# ARM

# MC404

# Laço For

```
for(i=0; i<100; i++)
   a += i;
```

**Suponha r0 = i**

**Suponha r1 = a**

```
      MOV r0, #0
      MOV r2, #100
Loop  ADD r1, r1, r0
      ADD r0, r0, #1
      CMP r0, r2
      BNE Loop
```

# Laço While

```
while x != y
{
   x += 2;
   y += 3;
}
```

**Suponha r0 = x**

**Suponha r1 = y**

```
Loop CMP r0, r1
     BEQ FimWhile
     ADD r0, r0, #2
     ADD r1, r1, #3
     BAL Loop
FimWhile
```

```
     CMP r0, r1
     BEQ FimWhile
Loop ADD r0, r0, #2
     ADD r1, r1, #3
     CMP r0, r1
     BNE Loop
FimWhile
```

# Execução Condicional

```
if (x > 10)
   y += 7;
else
   y -= 7;
```

**Suponha r0 = x**

**Suponha r1 = y**

```
MOV r2, #10
CMP r0, r2
BLE Else
ADD r1, r1, #7
BAL FimIf
Else  SUB r1, r1, #7
FimIf …
```

# Ordem dos bytes na memória

- Temos duas formas de armazenar uma palavra (32 bits = 4 bytes) na memória
  - Big Endian: Byte mais significativo primeiro
  - Little Endian: Byte menos significativo primeiro
- Exemplo:
  - 0x12345678

| Endereço | Big Endian | Little Endian |
|---|---|---|
| 0 | 0x12 | 0x78 |
| 1 | 0x34 | 0x56 |
| 2 | 0x56 | 0x34 |
| 3 | 0x78 | 0x12 |

# Impacto

- Não importa o formato se você vai ler o que escrever
  - Mesmo programa
  - Mesmo processador
- ARM é little endian por padrão
  - Cuidado quando for se comunicar com outros dispositivos que sejam Big Endian
  - Ex.: cabeçalhos do protocolo TCP/IP são codificados como Big Endian

# Funções

- Funções são trechos de código que executam uma tarefa específica e são organizados separadamente para facilitar reuso e legibilidade.

- Permite divisão de tarefas entre diversos desenvolvedores e também o uso de bibliotecas.

- Necessitam de um conjunto de convenções para funcionarem entre diversos programas:

  - Como passar parâmetros?

  - Que registradores salvar?

  - Como retornar o resultado?

  - Para onde retornar ao final?

# Exemplo

- Implementar uma função que some todos os elementos de dois vetores e retorne o total
- Como passar parâmetros?
  - Endereço do vetor em r0, tamanho em r1 (a função pode usar de r0-r3)
- Que registradores salvar?
  - A função deve preservar r4-r14
- Como retornar o resultado?
  - Retornar em r0-r3 os valores necessários
- Para onde retornar ao final?
  - Retornar para o endereço seguinte da chamada da função

# Resposta

```
Inicio
        ADR r0, vetor1
        MOV r1, #tam1
        BL  Soma
        MOV r4, r0
        ADR r0, vetor2
        MOV r1, #tam2
        BL  Soma
        ADD r4, r4, r0
        END

vetor1  DCD 7,8,15,10
vetor2  DCD 19,4,3,5,9,1
tam1    EQU 4
tam2    EQU 6
```

```
Soma
        MOV  r2, #0
        LDR  r2, [r0]
        SUBS r1, r1, #1
        BEQ  FimSoma
Loop
        LDR  r3, [r0 + #4]!
        ADD  r2, r2, r3
        SUBS r1, r1, #1
        BNE  Loop
FimSoma
        MOV  r0, r2
        MOV  PC, LR
```

# Variáveis em Memória

- Programas utilizam a memória para armazenar variáveis.

