“Arquitetura e Organização de Computadores I – Aula_03 – Linguagem de Máquina - Continuação”

Prof. Dr. Emerson Carlos Pedrino
DC/UFSCar
São Carlos

Procedimentos Aninhados

- Procedimento “Folha” -> não chama outro procedimento.
- Exemplo de procedimento não folha (cálculo de fatorial):

```
int fact (int n)
{
    if (n < 1) return (1);
        else return (n * fact(n-1));
}
```

Exemplo de estrutura de pilha para o exemplo anterior

$ra: endereço de retorno.

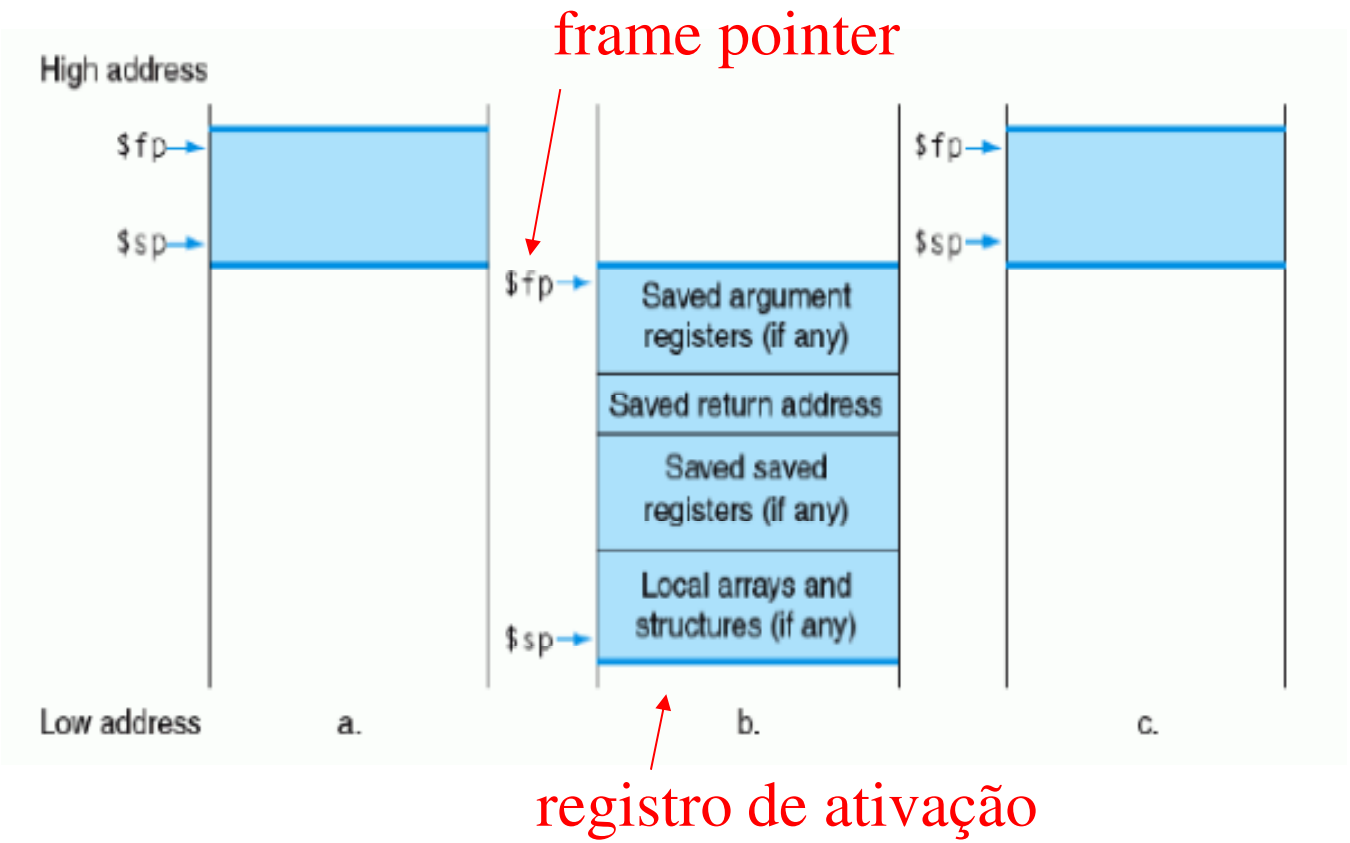
$a0: registrador de argumento.

