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 *vs*. x86-64 e RISC (MIPS e ARM)
6. Acesso e manipulação de dados estruturados

Revisão: conversão de HLL para Código máquina

- C

```
*dest = t;
```

  - guardar **`t`** na posição de memória indicada por **`dest`**

- *"Assembly"*

```
movl %eax, (%ebx)
```

  - Mover valor de 4-bytes de registo para a memória
  - Operandos:

    `t`: Registo **`%eax`**

    `dest`: Registo **`%ebx`**

    `*dest`: Memória **`M[%ebx]`**

- Código máquina

```
0x40059e: 89 03
```

```
100010 11   00 000 011
MOV r->x   Mod R    M
```

  - Instrução armazenada a partir do endereço **`0x40059e`**
  - Representação compacta da instrução (esta ocupa 2 bytes)
  - Adequado para a interpretação rápida do hardware

Revisão: Operandos em memória e LEA

- IA-32 suporta vários modos de endereçamento à memória:
  - Indirecto (R) `Mem[ Reg[R] ] ...`
  - Deslocamento D(R) `Mem[ Reg[R] + D ] ...`
  - **Indexado** **D(Rb,Ri,S)** `Mem[ Reg[Rb]+S*Reg[Ri]+D ]`
    - Exemplos:

Assembly	Equivalente em C
movl 4(%ebx,%edi,8), %eax	eax = *(ebx + edi*8 + 4)
addl 4(%ebx,%edi,8), %eax	eax += *(ebx + edi*8 + 4)

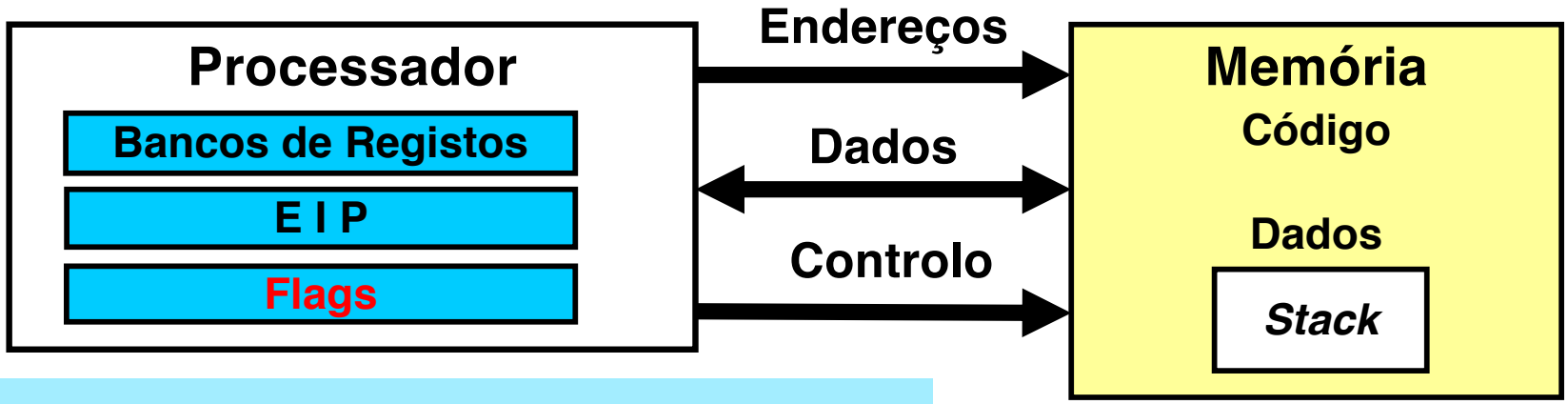
    - LEA não acede à memória

Assembly	Equivalente em C
lea 4(%ebx,%edi,8), %eax	eax = ebx + edi*8 + 4

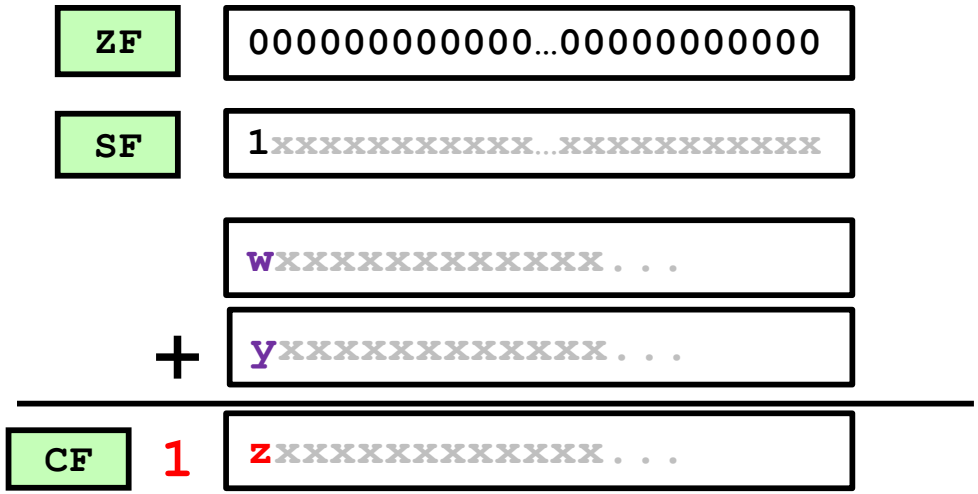
      - instrução desenhada para calcular o valor de um apontador
      - Exemplo: ptr = & (alunos[i].nota)
        - » Calcular a posição de memória onde está armazenada a nota de um aluno

