Linguagens de montagem
Capítulo 11 - ARM - Desvios e processamento de dados

Ricardo Anido
Instituto de Computação
Unicamp

Desvios com endereço alvo constante

$B\{cond\}$ *endereço*

onde

- *cond* é um sufixo de condição
- *endereço* é o endereço alvo do desvio, normalmente dado como um rótulo do programa, que deve estar dentro do intervalo

$$[\mathrm{pc} - (2^{25} - 1), \mathrm{pc} + 2^{25}]$$

Desvios com alvo constante

Exemplo:

```
    bne  loop                @ desvio condicional
    b    fim                 @ desvio incondicional
loop:                        @ um rótulo
    ...
fim:                         @ outro rótulo
    ...
```

Desvios por registrador

Formato geral:

$$BX\{cond\} \quad Rd$$

Exemplo:

```
1    bxmi  r10                @ desvio condicional
2    bx    r8                 @ desvio incondicional
```

Instruções de processamento

COM	Nome	Operação
AND	E-lógico	Rd $\leftarrow$ Rn $\wedge$ Operando2
EOR	Ou-exclusivo	Rd $\leftarrow$ Rn $\oplus$ Operando2
SUB	Subtração	Rd $\leftarrow$ Rn $-$ Operando2
RSB	Subtração reversa	Rd $\leftarrow$ Operando2 $-$ Rn
ADD	Adição	Rd $\leftarrow$ Rn $+$ Operando2
ADC	Adição com vai-um	Rd $\leftarrow$ Rn $+$ Operando2 $+$ C
SBC	Subtração com empresta-um	Rd $\leftarrow$ Rn $-$ Operando2 $+$ C $-$ 1
RSC	Subtração reversa com empresta-um	Rd $\leftarrow$ Operando2 $-$ Rn $+$ C $-$ 1
TST	Testa bits	Rn $\wedge$ Operando2
TEQ	Testa equivalência	Rn $\oplus$ Operando2
CMP	Comparação	Rn $-$ Operando2
CMN	Comparação negativa	Rn $+$ Operando2
ORR	Ou-lógico	Rd $\leftarrow$ Rn $\vee$ Operando2
MOV	Move registrador	Rd $\leftarrow$ Operando2
BIC	Desliga bit	Rd $\leftarrow$ Rn $\wedge$ *not* Operando2
MVN	Move registrador negado	Rd $\leftarrow$ *not* Operando2

Instruções de processamento

- Diferentemente do LEG, em que toda instrução aritmética ou lógica atualiza os bits de condição, no ARM é possível escolher se a instrução de processamento de dados deve ou não atualizar os bits de condição.
- o bit S da instrução determina se o processador deve ou não atualizar os bits de condição
- em linguagem de montagem, usamos o sufixo "s" para indicar que a instrução deve atualizar os bits de condição (como "ands").

Instruções de transferência entre registradores

O formato dos comandos MOV e MVN em linguagem de montagem é:

instr{*cond*}{*S*} `Rd`, *Operando2*

- `S` indica se a instrução deve alterar os bits de condição. Se presente, a instrução altera os bits de condição N, Z e C de acordo com o resultado da operação; se ausente os bits de condição não são alterados. O bit de condição V nunca é afetado.
- `Rd` é o registrador destino para a operação.

Instruções de transferência entre registradores

Operando2 pode ser

- um valor imediato; nesse caso a sintaxe em linguagem de montagem para Operando2 é #*expr*32. O montador procura encontrar um valor de oito bits e uma operação de deslocamento para montar respectivamente nos campos Imed8 e Shf da instrução que produzam o valor *expr*32. Se não encontra, um erro é gerado (o valor *expr*32 não pode ser carregado de forma imediata em um registrador).
- um registrador (possivelmente deslocado pela Unidade de Deslocamento); nesse caso a sintaxe em linguagem de montagem para Operando2 é

  $$Rm \{,shift\ Rs\}$$

  onde *shift* é um dos comandos seguintes.

