

Sistemas Hardware- Software

Aula 05 – Condicionais

Ciência da Computação

Carlos Menezes
Maciel C. Vidal
Igor Montagner

Estado do processador

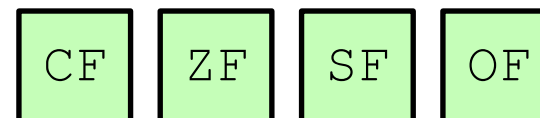
Informação sobre o programa sendo executado:

- Dados temporários (`%rax`, ...)
- Topo da pilha (`%rsp`)
- Posição da instrução atual (`%rip`, ...)
- **Flags de estado dos testes recentes (`CF`, `ZF`, `SF`, `OF`)**

Registradores

<code>%rax</code>	<code>%r8</code>
<code>%rbx</code>	<code>%r9</code>
<code>%rcx</code>	<code>%r10</code>
<code>%rdx</code>	<code>%r11</code>
<code>%rsi</code>	<code>%r12</code>
<code>%rdi</code>	<code>%r13</code>
<code>%rsp</code>	<code>%r14</code>
<code>%rbp</code>	<code>%r15</code>

`%rip` Instruction pointer



Códigos de condição

Códigos de condição

São como registradores de um bit só, que são preenchidos de acordo com o status de uma operação realizada.

Sigla	Nome	Significado
CF	Carry	Overflow unsigned
SF	Signal	Resultado da operação é negativo
OF	Overflow	Overflow signed (complemento de 2)
ZF	Zero flag	Resultado da operação é 0

Códigos de condição

Os códigos de condição são “efeitos colaterais” de operações aritméticas.

Considere a instrução **add S, D**, que calcula **T = S + D** e armazena o resultado **T** de volta em **D**:

Flag set?	Significado
CF	S + D deu carry-out. Equivale a overflow de unsigned.
ZF	T == 0
SF	T < 0 (interpretando T como signed, claro).
OF	S + D deu overflow de complemento-de-2, ou seja, (S > 0 && D > 0 && T < 0) (S < 0 && D < 0 && T >= 0)

Nota: a instrução **lea** não gera códigos de condição.

Instruções de comparação

Permitem preencher os códigos de condição sem modificar os registradores:

- Instrução **cmp A, B**
 - Compara valores A e B
 - Funciona como **sub A, B** sem gravar resultado no destino

Flag set?	Significado
CF	Carry-out em $B - A$
ZF	$B == A$
SF	$(B - A) < 0$ (quando interpretado como signed)
OF	Overflow de complemento-de-2: $(A > 0 \ \&\& \ B < 0 \ \&\& \ (B - A) < 0) \ $ $(A < 0 \ \&\& \ B > 0 \ \&\& \ (B - A) > 0)$

Exemplo

Dump of assembler code for function func:

```
0x0000000000000000 <+0>:      endbr64
0x0000000000000004 <+4>:      cmp      $0x5,%rdi
0x0000000000000008 <+8>:      setle   %al
0x000000000000000b <+11>:     movzbl  %al,%eax
0x000000000000000e <+14>:     retq
```

End of assembler dump.

Instruções de comparação

- Instrução **test A, B**
 - Testa o resultado de **A & B**
 - Funciona como **and A, B** sem gravar resultado no destino
 - Útil para checar um dos valores, usando o outro como máscara
 - Normalmente usado com A e B sendo o mesmo registrador, ou seja: **test %rdi, %rdi**

Flag set?	Significado
ZF	$A \& B == 0$
SF	$A \& B < 0$ (quando interpretado como signed)

Acessando os códigos de condição

Instruções **set**

- Preenchem o byte mais baixo do destino com 0x00 ou 0x01, dependendo de combinações de códigos de condição
- Não alteram os 7 bytes restantes

Acessando os códigos de condição

Instrução	Condição	Descrição
sete	ZF	Equal /Zero
setne	~ZF	Not Equal / Not Zero
sets	SF	(signed) Negativo
setns	~SF	(signed) Não-negativo
setl	(SF^OF)	(signed) Less than
setle	(SF^OF) ZF	(signed) Less than or Equal
setge	~(SF^OF)	(signed) Greater than or Equal
setg	~(SF^OF) & ~ZF	(signed) Greater than
setb	CF	(unsigned) Below
seta	~CF & ~ZF	(unsigned) Above

Exemplo

Dump of assembler code for function func:

```
0x0000000000000000 <+0>:      endbr64
0x0000000000000004 <+4>:      test    %edi,%edi
0x0000000000000006 <+6>:      sete    %al
0x0000000000000009 <+9>:      movzbl  %al,%eax
0x000000000000000c <+12>:     retq
```

End of assembler dump.

