

Linguagens de montagem

Capítulo 6 – Instruções lógicas e de deslocamento

Ricardo Anido
Instituto de Computação
Unicamp

Instruções lógicas

	<hr/> E-lógico <hr/>	<hr/> OU-lógico <hr/>	<hr/> OU-exclusivo <hr/>
Operando1	00001111	00001111	00001111
Operando2	00110011	00110011	00110011
Resultado	00000011	00111111	00111100

Instruções lógicas

OR													
OU-lógico													
Sintaxe	Operação	Flags	Codificação										
$or\ rd,\ rf$	$rd \leftarrow rd \vee rf$	NZ (C,V \leftarrow 0)	<table border="1"><tr><td>31</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0x31</td><td>-</td><td>rd</td><td>rf</td><td></td></tr></table>	31				0	0x31	-	rd	rf	
31				0									
0x31	-	rd	rf										

Instruções lógicas

XOR													
OU-EXCLUSIVO lógico													
Sintaxe	Operação	Flags	Codificação										
$xor\ rd,\ rf$	$rd \leftarrow rd \oplus rf$	NZ (C,V \leftarrow 0)	<table border="1"><tr><td>31</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0x32</td><td>-</td><td>rd</td><td>rf</td><td></td></tr></table>	31				0	0x32	-	rd	rf	
31				0									
0x32	-	rd	rf										

Instruções lógicas

TST													
Testa bits													
Sintaxe	Operação	Flags	Codificação										
<code>tst rd, rf</code>	$rd \wedge rf$	NZ (C,V←0)	<table border="1"><tr><td>31</td><td></td><td></td><td></td><td>0</td></tr><tr><td>0x33</td><td>-</td><td>rd</td><td>rf</td><td></td></tr></table>	31				0	0x33	-	rd	rf	
31				0									
0x33	-	rd	rf										

Problema

Escreva um trecho de programa para trocar os bits mais e menos significativos de `r0`, sem alterar os bits restantes, como ilustrado na Figura abaixo.



Antes



Depois

Solução

@ trecho para trocar o bit mais significativo com o menos significativo

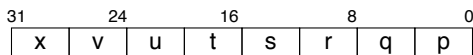
@ r0 tem ybbb...bbbx no início, deve ter xbbb...bbby ao final

inverte:

```
    set    r10,0x80000000 @ máscara para isolar bit y
    set    r11,1          @ máscara para isolar bit x
    set    r12,0x7fffffff @ máscara para desligar x e y
    mov    r1,r0          @ vamos usar r1 e
    mov    r2,r0          @ r2 como auxiliares
    and    r0,r12         @ r0 agora tem 0bbb..bbb0
    and    r1,r11         @ bit x é um?
    jz     bit_x_zero     @ se não é, desvia
    or     r0,r10         @ monta bit 1 no lugar de y em r0
bit_x_zero:              @ aqui r0 tem ybbb..bbb0
    and    r2,r10         @ bit y é um?
    jz     bit_y_zero     @ se não, desvia
    or     r0,r11         @ monta bit 1 no lugar de x em r0
bit_y_zero:              @ aqui r0 tem ybbb..bbbx
```


Comparando elementos de uma estrutura

suponha que uma estrutura de oito elementos com quatro bits cada é implementada usando uma palavra de 32 bits, como mostrado na Figura abaixo:



Suponha agora que desejamos testar se o elemento p da estrutura tem o mesmo valor que o elemento q.

Comparando elementos de uma estrutura

@ reserva espaço para a estrutura

estrutura:

```
.skip    1
```

```
...
```

@ aqui inicia o trecho para isolar o elemento 'p' em r10

```
ld      r10, estrutura    @ carrega estrutura
```

```
set     r0, 0x0f          @ máscara para isolar 4 bits menos signif.
```

```
and     r10, r0           @ r10 tem o elemento 'p' isolado
```

```
...
```

@ aqui inicia o trecho para isolar o elemento 'q' em r9

```
ld      r9, estrutura    @ carrega estrutura
```

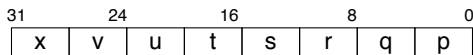
```
set     r0, 0xf0         @ máscara para isolar o elemento 'q'
```

```
and     r9, r0           @ r9 tem o elemento 'q' isolado, mas não na  
                               @ posição correta para comparar com 'p'
```

Comparando elementos de uma estrutura

Apesar de o elemento q estar isolado ao final do trecho, seus bits não estão corretamente posicionados para ter o valor do elemento comparado com o elemento p , isolado no registrador $r10$.