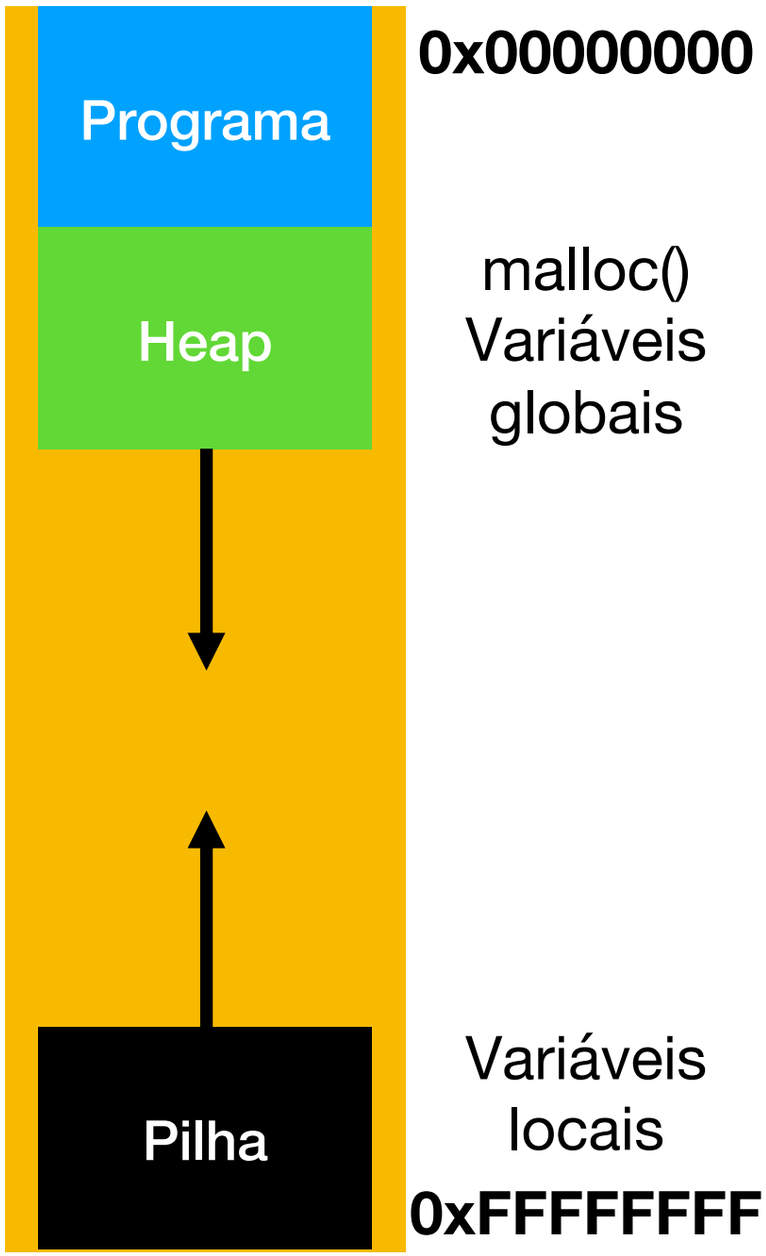
- Normalmente, o compilador é o responsável por organizar a memória e garantir que cada variável tenha seu espaço correto.

- Quando programamos em assembly, é nossa tarefa garantir o espaço de cada variável, como no código anterior com vetor1 e vetor2.

