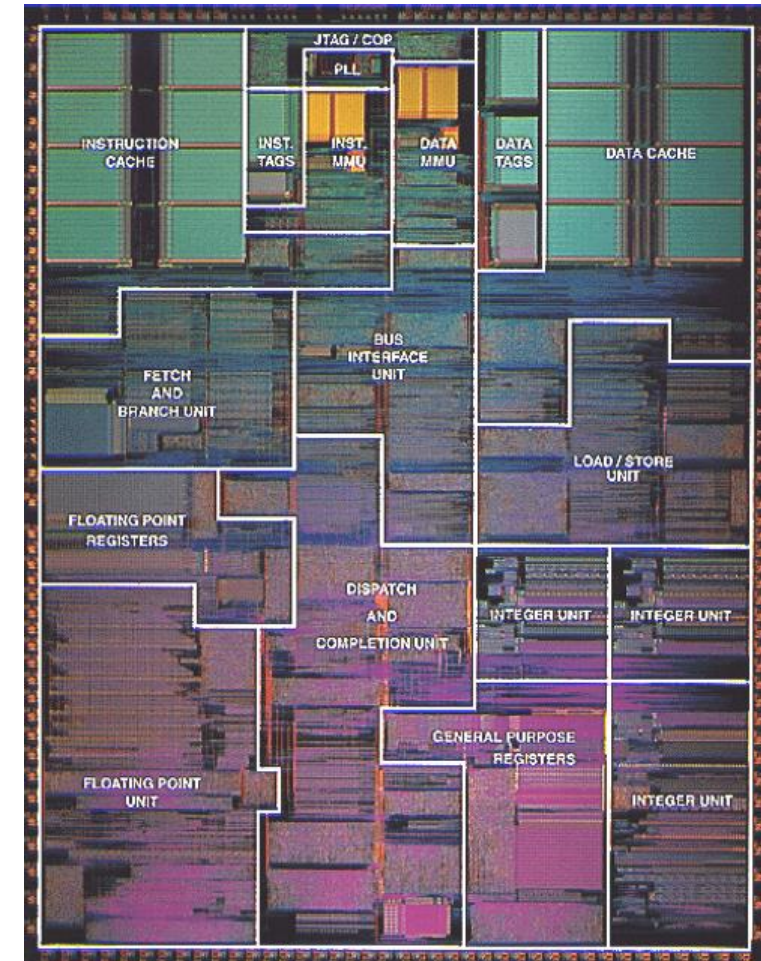
- Visão geral da memória:
 - “*Location, location, location*” – posição da memória referente ao processador importa.
 - Registradores (D flip flops) e Caches (SRAM) são localizados próximos.
 - A RAM (DRAM) se encontra distante.

Memory Type	Transistors per Bit Cell	Latency
flip-flop	~20	fast
SRAM	6	medium
DRAM	1	slow