O que é e o que não é preservado através da chamada de um procedimento?

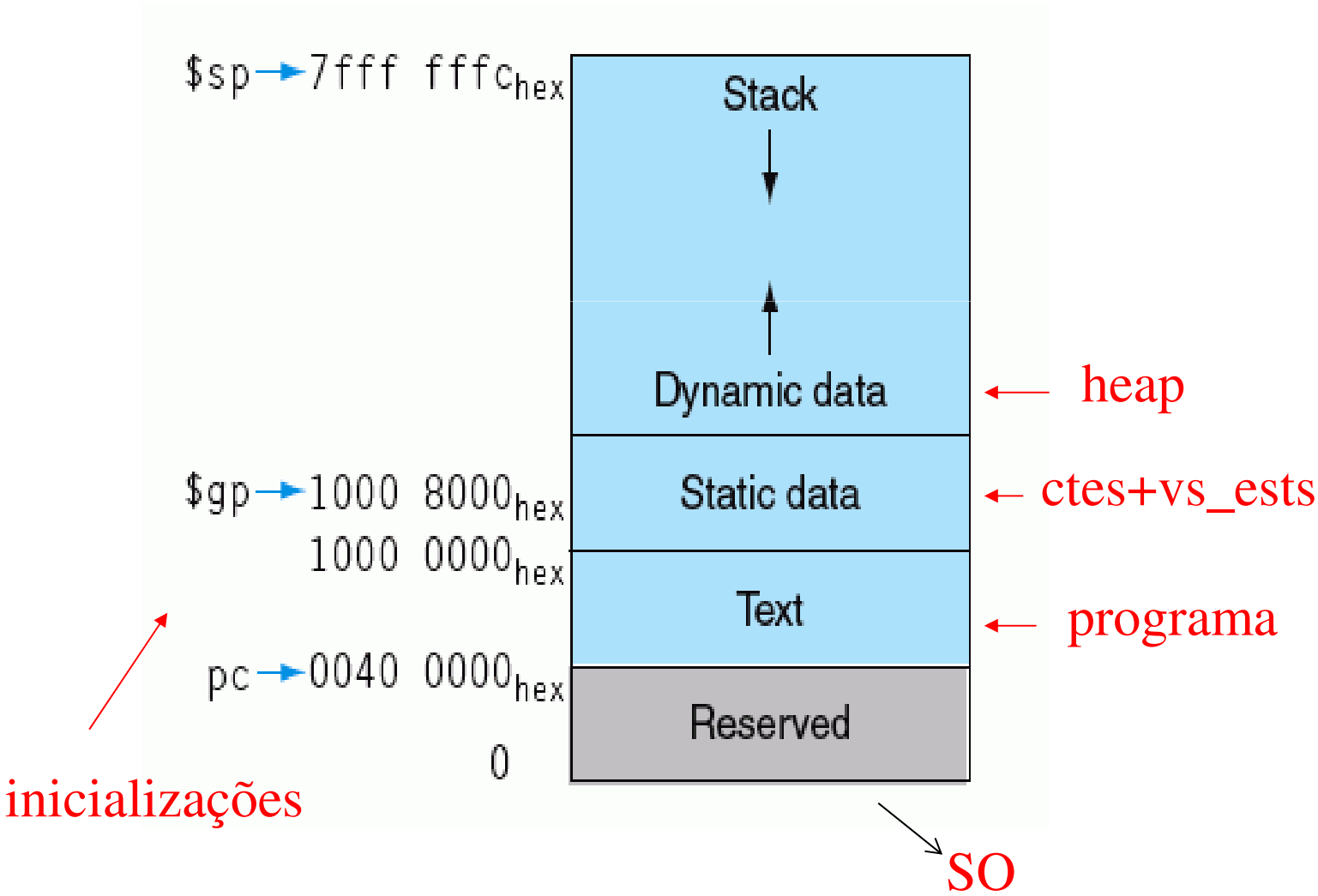
- A pilha acima de $sp é preservada (se o procedimento chamado não escrever acima de $sp).
- $sp é preservado pelo procedimento chamado somando-se o mesmo valor que foi subtraído dele.
- Registradores salvos na pilha e restaurados de lá.

