LABORATÓRIO DE SISTEMAS DIGITAIS

A Instrução SLT

SLT significa **Set on less Than**, (comparar se é menor que e setar o registrador destino). Essa instrução será muito utilizada em comparações entre registradores, para identificar quem tem o maior ou menor valor. A função desta instrução é comparar dois valores de dois registradores diferentes e atribuir o valor 1 a um terceiro registrador se o valor do primeiro registrador for menor que o valor do segundo registrador. Caso contrário, atribuir zero. A sintaxe é:

OpCode	RS	RT	RD	SHAMT	FUNCT
Código da Operação	Registrador Temporário	Registrador a ser comparado 2	Registrador a ser comparado 1	não usado	código da operação aritmética
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

SLT registrador_temporário, registrador1, registrador2

O formato da instrução é:

Vamos supor a seguinte instrução MIPS:

SLT $t0, $s1, $s2

Isso é o mesmo que:

$st0 = $s1 < $s2

O registrador temporário $t1 armazena o resultado da avaliação da expressão $s1 < $s2. Se for verdade que $s1 é menor que $s2, então, é atribuído ao registrador temporário o valor 1. Agora se o resultado da comparação entre os dois registradores for FALSO, ou seja, $s1 NÃO é menor que $s2, então, é atribuído ao registrador temporário o valor 0. O registrador temporário será utilizado por outra instrução, para realizar outra tarefa.

Exemplo

Considere o seguinte código em C:

```
1 if ( i < j )
2      a = b + c;
3 else
4      a = b – c;
```

Vamos fazer a compilação desse trecho de código em C para MIPS:

a. Linguagem de Montagem;
b. Linguagem de Máquina;
c. Representação e;
d. Código de Máquina.

Considere a = $s0, b = $s1, c = $s2, i = $s3, j = $s4.

a) Linguagem de Montagem

linha	código
1	slt $t0, $s3, $s4
2	bne $t0, $zero, ELSE
3	add $s0, $s1, $s2 *#a = b + c; (se $t0 <> 0)*
4	j Exit *#desvia para exit*
5	ELSE: sub $s0, $s3, $s4 *#a = b – c; (se $t0 = 0)*
6	Exit:

Primeiro é feita a comparação entre os registradores $s3 e $s4.

Se a resposta for verdadeira, o fluxo segue pelo lado direito do diagrama, em que o registrador $t0 receberá o valor 1.

Se a resposta for falsa, então o fluxo segue pelo lado esquerdo do diagrama, em que o registrador $t0 receberá o valor 0.

A partir daí, o fluxograma segue para uma próxima avaliação, que será a instrução BNE (branch not equal - desvie se não for igual).

Se o conteúdo do registrador $t0 for diferente de zero, que aqui é representado pelo registrador $zero (o qual contém o valor zero SEMPRE), então a resposta é verdadeira, e o fluxo do programa segue pelo lado direito.

Caso contrário, a resposta é falsa e o fluxo segue pelo lado esquerdo.