p. 267

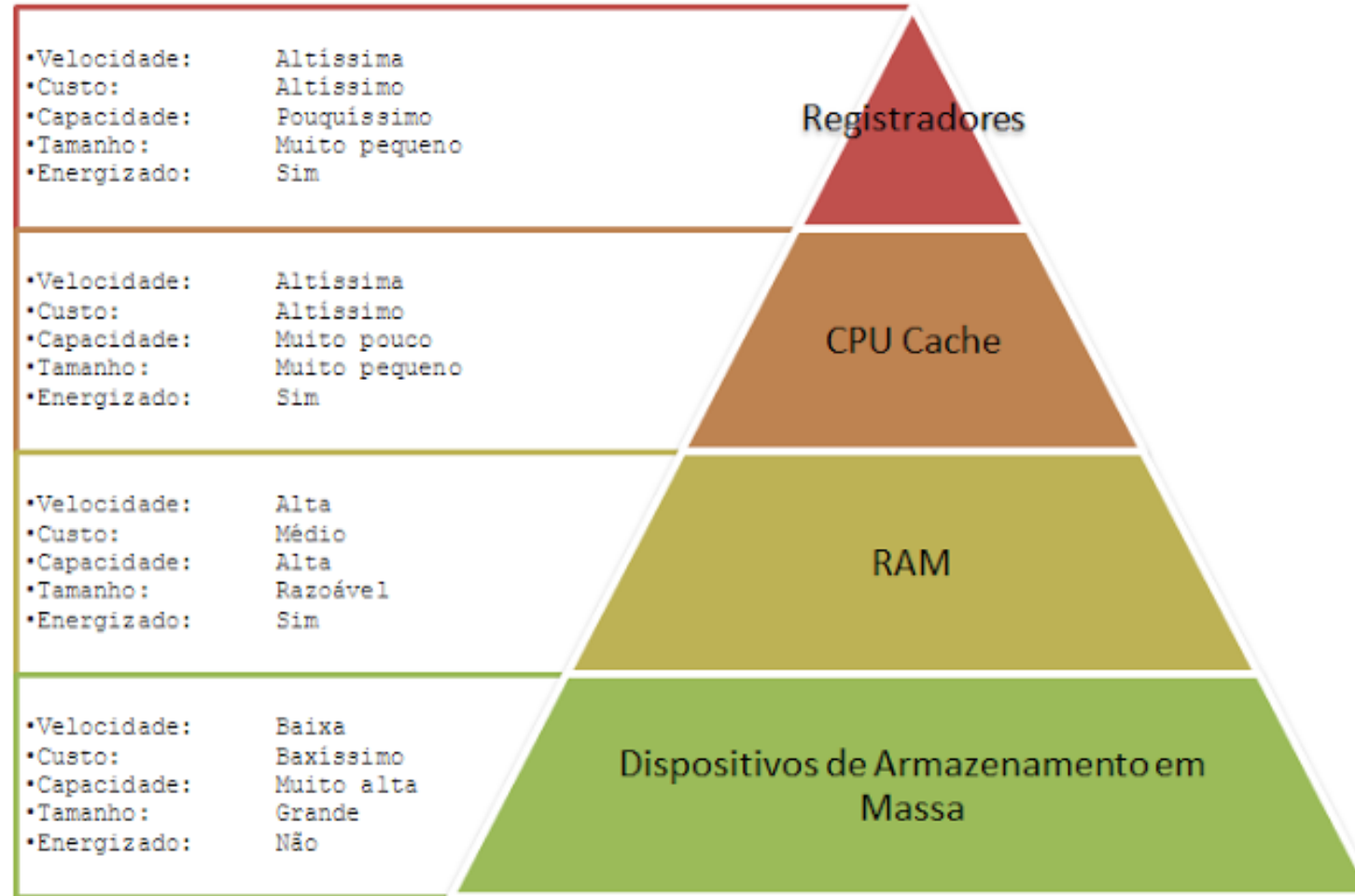
Motorola PowerPC 604



14/36

Fundamentos de Computação

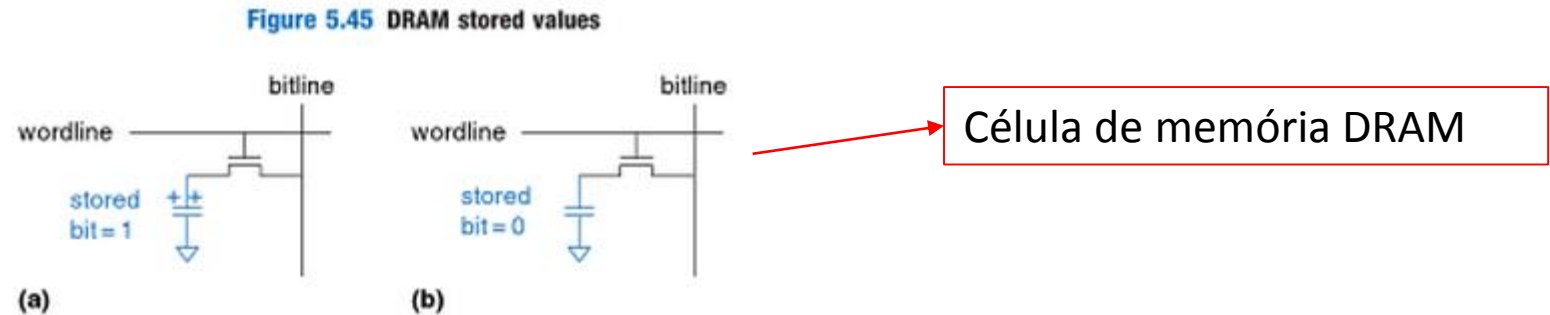
- **Visão geral da memória: hierarquia de memória.**



Fundamentos de Computação

- **Visão geral da memória:**

O bit é a unidade elementar de armazenamento de dados. Valor 0 ou 1, não existe bit vazio.



