Cap - 3: Linguagem de Máquina - MIPS

Arquitetura de Sistemas Computacionais
Prof. Ricardo Pannain

Arquitetura MIPS

- MIPS – Microprocessor without Interlocking Pipes Stages (Microprocessador sem Intertravamento entre os estágios de *pipe*)
- MIPS é uma marca registrada da MIPS Technology
- MIPS é inspirada na arquitetura RISC – Reduced Instruction Set Computer (Computador com conjunto de instruções reduzidas)

Origens da Arquitetura MIPS

- 1980 David A. Patterson e Carlo Séquin (Universidade da California, Berkely), começam a projetar pastilhas RISC VLSI (RISC I e RISC II)
- 1981 John L. Hennessy (Universidade Stanford,São Francisco) projetou e fabricou uma RISC um pouco diferente, que ele chamou de $MIPS_3$

Produtos com Arquitetura MIPS

- Pastilhas MIPS — Formação da MIPS Computer Systems, que fabrica pastilhas de CPU utilizadas nas máquinas RISC vendidas pela DEC e por outros fabricantes de computadores.
- Pastilhas RISC I e RISC II — Inspiração do projeto SPARC da Sun Microsystems

4

Instruções no MIPS

- No MIPS, instrução é uma palavra da linguagem de máquina.
- Vocabulário é o conjunto de instruções (*instruction set*)
- *Instruction Set* do MIPS (usado pela NEC, Nintendo, Silicon Graphics e Sony)

5

Operações Aritméticas no MIPS

- O MIPS trabalha com 3 operandos

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

- **Princípio de Projeto 1**:

  A simplicidade é favorecida pela regularidade.

6

Operações Aritméticas no MIPS

- Compilação de uma declaração C complexa

Programa em C	Assembly MIPS

f = (b + c) – (i + j);	add t0,g,h
	add t1,i,j
	sub f,t0,t1

- O compilador cria as variáveis temporárias **t0** e **t1**

7

Operandos no MIPS

- No MIPS são 32 registradores de 32 bits ($0 ... $31)
  (não há suporte em hardware para o conceito de variável)

Programa em C	**Assembly MIPS**

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

- **Princípio de Projeto 2**:

  Quanto menor, mais rápido ® manter o número de registradores tão pequeno quanto possível.

8

Instruções para movimentação de dados

- **lw** ® movimenta dados da memória para registrador (load word)
- sw ® movimenta dados do registrador para memória (store word)
- Formatos:

lw <registrador> <deslocamento>(<registrador>)
sw <registrador> <deslocamento>(<registrador>)

9

Atribuição com operando na memória

Exemplo 1:

Seja A um *array* com 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: **g = h + A[8];**

Solução:

Primeiro devemos carregar um registrador temporário com A[8]:
lw $t0,32($s3) # $t0 recebe A[8]

Agora basta executar a operação:
add $s1,$s2,$t0 # g recebe h + A[8]

10

Continuação...

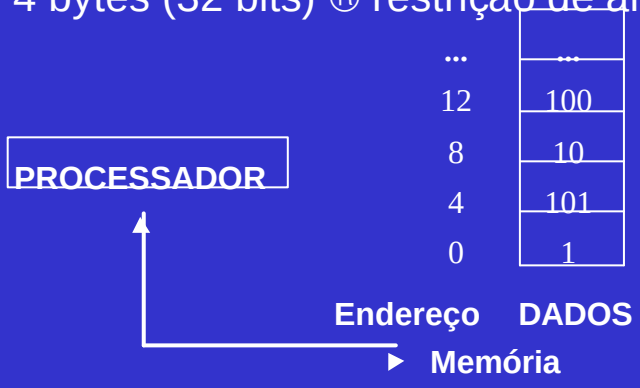
Notas:

- ✓ Devido à limitação quanto à quantidade de registradores, estruturas de dados tais como, arrays são mantidos em memória.
- ✓ A constante que aparece na instrução **lw** é o *deslocamento*, e o registrador cujo valor armazenado é somado a esta constante é chamado de *registrador-base*.