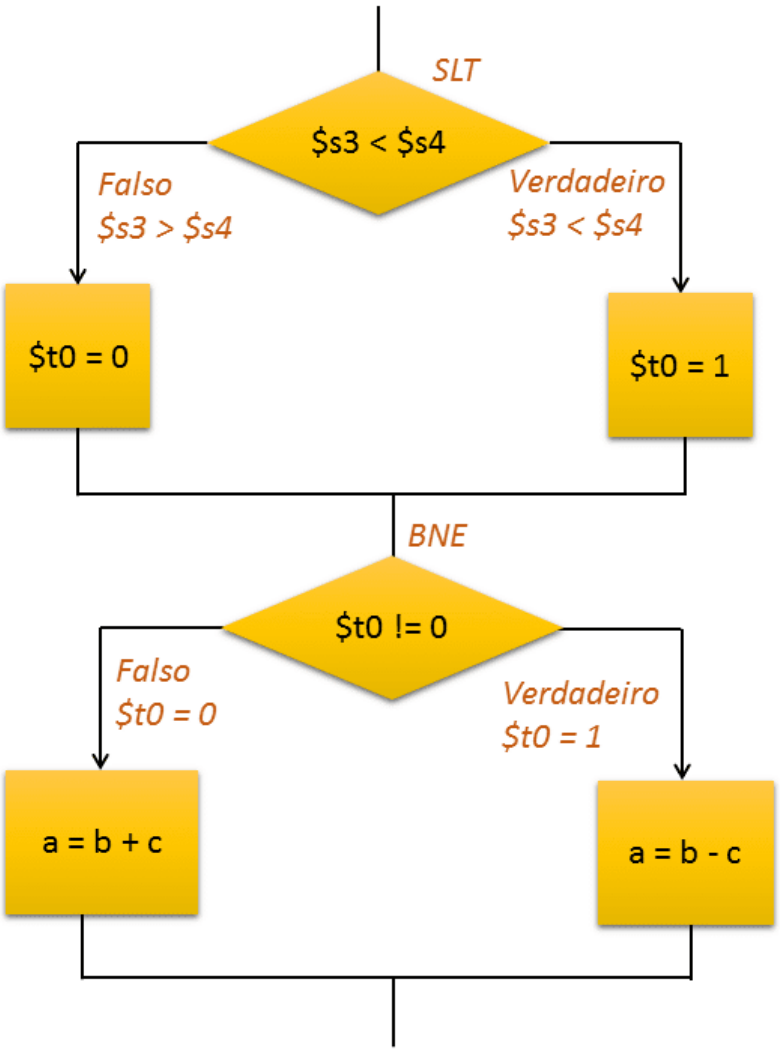


Figura 1: Fluxograma de SLT com BNE.

Vamos agora supor alguns valores para esses registradores, pra ficar mais claro o seu uso. Observe a Figura 2. Suponha que $s3 = 50 e $s4 = 100, assim 50 é menor que 100? Sim! Então a resposta é verdadeira. O registrador $t0 receberá o valor 1. Em seguida, BNE fará a comparação do registrador $t0 com o registrador $zero, como $t0 = 1, então t0 não é igual a zero, seguindo pelo lado direito do fluxograma.

