

MC542

**Organização de Computadores
Teoria e Prática**

2006

Prof. Paulo Cesar Centoducatte

ducatte@ic.unicamp.br

www.ic.unicamp.br/~ducatte

MC542

Arquitetura de Computadores

Conjunto de Instruções

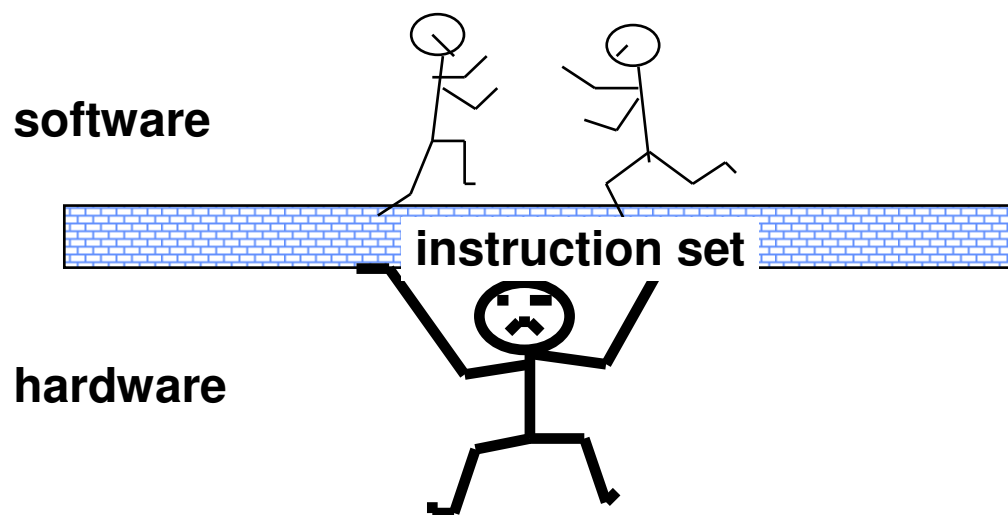
**"Computer Organization and Design:
The Hardware/Software Interface"
(Capítulo 2)**

Conjuntos de Instruções

- **Introdução**
- **Classificação do ISA**
- **Endereçamento de Memória**
- **Operações no Conjunto de Instruções**
- **Tipo e tamanho dos operandos**
- **Codificação do ISA**
 - 80x86
 - DLX
- **A Arquitetura MIPS**
- **Impacto Compiladores vs ISA**

Introdução

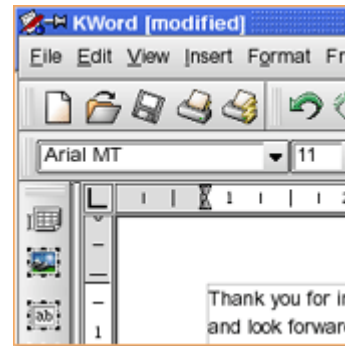
O ISA é a porção da máquina visível ao programador (nível de montagem) ou aos projetistas de compiladores



1. Quais as vantagens e desvantagens das diversas alternativas de ISA.
2. Como as linguagens e compiladores afetam (ou são afetados) o ISA.
3. Arquitetura DLX como exemplo de arquitetura RISC.

Introdução - ISA

software

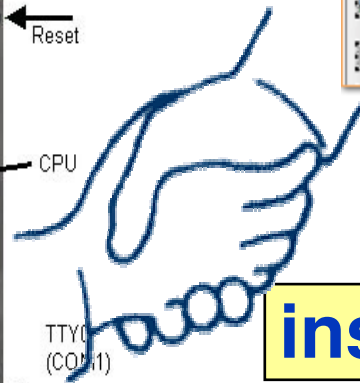


Power LEDs

Reset

CPU

TTY0 (COM1)



