Arquitectura de Computadores I 2007/08

1º Semestre de 2007/2008

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

Arquitectura de Computadores I 2007/08

Aula 9

Caracterização das subrotinas na perspectiva do "chamador" e do "chamado"

Convenções adoptadas quanto à passagem de parâmetros

Convenções adoptadas quanto à devolução de valores

Convenções adoptadas quanto à salvaguarda de registos

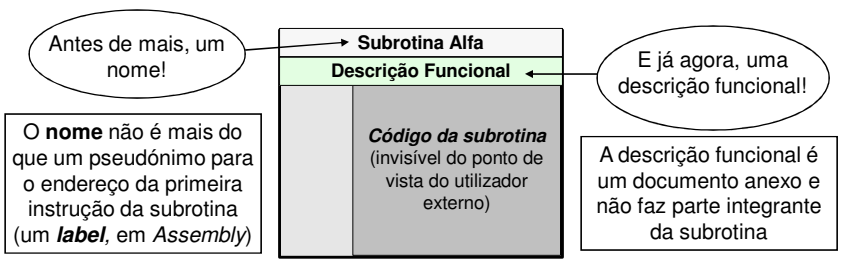
- *Caller saved versus callee saved*

Caracterização de uma subrotina na perspectiva do utilizador externo

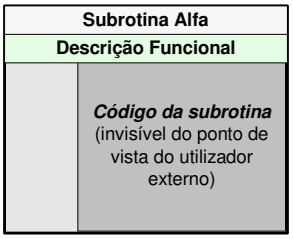
- A reusabilidade das subrotinas torna-as particularmente atractivas, em especial quando suportam funcionalidades básicas, quer do ponto de vista computacional como do ponto de vista do interface entre o computador, os periféricos e o utilizador humano.
- As subrotinas surgem assim frequentemente agrupadas em bibliotecas, a partir das quais podem ser evocadas por qualquer programa externo.
- Este facto determina naturalmente que o recurso a subrotinas escritas por outros para serviço dos nossos programas, não deverá implicar necessariamente o conhecimento dos detalhes da sua implementação
- Geralmente, o acesso ao código fonte da subrotina (conjunto de instruções originalmente escritas pelo programador) não é sequer possível, a menos que o mesmo seja tornado público pelo seu autor.

- Do ponto de vista do utilizador externo a subrotina tipo é, consequentemente, apenas uma caixa preta que encapsula uma determinada funcionalidade.

O que será então necessário acrescentar-lhe para permitir a sua utilização por terceiros?

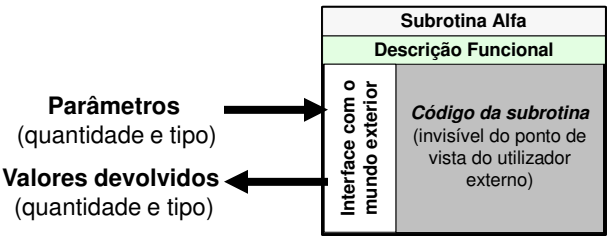


- Se a subrotina se destinar a executar uma tarefa simples, que não dependa de elementos externos e que não tenha que devolver qualquer resultado ao programa que a evocou, os elementos identificados são suficientes para permitir a sua reutilização
- Contudo, é fácil concluir que seria de grande utilidade poder "*configurar*" a subrotina por forma a que o seu comportamento pudesse reflectir alterações de uma ou mais variáveis externas. Ou seja torná-la **parametrizável**.

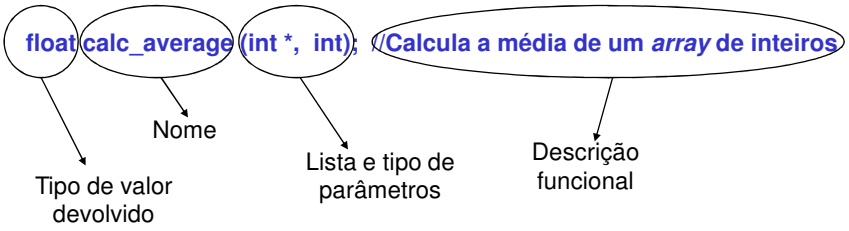


- Da mesma forma, é mais ou menos óbvio que a utilidade da subrotina aumentaria se esta tivesse possibilidade de devolver um ou mais valores (numéricos ou não) ao programa que a evocou.

Essas facilidades (possibilidade de **parametrização**, por um lado, e **devolução** de valores, por outro) são tão importantes que será necessário acrescentar ainda algo ao modelo da subrotina para o tornar completo!...



