AC1 2013-2014

universidade de aveiro theoria poiesis praxis
Departamento de Electrónica, Telecomunicações e Informática

Arquitectura de Computadores I

Pedro Miguel Cabral

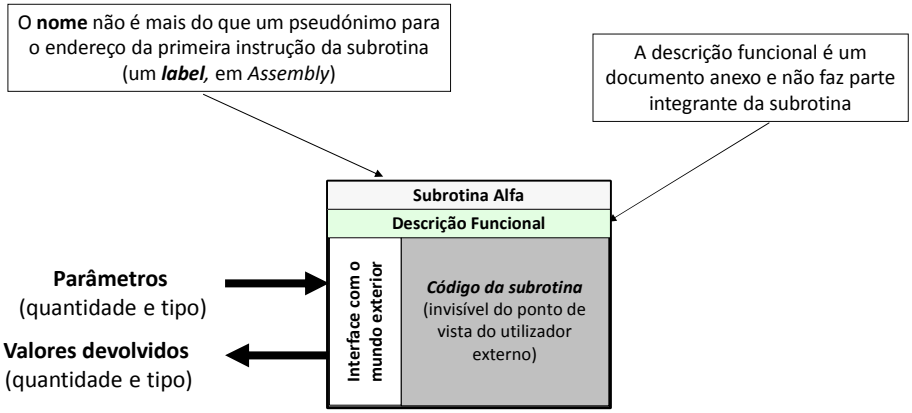
Aula 07

Subrotinas

1

AC1 2013-2014

Do ponto de vista do utilizador externo uma subrotina é uma caixa preta que encapsula uma determinada funcionalidade.



2

AC1 2013-2014

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** por esta ordem (o 1º parâmetro sempre em **$a0**, o segundo em **$a1** e assim sucessivamente).
- Caso o nº de parâmetros a passar nos registos **a_i$** seja superior a 4, os restantes (pela ordem em que são declarados) deverão ser passados na stack.
- No caso de um ou mais parâmetros serem do tipo float ou double, os registos utilizados para os passar serão os registos **$f12** e **$f14** do coprocessador de vírgula flutuante.

3

AC1 2013-2014

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**, 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.

4

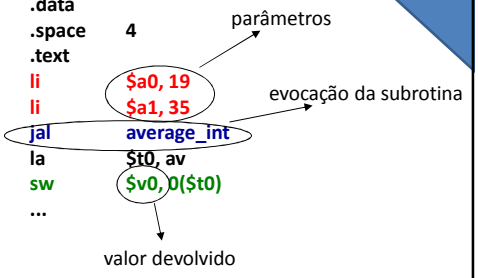
Exemplo:

AC1 2013-2014

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

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

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




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

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

5

AC1 2013-2014

Convenções adoptadas quanto à salvaguarda de registos

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

• O programa "*chamador*" tem a responsabilidade:

Salvaguardar o conteúdo da totalidade dos registos antes de evocar a subrotina

Posteriormente repor o seu valor

(admissível que salvaguarde apenas o conteúdo dos registos de que efectivamente venha a precisar mais tarde).

Estamos nesse caso perante uma estratégia "***caller-saved***".

6

Convenções adoptadas quanto à salvaguarda de registos

AC1 2013-2014

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

- A subrotina tem a responsabilidade de:

  Salvaguardar os registos de que possa necessitar,

  Repor o seu valor imediatamente antes de regressar ao programa "*chamador*".

Estamos nesse caso perante uma estratégia "***callee-saved***".

7

Convenções adoptadas quanto à salvaguarda de registos

AC1 2013-2014

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**

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**

Se uma dada subrotina precisar de usar um registo do tipo **$s**, compete a essa subrotina salvaguardar **previamente** o seu conteúdo para 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.

