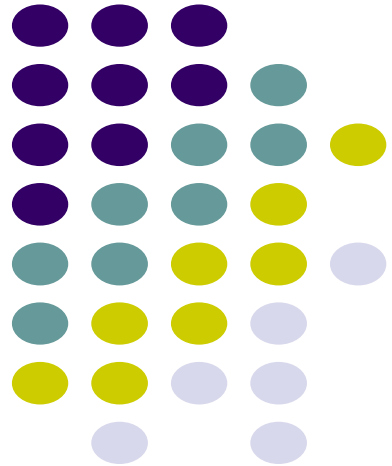
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Arquitetura e Organização de Computadores

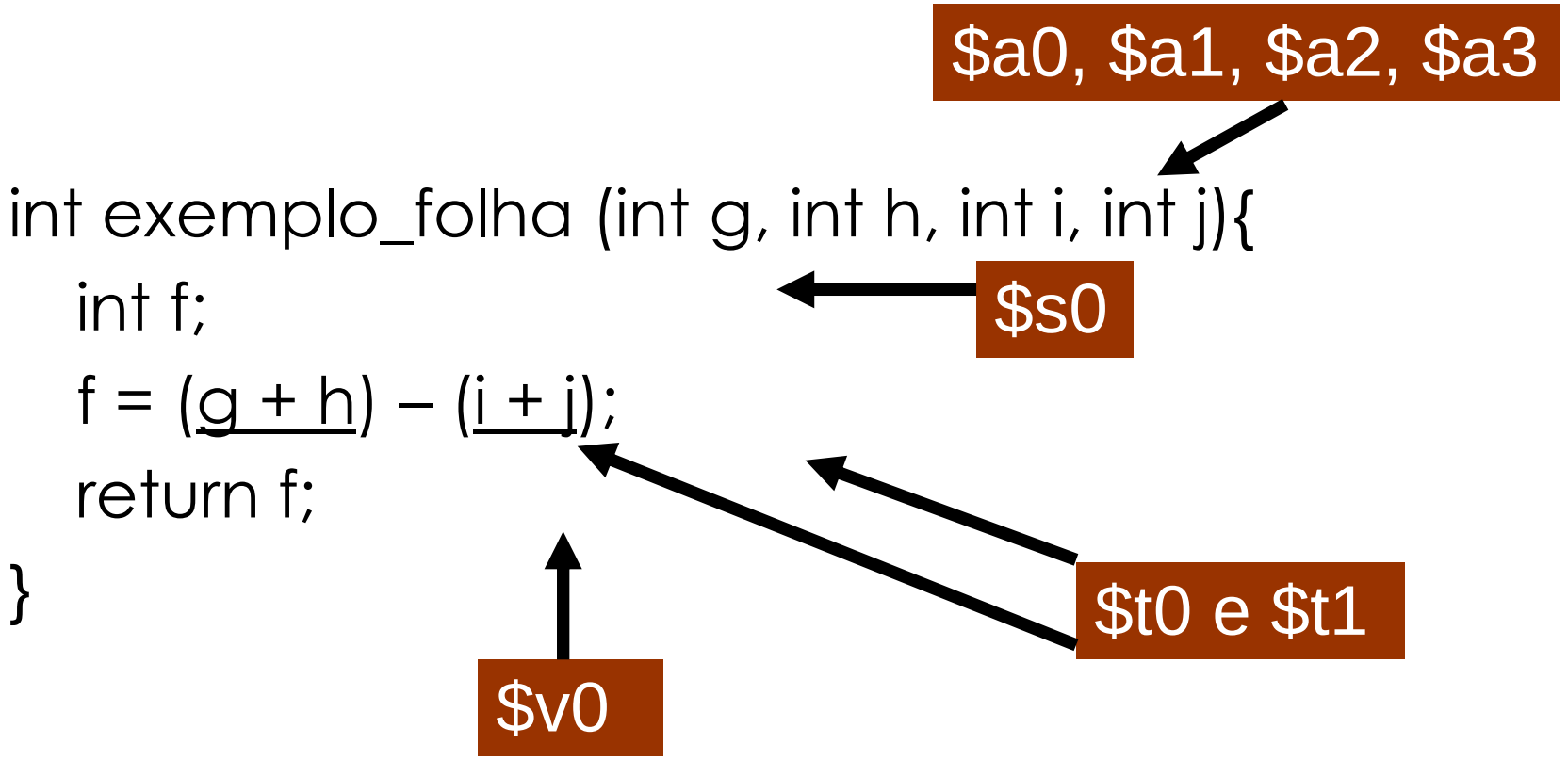
Instruções: a linguagem do computador
Parte V



