# Linguagens de montagem
# ARM - Tópicos

Ricardo Anido
Instituto de Computação
Unicamp

# Multiplicação

- As instruções MUL (multiplicação) e MLA (multiplicação com acumulação) multiplicam os valores de dois registradores e armazenam o resultado (em 32 bits) também em um registrador.
- As duas instruções funcionam tanto para operandos com sinal como para operandos sem sinal.

# Multiplicação

O formato dos comandos em linguagem de montagem é:

$$\begin{array}{ll} MUL\{cond\}\{S\} & Rd,\ Rm,\ Rs \\ MLA\{cond\}\{S\} & Rd,\ Rm,\ Rs,\ Rn \end{array}$$

onde

- *cond* é um sufixo de condição (eq, ne, etc.)
- S indica se a instrução deve alterar os bits de condição. Se presente, a instrução altera os bits de condição N e Z de acordo com o resultado da operação; se ausente os bits de condição não são alterados.
- Rd indica o registrador onde o resultado é armazenado. Para a instrução MUL o resultado é `Rm * Rs`; para a instrução MLA o resultado é `Rm * Rs + Rn`
- Os registradores Rd e Rm devem ser distintos

# Exemplos de uso, MUL e MLA

```
muls r1,r2,r3       @ multiplicação entre números com ou sem sinal
                    @ r1 = r2 * r3, altera bits de condição
mla  r0,r5,r2,r3    @ multipl. com acumulação, com ou sem sinal
                    @ r0 = r5 * r2 + r3
```

# Matrizes

Matrizes (vetores de duas dimensões) são armazenadas na memória do computador em endereços consecutivos: devem ser *linearizadas*.

- linearização por linhas: as linhas da matriz são colocadas em sequência na memória, elementos consecutivos nas linhas da matriz estão em posições consecutivas na memória;
- linearização por colunas: as colunas da matriz são colocadas em sequência, elementos consecutivos nas colunas da matriz estão em posições consecutivas na memória.

# Linearização de matriz

| | | |
|---|---|---|
| a | b | c |
| d | e | f |

(a) Uma matriz 2 x 3

*endereços crescentes* →

(b) armazenamento por linhas

(c) armazenamento por colunas

# Linearização de matriz

Na linearização por linhas, para determinar o endereço do elemento `mat[i][j]` é necessário conhecer:

- o número de elementos de cada linha, `NCOL`
- o número de bytes de cada elemento, *n_bytes_elem*
- o endereço do primeiro elemento da matriz `mat` na memória.

O endereço do elemento `mat[i][j]` na linearização por linhas é:

$$ender(\texttt{mat}[i,j]) = ender(\texttt{mat}[0][0]) + (i \times \texttt{NCOL} + j) \times n_bytes_elem$$

# Problema

Traduza para linguagem de montagem ARM o seguinte trecho de programa C, que contém uma atribuição a um elemento de um vetor de duas dimensões:

```
#define NLIN 10
#define NCOL 20
int a, b, mat[NLIN][NCOL];
...
    a = 5;
    b = 7;
    mat[a][b] = -1;
...
```

# Solução

```
@ define constantes
    .equ   NLIN,10
    .equ   NCOL,20
@ aloca espaço para as variáveis
a:    .skip 4
b:    .skip 4
mat:  .skip 4*NLIN*NCOL
...
     ldr     r3,=a                    @ endereço de a
     mov     r0,#5                    @ atribui o valor 5
     str     r0,[r3]                  @ à variável a
     ldr     r4,=b                    @ endereço de b
     mov     r1,#7                    @ atribui o valor 7
     str     r1,[r4]                  @ à variável b
     ldr     r2,=mat                  @ endereço de mat
     mov     r5,#NCOL
     mla     r0,r5,r0,r1              @ calcula a*NCOL+b
     mvn     r4,#0                    @ valor -1
     str     r4,[r2, r0, asl #2]      @ atribui ao elemento mat[a,b]
...
```

# Convenções padrão do ARM para C/C++

| Tipo C/C++ | Representação ARMv7 |
|---|---|
| `char` | `byte sem sinal` |
| `unsigned char` | `byte sem sinal` |
| `signed char` | `byte com sinal` |
| `short` | `meia palavra com sinal` |
| `unsigned short` | `meia palavra sem sinal` |
| `int` | `palavra com sinal` |
| `unsigned int` | `palavra sem sinal` |
| `long` | `palavra com sinal` |
| `unsigned long` | `palavra sem sinal` |
| `long long` | `palavra dupla com sinal` |
| `unsigned long long` | `palavra dupla sem sinal` |
| `float` | `formato curto IEEE 754` |
| `double` | `formato longo IEEE 754` |

# Pilha e registradores

- A implementação da pilha é do tipo descendente, cheia, usando o registrador `sp` (`r13`) como apontador de pilha.
- A pilha deve ser alinhada a palavras.
- `r0` a `r3` são usados para passar argumentos para procedimentos ou funções e para retornar valores de funções.
- `r0` a `r3` podem também ser usados para armazenar dados locais, e seus valores não precisam ser preservados em chamadas de procedimentos.
- `r4` a `r14` podem ser usados para armazenar variáveis locais, mas devem ser preservados em chamadas de procedimento.
- Os bits de condição C, N, V e Z são indefinidos na entrada e no retorno de procedimento; não necessitam ser preservados entre chamadas de procedimento.

# Procedimentos e parâmetros

- Para agilizar as chamadas de procedimentos, ao invés de usar a pilha, `r0` a `r3` são usados para passar os argumentos,
- Os registradores são associados aos argumentos na ordem de seus números, ou seja, `r0` armazena o valor do primeiro argumento, `r1` armazena o valor do segundo argumento, e assim por diante.
- Argumentos de tipos fundamentais com duas palavras (como `double` ou `long long`) são passados em dois registradores consecutivos (isto é, `r0` e `r1`, ou `r1` e `r2`).
- Se os argumentos para um procedimento não podem ser acomodados em em quatro registradores, a pilha deve ser usada. As primeiras quatro palavras da pilha, correspondendo aos primeiros quatro registradores (argumentos), devem ser reservadas.