11

Endereçamento no MIPS

- No MIPS a memória é organizada em bytes, embora endereçamento seja em palavras de 4 bytes (32 bits) ® restrição de alinhamento.



12

Continuação...

As máquinas que endereçam *bytes* podem dividir-se em duas categorias: aquelas que usam o byte mais à esquerda (*big endian*) e aquelas que o byte mais à direita (*little endian*). O MIPS está na categoria das máquinas *big endians*.

13

Atribuição com operando na memória

Exemplo **2**:

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 C: **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)   #  A[12] = h + A[8]
```

14

Atribuição com operando na memória

Exemplo 3:

Supor que o índice seja uma variável: g = h + A[i];

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

Solução:

```
add $t1,$s4,$s4   #  $t1 recebe 2 * i
add $t1,$t1,$t1   #  $t1 recebe 4 * i
add $t1,$t1,$s3   #  $t1 recebe o end. de A[i]
lw    $t0,0($t1)  #  $t0 recebe A[i]
add $s1,$s2,$t0   #  g recebe h + A[i]
```

15

Representação de instruções no COMPUTADOR

Registradores $s0..$s7 ® 16..23
Registradores $t0..$t7 ® 8..15

Exemplo:

Formato da instrução:

add $t1, $s1, $s2 # $t1 recebe $s1 + $s2

16

Continuação...

Representação da Instrução em Decimal

17

Formato das instruções e seus campos

R-type

6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
op	rs	rt	rd	shamt	funct

Onde:

op ® operação básica da instrução (opcode)
rs ® o primeiro registrador fonte
rt ® o segundo registrador fonte
rd ® o registrador destino
shamt ® para instruções de deslocamento (shift amount)
funct ® seleciona variações da operação especificada pelo opcode (function)

18

Continuação...

I-type

6 bits	5 bits	5 bits	16 bits
op	rs	rt	endereço

Princípio de Projeto 3:

Um bom projeto demanda compromisso. â compromisso escolhido pelos projetistas do MIPS foi manter todas as instruções do mesmo tamanho.

Exemplo de instrução I-type: **lw $st0,32($s3)**

19

Representação de instruções no COMPUTADOR

Inst.	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.**

20

Continuação...

Exemplo:

Dê 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      # st0 recebe h + A[300]
sw   $t0,1200($t1)    # A[300] recebe h + A[300]
```

21

Continuação...

Linguagem de Máquina

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

22

Instruções para tomada de decisões

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)

23

Desvio Condicional - Comando if

Exemplo: Compilando um comando if.

Seja o comando abaixo:

```
      if (i == j) goto L1;
      f = g + h;
L1:   f = f – i;
```

Supondo que as 5 variáveis correspondam aos registradores **$s0..$s4**, respectivamente.

Como fica o código MIPS para o comando?

24

Continuação...

Linguagem C

```
      if (i == j) goto L1;
      f = g + h;
L1:   f = f – i;
```

Solução: Assembly MIPS

```
      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
```

25

Instrução de Desvio Incondicional

j L1 â quando executado faz com que o programa seja desviado para L1

Exemplo: Compilando um comando if-then-else

Seja o comando: **if (i == j) f = g + h; else f = g - h;**

Solução: Assembly MIPS

```
        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:
```

26

Loops (laços)

® Usando if

Exemplo: Loop: g = g + A[i];
i = i + j;
if (i != h) goto Loop;

Solução: Loop: add $t1,$s3,$s3 # st1 = 2 * i
add $t1,$t1,$t1 # $t1 = 4 * i0
add $t1,$t1,$s5 # $t1 recebe end. 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

27

Loops (laços)

® Usando while

Exemplo: while (save[i] == k) i = i + j;

Solução: Para **i**, **j** e **k** correspondendo a **$s3**, **$s4** e **$s5**, respectivamente, e o endereço base do array em **$s6**, temos:

```
Loop: add $t1,$s3,$s3    # st1 = 2 * i
      add $t1,$t1,$t1    # $t1 = 4 * i0
      add $t1,$t1,$s6    # $t1 recebe 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:
```

28

Instrução para teste de maior ou menor

slt registrador temporário, registrador1, registrador2

® se registrador1 é menor que registrador2, então o registrador temporário é setado, caso contrário é resetado.