8

Considerações práticas sobre a utilização da convenção

AC1 2013-2014

- 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

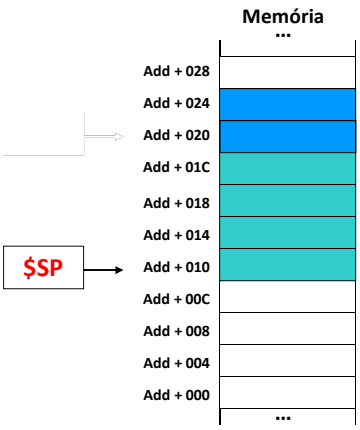
9

Regras de utilização da *stack* nas arquitecturas MIPS

AC1 2013-2014

1. O registo **$sp** (*stack pointer*) contém o endereço da última posição ocupada da *stack*

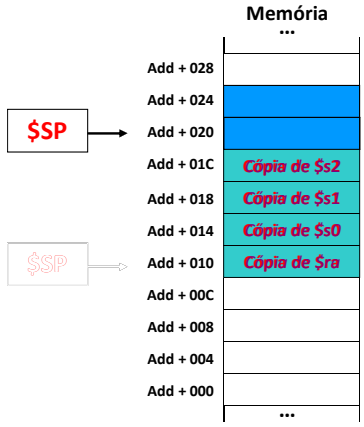
2. A *stack* cresce por ordem decrescente dos endereços da memória



AC1 2013-2014

Regras de utilização da *stack* nas arquitecturas MIPS

```
lab: subu $sp, $sp, 16 # Reserva espaço na stack
     sw   $ra, 0($sp)  # Copia registos
     sw   $s0, 4($sp)  # $ra, $s0 a $s2
     sw   $s1, 8($sp)  # para a
     sw   $s2, 12($sp) # stack
          (...)        # Código intermédio
     lw   $ra, 0($sp)  # Repõe o valor
     lw   $s0, 4($sp)  # dos
     lw   $s1, 8($sp)  # registos
     lw   $s2, 12($sp) # $ra, $s0 a $s2
     addu $sp, $sp, 16 # Liberta espaço na
                       # stack
```



AC1 2013-2014

Exemplo:

A codificação da rotina *main* está sujeita às mesmas regras que se aplicam às restantes rotinas. Porquê?

Text Segment

Bkpt	Address	Code	Basic		Source
	0x00400000	0x8fa40000	lw $4,0x0000($29)	171: lw $a0 0($sp)	# argc
	0x00400004	0x27a50004	addiu $5,$29,0x0004	172: addiu $a1 $sp 4	# argv
	0x00400008	0x24a60004	addiu $6,$5,0x0004	173: addiu $a2 $a1 4	# envp
	0x0040000c	0x00041080	sll $2,$4,0x0002	174: sll $v0 $a0 2	
	0x00400010	0x00c23021	addu $6,$6,$2	175: addu $a2 $a2 $v0	
	0x00400014	0x0c100009	jal 0x00400024	176: jal main	
	0x00400018	0x00000000	nop	177: nop	
	0x0040001c	0x2402000a	addiu $2,$0,0x000a	179: li $v0 10	
	0x00400020	0x0000000c	syscall	180: syscall	# syscall 10 (exit)
	0x00400024	0x24020004	addiu $2,$0,0x0004	10: li $v0, 4	
	0x00400028	0x3c011001	lui $1,0x1001	11: la $a0, str1	
	0x0040002c	0x34240000	ori $4,$1,0x0000		
	0x00400030	0x0000000c	syscall	12: syscall	
	0x00400034	0x24020005	addiu $2,$0,0x0005	14: li $v0, 5	
	0x00400038	0x0000000c	syscall	15: syscall	

Porque é que até agora ainda não houve a necessidade de salvaguardar pelo menos o **`$ra`**?

Até este momento a nossa rotina ***main*** tem sido sempre terminal!!

Exemplo:

```
        .data
str1:   .asciiz "Insira um numero: "
str2:   .asciiz "O quadrado de "
str3:   .asciiz " e´ "
        .text
        .globl main

main:
        li      $v0, 4
        la      $a0, str1
        syscall

        li      $v0, 5
        syscall
        move    $t0, $v0

        move    $a0, $t0
        jal     square
        move    $t1, $v0

        li      $v0, 4
        la      $a0, str2
        syscall

        li      $v0, 1
        move    $a0, $t0
        syscall

        li      $v0, 4
        la      $a0, str3
        syscall

        li      $v0, 1
        move    $a0, $t1
        syscall

        jr      $ra
                .
                .
                .
```

AC1 2013-2014

Quais os problemas com este código???