Arquitetura MIPS

Conjunto de instruções MIPS

Organização e Projeto de Computadores: A Interface Hardware/Software.
David A. Patterson e John L. Hennessy

Conjunto de instruções

- Instrução é uma palavra da linguagem de máquina
  - Neste caso, com 32 bits.
- ISA (*Instruction Set Architecture*)
  - Conjunto de instruções de uma máquina
- ISA MIPS - RISC
  - 3 formatos de instruções
  - instruções de (no máximo) 3 operandos

Programa em C	Assembly MIPS
a = b + c; d = a – c;	add a,b,c sub d,a,c

f = (g + h) – (i + j);	add t0,g,h
	add t1,i,j
	sub f,t0,t1
	(O compilador cria t0 e t1)

Operandos

- No MIPs, os registradores podem ser usados como operandos
  - 32 registradores de propósito geral de 32 bits identificados por $;
  - E se não for suficiente? Temos que usar a memória.

Programa em C	Assembly MIPS

f = (g + h) – (i + j);	add $t0,$s1,$s2
	add $t1,$s3,$s4
	sub $s0,$t0,$t1

Variáveis	Registradores
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4

Instruções de movimentação dos dados

- Load e Store

- lw: instrução de movimentação de dados da memória para registrador (*load word*)
  - Sintaxe: lw $destino, deslocamento($origem)

- sw: instrução de movimentação de dados do registrador para a memória (*store word*)
  - Sintaxe: sw $fonte, deslocamento($origem)

Exemplo

Seja A um array de 100 palavras. O compilador associou à variável g o registrador $s1 e a h $s2, além de colocar em $s3 o endereço base do vetor. Traduza o comando em C abaixo.

```
g = h + A[8];
```

Exemplo

Seja A um array de 100 palavras. O compilador associou à variável g o registrador $s1 e a h $s2, além de colocar em $s3 o endereço base do vetor. Traduza o comando em C abaixo.

g = h + A[8];

Solução:

Primeiro devemos carregar um registrador temporário com A[8]:

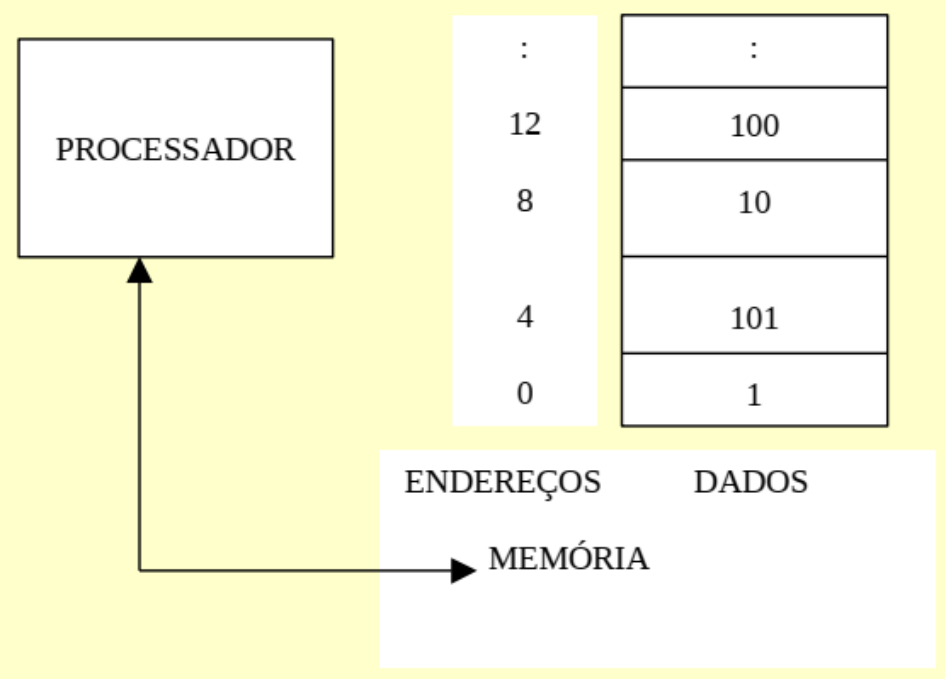
```
lw $t0, 8($s3)     # registrador temporário $t0 recebe A[8]
```

Agora basta executar a operação:

```
add $s1,$s2,$t0     # g = h + A[8]
```

MIPS - Organização da Memória

No MIPS a memória é organizada em bytes, embora o endereçamento seja em palavras de 4 bytes (32 bits) para dados e instruções.



Exemplo

Suponha que h seja associado com o registrador $s2 e o endereço base do array A armazenado em $s3. Qual o código MIPS para o comando abaixo?

A[12] = h + A[8];

Exemplo

Suponha que h seja associado com o registrador $s2 e o endereço base do array A armazenado em $s3. Qual o código MIPS para o comando abaixo?

A[12] = h + A[8];

Solução:

```
lw $t0,32($s3) # $t0 recebe A[8]

add $t0,$s2,$t0 # $t0 recebe h + A[8]

sw $t0,48($s3) # armazena o resultado em A[12]
```

Sintaxes:
lw $destino, deslocamento($origem)
sw $fonte, deslocamento($origem)

Exemplo

Supor que o índice seja uma variável:

g = h + A[i];

onde: i é associado a $s4, g a $s1, h a $s2 e o endereço base de A a $s3.

Exemplo

Supor que o índice seja uma variável:

g = h + A[i];

onde: i é associado a $s4, g a $s1, h a $s2 e o endereço base de A a $s3.

Solução:

```
add $t1,$s4,$s4
add $t1,$t1,$t1 # $t1 recebe 4*i ( porque??? )
add $t1,$t1,$s3 # $t1 recebe o endereço de A[i]
lw $t0,0($t1) # $t0 recebe a[i]
add $s1,$s2,$t0
```

Conjunto de Instruções MIPS (Parcial)

MIPS operands

Name	Example	Comments
32 registers	$s0, $s1, . . . , $t0, $t1, . . .	Fast locations for data. In MIPS, data must be in registers to perform arithmetic.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions in MIPS. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers.

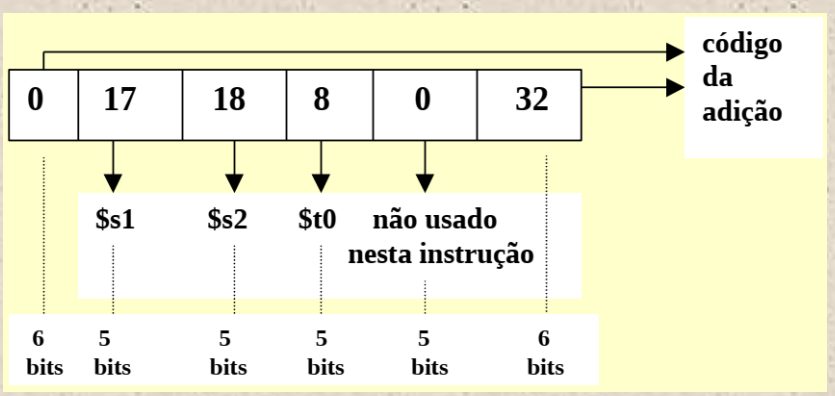
MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	three operands; data in registers
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	three operands; data in registers
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

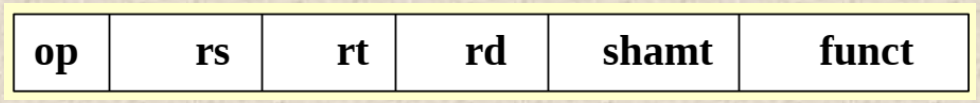
FIGURE 3.4 MIPS architecture revealed through section 3.3. Highlighted portions show MIPS assembly languag structures introduced in section 3.3.