instruction set

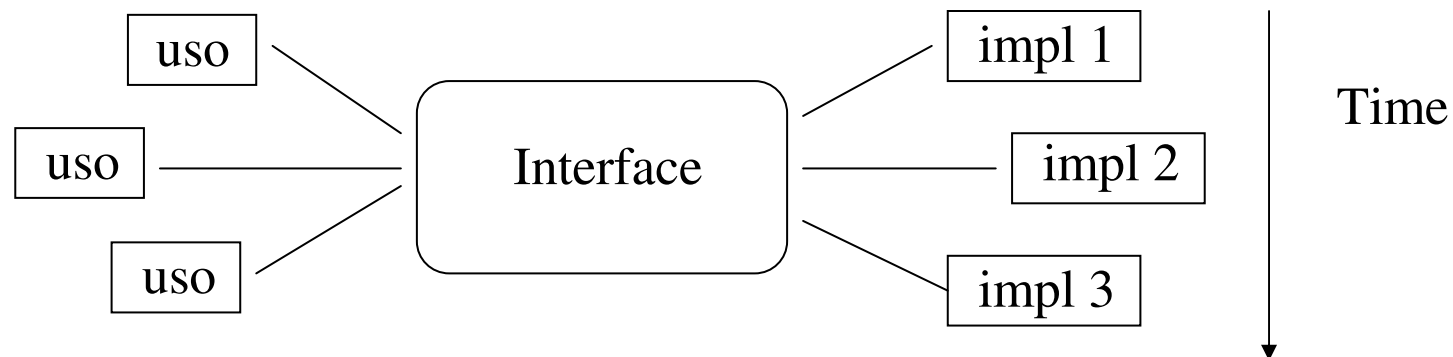
Interface entre o Hardware e o Usuário

hardware

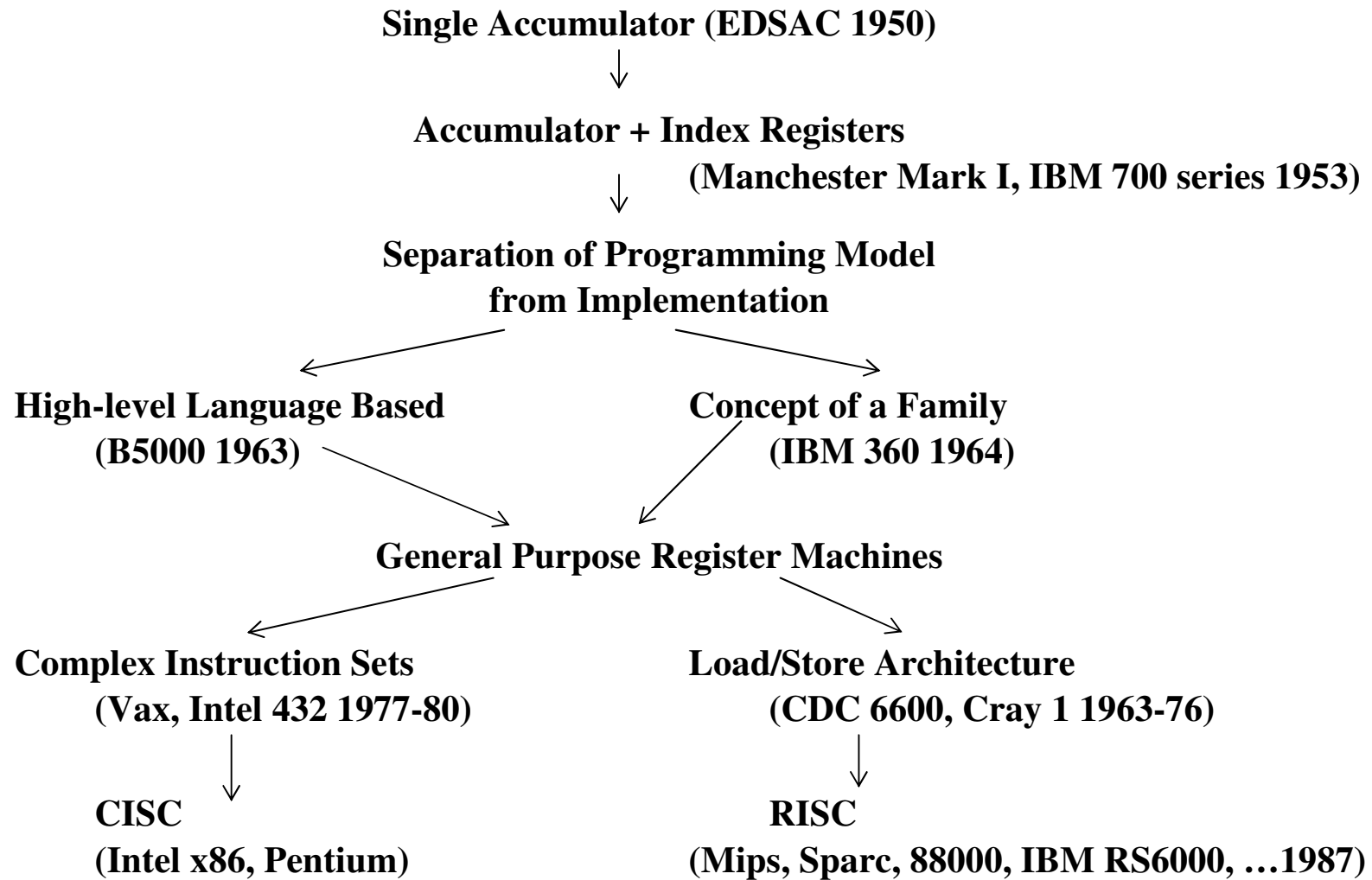
Introdução - Interface

Uma boa interface:

- Sobrevive a várias implementações (portabilidade, capacidade, escalabilidade, ...).
- Usada de diferentes formas (generalidade).
- Provê funcionalidades para os níveis superiores.
- Permite uma implementação eficiente no nível inferior (hw).



Evolução dos ISAs



Evolução dos ISAs

- As maiores vantagens em uma arquitetura, em geral, são associadas com as mudanças do ISA
 - Ex: Stack vs General Purpose Registers (GPR)
- Decisões de projeto que devem ser levadas em consideração:
 - tecnologia
 - organização
 - linguagens de programação
 - tecnologia em compiladores
 - sistemas operacionais

Projeto de um ISA

5 aspectos principais

- Número de operandos (explícitos) (0,1,2,3)
- Armazenamento do Operando. **Aonde ele está?**
- Endereço Efetivo. **Como é especificado?**
- Tipo & Tamanho dos operandos. **byte, int, float, ...
como eles são especificados?**
- Operações **add, sub, mul, ...
como elas são especificados?**

Projeto de um ISA

Outros aspectos

- **Sucessor** Como é especificado?
- **Condições** Como são determinadas?
- **Codificação** Fixa ou Variável? Tamanho?
- **Paralelismo**

Classes básicas de ISA

Accumulator:

1 address	add A	$acc \leftarrow acc + mem[A]$
1+x address	addx A	$acc \leftarrow acc + mem[A + x]$

Stack:

0 address	add	$tos \leftarrow tos + next$
-----------	-----	-----------------------------

General Purpose Register:

2 address	add A B	$EA(A) \leftarrow EA(A) + EA(B)$
3 address	add A B C	$EA(A) \leftarrow EA(B) + EA(C)$

Instruções da ALU podem ter dois ou três operandos.

Load/Store:

0 Memory	load R1, Mem1
	load R2, Mem2
	add R1, R2
1 Memory	add R1, Mem2

Instruções da ALU podem ter 0, 1, 2, 3 operandos.

Classes básicas de ISA

Código nas diferentes classes de endereçamento para:

$$C = A + B.$$

Stack	Accumulator	Register (Register-memory)	Register (load-store)
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R1, B	Load R2, B
Add	Store C	Store C, R1	Add R3, R1, R2
Pop C			Store C, R3

Vantagens e Desvantagens

Stack

V: Forma simples para avaliação de expressões (notação polonesa). Instruções curtas, podem levar a uma boa densidade de código.

D: A pilha não pode ser acessada randomicamente. Esta limitação torna mais complicada a geração de código eficiente. A implementação eficiente também é mais difícil, a pilha torna-se um gargalo.

Vantagens e Desvantagens

Accumulator

V: Minimiza o número de estados internos da máquina. Instruções curtas.

D: Uma vez que o acumulador é um temporário, o tráfego na memória é alto.

Vantagens e Desvantagens

Register

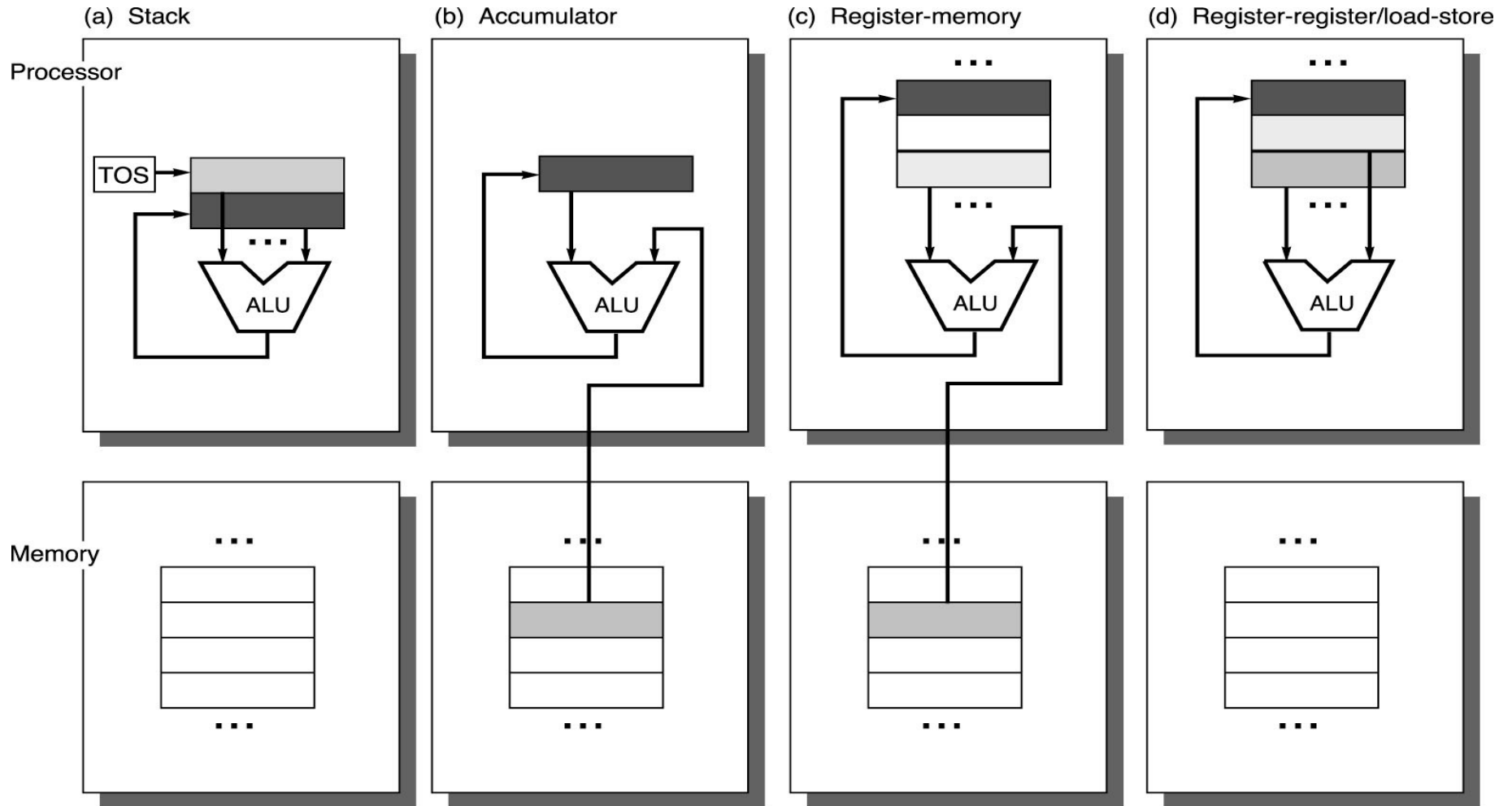
V: Formato mais geral para geração de código.

