

Introdução à Arquitetura e Linguagem Assembly de Processadores IA-32

Sistemas da Computação

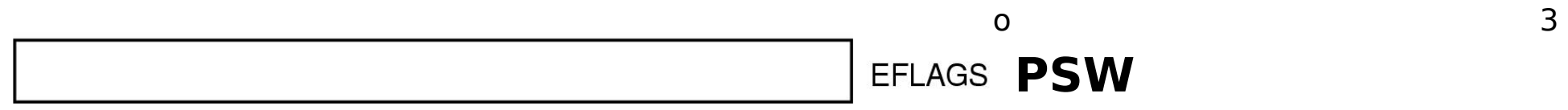
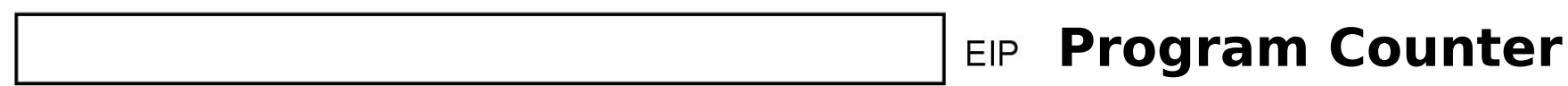
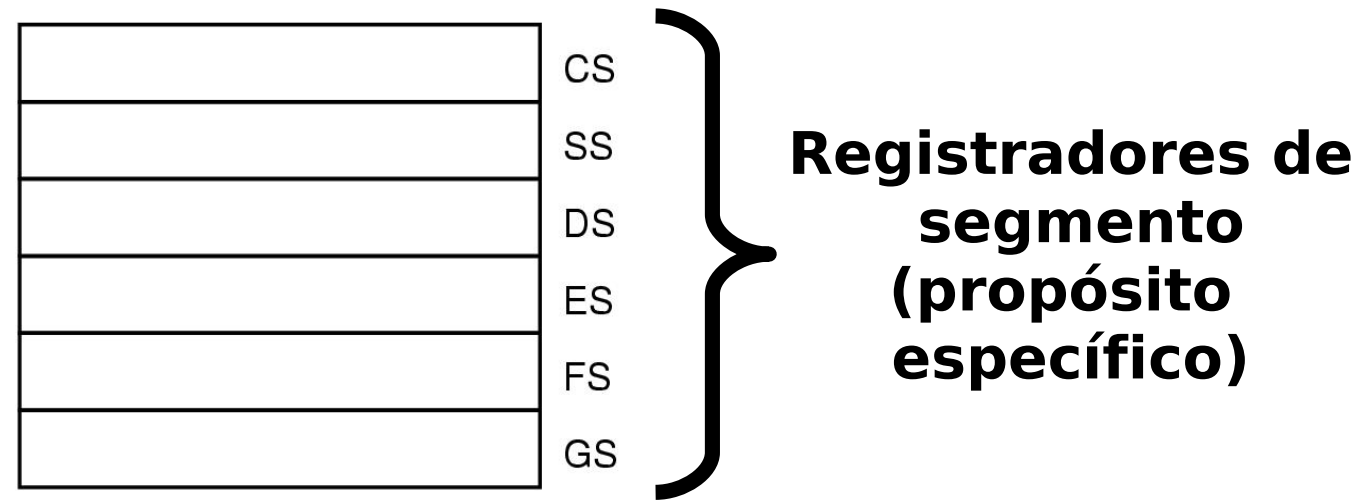
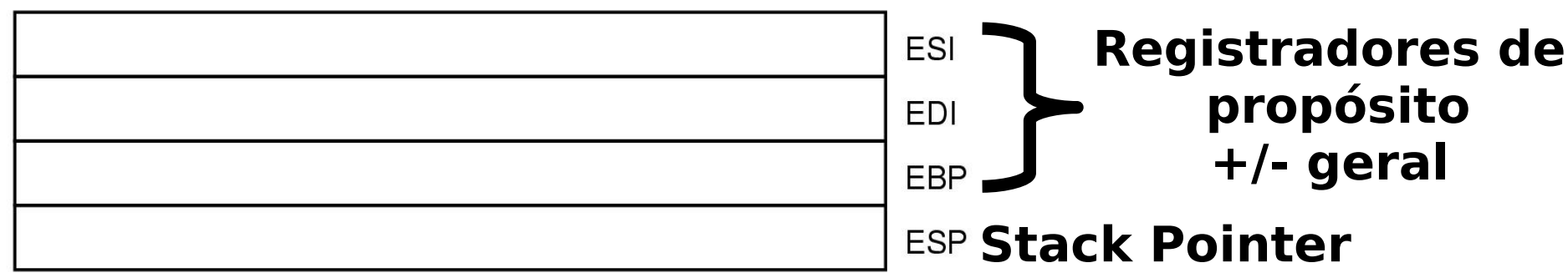
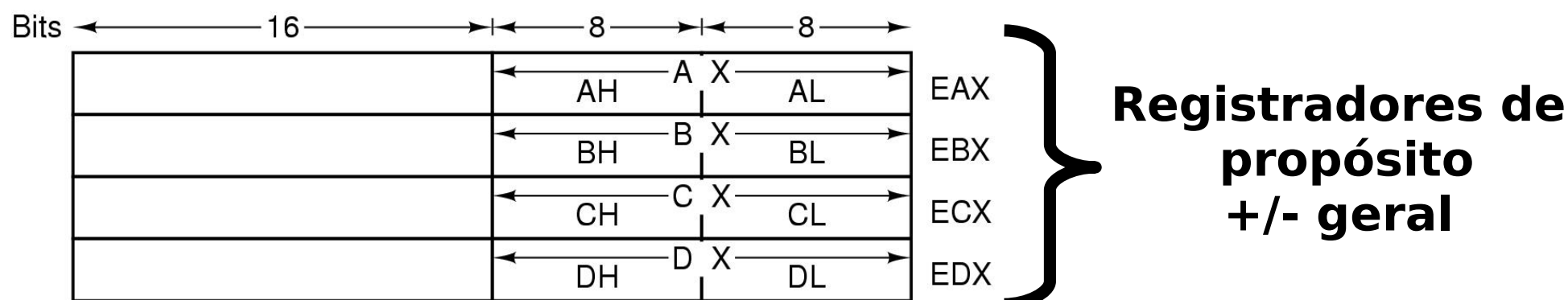
Prof. Rossano Pablo Pinto, Msc.