Um conjunto de oito bits é chamado de byte, possui 256 (2^8) valores possíveis.

Palavra, ou *word*, é a unidade natural de bits processados em uma arquitetura .

Fundamentos de Computação

- **Representações numéricas:** Um valor pode ser representado em diferentes “alfabetos”.
 - Hexadecimal possui 16 caracteres: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
 - Decimal possui 10 caracteres: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
 - Octal possui 8 caracteres: 0, 1, 2, 3, 4, 5, 6, 7.
 - Binário possui 2 caracteres: 0, 1.
- Hexadecimal e octal possuem uma relação semi-direta com binário e são amplamente utilizados para **melhor visualização de valores**.
- Notação: número_{base}

Fundamentos de Computação

- **Representações numéricas:** Decimal para binário.
 - Método do teste de resto na divisão por 2

Procure pela operação **shift right**, consegue perceber a similaridade com o que acabamos de fazer?

$$\begin{array}{r}
 2)254 - 0 \\
 2)127 - 1 \\
 2)63 - 1 \\
 2)31 - 1 \\
 2)15 - 1 \\
 2)7 - 1 \\
 2)3 - 1 \\
 2)1 - 1
 \end{array}
 \begin{array}{l}
 \uparrow \\
 \\
 \\
 \\
 \\
 \\
 \\
 \\
 \uparrow
 \end{array}
 \begin{array}{l}
 \text{LSB (Least Significant Bit)} \\
 \\
 \\
 = 11111110 \\
 \\
 \\
 \\
 \text{MSB (Most Significant Bit)}
 \end{array}$$

Fundamentos de Computação

- **Representações numéricas:** Binário para decimal.

- Método da notação posicional

- $$\begin{array}{cccccccc} 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}$$

$$= 1*2^7+1*2^6+1*2^5+1*2^4+1*2^3+1*2^2+1*2^1+0*2^0 = \mathbf{254}$$

Agora procure pela operação **shift left**, consegue perceber a similaridade com o que acabamos de fazer?

Fundamentos de Computação

- **Representações numéricas:** Octal e hexadecimal para decimal e vice-versa.
- Valores tabelados, são equivalências alfabéticas

Agora procure pela operação **shift left**, consegue perceber a similaridade com o que acabamos de fazer?

Decimal	Hexadecimal	Octal
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	-
9	9	-
10	A	-
11	B	-
12	C	-
13	D	-
14	E	-
15	F	-

Fundamentos de Computação

- **Representações numéricas:** Octal e hexadecimal para binário e vice-versa.
 - O valor máximo de um dígito octal é 7, equivale a 111 em binário, portanto há uma equivalência de 3 bits para cada dígito octal.
 - Similarmente para um dígito hexadecimal, cujo máximo é F (15 dec.), equivale a 1111 em binário, equivalência de 4 bits por dígito.

Agora procure pela operação **shift left**, consegue perceber a similaridade com o que acabamos de fazer?

Binário	Decimal	Hexadecimal	Octal
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	8	8	-
1001	9	9	-
1010	10	A	-
1011	11	B	-
1100	12	C	-
1101	13	D	-
1110	14	E	-
1111	15	F	-

Fundamentos de Computação

- **Esquemas de codificação:**
 - No computador, letras são números. O significado de cada número é resultado de um padrão ou convenção.
 - Esquemas comuns: ASCII, EBCDIC, Unicode
 - ASCII: Representa caracteres de 8 bits.
 - Unicode: Acomoda alfabetos com mais de 256 caracteres, utiliza 16 bits para representação. 65.536 valores possíveis.

