Arquitetura de Computadores I

Cap. 02 – Instruções Básicas: A Linguagem da Máquina

(Aula 05)

Plano de aula

- Procedimentos aninhados
- Registrador $fp

Procedimentos Aninhados

- Procedimentos que não chamam outros procedimentos são chamados de procedimentos-folha.

- É necessário redobrar o cuidado com o uso de registradores em procedimentos não-folha.

- Exemplo de uma situação problemática
  - Programa principal chama procedimento A, com um argumento 3 colocado no registrador $a0 e executando `jal A`
  - Procedimento A chama um procedimento B executando a instrução `jal B`, com um argumento 7 também colocado em `$a0`.

Procedimentos Aninhados

- Solução da situação problemática
  - Armazenar na pilha todos os registradores que precisem ser preservados
  - O procedimento chamador coloca na pilha todos os registradores de argumento (`$a0-$a3`) ou registradores temporários (`$t0-$t7`) que sejam necessários após a chamada.
  - O procedimento chamado coloca na pilha o endereço de retorno, armazenado em `$ra`, e todos os registradores de salvamento utilizados por ele (`$s0-$s7`)
  - O stack pointer `$sp` é ajustado para acomodar a quantidade de registradores colocados na pilha.
  - Quando do retorno, os valores dos registradores são restaurados a partir da pilha, e o stack pointer também volta ao seu valor inicial.

Procedimentos Aninhados

- Exemplo

```
int func2 (int z) {
    if (z > 0) {
        return (1);
         else
        return (z);
}

int func1 (int x, int y)
{
    temp =  x – func2(y)
    Return (temp);
}

int main (void) {
    a = func1 (b, c);
}
```

```
Func2: slt $t0, $zero, $a0
       beq $t0, $zero, ELSE
       addi $v0, $zero, 1
       j EXIT
 ELSE: add $v0, $zero, $a0
 EXIT: jr $ra

Func1: addi, $sp, $sp, -8
       sw $a0, 4($sp)
       add $a0, $zero, $a1
       sw $ra, 0($sp)
       jal Func2
       lw $ra, 0($sp)
       lw $a0, 4($sp)
       addi $sp, $sp, 8
       sub $v0, $a0, $v0
       jr $ra

Main:  addi $a0, $zero, $s1
       addi $a1, $zero, $s2
       jal func1
       add $s0, $zero, $v0
```

Procedimentos Aninhados

- Ex.

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

```
Fact:
    addi $sp, $sp, -8     #Ajusta a pilha para receber 2 itens
    sw $ra, 4($sp)    #Salva o endereço de retorno
    sw $a0, 0($sp)    #Salva o argumento n
    addi $t1, $zero, 1    # $t1 recebe 1
    slt $t0, $a0, $t1         #Testa se n < 1
    beq $t0, $zero, L1    #Se n >= 1, desvia para L1
    addi $v0, $zero, 1    #Retorna o valor 1
    addi $sp, $sp, 8      #Elimina 2 itens da pilha
    jr   $ra          #Retorna para depois da instrução jal
L1: addi $a0, $a0, -1         # n >= 1: argumento recebe (n-1)
    jal fact          # chama fact com argumento (n-1)
    lw $a0, 0($sp)    # retorna de jal: restaura argumento n
    lw $ra, 4($sp)    # restaura o endereço de retorno
    addi $sp, $sp, 8 # ajusta o stack pointer para eliminar 2 itens
    mult $v0, $a0, $v0    #retorna n*fact(n-1)
    jr $ra   #retorna para o chamador
```

Procedimentos Aninhados

Preservados	Não-Preservados
Registradores de salvamento: $s0–$s7	Registradores temporários: $t0–$t9
Registrador stack pointer: $sp	Registradores de argumento: $a0–$a3
Registrador de endereço de retorno: $ra	Registradores de retorno de valores: $v0–$v1
Pilha acima do stack pointer	Pilha abaixo do stack pointer

Variáveis locais e parâmetros

- As variáveis locais de um procedimento quando, numerosas podem ser armazenadas na pilha.