D: Todos os operandos devem ser nomeados, instruções mais longas.

Enquanto as máquinas mais antigas usavam o estilo "stack" ou "accumulator", arquiteturas modernas (projetadas nos últimos 10-20 anos) usam "general-purpose register"

- Registradores são mais rápidos
- O uso de Registradores é mais fácil para os compiladores
- Registradores podem ser utilizados mais efetivamente como forma de armazenamento

Tipos de Máquinas



Machine	Number of general-purpose registers	Architectural style	Year
EDSAC	1	Accumulator	1949
IBM 701	1	Accumulator	1953
CDC 6600	8	Load-store	1963
IBM 360	16	Register- memory	1964
DEC PDP-8	1	Accumulator	1965
DEC PDP-11	8	Register- memory	1970
Intel 8008	1	Accumulator	1972
Motorola 6800	2	Accumulator	1974
DEC VAX	16	Register- memory, memory- memory	1977
Intel 8086	1	Extended accumulator	1978
Motorola 68000	16	Register- memory	1980
Intel 80386	8	Register- memory	1985
MIPS	32	Load-store	1985
HP PA-RISC	32	Load-store	1986
SPARC	32	Load-store	1987
PowerPC	32	Load-store	1992
DEC Alpha	32	Load-store	1992

Registradores no Intel 80X86

GPR0	EAX	Accumulator
GPR1	ECX	Count register, string, loop
GPR2	EDX	Data Register; multiply, divide
GPR3	EBX	Base Address Register
GPR4	ESP	Stack Pointer
GPR5	EBP	Base Pointer - for base of stack seg.
GPR6	ESI	Index Register
GPR7	EDI	Index Register
	CS	Code Segment Pointer
	SS	Stack Segment Pointer
	DS	Data Segment Pointer
	ES	Extra Data Segment Pointer
	FS	Data Seg. 2
	GS	Data Seg. 3
PC	EIP	Instruction Counter
	Eflags	Condition Codes

Modos de Endereçamento

Interpretando endereços de memória

Qual objeto é acessado em função do endereço e qual o seu tamanho?

Objetos endereçados a byte - um endereço refere-se ao número de bytes contados do início da memória.

Little Endian - o byte cujo endereço é xx00 é o byte menos significativo da palavra.

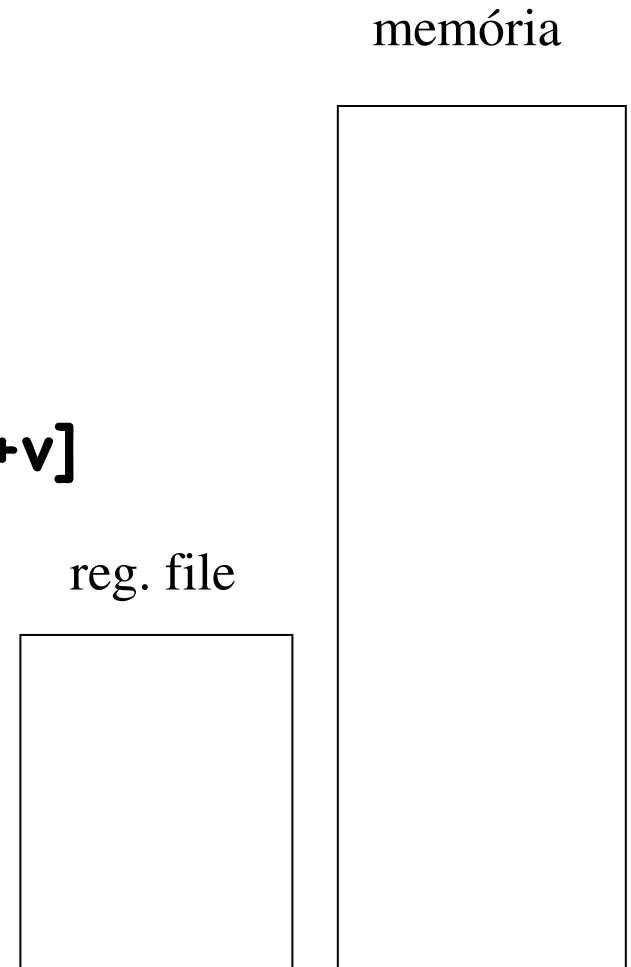
Big Endian - o byte cujo endereço é xx00 é o mais significativo da palavra.

Alinhamento - o dado deve ser alinhado em fronteiras iguais a seu tamanho.

- $\text{address} / \text{sizeof}(\text{datatype}) == 0$
- bytes pode ser alinhado em qualquer endereço
- inteiros de 4 bytes são alinhados em endereços múltiplos de 4

Modos de Endereçamento

- Register direct R_i
- Immediate (literal) v
- Direct (absolute) $M[v]$
- Register indirect $M[R_i]$
- Base+Displacement $M[R_i + v]$
- Base+Index $M[R_i + R_j]$
- Scaled Index $M[R_i + R_j * d + v]$
- Autoincrement $M[R_i++]$
- Autodecrement $M[R_i--]$
- Memory indirect $M[M[R_i]]$



Modos de Endereçamento

Addressing Mode	Example Instruction	Meaning	When Used
Register	Add R4, R3	$R[R4] \leftarrow R[R4] + R[R3]$	When a value is in a register.
Immediate	Add R4, #3	$R[R4] \leftarrow R[R4] + 3$	For constants.
Displacement	Add R4, 100(R1)	$R[R4] \leftarrow R[R4] + M[100+R[R1]]$	Accessing local variables.
Register Deferred	Add R4, (R1)	$R[R4] \leftarrow R[R4] + M[R[R1]]$	Using a pointer or a computed address.
Absolute	Add R4, (1001)	$R[R4] \leftarrow R[R4] + M[1001]$	Used for static data.

Modos de Endereçamento

Displacement

Qual o tamanho do deslocamento?

r: Depende do espaço reservado na codificação da instrução.

Para endereços dentro do alcance do deslocamento:

```
Add R4, 10000 (R0)
```

Para endereços fora do alcance do deslocamento, o compilador deve gerar:

```
Load R1, address
```

```
Add R4, 0 (R1)
```

No IA32 e no DLX, o espaço alocado é de 16 bits.