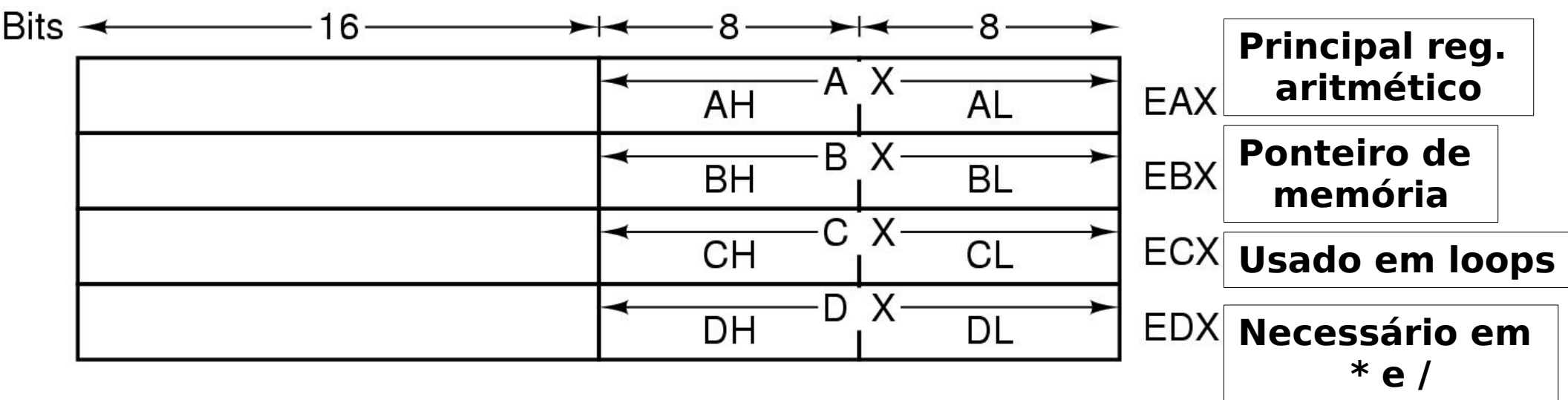
rossano at gmail com

2 semestre 2007

Introdução à Arquitetura IA-32

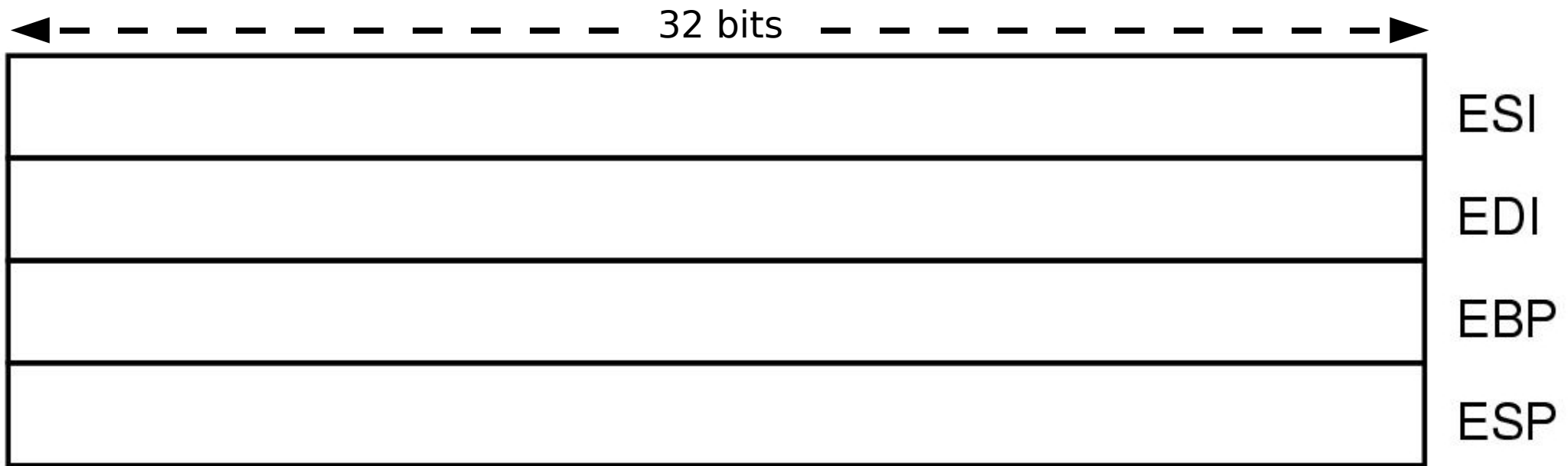
CHIP	Ano	MHz	Transistors	Mem
8086	1978	5-10	29.000	1 MB
8088	1979	5-8	29.000	16 MB
80286	1982	8-12	134.000	16 MB
80386	1985	16-33	275.000	4 GB
80486	1989	25-100	1.2 M	4 GB
Pentium	1993	60-233	3.1 M	4 GB
Pentium Pro	1995	150-200	5.5 M	4 GB
Pentium II	1997	233-400	7.5 M	4 GB
Pentium III				
Pentium 4				
Core 2 Duo				





- **EAX** em conjunto c/ **EDX** armazenam produtos e dividendos de 64 bits
- **E?X = ?X + Ext**
- **?X = ?L + ?H**: 16 bits
- **?L**: A Low (low order): 8 bits
- **?H**: A High (high order): 8 bits
