

Assembly - Tradução de Mecanismos de Controle

Noemi Rodriguez

Ana Lúcia de Moura

Raúl Renteria

Alexandre Meslin

<http://www.inf.puc-rio.br/~inf1018>

Fluxo de Execução

MOVQ L1, %rdx
MOVL (%rdx),%ecx
SUBL VG, %ecx
ANDL \$1, %ecx
ADDL %eax, %ecx
ADDL (%rsi), %ecx

fetch-decode-execute
busca automaticamente a
instrução seguinte à executada
anteriormente

Registradores e Execução

Registrador RIP:
endereço da próxima instrução a ser executada

Fluxo de Execução

MOVQ L1, %rdx
MOVL (%rdx),%ecx
SUBL VG, %ecx
ANDL \$1, %ecx
ADDL %eax, %ecx
ADDL (%rsi), %ecx

fetch-decode-execute
busca automaticamente a
instrução seguinte à executada
anteriormente

... mas então vamos repetir
500 vezes um grupo de
instruções quando quisermos
tratar um array de 500
posições?

Fluxo de Execução

MOVQ L1, %rdx
MOVL (%rdx),%ecx
SUBL VG, %ecx
ANDL \$1, %ecx
ADDL %eax, %ecx
ADDL (%rsi), %ecx

fetch-decode-execute
busca automaticamente a
instrução seguinte à executada
anteriormente

instruções de desvio de
controle permitem alterar esse
fluxo!

Fluxo de Execução

1150	MOVQ L1, %rdx
1158	MOVL (%edx),%ecx
115b	SUBL VG, %ecx
1162	ANDL \$1, %ecx
1165	ADDL %eax, %ecx
1167	ADDL (%rsi), %ecx
1169	JZ 115b

instruções de desvio de controle permitem alterar esse fluxo!

Fluxo de Execução

1150	MOVQ L1, %rdx
1158	MOVL (%edx),%ecx
115b	SUBL VG, %ecx
1162	ANDL \$1, %ecx
1165	ADDL %eax, %ecx
1167	ADDL (%rsi), %ecx
1169	JZ 115b

se o **resultado da operação anterior** for zero:
DESVIA para endereço 115b

Fluxo de Execução em Assembly

- Em C:
 - a sequência de operações executada depende do resultado de testes aplicados aos dados: **execução condicional**

```
if (d < a)
    c = a - d;
else
    c = d - a;
while (a <= b) {
    ...
    a++;
}
...
```

Fluxo de Execução em Assembly

- Em C:
 - a sequência de operações executada depende do resultado de testes aplicados aos dados: **execução condicional**
- Na linguagem de máquina também, mas com mecanismos bem mais limitados:
 - testar resultado de operações aritméticas e lógicas
 - *desviar* (alterar o fluxo de controle do programa) conforme esse resultado

Registradores e Execução

Registrador RIP:
endereço da próxima instrução a ser executada

Testes sobre Dados

- O registrador RFLAGS é um conjunto de bits que descreve o resultado da última operação aritmética/lógica realizada
 - ZF, SF, CF, OF
- Combinações desses bits podem ser usadas como condições para alterar o fluxo de controle
 - decisão sobre qual a próxima instrução a ser executada

Instruções de Desvio

- Podem alterar a sequência de execução (“goto”)
- O destino do desvio é indicado no código assembly por um “label simbólico”
 - traduzido pelo assembler + linker para um endereço de memória

L1:

```
    cmpl    %ebx, %ecx
```

```
    jz      L2      → condicional
```

```
    ...
```

```
    jmp    L1      → incondicional
```

L2:

```
    xorl    %eax, %eax
```

Desvios Condicionais

Instrução	Sinônimo	Descrição
<code>je Label</code>	<code>jz</code>	equal/zero
<code>jne Label</code>	<code>jnz</code>	not equal/not zero
<code>js Label</code>		negative
<code>jns Label</code>		non negative
<code>jg Label</code>	<code>jnle</code>	> (greater)
<code>jge Label</code>	<code>jnl</code>	>= (greater or equal)
<code>jl Label</code>	<code>jnge</code>	< (less)
<code>jle Label</code>	<code>jng</code>	<= (less or equal)

Desvios Condicionais

Instrução	Sinônimo	Descrição
<code>je Label</code>	<code>jz</code>	equal/zero
<code>jne Label</code>	<code>jnz</code>	not equal/not zero
<code>js Label</code>		negative
<code>jns Label</code>		non negative
<code>jg Label</code>	<code>jnle</code>	> (greater)
<code>jge Label</code>	<code>jnl</code>	>= (greater or equal)
<code>jl Label</code>	<code>jnge</code>	< (less)
<code>jle Label</code>	<code>jng</code>	<= (less or equal)

} comparação signed!

Desvios Condicionais

Instrução	Sinônimo	Descrição
je Label	jz	equal/zero
jne Label	jnz	not equal/not zero
ja Label	jnbe	> (above)
jae Label	jnb	>= (above or equal)
jb Label	jnae	< (below)
jbe Label	jna	<= (below or equal)

} comparação UNsigned!