Fundamentos de Computação

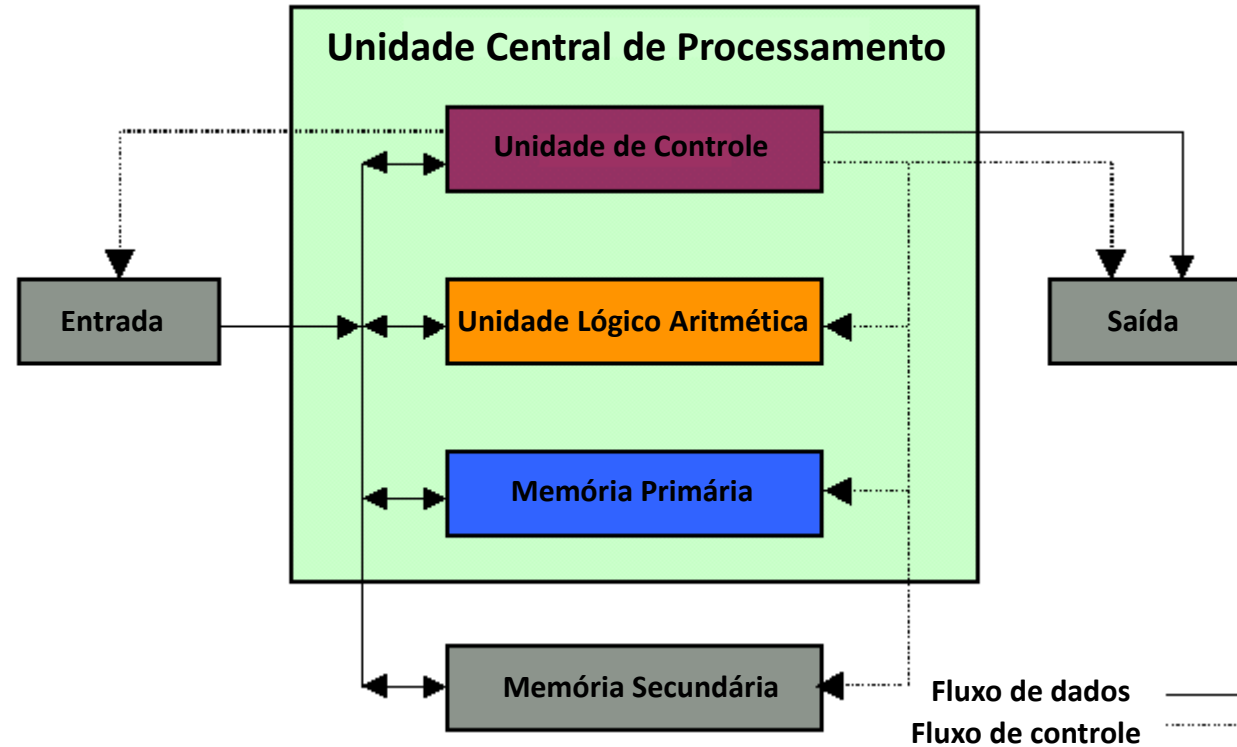
- Esquemas de codificação: ASCII.
- American Standard Code for Information Interchange

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	:	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Quando estiver programando, faça um programa que escreva todos os caracteres de 0 a 255 no console. Há outros caracteres que não estão na tabela ASCII? Por que?

Fundamentos de Computação

- Visão geral da arquitetura de processadores:



Fundamentos de Computação

- **Unidade de Controle:**

Responsável por manter o ciclo

Fetch – Decode – Execute – Store

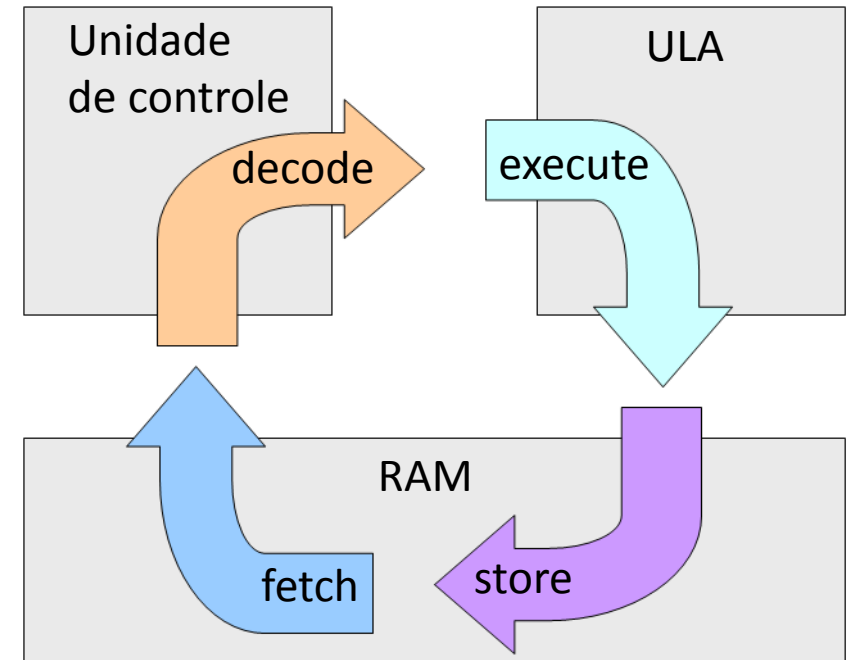
Fetch: busca a instrução endereçada pelo PC