Modos de Endereçamento

Immediate Address

Usado quando se deseja ter o valor numérico codificado junto com a instrução (constantes).

```
a = b + 3;
```

```
if ( a > 17 )
```

```
goto Addr
```

Em Assembler:

```
Load R2, 3  
Add R0, R1, R2
```

```
Load R2, 17  
CMPBGT R1, R2
```

```
Load R1, Address  
Jump (R1)
```

Em Assembler:

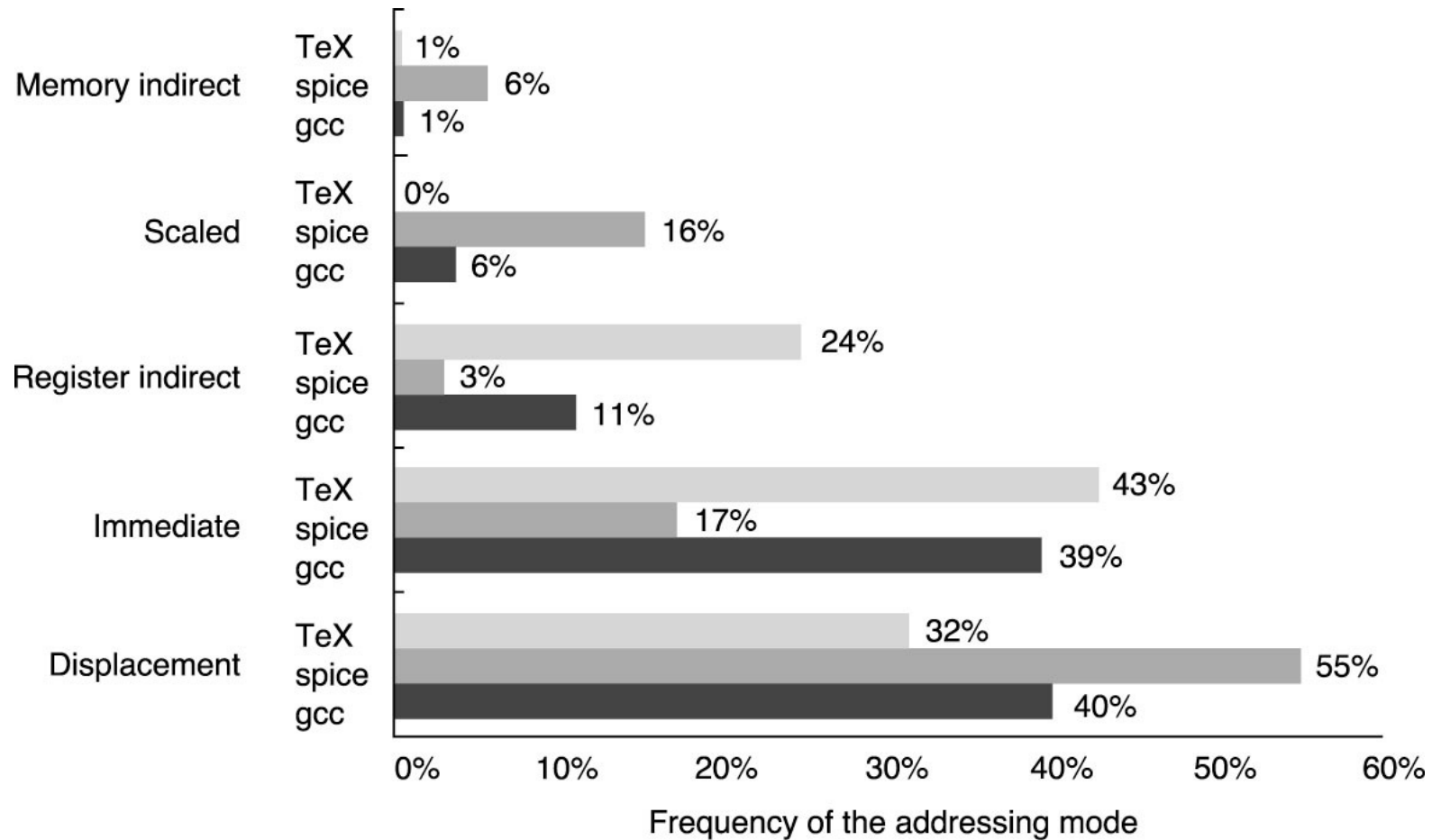
(com imediato na adição)

```
Addi R0, R1, 3
```

```
Load R2, 17  
CMPBGT R1, R2
```

```
Load R1, Address  
Jump (R1)
```

Modos de Endereçamento



Operações em um ISA

Arithmetic and logical -	and, add
Data transfer -	move, load
Control -	branch, jump, call
System -	system call, traps
Floating point -	add, mul, div, sqrt
Decimal -	add, convert
String -	move, compare
Multimedia -	2D, 3D? e.g., Intel MMX and Sun VIS

Operações em um ISA

Rank	80x86 instruction	Integer average (% total executed)
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

FIGURE 2.11 The top 10 instructions for the 80x86.

Instruções de Controle

(20% das instruções são desvios condicionais)

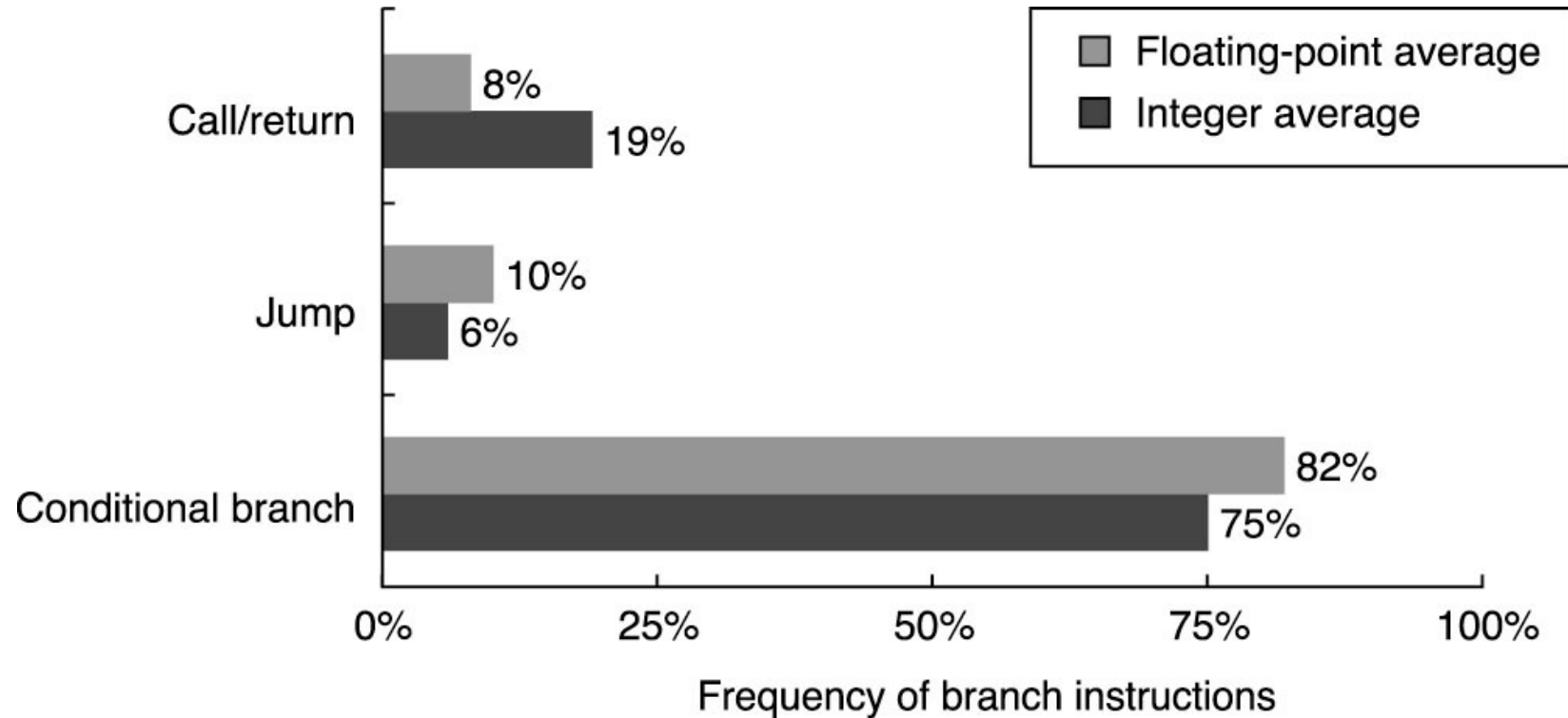
Control Instructions:

- tomar ou não
- aonde é o alvo
- link return address
- salvar ou restaurar

Instruções que alteram o PC:

- (condicional) branches, (incondicional) jumps
- chamadas de funções, retorno de funções
- system calls, system returns

Instruções de desvio



© 2003 Elsevier Science (USA). All rights reserved.