Prof. Sílvio Fernandes

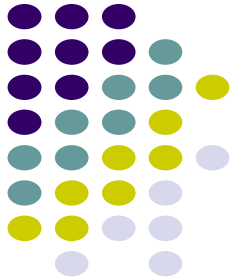
Anteriormente...

- Fazendo o mapeamento dos elementos da função para registradores, teremos:



Anteriormente...

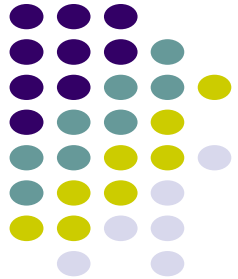
```
int exemplo_folha (int g, int h, int i, int j){
        int f;
        f = (g + h) – (i + j);
        return f;
}
```



Alguém chama jal exemplo_folha

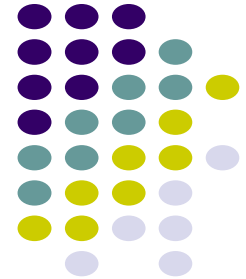
```
exemplo_folha:
# ajusta a pilha e salva reg.
addi $sp, $sp, -12
sw $t1, 8($sp)
sw $t0, 4($sp)
sw $s0, 0($sp)
add $t0, $a0, $a1    # $t0 = g+h
add $t1, $a2, $a3    # $t1 = i + j
sub $s0, $t0, $t1    # $s0 = (g+h) – (i + j)
#Copiando o valor de f para ser retornado
add $v0, $s0, $zero
#Restraurando reg. da pilha
lw $s0, 0($sp)
lw $t0, 4($sp)
lw $t1, 8($sp)
addi $sp, $sp, 12
# E finalmente, o procedimento termina
jr $ra     # Volta à rotina que chamou
```

O que é preservado e o que não é em chamada de procedimentos



Preserved	Not preserved
Saved registers: `$s0–$s7`	Temporary registers: `$t0–$t9`
Stack pointer register: `$sp`	Argument registers: `$a0–$a3`
Return address register: `$ra`	Return value registers: `$v0–$v1`
Stack above the stack pointer	Stack below the stack pointer

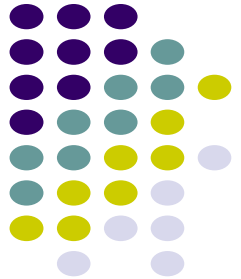
Suporte a procedimentos no hardware do computador



- O código mostrado anteriormente é perfeitamente aceitável para funções que não fazem chamadas a outras funções.
- Funções que não fazem chamadas a outras funções são denominadas folhas. As que chamam outras são denominadas aninhadas.
- E para funções aninhadas, como seria?

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

Suporte a procedimentos no hardware do computador



#Salvando registradores

fact:

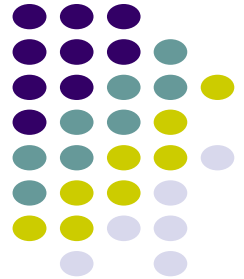
addi $sp, $sp, -8 #ajusta pilha para 2 itens

sw $ra, 4($sp) # salva o endereço de retorno

sw $a0, 0($sp) # salva o argumento n

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

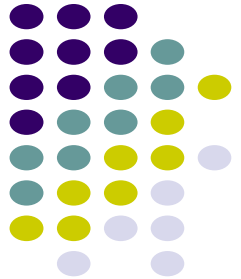
Suporte a procedimentos no hardware do computador



```
#Condição
slti   $t0, $a0, 1        # teste para n < 1
beq $t0, $zero, L1        # se n>=1 vai para L1
```

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

Suporte a procedimentos no hardware do computador

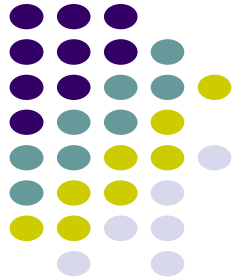


```
#Senão for maior que 1, devolve o valor 1.
addi $v0, $zero, 1 # retorna 1
addi $sp, $sp, 8     # retira 2 itens da pilha
jr      $ra                #retorna para depois de jal
```

Pessoal, não entendi. Alguém me responda. Porque não restauramos os valores da pilha nesse caso?