Decode: determina qual o estado da instrução

Execute: efetua a operação da instrução

Store: armazena dados na memória

Lógica sequencial:
hardwired ou microcode



Debugger: PEb.exe [CPU: main thread, module: PEb.exe]

Registers (FPU):

```

EAX 00241EBC
ECX 7C91056D ntdll.7C91056D
EDX 00140608
EBX 00400000 PEB.00400000
ESP 0012FFAC
EBP 0012FFB8
ESI 7FFD4000
EDI 0040316D PEB.0040316D
EIP 00401231 PEB.00401231
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
S 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
O 0 FS 003B 32bit 7FFD000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_ALREADY_EXISTS (000000B7)
EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty 6.4522409303522723840e-4932
ST1 empty -UNORM EAD0 0012FA38 00000000
ST2 empty 6.6973533377667665920e+1824
ST3 empty +UNORM 0020 0000003B F5A559F0
ST4 empty +UNORM 003B 0012F9C0 00000000
ST5 empty -UNORM F9D4 00000202 0000001B
ST6 empty 1.000000000000000000000000
ST7 empty 1.000000000000000000000000
          3 2 1 0   E S P U O Z D I
FST 4000 Cond 1 0 0 0 Err 0 0 0 0 0 0 (EQ)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
    
```

Disassembly:

```

00401231 55      PUSH   EBP
00401232 8BEC   MOV    EBP,ESP
00401233 83C4   ADD    ESP,-8
00401234 68     PUSHAD
00401237 8B75   MOV    ESI,DI
00401238 803D 70334000 LEA   EDI,DWORD PTR DS:[403370]
00401241 XOR    EAX,EAX
00401243 8A06   MOV    AL,BYTE PTR DS:[ESI]
00401245 8B07   MOV    BYTE PTR DS:[EDI],AL
00401247 47     INC    EDI
00401248 83C6   ADD    ESI,2
00401249 8B75   MOV    ESI,DI
0040124E 75 F3  JNZ   SHORT PEB.00401245
00401250 66:C707 0D0A MOV   WORD PTR DS:[EDI],0A0D
00401255 83C7   ADD    EDI,2
00401258 66:C707 0D0A MOV   WORD PTR DS:[EDI],0A0D
0040125D 33C0   XOR    EAX,EAX
0040125F 803D 70334000 LEA   EDI,DWORD PTR DS:[403370]
00401265 B9 FFFFFFFF MOV   ECX,FFFFFFFF
0040126A F2:AE  REPNE SCAS BYTE PTR ES:[EDI]
0040126C NOT    ECX
0040126E 49     DEC    ECX
0040126F 894D FC MOV   DWORD PTR SS:[EBP-4],ECX
00401272 6A 02  PUSH  2
00401274 6A 00  PUSH  0
00401276 6A 00  PUSH  0
00401278 FF35 6D324000 PUSH DWORD PTR DS:[40326D]
0040127E E8 4B000000 CALL  <JMP.<kernel32.SetFilePointer>
00401283 6A 00  PUSH  0
00401285 8045 F8 LEA   EAX,DWORD PTR SS:[EBP-8]
00401288 50     PUSH  EAX
00401289 50     PUSH  EAX
0040128C 68 70334000 PUSH DWORD PTR SS:[EBP-4]
00401291 FF35 6D324000 PUSH DWORD PTR DS:[40326D]
00401297 E8 3B000000 CALL  <JMP.<kernel32.WriteFile>
0040129C 68 FF000000 PUSH  0FF
004012A1 68 70334000 PUSH  PEB.00403370
004012A6 E8 1D000000 CALL  <JMP.<kernel32.RtlZeroMemory>
004012AB 61     POPAD
004012AC 49     LEAVE
004012AD C2 0400 RETN  4
004012B0 5A     JMP   DWORD PTR DS:[<<kernel32.CloseHandle>]
004012B6 5A     JMP   DWORD PTR DS:[<<kernel32.CreateFileA>]
004012BC 5A     JMP   DWORD PTR DS:[<<kernel32.ExitProcess>]
004012C2 5A     JMP   DWORD PTR DS:[<<kernel32.GetCommandLineA>]
004012C8 5A     JMP   DWORD PTR DS:[<<kernel32.RtlZeroMemory>]
004012CE 5A     JMP   DWORD PTR DS:[<<kernel32.SetFilePointer>]
004012D4 5A     JMP   DWORD PTR DS:[<<kernel32.WriteFile>]
004012DA 5A     JMP   DWORD PTR DS:[<<kernel32.lstrcatA>]
004012E0 5A     JMP   DWORD PTR DS:[<<kernel32.lstrcpyA>]
004012E6 5A     JMP   DWORD PTR DS:[<<user32.wsprintfA>]
004012EC 5A     JMP   DWORD PTR DS:[<<user32.MessageBoxA>]
004012F2 5A     JMP   DWORD PTR DS:[<<shell32.ShellExecuteA>]
004012F8 00     DB    00
004012FA 00     DB    00
004012FB 00     DB    00
004012FC 00     DB    00
004012FD 00     DB    00
004012FE 00     DB    00
004012FF 00     DB    00
00401300 00     DB    00
00401301 00     DB    00
00401302 00     DB    00
00401303 00     DB    00
    
```

