

Lic. Ciências da Computação

1º ano

2007/08

A.J.Proença

Tema
ISA do IA32

Estrutura do tema ISA do IA32

1. Desenvolvimento de programas no IA32 em Linux
2. Acesso a operandos e operações
3. Suporte a estruturas de controlo
4. Suporte à invocação/retorno de funções
5. Acesso e manipulação de dados estruturados
6. Análise comparativa: IA-32 (CISC) e MIPS (RISC)

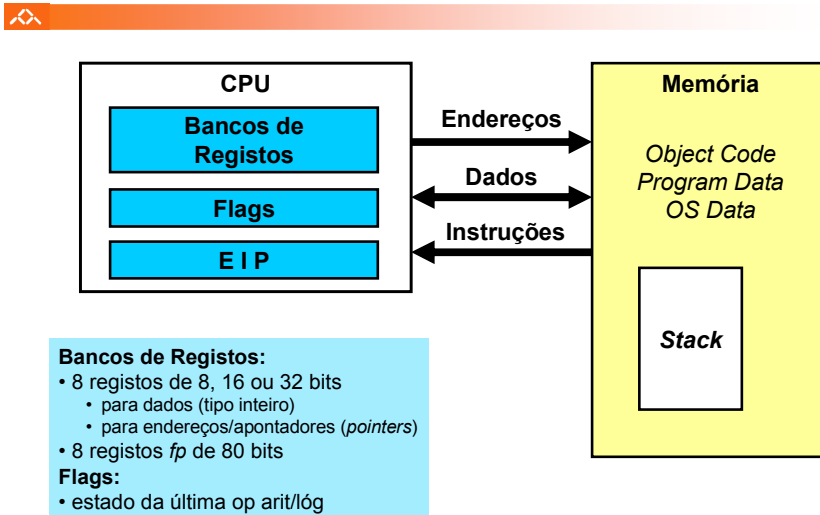
Evolução do Intel x86 : pré-Pentium
(visão do programador)

Nome	Data	Nº transístores
8086	1978	29K
– processador 16-bit; base do IBM PC & DOS – espaço de endereçamento limitado a 1MB (DOS apenas vê 640K)		
80286	1982	134K
– endereçamento mais complexo, mas de utilidade duvidosa – base do IBM PC-AT & Windows		
386	1985	275K
– estendido para 32 bits, com “flat addressing” – capaz de correr Unix – Linux/gcc não usa instruções introduzidas em versões posteriores!		
486	1989	1.9M

→ **primeiro IA32 !!**Evolução do IA32: família Pentium
(visão do programador)

Nome	Data	Nº transístores
Pentium	1993	3.1M
PentiumPro	1995	6.5M
– com instruções de <i>move</i> condicional – alteração significativa na microarquitetura		
Pentium/MMX	1997	4.5M
– com instruções para operar com vectores de 64-bits com dados inteiros de 1, 2, ou 4 bytes		
Pentium II	1997	7M
(= Pro + MMX)		
Pentium III	1999	8.2M
– com instruções “streaming SIMD” para operar com vectores de 128-bits com dados int ou fp de 1, 2, ou 4 bytes		
Pentium 4	2001	42M
– 144 novas instruções “streaming SIMD” e com dados de 8-bytes		

O modelo CPU-Mem no IA32 (visão do programador)



Representação de operandos no IA32

Tamanhos de objectos em C (em *bytes*)

Declaração em C	Designação Intel	Tamanho IA32
char	byte	1
short	word	2
int	double word	4
long int	double word	4
float	single precision	4
double	double precision	8
long double	extended precision	10/12
char* (ou qq outro apontador)	double word	4

Ordenação dos bytes na memória

- O IA32 é um processador *little endian*
- Exemplo:
representação de 0x01234567, cujo endereço dado por `&var` é 0x100

0x100	0x101	0x102	0x103				
	67	45	23	01			

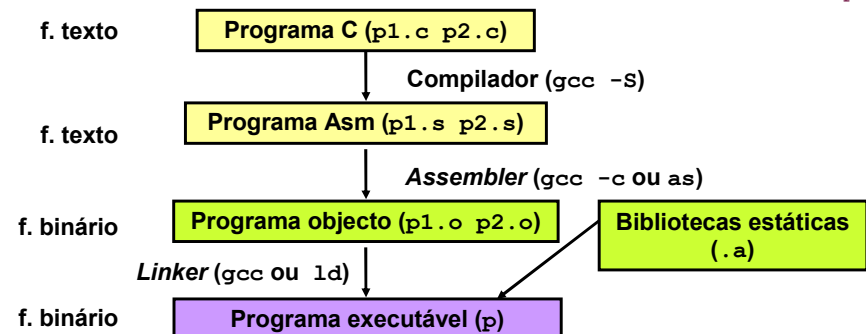
Tipos de instruções básicas no IA32

Operações primitivas:

- Efectuar operações aritméticas/lógicas com dados em registo ou em memória
 - dados do tipo *integer* de 1, 2 ou 4 bytes
 - dados em formato *fp* de 4, 8 ou 10 bytes
 - apenas dados escalares: *arrays* ou *structures* são vistos apenas como *bytes* continuamente alocados em memória
- Transferir dados entre células de memória e um registo
 - carregar (*load*) em registo dados da memória
 - armazenar (*store*) na memória dados em registo
- Transferir o controlo da execução das instruções
 - saltos incondicionais de/para funções/procedimentos
 - saltos ramificados (*branches*) condicionais

Conversão de um programa em C em código executável (exemplo)

- Código C nos ficheiros `p1.c p2.c`
- Comando para a "compilação": `gcc -O p1.c p2.c -o p`
 - usa optimizações (`-O`)
 - coloca binário resultante no ficheiro `p`



Código C

```
int sum(int x, int y)
{
    int t = x+y;
    return t;
}
```

Assembly gerado

```
_sum:
    pushl   %ebp
    movl    %esp,%ebp
    movl    12(%ebp),%eax
    addl    8(%ebp),%eax
    movl    %ebp,%esp
    popl    %ebp
    ret
```

gcc -O -S p2.c

p2.s

Assembly

```
_sum:
    pushl   %ebp
    movl    %esp,%ebp
    movl    12(%ebp),%eax
    addl    8(%ebp),%eax
    movl    %ebp,%esp
    popl    %ebp
    ret
```

p2.s

Código binário

```
0x401040 <sum>:
0x55
0x89
0xe5 • Começa
0x8b no
0x45 endereço
0x0c 0x401040
0x03
0x45 • Total 13
0x08 bytes
0x89
0xec
0x5d • Cada
0xc3 instrução
1, 2, ou 3
bytes
```

Papel do linker

- Resolve as referências entre ficheiros
- Junta as *static run-time libraries*
 - E.g., código para `malloc`, `printf`
- Algumas bibliotecas são *dynamically linked*
 - E.g., junção ocorre no início da execução

Desmontagem de código binário executável (exemplo)

objdump -d p

Código binário desmontado

```
00401040 <_sum>:
0:      55                push    %ebp
1:      89 e5             mov     %esp,%ebp
3:      8b 45 0c          mov     0xc(%ebp),%eax
6:      03 45 08          add     0x8(%ebp),%eax
9:      89 ec             mov     %ebp,%esp
b:      5d                pop     %ebp
c:      c3                ret
d:      8d 76 00          lea     0x0(%esi),%esi
```

Método alternativo de análise do código binário executável (exemplo)

Entrar primeiro no depurador `gdb`: `gdb p` e...

- examinar apenas alguns bytes: `x/13b sum`

```
0x401040<sum>: 0x55 0x89 0xe5 0x8b 0x45 0x0c 0x03 0x45
0x401040<sum+8>: 0x08 0x89 0xec 0x5d 0xc3
```

... OU

- proceder à desmontagem do código: `disassemble sum`

```
0x401040 <sum>:      push    %ebp
0x401041 <sum+1>:     mov     %esp,%ebp
0x401043 <sum+3>:     mov     0xc(%ebp),%eax
0x401046 <sum+6>:     add     0x8(%ebp),%eax
0x401049 <sum+9>:     mov     %ebp,%esp
0x40104b <sum+11>:    pop     %ebp
0x40104c <sum+12>:    ret
0x40104d <sum+13>:    lea     0x0(%esi),%esi
```

Qualquer ficheiro que possa ser interpretado como código executável
– o *disassembler* examina os *bytes* e reconstrói a fonte *assembly*

```
% objdump -d WINWORD.EXE

WINWORD.EXE:      file format pei-i386

No symbols in "WINWORD.EXE".
Disassembly of section .text:

30001000 <.text>:
30001000:  55                push    %ebp
30001001:  8b ec            mov     %esp,%ebp
30001003:  6a ff            push    $0xffffffff
30001005:  68 90 10 00 30   push    $0x30001090
3000100a:  68 91 dc 4c 30   push    $0x304cdc91
```