- **? = A | B | C | D**
- **Nos 8088 e 80286 só existiam registradores de 8 e 16 bits**
- **Registradores de 32 bits foram inseridos no 80386 (foi adicionado o prefixo E - Extendido, como em EAX, EBX, ECX e EDX)**

Seção 5.1.5 do Tanenbaum



- **ESI** e **EDI**: usados p/ ponteiro de memória em instruções de manipulação de strings
- **ESI**: endereço da string de origem (source)
- **EDI**: endereço da string de destino (destiny)

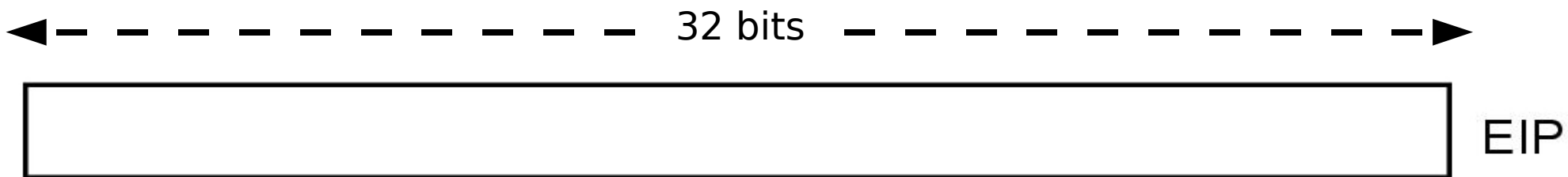
- **EBP**: geralmente utilizado como ponteiro para a base do frame da pilha corrente
- **ESP**: ponteiro para topo da pilha (registrador de propósito específico)