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

Suporte a procedimentos no hardware do computador

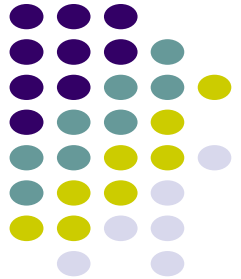


```
#Senão for maior que 1, devolve o valor 1.
addi $v0, $zero, 1 # retorna 1
addi $sp, $sp, 8     # retira 2 itens da pilha
jr      $ra                #retorna para depois de jal
```

Observem que no caso base, o valor do registrador $ra e $a0 não é alterado, logo não é necessário recuperar os valores dele da memória

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

Suporte a procedimentos no hardware do computador

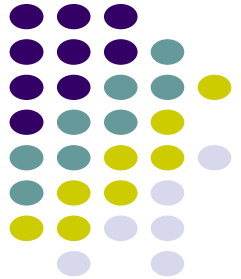


- #Se for maior que 1

```
L1: addi $a0, $a0, -1   #arg1 = n – 1;
    jal    fact         #chama fact(n-1);
```

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

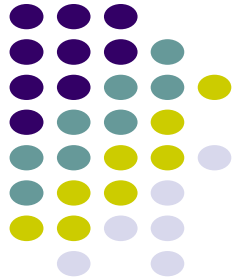
Suporte a procedimentos no hardware do computador



```
#Restaurando registradores.
#A próxima instrução é onde fact retorna.
lw $a0, 0($sp)      #retorna de jal: restaura n
lw $ra, 4($sp)      #restaura endereço de retorno
addi $sp, $sp, 8    #ajusta stack pointer
```

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

Suporte a procedimentos no hardware do computador



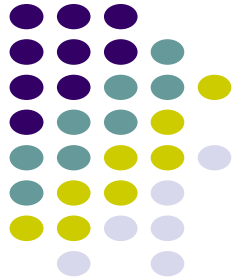
#Devolvendo o novo $v0

mul $v0, $a0, $v0 # retorna n * fact(n - 1)

jr $ra # retorna para o procedimento que o chamou

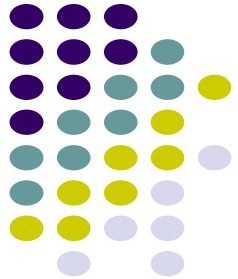
```
int fact (int n){
        if (n < 1) return (1);
            else return (n * fact(n – 1));
    }
```

