Linguagens de montagem
Capítulo 10 - ARM - transferência entre registrador e memória

Ricardo Anido
Instituto de Computação
Unicamp

Carga e Armazenamento

- Instruções para carga e armazenamento de registradores em memória são respectivamente LDR (do inglês *load register*) e STR (do inglês *store register*).
- Codificação:

31	27	26	25	24	23	22	21	20	16	12	0
Cond	0	1	I	P	U	B	W	L	Rn	Rd	Deslocamento

Carga e armazenamento

Formato básico em linguagem de montagem:

LDR *Rd, endereço*
STR *Rd, endereço*

onde *endereço* depende do modo de endereçamento.

Endereçamento direto

instr{*cond*}{*B*} `Rd`, *endereço*

onde

- *instr* especifica a instrução e pode ser LDR ou STR.
- *cond* é um sufixo de condição.
- B, se presente, faz com que o montador coloque o valor 1 no bit 22 da instrução (bit B, indicando que a operação deve carregar ou armazenar bytes; o byte menos significativo do registrador `Rd` é carregado ou armazenado (na instrução LDR os outros 24 bits do registrador são zerados). Se não presente, o bit B da instrução é 0, indicando que a operação deve carregar ou armazenar palavras.
- `Rd` é o registrador destino (onde o valor lido da memória é carregado) para instruções LDR e o registrador fonte (contém valor a ser armazenado na memória) para instruções STR.

Endereçamento direto

- *endereço* é o endereço do operando na memória (normalmente especificado por um rótulo do programa). O montador monta no campo Deslocamento da instrução a diferença entre *endereço* e o ponto corrente de montagem. Em outras palavras, o endereçamento é relativo ao valor do registrador contador de programa `pc`. No momento da execução, o processador calcula o endereço efetivo do operando adicionando o valor do registrador `pc` ao valor do campo Deslocamento com o sinal estendido para 32 bits. Um erro é gerado pelo montador se a diferença entre o endereço do operando e o ponto corrente de montagem não puder ser codificado em 12 bits.

Endereçamento direto

```
   .org 0x1000
   ldr    r1, var1  @ carrega palavra em r1, incondicional
   ldrgt  r2, var1  @ carrega palavra em r2, condicional
   ldreqb r3, var2  @ carrega byte em r3, condicional

   str    r4, var1  @ armazena r4 em  palavra, incondicional
   strcc  r5, var1  @ armazena r5 em palavra, condicional
   strleb r6, var2  @ armazena r6 em byte, condicional

   ldr    r7, var_longe @ gera erro, endereço muito distante

   .org 0x2000
var1:
   .word 0x12341234
var2:
   .byte 0xff

   .org 0x20000
var_longe:
   .word 0x0
```

Endereçamento indireto por registrador

Formato geral em linguagem de montagem:

instr{*cond*}{*B*} *Rd, [Rn]*

Exemplos:

```
ldr    r1,[r0]  @ carrega palavra em r1, incondicional
ldrmi  r2,[r8]  @ carrega palavra em r2, condicional
ldrplb r3,[r9]  @ carrega byte em r3, condicional

str    r4,[r10] @ armazena r4 em  palavra, incondicional
strlt  r5,[r11] @ armazena r5 em palavra, condicional
strgtb r6,[r12] @ armazena r6 em byte, condicional
```

Modos de endereçamento pré-indexados

- Nos modos de endereçamento pré-indexados, o endereço efetivo do operando em memória é formado pelo valor do registrador base mais (ou menos) um *deslocamento*, codificado no campo Deslocamento da instrução.
- A forma do cálculo desse *deslocamento* dá origem a diferentes modos de endereçamento.

Modos de endereçamento pré-indexados

O *deslocamento* pode ser dado por:

- um valor imediato, constante, dando origem ao modo de endereçamento *indireto por registrador mais constante*
- o valor de um registrador, dando origem ao modo de endereçamento *indireto por registrador base e registrador índice*
- o valor de um registrador deslocado com o auxílio da unidade de deslocamento (*barrel shifter*), dando origem ao modo de endereçamento *indireto por registrador e registrador índice escalado*.

- Há ainda a possibilidade de atualizar o valor do registrador base com o valor do endereço efetivo, dessa forma fazendo com que o registrador "avance" (ou "recue") automaticamente por um valor controlado, dentro do mesmo ciclo de instrução.
- usado na implementação de comandos de repetição e na varredura de estruturas de dados sequenciais (como vetores, por exemplo).

Endereçamento pré-indexado indireto por registrador mais constante

Formato geral:

instr{*cond*}{*B*} *Rd*, [*Rn*, #*expr*12] {!}

Exemplos:

```
ldr    r1, [r0,#1]    @ carrega palavra em r1, incondicional
ldrmi  r2, [r8,#4]!   @ carrega palavra em r2, condicional
                      @ r8 é incrementado de 4
ldrplb r3, [r9,#2]    @ carrega byte em r3, condicional

str    r4, [r10,#-2]! @ armazena r4 em  palavra, incondicional
                      @ r10 é decrementado de 2
strlt  r5, [r11,#4]   @ armazena r5 em palavra, condicional
strgtb r6, [r12,#1]!  @ armazena r6 em byte, condicional
                      @ r12 é incrementado de 1
```