Formato de instruções

Formato da instrução add $t0,$s1,$s2



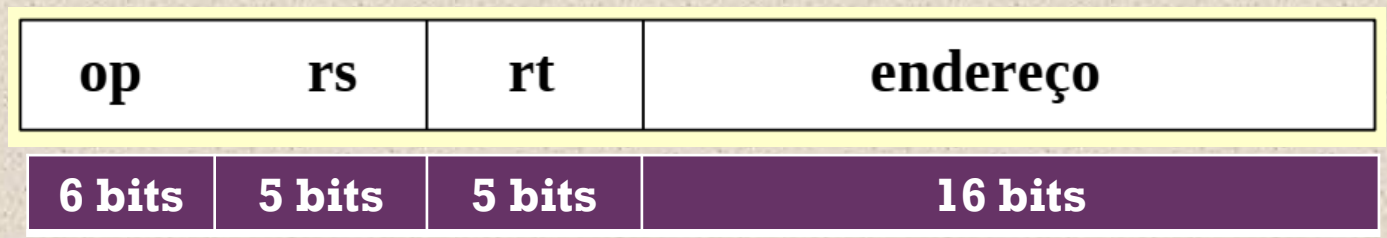
Formato das instruções tipo R (de registrador) e seus campos



- op: operação básica da instrução (opcode)
- rs: o primeiro registrador fonte
- rt: o segundo registrador fonte
- rd: o registrador destino
- shamt: *shift amount*, para instruções de deslocamento
- funct: *function*, que seleciona variações das operação especificadas pelo opcode, como sua extensão.

Formato de instruções

- Formato das Instruções tipo I (imediato)



- Exemplo de instruções I-type
  - lw $t0, 32($s3)

- Codificação de Instruções MIPS

Instrução	Formato	Op	rs	rt	rd	Sham t	func	end.
Add	R	0	reg	reg	reg	0	32	n.d
Sub	R	0	reg	reg	reg	0	34	n.d
Lw	I	35	reg	reg	n.d.	n.d	n.d	end.
Sw	I	43	reg	reg	n.d	n.d	n.d	end.

Exemplo

Escreva o código assembly do MIPS e o código de máquina para o seguinte comando em C: "**A[300] = h + A[300];**", onde $t1 tem o endereço base do vetor A e $s2 corresponde a h.

Exemplo

Escreva o código assembly do MIPS e o código de máquina para o seguinte comando em C: "**A[300] = h + A[300];**", onde $t1 tem o endereço base do vetor A e $s2 corresponde a h.

Solução:

lw $t0,1200($t1) # $t0 recebe A[300]

add $t0,$s2,$t0 # $t0 recebe h + A[300]

sw $t0,1200($t1) # A[300] recebe h + A[300]

Linguagem de máquina:

Instrução	Formato	Op	rs	rt	rd	Shamt	func	end.
Add	R	0	reg	reg	reg	0	32	n.d
Sub	R	0	reg	reg	reg	0	34	n.d
Lw	I	35	reg	reg	n.d.	n.d	n.d	end.
Sw	I	43	reg	reg	n.d	n.d	n.d	end.

Op	rs	rt	rd	end/shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

MIPS operands

Name	Example	Comments
32 registers	`$s0, $s1, ..., $s7` `$t0, $t1, ..., $t7`	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. Registers `$s0–$s7` map to 16–23 and `$t0–$t7` map to 8–15.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Accessed only by data transfer instructions in MIPS. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`	Three operands; data in registers
	subtract	`sub $s1,$s2,$s3`	`$s1 = $s2 - $s3`	Three operands; data in registers
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

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)`
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 format

FIGURE 3.6 MIPS architecture revealed through section 3.4. Highlighted portions show MIPS machine language structures introduced in section 3.4. The two MIPS instruction formats so far are R and I. The first 16 bits are the same: both contain an *op* field, giving the base operation; an *rs* field, giving one of the sources; and the *rt* field, which specifies the other source operand, except for load word, where it specifies the destination register. R-format divides the last 16 bits into an *rd* field, specifying the destination register; *shamt* field, which is unused in Chapter 3 and hence always is 0; and the *funct* field, which specifies the specific operation of R-format instructions. I-format keeps the last 16 bits as a single *address* field.

Instruções de desvio condicional

- beq registrador1, registrador2, L1
  - se o valor do registrador1 for igual ao do registrador2, o programa será desviado para o *label* L1
  - (beq = *branch if equal*).

- bne registrador1, registrador2, L1
  - se o valor do registrador1 não for igual ao do registrador2, o programa será desviado para o *label* L1
  - (bne = *branch if not equal*).

Exemplo

Compilando um comando IF.

Seja o comando abaixo:

```
    if ( i == j ) go to L1;
    f = g + h;
L1: f = f - i;
```

Variáveis	Registradores
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4

Supondo que as 5 variáveis correspondam aos registradores $s0..$s4, respectivamente, como fica o código MIPS para o comando?

Exemplo

Compilando um comando IF.

Seja o comando abaixo:

```
    if ( i == j ) go to L1;
    f = g + h;
L1: f = f - i;
```

Variáveis	Registradores
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4

Supondo que as 5 variáveis correspondam aos registradores $s0..$s4, respectivamente, como fica o código MIPS para o comando?

