

A arquitectura IA32

A arquitectura de um processador é caracterizada pelo conjunto de atributos que são visíveis ao programador.

- Tamanho da palavra
- Número de registos visíveis
- Número de operandos
- Endereçamento de operandos
- O conjunto de instruções

IA32 – O tamanho da palavra

Este é um parâmetro fundamental do sistema que determina, em *bits*:

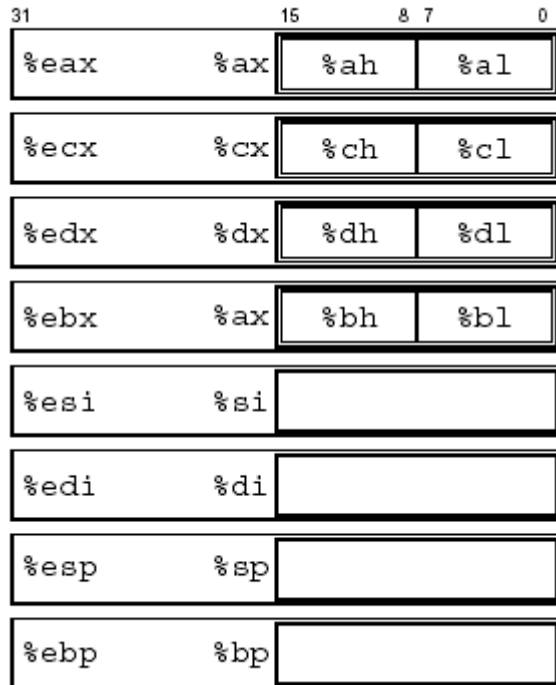
- o tamanho, por defeito, dos números inteiros
- o tamanho dos endereços
- o tamanho dos registos de uso genérico
- o tamanho, por defeito, do bloco de dados que é lido da memória

O IA32 tem uma palavra de 32 *bits*, mas, por razões históricas, são as quantidades de 16 *bits* que são referenciadas como palavras (*w*), sendo as quantidades de 32 *bits* referenciadas como palavras longas (*l*).

32 *bits* permitem endereçar 2^{32} *bytes* = 4 Gbytes

IA32 – Registos visíveis

8 registos genéricos de 32 *bits* acessíveis em quantidades de 8 e 16 *bits*



%ax, %bx, %cx, %dx – 16 bits menos significativos dos respectivos registos

%ah, %bh, %ch, %dh – 2º byte menos significativo dos respectivos registos

%al, %bl, %cl, %dl – byte menos significativo dos respectivos registos

Apesar de genéricos, alguns destes registos são implicitamente usados por algumas instruções: %eax, %esp.

Número de operandos

$a = b + c$; uma operação aritmética binária tem 3 operandos.

O conjunto de instruções de uma máquina pode suportar um número diferente de operandos:

0 operandos – os operandos estão no topo da stack e o resultado é posto no topo da mesma; **EX:** `add => push (pop+pop)`

1 operando – um dos operandos e o destino da operação é um registo designado por acumulador; **EX:** `add b => acc = acc + b`

2 operandos – um dos operandos funciona também como destino da operação; **EX:** `add b, c => c = c + b`

3 operandos – todos os operandos são especificados na instrução; **EX:** `add b, c, d => b = c + d`

IA32 – 2 operandos

```
addl %eax, %ebx      ; %ebx = %ebx+%eax
movl %eax, %ebx      ; %ebx = %eax
subl %eax, %ebx      ; %ebx = %ebx-%eax
imull %esi, %esp     ; %esp = %esp * %esi (32 bits)
```

mas também existem instruções de 1 operando:

```
incl %ecx             ; %ecx = %ecx+1
imull %ecx            ; %edx:%eax = %eax * %ecx (64 bits)
```

IA32 – Modos de endereçamento

Como indicar numa instrução quais os operandos?