← - - - 16 bits - - - →



- CS Segmento de código**
- SS Segmento da pilha do programa**
- DS Segmento de dados**
- ES Ponteiro extra de segmento**
- FS Ponteiro extra de segmento**
- GS Ponteiro extra de segmento**

- Registradores de segmento
- Utilizados inicialmente no 8088 p/ endereçar 2^{20} bytes de memória com endereços de 16 bits
- Atualmente, quando o processador é configurado p/ operar endereços lineares de 32 bits, estes registradores podem ser ignorados



- **EIP (Extended Instruction Pointer) - Program Counter - PC**



- **EFLAGS (Extended Flags) - Program Status Word - PSW**

Intel x86 FLAGS Register			
Bit #	Abbreviation	Description	Category*
FLAGS			
0	CF	Carry flag	S
1		1 Reserved	
2	PF	Parity flag	S
3		0 Reserved	
4	AF	Auxiliary flag	S
5		0 Reserved	
6	ZF	Zero flag	S
7	SF	Sign flag	S
8	TF	Trap flag (single step)	X
9	IF	Interrupt enable flag	X
10	DF	Direction flag	C
11	OF	Overflow flag	S
12,13	IOPL	I/O privilege level (286+ only)	X
14	NT	Nested task flag (286+ only)	X
15		0 Reserved	
EFLAGS			
16	RF	Resume flag (386+ only)	X
17	VM	Virtual-8086 mode flag (386+ only)	X
18	AC	Alignment check (486SX+ only)	X
19	VIF	Virtual interrupt flag (Pentium+)	X
20	VIP	Virtual interrupt pending (Pentium+)	X

Intel x86 FLAGS Register			
Bit #	Abbreviation	Description	Category*
FLAGS			
21	ID	Identification (Pentium+)	X
22		0 Reserved	
23		0 Reserved	
24		0 Reserved	
25		0 Reserved	
26		0 Reserved	
27		0 Reserved	
28		0 Reserved	
29		0 Reserved	
30		0 Reserved	
31		0 Reserved	
RFLAGS			
32-63		0 Reserved	

*Categories

S: Status flag

C: Control flag

X: System flag

Introdução à Arquitetura IA-32

- Três modos de operação
 - Modo Real (Real Mode)
 - Modo Virtual 8086 (Virtual 8086 Mode)
 - Modo Protegido (Protected Mode)