Solução:

```
beq $s3,$s4,L1      # vá para L1 se i = j
add $s0,$s1,$s2     # f = g + h, executado se i != j
L1: sub $s0,$s0,$s3 # f = f – i, executado se i = j
```

Instrução de desvio incondicional

J L1 (*jump* sem registrador, só imediato)

- quando executado faz com que o programa seja desviado para L1

Exemplo – Compile o comando if-then-else

Seja o comando abaixo:

```
if (i == j)
    f = g + h;
else
    f = g – h;
```

Variáveis	Registradores
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4

Instrução de desvio incondicional

J L1

- quando executado faz com que o programa seja desviado para L1

Exemplo – Compile o comando if-then-else

Seja o comando abaixo:

```
if (i == j)
    f = g + h;
else
    f = g – h;
```

Variáveis	Registradores
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4

Solução:

```
        bne $s3,$s4,Else     # vá para Else se i != j
        add $s0,$s1,$s2      # f = g + h, se i == j
        j Exit               # vá para Exit
Else: sub $s0,$s1,$s2        # f = g – h, se i != j
Exit:
```

Loops

Usando IF

Exemplo

```
Loop:  g = g + A[i];
       i = i + j;
       if ( i != h ) go to Loop
```

Variáveis	Registradores
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4
A	$s5

Loops

Usando IF

Exemplo

```
Loop:  g = g + A[i];
       i = i + j;
       if ( i != h ) go to Loop
```

Variáveis	Registradores
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4
A	$s5

Solução:

```
Loop: add $t1,$s3,$s3      # $t1 = 2 * i
      add $t1,$t1,$t1      # $t1 = 4 * i
      add $t1,$t1,$s5      # $t1 recebe endereço de A[i]
      lw $t0,0($t1)        # $t0 recebe A[i]
      add $s1,$s1,$t0      # g = g + A[i]
      add $s3,$s3,$s4      # i = i + j
      bne $s3,$s2,Loop     # se i != h vá para Loop
```

Loops

Usando while

Exemplo:

```
while(save[i]==k)
    i = i + j
```

Variáveis	Registradores
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4
k	$s5
save	$s6

Loops

Usando while

Exemplo:

```
while(save[i]==k)
    i = i + j
```

Variáveis	Registradores
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4
k	$s5
save	$s6

Solução:

```
Loop: add $t1,$s3,$s3      # $t1 = 2 * i
      add $t1,$t1,$t1      # $t1 = 4 * i
      add $t1,$t1,$s6      # $t1 = endereço de save[i]
      lw $t0,0($t1)        # $t0 recebe save[i]
      bne $t0,$s5,Exit     # va para Exit se save[i] != k
      add $s3,$s3,$s4      # i = i + j
      j Loop
Exit:
```

Instruções para teste de maior ou menor

- slt reg_temp, reg1, reg2
- slt = *set less than*
  - se reg1 é menor que reg2, reg_temp é setado, caso contrário é resetado;
  - Sem desvio automático, sendo necessária alguma instrução de desvio;
  - Nos processadores MIPS, o registrador $0 possui o valor zero ($zero).

Exemplo

Exemplo: Compilando o código abaixo

```
if(a < b) {
        …
}
```

Variáveis	Registradores
a	$s0
b	$s1

Exemplo

Exemplo: Compilando o código abaixo

```
if(a < b) {
        …
}
```

Variáveis	Registradores
a	$s0
b	$s1

Solução:

```
slt $t0,$s0,$s1              # $t0 é setado se $s0 < $s1
beq $t0,$zero,NotLess   # vá para NotLess, se $t0 == 0 , ou seja a≥b
```

Instruções Switch-Case

Instruções Switch-Case contém linhas dos cases em uma tabela **na memória**.

```
switch(k) {
    case 0: ...; break;
    case 1: ...; break;
    case 2: ...; break;
    ...
    default: ...;
}
```

K	Label 0
K+4	Label 1
K+8	Label 2
...	
K+32	Label x

Por isso, a variável k precisa ser inteira.

Exemplo

Seja o comando abaixo:

```
switch (k) {
    case 0: f = i + j; break;
    case 1: f = g + h; break;
}
```

Obs.:

1) Supor que registrador temporário contém posição inicial da tabela de desvio (a posição de k);

2) Instrução *Jump Register*: jr Reg

Realiza um desvio para o endereço da linha contida no registrador Reg;

3) Use a constante 2 pré atribuída para um registrador temporário.

Exemplo

Solução: supor que $t2 tenha 2 e f..k = $s0..$s5, respectivamente.

```
    slt $t3,$s5,$zero      # teste se k < 0
    bne $t3,$zero,Exit     # se k < 0 vá para Exit
    slt $t3,$s5,$t2        # teste se k < 2
    beq $t3,$zero,Exit     # se k>=2 vá para Exit
    add $t1,$s5,$s5        # $t1 = 2 * k
    add $t1,$t1,$t1        # $t1 = 4 * k
    # assumindo que 4 palavras na memória, começando no
    endereço contido em $t4, tem endereçamento
    correspondente a L0, L1, L2
    add $t1,$t1,$t4        # $t1 = endereço de tabela[k]
    lw $t0,0($t1)          # $t0 = tabela[k]
    jr $t0                 # salto para endereço carregado em $t0
L0: add $s0,$s3,$s4        # k = 0 : f = i + j
    j Exit
L1: add $s0,$s1,$s2        # k = 1 : f = g + h
Exit:
```

```
switch (k) {
    case 0: f = i + j;
break;
    case 1: f = g + h;
break;
}
```

Var.	Regs
f	$s0
g	$s1
h	$s2
i	$s3
j	$s4
k	$s5
2	$t2

Conjunto de Instruções MIPS (Parcial)

MIPS operands

Name	Example	Comments
32 registers	$s0, $s1, . . . , $s7 $t0, $t1, . . . , $t7, $zero	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. Registers $s0–$s7 map to 16–23 and $t0–$t7 map to 8–15. MIPS register $zero always equals 0.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions in MIPS. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; data in registers
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
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
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $t1	go to $t1	For *switch* statements

Conjunto de Instruções MIPS (Parcial)

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)
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 3.8)
jr	R	0	9	0	0	0	8	jr $t1
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

FIGURE 3.9 MIPS architecture revealed through section 3.5. Highlighted portions show MIPS structures introduced in section 3.5. The J-format, used for jump instructions, is explained in section 3.8. Section 3.8 also explains the proper values in address fields of branch instructions.

Suporte a Procedimentos

- Para a execução de um procedimento deve-se:
  - Colocar os parâmetros em um local onde o procedimento possa acessá-los;
  - Transferir o controle ao procedimento;
  - Adquirir os recursos necessários ao procedimento;
  - Executar a tarefa;
  - Colocar o resultado em um local onde o programa possa acessá-lo;
  - Retornar o controle ao ponto onde o procedimento foi chamado.

Suporte a Procedimentos

- Para este mecanismo, o MIPS aloca seus registradores, para chamada de procedimentos, da seguinte maneira:
  - $a0 .. $ a3: 4 registradores para passagem de argumentos;
  - $v0 .. $v1: para retornar valores (com mais de um *return*);
  - $ra: (*return address*) para guardar o endereço de retorno.
- Instrução para chamada de procedimento
  - jal End_proc - (*jump-and-link*): desvia para o procedimento e salva o endereço de retorno (PC+4) em $ra (return address - $31)
- Instrução para retorno de chamada de procedimento
  - jr $ra: desvia para o ponto de onde foi chamado o procedimento

Exemplo

- Seja o procedimento abaixo:
- Os parâmetros g, h, i e j correspondem a $a0 .. $a3, respectivamente, e f a $v0.