Obs.: Para utilizações específicas, os compiladores MIPS associam o registrador $0 ao valor zero ($zero).

Exemplo: Compilando o teste **less than**

```
slt   $t0,$s0,$s1        # $t0 é setado se $s0 < $s1
bne $t0,$zero,Less       # vá para Less, se $t0 != 0, ou seja a < b
```

29

Comando Seletivo

Exemplo: Compilando o comando **switch**

Seja o comando abaixo:

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

Suponha que as seis variáveis correspondam aos registradores de $s0 a $s5, e que o registrador $t2 contenha o valor 2.

Qual é o código correspondente na linguagem de montagem do MIPS?

30

Continuação...

Solução:

```
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
bne $t3,$zero,Exit     # se k >= 2 vá para Exit

add $t1,$s5,$s5        # $t1 = 2 * k
add $t1,$t1,$t1        # $t1 = 4 * k
```

31

Continuação...

```
# assumindo que 4 palavras na memória, começando no
   endereço contido em $t4, tem endereçamento
   correspondente a L0, L1 e L2

      add $t1,$t1,$t4     # $t1 = end. de tabela[k]
      lw   $t0,0($t1)      # $t0 = tabela[k]
      jr     $t0              # salto para end. carregado em $t0
L0:  add $s0,$s3,$s4   # k = 0 portanto f = i + j
      j      Exit
L1:  add $s0,$s1,$s1   # k = 1 portanto f = g + h
Exit:
```

32

Suporte a Procedimento

® **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.**

33

Continuação...

® **Para este mecanismo, o MIPS aloca seus registradores, para chamada de procedimentos, da seguinte maneira:**

- ✓ **$a0..$a3** â 4 registradores para passagem de argumentos
- ✓ **$v0..$v1** â 2 registradores para retornar valores
- ✓ **$ra** â para guardar o endereço de retorno

® **Instrução para chamada de Procedimento**

jal End_Proc (**jump-and-link**) â desvia para o procedimento e salva o endereço de retorno (PC+4) em $ra (return address - $31)

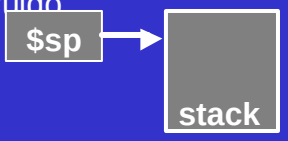
34

Continuação...

- **Instrução para chamada de Procedimento**

  jr $ra â desvia para o ponto de onde foi chamado o procedimento.

- **Qual o problema para chamadas aninhadas?**
  R. O registrador $ra é destruído.



- **Qual a solução?**
  R. Utilizar uma PILHA.

- **Registrador utilizado pelo stack pointer â $sp($29)**

35

Continuação...

Exemplo:

Seja o procedimento abaixo:

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

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

36

Continuação...

Solução:

Os parâmetros **g**, **h**, **i** e **j** correspondem a **$a0..$a3**, respectivamente e **f** a **$s0**.

Antes precisaremos salvar **$s0**, **$t0** e **$t1** na pilha, pois serão usados no procedimento.

```
sub $sp,$sp,12   # ajuste do sp para empilhar 3 palavras
sw   $t1,8($sp)   # salva $s1 na pilha
sw   $t0,4($sp)   # salva $t0 na pilha
sw   $s0,0($sp)   # salva $s0 na pilha
```

37

Continuação...

No procedimento

```
add $t0,$a0,$a1
add $t1,$a2,$a3
sub $s0,$t0,$t1
```

Para retornar o valor f

```
add $v0,$s0,$zero
```

Antes do retorno é necessário restaurar os valores dos regi-tradores salvos na pilha.