Comments:

```

Origin = FILE_END
pOffsetHi = NULL
OffsetLo = 0
hFile = 00000080 (window)
SetFilePointer
pOverlapped = NULL
pBytesWritten
nBytesToWrite
Buffer = PEB.00403370
hFile = 00000080 (window)
WriteFile
Length = FF (255.)
Destination = PEB.00403370
kernel32.CloseHandle
kernel32.CreateFileA
kernel32.ExitProcess
kernel32.GetCommandLineA
ntdll.RtlZeroMemory
kernel32.SetFilePointer
kernel32.WriteFile
kernel32.lstrcatA
kernel32.lstrcpyA
user32.wsprintfA
user32.MessageBoxA
shell32.ShellExecuteA
    
```

Registers (FPU) (continued):

```

RETURN to PEB.00401136 from PEB.00401231
PEB.00400000
RETURN to PEB.<ModuleEntryPoint>+46 from PEB.00401067
RETURN to kernel32.7C816D4F
ntdll.7C910738
End of SEH chain
SE handler
kernel32.7C816D58
PEB.<ModuleEntryPoint>
    
```

Hex dump:

Address	Hex dump	ASCII
00403000	47 61 74 68 65 72 65 64 20 69 6E 66 6F 72 6D 61	Gathered informa
00403010	74 69 6F 6E 3A 20 4F 4E 00 00 0A 00 72 65 70	tion: OK...rep
00403020	6F 72 74 78 74 00 50 45 42 20 20 30 47 65	ort.txt.PEB - 5
00403030	74 20 61 6C 6C 20 44 6C 6C 27 73 00 6F 70 6E	t all Dll's open
00403040	00 50 45 42 20 61 64 64 72 65 73 73 20 61 74	.PEB address at
00403050	20 25 30 38 6C 58 00 4C 64 72 20 69 73 20 61	%08lx.Ldr is at
00403060	20 3A 20 25 30 38 6C 58 00 5F 4C 49 53 54 5F 45	: %08lx._LIST_E
00403070	4E 54 52 59 20 61 74 20 3A 20 25 30 38 6C 58 00	NTRY at: %08lx.
00403080	49 6D 61 67 65 42 61 73 65 20 3A 20 25 30 38 6C	ImageBase: %08lx
00403090	58 00 46 61 69 6E 6B 20 61 74 3A 20 25 30 38 6C	X.Flink at: %08lx
004030A0	58 00 42 6C 69 6E 6B 20 61 74 3A 20 25 30 38 6C	X.Blink at: %08lx
004030B0	58 00 44 6C 6C 20 62 61 73 65 20 3A 20 25 30 38	X.Dll base: %08lx
004030C0	6C 58 00 45 6E 74 72 79 50 6F 69 6E 74 20 3A 20	IX.EntryPoint:
004030D0	25 30 38 6C 58 00 53 69 7A 65 4F 66 49 6D 61 67	%08lx.SizeOfImag
004030E0	65 20 3A 20 25 30 38 6C 58 00 50 72 6F 63 65 73	e: %08lx.Process
004030F0	73 20 64 65 62 75 67 67 65 64 00 50 72 6F 63 65	s debugged.Proce
00403100	73 73 20 64 65 62 75 67 67 65 64 00 50 72 6F 63	ss not debugged.
00403110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Command: []