Suporte a procedimentos no hardware do computador



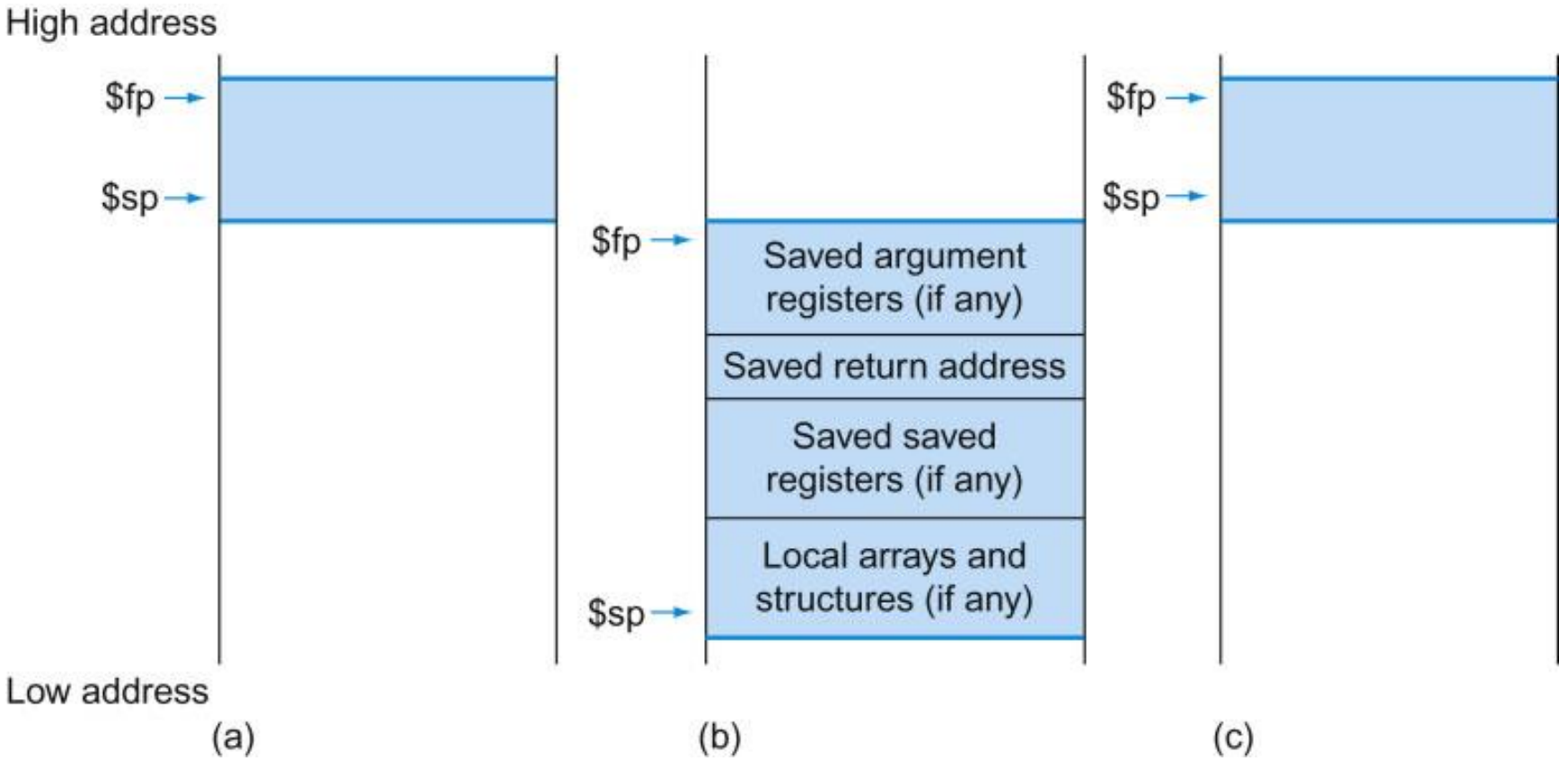
- Para resumo de história, quem salva o que na pilha?
- **Caller**
  - Salva os registradores de argumentos.
  - Salva os registradores temporários ($t0-$t9) que serão necessários após a chamada da função.
- **Callee**
  - Salva os registradores salvos ($s0-$s8)
  - Salva o registrador que armazena o endereço de retorno ($ra)

Suporte a procedimentos no hardware do computador

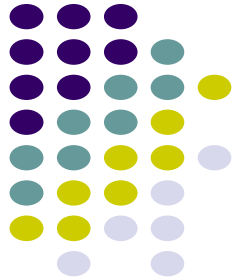


- Além dos elementos já citados, a pilha também pode ser utilizada para armazenar outros elementos.
  - Alguns exemplos são variáveis locais (tais como arrays ou estruturas) que não cabem em registradores.
- Este segmento da pilha relativo a um procedimento é conhecido como ***frame*** ou ***registro de ativação***.
- Em geral os processadores possuem um registrador específico para apontar para o início do frame, conhecido como ***frame pointer*** ($fp).