Alocando espaço para novos dados na pilha

- Exemplo de alocação da pilha antes, durante e depois da chamada de um procedimento.



Alocação de memória para programa e dados no MIPS (Convenção de *Software*)



Resumo

MIPS operands

Name	Example	Comments
32 registers	$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. $gp (28) is the global pointer, $sp (29) is the stack pointer, $fp (30) is the frame pointer, and $ra (31) is the return address.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	three register operands
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Data from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Data from register to memory
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $$s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,L	if ($s1 == $s2) go to L	Equal test and branch
	branch on not equal	bne $s1,$s2,L	if ($s1 != $s2) go to L	Not equal test and branch
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; used with beq, bne
	set on less than immediate	slt $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than immediate; used with beq, bne
Unconditional jump	jump	j L	go to L	Jump to target address
	jump register	jr $ra	go to $ra	For procedure return
	jump and link	jal L	$ra = PC + 4; go to L	For procedure call

Resumo

MIPS machine language

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
and	R	0	18	19	17	0	36	and $s1,$s2,$s3
or	R	0	18	19	17	0	37	or $s1,$s2,$s3
nor	R	0	18	19	17	0	39	nor $s1,$s2,$s3
andi	I	12	18	17	100			andi $s1,$s2,100
ori	I	13	18	17	100			ori $s1,$s2,100
sll	R	0	0	18	17	10	0	sll $s1,$s2,10
srl	R	0	0	18	17	10	2	srl $s1,$s2,10
beq	I	4	17	18	25			beq $s1,$s2,100
bne	I	5	17	18	25			bne $s1,$s2,100
slt	R	0	18	19	17	0	42	slt $s1,$s2,$s3
j	J	2	2500					j 10000 (see Section 2.9)
jr	R	0	31	0	0	0	8	jr $ra
jal	J	3	2500					jal 10000 (see Section 2.9)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer, branch format

Instruções para mover *bytes*

- lb (*load byte*) e sb (*store byte*).
- Exemplo: (modo *big endian*)

```
lb $t0,0($sp)          # Read byte from source
sb $t0,0($gp)          # Write byte to destination
```

Tabela ASCII

ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	DEL

Representação de *strings*

- 0 – *null* em ASCII. Usado em *C:* final de *string*.
- *Exercício: Qual é a representação da *string* “Cal” em *C*?
  - Sol.: 67, 97, 108, 0.

Exemplo

```
void strcpy (char x[], char y[])
{
     int i;

     i = 0;
     while ((x[i] = y[i]) != '\0') /* copy & test byte */
      i += 1;
}
```

Copia *string* y na *string* x

Exemplo: código em *assembly*

- Endereços:
  - x -> $a0 e y-> $a1; x e y: *arrays;*
  - i -> $s0;

```
strcpy:
    addi   $sp,$sp,-4   # adjust stack for 1 more item
    sw     $s0, 0($sp)  # save $s0
```

Exemplo: código em *assembly*

```
add    $s0,$zero,$zero  # i = 0 + 0

L1:  add    $t1,$s0,$a1  # address of y[i] in $t1

lb     $t2, 0($t1)  # $t2 = y[i]

add    $t3,$s0,$a0  # address of x[i] in $t3
sb     $t2, 0($t3)  # x[i] = y[i]
```

Exemplo: código em *assembly*

```
beq   $t2,$zero,L2 # if y[i] == 0, go to L2
```

```
addi  $s0, $s0,1   # i = i + 1
j     L1           # go to L1
```

```
L2: lw    $s0, 0($sp)  # y[i] == 0: end of string;
                       # restore old $s0
    addi  $sp,$sp,4    # pop 1 word off stack
    jr    $ra          # return
```

Endereçamento MIPS (*lui: load upper immediate*)

- Operandos imediatos de 32 bits.
- Registrador $at: temporário e disponível para o montador lidar com constantes longas.

