Universidade Federal de Pelotas
Centro de Desenvolvimento Tecnológico
Bacharelado em Ciência da Computação
Engenharia de Computação

Arquitetura e Organização de Computadores I

Prática

Aula 10

Revisão

Prof. Guilherme Corrêa
gcorrea@inf.ufpel.edu.br

Prof. Bruno Zatt

Assembly Estendido

Pseudo-instrução `move`

- Atribui o valor de um registrador a outro

  Exemplo: `move $t0, $s0`

  Atribui (copia) o valor de **$s0** para **$t0**.

Pseudo-instrução `li`

- Carrega no registrador destino o valor inteiro positivo ou negativo de até 32 bits

  Exemplo: `li $t0, 125`

  Atribui o valor **125** para **$t0**.

Assembly Estendido

Pseudo-instruções para acesso à memória

`lw rd, endereco1`	`# carrega rd com um valor` `# armazenado em endereco1`
`la rd, endereco2`	`# carrega rd com o endereço` `# de memoria de endereco2`
`sw rd, endereco3`	`# escreve conteúdo de rd no` `# endereço simbolico endereco3`

Exemplo:

```
        .text
        lw $t0, endereco1
        la $t1, endereco2
        sw $t2, endereco3

        .data
        endereco1: .word 5
        endereco2: .word 10
        endereco3: .word
```

Strings

- As **strings** são escritas em memória, em **bytes contíguos**;
- Cada caractere é representado em um byte;
- Endereços são **alinhados por byte**;
- Codificação **ASCII**;
- Definimos as strings no código assembly **entre aspas** após **.asciiz** na seção de dados da memória.

Exemplo:

```
.data
texto: .asciiz "Harry Potter"
```

Tabela ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Vetores

- O tamanho do vetor é definido por um inteiro (*word*);
- O vetor é inicializado com valores separados por vírgulas;
- Os endereços são alinhados por *words*.

Exemplo:

```
.data
  tamanho: .word  12
  vetor:   .word  4,-2,33,52,1,17,11,7,90,-7,8,-13
```

Tratamento de Exceções

Instrução `syscall`

- Serviço especificado por um **código em $v0**;
- Serviços diferentes esperam **parâmetros** em registradores diferentes e nem todos os serviços retornam valores.

```
li   $v0, codigo        # Carrega em $v0 o código de
                        # um serviço do SO.

.......                 # Coloca parâmetros para o serviço nos
.......                 # registradores $a0, $a1
                        # ou registradores $f (float)

syscall                 # Invoca o sistema operacional.
                        # Valor de retorno (se existir)
                        # em $v0 ou $f0
```

Tratamento de Exceções

Registradores para uso em `syscall`

Serviço	Código ($v0)	Argumentos	Retorno
imprime inteiro	1	$a0 (inteiro)	-
imprime float	2	$f12 (float)	-
imprime double	3	$f12, $f13 (double)	-
imprime string	4	$a0 (endereço)	-
lê inteiro	5	-	$v0 (inteiro)
lê float	6	-	$f0 (float)
lê double	7	-	$f0, $f1 (double)
lê string	8	$a0 (endereço) $a1 (tamanho)	-
aloca memória	9	$a0 (n. de bytes)	$v0 (endereço)
sai	10	-	-

Subrotinas

Instrução `jal`

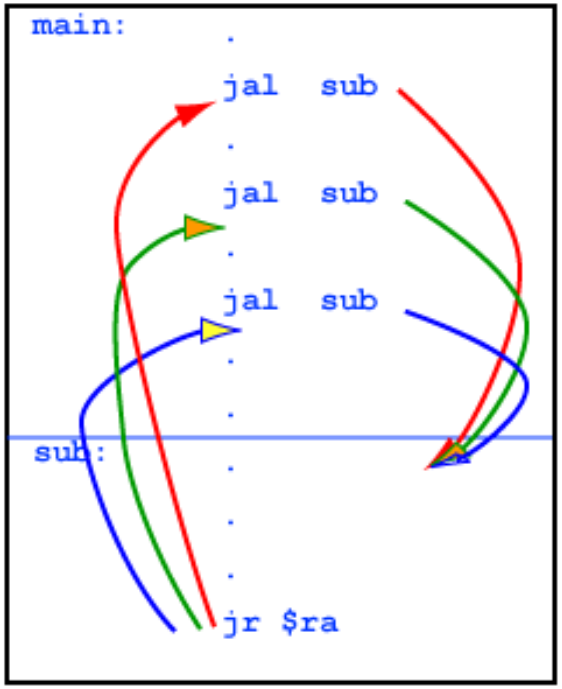
- As **chamadas** para subrotinas (funções) no MIPS são feitas através da instrução **`jal`** (*jump and link*);
- O registrador especial **`$ra`** (**`$31`**) recebe o endereço de **retorno** da subrotina;
- O endereço de retorno equivale à instrução **após o *delay slot***.

```
jal sub         # $ra ($31) <- PC+8
                # (endereço a 8 bytes da instrução jal)
                # PC  <- sub   PC recebe o endereço de
                # entrada da subrotina
                # a instrução precisa de um delay slot
```