Exemplo de um protótipo de uma função em linguagem C

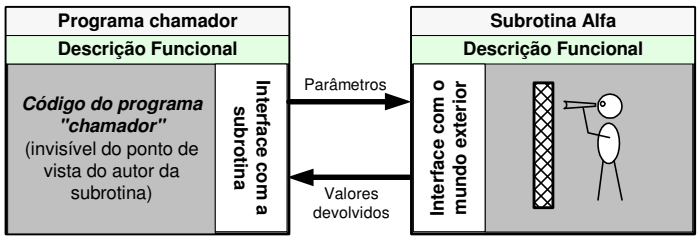


E a sua evocação a partir de um outro programa

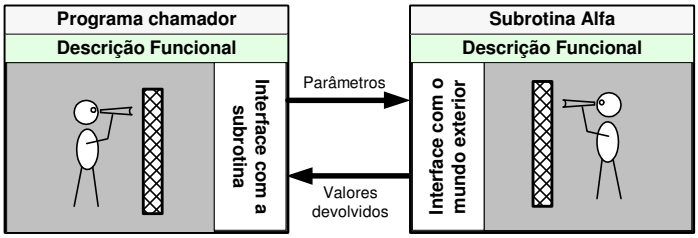
```
float av_val;
int array[200];

av_val = calc_average (array, 200);
```


Caracterização de uma subrotina na perspectiva do programador

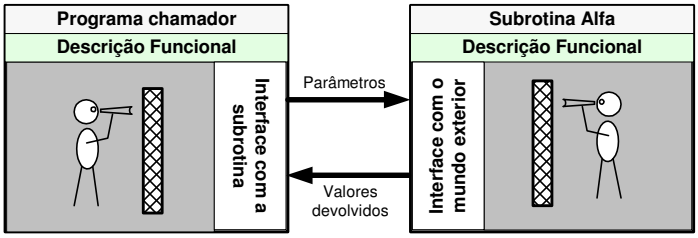
- Na perspectiva do programador, a subrotina que este tem a responsabilidade de escrever é um trecho de código isolado, com uma funcionalidade bem definida, e com um interface que ele próprio pode determinar em função das necessidades
- Em contrapartida, o facto de a subrotina ter de ser escrita para ser reutilizada implica necessariamente que o programador não conhece antecipadamente as características do programa que irá evocar o seu código.



Na realidade, a utilidade do *software* enquanto módulo reutilizável implica necessariamente que, no momento em que são escritos pelo programador, quer o programa "*chamador*" quer a subrotina "*chamada*" têm que garantir que o seu código não depende do facto de, circunstancialmente, serem conhecidos os detalhes da implementação de um ou de outro!



Torna-se assim aparente a necessidade de definir um conjunto de regras que regulem a relação entre o programa "*chamador*" e a subrotina "*chamada*", quer no que respeita à definição do interface entre ambos, como no que respeita aos princípios que assegurem uma "*sã convivência*"!



Regras a definir entre o programa "*chamador*" e a subrotina "*chamada*"

Ao nível do interface:

- Como passar parâmetros do "*chamador*" para o "*chamado*", quantos e onde;
- Como receber, do lado do "*chamador*", valores devolvidos pelo "*chamado*";

Ao nível das regras de "*sã convivência*":

- Que registos do CPU podem "*chamador*" e "*chamado*" usar, sem que isso represente um "conflito de interesses" (pelo facto de um alterar o conteúdo de um registo que está simultaneamente a ser usado pelo outro)
- Como partilhar a memória usada para armazenar dados, sem risco de sobreposição (e consequente perda de informação armazenada)

Convenções adoptadas quanto à passagem de parâmetros

Caso a subrotina defina parâmetros no seu interface, a passagem desses parâmetros entre "*chamador*" e "*chamado*" deve, no caso do MIPS, respeitar as seguintes regras:

- Os parâmetros que possam ser armazenados na dimensão de um registo (32 bits) devem ser passados à subrotina nos registos **$a0** a **$a3** (**$4** a **$7**) por esta ordem (o primeiro parâmetro sempre em **$a0**, o segundo em **$a1** e assim sucessivamente).
- *Caso o número de parâmetros a passar nos registos **a_i$** seja superior a quatro, os restantes (pela ordem em que são declarados) deverão ser passados na stack.*
- *No caso de um ou mais parâmetros serem do tipo float ou double, os registos utilizados para os passar serão os registos **$f12** e **$f14** do coprocessador de vírgula flutuante*

Convenções adoptadas quanto à devolução de valores

Caso a subrotina pretenda devolver um ou mais valores ao programa "*chamador*", essa devolução deve, no caso do MIPS, respeitar as seguintes regras:

- A subrotina pode devolver um ou dois valores desde que estes possam ser armazenados na dimensão de um registo (32 bits). Nesse caso esse ou esses valores devem ser devolvidos através dos registos **$v0** e **$v1** (**$2** e **$3**), por esta ordem.
- *No caso de o valor a devolver ser do tipo float ou double, o registo a utilizar será o registo **$f0** do coprocessador de vírgula flutuante.*

Exemplo:

```
int average_int (int a, int b) ;

void main(void)
{
    static int av;
    av = average_int (19, 35);
}
```

Em *Assembly*:

```
        .data
av:     .space  4
        .text
main:   li      $a0, 19
        li      $a1, 35
        jal     average_int
        la      $t0, av
        sw      $v0, 0($t0)
        ...
```

parâmetros

evocação da subrotina

valor devolvido

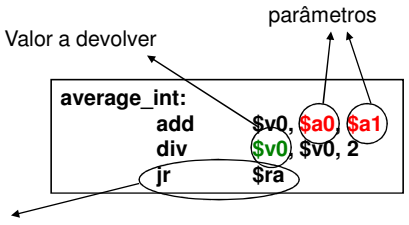
Note-se que , para escrever o programa "chamador", não é necessário conhecer os detalhes de implementação do "chamado"