- Quando é necessário passar mais de 4 parâmetros para um procedimento, pode-se utilizar também a pilha para passagem dos parâmetros excedentes.

- A linguagem C possui duas classes de variáveis:
  - Variáveis Locais (ou variáveis automáticas)
    - São locais a um determinado procedimento
    - São descartadas quando o procedimento termina
  - Variáveis Globais (ou variáveis estáticas)
    - Sobrevivem aos procedimentos
    - São declaradas fora de qualquer procedimento

- Para simplificar o acesso aos dados estáticos, existe o registrador `$gp` (*global pointer*)

Alocação de Espaço para Novos Dados

- A pilha também é usada para guardar variáveis que são locais ao procedimento e que, por serem numerosas, não podem ser armazenadas nos registradores.

- O seguimento da pilha que contém os registradores de salvamento do procedimento e suas variáveis locais é chamado de *quadro do procedimento*

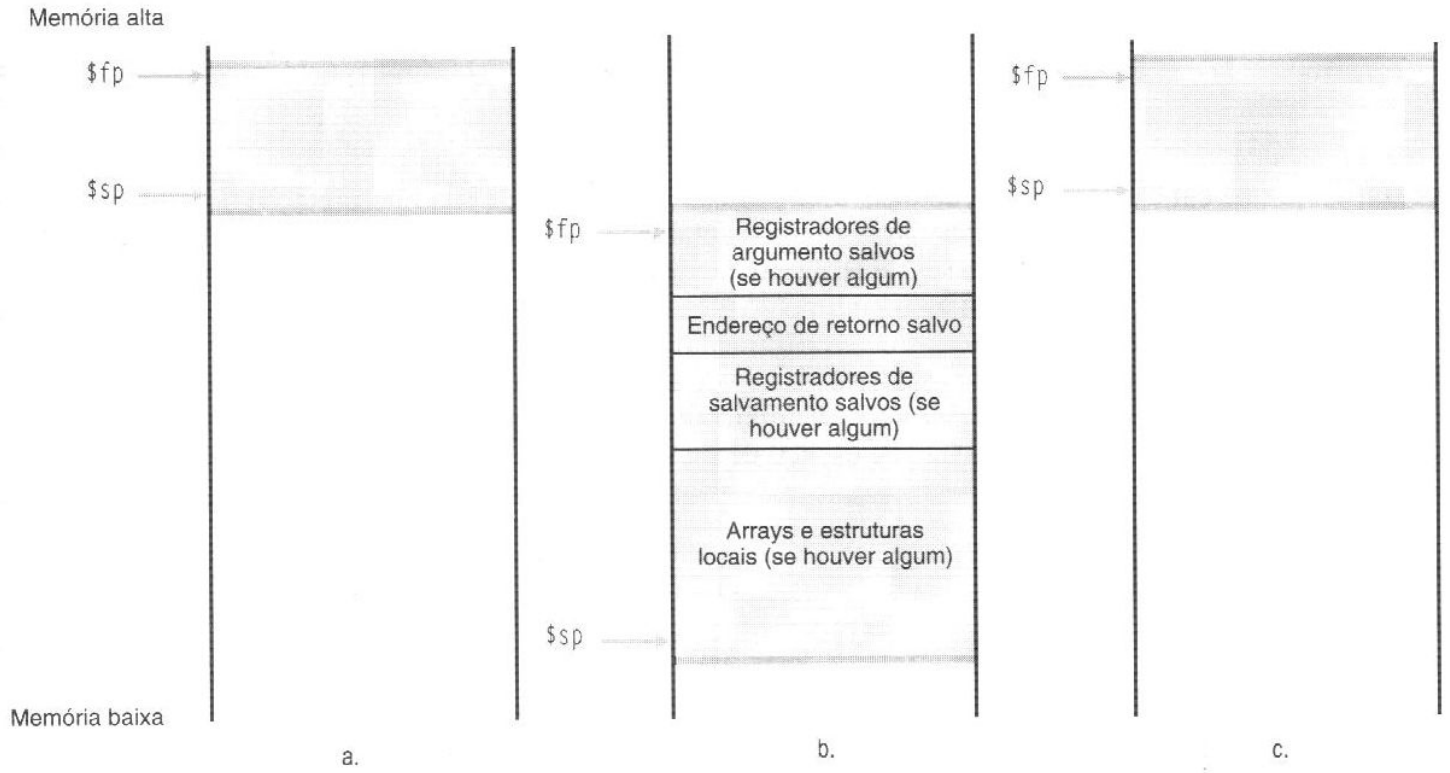
- O registrador `frame pointer ($fp)` aponta para o quadro de um procedimento.

