

Sub-rotinas

Arquitetura AARCH64

João Canas Ferreira

Março 2019

Assuntos

- 1 Sub-rotinas: aspetos gerais
- 2 Organização de sub-rotinas
- 3 Comunicação C \leftrightarrow Assembly
- 4 Exemplos

Decomposição funcional

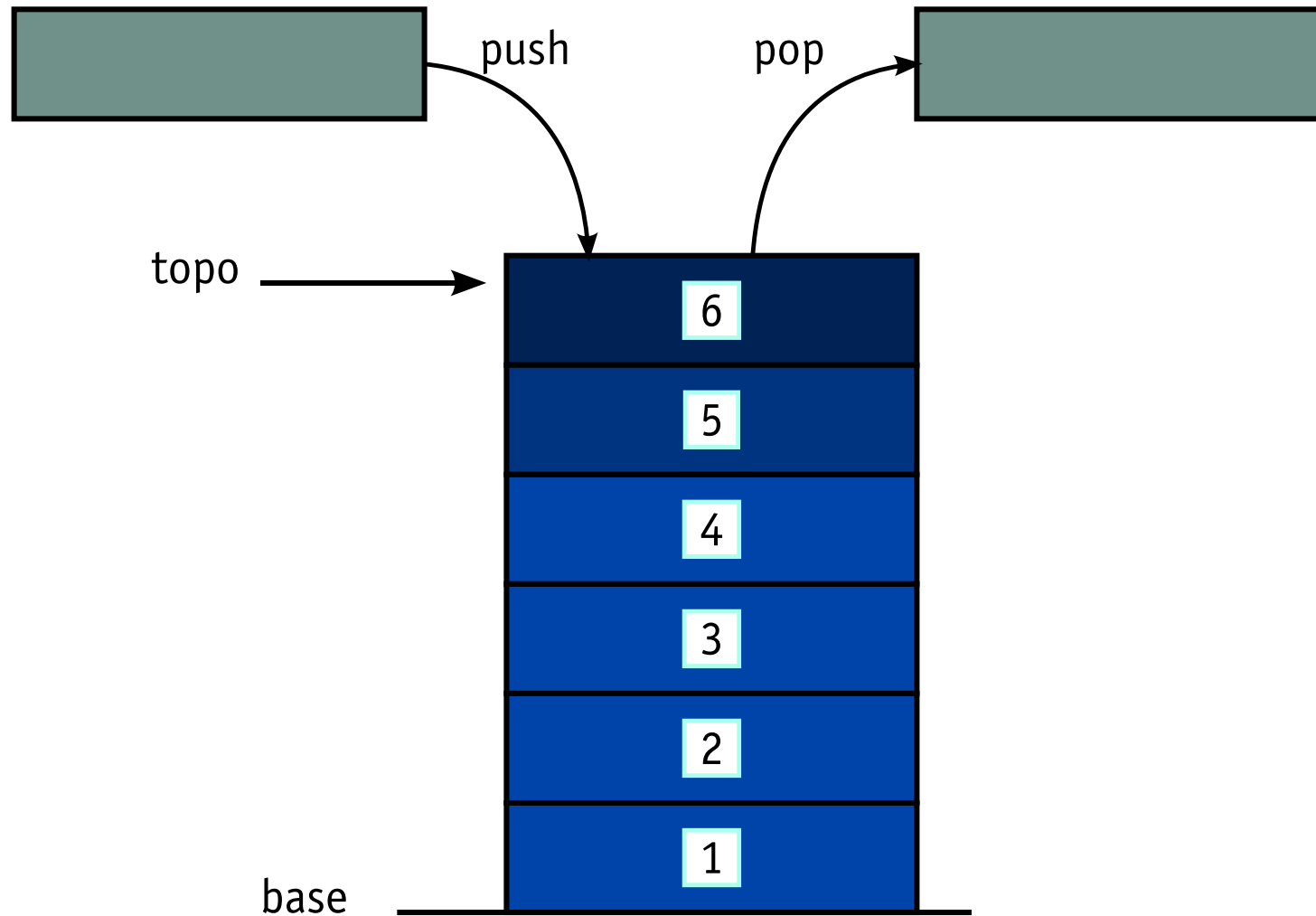
- Programar também é **gerir complexidade** (da especificação e da implementação)
- A decomposição funcional envolve:
 - projetar programa antes de iniciar a codificação
 - decompor tarefas maiores em tarefas mais pequenas (sub-rotinas)
 - criar uma estrutura hierárquica de sub-rotinas
 - testar sub-rotinas individualmente
- A utilização de sub-rotinas é uma forma de *reutilização de código*
- Sub-rotinas podem ser:
 - procedimentos: a sua invocação não produz um valor
 - funções: a sua invocação produz um valor
- Em *assembly* não existe distinção formal entre procedimentos e funções: a designação usada é *procedure* (procedimento)
- CPU suporta sub-rotinas através das instruções: BL/BLR e RET