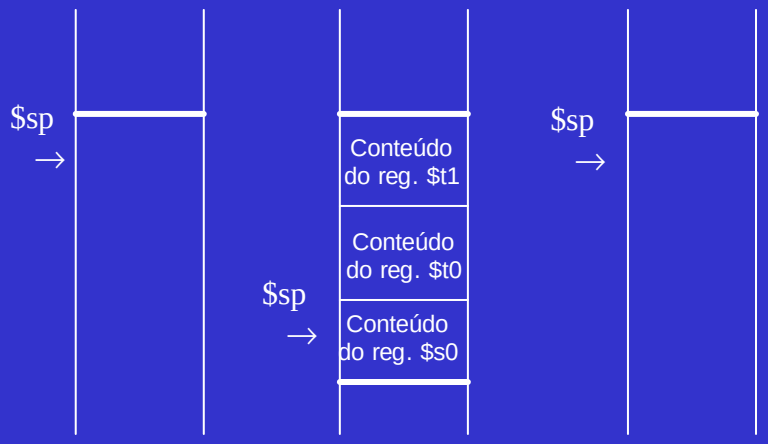
```
lw   $s0,0($sp)
lw   $t0,4($sp)
lw   $s1,8($sp)
add $sp,$sp,12
```

Retornar

```
jr $ra
```

38

Continuação...



Valores de $sp antes, durante e depois da chamada do procedimento

39

Continuação...

® Observações:

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

® Exemplo: **Procedimento Recursivo**

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

40

Continuação...

Solução:

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

41

Continuação...

- **Alocação de espaço para novos dados:**

O segmento de pilha que contém os registradores do procedimento salvos e as variáveis locais é chamado de *procedure frame* ou *activcation record*. O registrador **$fp** é usado para apontar para a primeira palavras deste segmento.

O que é preservado ou não numa chamada de procedimento

Registradores Preservados	Registradores não Preservados
Salvos: $s0-$s7	Temporários: $t0-$t7
Apontador para pilha: $sp	Argumentos: $a3
Endereço de retorno: $ra	Valores de retorno: $v0-$v1
Pilha acima do Apontador para pilha	Pilha abaixo do apontador para Pilha

42

Continuação...

Memória alta

Memória baixa

Pilha antes, durante e depois da chamada de procedimento

43

Convenção dos Registradores no MIPS

Nome	Número	Uso	Preservados em chamadas?
$zero	0	Constante 0	n.d.
$v0-$v1	2-3	Resultados e avaliações de expressões	Não
$a0-$a3	4-7	Argumentos	Sim
$t0-$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 par Pilha	Sim
$fp	30	Ponteiro para Frame	Sim
$ra	31	Endereço de Retorno	Sim

44

Outros Estilos de Endereçamento no MIPS

® Os projetistas do MIPS desenvolveram mais duas vias de acesso a operandos.
A primeira delas tem por objetivo tornar mais rápido o acesso a constantes pequenas, e a segunda visa tornar os desvios mais eficientes.

® O endereço imediato ilustra o último dos quatro princípios básicos do projeto do hardware.

Princípio de Projeto 4:

Torne o caso comum mais rápido.

45

Continuação...

® **Operandos Imediatos ou Constantes:**

Motivação:

- É muito comum que programas usem constantes em diversas operações que ocorrem com muita freqüência, tais como, o incremento de um índice para fazê-lo apontar para o próximo elemento de um *array*, a contagem de iterações de um laço, ou o ajuste do *stack pointer* em chamadas a procedimentos aninhados.

- Para utilizar uma constante qualquer, e com base somente nas instruções abordadas anteriormente, é necessário uma busca na memória, para carregá-la em um registrador, antes que se possa utilizá-la. Considere o exemplo a seguir.

46

Continuação...

Exemplo:

Para somar a constante **4** ao conteúdo do registrador **$sp**, poderíamos usar o seguinte código:

```
lw   $t0,addrConstant4($zero)  # $t0 recebe a constante 4
add $sp,$sp,$t0                # $sp recebe $sp + 4 (em $t0)
```

Uma alternativa, que evita acessos à memória, é oferecer, no conjunto de instruções, versões de instruções aritméticas nas quais um dos operandos é uma constante, mantida dentro da própria instrução. Considerando o exemplo acima, temos:

```
addi $sp,$sp,4
```

47

Continuação...

Exemplo: instrução addi (add immediate)

Na instrução add do tipo **I**, chamada **addi** (add immediate), um dos seus operandos é uma constante (ou um imediato).

