



## 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)



- **Por omissão, as instruções são sempre executadas sequencialmente, i.e., uma após outra** (em HLL & em ling. máq.)
- **Em HLL o fluxo de instruções poderá ser alterado:**
  - na execução de estruturas de controlo (**adiante...**)
  - na invocação / retorno de funções (**mais adiante...**)
  - na ocorrência de excepções / interrupções (**mais adiante?**)
- **Em linguagem máquina isso traduz-se na alteração do IP, de modo incondic/condicional, por um valor absoluto/relativo**
  - **jump / branch**
  - **call** (com salvaguarda do endereço de regresso) e **ret**
  - em excepções / interrupções . . .

## Codificação das condições no IA32 para utilização posterior



### • Condições codificadas em registos de 1 bit -> **Flag**

CF *Carry Flag*      SF *Sign Flag*  
ZF *Zero Flag*      OF *Overflow Flag*

### • As **Flags** podem ser implícita / explicitamente alteradas:

– Implícitamente, por operações aritméticas/lógicas

`addl Src, Dest`      **Equivalente em C:** `t = a + b`  
**Flags afectadas:**    CF ZF SF OF

– Explicitamente, por instruções de comparação e teste

`cmpl Src2, Src1`      **Equivalente em C...** apenas calcula `Src1-Src2`  
**Flags afectadas:**    CF ZF SF OF

`testl Src2, Src1`      **Equivalente em C...** apenas calcula `Src1&Src2`  
**Flags afectadas:**    CF ZF SF OF

## Utilização das **Flags** no IA32



### A informação das **Flags** pode ser:

- Colocada directamente num de 8 registos de 8 bits;      **ou...**  
`setcc Dest      Dest %al %ah %dl %dh %ch %cl %bh %bl`  
**Nota:** não altera restantes 3 bytes; usada com `movzbl`
- Usada numa instrução de salto condicional:  
`jcc Label      Label endereço destino ou distância para destino`

### Interpretação das **Flags**:

| (set/j) cc | Descrição           | Flags      |
|------------|---------------------|------------|
| (set/j) e  | <b>Equal</b>        | <b>ZF</b>  |
| (set/j) ne | <b>Not Equal</b>    | <b>~ZF</b> |
| (set/j) s  | <b>Sign (-)</b>     | <b>SF</b>  |
| (set/j) ns | <b>Not Sign (-)</b> | <b>~SF</b> |

|            |               |              |
|------------|---------------|--------------|
| (set/j) g  | > (c/ sinal)  | ~(SF^OF)&~ZF |
| (set/j) ge | >= (c/ sinal) | ~(SF^OF)     |
| (set/j) l  | < (c/ sinal)  | (SF^OF)      |
| (set/j) le | <= (c/ sinal) | (SF^OF) ZF   |
| (set/j) a  | > (s/ sinal)  | ~CF&~ZF      |
| (set/j) b  | < (s/ sinal)  | CF           |

• Estruturas de controlo do C

- if-else statement
- do-while statement
- while statement
- for loop
- switch statement

Análise de um exemplo

```
int absdiff(int x, int y)
{
    if (x < y)
        return y - x;
    else
        return x - y;
}
```

C original

```

movl 8(%ebp),%edx
movl 12(%ebp),%eax
cmpl %eax,%edx
jl .L3
subl %eax,%edx
movl %edx,%eax
jmp .L5
.L3:
    subl %edx,%eax
.L5:
```

Corpo

```
int goto_diff(int x, int y)
{
    int rval;
    if (x < y)
        goto then_statement;
    rval = x - y;
    goto done;
then_statement:
    rval = y - x;
done:
    return rval;
}
```

Versão goto

```

# edx = x
# eax = y
# compare x : y
# if <, goto then_statement
# edx = x - y
# return value = edx
# goto done
# then_statement:
# return value = y - x
# done:
```

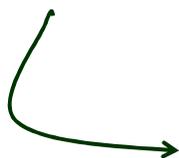
Generalização

```
if (expressão_de_teste)
    then_statement
else
    else_statement
```

Forma genérica em C

```
cond = expressão_de_teste
if (cond)
    goto true;
else_statement
goto done;
true:
    then_statement
done:
```

Versão com goto, ou assembly com sintaxe C



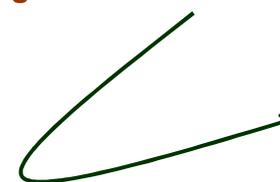
Generalização

```
do
    body_statement
while (expressão_de_teste);
```

Forma genérica em C

```
loop:
    body_statement
    cond = expressão_de_teste
    if (cond)
        goto loop;
```