Registos

O valor do operando é o valor armazenado no registo.

```
addl %eax, %esi
```

```
%esi = %esi + %eax
```

Constante

(modo imediato)

O valor do operando é uma constante especificada na instrução.

```
addl $5, %esi
```

```
%esi = %esi + 5
```

IA32 – Modos de endereçamento

Um, e apenas um, dos operandos pode estar armazenado em memória.

Como especificar o endereço de memória?

O endereço é o resultado da avaliação da expressão:

$$\text{Endereço} = \text{Base} + \text{Índice} * \text{Escala} + \text{Deslocamento}$$

onde:

- Base e Índice são registos
- Escala é uma constante com o valor 1, 2, 4 ou 8
- Deslocamento é uma constante com 32 *bits* no máximo

NOTA: Qualquer um destes campos pode estar omissos!

IA32 – Modos de endereçamento

```
addl %eax, 1000(%ebp, %esi, 4)
```

Significado: somar a %eax a palavra (4 bytes) cujo endereço começa em %ebp + %esi*4 + 1000. Resultado guardado em memória.

```
andl (,%eax, 8), %edx
```

Significado: AND da palavra (4 bytes) cujo endereço começa em %eax*8 com o conteúdo de %edx. Onde fica o resultado?

```
subw %ax, 0x70AA
```

Significado: somar à palavra (2 bytes) cujo endereço começa em 0x70AA o valor de %ax. Resultado guardado em memória.

IA32 – Modos de endereçamento

Type	Form	Operand value	Name
Immediate	$\$Imm$	Imm	Immediate
Register	E_a	$R[E_a]$	Register
Memory	Imm	$M[Imm]$	Absolute
Memory	(E_a)	$M[R[E_a]]$	Indirect
Memory	$Imm (E_b)$	$M[Imm + R[E_b]]$	Base + displacement
Memory	(E_b, E_i)	$M[R[E_b] + R[E_i]]$	Indexed
Memory	$Imm (E_b, E_i)$	$M[Imm + R[E_b] + R[E_i]]$	Indexed
Memory	$(, E_i, s)$	$M[R[E_i] \cdot s]$	Scaled indexed
Memory	$Imm (, E_i, s)$	$M[Imm + R[E_i] \cdot s]$	Scaled Indexed
Memory	(E_b, E_i, s)	$M[R[E_b] + R[E_i] \cdot s]$	Scaled indexed
Memory	$Imm (E_b, E_i, s)$	$M[Imm + R[E_b] + R[E_i] \cdot s]$	Scaled indexed

IA32 – Modos de endereçamento

Memória	
Endereço	Valor
0x100	0xFF
0x104	0xAB
0x108	0X13
0x10C	0x11

Registo	Valor
%eax	0x100
%ecx	1
%edx	3

Operando	Valor
%eax	0x100
(%eax)	0xFF
0x104	0xAB
\$0x104	0x104
4(%eax)	0xAB
(%eax,%ecx,8)	0x13
0x100(%ecx,%edx)	0xAB
0x104(,%ecx,8)	0x11

IA32 – O conjunto de instruções: operações lógico-aritméticas

2 operandos

```
add? src, dest  
sub? src, dest  
and? src, dest  
imull? src, dest  
sar? shamt5, dest
```

onde

? – b, w, l

shamt5 ∈ [0..31]

1 operando

```
inc? dest  
dec? dest  
imull src  
    %edx:%eax = %eax * src  
idivl src  
    %edx = %eax mod src  
    %eax = %eax / src
```

onde

? – b, w, l

IA32 – O conjunto de instruções:

operações de transferência de informação

```
mov? src, dest  
lea? Imm(R1, R2, s), dest
```