Introdução à Arquitetura IA-32

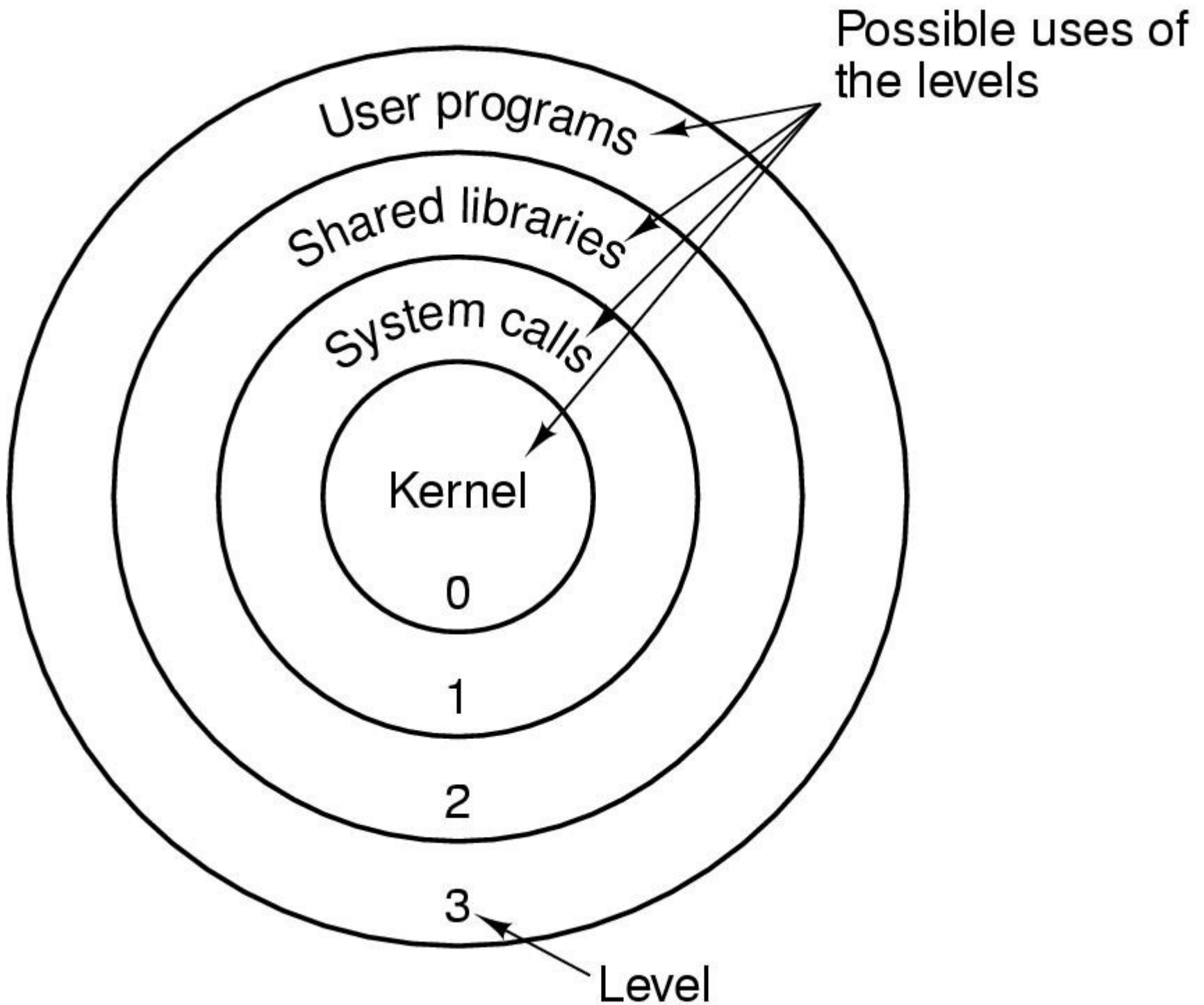
- Modo Real (Real Mode)
 - Desabilita todas as funções posteriores ao 8088
 - Não oferece proteções – se um programa falhar a máquina inteira falha (trava)

Introdução à Arquitetura IA-32

- Modo Virtual 8086 (Virtual 8086 Mode)
 - Roda programas feitos para o 8088 de forma protegida
 - SO controla máquina toda
 - SO cria um ambiente isolado que age com um 8088
 - Se o programa trava, o SO é notificado ao invés de travar a máquina toda
 - “Janela MS-DOS” roda neste modo p/ evitar programas DOS mal-comportados

Introdução à Arquitetura IA-32

- Modo Protegido (Protected Mode)
 - Age como um 80386 +, como deveria!!!
 - Habilita 4 níveis de proteção:
 - Level 0 – kernel mode
 - Level 1 – RARAMENTE UTILIZADO
 - Level 2 – RARAMENTE UTILIZADO
 - Level 3 – user mode
 - Linux utiliza levels 0 e 3



Introdução à Arquitetura IA-32

- Level 0 – kernel mode
 - Acesso total à máquina – **PODE EXECUTAR QUALQUER INSTRUÇÃO EXISTENTE**
 - O Sistema Operacional é executado neste modo

Introdução à Arquitetura IA-32

- Level 3 – user mode
 - bloqueia o acesso/execução à instruções críticas
 - bloqueia o acesso/alteração à registradores de controle