Interoperabilidade: Convenção de invocação de sub-rotinas

- Uma convenção de invocação de sub-rotinas [CIS] (*Procedure Call Standard*) define como é que sub-rotinas *compiladas separadamente* podem “trabalhar” em conjunto.
- Faz parte da Interface Binária da Aplicação (ABI=Application Binary Interface)
- Respeitar a CIS definida pela ARM para a arquitetura AArch64 implica:
 - 1 Respeitar as restrições de alinhamento da pilha
 - 2 Respeitar o tipo de uso dos registos
 - 3 Respeitar regras na representação de dados em memória (*data layout rules*)

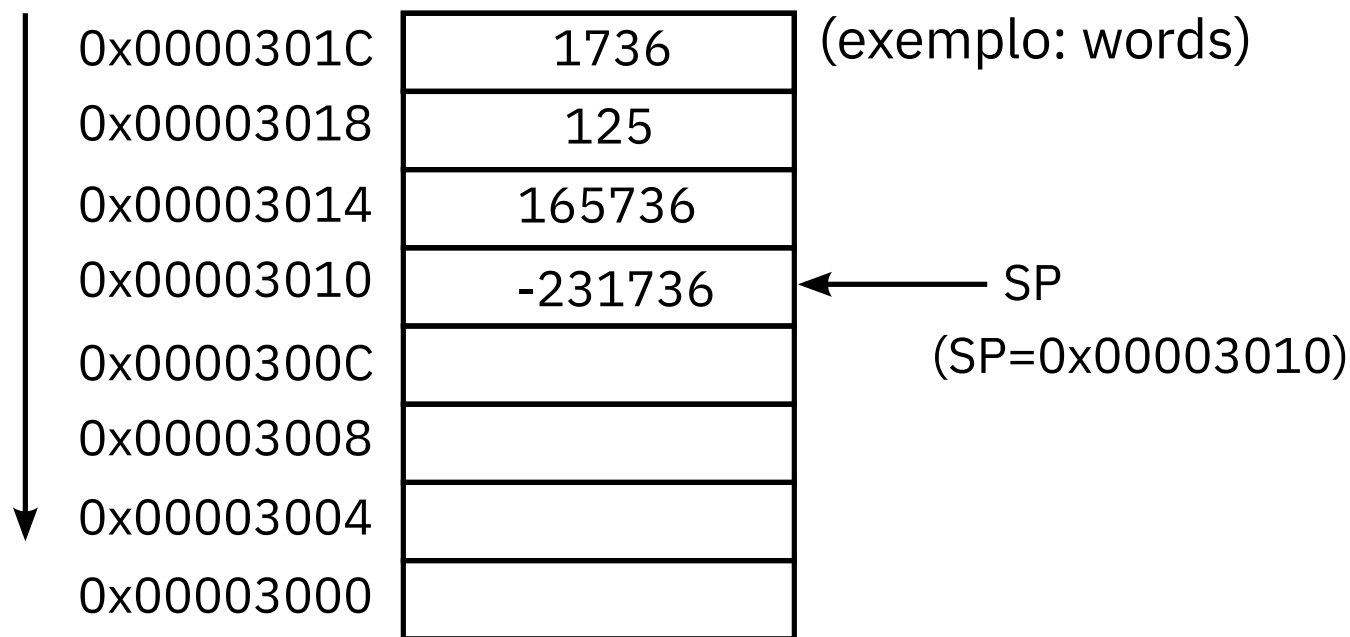
Pilha

- Durante a execução, os programas mantêm uma pilha de dados



Gestão da pilha

- Pilha: zona contígua de memória gerida segundo o princípio LIFO (*Last-In First-Out*).
- Usada para passar parâmetros (se não couberem em registos), guardar variáveis locais e preservar endereços de retorno.
- Pilha gerida através de um apontador para o topo da pilha: SP (registo reservado para esta função)



- O que está na posição 0x0000300C?