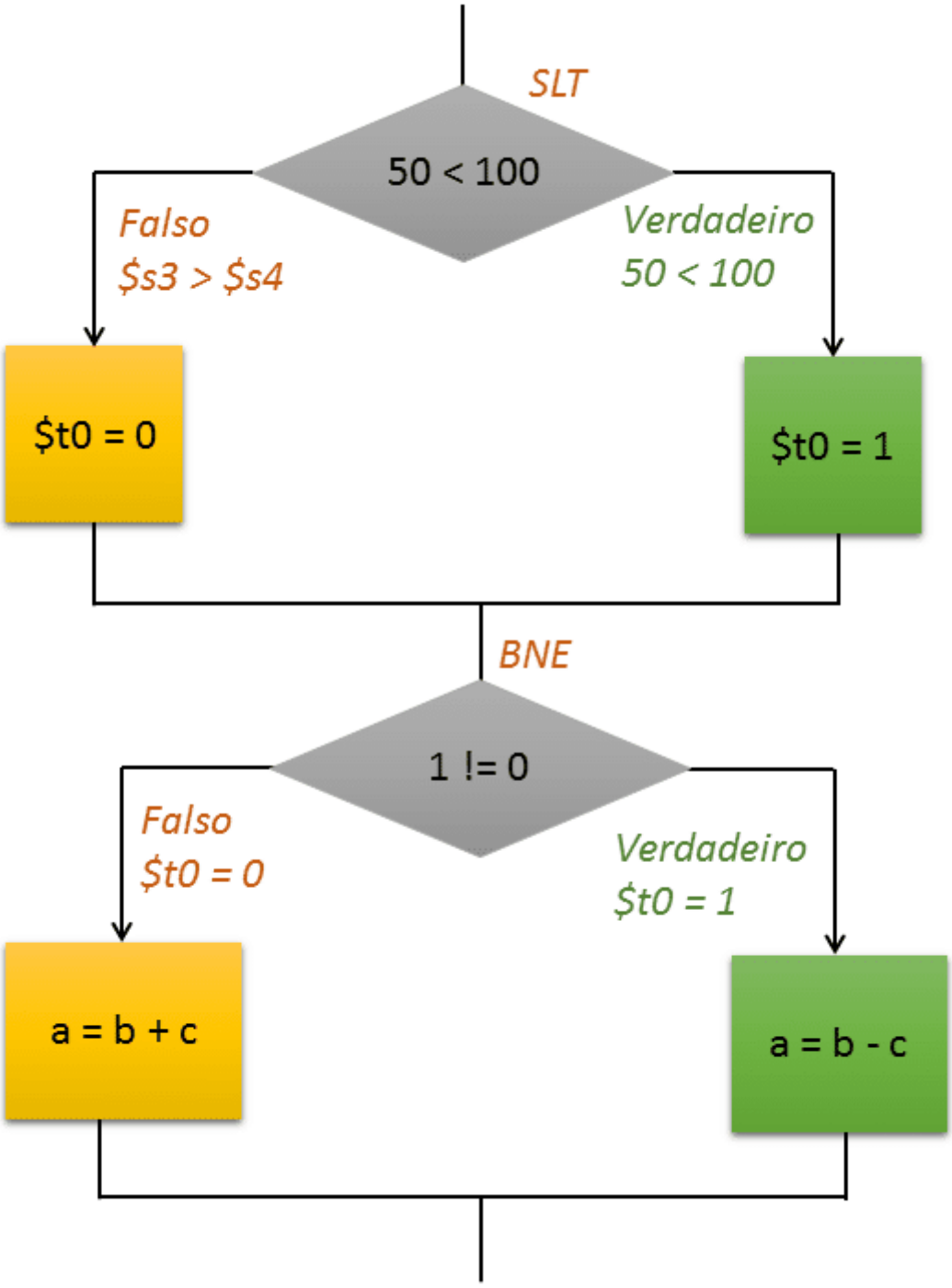


Figura 2: Representando o Fluxograma quando a resposta é verdadeira.

Observe a Figura 3. Suponha que $s3 = 200 e $s4 = 60, assim 200 é menor que 60? Não! Então a resposta é falsa. O registrador $t0 receberá o valor 0. Em seguida, BNE fará a comparação do registrador

$t0 com o registrador $zero, como $t0 = 0, então t0 é igual a zero, seguindo pelo lado esquerdo do fluxograma.

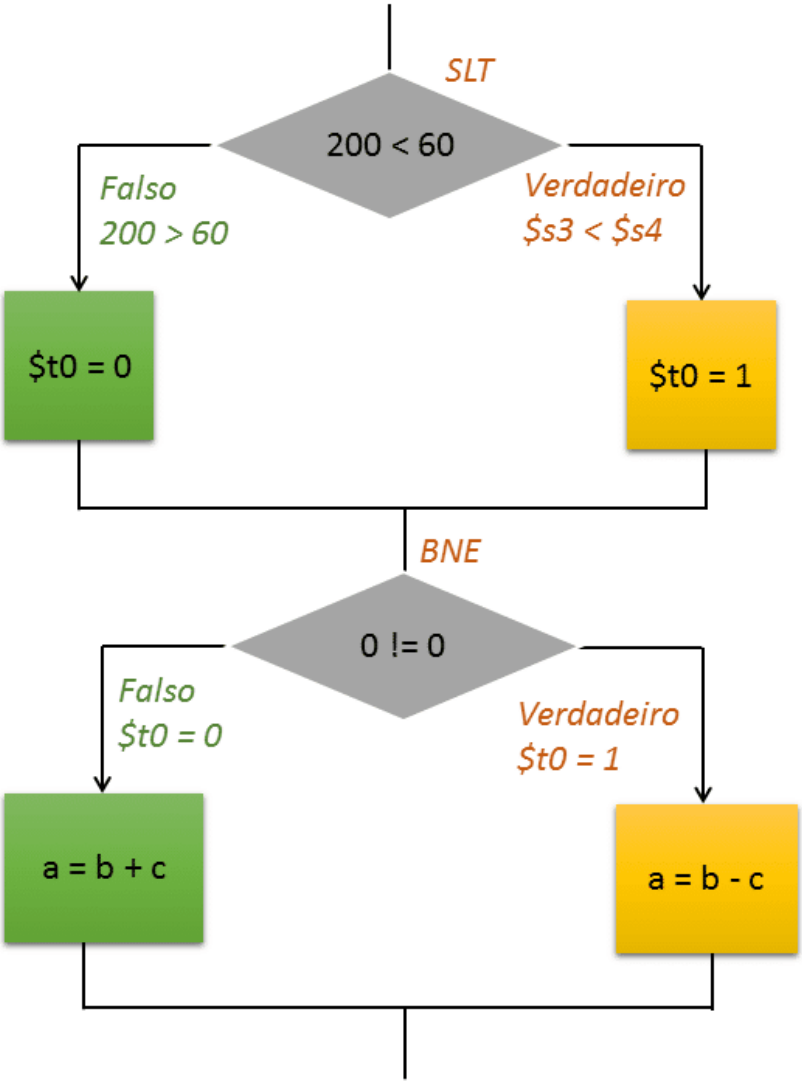


Figura 3: Representando o fluxograma quando a resposta é falsa.