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

Variáveis	Registradores
f	$v0
g	$a0
h	$a1
i	$a2
j	$a3

Exemplo

```
int exemplo (int g, int h, int i, int j)
{
        int f;

        f = (g + h) – (i + j);
        return f;
}
```

Proc: add $t0,$a0,$a1

add $t1,$a2,$a3

sub $v0,$t0,$t1

Retornar

jr $ra

Variáveis	Registradores
f	$v0
g	$a0
h	$a1
i	$a2
j	$a3

Observações

- $t0 .. $t9: 10 registradores temporários que não são preservados em uma chamada de procedimento;
- $s0 .. $s7: 8 registradores que devem ser preservados em uma chamada de procedimento.

E se quisermos manter o valor dos registradores utilizados?

- Guardar valor de registradores na pilha (exemplo para preservar $t0 e $t1);
- Utiliza-se o registrador $sp para indicar o topo da pilha;
- Pilha cresce de cima para baixo.

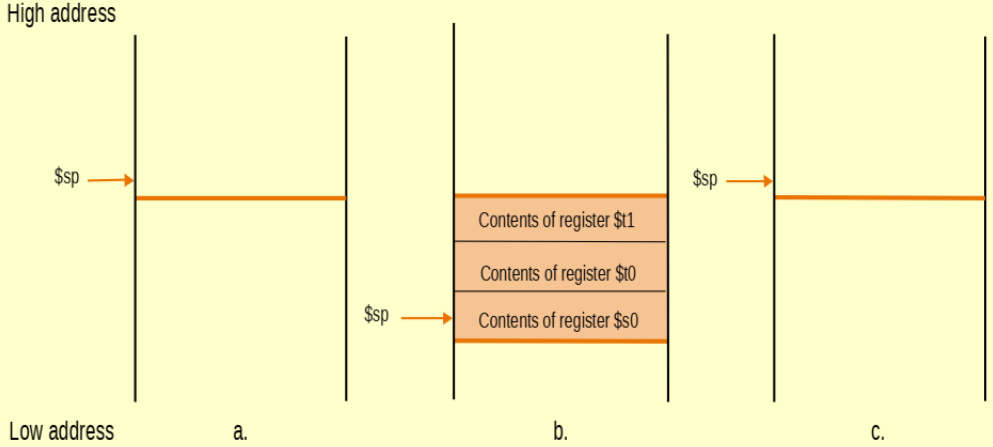


Figura 3.10 – Valores de sp antes, durante e depois da chamada do procedimento

Exemplo

- Seja o procedimento abaixo:
- Os parâmetros g, h, i e j correspondem a $a0 .. $a3, respectivamente e f a $v0. **Antes, precisaremos salvar $t0 e $t1 na pilha, pois serão usados no procedimento.**

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

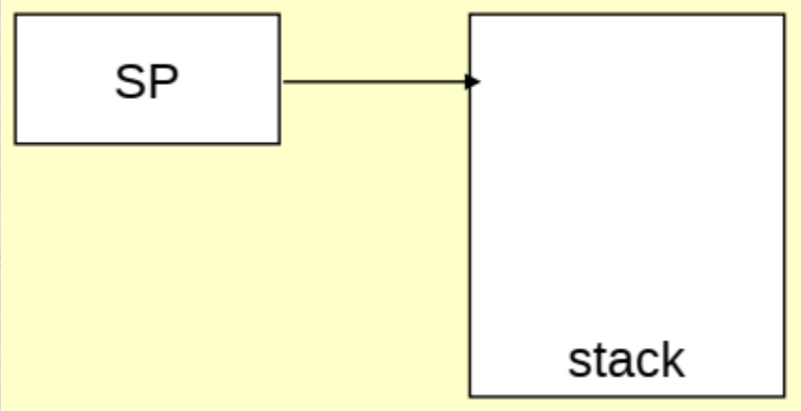
Exemplo

```
    addi $sp, $sp, -8    # ajuste do $sp para empilhar 2 palavras
    sw $t1, 4($sp)     # salva $t1 na pilha
    sw $t0, 0($sp)     # salva $t0 na pilha
    jal Proc           # desvia para o procedimento
    lw $t0, 0($sp)     # restaura $t0 da pilha
    lw $t1, 4($sp)     # restaura $t1 da pilha
    addi $sp, $sp, 8   # ajuste do $sp para desempilhar 2
palavras

Proc: add $t0, $a0, $a1
      add $t1, $a2, $a3
      sub $v0, $t0, $t1
      jr $ra           # retorna para chamada
```

```
int exemplo (int g, int h, int i, int j)
{
      int f;

      f = (g + h) – (i + j);
      return f;
}
```

Chamada de procedimentos recursiva ou aninhada

- Qual o problema para chamadas aninhadas?
  - $ra é destruído;
  - Pode perder outros registradores.
- Qual a solução? Utilizar uma pilha (LIFO)



- Registrador utilizado para o *stack pointer*: $sp ($29)

Exemplo

- Exemplo – procedimento recursivo

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

Exemplo

Supor n correspondente a $a0, e $t1 contém o valor 1.

```
fact:
    addi $sp,$sp,-8        # ajuste da pilha
    sw $ra,4($sp)          # salva o endereço de retorno
    sw $a0,0(sp)           #salva o argumento n
    slt $t0,$a0,1          #teste para n<1
    beq $t0,$zero,L1       #se n>=1, vá para L1
    addi $v0,$zero,1       #retorna 1 se n < 1
    addi $sp,$sp,8         #pop 2 itens da pilha
    jr $ra
L1:
    sub $a0,$a0, $t1       #n>=1, n-1
    jal fact #chamada com n-1
    lw $a0,0($sp)          #retorno do jal; restaura n
    lw $ra,4($sp)
    addi $sp,$sp,8
    mult $v0,$a0,$v0   #retorna n*fact(n-1)
    jr $ra
```

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

Alocação de espaço para novos dados

- *Procedure frame* ou *activation record*: Segmento de pilha que contém os registradores do procedimento salvos e as variáveis locais.
- O registrador $fp é usado para apontar para a primeira palavra deste segmento.

- **Figura 3.11 – O que é preservado ou não numa chamada de procedimento.**

Registradores Preservados	Registradores Não Preservados
Salvos: $s0-$s7	Temporários: $t0-$t9
Apontador para pilha: $sp	Argumentos: $a0-$a3
Endereço de retorno: $ra	Valores de Retorno: $v0-$v1
Pilha acima do Apontador para pilha	Pilha abaixo do Apontador para pilha