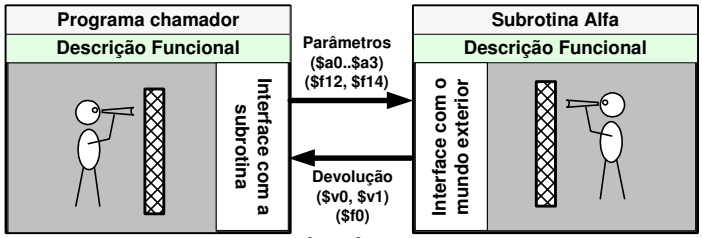
Exemplo (continuação):

```
int average_int (int a, int b)
{
    return (a+b) / 2;
}
```

Note-se que , para escrever o programa "chamado", não é necessário conhecer os detalhes de implementação do "chamador"

Em *Assembly*:





Interface:

```
int average_int (int a, int b);
```

```
        .data
av:     .space  4
        .text
main:   li      $a0, 19
        li      $a1, 35
        jal     average_int
        la      $t0, av
        sw      $v0, 0($t0)
        ...
```

The Big Picture

```
average_int:
        add     $v0, $a0, $a1
        div     $v0, $v0, 2
        jr      $ra
```

Convenções adoptadas quanto à salvaguarda de registos

Que registos pode usar uma subrotina, sem que se corra o risco de que os mesmos registos estejam a ser usados pelo programa "*chamador*", potenciando assim a destruição de informação vital para a execução do programa como um todo?

- Uma hipótese seria dividir de forma estática os registos existentes entre "*chamador*" e "*chamado*"? Mas o que fazer então se o "*chamado*" fosse simultaneamente "*chamador*" (subrotina que chama outra subrotina)?
- Outra hipótese, mais praticável, consiste em atribuir a um dos "parceiros" a responsabilidade de copiar previamente para a memória externa o conteúdo de qualquer registo que pretenda utilizar (**salvaguardar o registo**) e posteriormente, repor o valor original lá armazenado.

Convenções adoptadas quanto à salvaguarda de registos

Há duas estratégias que são geralmente adoptadas (em alternativa) pela maior parte das arquitecturas:

- Deixa-se ao cuidado do programa "*chamador*" a responsabilidade de salvaguardar o conteúdo da totalidade dos registos antes de evocar a subrotina, cabendo-lhe também a tarefa de repor posteriormente o seu valor (embora, no limite, seja admissível que o primeiro salvaguarde apenas o conteúdo dos registos de que efectivamente venha a precisar mais tarde). Estamos nesse caso perante uma estratégia "***caller-saved***".
- Entrega-se à subrotina a responsabilidade pela prévia salvaguarda dos registos de que possa necessitar, assegurando a mesma subrotina a tarefa de repor o seu valor imediatamente antes de regressar ao programa "*chamador*". Estamos nesse caso perante uma estratégia "***callee-saved***".

Convenções adoptadas quanto à salvaguarda de registos

No caso do MIPS, a estratégia adoptada é uma versão mista das anteriores, e baseia-se nas duas regras seguintes:

- **Os registos $t0..$t9, $v0..$v1 e $a0..$a3 podem ser livremente utilizados e alterados pelas subrotinas**
- **Os valores dos registos $s0..$s7 não podem ser alterados pelas subrotinas**

Arquitectura de Computadores I 2007/08

Convenções adoptadas quanto à salvaguarda de registos

- **Os registos $t0..$t9, $v0..$v1 e $a0..$a3 podem ser livremente utilizados e alterados pelas subrotinas**

Esta regra implica, na prática, que um programa "*chamador*" que esteja a usar um ou mais destes registos, deverá salvaguardar o seu conteúdo antes de evocar uma subrotina, sob pena de que esta os venha a alterar.

- **Os valores dos registos $s0..$s7 não podem ser alterados pelas subrotinas**

A segunda regra implica que se uma dada subrotina precisar de usar um registo do tipo **$s**, compete a essa subrotina copiar **previamente** o seu conteúdo para um lugar seguro (memória externa), repondo-o imediatamente antes de terminar. Dessa forma, do ponto de vista do programa "*chamador*" (que não "vê" o código da subrotina) é como se esse registo não tivesse sido usado ou alterado.

Considerações práticas sobre a utilização da convenção

- Subrotinas terminais (que não chamam qualquer subrotina)
  - Só devem utilizar registos que não necessitam de ser salvaguardados (**$t0..$t9**, **$v0..$v1** **e** **$a0..$a3**)
- Subrotinas que chamam outras subrotinas
  - Devem utilizar os registos **$s0..$s7** para o armazenamento de valores que se pertenda preservar. A utilização destes registos implica a sua prévia salvaguarda na memória externa logo no início da subrotina e a respectiva reposição no final
  - Devem utilizar os registos **$t0..$t9**, **$v0..$v1** **e** **$a0..$a3** para os restantes valores