Tipos e Tamanhos dos Operandos

O tipo do operando, em geral, é codificado no opcode - LDW significa "loading of a word".

Tamanhos típicos são:

- Character (1 byte)
- Half word (16 bits)
- Word (32 bits)
- Single Precision Floating Point (1 Word)
- Double Precision Floating Point (2 Words)

Inteiros são representados em complemento de dois.

Floating point, em geral, usa o padrão IEEE 754.

Algumas linguagens (como COBOL) usam packed decimal.

RISC vs CISC

RISC = Reduced Instruction Set Computer

- Conjunto de Instruções pequeno
- Instruções de tamanho fixo
- Operações executadas somente em registradores
- Chip simples, em geral, executam com velocidade de clock elevada.

CISC = Complex Instruction Set Computer

- Conjunto de Instruções grande
- Instruções Complexas e de tamanho variável
- Operações Memória-Memória

Projeto \Rightarrow CISC premissas

- Conjunto de Instruções farto pode simplificar o compilador.
- Conjunto de Instruções farto pode aliviar o software.
- Conjunto de Instruções farto pode dar qualidade a arquitetura.
 - Se o tempo de execução for proporcional ao tamanho do programa, técnicas de arquitetura que levem a programas menores também levam a computadores mais rápidos.

Projeto \Rightarrow RISC premissas

- As funções devem ser simples, a menos que haja uma razão muito forte em contrário.
- Decodificação simples e execução pipelined são mais importantes que o tamanho do programa.
- Tecnologias de compiladores podem ser usadas para simplificar as instruções ao invés de produzirem instruções complexas.

Codificação do conjunto de Instruções

codificação do 80x86

Visual Studio - sem otimizações

```
for ( index = 0; index < iterations; index++ )
```

```
0040D3AF    C7 45 F0 00 00 00 00    mov     dword ptr [ebp-10h],0
0040D3B6    EB 09                   jmp     main+0D1h (0040d3c1)
0040D3B8    8B 4D F0               mov     ecx,dword ptr [ebp-10h]
0040D3BB    83 C1 01               add     ecx,1
0040D3BE    89 4D F0               mov     dword ptr [ebp-10h],ecx
0040D3C1    8B 55 F0               mov     edx,dword ptr [ebp-10h]
0040D3C4    3B 55 F8               cmp     edx,dword ptr [ebp-8]
0040D3C7    7D 15                   jge     main+0EEh (0040d3de)
```

```
    long_temp = (*alignment + long_temp) % 47;
```

```
0040D3C9    8B 45 F4               mov     eax,dword ptr [ebp-0Ch]
0040D3CC    8B 00                   mov     eax,dword ptr [eax]
0040D3CE    03 45 EC               add     eax,dword ptr [ebp-14h]
0040D3D1    99                       cdq
0040D3D2    B9 2F 00 00 00        mov     ecx,2Fh
0040D3D7    F7 F9                   idiv   eax,ecx
0040D3D9    89 55 EC               mov     dword ptr [ebp-14h],edx
0040D3DC    EB DA                   jmp     main+0C8h (0040d3b8)
```

Codificação do conjunto de Instruções

codificação do 80x86

Visual Studio - com otimizações

```
for ( index = 0; index < iterations; index++ )
```

```
00401000      8B 0D 40 54 40 00      mov     ecx,dword ptr ds:[405440h]
00401006      33 D2                  xor     edx,edx
00401008      85 C9                  test    ecx,ecx
0040100A      7E 14                  jle    00401020
0040100C      56                     push   esi
0040100D      57                     push   edi
0040100E      8B F1                  mov     esi,ecx
```

```
    long_temp = (*alignment + long_temp) % 47;
```

```
00401010      8D 04 11              lea    eax,[ecx+edx]
00401013      BF 2F 00 00 00       mov     edi,2Fh
00401018      99                    cdq
00401019      F7 FF                idiv   eax,edi
0040101B      4E                    dec    esi
0040101C      75 F2                jne    00401010
0040101E      5F                    pop    edi
0040101F      5E                    pop    esi
00401020      C3                    ret
```

Codificação do conjunto de Instruções

codificação do 80x86

gcc - com otimizações

```
for ( index = 0; index < iterations; index++ )
```

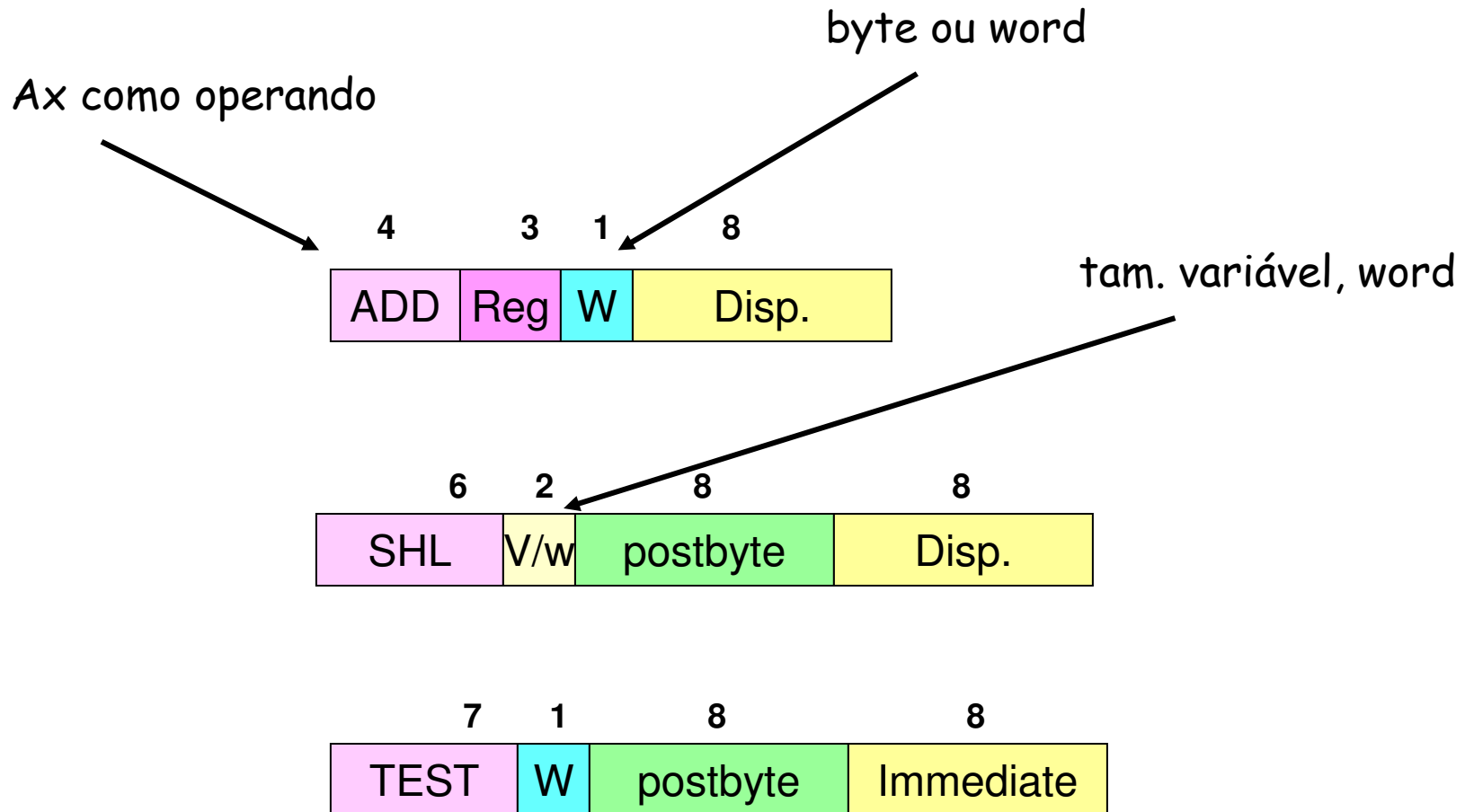
```
0x804852f <main+143>:  add    $0x10,%esp
0x8048532 <main+146>:  lea    0xffffffff8(%ebp),%edx
0x8048535 <main+149>:  test   %esi,%esi
0x8048537 <main+151>:  jle    0x8048543 <main+163>
0x8048539 <main+153>:  mov    %esi,%eax
0x804853b <main+155>:  nop
0x804853c <main+156>:  lea    0x0(%esi,1),%esi
```

```
long_temp = (*alignment + long_temp) % 47;
```

```
0x8048540 <main+160>:  dec    %eax
0x8048541 <main+161>:  jne    0x8048540 <main+160>
0x8048543 <main+163>:  add    $0xffffffff4,%esp
```