Alocação de espaço para novos dados

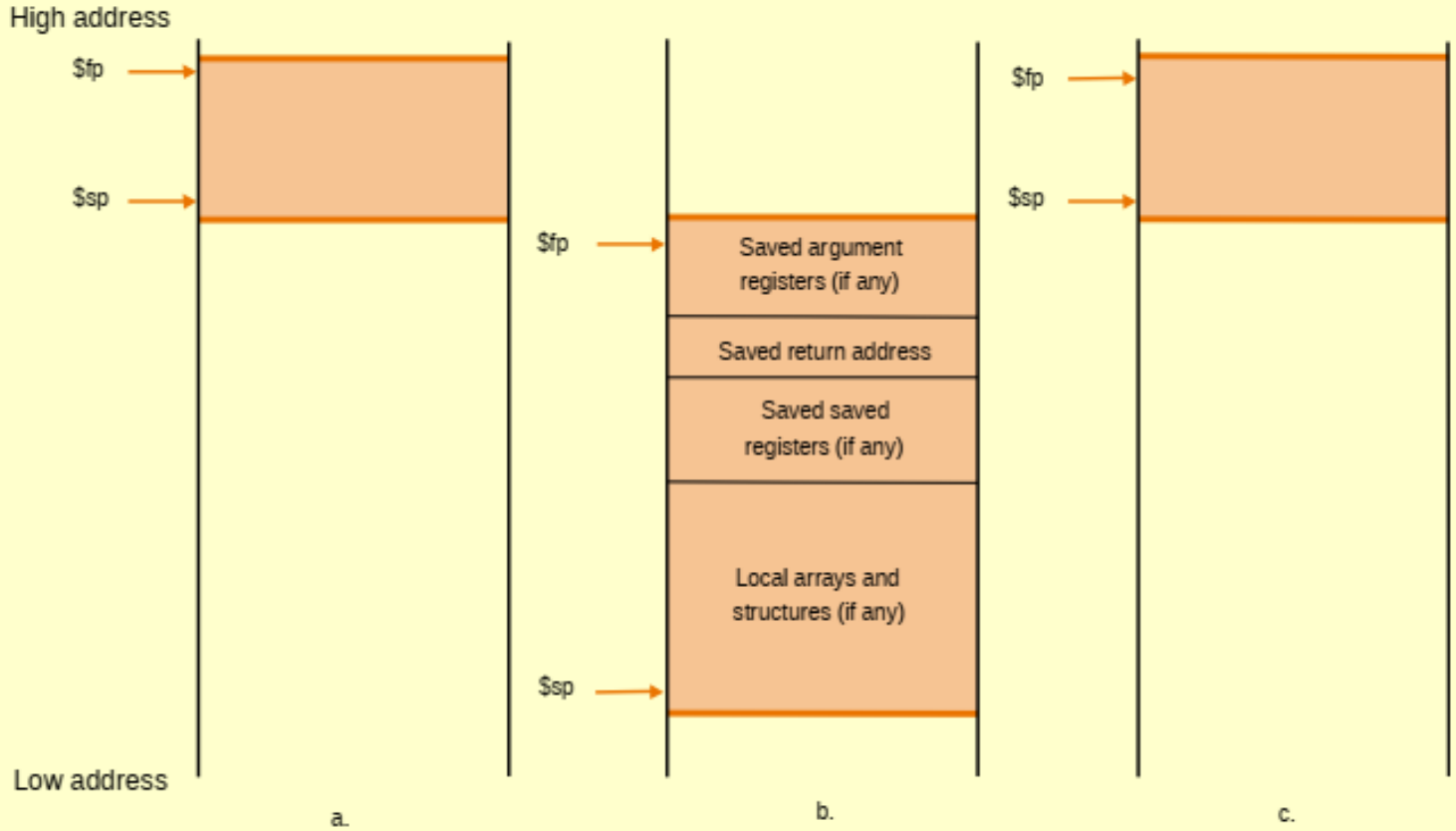


Figura 3.12 – Ilustração da pilha antes, durante e depois da chamada de procedimento.

Convenção de registradores no MIPS

Nome	Número	Uso	Preservado em chamadas?
$zero	0	Constante 0	n.d
$v0-$v1	2-3	Resultados e avaliações de expressões	Não
$a0-$a3	4-7	Argumentos	Não
$t0-$t7	8-15	Temporários	Não
$s0-$s7	16-23	Salvos	Sim
$t8-$t9	24-25	Temporários	Não
$gp	28	Ponteiro global	Sim
$sp	29	Ponteiro para pilha	Sim
$fp	30	Ponteiro para frame	Sim
$ra	31	Endereço de retorno	Sim

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 words differ by 4. Memory holds data structures, such as 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 operands; data in registers
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three operands; data in registers
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
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; for beq, bne
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

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)
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 3.8)
jr	R	0	31	0	0	0	8	jr $ra
jal	J	3	2500					jal 10000 (see section 3.8)
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

FIGURE 3.14 MIPS architecture revealed through section 3.6. Highlighted portions show MIPS assembly language structures introduced in section 3.6. The J-format, used for jump and jump-and-link instructions, is explained in section 3.8. This section also explains why putting 25 in the address field of beq and bne machine language instructions is equivalent to 100 in assembly language.

Endereçamento no MIPS

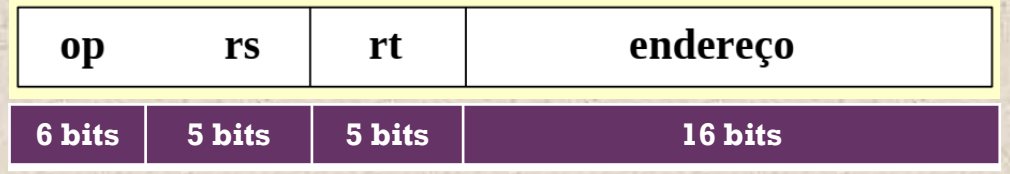
- Quais são os modos de endereçamento no MIPS?
  - Como utilizar constantes de mais de 16 bits para dados além de desvios?

- Quais são os modos de endereçamento
  - das instruções *branch*
  - das instruções *jump*
  - de outras instruções

Endereçamento no MIPS

- Exemplo de problema com imediatos de 16 bits:
  - Alterar $sp para endereço distante
- Alternativa:
  - Buscar constante na memória próxima:

```
lw $t0,end_constante($zero) #dado com 32 bits
add $sp,$sp,$t0
```

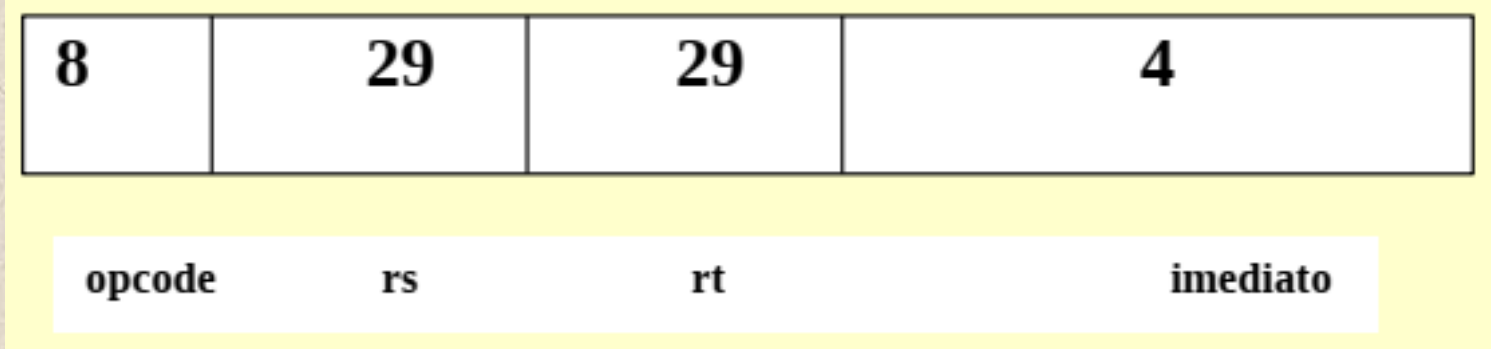


- Problema:
  - Acesso à memória é lento.
- Alternativa 2:
  - Permitir instruções aritméticas do tipo I e
  - Atribuir constantes a registrador por carga