Breakpoint at PEB.00401231

Paused

Observar:

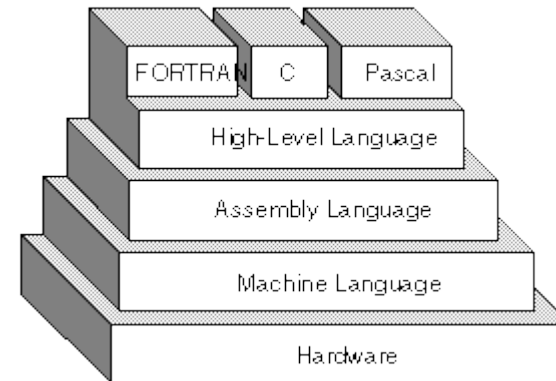
- Instruction Pointer
- Ciclo fetch-decode-execute-store
- Pilha (stack)
- Heap
- Registradores
- Registradores EBP e ESP
- Instruções

Linguagens de Alto Nível

- **Tipos de Linguagens de Programação:**

- Máquina: códigos numéricos relativos a instruções e operandos (*bit encoding*).

75 F3



- Assembly (montagem): representação simbólica do código de máquina, com suporte a labels, macros e estruturas básicas (*symbolic encoding*).

JNZ PEB.00401243

Linguagens de Alto Nível

- **Problemas do código de montagem:**
 - Difícil compreensão humana.
 - Por consequência, difícil manutenção e dificuldades de depuração.
 - Específico ao hardware.
 - Ausência de código estruturado leva ao código espaguete!




Linguagens de Alto Nível

- **Problemas do código de montagem:**
 - O código é fixo, uma vez definido não muda mais.
 - Controvérsia da localização da complexidade. No processador (CISC), no compilador (RISC), na máquina virtual?

Linguagens de Alto Nível

- **Tipos de Linguagens de Programação:**

- Alto nível: abstraem detalhes da implementação de hardware e introduzem conceitos de código estruturado.

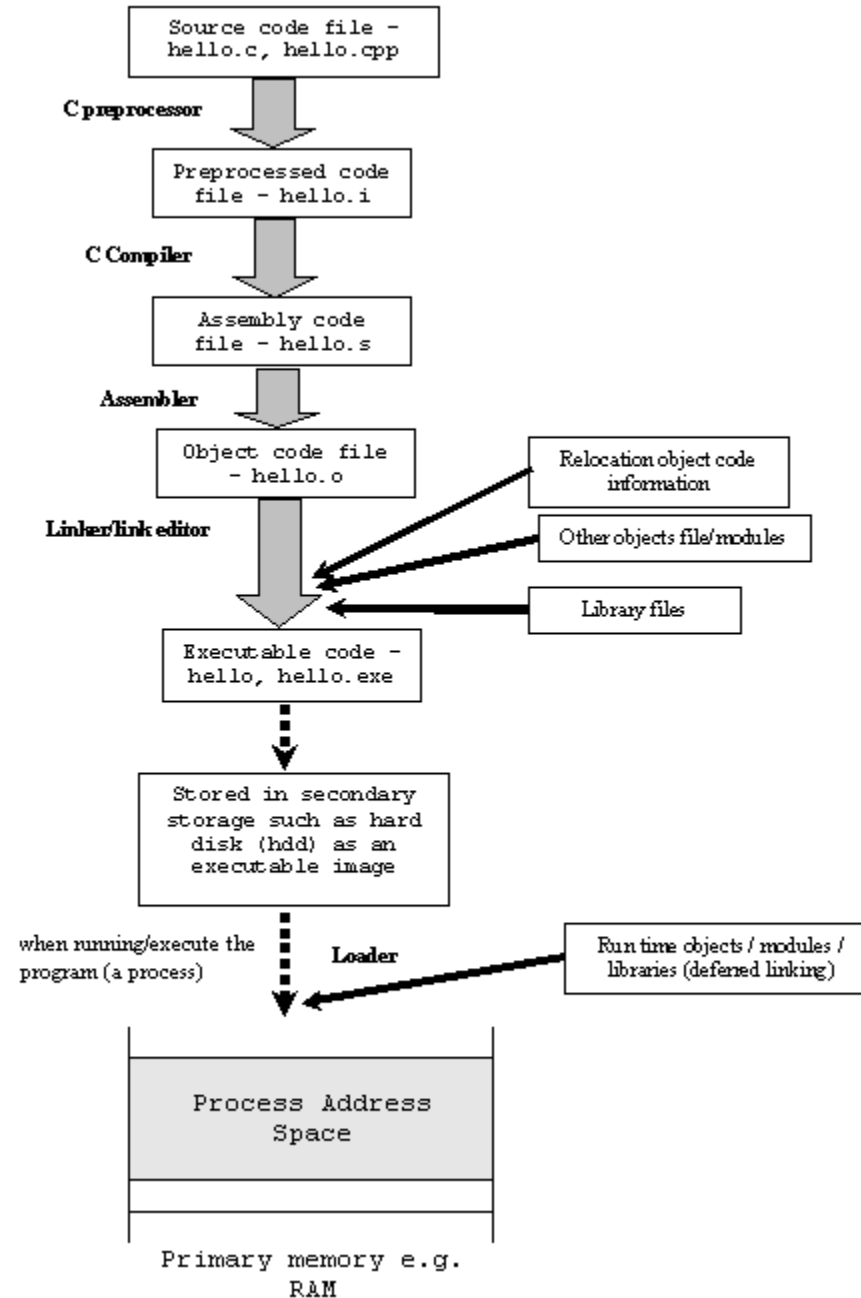
O “nível” é relativo. C já foi uma linguagem de alto nível. 
Atualmente é considerada apenas um nível acima de abstração de assembly, alguns consideram C uma linguagem de médio nível. (tópico controverso)