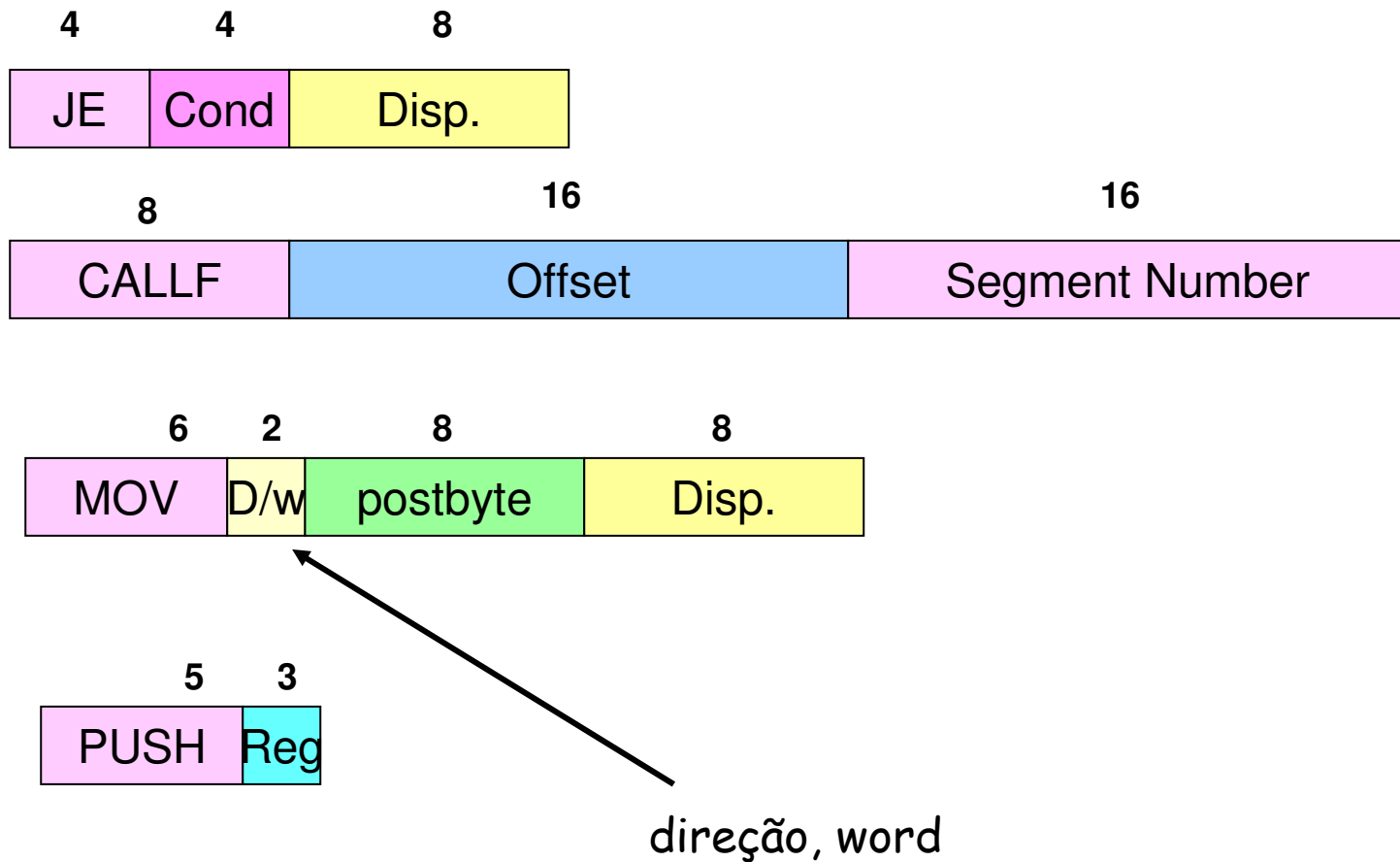
Codificação do conjunto de Instruções

codificação do 80x86 - 1 a 6 bytes



Codificação do conjunto de Instruções

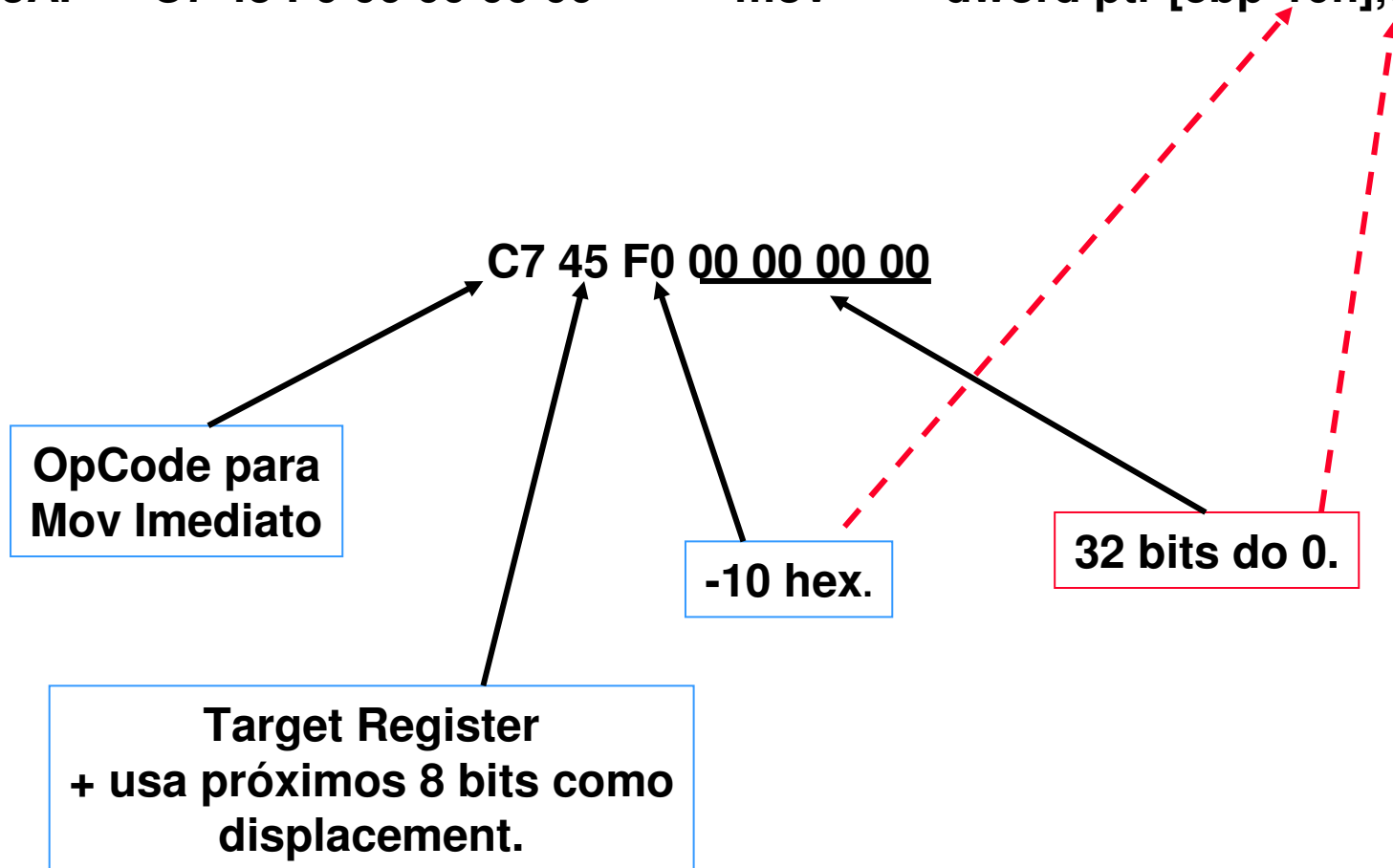
codificação do 80x86 - 1 a 6 bytes



Codificação do conjunto de Instruções

codificação do 80x86

0040D3AF C7 45 F0 00 00 00 00 mov dword ptr [ebp-10h],0



Codificação do conjunto de Instruções

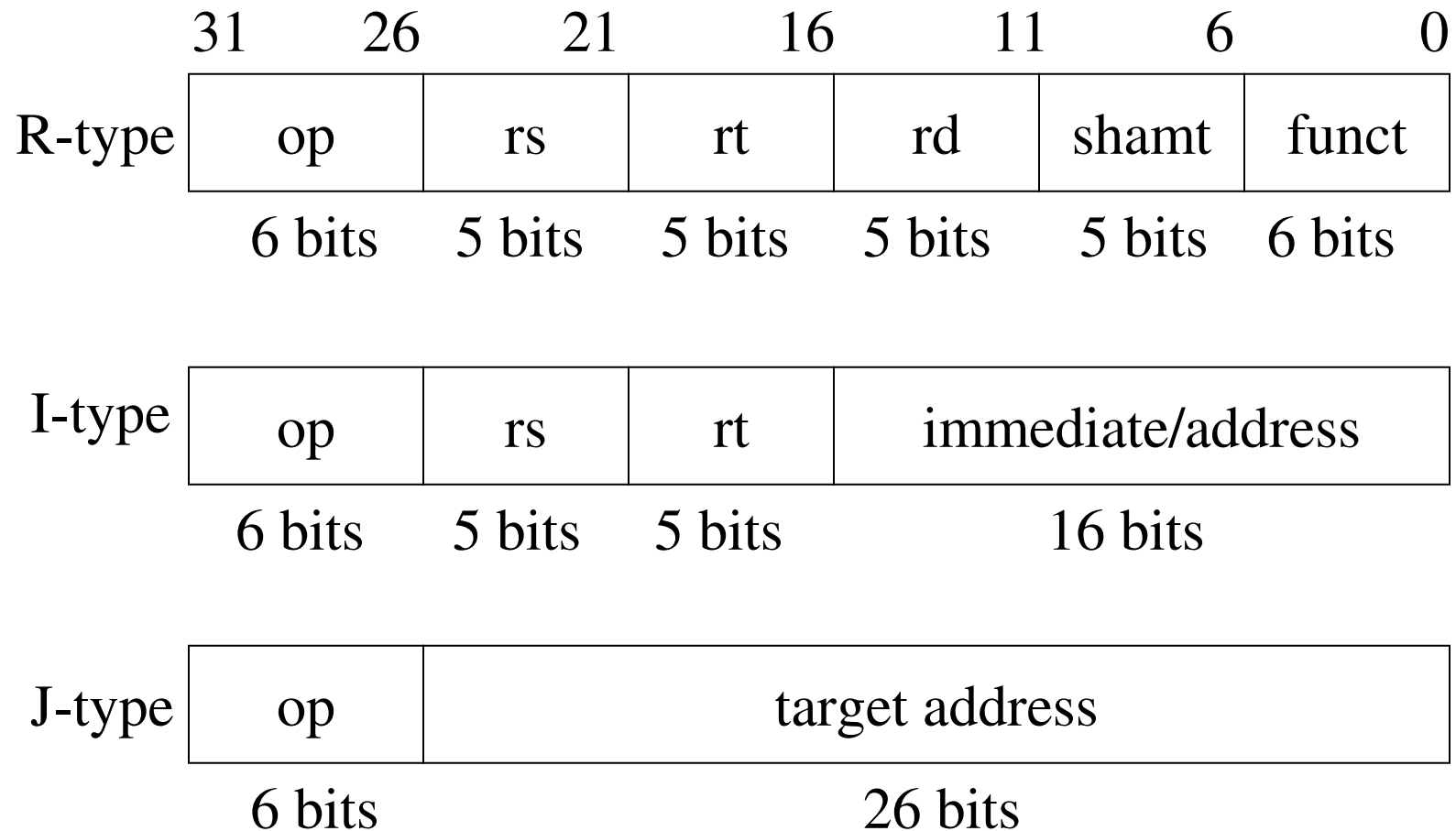
codificação de um RISC típico

- instruções de tamanho fixo (32-bit) (3 formatos)
- 32 32-bit general-purpose registers (R0 contains zero, números de precisão dupla usam dois registradores)
- Modo de endereçamento simples para load/store:
base + displacement (sem indireção)
- Desvios condicionais simples
- Delayed branch para evitar penalidade no pipeline
- Exemplos: DLX, SPARC, MIPS, HP PA-RISC, DEC Alpha, IBM/Motorola PowerPC, Motorola M88000

Codificação do conjunto de Instruções

codificação de um RISC típico

3 formatos - DLX



Arquitetura MIPS