onde
? – b, w, l

Stack

```
pushl src  
popl dest
```

IA32 – A stack

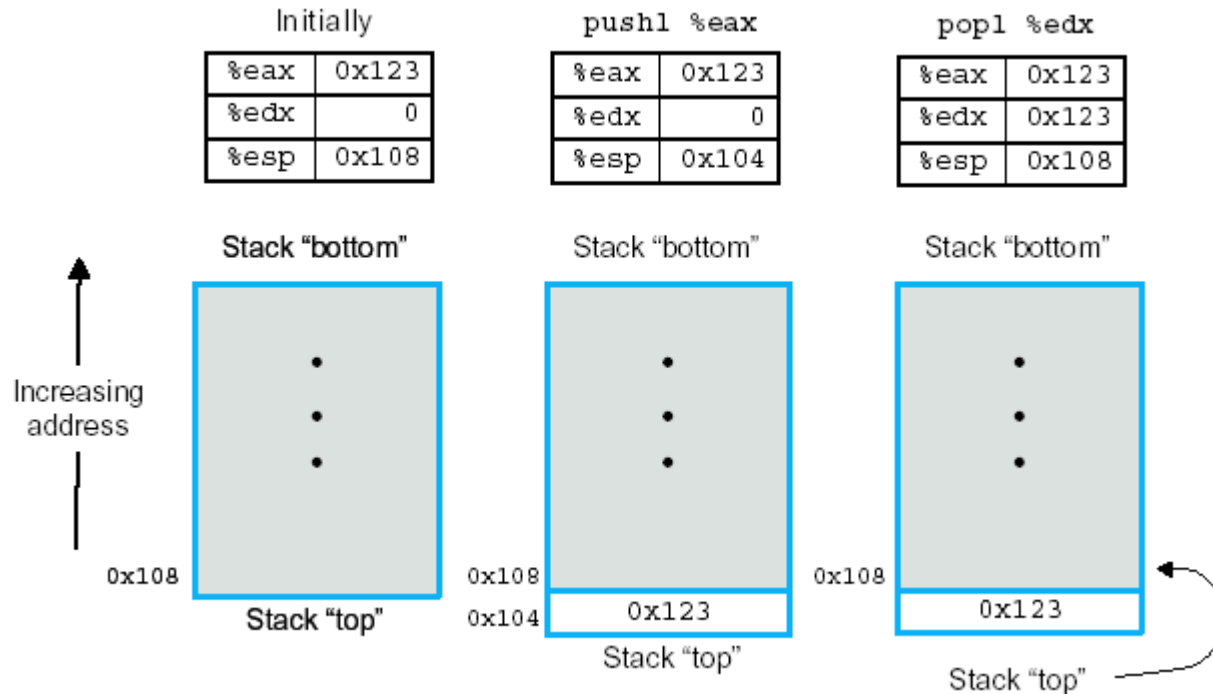
Estrutura de dados mantida em memória, do tipo LIFO.

O *stack pointer* (`%esp`) aponta para o topo da stack.

Esta cresce de endereços mais altos para endereços mais baixos.

`pushl` subtrai 4 ao `%esp` e carrega uma palavra na memória.

`popl` lê uma palavra da stack e soma 4 ao `%esp`.



IA32 – O conjunto de instruções:

Flags e operações de teste

4 flags: variam conforme o resultado da última operação

ZF = 1 se zero

SF = 1 se < zero

CF = 1 se transporte

OF = 1 se overflow

```
cmp? src1, src2
```

```
# src2 - src1
```

```
# resultado não é guardado
```

```
test? src1, src2
```

```
# src2 AND src1
```

```
# resultado não é guardado
```

IA32 – O conjunto de instruções:

Operações de controlo de fluxo

Salto Incondicionais

```
jmp label  
jmp *src
```

Salto Condicionais (testam as flags)

```
je label  
js label  
jg label  
jl label  
jne label  
jns label  
jge label  
jle label
```

IA32 – O Conjuntos de instruções:

Invocação de procedimentos

```
call label
call *Op
# coloca endereço de retorno na stack

ret
# lê endereço de retorno da stack

leave
# realiza algumas operações relacionadas com o acesso a
# variáveis locais e parâmetros
```


Sumário

Tema	Hennessy [COD]	Bryant [CS:APP]
IA32		Sec 3.4 a 3.6.3