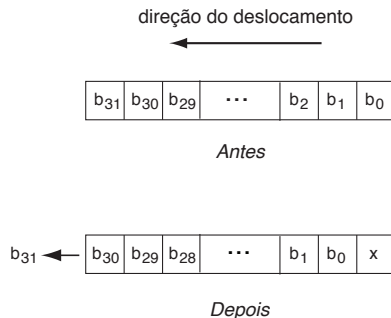
Para tanto, seria necessário deslocar o valor de $r9$ de quatro bits para a direita.



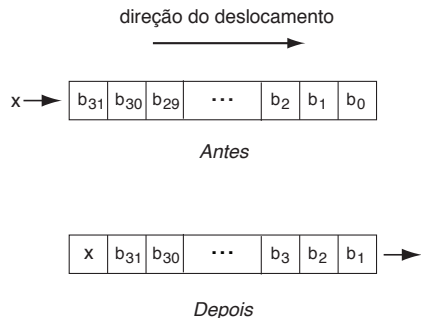
Instruções de deslocamento

- ▶ As instruções de deslocamento do LEG operam sempre sobre um registrador, e deslocam os bits do registrador operando para a direita ou para a esquerda.
- ▶ Todos os bits são deslocados ao mesmo tempo.
- ▶ Por exemplo, no deslocamento para a esquerda, o bit b_0 (menos significativo) do registrador é deslocado para a posição do bit b_1 , que por sua vez é deslocado para a posição do bit b_2 , e assim por diante.

Instruções de deslocamento



(a) Deslocamento para a esquerda

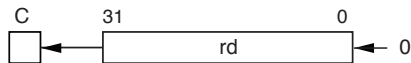


(b) Deslocamento para a direita

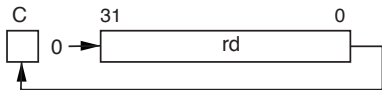
O LEG tem três tipos diferentes de instruções de deslocamento:

- ▶ Deslocamento para a esquerda, cujo comando em linguagem de montagem é SHL (do inglês *shift left*).
- ▶ Deslocamento para a direita, cujo comando em linguagem de montagem é SHR (do inglês *shift right*).
- ▶ Deslocamento aritmético para a direita, cujo comando em linguagem de montagem é SAR (do inglês *shift arithmetic right*).

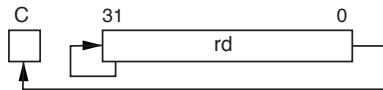
Instruções de deslocamento



SHL
deslocamento para a esquerda



SHR
deslocamento para a direita



SAR
deslocamento aritmético para a direita

Deslocamento para a esquerda

SHL			
Deslocamento para a esquerda			
Syntax	Operação	Flags	Codificação
<code>shl rd, expr5</code>	repita imd_5 vezes { $rd[i + 1] \leftarrow rd[i]$ $C \leftarrow rd[31]$ $rd[0] \leftarrow 0$ }	CNZ ($V \leftarrow 0$)	31 0 0x42 imd_5 rd -
<code>shl rd, rf</code>	repita rf vezes { $rd[i + 1] \leftarrow rd[i]$ $C \leftarrow rd[31]$ $rd[0] \leftarrow 0$ }	CNZ ($V \leftarrow 0$)	31 0 0x43 - rd rf

Deslocamento para a direita

SHR							
Deslocamento para a direita							
Syntax	Operação	Flags	Codificação				
$\text{shr } rd, \text{expr5}$	repita imd_5 vezes { $rd[i] \leftarrow rd[i + i]$ $C \leftarrow rd[0]$ $rd[31] \leftarrow 0$ }	CNZ ($V \leftarrow 0$)	31 0 <table border="1"><tr><td>0x40</td><td>imd_5</td><td>rd</td><td>-</td></tr></table>	0x40	imd_5	rd	-
0x40	imd_5	rd	-				
$\text{shr } rd, rf$	repita rf vezes { $rd[i] \leftarrow rd[i + i]$ $C \leftarrow rd[0]$ $rd[31] \leftarrow 0$ }	CNZ ($V \leftarrow 0$)	31 0 <table border="1"><tr><td>0x41</td><td>-</td><td>rd</td><td>rf</td></tr></table>	0x41	-	rd	rf
0x41	-	rd	rf				