Introdução à Arquitetura IA-32

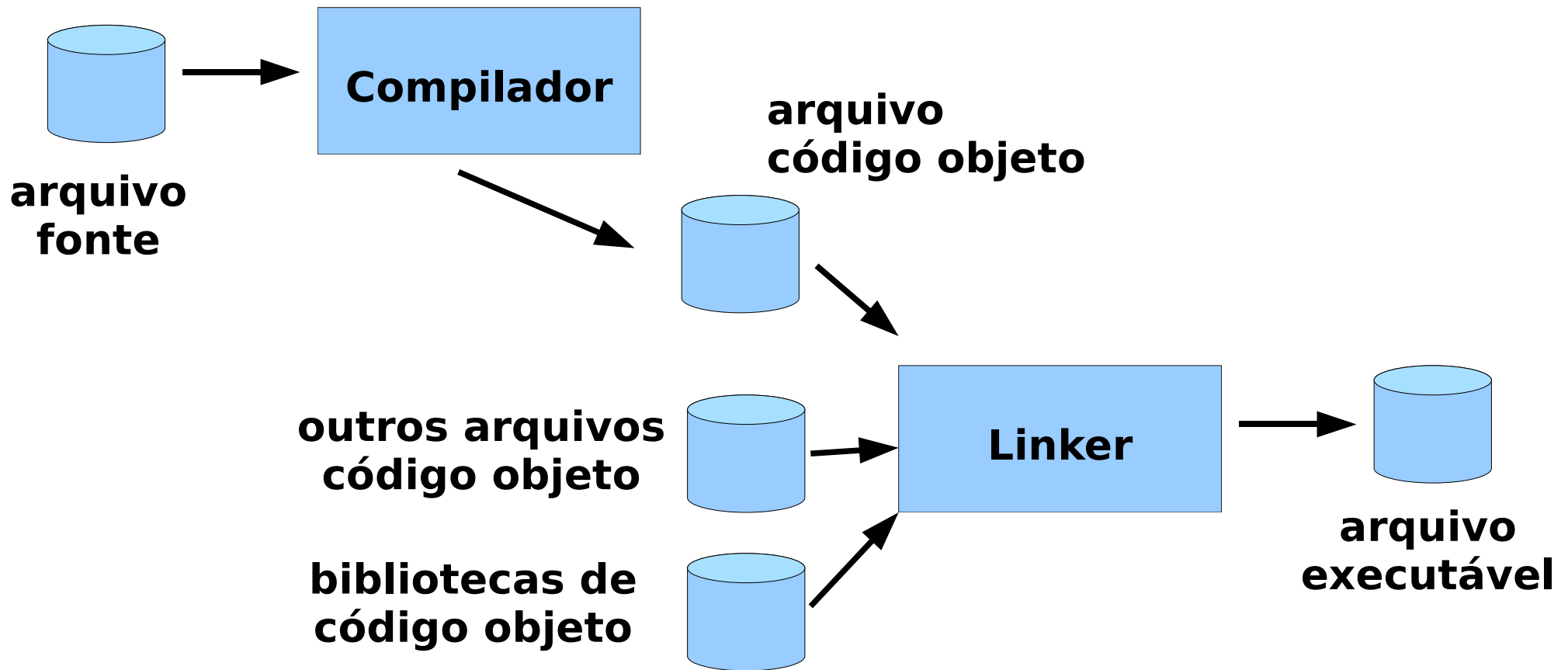
- Linguagens de alto nível: características
 - Portabilidade
 - Fácil manutenção
 - Menos linhas de código fonte
- Tipos
 - Compiladas
 - Interpretadas
 - Híbridas

Introdução à Arquitetura IA-32

- Compiladas
 - programa texto é convertido em programa binário
 - geralmente é feito em duas etapas
 - compilação propriamente dita
 - link edição

Introdução à Arquitetura IA-32

- Compiladas



Introdução à Arquitetura IA-32

- **Compiladas**

Compilador

Gera código dependente de:

- **Processador**
- **Sistema Operacional**

Ex.:

- **IA-32 com Linux**
- **IA-32 com MacOS X**

Código fonte é o mesmo, mas código executável é diferente

Introdução à Arquitetura IA-32

- Interpretadas
 - O programa não executa diretamente no processador
 - um interpretador se encarrega de ler o programa fonte e, enquanto lê, gerar instruções que são imediatamente colocadas para execução

Introdução à Arquitetura IA-32

- Híbridas
 - Programa fonte é compilado para um máquina virtual.
 - Ex.: Java
 - Programa Java -> Byte Code -> JVM

Introdução à Arquitetura IA-32

- Linguagem de baixo nível: Assembly
 - Assembly: linguagem de montagem
 - Assembler: montador (equivalente ao compilador)

Introdução à Arquitetura IA-32

- Exemplos:
 - C (Página 8 – livro Professional Assembly Language)
 - código de máquina IA-32 (Página 8)
 - Assembly IA-32 (Página 11)

Introdução à Arquitetura IA-32

- Exemplos: Programa em C

```
int main()  
{  
    int i = 1;  
    exit(0);  
}
```

Introdução à Arquitetura IA-32

- Exemplos: Programa em Cód. de Máquina

55

89 E5

83 EC 08

C7 45 FC 01 00 00 00

83 EC 0C

64 00

E8 D1 FE FF FF

Introdução à Arquitetura IA-32

- Exemplos: Programa em Assembly

```
push %ebp
```

```
mov %esp, %ebp
```

```
sub $0x8, %esp
```

```
movl $0x1, -4(%ebp)
```

```
sub $0xc, %esp
```

```
push $0x0
```

```
call 8048348
```