Organização

Acesso à memória alinhado a:

- Byte - dados

0	8 bits of data
1	8 bits of data
2	8 bits of data
3	8 bits of data
4	8 bits of data
5	8 bits of data
6	8 bits of data

...

- Word - instruções

0	32 bits of data
4	32 bits of data
8	32 bits of data
12	32 bits of data

Arquitetura MIPS

Organização

- Palavras de 32 bits
- 3 formatos de instruções

R	op	rs	rt	rd	shamt	funct
I	op	rs	rt	16 bit address		
J	op	26 bit address				

Arquitetura MIPS

Organização

Código C: `A[300] = h + A[300];`

Código MIPS: `lw $t0, 1200($t1)`
`add $t0, $s2, $t0`
`sw $t0, 1200($t1)`

op	rs	rt	rd	address/shamt	address/funct
35	9	8		1200	
0	18	8	8	0	32
43	9	8		1200	

Arquitetura MIPS

Organização

Name	Register number	Usage
\$zero	0	the constant value 0
\$v0-\$v1	2-3	values for results and expression evaluation
\$a0-\$a3	4-7	arguments
\$t0-\$t7	8-15	temporaries
\$s0-\$s7	16-23	saved
\$t8-\$t9	24-25	more temporaries
\$gp	28	global pointer
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	return address