Estrutura do tema ISA do IA32

1. Desenvolvimento de programas no IA32 em Linux
2. Acesso a operandos e operações
3. Suporte a estruturas de controlo
4. Suporte à invocação/retorno de funções
5. Acesso e manipulação de dados estruturados
6. Análise comparativa: IA-32 (CISC) e MIPS (RISC)

Acesso a operandos no IA32: sua localização e modos de acesso

Localização de operandos no IA32

- valores de constantes (ou valores imediatos)
 - incluídos na instrução, i.e., no Reg. Instrução
- variáveis escalares
 - sempre que possível, em registos (inteiros/apont) / *fp* ; se não...
 - na memória
- variáveis estruturadas
 - sempre na memória, em células contíguas

Modos de acesso a operandos no IA32

- em instruções de transferência de informação
 - instrução mais comum: `movx`, sendo *x* o tamanho (*b*, *w*, *l*)
 - algumas instruções actualizam apontadores (por ex.: `push`, `pop`)
- em operações aritméticas/lógicas

Análise de uma instrução de transferência de informação

• Transferência simples

`movl Source, Dest`

- move uma *word* de 4 bytes ("long")
- instrução mais comum em código de IA32

• Tipos de operandos

- imediato: valor constante do tipo inteiro
 - como a constante *C*, mas com prefixo '\$'
 - ex.: `$0x400`, `$-533`
 - codificado com 1, 2, ou 4 bytes
- em registo: um de 8 registos inteiros
 - mas... `%esp` and `%ebp` reservados...
 - outros poderão ser usados implicitamente...
- em memória: 4 bytes consecutivos de memória
 - vários modos de especificar o endereço...

<code>%eax</code>
<code>%edx</code>
<code>%ecx</code>
<code>%ebx</code>
<code>%esi</code>
<code>%edi</code>
<code>%esp</code>
<code>%ebp</code>



	Fonte	Destino	Equivalente em C
movl	Imm	Reg	<code>movl \$0x4,%eax temp = 0x4;</code>
		Mem	<code>movl \$-147, (%eax) *p = -147;</code>
	Reg	Reg	<code>movl %eax,%edx temp2 = temp1;</code>
		Mem	<code>movl %eax, (%edx) *p = temp;</code>
	Mem	Reg	<code>movl (%eax), %edx temp = *p;</code>
		Mem	não é possível no IA32 efectuar transferências memória-memória numa só instrução



- **Indirecto (normal)** **(R)** **Mem[Reg[R]]**
 - registo R especifica o endereço de memória`movl (%ecx), %eax`
- **Deslocamento** **D(R)** **Mem[Reg[R]+D]**
 - registo R especifica início da região de memória
 - deslocamento constante D especifica distância do início`movl 8(%ebp), %edx`

Exemplo de utilização de modos simples
de endereçamento à memória no IA32 (1)



```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
swap:
    pushl %ebp
    movl %esp, %ebp
    pushl %ebx
    movl 12(%ebp), %ecx
    movl 8(%ebp), %edx
    movl (%ecx), %eax
    movl (%edx), %ebx
    movl %eax, (%edx)
    movl %ebx, (%ecx)
    movl -4(%ebp), %ebx
    movl %ebp, %esp
    popl %ebp
    ret
```

Arranque

Corpo

Término

Exemplo de utilização de modos simples
de endereçamento à memória no IA32 (2)

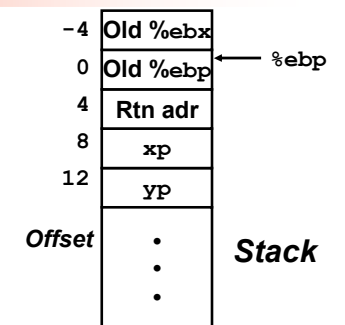


```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

Registo	Variável
%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

Corpo

```
movl 12(%ebp), %ecx    # ecx = yp
movl 8(%ebp), %edx    # edx = xp
movl (%ecx), %eax    # eax = *yp (t1)
movl (%edx), %ebx    # ebx = *xp (t0)
movl %eax, (%edx)    # *xp = eax
movl %ebx, (%ecx)    # *yp = ebx
```



Exemplo de utilização de modos simples de endereçamento à memória no IA32 (3)

	Offset	Endereço
%eax		
%edx	-4	0x100
%ecx	%ebp → 0	0x104
%ebx	4	Rtn adr 0x108
%esi	xp 8	0x124 0x10c
%edi	yp 12	0x120 0x110
%esp		0x114
%ebp		0x104

Corpo