Deslocamento aritmético para a direita

SAR							
Deslocamento aritmético para a direita							
Syntax	Operação	Flags	Codificação				
$\text{sar } rd, \text{expr5}$	repita imd_5 vezes { $rd[i] \leftarrow rd[i + i]$ $C \leftarrow rd[0]$ $rd[31] \leftarrow rd[31]_{\text{ant}}$ }	CNZ ($V \leftarrow 0$)	31 0 <table border="1"><tr><td>0x44</td><td>imd_5</td><td>rd</td><td>-</td></tr></table>	0x44	imd_5	rd	-
0x44	imd_5	rd	-				
$\text{sar } rd, rf$	repita rf vezes { $rd[i] \leftarrow rd[i + i]$ $C \leftarrow rd[0]$ $rd[31] \leftarrow rd[31]_{\text{ant}}$ }	CNZ ($V \leftarrow 0$)	31 0 <table border="1"><tr><td>0x45</td><td>-</td><td>rd</td><td>rf</td></tr></table>	0x45	-	rd	rf
0x45	-	rd	rf				

Suponha que `r12` contém o endereço inicial de uma cadeia de caracteres '0' ou '1' que representa um número em notação binária, sendo que `r12` aponta para o caractere que representa o “bit mais significativo” da cadeia. O número de caracteres da cadeia, entre 1 e 32, é dado no registrador `r1`. Escreva um trecho de programa para colocar em `r0` o valor que a cadeia dada representa. Por exemplo, se a cadeia de caracteres é '01011011' (em hexadecimal, os valores dos caracteres são 0x30, 0x31, 0x30, 0x31, 0x31, 0x30, 0x31, 0x31), o valor de `r0` ao final do trecho deve ser 0x5b.

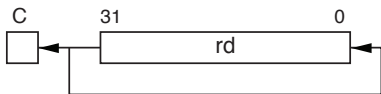
@ trecho para calcular valor binário representado por uma cadeia de
@ caracteres '0' e '1'
@ r2 tem endereço do primeiro caractere (o 'dígito mais signif.)
@ r1 tem número de caracteres da cadeia

```
xor  r0,r0          @ zera r0
                        @ vamos montar valor bit a bit em r0

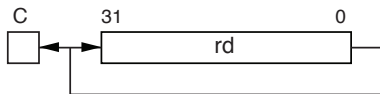
prox_dig:
  ldb  r2,[r12]      @ r2 tem dígito: 0x30 ou 0x31
  shl  r0,1          @ prepara espaço para novo bit
  sub  r2,0x30       @ r2 tem o valor do dígito: 0 ou 1
  or   r0,r2         @ monta novo bit com valor 0 ou 1
  add  r12,1         @ avança apontador
  sub  r1,1          @ chegou ao final da cadeia?
  jnz  prox_dig      @ não, trata mais dígitos
...                @ sim, término; valor do byte em r0
```

Instruções de rotação

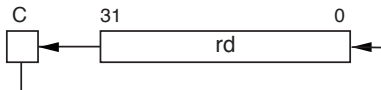
Instruções de rotação são similares a instruções de deslocamento, mas o bit ejetado do registrador é injetado novamente no operando, no lado oposto de onde o bit foi ejetado.



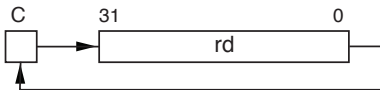
ROL
rotação para a esquerda



ROR
rotação para a direita



RCL
rotação com vai-um para a esquerda



RCR
rotação com vai-um para a direita

Instruções de rotação

- ▶ rotação para a esquerda, cujo comando em linguagem de montagem é ROL (do inglês *Rotate Left*),
- ▶ rotação para a direita, cujo comando em linguagem de montagem é ROR (do inglês *Rotate Right*),
- ▶ rotação para a esquerda com vai-um, cujo comando em linguagem de montagem é RCL (do inglês *Rotate with Carry Left*),
- ▶ rotação para a direita com vai-um, cujo comando em linguagem de montagem é RCR (do inglês *Rotate with Carry Right*).

Os bits de estado C (vai-um), N (sinal) e Z (zero) são afetados pelo resultado das instruções de deslocamento; o bit de estado V (estouro de campo) é sempre zerado nessas instruções.

Solução alternativa, com rotação

@ trecho para calcular valor binário representado por uma cadeia de
@ caracteres '0' e '1' apontada por r12; r1 tem número de caracteres
@ segunda versão, usando instruções de rotação

```
xor    r0,r0        @ r0 será usado como temporário
                    @ vamos montar valor bit a bit
```

prox_dig:

```
ldb    r2,[r12]    @ r2 tem dígito: 0x30 ou 0x31, coloca
rcr    r2,1        @ novo bit 0 ou 1 no bit C ('vai-um')
rcl    r0,1        @ monta novo bit em r0
add    r12,1       @ avança apontador da cadeia
sub    r1,1        @ chegou ao final da cadeia?
jnz    prox_dig    @ não, trata mais dígitos
```