

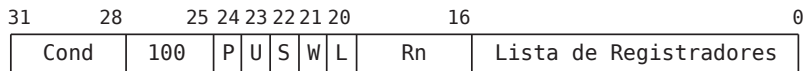
Linguagens de montagem

Capítulo 12 - ARM - Transferências múltiplas, pilhas e procedimentos

Ricardo Anido
Instituto de Computação
Unicamp

Instruções para transferências múltiplas

- ▶ Transferem um grupo de registradores da memória para o processador ou do processador para a memória.
- ▶ Codificação:



- ▶ O campo Lista de registradores contém um bit para cada registrador visível, e controla quais registradores devem ser transferidos.
- ▶ Os registradores são armazenados ou carregados de endereços consecutivos de memória.

Instruções para transferências múltiplas

- ▶ Cada modo de endereçamento tem um comando específico em linguagem de montagem.
- ▶ Cada modo de endereçamento tem dois nomes: um para ser usado quando a instrução está sendo utilizada para implementar uma pilha e outro nome para outros usos.

Pilha	Outros	Descrição	L	U	P
LDMED	LDMIB	carrega com pré-incremento	1	1	1
LDMFD	LDMIA	carrega com pós-incremento	1	1	0
LDMEA	LDMDB	carrega com pré-decremento	1	0	1
LDMFA	LMDMA	carrega com pós-decremento	1	0	0
STMFA	STMIB	armazena com pré-incremento	0	1	1
STMEA	STMIA	armazena com pós-incremento	0	1	0
STMFD	STMDB	armazena com pré-decremento	0	0	1
STMED	STMDA	armazena com pós-decremento	0	0	0

Instruções para transferências múltiplas

Os formatos dos comandos LDM e STM em linguagem de montagem são:

LDM*modo*{*cond*} *Rn*{!}, *lista_de_registradores*

STM*modo*{*cond*} *Rn*{!}, *lista_de_registradores*

Exemplos:

```
stmia    r0,{r0-r15}    @ salva todos os registradores na memória
stmfd    sp!,{r0,r1,r3} @ empilha três registradores
ldmfd    sp!,{r1-r3,r14} @ desempilha quatro registradores,
                               @ r1, r2, r3 e r14
```

Instruções para transferências múltiplas

O montador aceita também comandos PUSH e POP similares aos utilizados pelo processador LEG. Nesse formato, o registrador base é implícito (SP) e o modo de endereçamento, também implícito, é (FD):

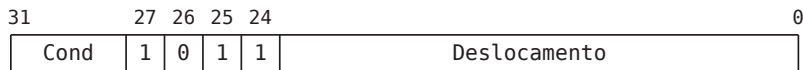
```
PUSH{cond}  lista_de_registradores
```

```
POP{cond}   lista_de_registradores
```

Exemplos:

```
push    {r0-r8}      @ empilha registr. r0 a r8, incondicional
popgt   {r0,r1,r3}   @ desempilha três registradores, condicional
pop     {r0-r3,r9}   @ desempilha cinco registr.: r0 a r3, e r9
```

- ▶ O processador ARM não utiliza a pilha para armazenar o endereço de retorno em chamadas de procedimento.
- ▶ O endereço de retorno é armazenado, no momento da chamada, no registrador ligador (r14, ou lr).
- ▶ A Figura abaixo mostra a codificação da instrução de chamada de procedimento.



Procedimentos

- ▶ O comando em linguagem de montagem para a instrução de chamada de procedimento é BL (do inglês *branch and link*).
- ▶ O formato geral do comando em linguagem de montagem é igual ao comando de desvio com endereço alvo constante:

BL{*cond*} *endereço*

Exemplo:

```
blpl calcula           @ chamada de procedimento condicional
bl  imprime           @ chamada de procedimento incondicional
...
calcula:              @ um procedimento
...
imprime:              @ outro procedimento
...
```

- ▶ No ARM não existe uma instrução específica para retorno de procedimento.
- ▶ Como o endereço de retorno está no registrador `lr`, basta executar um desvio por registrador.

Exemplo:

```
calcula:                               @ um procedimento
    ...
    bx    lr                            @ retorna do procedimento
```


Escreva um procedimento `ordena` para ordenar um vetor de inteiros com sinal. O endereço do vetor é dado no registrador `r0` e o número de elementos no registrador `r1`.

```
void Ordena(int a[], int compr) {
    int i,j,val;
    for (i=1; i<compr; i++) {
        val=a[i];
        for (j=i-1; j>=0; j--) {
            if (a[j]<=val) break;
            a[j+1]=a[j];
        }
        a[j+1]=val;
    }
}
```

Solução

```
@ *****
@ ordena
@ *****
@ Ordena vetor de inteiros com sinal
@ entrada: endereço do vetor em r0, número de bytes em r1 (r1>0)
@ saída: vetor ordenado crescentemente
@ destrói: r2,r3,r4,r5,r6,r7 e flags
```

ordena:

```
for_i:
    mov    r2,#1           @ r2 é variável i
laco_for_i:
    cmp    r2,r1           @ i<compr?
    bxge   lr              @ se não, então terminou, retorna
    ldr    r4, [r0,r2,lsl#2] @ val = a[i], r4 é variável val
for_j:
    mov    r5,r2           @ j=i, r5 é variável j
```

```
laco_for_j:
    subs    r5,r5,#1           @ j--
    bmi     fim_for_j         @ se j>=0 é falso desvia, terminou for j
    ldr     r6,[r0,r5,ls1#2]   @ r6 é a[j]
    cmp     r6,r4             @ if (a[j] <= val)
    ble     fim_for_j         @ break, encontramos a posição de val
    add     r7,r5,#1          @ r7 tem j+1
    str     r6,[r0,r7,ls1#2]   @ a[j+1]=a[j]
    b       laco_for_j        @ continua o for j

fim_for_j:
    add     r6,r5,#1          @ r7 tem j+1
    str     r4,[r0,r6,ls1#2]   @ a[j+1]=val, elemento val na posição
    add     r2,r2,#1          @ i++
    b       laco_for_i        @ continua o for i
```

Variáveis locais a procedimentos

- ▶ Para facilitar a implementação de variáveis locais armazenadas na pilha podemos manter um *apontador de quadro* para a chamada.
- ▶ Qualquer registrador de propósito geral do ARM pode ser utilizado como apontador de quadro, mas normalmente o registrador r11 é usado para esse fim, e o montador GNU-ARM aceita também fp como outro nome para r11.

Exemplo:

```
proc_folha:
```

```
    push    {fp}           @ salva apontador quadro anterior
    mov     fp, sp         @ prepara o quadro
    sub     sp,#8          @ aloca duas variáveis locais
    ...
    ...                   @ corpo do procedimento, não mostrado
    ...                   @ var. locais acessadas com fp-4 e fp-8
    mov     sp, fp         @ desaloca variáveis locais
    pop     {fp}           @ restaura apontador anterior
    bx     lr              @ e retorna
```