```

movl 12(%ebp), %ecx # ecx = yp
movl 8(%ebp), %edx  # edx = xp
movl (%ecx), %eax   # eax = *yp (t1)
movl (%edx), %ebx   # ebx = *xp (t0)
movl %eax, (%edx)   # *xp = eax
movl %ebx, (%ecx)   # *yp = ebx
  
```

Exemplo de utilização de modos simples de endereçamento à memória no IA32 (4)

	Offset	Endereço
%eax		
%edx	-4	0x100
%ecx	%ebp → 0	0x104
%ebx	4	Rtn adr 0x108
%esi	xp 8	0x124 0x10c
%edi	yp 12	0x120 0x110
%esp		0x114
%ebp		0x104

Corpo

```

movl 12(%ebp), %ecx # ecx = yp
movl 8(%ebp), %edx  # edx = xp
movl (%ecx), %eax   # eax = *yp (t1)
movl (%edx), %ebx   # ebx = *xp (t0)
movl %eax, (%edx)   # *xp = eax
movl %ebx, (%ecx)   # *yp = ebx
  
```

Exemplo de utilização de modos simples de endereçamento à memória no IA32 (4)

	Offset	Endereço
%eax		
%edx	-4	0x100
%ecx	%ebp → 0	0x104
%ebx	4	Rtn adr 0x108
%esi	xp 8	0x124 0x10c
%edi	yp 12	0x120 0x110
%esp		0x114
%ebp		0x104

Corpo

```

movl 12(%ebp), %ecx # ecx = yp
movl 8(%ebp), %edx  # edx = xp
movl (%ecx), %eax   # eax = *yp (t1)
movl (%edx), %ebx   # ebx = *xp (t0)
movl %eax, (%edx)   # *xp = eax
movl %ebx, (%ecx)   # *yp = ebx
  
```

Exemplo de utilização de modos simples de endereçamento à memória no IA32 (5)

	Offset	Endereço
%eax		456
%edx	-4	0x100
%ecx	%ebp → 0	0x104
%ebx	4	Rtn adr 0x108
%esi	xp 8	0x124 0x10c
%edi	yp 12	0x120 0x110
%esp		0x114
%ebp		0x104

Corpo

```

movl 12(%ebp), %ecx # ecx = yp
movl 8(%ebp), %edx  # edx = xp
movl (%ecx), %eax   # eax = *yp (t1)
movl (%edx), %ebx   # ebx = *xp (t0)
movl %eax, (%edx)   # *xp = eax
movl %ebx, (%ecx)   # *yp = ebx
  
```

Exemplo de utilização de modos simples de endereçamento à memória no IA32 (6)

	Offset	Endereço
%eax 456		
%edx 0x124	-4	0x100
%ecx 0x120	%ebp → 0	0x104
%ebx 123	4	Rtn adr 0x108
%esi	xp 8	0x124 0x10c
%edi	yp 12	0x120 0x110
%esp		0x114
%ebp 0x104		0x118
		0x11c
		456 0x120
		123 0x124

Corpo

```

movl 12(%ebp), %ecx # ecx = yp
movl 8(%ebp), %edx # edx = xp
movl (%ecx), %eax # eax = *yp (t1)
movl (%edx), %ebx # ebx = *xp (t0)
movl %eax, (%edx) # *xp = eax
movl %ebx, (%ecx) # *yp = ebx

```

Exemplo de utilização de modos simples de endereçamento à memória no IA32 (7)

	Offset	Endereço
%eax 456		
%edx 0x124	-4	0x100
%ecx 0x120	%ebp → 0	0x104
%ebx 123	4	Rtn adr 0x108
%esi	xp 8	0x124 0x10c
%edi	yp 12	0x120 0x110
%esp		0x114
%ebp 0x104		0x118
		0x11c
		456 0x120
		456 0x124

Corpo

```

movl 12(%ebp), %ecx # ecx = yp
movl 8(%ebp), %edx # edx = xp
movl (%ecx), %eax # eax = *yp (t1)
movl (%edx), %ebx # ebx = *xp (t0)
movl %eax, (%edx) # *xp = eax
movl %ebx, (%ecx) # *yp = ebx

```

Exemplo de utilização de modos simples de endereçamento à memória no IA32 (8)

	Offset	Endereço
%eax 456		
%edx 0x124	-4	0x100
%ecx 0x120	%ebp → 0	0x104
%ebx 123	4	Rtn adr 0x108
%esi	xp 8	0x124 0x10c
%edi	yp 12	0x120 0x110
%esp		0x114
%ebp 0x104		0x118
		0x11c
		123 0x120
		456 0x124

Corpo