Para somar **4** a **$sp** temos: **addi $sp,$sp,4**

Código de máquina para addi (em decimal)

opcode	**rs**	**rt**	**imediato**
6 bits	5 bits	5 bits	16 bits

48

Continuação...

Operandos imediatos em comparações:

- Os operandos imediatos também são muito usados em comparações. Como o registrador **$zero** sempre contém a constante 0, é fácil comparar um determinado valor com 0. Para comparar com **outros valores que não 0**, existe uma versão imediata da instrução *set on less than*. Por exemplo:

  slti $t0,$s2,10 # $t0 recebe 1 se $s2 < 10

Instrução lui – *load upper immediate*:

- A instrução **lui** carrega os 16 bits de **mais alta ordem** de uma constante em um registrador, permitindo que a instrução subseqüente especifique os 16 bits de mais baixa ordem desta constante.

49

Continuação...

Exemplo 1: **carga de uma constante de 16 bits**

lui $t0,255 # carrega 255 nos 16 bits de mais alta ordem de $t0

Código de Máquina da instrução **lui** (**I-type**)

6 bits | 5 bits | 5 bits | 16 bits - imediato

16 bits mais significativos | 16 bits menos significativos

50

Continuação...

Exemplo 2: **carga de uma constante de 32 bits**

Qual o código MIPS para carregar uma cte. de **32 bits** em **$s0**?

Solução:

Considere a constante: **0000 0000 0011 1101 0000 1001 0000 0000**, o primeiro passo é carregar os 16 bits de mais ordem em **$s0**.

lui **$**s0,61 # 61 dec. = 0000 0000 0011 1101 binário.

O próximo passo é somar ao conteúdo de **$s0** o valor **2.304**.

addi $s0,$s0,2304 # 2304 dec. = 0000 1001 0000 0000 binário.

51

Continuação...

® **Endereçamento nos desvios incondicionais (jumps) e condicionais (branchs):**

- Considere a instrução de desvio incondicional abaixo.

j 10000 # desvia para o endereço 10.000

A respectiva instrução (**J-type**) em **código de máquina** é:

6 bits **26 bits**

52

Continuação...

- Considere a instrução de desvio condicional abaixo.

bne $s0,$s1,Exit # desvia para Exit se $s0 ≠ $s1

A respectiva instrução (I-type) em **código de máquina** é:

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

O fato de a instrução deixar apenas 16 bits para o endereço-alvo do desvio, cria um pequeno problema: nenhum programa pode ter mais do que $\mathbf{2^{16}}$ **bytes**, pois aquilo que excede a esse valor não poderá ser alcançado pelos desvios.

53

Continuação...

- De modo a aumentar o domínio da variação dos endereços- alvo de desvio condicional, uma possível solução de contorno seria utilizar um registrador (**no caso o PC**) e somar seu conteúdo ao campo de 16 bits correspondente a instrução de desvio condicional. Desse modo, numa instrução de desvio condicional o montador deve calcular:

PC ¬ PC + Campo do Endereço, onde o Campo do Endereço é em unidades de palavras (4 bytes).

Esta forma de implementar o endereço de desvio em uma instrução de desvio condicional é chamada de **endereçamento relativo ao PC**.

54

Continuação...

Exemplo: endereçamento relativo ao PC em um loop *while*.