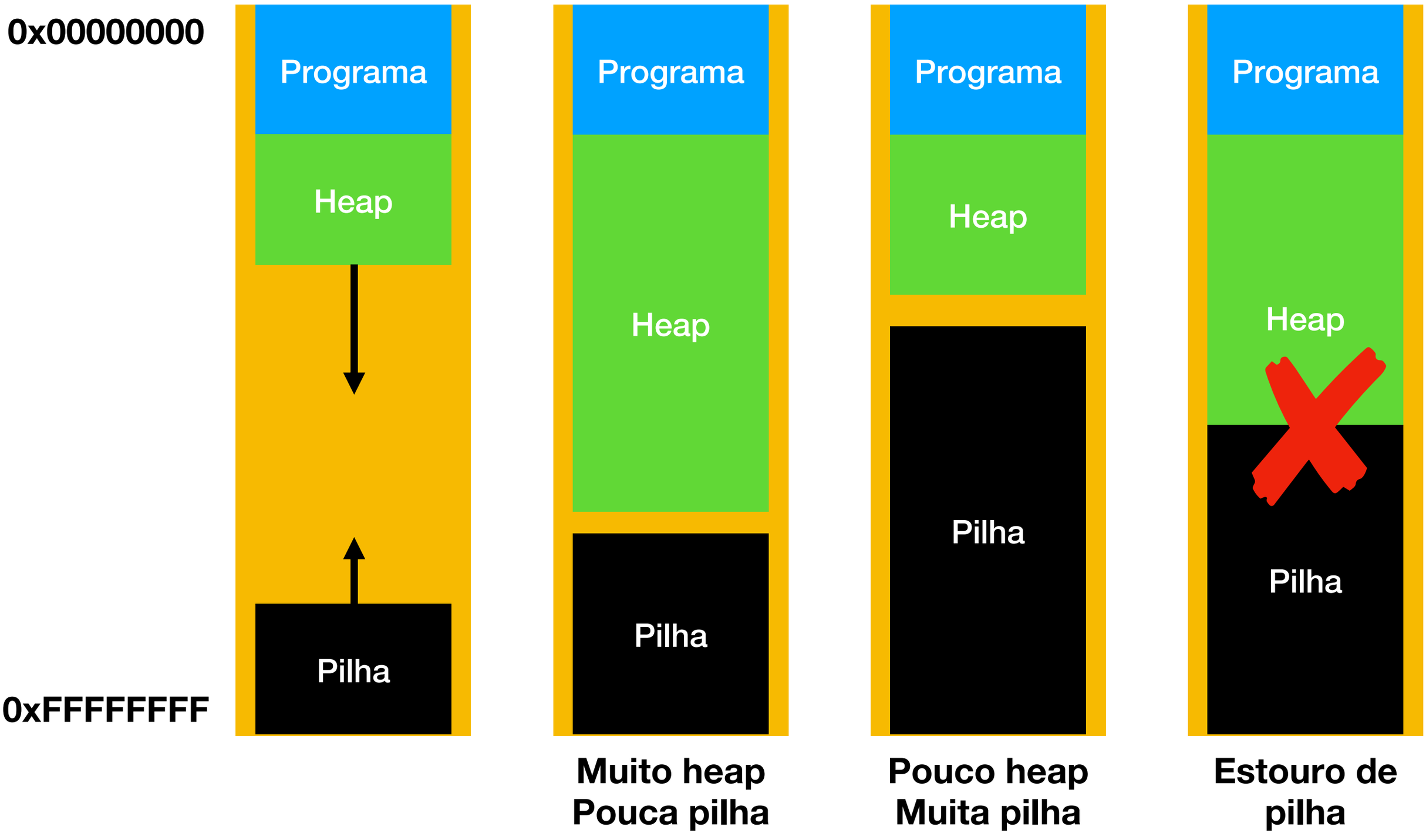
- Existem dois lugares na memória para alocação de variáveis:

  - Heap: para variáveis globais (vetor1, vetor2)

  - Pilha: para variáveis locais



# Heap e Pilha têm tamanhos variáveis



# Onde ficam as variáveis?

## Heap ou Pilha

```
Inicio
        ADR r0, vetor1
        MOV r1, #tam1
        BL  Soma
        MOV r4, r0
        ADR r0, vetor2
        MOV r1, #tam2
        BL  Soma
        ADD r4, r4, r0
        END

vetor1  DCD 7,8,15,10
vetor2  DCD 19,4,3,5,9,1
tam1    EQU 4
tam2    EQU 6
```

```
Soma
        MOV  r2, #0
        LDR  r2, [r0]
        SUBS r1, r1, #1
        BEQ  FimSoma
Loop
        LDR  r3, [r0 + #4]!
        ADD  r2, r2, r3
        SUBS r1, r1, #1
        BNE  Loop
FimSoma
        MOV  r0, r2
        MOV  PC, LR
```

# Preservando registradores na pilha

- Ao implementar seu código, todo registrador a ser preservado deve ser guardado na pilha.

- Utilizar instruções STM e LDM que armazenam/recuperam múltiplos registradores de uma só vez
  - LDM<sufixo> r1, {r0, r2, r3}
  - STM<sufixo> r1, {r0, r2, r3}

| Sufixo | Significado |
|---|---|
| IA | Increase After |
| IB | Increase Before |
| DA | Decrease After |
| DB | Decrease Before |
| FA | Full Ascending |
| FD | Full Descending |
| EA | Empty Ascending |
| ED | Empty Descending |

# Exemplo

- Suponha rX = 10X (r7 = 70)
- R13 = 144 (SP)
- **STMFD r13!, {r4-r7}**
- Use os registradores 4-7 como quiser
- **LDMFD r13!, {r4, r7}**