b) Linguagem de Máquina

```
slt $8, $19, $20
bne $8, $zero, ELSE
add $17, $18, $16
j Exit
ELSE: sub $19, $20, $16
Exit:
```

c) Representação

Endereço de Memória	Representação					
	OPCODE	RS	RT	RD	SHAMT	FUNCT
80000	0	$19	$20	$8	0	42
80004	5	$8	$zero	80016		
80008	0	$17	$18	$16	0	32
80012	2	80020				
80016	0	$19	$20	$16	0	34
80020	. . .					

Agora vamos entender algo importante sobre os valores que devem ser inseridos no lugar dos labels ELSE e EXIT. Sabemos que o endereço 80004 pula para o endereço 80016, que é a nossa quinta linha de código, ou seja, o ELSE.

Precisamos lembrar que as instruções MIPS possuem endereços em bytes, de modo que os endereços das palavras sequenciais diferem em 4 bytes. Começamos esse bloco de comando no endereço 80000 e terminamos no endereço 80020 (exit).

Cada instrução está inserida no seu endereço correspondente que difere em quatro bytes. O cálculo que devemos fazer deve considerar o endereço SEGUINTE e NÃO o endereço atual da instrução.

Assim, a instrução BNE, na segunda linha, acrescenta duas palavras, ou oito bytes, ao endereço da instrução SEGUINTE, especificando o destino do desvio em relação à instrução seguinte, e não em relação à instrução de desvio.

O endereço da instrução seguinte é 80008. Agora, veja que curioso, se você fizer 80016 - 80008, tem-se como resultado o número 8, isto é, oito bytes, o que significa que devemos "pular" duas posições na memória, portanto, o número 2 é quem deve ir no campo endereço da instrução BNE. Se você fizer 80008 + 8, o resultado será 80016, que é exatamente o endereço para onde queremos ir.

Ainda está dificil de entender? Vou tentar simplificar, veja como a tabela deve ficar:

Endereço de Memória	**Representação**					
	OPCODE	**RS**	**RT**	**RD**	**SHAMT**	**FUNCT**
80000	0	$19	$20	$8	0	42
80004	5	$8	$zero	2		
80008	0	$17	$18	$16	0	32
80012	2	1				

80016	0	$19	$20	$16	0	34
80020	. . .					

O número 2 é o número de instruções de distância para se desviar até ELSE. O número 1 é o número de instruções de distância para se desviar até EXIT. Abstraindo, podemos resumir o cálculo MANUAL em duas fases:

a) Número de Instruções de Distância do Desvio

Qtde de bytes = endereço de memória de desvio - endereço de memória seguinte

Qtde de bytes = 80016 - 80008 = 8 bytes => 2 instruções de distância
Qtde de byte = 80020 - 80016 = 4 bytes => 1 instrução de distância

b) Cálculo do endereço de desvio

endereço desejado = endereço seguinte ao atual + quantidade de bytes

endereço desejado = 80008 + 8 = 80016
endereço desejado = 80016 + 4 = 80020

É claro que os compiladores fazem esse cálculo de forma automática, aqui estamos apresentando algo bem manual, no sentido de aprendizagem didática. Mas, no geral, o cálculo automático é feito usando o PC (Contador de Programa), que contém o valor da instrução corrente. O endereço relativo ao PC é o endereço relativo à instrução seguinte, isto é:

PC = PC + 4.

(PC = Endereço Atual + 4 bytes)

Assim, EXIT seria calculado como o conteúdo do registrador mais o campo do endereço, o qual é calculado pela própria instrução de desvio condicional. Portanto, a instrução de desvio poderia calcular algo como:

Contador de Programa = Contador de Programa + Endereço de Desvio.