Suporte a procedimentos no hardware do computador

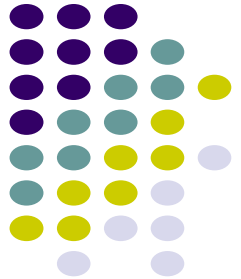


Suporte a procedimentos no hardware do computador



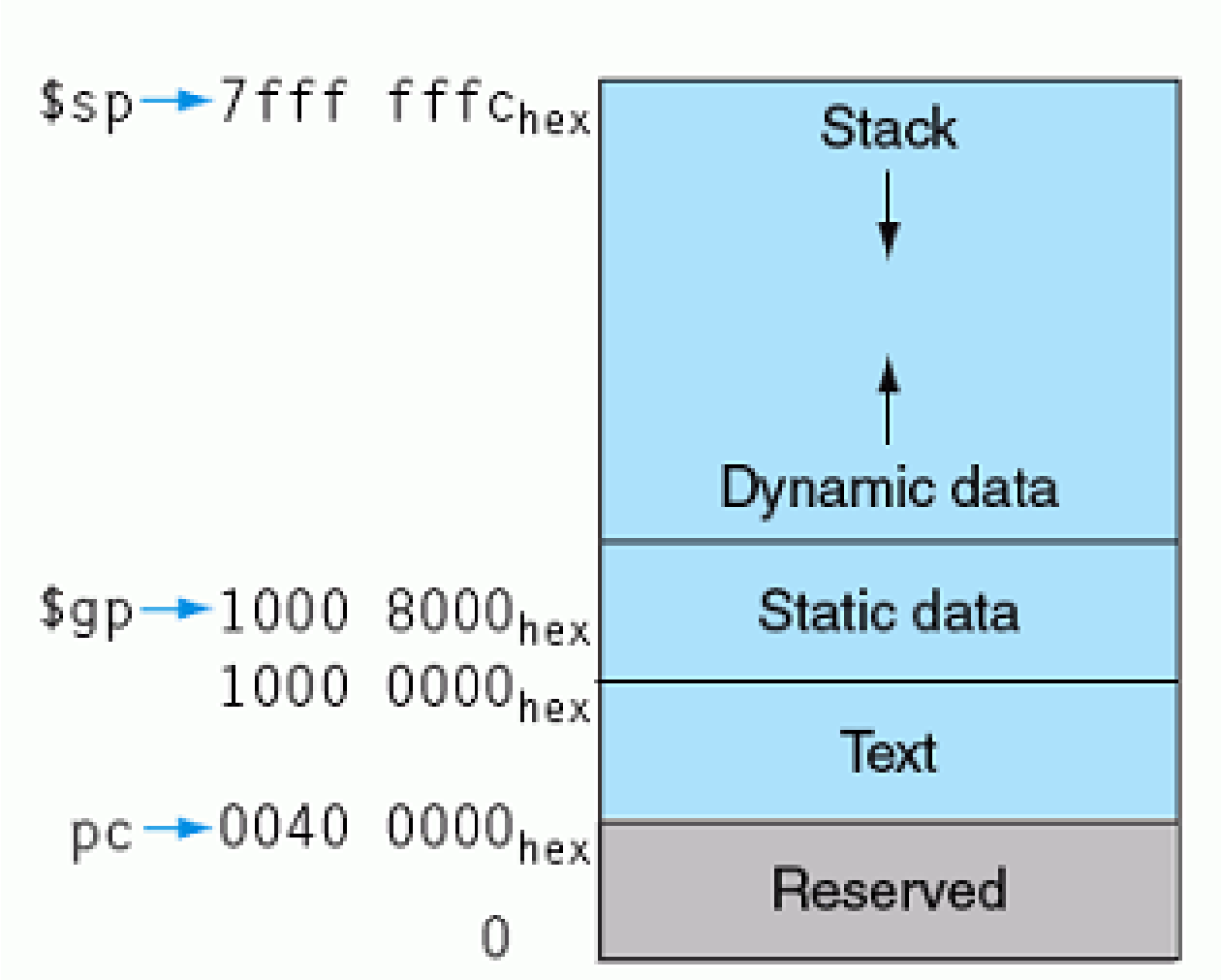
- O frame pointer, assim como o stack pointer, é utilizado nas instruções que alocam e recuperam dados do frame.
- Mas se ambos registradores possuem a mesma funcionalidade, então para que termos o frame pointer?
  - O frame pointer, ao contrário do stack pointer, não tem seu valor alterado durante o procedimento. Logo se torna mais fácil recuperar dados utilizando o valor do stack pointer.