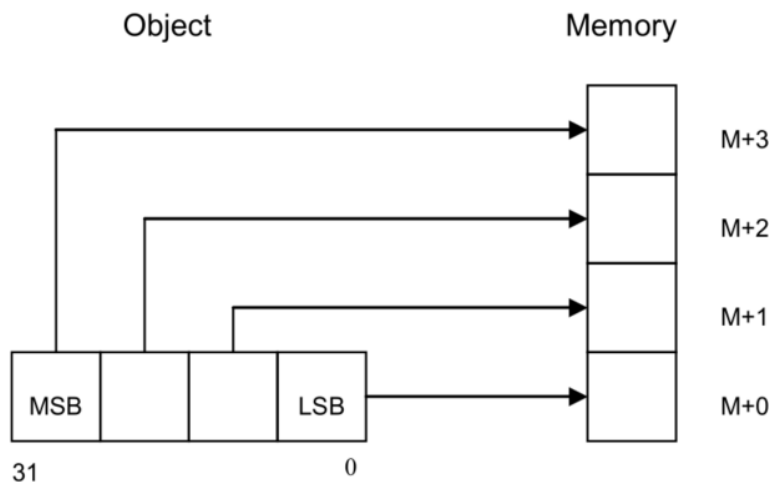
Regras para utilização da pilha

- Valor de SP deve ser sempre múltiplo de 16
- Usar modos de endereçamento apropriados

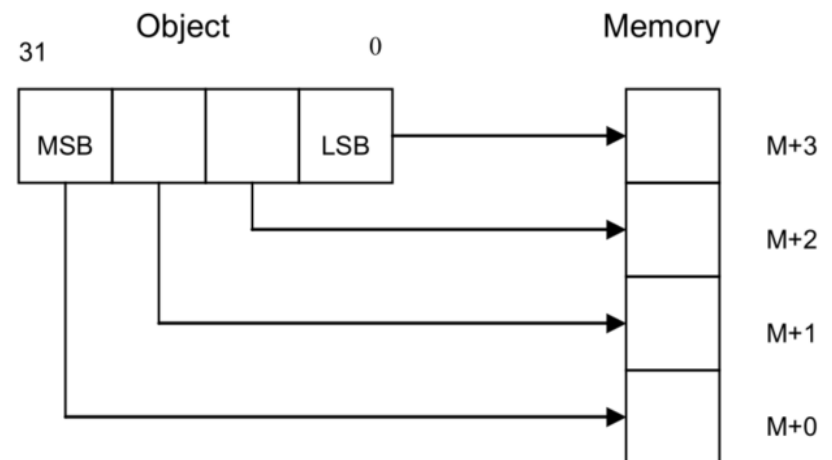
➡ Qual é o valor de W10 nos seguintes casos?

- 1 `ldr W10, [SP, #4]`
- 2 `ldrh W10, [SP, #8]`

➡ “Layout” de objetos



Little-endian



Big-endian

Assuntos

- 1 Sub-rotinas: aspetos gerais
- 2 Organização de sub-rotinas
- 3 Comunicação C \leftrightarrow Assembly
- 4 Exemplos

Colocação dos dados em memória

| Type Class | Machine Type | Byte size | Natural Alignment (bytes) |
|------------|----------------------|-----------|---------------------------|
| Integral | Unsigned byte | 1 | 1 |
| | Signed byte | 1 | 1 |
| | Unsigned half-word | 2 | 2 |
| | Signed half-word | 2 | 2 |
| | Unsigned word | 4 | 4 |
| | Signed word | 4 | 4 |
| | Unsigned double-word | 8 | 8 |
| | Signed double-word | 8 | 8 |
| | Unsigned quad-word | 16 | 16 |
| | Signed quad-word | 16 | 16 |
| Pointer | Data pointer | 8 | 8 |
| | Code pointer | 8 | 8 |