- O conteúdo do registrador $sp pode mudar durante a execução de um procedimento, e, portanto as referências a uma variável local na memória podem ter diferentes deslocamentos.

- O frame pointer oferece um registrador-base estável para as variáveis locais a um procedimento referenciarem a memória.

Alocação de Espaço para Novos Dados

- Temos evitado o uso do registrador $fp, não modificando o $sp dentro de um procedimento.
  - Nos exemplos a pilha só é ajustada no início e no fim de um procedimento



Convenções empregadas

- Objetivo: Utilizar os registradores de maneira racional pela linguagem de montagem do MPIS.

Nome	Número do registrador	Utilização	Preservado na chamada?
$zero	0	valor da constante 0	n.-a.
$v0–$v1	2-3	valores para guardar resultados e para avaliar expressões	não
$a0–$a3	4-7	argumentos	sim
$t0–$t7	8-15	temporários	não
$s0–$s7	16-23	salvos	sim
$t8–$t9	24-25	mais temporários	não
$gp	28	global pointer	sim
$sp	29	stack pointer	sim
$fp	30	frame pointer	sim
$ra	31	endereço de retorno a procedimento	sim

Figura 3.13 Convenção do MIPS para utilização de registradores. O registrador 1, chamado $at, é reservado para o montador (veja Seção 3.9), e os registradores 26–27, chamados $k0–$k1, são reservados para o sistema operacional.

Operandos do MIPS

Nome	Exemplo	Comentários
32 registradores	`$s0–$s7`, `$t0–$t9`, `$zero`, `$a0–$a3`, `$v0–$v1`, `$gp`, `$fp`, `$sp`, `$ra`	Posições de acesso rápido para armazenamento de dados. No MIPS, os dados devem estar em registradores para que as operações aritméticas possam ser realizadas. Os registradores `$s0–$s7` são mapeados nos registradores reais de número 16–23 e os registradores `$t0–$t7` nos de número 8–15. O registrador `$zero` do MIPS sempre tem o valor `0` armazenado nele. **O registrador `$gp` (28) é o ponteiro global, o `$sp` (29) é o apontador de pilha, o `$fp` (30) é o ponteiro de frame, e o registrador `$ra` (31) guarda o valor do endereço de retorno.**
2^{30} palavras de memória	Memória[0], Memória[4], ..., Memória[4294967292]	No MIPS, estas posições só são acessadas por instruções de transferência de dados. O MIPS endereça byte, de modo que endereços de palavras consecutivas diferem de 4 unidades. A memória armazena estruturas de dados, como arrays, além dos registradores "derramados", **tais como os salvos na chamada a procedimentos.**

Linguagem de montagem do MIPS

Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	add	`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`	Três operandos; dados em registradores
	subtract	`sub $s1, $s2, $s3`	`$s1 = $s2 – $s3`	Três operandos; dados em registradores
Transferência de dados	load word	`lw $s1, 100 ($s2)`	`$s1 = Memória[$s2 + 100]`	Dados transferidos da memória para registradores
	store word	`sw $s1, 100 ($s2)`	`Memória[$s2 + 100] = $s1`	Dados transferidos de registradores para a memória
Desvio condicional	branch on equal	`beq $s1, $s2, L`	`se ($s1 == $s2)` desvia para L	Testar a igualdade e desviar se verdadeira
	branch on not equal	`bne $s1, $s2, L`	`se ($s1 != $s2)` desvia para L	Testar a desigualdade e desviar se verdadeira
	set on less than	`slt $s1, $s2, $s3`	`se ($s2 < $s3) $s1 = 1;` senão `$s1 = 0`	Compare se menor ou igual; usada junto a `beq` e a `bne`
Desvio incondicional	jump	`j 2500`	`desvia para 10000`	Desviar para o endereço-alvo
	jump register	`jr $ra`	`desvia para $ra`	Para comandos switch e **retorno de procedimentos**
	jump and link	`jal 2500`	**`$ra = PC + 4;` desvia para 10000**	**Para chamadas a procedimentos**