Versão com goto, ou assembly com sintaxe C



**Análise de um exemplo**  
 - série de Fibonacci:

$$F_1 = F_2 = 1$$

$$F_n = F_{n-1} + F_{n-2}, n \geq 3$$

```
int fib_dw(int n)
{
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

**C original**

```
int fib_dw_goto(int n)
{
    int i = 0;
    int val = 0;
    int nval = 1;

loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;
    return val;
}
```

**Versão com goto**

**Análise de um exemplo**  
 - série de Fibonacci

| Utilização dos registos |          |               |
|-------------------------|----------|---------------|
| Registo                 | Variável | Valor inicial |
| %ecx                    | i        | 0             |
| %esi                    | n        | n             |
| %ebx                    | val      | 0             |
| %edx                    | nval     | 1             |
| %eax                    | t        | 1             |

Corpo (loop)

```
.L2:
    leal (%edx,%ebx),%eax
    movl %edx,%ebx
    movl %eax,%edx
    incl %ecx
    cmpl %esi,%ecx
    jl .L2
    movl %ebx,%eax
```

```
int fib_dw_goto(int n)
{
    int i = 0;
    int val = 0;
    int nval = 1;

loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;
    return val;
}
```

**Versão goto**

```
# loop:
# t = val + nval
# val = nval
# nval = t
# i++
# compare i : n
# if <, goto loop
# return val
```

**Generalização**

```
while (expressão_de_teste)
    body_statement
```

Forma genérica em C

```
loop:
    cond = expressão_de_teste
    if (! cond)
        goto done;
    body_statement
    goto loop;
done:
```

**Versão com goto**

```
if (! expressão_de_teste)
    goto done;
do
    body_statement
while (expressão_de_teste);
done:
```

Conversão while em do-while

```
cond = expressão_de_teste
if (! cond)
    goto done;
loop:
    body_statement
    cond = expressão_de_teste
    if (cond)
        goto loop;
done:
```

**Versão do-while com goto**

**Análise de um exemplo**  
 - série de Fibonacci

```
int fib_w(int n)
{
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

**C original**

```
int fib_w_goto(int n)
{
    int i = 1;
    int val = 1;
    int nval = 1;

    if (i >= n);
        goto done;

loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i < n);
        goto loop;
done:
    return val;
}
```

**Versão do-while com goto**

### while statement (3)

#### Análise de um exemplo – série de Fibonacci

| Utilização dos registos |          |               |
|-------------------------|----------|---------------|
| Registo                 | Variável | Valor inicial |
| %esi                    | n        | n             |
| %ecx                    | i        | 1             |
| %ebx                    | val      | 1             |
| %edx                    | nval     | 1             |
| %eax                    | t        | 2             |

```
int fib_w_goto(int n)
{
    (...)
    if (i>=n);
        goto done;

loop:
    (...)
    if (i<n);
        goto loop;
done:
    return val;
}
```

Versão do-while com goto

Corpo {

```
(...)
    cmpl %esi,%ecx
    jge .L7
.L5:
    (...)
    cmpl %esi,%ecx
    jl .L5
.L7:
    movl %ebx,%eax
# esi=n, i=val=nval=1
# compare i : n
# if >=, goto done
# loop:
# compare i : n
# if <, goto loop
# done:
# return val
```

Nota: Código gerado com gcc -O1 -S

### for loop (1)

#### Generalização

```
for (expr_inic; expr_test; act_expr)
    body_statement
```

Forma genérica em C

```
expr_inic;
while (expr_test) {
    body_statement
    act_expr ;
}
```

Conversão for em while

```
expr_inic ;
if (! expr_test)
    goto done;
do {
    body_statement
    act_expr ;
} while (expr_test) ;
done:
```

Conversão para do-while

```
expr_inic ;
cond = expr_test ;
if (! cond)
    goto done;
loop:
    body_statement
    act_expr ;
cond = expr_test ;
if (cond)
    goto loop;
done:
```

Versão do-while com goto

### for loop (2)

#### Análise de um exemplo – série de Fibonacci

```
int fib_f(int n)
{
    int i;
    int val = 1;
    int nval = 1;

    for (i=1; i<n; i++) {
        int t = val + nval;
        val = nval;
        nval = t;
    }

    return val;
}
```

C original

```
int fib_f_goto(int n)
{
    int val = 1;
    int nval = 1;

    int i = 1;
    if (i>=n);
        goto done;

loop:
    int t = val + nval;
    val = nval;
    nval = t;
    i++;
    if (i<n);
        goto loop;
done:
    return val;
}
```

Versão do-while com goto  
Nota: gcc gera mesmo código...

### switch statement

#### "Salto" com escolha múltipla; alternativas de implementação:

- Sequência de if-else statements
- Com saltos "indirectos": endereços especificados numa tabela de salto (jump table)