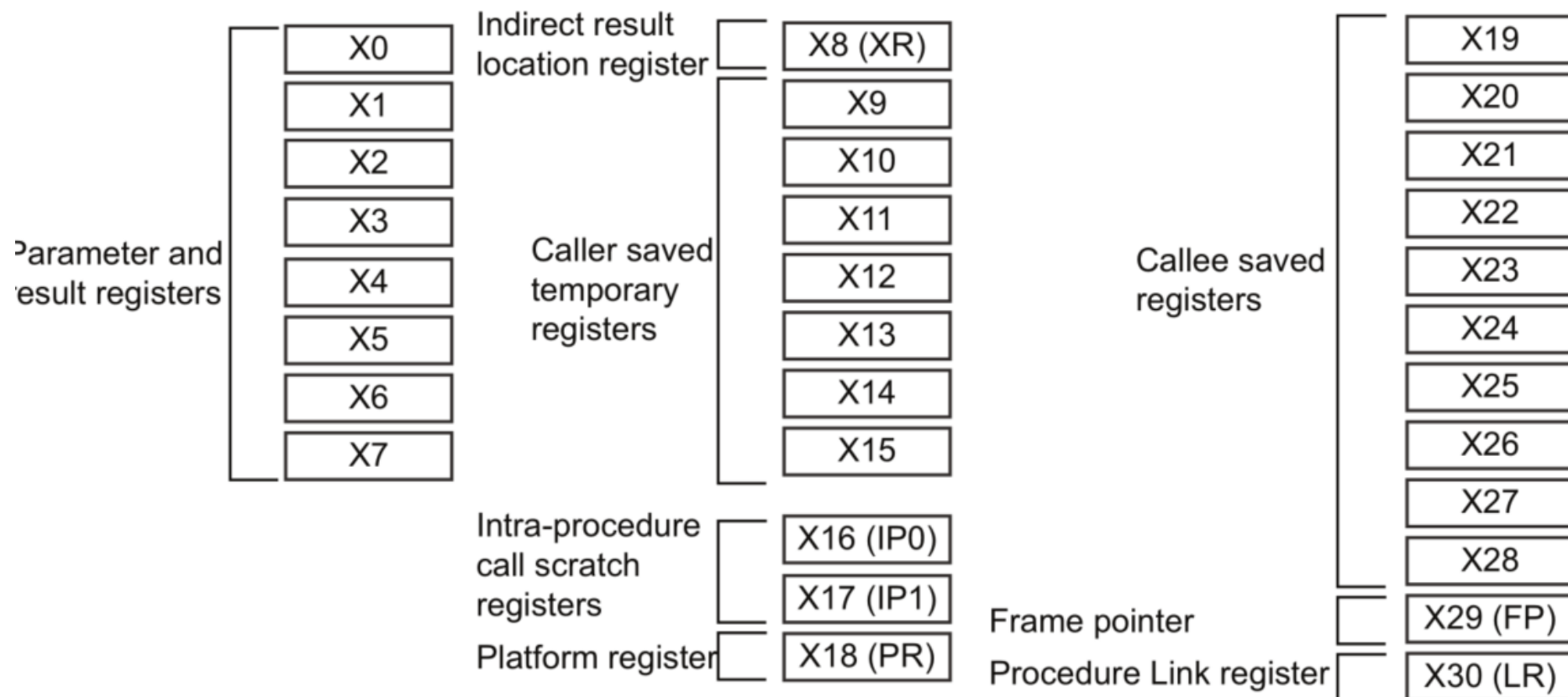
Utilização de registos (1/3)

| Register | Special | Role in the procedure call standard |
|-----------|---------|---|
| SP | | The Stack Pointer. |
| r30 | LR | The Link Register. |
| r29 | FP | The Frame Pointer |
| r19...r28 | | Callee-saved registers |
| r18 | | The Platform Register, if needed; otherwise a temporary register. See notes. |
| r17 | IP1 | The second intra-procedure-call temporary register (can be used by call veneers and PLT code); at other times may be used as a temporary register. |
| r16 | IP0 | The first intra-procedure-call scratch register (can be used by call veneers and PLT code); at other times may be used as a temporary register. |
| r9...r15 | | Temporary registers |
| r8 | | Indirect result location register |
| r0...r7 | | Parameter/result registers |