Subrotinas

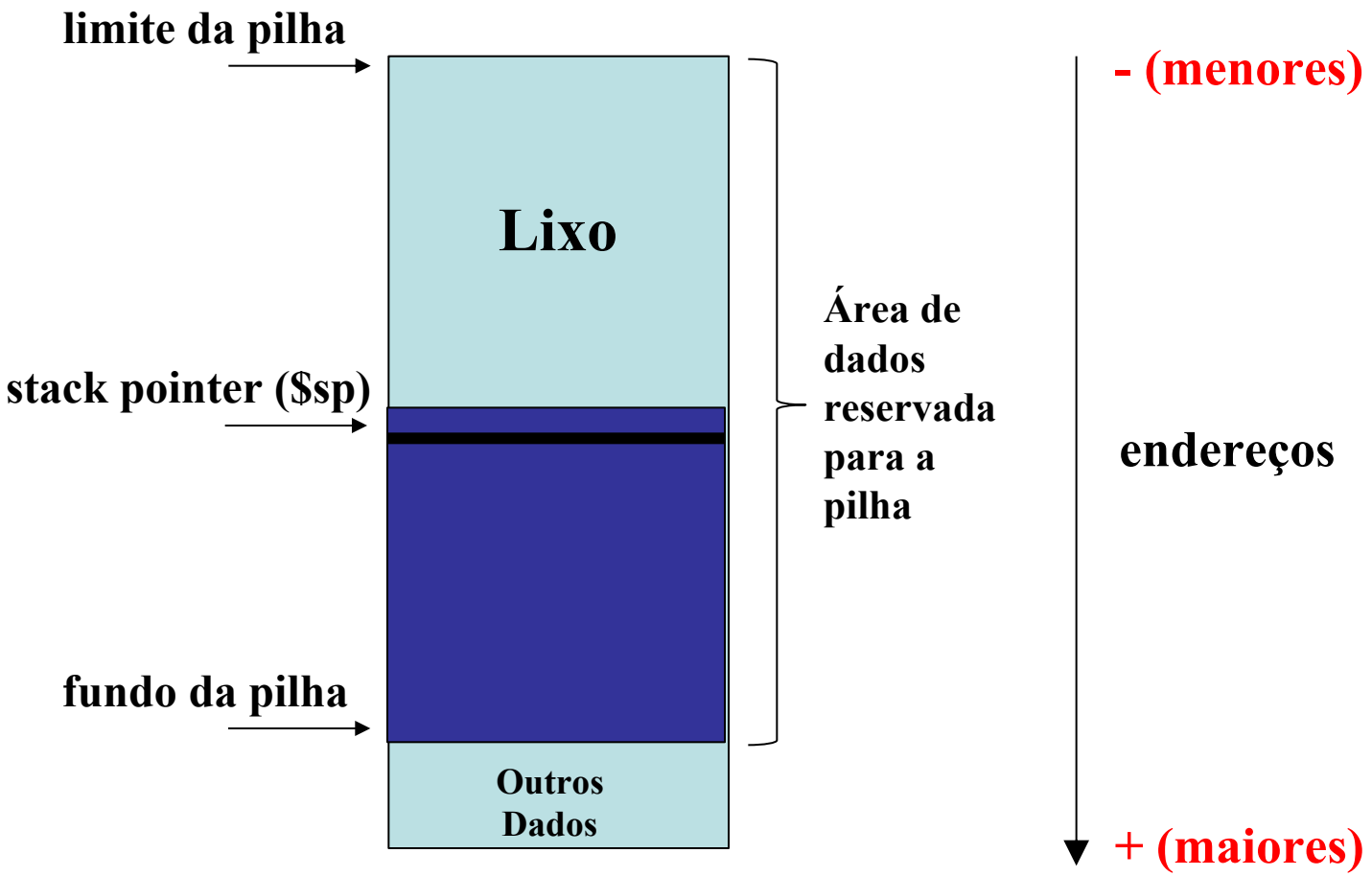
Instrução `jal`

- A cada chamada a subrotina (`jal`), `$ra` recebe o endereço de retorno apropriado;
- `jr $ra` (retorno da subrotina) salta para endereço após o ***delay slot*** do `jal` correspondente.