Endereçamento no MIPS

- A instrução add do tipo I é chamada addi (add immediate). Para somar 4 a $sp temos:

addi $sp,$sp,4

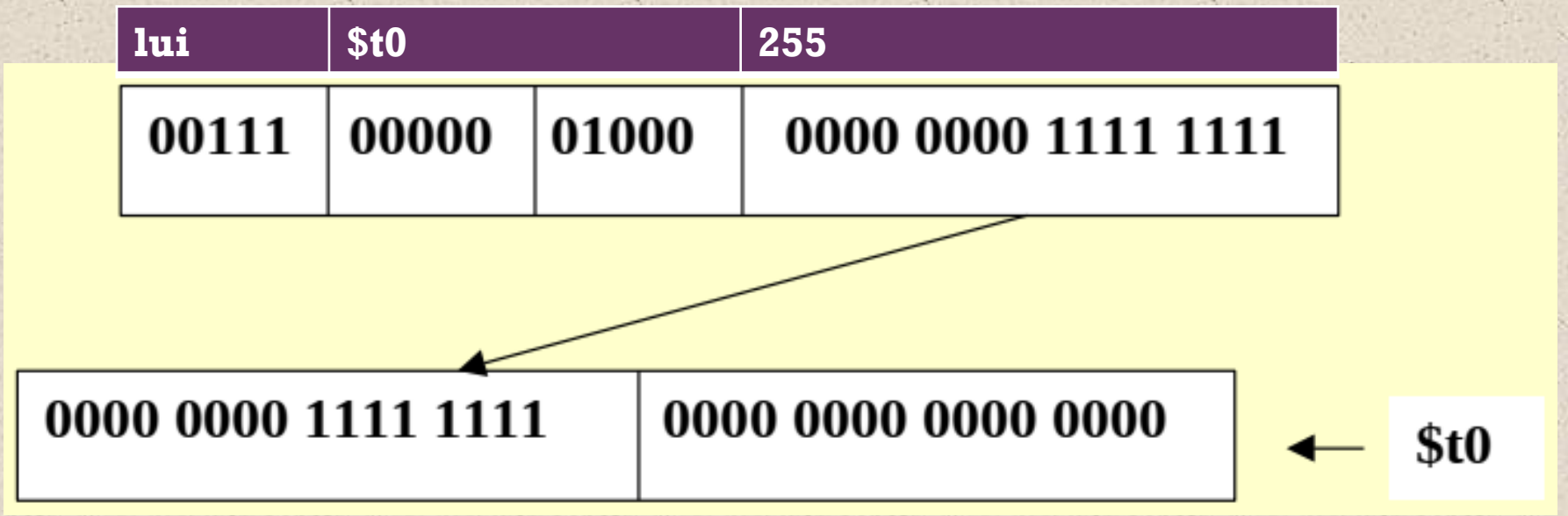


- Em comparações
  - slti $t0,$s2,10 # $t0 =1 se $s2 < 10

Instruções de carga

Os outros 16 bits são realizados por instrução especial de carga com uma instrução addi:

lui $t0,255 *#load upper immediate*



Exercício

Qual o código MIPS para carregar uma constante de 32 bits no registrador $s0?

0000 0000 0011 1101 0000 1001 0000 0000

Exercício

Qual o código MIPS para carregar uma constante de 32 bits no registrador $s0?

0000 0000 0011 1101 0000 1001 0000 0000

61_d 2304_d

Solução:

addi $s0,$zero,2304 #2304_d = 0000 1001 0000 0000$_b$

lui $s0,61 #$61_d$ = 0000 0000 0011 1101$_b$

Endereçamento no MIPS

- Atualizar $sp com imediato de 32 bits sem acessar a memória:

```
addi $t0,$zero,const_low

lui $t0,const_hi

add $sp, $sp, $t0
```

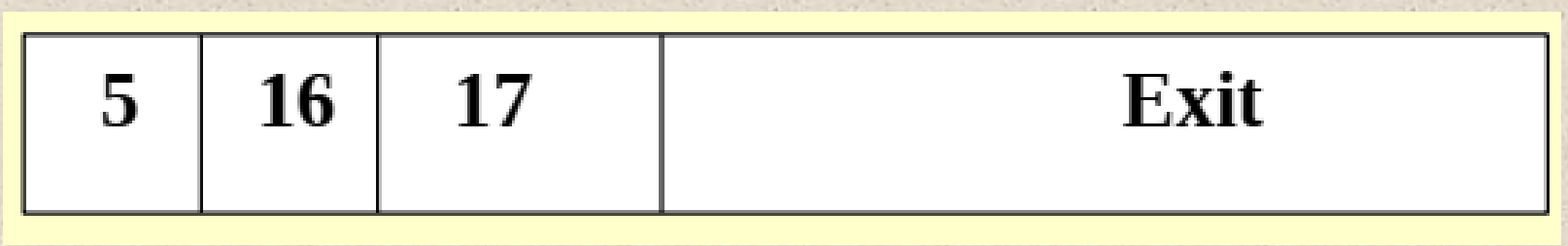
- Tanto instrução addi como lui possuem modo de endereçamento:
  - Registrador;
  - Imediato de 16 bits.

Endereçamento relativo ao PC

- Branch (*I-type*)

Exemplo

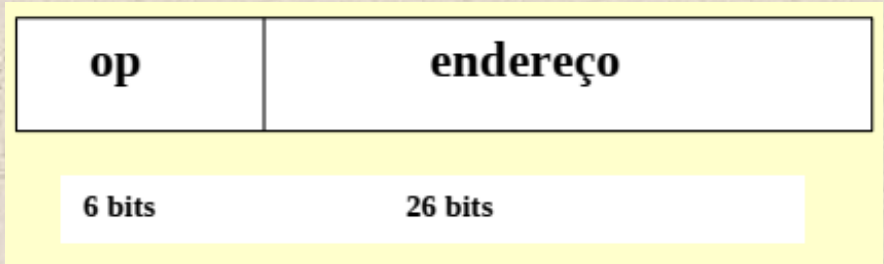
bne $s0,$s1,Exit



PC ← PC + Exit

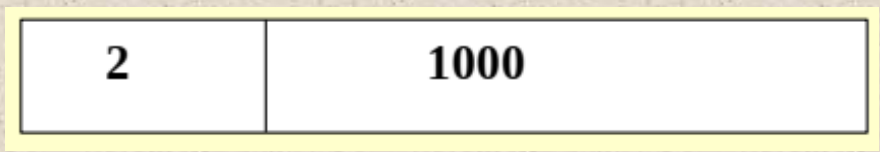
Endereçamento pseudo-direto

- Instruções *J-type* – 4 bits mais significativos vêm do PC



Exemplo

j 1000 # vá para 1000



Essa função será executada por:

- Deslocamento para a esquerda (para que o endereço seja multiplicado por 4);
- Concatena o PC nos 4 bits mais significativos.