The machine language version of `lui $t0, 255` `# $t0 is register 8:`

001111	00000	01000	0000 0000 1111 1111

Contents of register `$t0` after executing `lui $t0, 255`:

0000 0000 1111 1111	0000 0000 0000 0000

Exercício*

- Qual é o código *assembly* do MIPS para carregar a constante de 32 bits dada a seguir no registrador $s0?
- Cte -> 0000 0000 0011 1101 0000 1001 0000 0000
- Sol.:
  - lui $s0,61
  - ori $s0,$s0,2304
  - *Obs.: Mostrar o que aconteceria se fosse utilizada a instrução *addi* em vez de *ori*?

Endereçamento em desvios condicionais e *jumps*

- Formato do tipo J.
- Ex: j 10000 # vai para a posição 10000 (x 4).
- Ex: instrução de *jump.*

2	10000
6 bits	26 bits

Relativo à WORD. Dois Bits implícitos (28 bits).

Instrução de desvio condicional

- Ex: bne $s0,$s1,Exit # vai para Exit se $s0 <> $s1.
- End = 16 bits -> 2^{16}.
- Pode-se desviar dentro de $\pm 2^{15}$ em relação a (PC+4).

5	16	17	Exit
6 bits	5 bits	5 bits	16 bits

Exemplo

```
Loop:sll    $t1,$s3,2   # Temp reg $t1 = 4 * i
     add $t1,$t1,$s6    # $t1 = address of save[i]
     lw  $t0,0($t1)     # Temp reg $t0 = save[i]
     bne $t0,$s5, Exit  # go to Exit if save[i] ≠ k
     addi $s3,$s3,1     # i = i + 1
     j   Loop           # go to Loop
Exit:
```

80000	0	0	19	9	4	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024	...					

2 words + 80016

20000x4 Loop

Exit

Desviando para um lugar mais distante

- Exercício*
  - Dado um desvio onde o registrador $s0 é igual ao registrador $s1, beq $s0,$s1,L1, substitua-o por um par de instruções que ofereça uma distância de desvio muito maior.
  - Sol.:
  - bne $s0,$s1,L2
  - j L1
  - L2:

Resumo dos modos de endereçamento no MIPS

- Endereçamento imediato: addi $s1, $s2, 100
- Endereçamento de registrador: add $s1, $s2, $s3
- Endereçamento de base: lw $s1, 100($s2)
- Endereçamento relativo ao PC: beq $s1, $s2, DESTINO
- Endereçamento absoluto: j DESTINO

Formato de instruções do MIPS

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

Resumo até aqui

MIPS operands

Name	Example	Comments
32 registers	$s0–$s7, $t0–$t9, $zero, $a0–$a3, $v0–$v1, $gp, $fp, $sp, $ra, $at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. Register $at is reserved for the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory($s2 + 100)	Word from memory to register
	store word	sw $s1,100($s2)	Memory($s2 + 100) = $s1	Word from register to memory
	load half	lh $s1,100($s2)	$s1 = Memory($s2 + 100)	Halfword memory to register
	store half	sh $s1,100($s2)	Memory($s2 + 100) = $s1	Halfword register to memory
	load byte	lb $s1,100($s2)	$s1 = Memory($s2 + 100)	Byte from memory to register
	store byte	sb $s1,100($s2)	Memory($s2 + 100) = $s1	Byte from register to memory
	load upper immed.	lui $s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than constant
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call