Vale ressaltar aqui que os desvios condicionais tendem a desviar para a instrução mais próxima e quase metade de todos os desvios condicionais é para endereços situados a menos de 16 instruções (de "distância") da origem do desvio. Se você ainda não conseguiu entender, não se preocupe! Voltarei a falar sobre endereçamento, especificamente sobre eles, então não se preocupe tanto por hora.

d) Código de Máquina

Endereço de Memória	Representação					
	OPCODE	RS	RT	RD	SHAMT	FUNCT
80000	000 000	10 011	10 100	000 101	00 000	101 010
80004	000 101	01000	00 000	0000 0000 0000 0010		
80008	000000	10 001	10 010	10 000	00 000	100 000
80012	000 010	0000 0000 0000 0000 0000 0000 01				

80016	000 000	10 011	10 100	10 000	00 000	100 010
80020	...					

Para concluir; a arquitetura MIPS não inclui uma instrução "desvie se menor que".
Compiladores MIPS utilizam as instruções SLT, SLTI, BEQ, BNE e o valor fixo ZERO (registrador $zero) para criar todas as condições de operações relacionais: **igual, diferente, menor que, maior que, menor ou igual, maior que é maior ou igual.**

SWITCH/CASE

Este comando permite que seja feita uma escolha dentre várias, assim poderíamos ter, por exemplo, quatro opções e escolher apenas uma, como um menu. Esse tipo de seleção pode ser implementada com o switch/case ou, então, podemos usar if/then/else para fazer a mesma coisa. Veja o exemplo de código-fonte em linguagem C abaixo:

```
1  switch (k) {
2   case 0:
3     f = i + j; // k = 0
4     break;
5  case 1:
6     f = g + h; // k=1
```

```
 7      break;
 8  case 2:
 9      f = g – h; // k = 2
10      break;
11  case 3:
12      f = i – j; // k = 3
13      break;
14  }
```

K é a variável que contém o número escolhido da seleção, por exemplo, vamos supor que algo aconteceu antes de se chegar nesse trecho de código e o número 2 foi selecionado, assim, *K = 2*?

case 0 é diferente de 2, então pula,

case 1 é diferente de 2 então pula,

mas *case 2* é igual a 2, então executa o bloco de código que está ali dentro e, quando termina, sai do bloco e continua a execução das instruções seguintes.

Cada *case* aqui no MIPS será tratado como um LABEL, que chamaremos de L0, L1, e assim por diante, sendo que cada um deles será correspondente a um endereço de memória, formando algo como uma Tabela de Endereços de Desvios.

Não nos esqueçamos que *switch/case* é um comando de controle que desvia a execução do programa para o ponto desejado. Podemos imaginar essa Tabela de Endereços de Desvios da seguinte forma:

ENDEREÇO	LABEL	INSTRUÇÃO
0x0040003c	L0	f = i + j;
0x00400044	L1	f = g + h;
0x0040004c	L2	f = g – h;

0x00400054	L3	f = i – j;

Assim, precisamos criar no MIPS essa Tabela, que nada mais é que um Array:

```
.data
jTable: .word L0,L1,L2,L3      #jump table definition
```

Dei o nome de *jTable* para a minha Tabela de Endereços e defini quatro Labels *L0, L1, L2* e *L3*, agora é possível usá-los como *cases*. Agora precisamos atribuir o endereço base dessa tabela para algum registrador, da seguinte forma:

```
la  $t4, jTable                  # $t4 = base address of the jump table
```

Carregaremos em um registrador temporário ($t4) o endereço da Tabela para que ela possa ser referenciada no código Assembly, algo muito parecido com ponteiros em linguagem C.
Depois de já termos a Tabela definida, podemos continuar com o restante da tradução. Antes de testar se *case 1 = 2*, por exemplo, precisamos verificar se *K* é válido, isto é, ele precisa ser igual a um dos valores presentes nos Labels, portanto, *K* deve estar entre 0 e 3 (4 valores diferentes). Se *K* não estiver dentro dessa faixa de valores, o *switch* não pode ser executado. Para fazermos isso temos de utilizar instruções de comparação, a **SLT** pode ser então escolhida para fazer essa tradução:

```
slt $t3, $s5, $zero              # $t3 = ($s5 <= 0)
```