Isso totaliza um endereço de 32 bits.

Exemplo

```
Loop:
      add  $t1,$s3,$s3      # $t1 = 2 * i
      add  $t1,$t1,$t1      # $t1 = 4 * i
      add  $t1,$t1,$s6      # $t1 = endereço de save[i]
      lw   $t0,0($t1)       # $t0 recebe save[i]
      bne  $t0,$s5,Exit     #vá para Exit se save[i] != k
      add  $s3,$s3,$s4      #i = i+j
      j    Loop
Exit:
```

Assumindo que o loop está alocado inicialmente na posição 80000 na memória, teremos a seguinte seqüência de código em linguagem de máquina:

80000	0	19	19	9	0	32
80004	0	9	9	9	0	32
80008	0	9	21	9	0	32
80012	35	9	8	0		
80016	5	8	21	8		
80020	0	19	20	19	0	32
80024	2	80000				
80028	.....					

Exemplo

- Dado o *branch* abaixo, reescrevê-lo de tal maneira a oferecer um *offset* maior (L1 está em um endereço distante)

```
beq $s0,$s1,L1
```

Exemplo

- Dado o *branch* abaixo, reescrevê-lo de tal maneira a oferecer um *offset* maior (L1 está em um endereço distante)

```
beq $s0,$s1,L1
```

Solução:

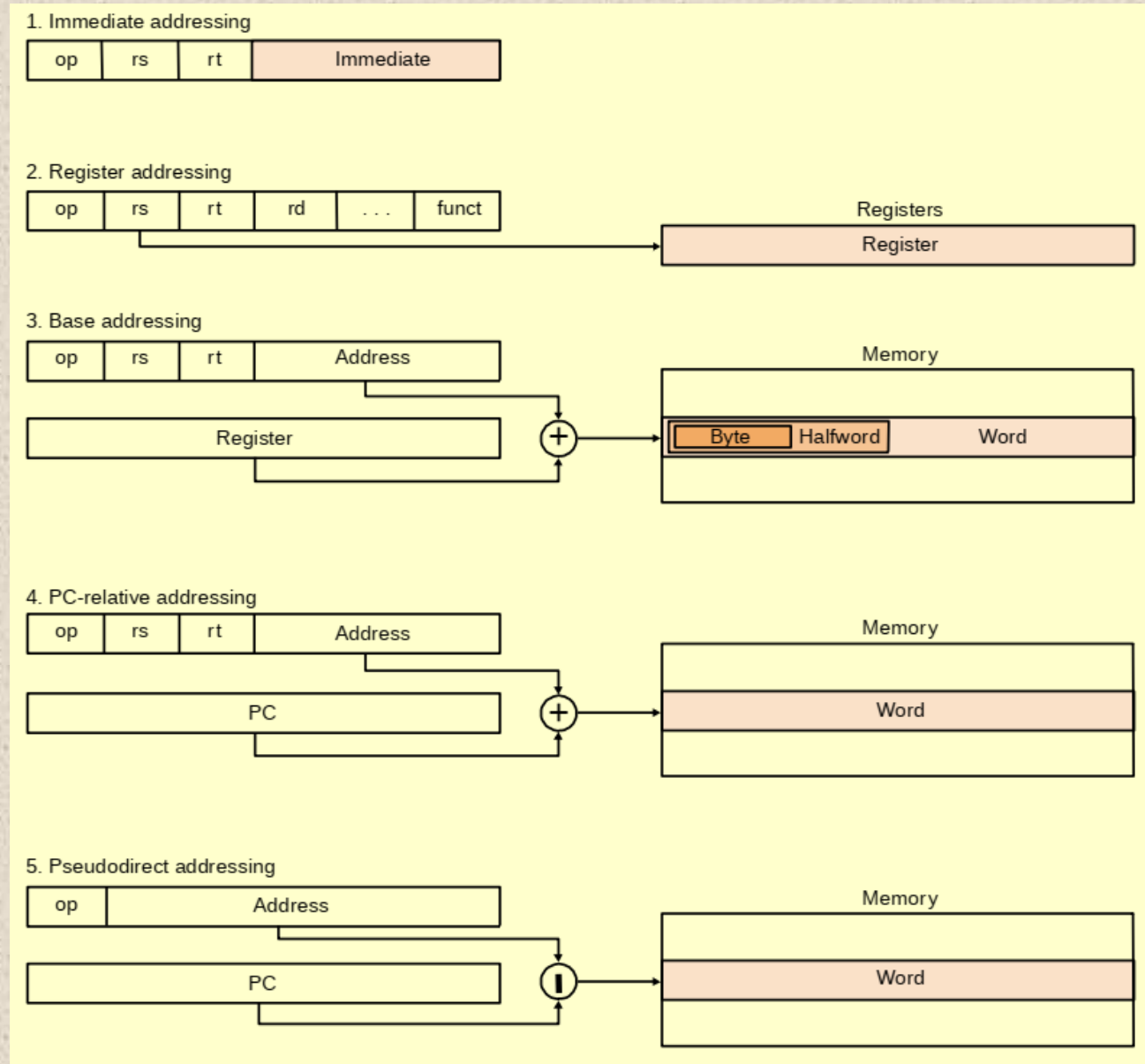
```
    bne $s0,$s1,L2
    j L1
L2:
```

Inverte a lógica para uso do j que permite endereçamento com 28 bits

Endereçamento MIPS - Resumo

- Endereçamento imediato: onde o operando é uma constante na própria instrução. (Ex. Addi)
- Endereçamento por registrador: o operando é um registrador. (Ex. add)
- Endereçamento por base ou deslocamento: o operando é uma localização de memória cujo endereço é a soma de um registrador e uma constante na instrução. (Ex. lw)
- Endereçamento relativo ao PC: onde o endereço é a soma de PC e uma constante da instrução. (Ex. beq)
- Endereçamento pseudodireto: onde o endereço de desvio (26 bits) é multiplicado por 4 e concatenado com os 4 bits mais significativos do PC. (Ex. j)

Endereçamento MIPS - Resumo



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

FIGURE 3.19 MIPS instruction formats in Chapter 3. Highlighted portions show instruction formats introduced in this section.