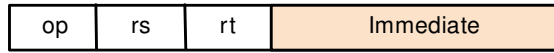
\$1 = \$at: reservado para o assembler

\$26-27 = \$k0-\$k1: reservados para o sistema operacional

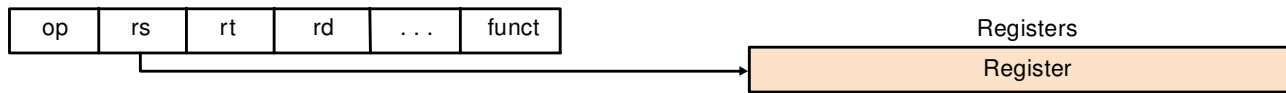
Arquitetura MIPS

Organização

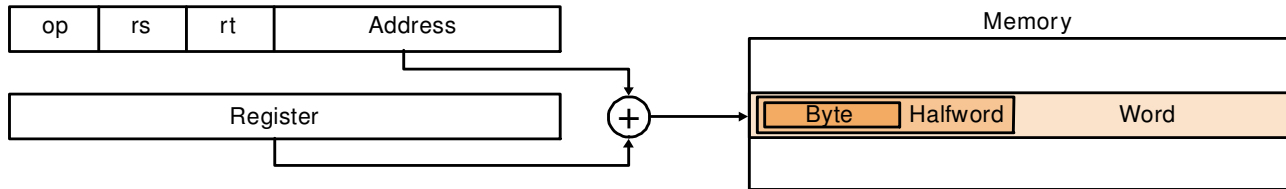
1. Immediate addressing



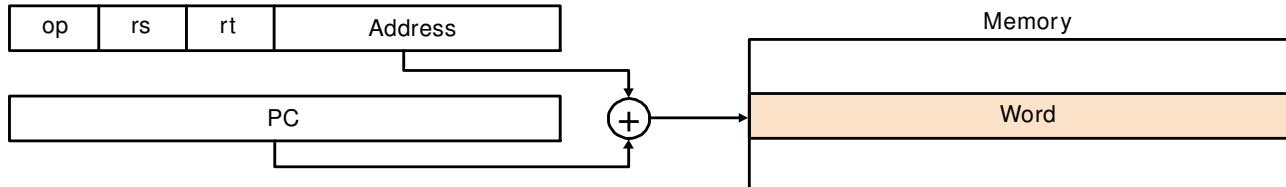
2. Register addressing



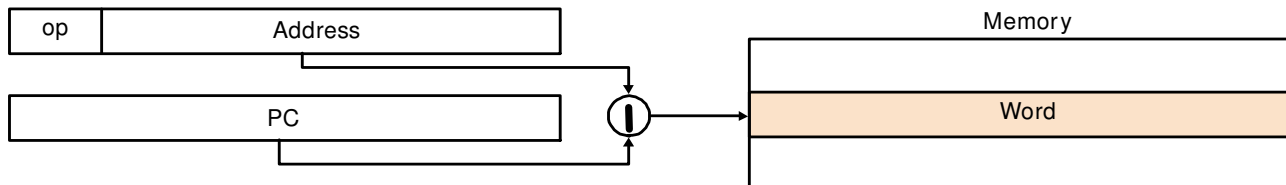
3. Base addressing



4. PC-relative addressing



5. Pseudodirect addressing



Impacto da tecnologia de Compiladores nas Decisões sobre a Arquitetura

- A interação entre os compiladores e as linguagens de alto nível afetam significativamente como os programas usam o conjunto de instruções.
- Como as variáveis são alocadas e endereçadas? Quantos registradores são necessários?
- Qual o impacto das otimizações no mix de instruções efetivamente usados?
- Como as estruturas de controle são usadas e com qual frequência?

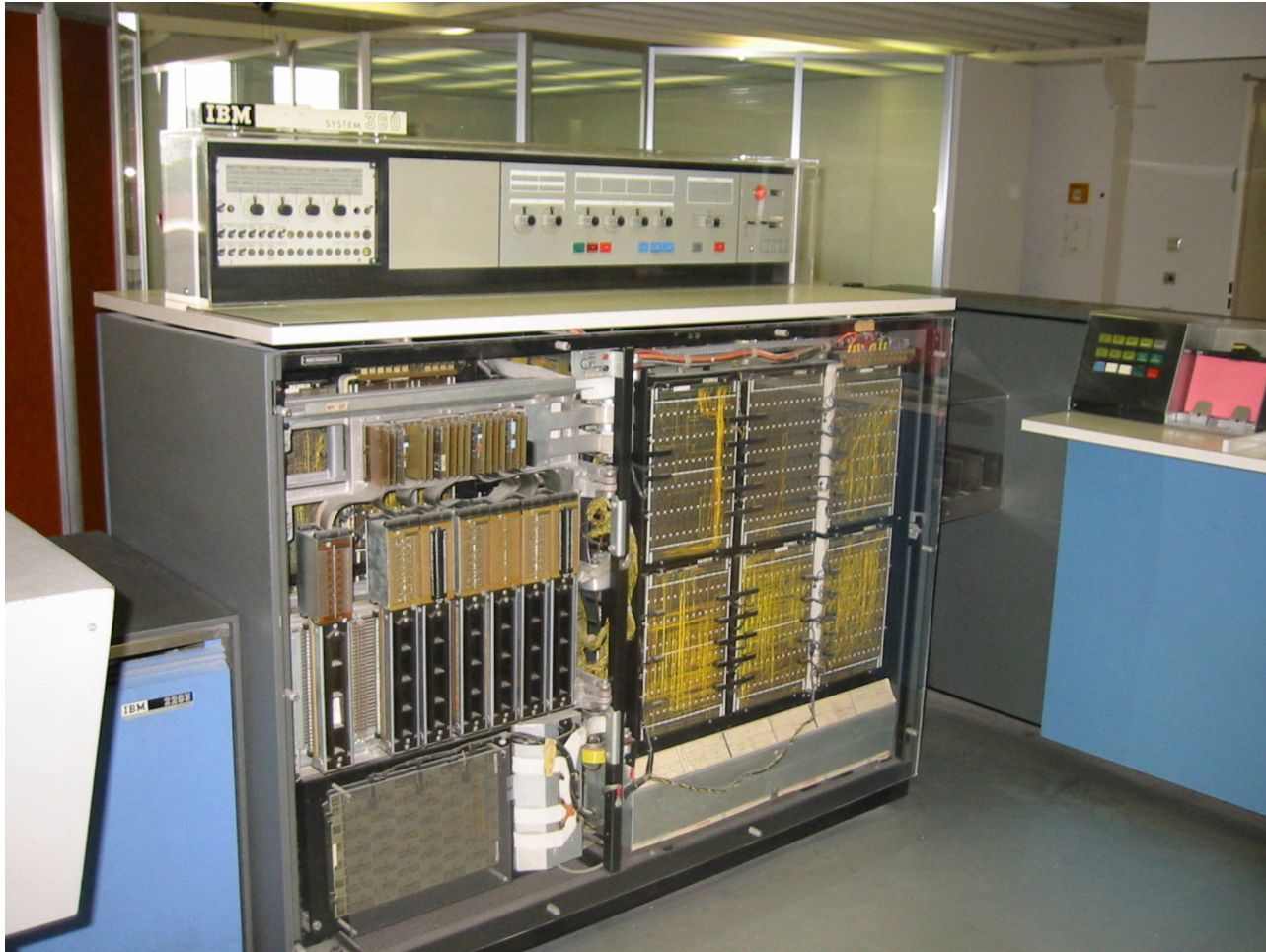
Propriedades do ISA que Simplificam o Compilador

- Regularidade.
- Fornecer Primitivas (e não soluções)
- Simplificar as alternativas de compromissos.
- Instruções que usam quantidades conhecidas em tempo de compilação como constantes.

Métricas para ISA

- **Regularidade**
 - Não usar registradores especiais, poucos casos especiais, todos modos de operandos disponíveis para qualquer tipo de dados ou instruções.
- **Primitivas, não soluções**
- **Completude**
 - Suporte a um grande número de operações e aplicações alvo
 - Os Recursos necessários devem ser facilmente identificados
- **Fácil para a compilação**
- **Fácil de se implementar**
- **Escalabilidade**

IBM 360



Cray



MO401
3.50