Utilização de registos (2/3)

- Para a arquitetura de 64 bits, $R\langle n \rangle = X\langle n \rangle$!
 - $x0-x7$: passar argumentos (na chamada) e resultados (no retorno); podem ser alterados pela sub-rotina.
 - $x9-x15$: podem ser usados livremente pela sub-rotina.
 - $x19-x28$: devem ser preservados pela sub-rotina (“callee”).
 - $x8, x16-x18$: não usar!
 - SP contém endereço do “topo” da pilha (endereço mais baixo);
 - LR contém endereço de retorno (link register, $x30$);
 - FP contém endereço para a *moldura* da sub-rotina que invocou esta (“caller”); registo $x29$;
 - *moldura*: região da pilha reservada por cada invocação de uma sub-rotina para guardar valores temporariamente.
- Casos não tratados nesta u.c.: argumentos ou resultado não cabem nos registos disponíveis.

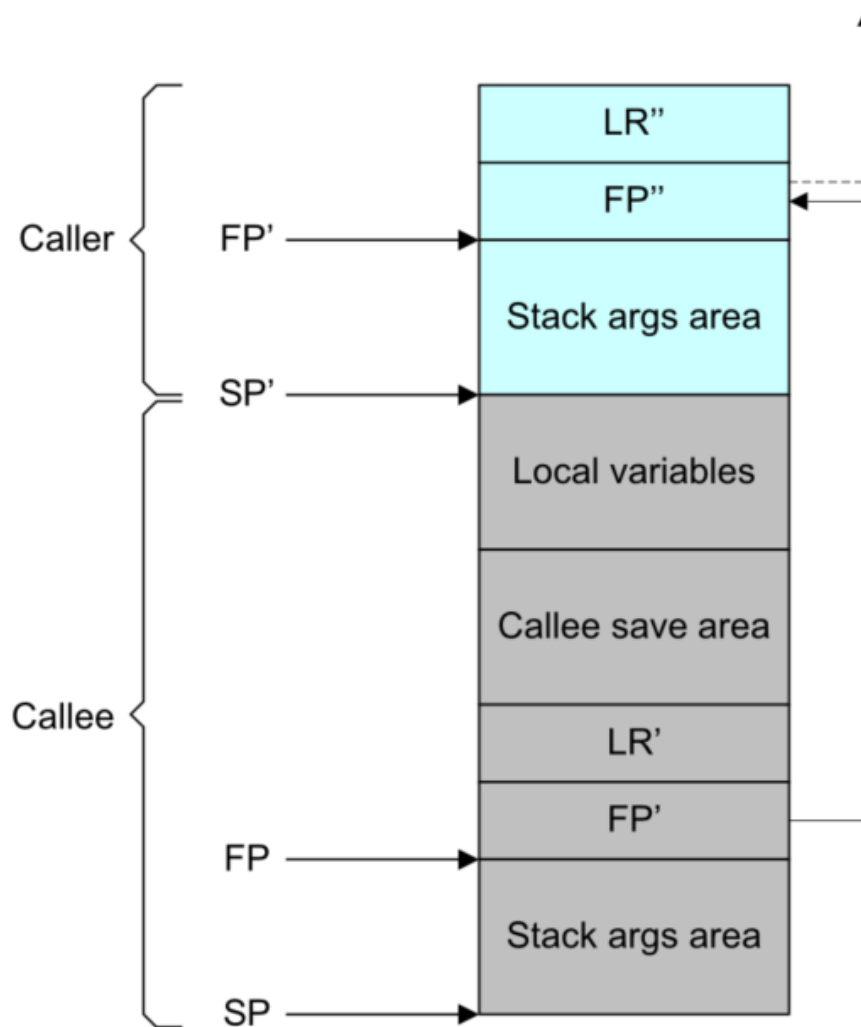
Utilização de registos (3/3)