- Os compiladores recentes tendem a utilizar LEA para efetuar cálculos complexos com uma só instrução: a = b + d*8 + 4;

Flags (códigos de condição)



Flags:
– bits com estado da última operação aritmética/lógica




```
exemplo:
        a += b; // addl %ebx,%eax
```

ZF: *Zero Flag* se a==0

SF: *Sign Flag* se a<0

CF: *Carry Flag* se transporte (unsigned)

OF: *Overflow Flag* se *overflow* em comp.p/2

OF w == y && w != z

Instruções de comparação e de salto

- **Por omissão, as instruções são executadas em sequência**
  - adiciona ao IP o número de bytes da instrução executada
- **Instruções de salto:**
  - permitem alterar o IP para um valor específico
  - podem ser condicionadas pelo valor das *flags*
- **Instr. de comparação**
  - `cmpl a,b` (& `test a,b`)
  - calcula (b – a) e atualiza as *flags*, mas **não altera b**
  - Usada para implementar

    if (a<b) ...

jX	Condition	Description
`jmp`	`1`	**Unconditional**
`je`	`ZF`	**Equal / Zero**
`jne`	`~ZF`	**Not Equal / Not Zero**
`js`	`SF`	**Negative**
`jns`	`~SF`	**Nonnegative**
`jg`	`~(SF^OF)&~ZF`	**Greater (Signed)**
`jge`	`~(SF^OF)`	**Greater or Equal (Signed)**
`jl`	`(SF^OF)`	**Less (Signed)**
`jle`	`(SF^OF)|ZF`	**Less or Equal (Signed)**
`ja`	`~CF&~ZF`	**Above (unsigned)**
`jb`	`CF`	**Below (unsigned)**

Estruturas de controlo em C

- **`if-else` *statement***

Estrutura geral:

```
...
  if (condição)
        expressão_1;
  else
        expressão_2;
...
```

Exemplo:

```
int absdiff(int x, int y)
{
  if (x < y)
    ret = y - x;
  else
    ret = x - y;
  ...
}
```

- **`do-while` *statement***
- **`while` *statement***
- **`for` *loop***

if-then-else statement (1)

Análise de um exemplo

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

C original

```
int goto_absdiff(int x, int y)
{
  if (x >= y) goto Else
  ret = y - x;
  goto Done
Else:
  ret = x - y;
Done:
}
```

C versão *goto*

Corpo

```
    movl  ...,%edx          # edx = x
    movl  ...,%eax          # eax = y
    cmpl  %eax,%edx         # compare x : y  (≈ x - y)
    jge    .L3              # if x - y>=0, then goto else
    subl  %edx,%eax         # compute y - x
    ...                     # return the value (y - x)
    jmp done                # goto done
.L3:                        # else:
    subl  %eax,%edx         # return the value (x - y)
   ...
```

Generalização

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

Forma genérica em C

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

Generalização alternativa (menos utilizada)

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

Forma genérica em C

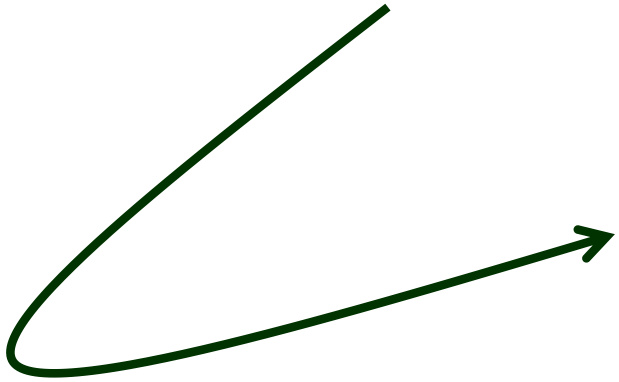
```
 cond = expressão_de_teste
 if (cond ) goto true;
 else_statement
 goto done;
true:
 then_statement
done:
```

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		
Variável	Registo	Valor inicial
n	**%esi**	**n** (argumento)
i	**%ecx**	**0**
val	**%ebx**	**0**
nval	**%edx**	**1**
t	**%eax**	**1**

```
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*

Corpo (*loop*)

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

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

Generalização pouco utilizada

```
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*

Generalização mais utilizada

```
while(expressão_de_teste)
  body_statement
```

Forma genérica em C

```
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*

Análise de um exemplo

– série de Fibonacci

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

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

Nota: Código gerado com *gcc –O1 –S*

Generalização

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

Forma genérica em C

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

Conversão for em while

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

Conversão para do-while

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

Versão do-while com *goto*

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...