Compilador/Interpretador

- **Dois paradigmas distintos:**
 - Compiladores processam o código fonte e geram um executável. C é uma linguagem compilada.
 - Interpretadores processam o código fonte durante o tempo de execução.

Compilador/Interpretador

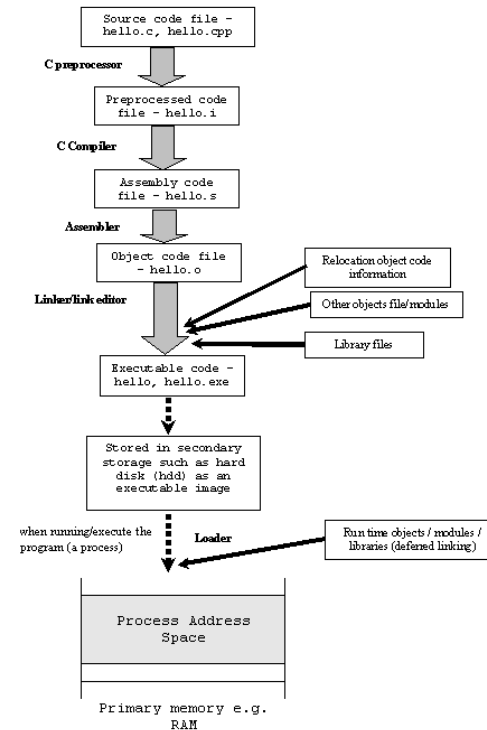
- **Compilador:**



Compilador/Interpretador

Etapas da compilação:

- **Pré-processador:** altera os arquivos fontes de acordo com diretivas de pré-processamento.
- **Compilador:** traduz o código para assembly, considerando as análises léxica, sintática e semântica. Efetua otimizações.
- **Assembler:** traduz o código assembly para código de máquina, mantendo alguns parâmetros (labels para endereços, por exemplo).
- **Linker:** efetua a ligação de todos os objetos e bibliotecas em um código de máquina final.



História e Introdução a C

- **Origens:**

- Primeira versão do Unix: escrito em assembly por Ken Thompson e Dennis Ritchie



- Depois traduzido para B, uma implementação simplificada do BCPL por Thompson
- B era limitado: orientada a *words* e não poderia acessar bytes individuais ou valores de ponto flutuante, problemático em novos hardwares

História e Introdução a C

- **Evolução:**

- Mudanças no acesso a memória e funcionamento de ponteiros

- Alterações sintáticas

- C!

Algol	• International Group
BCPL	• Martin Richards
B	• Ken Thomson
Traditional C	• Dennis Ritchie
K&R C	• kernighan & Ritchie
ANSI C	• ANSI Commitee
ANSI/ISO C	• ISO Commitee
C99	• Standerd Commitee

The Development of the C Language*

*Dennis M. Ritchie
Bell Labs/Lucent Technologies
Murray Hill, NJ 07974 USA*

Leitura!

História e Introdução a C

- Genealogia:

Genealogy of programming languages