Regras para invocação de uma sub-rotina

- Em “caller”:
 - 1 Os argumentos da função são colocados por ordem (da esquerda para a direita) nos registos x0–x7.
 - 2 A instrução BL é usada para invocar a sub-rotina.
- Na sub-rotina invocada (“callee”):
 - 1 Construir a moldura (“frame record”).
 - 2 Guardar valores de FP e LR na moldura.
 - 3 Fazer os cálculos observando as regras de utilização de registos.
 - 4 Colocar o resultado no registo x0.
 - 5 Recuperar os valores originais de FP e LR.
 - 6 Terminar a execução da sub-rotina com RET (equivalente: BR x30).
- Execução continua em “caller”.

Organização da moldura



➡ “Stacks arg area” não é usada nesta u.c. (todos os argumentos são passados em registos)

Assuntos

- 1 Sub-rotinas: aspetos gerais
- 2 Organização de sub-rotinas
- 3 Comunicação C \leftrightarrow Assembly**
- 4 Exemplos

Invocação a partir de C

- ▢ Declarar a sub-rotina na sintaxe de C++ [sub-rotina externa] (possivelmente em “header file” com extensão *.h)

```
extern "C" tipo_resultado nome_func (arg1, arg2, ..., argN);
```

- ▢ Usar normalmente como qualquer função de C++:

```
res = nome_func (12, x, ..., y);
```

- ▢ Os tipos de dados devem respeitar a correspondência da página 18.

- ▢ Para usar uma rotina de C:

- 1 Colocar argumentos nos registos corretos
- 2 Invocar a sub-rotina com BL
- 3 Usar o resultado

- ▢ Variáveis globais podem ser declaradas em C++ ou em assembly

Correspondência de tipos

| C/C++ Type | Machine Type |
|----------------|--|
| char | unsigned byte |
| unsigned char | unsigned byte |
| signed char | signed byte |
| [signed] short | signed halfword |
| unsigned short | unsigned halfword |
| [signed] int | signed word |
| unsigned int | unsigned word |
| [signed] long | signed word or signed double-word |
| unsigned long | unsigned word or unsigned double-word |

➡ Apontadores são valores de 64 bits.

Assuntos

- 1 Sub-rotinas: aspetos gerais
- 2 Organização de sub-rotinas
- 3 Comunicação C \leftrightarrow Assembly
- 4 Exemplos

Invocar sub-rotina em assembly

Ficheiro: t1.c _____

```
#include <stdio.h>
extern int add2(int);
```

```
int main(void)
{
    int x = add2(10);
    printf("%d\n", x);
    return 0;
}
```

Ficheiro: add2.s _____

```
.global add2
.type add2, %function
```

```
.text
```

```
add2:    stp x29,x30,[sp,#-16]!
         mov x29,sp
         add w0,w0,#2
         ldp x29,x30,[sp],#16
         ret
```

▣➡ Novas instruções:

stp store pair of registers

ldp load pair of registers

Versões em C e assembly (1/4)

Ficheiro: max.c _____

```
int func(int a, int b)
{
    int m;
    if (a>b)
        m=a;
    else
        m=b;
    return m;
}
```

Ficheiro: max.s _____

```
.text
.global func
.type func, %function
func:
    stp    x29, x30, [sp, #-16]!
    mov    x29, sp
    cmp    w0, w1
    csel    w0, w0, w1, ge
    ldp    x29, x30, [sp], #16
    ret
```

Versões em C e assembly (2/4)

Ficheiro: test.c _____

```
long test(long a, long b)
{
    long m;
    if (a>b)
        m=a/b;
    else
        m=b/a;
    return m;
}
```

gerado por compilador →

Ficheiro: test.s _____

```
.text
.global test
.type test, %function
test:
    stp    x29, x30, [sp, #-16]!
    mov    x29, sp
    cmp    x0, x1
    bgt    .L5
    sdiv   x0, x1, x0
.L1:
    ldp    x29, x30, [sp], #16
    ret
.L5:
    sdiv   x0, x0, x1
    b      .L1
```

Versões em C e assembly (3/4)