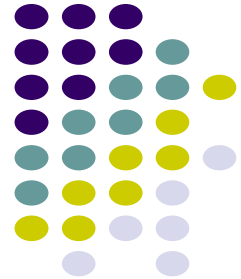
Suporte a procedimentos no hardware do computador



- O assunto que vimos até o momento, é aplicado apenas para variáveis automáticas (ou seja) locais aos procedimentos.
- No entanto ainda faltam tratar dois casos:
  - Variáveis globais (ou estáticas).
  - Estruturas de dados dinâmicas.
- Estes elementos, quando em memória, não são alocados no stack.
- Os dados estáticos podem ser acessados através do registrador **$gp**.
- Os dados dinâmicos é armazenado em uma região de memória denominada heap.

Suporte a procedimentos no hardware do computador

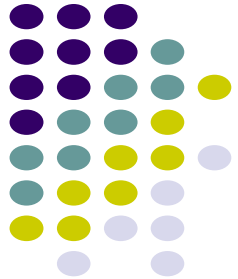




Comunicando-se com pessoas

Vocês conseguem ler isso?

!(@ | = >



Vocês conseguem ler isso?

!(@ | = >

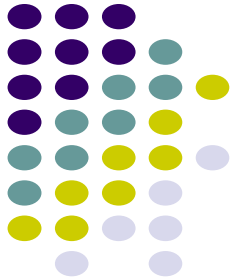
wow open tab at bar is great

Quarta linha do poema de teclado “Hatless Atlas”, 1991

Em português seria algo do tipo

“ah aba aberta no bar é ótimo”

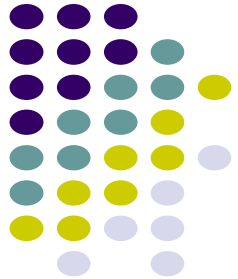
Comunicando-se com as pessoas



- A maioria dos computadores hoje utiliza bytes de 8 bits para representar caracteres obedecendo um padrão (como ASCII)

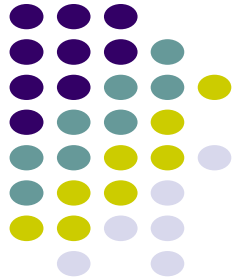
ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter	ASCII value	Char-acter
32	space	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	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

Comunicando-se com as pessoas



- MIPS oferece instruções para mover bytes
- Load byte (**lb**) lê um byte da memória, colocando-o nos 8 bits mais à direita de um registrador
- Store byte (**sb**) separa o byte mais à direita de um registrador e o escreve na memória
- Sintaxe do lb e sb
  - lb $t0, 0($sp) # lê byte da origem
  - sb $t0, 0($gp) # escreve byte no destino

Comunicando-se com as pessoas

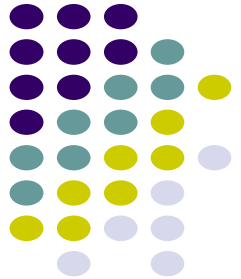


- Qual é o código correspondente no MIPS para

```
void strcpy(char x[], char y[]){
    int i = 0;
    while((x[i] = y[i]) != '\0') /* copia e testa byte*/
        i += 1;
}
```

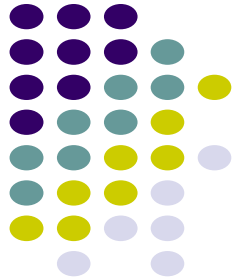
Considere os endereços base de x e y encontrados em $a0 e $a1, enquanto i está em $s0