```

movl 12(%ebp), %ecx # ecx = yp
movl 8(%ebp), %edx # edx = xp
movl (%ecx), %eax # eax = *yp (t1)
movl (%edx), %ebx # ebx = *xp (t0)
movl %eax, (%edx) # *xp = eax
movl %ebx, (%ecx) # *yp = ebx

```

Modos de endereçamento à memória no IA32 (2)

- Indirecto (R) $\text{Mem}[\text{Reg}[\text{R}]] \dots$
 - Deslocamento $\text{D}(\text{R}) \quad \text{Mem}[\text{Reg}[\text{R}] + \text{D}] \dots$
 - Indexado $\text{D}(\text{Rb}, \text{Ri}, \text{S}) \quad \text{Mem}[\text{Reg}[\text{Rb}] + \text{S} * \text{Reg}[\text{Ri}] + \text{D}]$
 - D: Deslocamento constante de 1, 2, ou 4 bytes
 - Rb: Registo Base: quaisquer dos 8 Reg Int
 - Ri: Registo Indexação: qualquer, excepto %esp
 - S: Scale: 1, 2, 4, ou 8
- Casos particulares:**
- | | |
|-------------|--|
| (Rb, Ri) | $\text{Mem}[\text{Reg}[\text{Rb}] + \text{Reg}[\text{Ri}]]$ |
| D(Rb, Ri) | $\text{Mem}[\text{Reg}[\text{Rb}] + \text{Reg}[\text{Ri}] + \text{D}]$ |
| (Rb, Ri, S) | $\text{Mem}[\text{Reg}[\text{Rb}] + \text{S} * \text{Reg}[\text{Ri}]]$ |

Exemplo de instrução do IA32 apenas
para cálculo do endereço efectivo do operando (1)



leal Src, Dest

- **Src** contém a expressão para cálculo do endereço
- **Dest** vai receber o resultado do cálculo da expressão
- **Tipos de utilização desta instrução:**
 - cálculo de um endereço sem acesso à memória
 - Ex.: tradução de `p = &x[i];`
 - cálculo de expressões aritméticas do tipo

$$x + k \cdot y \quad \text{para } k = 1, 2, 4, \text{ or } 8$$
- **Exemplo ...**

Exemplo de instrução do IA32 apenas
para cálculo do endereço efectivo do operando (2)



leal Source, %eax

%edx	0xf000
%ecx	0x100

Source	Expressão	-> %eax
0x8(%edx)	0xf000 + 0x8	0xf008
(%edx, %ecx)	0xf000 + 0x100	0xf100
(%edx, %ecx, 4)	0xf000 + 4*0x100	0xf400
0x80(, %edx, 2)	2*0xf000 + 0x80	0x1e080

Instruções de transferência de informação no IA32



movx S, D $D \leftarrow S$ Move (byte, word, long-word)
movsbl S, D $D \leftarrow \text{SignExtend}(S)$ Move Sign-Extended Byte
movzbl S, D $D \leftarrow \text{ZeroExtend}(S)$ Move Zero-Extended Byte

push S $\%esp \leftarrow \%esp - 4; \text{Mem}[\%esp] \leftarrow S$ Push
pop D $D \leftarrow \text{Mem}[\%esp]; \%esp \leftarrow \%esp + 4$ Pop

leal S, D $D \leftarrow \&S$ Load Effective Address

D – destino [Reg | Mem] **S** – fonte [Imm | Reg | Mem]
D e S não podem ser ambos operandos em memória

Operações aritméticas e lógicas no IA32



inc D $D \leftarrow D + 1$ Increment
dec D $D \leftarrow D - 1$ Decrement
neg D $D \leftarrow -D$ Negate
not D $D \leftarrow \sim D$ Complement

add S, D $D \leftarrow D + S$ Add
sub S, D $D \leftarrow D - S$ Subtract
imul S, D $D \leftarrow D * S$ 32 bit Multiply

and S, D $D \leftarrow D \& S$ And
or S, D $D \leftarrow D | S$ Or
xor S, D $D \leftarrow D \wedge S$ Exclusive-Or

shl k, D $D \leftarrow D \ll k$ Left Shift
sar k, D $D \leftarrow D \gg k$ Arithmetic Right Shift
shr k, D $D \leftarrow D \gg k$ Logical Right Shift