

Mestr. Integr. Eng.^a Informática

1º ano

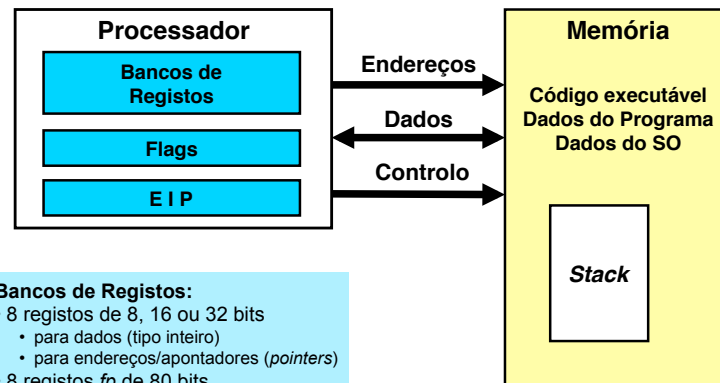
2018/19

A.J.Proença

Tema
ISA do IA-32

Estrutura do tema ISA do IA-32

1. Desenvolvimento de programas no IA-32 em Linux
2. Acesso a operandos e operações
3. Suporte a estruturas de controlo
4. Suporte à invocação/regresso de funções
5. Análise comparativa: IA-32, x86-64 e MIPS (RISC)
6. Acesso e manipulação de dados estruturados

O modelo Processador-Mem no IA-32
(visão do programador)

Bancos de Registos:

- 8 registos de 8, 16 ou 32 bits
 - para dados (tipo inteiro)
 - para endereços/apontadores (*pointers*)
- 8 registos *fp* de 80 bits

Flags:

- estado da última op aritm/lóg

O banco de registos para
inteiros / apontadores

Integer Registers (IA32)

				Origin (mostly obsolete)
general purpose	%eax	%ax	%ah %al	accumulate
	%ecx	%cx	%ch %cl	counter
	%edx	%dx	%dh %dl	data
	%ebx	%bx	%bh %bl	base
	%esi	%si		source index
	%edi	%di		destination index
	%esp	%sp		stack pointer
	%ebp	%bp		base pointer
16-bit virtual registers (backwards compatibility)				

Tamanhos de objetos em C (em bytes)

Declaração em C	Designação Intel	Tamanho IA-32
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 IA-32 é um processador *little endian*

Exemplo:

valor de `var` (0x01234567) na memória, cujo endereço &`var` é 0x100

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

Operações primitivas:

– Efetuar operações aritméticas/lógicas

com dados em registo ou em memória

- dados do tipo *integer* de 1, 2 ou 4 bytes; em complemento p/ 2
- dados em formato *fp* de 4, 8 ou 10 bytes; precisão simples ou dupla
- operações só com dados escalares; op's com vetores possível
- arrays* ou *structures*; bytes continuamente alocados em memória

– Transferir dados entre células de memória e um registo

- carregar (*load*) em registo dados copiados da memória
- armazenar (*store*) na memória valores guardados em registo

– Transferir o controlo da execução das instruções

- saltos incondicionais para outras partes do programa/módulo
- saltos incondicionais para/de 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 -O2 p1.c p2.c -o p`

- usa otimizações (O2)
- coloca binário resultante no ficheiro p (o)

fich. texto

Programa C (p1.c p2.c)

Compilador (gcc -S)

fich. texto

Programa Asm (p1.s p2.s)

Assembler (gcc -c ou as)

fich. binário

Programa objeto (p1.o p2.o)

Bibliotecas estáticas (.a)

Linker (gcc ou ld)

fich. binário

Programa executável (p)

A compilação de C para assembly (exemplo)