Linguagem de máquina do MIPS

Nome	Formato	Exemplo						Comentários
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)
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 (veja Seção 3.8)**
jr	**R**	**0**	**9**	**0**	**0**	**0**	**8**	**jr $t1**
Tamanho do campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções do MIPS têm 32 bits
Formato R	R	op	rs	rt	rd	shamt	funct	Formato das instruções aritméticas
Forrmato I	I	op	rs	rt	endereço			Formato das **instruções** de transferência de dados

Além dos números

- Como representar textos?

- A maioria das máquinas utiliza 1 byte (8 bits) para representar caracteres internamente.
  - utilizando o chamado código ASCII (American Standard Code for Information Interchange)

Além dos número

A tabela ASCII

Valor em ASCII	Caractere	Valor em ASCII	Caractere	Valor em ASCII	Caractere	Valor em ASCII	Caractere	Valor em ASCII	Caractere	Valor em ASCII	Caractere
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

Figura 3.15 Representação de caracteres em ASCII. Note que os mesmos caracteres, quando expressos em maiúsculas e em minúsculas, diferem exatamente de 32 unidades; esta observação pode cortar caminho na verificação e na troca da caixa de caracteres em geral. Os caracteres de formatação não estão nesta tabela. Por exemplo, 9 representa o caractere tab e 13 o retorno de carro. Outros caracteres úteis são o 8, que representa o backspace, e o 0 para o null, valor este que é usado nos programas em C para marcar o fim de um string.

Além dos número

- O MIPS possui instruções especiais para mover bytes.
  - `Load byte – lb`
    - Carrega um byte da memória e armazena seu conteúdo nos oito bits mais à direita (`bigendian`) do registrador destino indicado na própria instrução
  - `Store byte – sb`
    - Pega o byte mais a direita do registrador indicado na instrução e armazena seu conteúdo na memória.
- Os caracteres são combinados normalmente em strings
- Cada string possui possuindo um número variável de caracteres
- Existem três escolhas possíveis para representar uma string
  1. A primeira posição da string é reservada para fornecer o tamanho da mesma
  2. Uma variável associada à string informa o seu tamanho
  3. A última posição da string é usada por um caracter que marca o final da string

Além dos números

- Exemplo

```
void strcpy (char x[], char y[])
{
     int i;
     i=0;
     while ((x[i]=y[i]) != 0)
          i = i + 1;
}
```

```
   Strcpy:
        addi $sp, $sp, -4     #ajusta a pilha para receber mais 1 item
        sw $s0, 4($sp)        #salva $s0
        add $s0, $zero, $zero #i = 0 + 0
         L1:add $t1, $a1, $s0     #endereço de y[i] vai para $t1
        lb $t2, 0($t1)        #reg $t2 recebe y[i]
        add $t3, $a0, $s0         #endereco de x[i] vai para $t3
        sb $t2, 0($t3)        #x[i] recebe y[i]
        add $s0, $s0, 1       #i = i + 1
        bne $t2, $zero, L1  #se y[i] for diferente de 0, desvia para
   L1
        lw $s0, 4($sp)        # restaura o valor antigo de S0
        add $sp, $sp, 4
        jr $ra
```

Reflexão

“Nunca ande pelo caminho traçado, pois ele conduz somente até onde os outros foram!”

Referências

- Hennessy, J. L., Patterson, D. A. *Organização e projeto de computadores: A Interface hardware/software*. 2 ed, Rio de Janeiro, LTC, 2000.