Ficheiro: loop.c _____

```
int loop(int n)
{
    int i, j;

    j = 0;
    for (i=1; i<=n; i++)
        j = j + i;
    return j;
}
```

gerado por compilador →

Ficheiro: loop.o

```
loop:
    stp    x29, x30, [sp, #-16]!
    add    x29, sp, #0    ///  
    cmp    w0, #0
    ble    .L4
    mov     w2, w0
    mov     w0, #0
    mov     w1, #1
.L3:
    add     w0, w0, w1
    add     w1, w1, #1
    cmp     w2, w1
    bge     .L3
.L1:
    ldp     x29, x30, [sp], #16
    ret
.L4:
    mov     w0, #0
    b       .L1
```

Versões em C e assembly (4/4)

Ficheiro: loop2.c _____

```
long loop2(long *vect,
           int n)
{
    int i;
    long j=0;
    for (i=0; i<n; i++)
        j = j + vect[i];
    return j;
}
```

gerado por compilador →

Constantes **não precisam** de #

Ficheiro: loop2.s _____

```
loop2:
    stp x29, x30, [sp, -16]!
    add x29, sp, 0
    cmp w1, 0
    ble .L4

    mov x2, x0
    sub w1, w1, 1
    add x0, x0, 8
    add x3, x0, x1, lsl 3
    mov x0, 0

.L3:
    ldr x1, [x2], 8
    add x0, x0, x1
    cmp x2, x3
    bne .L3

.L1:
    ldp x29, x30, [sp], 16
    ret

.L4:
    mov x0, 0
    b .L1
```

Utilização de variáveis globais

- Sub-rotina que retorna letra da posição “n” de uma cadeia de caracteres.

```
.arch armv8-a
.global nome
.data
.align 3
//alinhamento 8
nome:
.string "MPCP 2018/19"
```

```
.text
.align 2 // alinhamento 4
.global get_letter
.type get_letter,%function
get_letter:
    stp    x29, x30, [sp,-16]!
    mov    x29, sp
    cmp    w0, 12
    bhi    L1
    ldr     x1, =msg
    ldrb    w0, [x1, w0, sxtw]
    b       Lfim
L1:
    mov     w0, 0
Lfim:
    ldp     x29, x30, [sp], 16
    ret
```


Diretivas de declaração de dados

| Diretiva | Efeito |
|-------------------------|---|
| .byte valor{,valor} | espaço inicializado com valores de 1 byte |
| .hword valor{,valor} | espaço inicializado com valores de tipo halfword |
| .word valor {,valor} | espaço inicializado com valores de tipo word |
| .quad valor {,valor} | espaço inicializado com valores inteiros de 8 bytes |
| .string "str" | espaço inicializado com os caracteres de “str” com 0 no final |
| .space tamanho {,valor} | inicializa “tam” bytes com valor (ou 0 se valor for omitido) |

Invocar funções de C

Ficheiro: print_msg_tb.c _____

```
#include <stdio.h>
extern void
    print_msg(char *msg);

int main(void)
{
    char mensagem[]="ARMv8-A!";
    print_msg(mensagem);
    return 0;
}
```

▣ Como exemplo, print_msg acrescenta 10 ao código da 1ª letra (M → K)

Ficheiro: print_msg.s _____

```
.text
.align 2
.global print_msg
.type print_msg,%function
```

```
print_msg:
    stp    x29, x30, [sp, -16]!
    mov    x29, SP
    ldrb   w9, [x0]
    add    w9, w9, 10
    strb   w9, [x0]
    // invocar
    bl     puts
    ldp    x29, x30, [sp], 16
    ret
```

Mais variáveis globais

Ficheiro: addr_tb.c _____

```
#include <stdio.h>
unsigned long secret =
    0xaabbccdd12345678;
extern void func_addr(void);

int main(void)
{
    func_addr();
    printf("0x%lx\n", secret);
    return 0;
}
```

▢ Este exemplo imprime
0xbc f023552468ac f0

Ficheiro: addr.s _____

```
.data
.extern secret
.align 3
num: .quad 0x1234567812345678

.text
.align 2
.global func_addr
.type func_addr,%function

func_addr:
    ldr x0, =num
    ldr x1, =secret
    ldr x2, [x0]
    ldr x3, [x1]
    add x3, x3, x2
    str x3, [x1]
    ret
```