Mecanismos de controle: 'if'

- Suponha o código C:

```
if (a==b)
```

```
    c=d;    →    suponha a em %eax, b em %ebx,  
                c em %ecx, d em %edx
```

```
d=a+c;
```

Mecanismos de controle: 'if'

- Suponha o código C:

```
if (a==b) c=d; → %eax, %ebx, %ecx, %edx  
d=a+c;
```

- Esse código pode ser traduzido da seguinte forma:

```
cmpl %ebx, %eax
```

```
jne depois_if → condição é negativa do if!
```

```
movl %edx, %ecx
```

```
depois_if:
```

```
movl %eax, %edx
```

```
addl %ecx, %edx
```

Exemplo do Laboratório

- Impressão dos números pares de um array

```
    movq  $S2, %rcx

L1:
    movl  (%rcx), %eax
    cmpl  $0, %eax
    je L3
    movl  %eax,%edx
    andl  $0x01,%edx /* ímpar? */
    jnz  L2
    ...
    call printf
    ...
L2:
    addq  $4, %rcx /* rcx+=4; */
    jmp   L1

L3:
```

Exemplo do Laboratório

- Impressão dos números pares de um array

```
movq  $S2, %rcx

L1:
movl  (%rcx), %eax
cmpl  $0, %eax
je    L3
movl  %eax,%edx
andl  $0x01,%edx /* ímpar? */
jnz   L2
...
call  printf
...
L2:
addq  $4, %rcx /* rcx+=4; */
jmp   L1

L3:
```

```
if (a % 2 == 0)
    printf(...);
```

Exemplo do Laboratório

- Impressão dos números pares de um array

```
movq $S2, %rcx

L1:
movl (%rcx), %eax
cmpl $0, %eax
je L3
movl %eax,%edx
andl $0x01,%edx /* ímpar? */
jnz L2
...
call printf
...
L2:
addq $4, %rcx /* rcx+=4; */
jmp L1

L3:
```

```
if (a % 2 == 0)
    printf(...);
```

```
if (!t) goto done;
then-statement
done:
```

Traduzindo “if-else”

```
d = 16;  
a = 10;  
if (d < a) c = a - d;  
else c = d - a;
```

Traduzindo “if-else”

```
if (t)
  then-statement;
else
  else-statement;
```

↓ Esquema geral ↓

```
if (!t) goto false;
  then-statement
  goto done;
false:
  else-statement
done:
```

Traduzindo “if-else”

```
if (t)
  then-statement;
else
  else-statement;
```

↓ Esquema geral ↓

```
if (!t) goto false;
then-statement
goto done;
false:
  else-statement
done:
```

```
d = 16;
a = 10;
if (d < a) c = a - d;
else c = d - a;
```

↓ Caso Específico ↓

```
movl $0x10, %edx
movl $0xA, %eax
cmpl %eax, %edx
jge L1
movl %eax, %ecx
subl %edx, %ecx
jmp L2
L1:          /* false */
movl %edx, %ecx
subl %eax, %ecx
L2:          /* done*/
```

Traduzindo “while”

```
while (a <= b) {  
    ...  
    a++;  
}
```

Traduzindo “while”

```
while (t)  
  Body
```

↓ Esquema geral ↓

```
loop:  
  if (!t) goto after;  
  Body  
  goto loop;  
after:
```

Traduzindo “while”

```
while (t)
  Body
```

↓ Esquema geral ↓

```
loop:
  if (!t) goto after;
  Body
  goto loop;
after:
```

```
while (a <= b) {
  ...
  a++;
}
```

↓ Caso Específico ↓

```
loop:
  cmpl %ebx, %eax
  jg after /* se a>b */
  ...
  incl %eax
  jmp loop
after:
```

Traduzindo o “for”

```
for (Init; Test; Update )  
    Body
```

Traduzindo o “for”

```
for (Init; Test; Update )  
    Body
```



```
Init;  
while (Test) {  
    Body ;  
    Update ;  
}
```

Traduzindo o “for”

```
for (Init; Test; Update )  
  Body
```



```
Init;  
while (Test) {  
  Body ;  
  Update ;  
}
```



```
Init;  
loop:  
  if (!Test)  
    goto after;  
  Body ;  
  Update ;  
  goto loop;  
after:
```

curto circuito em condições lógicas

```
if ((a==b) || (c<d)) {  
    a = c;  
}  
c = d;
```

não existem
operadores *and* e *or*
lógicos nas
linguagens de
máquina!

curto circuito em condições lógicas

```
if ((a==b) || (c<d)) {  
    a = c;  
}  
c = d;
```

temos que testar cada uma delas e construir caminhos diferentes no código!

- em C, como em outras linguagens, avaliação é interrompida quando já se conhece o resultado

A && B

se A é falso...

A || B

se B é verdadeiro...

Avaliando condições com “curto circuito”

- A condição de teste em uma construção de controle pode conter operadores lógicos
 - Exemplo: `((c>a) || ((a ==1) && (d < b)))`
- Em C e outras linguagens de alto nível, a avaliação é interrompida assim que o resultado é conhecido (“*curto circuito*”)
 - `(x || y)` se x resulta em true, não avalia y
 - `(x && y)` se x resulta em false, não avalia y
- Isto é refletido no código Assembly gerado

Exemplo de curto circuito

```
if ((a==b) || (c<d)){  
    a = c;  
}  
c = d;
```

Exemplo de curto circuito

```
if ((a==b) || (c<d)){  
    a = c;  
}  
c = d;
```

```
    cmp %ebx, %eax  
    je L1  
    cmp %edx, %ecx  
    jge L2  
L1:  
    movl %ecx, %eax  
L2:  
    movl %edx, %ecx
```

Assembly - Tradução de Mecanismos de Controle

Noemi Rodriguez

Ana Lúcia de Moura

Raúl Renteria

Alexandre Meslin

<http://www.inf.puc-rio.br/~inf1018>