Atividade prática

Expressões booleanas

1. Identificar expressões booleanas a partir de código assembly
2. Reconstruir expressões booleanas em C a partir de sequências de instruções cmp/test e set*

Desvios (ou saltos) condicionais

Permitem saltar para outra parte do código dependendo dos códigos de condição. **Finalmente vamos ter `if` !!!**

Equivalem ao código C:

```
if (condição) {  
    goto label;  
}
```

Exemplo:

```
cmp    $0xa,%rdi    # Compara %rdi:10  
jg     400573        # Se >, pula para 400573
```

Desvios (ou saltos) condicionais

Instrução	Condição	Descrição
jmp	1	Incondicional
je	ZF	Equal /Zero
jne	~ZF	Not Equal / Not Zero
js	SF	(signed) Negativo
jns	~SF	(signed) Não-negativo
jl	(SF^OF)	(signed) Less than
jle	(SF^OF) ZF	(signed) Less than or Equal
jge	~(SF^OF)	(signed) Greater than or Equal
jg	~(SF^OF) & ~ZF	(signed) Greater than
jb	CF	(unsigned) Below
ja	~CF & ~ZF	(unsigned) Above

O comando **goto**

Definimos um *label* usando a sintaxe nome :

goto desvia o fluxo para a linha de código abaixo do label

```
int main(int argc, char **argv) {  
    goto pula_para_ca;  
    printf("Este printf não aparece!\n");  
pula_para_ca:  
    printf("Print2!\n");  
}
```

goto só funciona dentro de uma mesma função

O par de comandos **if-goto**

O par de comandos if-goto é equivalente às instruções
cmp/test seguidas de um jump condicional

```
cmp 0x4, %rdi
jle label
(bloco 1)
label:
...
```

```
if (a <= 4) {
    goto label;
}
(bloco1)
label:
. . .
```

O par de comandos **if-goto**

O par de comandos if-goto é equivalente às instruções `cmp/test` seguidas de um `jump` condicional

```
cmp 0x4, %rdi
jle label
(bloco 1)
label:
...
```

```
if (a <= 4) {
    goto label;
}
(bloco1)
label:
. . .
```

Vamos chamar código **C** que use somente `if-goto` de **gotoC!**

Padrões de geração de código

Compiladores transformam o código **C** de diversas maneiras durante geração de código.

C

```
if (cond) {  
    (bloco1)  
}  
. . .
```

gotoC

```
if (!cond)  
    goto depois;  
  
(bloco1)  
  
depois:  
. . .
```

Padrões de geração de código

Compiladores transformam o código **C** de diversas maneiras durante geração de código.

C

```
if (cond) {  
    (bloco1)  
} else {  
    (bloco2)  
}  
. . .
```

gotoC

```
if (!cond)  
    goto else;  
  
(bloco1)  
goto fim;  
  
else:  
(bloco2)  
  
fim:  
. . .
```

Código C com goto

Para entender o código assembly, devemos traduzir código C normal em código C com **goto**

```
long foo(long x, long y) {  
    long result;  
    if (x > y) {  
        result = x - y;  
    }  
    else {  
        result = y - x;  
    }  
    return result + 1;  
}
```

```
long foo_j(long x, long y) {  
    long result;  
  
    int ntest = x <= y;  
    if (ntest) goto Else;  
    result = x - y;  
    goto Done;  
  
Else:  
    result = y - x;  
  
Done:  
    result = result + 1;  
    return result;  
}
```

Código C com goto

```
long foo_j(long x, long y) {  
    long result;  
  
    int ntest = x <= y;  
    if (ntest) goto Else;  
  
    result = x - y;  
    goto Done;  
  
Else:  
    result = y - x;  
  
Done:  
    result = result + 1;  
    return result;  
}
```

0000000000000000 <foo>:

0:	48 39 f7	cmp	%rsi,%rdi
3:	7e 08	jle	d <foo+0xd>
5:	48 29 f7	sub	%rsi,%rdi
8:	48 89 fe	mov	%rdi,%rsi
b:	eb 03	jmp	10 <foo+0x10>
d:	48 29 fe	sub	%rdi,%rsi
10:	48 8d 46 01	lea	0x1(%rsi),%rax
14:	c3	retq	

Atividade prática

Condicionais: if e if/else

1. Identificar as expressões booleanas testadas em instruções de pulo condicional
2. Reconstruir o fluxo de controle de um programa em C a partir de sua versão compilada

Insper

www.insper.edu.br