Correct Subroutine Linkage

Pilhas

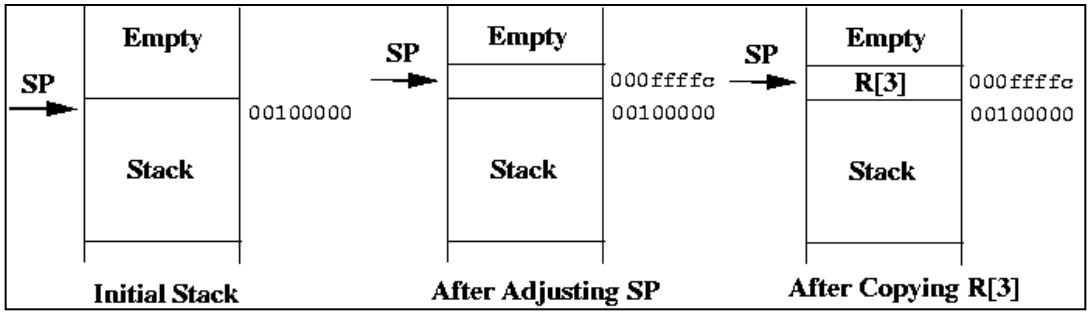

limite da pilha
Lixo
stack pointer ($sp)
fundo da pilha
Outros
Dados
Área de
dados
reservada
para a
pilha
- (menores)
endereços
+ (maiores)

Pilhas

Operação push

- **Armazena** uma palavra no **topo** da pilha;
- Atualiza $sp:

  $sp ← $sp - 4
- Escreve a palavra a ser armazenada no endereço indicado por $sp



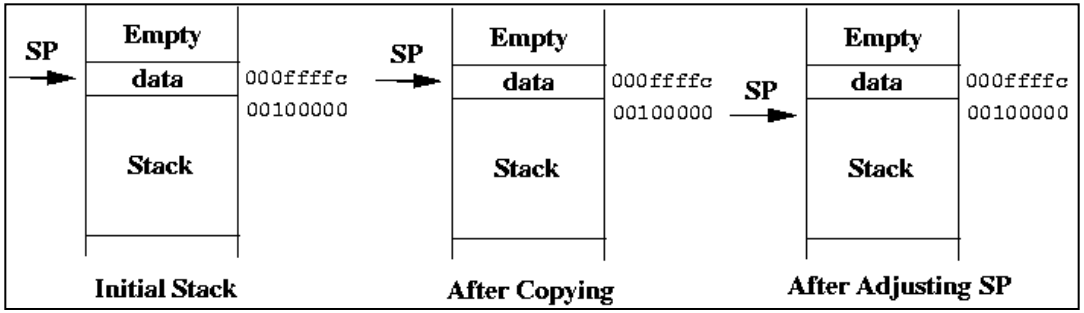

```
addi $sp,$sp,-4      # aponta para o endereço do novo item,
sw   $t0,($sp)       # armazena o conteúdo de $t0 no novo topo
```

Pilhas

Operação pop

- **Retira** uma palavra do **topo** da pilha;
- Lê a palavra do endereço indicado por $sp e salva em um registrador;
- Atualiza $sp:

  $sp ← $sp + 4




```
lw   $t0,($sp)        #   Copia o item para $t0.
addi $sp,$sp,4        #   Atualiza topo.
```

Pilha de Execução

Guia para utilização da pilha de execução

- Chamada da subrotina (executado pelo *chamador*):
  1. Empilhar registradores **$t0-$t9** que precisem ser salvos, se existirem. A subrotina pode alterá-los.
  2. Colocar argumentos em **$a0-$a3**.
  3. Chamar a subrotina usando **jal**.

- Prólogo da subrotina (executado no início da subrotina):
  4. Se a subrotina chama outras subrotinas, salvar **$ra** na pilha.
  5. Salvar na pilha os registradores **$s0-$s7** que a subrotina venha a alterar.

Pilha de Execução

Guia para utilização da pilha de execução

- Recuperando o controle (executado pelo chamador após a subrotina):

12. Desempilhar (ordem inversa) quaisquer registradores **$t0-$t9** que tenham sido salvos no passo 1.

Observe que este modelo suporta recursividade!