Código C

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

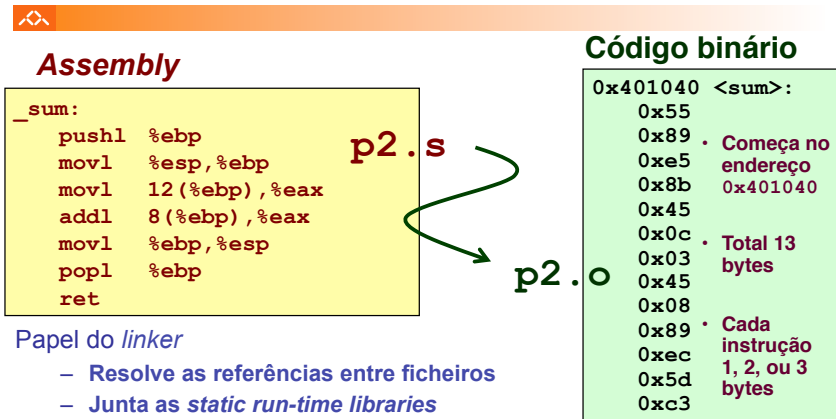
Assembly gerado

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

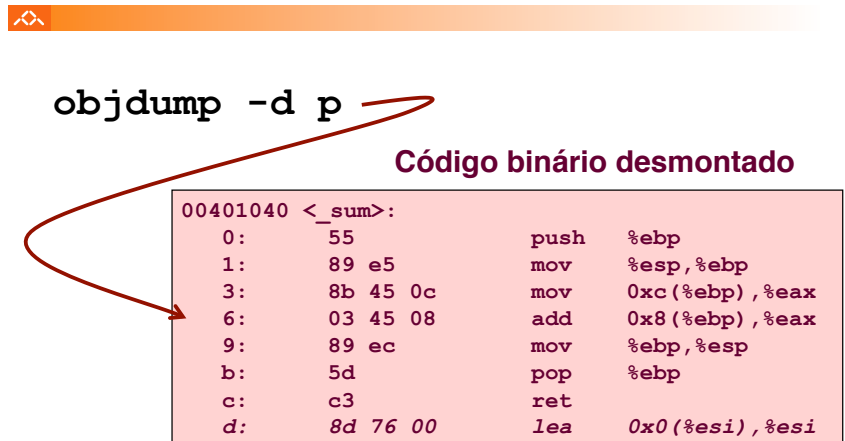
`gcc -O2 -S p2.c`

`p2.s`

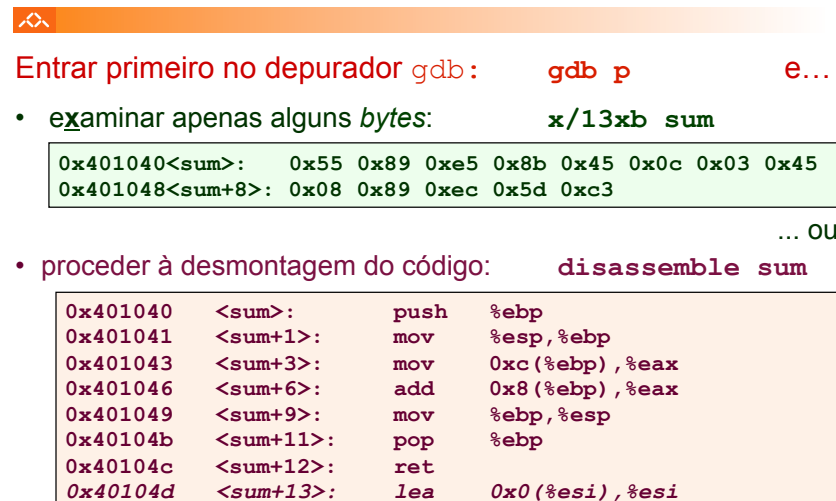
De assembly para objeto e executável (exemplo)



Desmontagem de código binário executável (exemplo)



Método alternativo de análise do código binário executável (exemplo)



Que código pode ser desmontado?

Qualquer ficheiro que possa ser interpretado como código executável

- o *disassembler* examina os *bytes* e reconstrói o código em *assembly*