Endereçamento pré-indexado indireto por registrador base e registrador índice

Formato geral:

$$instr\{cond\}\{B\} \quad Rd,\ [Rn,\ \{+|-\}\ Rm]\ \{!\}$$

Exemplos:

```
ldr    r1, [r0,r2]       @ carrega palavra em r1, incondicional
                         @ endereço efetivo é r0+r2
                         @ r0 não é alterado
ldrvs  r2, [r8,-r3]!     @ carrega palavra em r2, condicional
                         @ endereço efetivo é r8-r3
                         @ r8 tem valor alterado ao final
ldrvcb r3, [r9,+r10]     @ carrega byte em r3, condicional
                         @ endereço efetivo é r9+r10
                         @ r9 não é alterado
str    r4, [r10,-r0]!    @ armazena r4 em  palavra, incondicional
                         @ endereço efetivo é r10-r0
                         @ r10 tem valor alterado ao final
strhi  r5, [r11,r0]      @ armazena r5 em palavra, condicional
                         @ endereço efetivo é r11+r0
```

Endereçamento pré-indexado indireto por registrador base e registrador índice escalado

Formato geral:

instr{*cond*}{*B*} *Rd, [Rn,* {+|−} *Rm, shift #expr5]* {*!*}

Exemplos:

```
ldr    r1, [r0,r2,lsl#1]     @ carrega palavra em r1, incondicional
                             @ endereço efetivo é r0+(r2<<1)
                             @ r0 não é alterado
ldrgt  r2, [r8,-r3,lsr#2]!   @ carrega palavra em r2, condicional
                             @ endereço efetivo é r8-(r3>>2)
                             @ r8 tem valor alterado ao final
ldrltb r3, [r9,+r10,asr#4]   @ carrega byte em r3, condicional
                             @ endereço efetivo é r9+(r10 >>arit. 4)
                             @ r9 não é alterado
str    r4, [r10,-r0,ror#5]!  @ armazena r4 em  palavra, incondicional
                             @ endereço efetivo é:
                             @    r10-(r0 rotacionado 5 posições)
                             @ r10 tem valor alterado ao final
```

Modos de endereçamento pós-indexados

- Os modos de endereçamento pós-indexados são muito similares aos modos de endereçamento pré-indexados.
- A diferença é que, nos modos de endereçamento pós-indexados, o endereço efetivo do operando é o valor do registrador base, mas após o acesso à memória, o registrador base é (sempre) atualizado, adicionando ou subtraindo o valor do operando indexador (constante, registrador índice ou registrador índice escalado).
- Como o registrador base é sempre atualizado 0 indicador "!" não é utilizado.
- O montador diferencia entre os modos pré-fixado e pós-fixado pela sintaxe do comando, que é ligeiramente diferente (note a posição dos operandos em relação às chaves).

Endereçamento pós-indexado indireto por registrador mais constante

Formato geral:

instr{*cond*}{*B*} *Rd*, [*Rn*], #*expr*12

Exemplos:

```
ldrplb r3,[r9],#2          @ carrega byte em r3, condicional
                           @ endereço efetivo é r9
                           @ r9 é atualizado para r9+2
str    r4,[r10],#-2        @ armazena r4 em  palavra, incondicional
                           @ endereço efetivo é r10
                           @ r10 é atualizado para r10-2
```

Endereçamento pós-indexado indireto por registrador base e registrador índice

Formato geral:

$$instr\{cond\}\{B\} \quad Rd,\ [Rn],\ \{+|-\}\ Rm$$

Exemplos:

```
ldrvs  r2,[r8],-r3        @ carrega palavra em r2, condicional
                          @ endereço efetivo é r8
                          @ r8 é atualizado para r8-r3
ldrvcb r3,[r9],+r10       @ carrega byte em r3, condicional
                          @ endereço efetivo é r9
                          @ r9 é atualizado para r9+r10
str    r4,[r10],-r0       @ armazena r4 em  palavra, incondicional
                          @ endereço efetivo é r10
                          @ r10 é atualizado para r10-r0
```

Endereçamento pós-indexado indireto por registrador base e registrador índice escalado

Formato geral:

instr{*cond*}{*B*} *Rd, [Rn], {+|−} Rm, shift #expr5*

Exemplos:

```
strhi  r5,[r11],r0          @ armazena r5 em palavra, condicional
                            @ endereço efetivo é r11
                            @ r11 é atualizado para r11+r0
ldr    r1,[r0],r2,lsl#1     @ carrega palavra em r1, incondicional
                            @ endereço efetivo é r0
                            @ r0 é atualizado para r0+(r2<<1)
ldrgt  r2,[r8],-r3,lsr#2    @ carrega palavra em r2, condicional
                            @ endereço efetivo é r8
                            @ r8 é atualizado para r8-(r3>>2)
strltb r3,[r9],+r10,asr#4   @ armazena r3 em byte, condicional
                            @ endereço efetivo é r9
                            @ r9 é atualizado para r9+(r10 >>arit.4)
```

Problema

Traduza para linguagem de montagem ARM o seguinte trecho de programa C:

```
int a,i,vetor_int[100];
char b,vetor_char[100];
...
    vetor_char[i]=b;
    vetor_int[i]=a;
...
```

Exemplo de solução

```
a:
    .skip 4
i:
    .skip 4
vetor_int:
    .skip 100*4
b:
    .skip 1
    .align 2
vetor_char:
    .skip 100
ender_vetor_char:
    .word vetor_char
ender_vetor_int:
    .word vetor_int
```

Exemplo de solução

```
ldr   r0,a                  @ carrega valor de a
ldr   r1,i                  @ índice
ldr   r3,ender_vetor_char   @ endereço de vetor_char
str   r0,[r3,r1]            @ armazena em vetor_char[i]
ldr   r0,b                  @ carrega valor de b
ldr   r3,ender_vetor_int    @ endereço base
str   r0,[r3,r1,LSL#2]      @ armazena em vetor_int[i]
```