Linguagem assembly do MIPS

MIPS operands

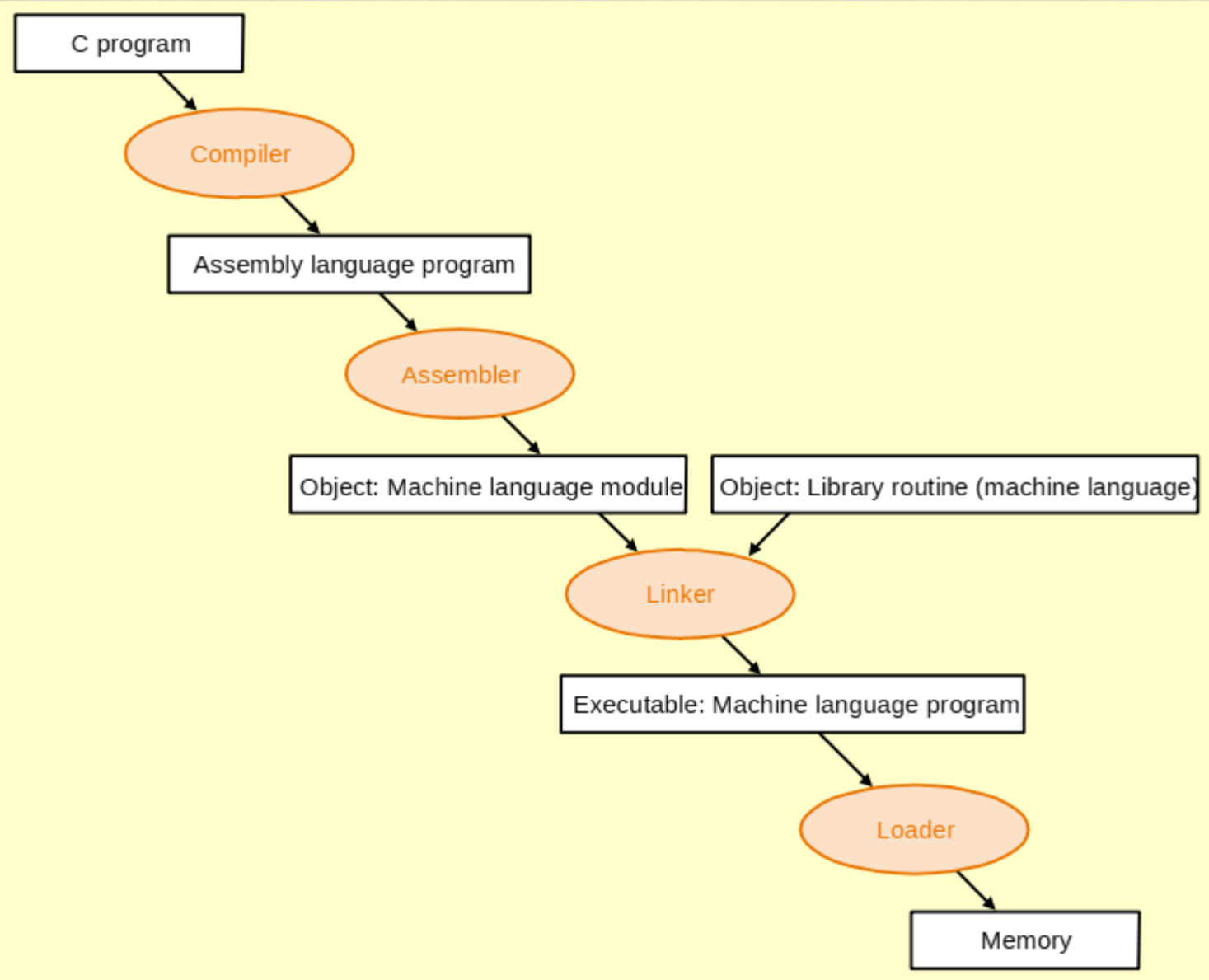
Name	Example	Comments
32 registers	$s0–$s7, $t0–$t9, $zero, $a0–$a3, $v0–$v1, $gp, $fp, $sp, $ra, $at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. Register $at is reserved for the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as 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 operands; data in registers
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; data in registers
	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 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 immediate	lui $s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
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
Uncondi-tional 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

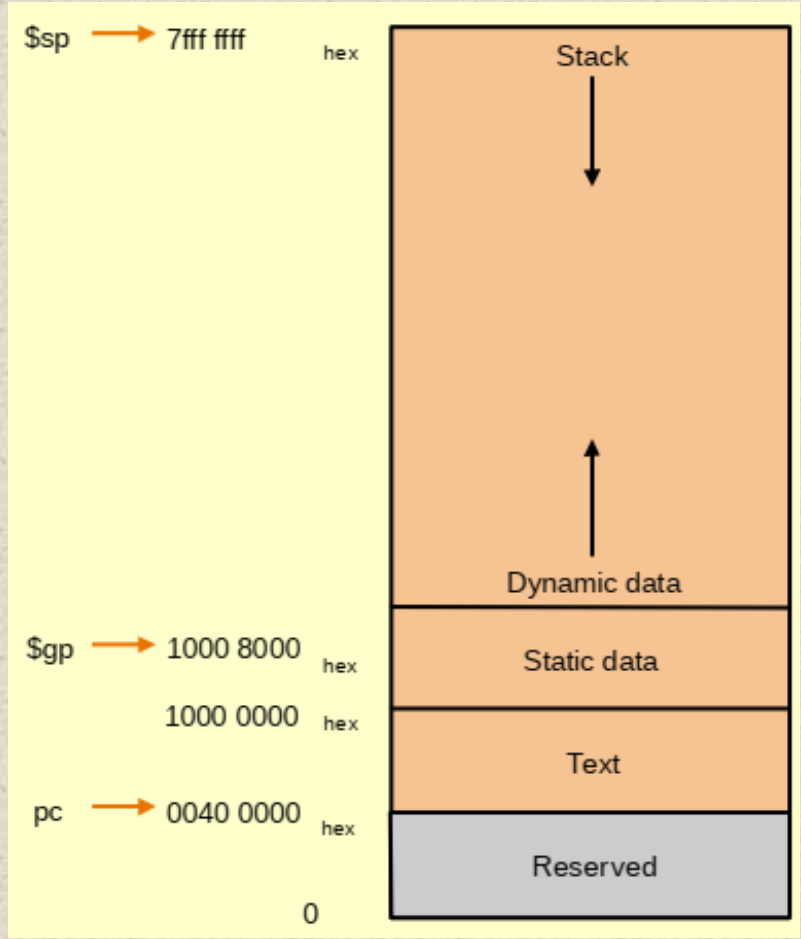
FIGURE 3.20 MIPS assembly language revealed in Chapter 3. Highlighted portions show portions from sections 3.7 and 3.8.

Traduzindo um programa



Registradores - Resumo

- Quando da tradução de C para assembly deve-se fazer:
  - alocar registradores para as variáveis do programa
  - produzir código para o corpo do procedimento
  - preservar os registradores durante a chamada do procedimento



Arquitetura MIPS

Conjunto de instruções MIPS

Organização e Projeto de Computadores: A Interface Hardware/Software.
David A. Patterson e John L. Hennessy

Agradeço a Prof. Dr. Fábio A. M. Cappabianco, Dave Patterson e Paulo Centoducatte pelos materiais disponibilizados.