Mestr. Integr. Eng.ª Informática

1º ano

2017/18

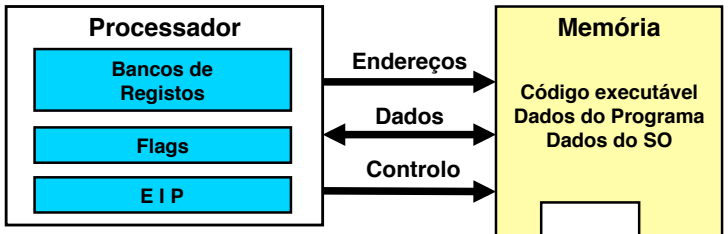
A.J.Proença

Tema

ISA do IA-32

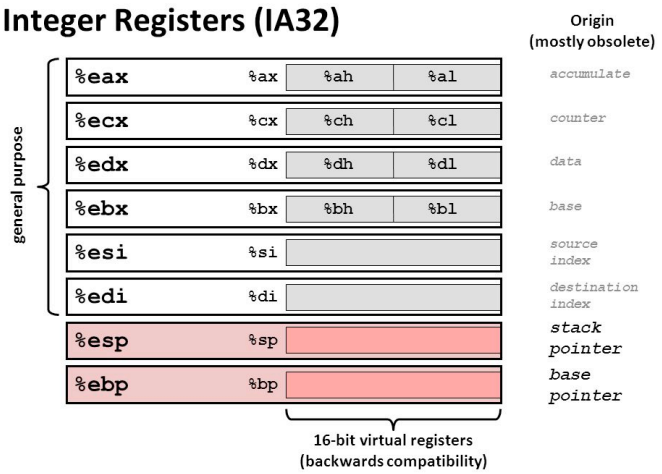
Estrutura do tema ISA do IA-32

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



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

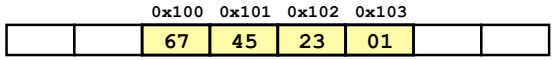
Flags:
- estado da última op aritm/lóg



- 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

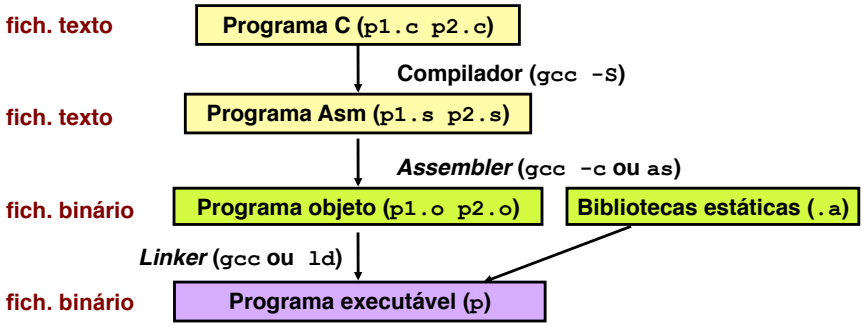
- Ordenação dos *bytes* na memória
  - **O IA-32 é um processador *little endian***
  - **Exemplo:**
    valor de `var` (`0x01234567`) na memória, cujo endereço `&var` é `0x100`



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

- 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`)



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)

Assembly

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

p2.s → p2.o

Código binário

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

- Começa no endereço 0x401040
- Total 13 bytes
- Cada 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…

- e**x**aminar apenas alguns *bytes*: **`x/13xb sum`**

```
0x401040<sum>:   0x55 0x89 0xe5 0x8b 0x45 0x0c 0x03 0x45
0x401048<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
```


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*

```
% 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
```