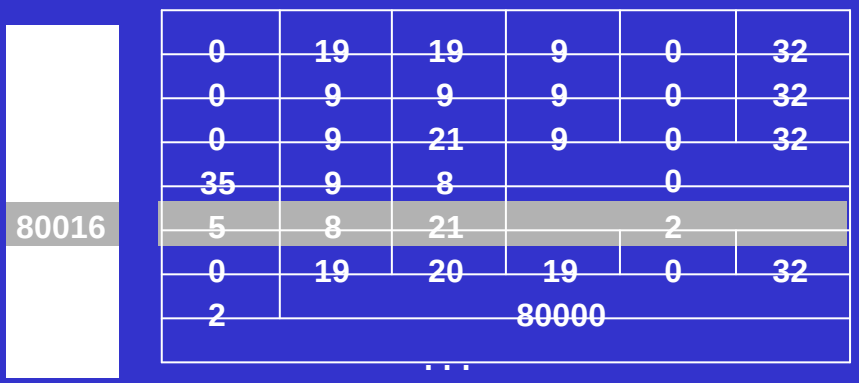
```
Loop: add $t1,$s3,$s3   # $t1 recebe 2 * i
      add $t1,$t1,$t1    # $t1 recebe 4 * i
      add $t1,$t1,$s5    # $t1 recebe endereço de save[i]
      lw  $t0,0($t1)     # $t0 recebe save[i]
      bne $t0,$s5,Exit   # vá para Exit se save[i] ≠ k
      add $s3,$s3,$s4    # i recebe i + j
      j 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

Se admitirmos que o *label Loop* foi armazenado no endereço 80.000 da memória, qual o código de máquina para esse laço *while*?

55

Continuação...

Solução: Código de Máquina para o loop *while*.



Nota: A instrução **bne** na quinta linha soma 2 words ao endereço da próxima instrução.

56

Continuação...

® **Desviando para mais longe**

Exemplo:

Considere uma instrução de desvio condicional que desvie para um label se o conteúdo de **$s0** for igual a **$s1**, como segue:

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

Pode-se substituir esta construção por um par de instruções de forma a possibilitar desviar para muito mais longe, como:

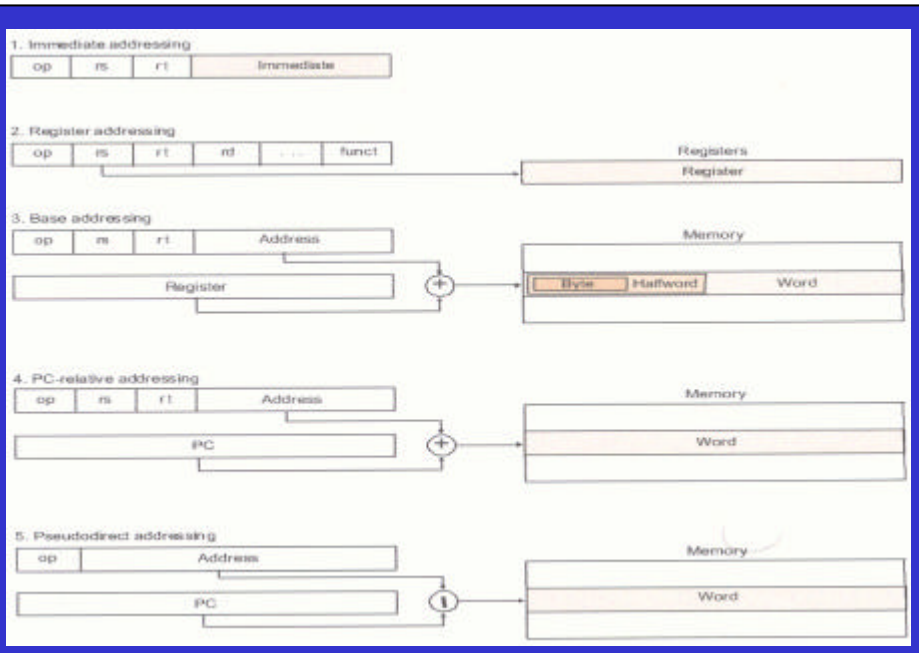
```
    bne $s0,$s1,L2
    j   L1
L2:
```

57

Resumo dos Modos de Endereçamento no MIPS

- Endereçamento por registrador : o operando é um registrador.
- 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.
- Endereçamento imediato : onde o operando é uma constante na própria instrução.
- Endereçamento relativo ao PC : onde o endereço é a soma de PC e uma constante da instrução.
- Endereçamento pseudodireto : onde o endereço de desvio (26 bits) é concatenado com os 4 bits mais significativos do PC.

58



59

Traduzindo um Programa

Etapas:

1) Programa C **=> COMPILADOR =>** Programa Assembly

2) Programa Assembly **=> ASSEMBLER =>** Módulo em LM

3) Módulo em LM+Library em L.M. **=> LINKER =>** Executável

4) Executável **=> LOADER =>** Programa em Memória**.**

FIM.

60