Operações de deslocamento

Comando	Operação
LSL	Deslocamento lógico para a esquerda
LSR	Deslocamento lógico para a direita
ASR	Deslocamento aritmético para a direita
ROR	Rotação para a direita

Instruções de transferência entre registradores

```
movs   r11,#0          @ carrega registrador com valor imediato
                       @ r11 <-- 0, faz N=0, Z=1
mov    r1,r2           @ cópia simples, incondicional
                       @ r1 <-- r2
mvn    r3,r4           @ cópia com negação, incondicional
                       @ r3 <-- -r4
mvneqs r10,#0x80000000   @ carga de registrador com valor
                         @ imediato
                         @ r10 <-- 0x80000000
                         @ condicional; se executada,
                         @ bits N=1, Z=0
mov    r12,r8,lsr#2 @ cópia com deslocamento, incondicional
                       @ r12 <-- r8 / 4
movgt  r9,r9,lsl#2  @ deslocamento, condicional
                       @ r9 <-- r9 * 4
movcc  r13,r10,ror r2 @ cópia com deslocamento,
                       @incondicional
                       @ r9 <-- r10 rotacionado r2 bits
```

Instruções que não armazenam resultado

As instruções CMP (compara), CMN (compara com negação), TEQ (testa equivalência) e TST (testa bits) não armazenam o resultado. Formato geral:

instr{*cond*} *Rn*, *Operando2*

Exemplos:

```
cmp    r1,r2           @ comparação simples, incondicional
                       @ [C,N,V,Z] <-- r1-r2
cmnmi  r10,r12         @ comparação com negação, condicional
                       @ [C,N,V,Z] <-- r10-(-r12)
teqeq  r10,#1          @ testa bits com valor imediato
                       @ [C,N,V,Z] <-- r10 ou_exclusivo 1
                       @ condicional
tst    r9,r8,lsl#3     @ testa bits com registrador, incond.
                       @ [C,N,V,Z] <-- r10 e_logico r8*8
tstcs  r0,r11,ror r1   @ testa bits com registrador, condicional
                       @ [C,N,V,Z] <--
                       @   r0 e_logico (r11 rotacionado r1 bits)
```

Instruções que aarmazenam o resultado

- ADD (adição)
- SUB (subtração)
- ADC (adição com vai-um)
- SBC (subtração com empresta-um)
- RSB (subtração reversa)
- AND (e-lógico)
- ORR (ou-lógico)
- EOR (ou-exclusivo)
- RSC (subtração reversa com empresta-um)
- BIC (desliga bit)

Instruções que aarmazenam o resultado

Formato geral em linguagem de montagem:

instr{*cond*}{S} *Rd, Rn, Operando2*

Exemplos:

```
add    r1,r2,r3           @ adição, incondicional
                          @ r1 <-- r2 + r3
                          @ bits de condição não são alterados
subeqs r3,r7,r9,lsr r10   @ subtração, condicional
                          @ r3 <-- r7 + (r9 desloc. direita por r10)
                          @ bits de condição são atualizados
rsbpls r4,r6,r8,lsl #4    @ subtração reversa, condicional
                          @ r4 <-- (r8 * 16) - r6
                          @ bits de condição são atualizados
ands   r14,r14,#2048      @ e-lógico, incondicional
                          @ r14 <-- r14 & 0x800
```

Problema

Escreva um trecho de programa em linguagem de montagem ARM para substituir, em uma cadeia de caracteres, toda ocorrência do caractere "espaço-em-branco" (byte `0x20`) pelo caractere '-' (byte `0x2d`). Suponha que o endereço da cadeia seja dado em `r0` e o número de elementos em `r1`.

Solução

```
inicio:
    ldr     r0,ender_cadeia     @ carrega endereço de cadeia em r0, e
    ldr     r1,tamanho_cadeia   @ número de elementos da cadeia em r1

substitui:
    cmp     r1,#0               @ se comprimento é zero,retorna
    ble     final
    add     r2,r0,r1            @ r2 marca final da cadeia
    mov     r1,#0x2d            @ r1 contém caractere '-'
loop:
    ldrb    r3,[r0],#1          @ examina caractere corrente
                                @ pós-indexada, r0 é atualizado
    cmp     r3,#0x20            @ caractere é espaço?
    streqb  r1,[r0,#-1]         @ se sim, substitui
    cmp     r0,r2               @ chegou ao final da cadeia?
    bne     loop                @ não, continua
final:
...                             @ continua o programa (não mostrado)
```

Problema

Traduza o trecho de programa em C a seguir, que contém um comando condicional com duas expressões lógicas, para linguagem de montagem do ARM.

```
int x, y, z;
...
if (x==y && x==z)
   x=0;
else
   x=100;
```

Solução

```
x:   .skip 4
y:   .skip 4
z:   .skip 4
...
     ldr    r0,x            @ carrega os valores de x, y e z
     ldr    r1,y            @ respectivamente nos registradores
     ldr    r2,z            @ r0, r1 e r2

     cmp    r0,r1           @ primeira expressão do if, x==y
     cmpeq  r0,r2           @ computa segunda expressão (x==z) somente
                            @ se primeira for verdadeira
     moveq  r3,#0           @ prepara registrador r3 com constante
     movne  r3,#100         @ uma delas será atribuída a x
     str    r3,x            @ armazena constante em x

...
```