Comunicando-se com as pessoas



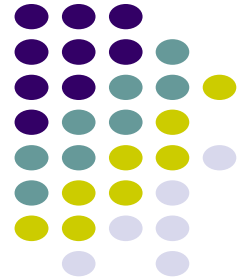
```
strcpy:
        addi    $sp, $sp, -4        #ajusta pilha para mais 1 item
        sw      $s0, 0($sp)         #salva $s0
        add     $s0, $zero, $zero # i = 0+0
L1:     add     $t1, $s0, $a1       #endereço de y[i] em $t1
        lb      $t2, 0($t1)         # $t2 = y[i]
        add     $t3, $s0, $a0       #endereço de x[i] em $t3
        sb      $t2, 0($t3)         # x[i] = y[i]
        beq     $t2, $zero, L2      #se y[i] == 0, vai para L2
        addi    $s0, $s0, 1         # i = i+1
        j       L1                  # vai para L1
L2:     lw      $s0, 0($sp)         # y[i] == 0; fim da string; restura $s0
        addi    $sp, $sp, 4         #retira 1 word da pilha
        jr      $ra                 #retorna
```

Comunicando-se com as pessoas



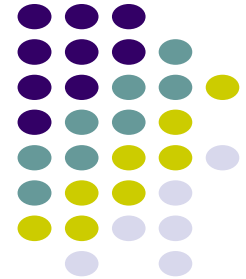
- MIPS possui instruções para carregar e armazenar quantidades de 16 bits, chamadas *halfwords* (padrão Unicode)
- Load half (**lh**) lê uma *halfword* da memória, colocando-a nos 16 bits mais à direita de um registrador
  - lh $t0, 0($sp) #lê halfword (16 bits) da origem
- Store half (**sh**) separa a *halfword* correspondente aos 16 bits mais à direita de um registrador e escreve na memória
  - sh $t0, 0($gp) #escreve halfword (16 bits) no destino

Endereçamento no MIPS para operandos e endereços de 32 bits

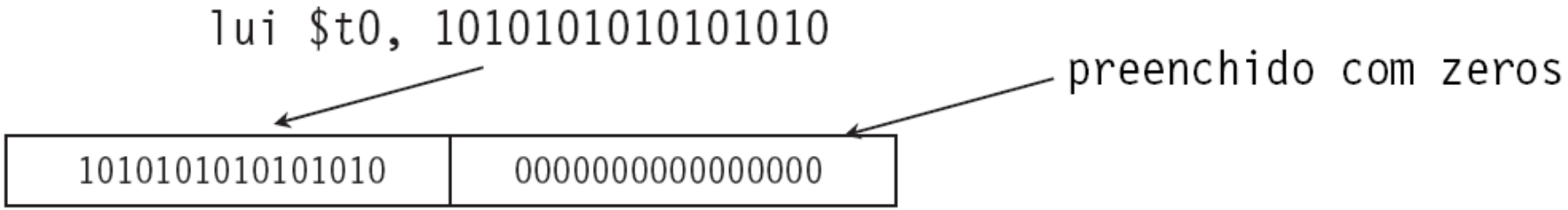


- Até o momento, consideramos que endereços de memória já estariam em registradores. Lembrem-se que cada endereço de memória é representado por 32 bits.
- Lembrando que as instruções com constantes (imediatas) do MIPS só trabalham com 16 bits.
- Como fazemos para carregar constate de 32 bits no registrador $s0?

Endereçamento no MIPS para operandos e endereços de 32 bits

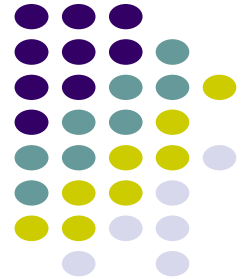


- Precisamos usar duas instruções; nova instrução "*load upper immediate*":



- Binário não é reconhecido no MARS
- Uma forma de escrever valores "grande" é com representação hexa: início "0x"
  - lui $t1, 0x003D #$t1 = 003D 0000 (em hexa)

Endereçamento no MIPS para operandos e endereços de 32 bits



- Depois, precisamos acertar os bits de ordem inferior, por exemplo:

```
ori $t0, $t0, 1010101010101010
```

1010101010101010	0000000000000000
0000000000000000	1010101010101010

ori

1010101010101010	1010101010101010

Referências

- PATTERSON, D. A. ; HENNESSY, J.L. Organização e projeto de computadores – a interface hardware software. 3. ed. Editora Campus, 2005.
- Notas de aula do Prof. André Luis Meneses Silva