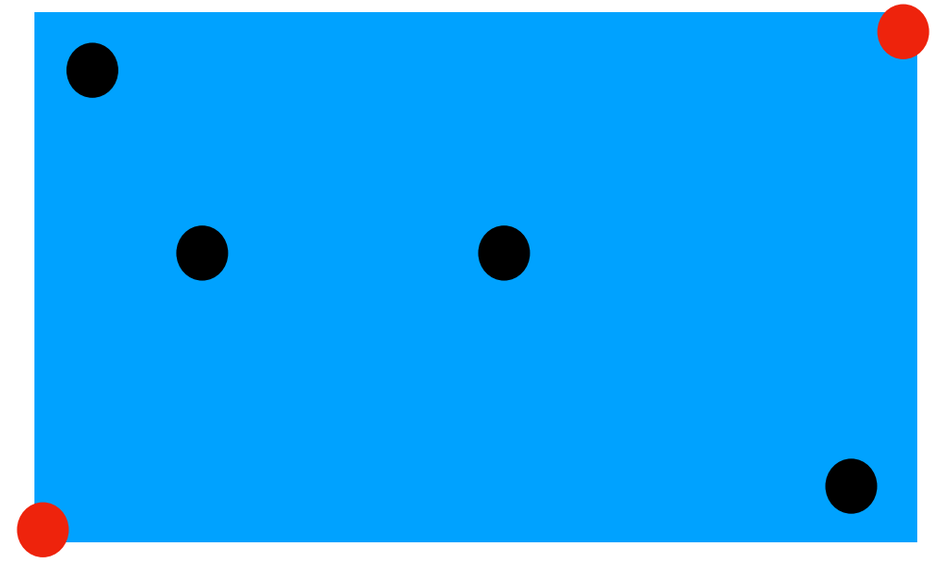
| Endereço | Valor | |
|---|---|---|
| 100 | | |
| 104 | | |
| 108 | | |
| 112 | | |
| 116 | | |
| 120 | | |
| 124 | | |
| 128 | 70 | ← SP |
| 132 | 60 | |
| 136 | 50 | |
| 140 | 40 | |
| 144 | 17 | ← SP |

**SP inicial**

# Exercício

- Dado um vetor de pontos onde cada um é representado pelas coordenadas X e Y. Encontre o retângulo que contém todos os pontos do vetor. Suponha coordenadas inteiras. Não pode ter pontos na borda do retângulo. Represente o retângulo através dos dois pontos vermelhos da figura (inferior esquerdo e superior direito).

- Crie as funções:
  - AjustaBordaInfEsq
  - AjustaBordaSupDir



- Que recebem o ponto atual da borda (vermelho) e um ponto e atualiza a própria borda. Chame as duas funções para cada um dos pontos.

# Resumo ARM

add x, y, z

| Mnemônico | Operação |
|---|---|
| add | X = Y + Z |
| adc | X = Y + Z + carry |
| sub | X = Y - Z |
| sbc | X = Y - Z + carry - 1 |
| rsb | X = Z - Y |
| rsc | X = Z - Y + carry - 1 |
| cmp | Y - Z |
| cmn | Y + Z |
| tst | Y AND Z |
| teq | Y XOR Z |
| and | X = Y AND Z |
| eor | X = Y XOR Z |
| orr | X = Y OR Z |
| bic | X = Y AND NOT Z |
| mov | X = Y |
| mvn | X = NOT Y |

| Cond | Descrição |
|---|---|
| EQ | igual |
| NE | diferente |
| HS / CS | >= sem sinal |
| LO / CC | < sem sinal |
| MI | negativo |
| PL | positivo ou zero |
| VS | overflow |
| VC | sem overflow |
| HI | maior sinalizado |
| LS | <= sem sinal |
| GE | >= |
| LT | < |
| GT | > |
| LE | <= |
| AL | Sempre |
| NV | Reservado |

| Info | Descrição |
|---|---|
| Regist. | r0, …, r15 |
| r15 | PC |
| r14 | LR |
| r13 | SP |
| Salto | B<cond> |
| Chamar função | BL<cond> |
| Retornar função | mov pc,lr |
| Ler Memória | LDR r0,[r1,#20] |
| Escrever Memória | STR r0,[r1,#12] |
| Constante | #20 |