O registrador **$t3** é temporário e armazenará o resultado da comparação entre o registrador **$s5**, que é *K*, e o registrador **$zero**, testando assim se **K < 0.** A instrução **BNE** pode nos ajudar a direcionar os desvios, lembrando que essa instrução significa “*Branch Not Equal*”,

ou seja, “desvie se não for igual”, assim, se **$t3** for diferente de zero, então desvia para **EXIT**, caso contrário executa as próximas instruções:

```
bne $t3, $zero, EXIT          # desvia para EXIT se $t3 < > 0
```

Ótimo, testamos se **K < 0**, agora precisamos testar o outro limite, isto é, *K* precisa estar entre 0 e 3, portanto testa se **K < 4**:

```
slti $t3, $s5, 4          # $t3 = ($s5 <= 4)
```

Usamos agora a instrução **SLTI** pois precisamos comparar o valor de **$s5** (K) com um imediato (4). Lembrando que o resultado da comparação entre o registrador **$s5** (K) é armazenado em **$t3**, o qual será usado na instrução **BEQ** (*Branch if Equal* – “desvie se igual”) para desviar ou não para **EXIT**. Se **$t3** for igual a zero, então desviará para **EXIT**, caso contrário executará o bloco de comandos.

```
beq $t3, $zero, EXIT          # desvia para EXIT se $t3 = 0
```

Dessa forma vai desviar para EXIT se **K >= 4**. Para elucidar e ficar melhor de entender esta sequência, vamos supor que K = 6.

slt $t3, $s5, $zero	$s5 <= $zero 6 <= 0 F Portanto, $t3 = 0
bne $t3, $zero, EXIT	$t3 < > $zero 0 < > 0 F Portanto, não desvia!
slti $st3, $s5, 4	$s5 <= 4 6 <= 4

	F Portanto, $t3 = 0
beq $t3, $zero, EXIT	$t3 = $zero 0 = zero V Portanto, desvia para EXIT

Vamos ver outro exemplo sendo K = 2.

slt $t3, $s5, $zero	$s5 <= $zero 2 <= 0 F Portanto, $t3 = 0
bne $t3, $zero, EXIT	$t3 < > $zero 0 < > 0 F Portanto, não desvia!
slti $t3, $s5, 4	$s5 < 4 2 < 4 V Portanto, $t3 = 1
beq $t3, $zero, EXIT	$t3 = $zero 1 = zero F Portanto, não desvia

Vamos ver outro exemplo sendo K = 0.

slt $t3, $s5, $zero	$s5 <= $zero 0 <= 0 F Portanto, $t3 = 0

bne $t3, $zero, EXIT	$t3 < > $zero 0 < > 0 F Portanto, não desvia!
slti $st3, $s5, 4	$s5 < 4 0 < 4 V Portanto, $t3 = 1
beq $t3, $zero, EXIT	$t3 = $zero 1 = zero F Portanto, não desvia

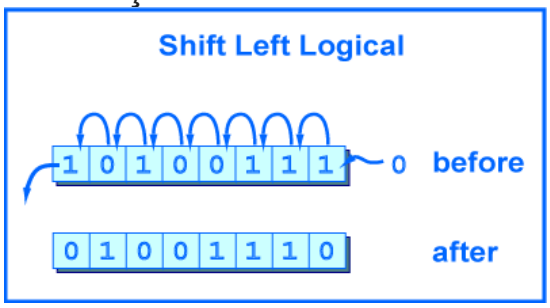
Vamos finalizar com K = -1

slt $t3, $s5, $zero	$s5 <= $zero -1 <= 0 V Portanto, $t3 = 1
bne $t3, $zero, EXIT	$t3 < > $zero 1 < > 0 V Portanto, desvia para EXIT!

Agora que a verificação de K está traduzida, precisamos fazer o cálculo do endereço dos labels, assim o primeiro passo é deslocar em 4 bytes para cada valor de k, de modo a selecionar o endereço correto no array.

```
sll $t1, $s5, 2          # calcula o endereçamento de 4 bytes
```

A instrução SLL fará a soma de mais 4 bytes (ao endereço de $s5 (k)),



agora precisamos somar isso com o endereço da Tabela de Desvios ($t4):

```
add $t1, $t1, $t4                # $t1 será o endereço de jTable
```

De posse do endereço completo, podemos agora carregar o valor da Tabela. Não nos esqueçamos que a *jTable* começa no endereço que está em $t4, assim o endereço do desvio é carregado em um registrador temporário que neste exemplo será $t0:

```
lw  $t0, 0($t1)                  # $t0 é onde está o label desejado
```

A execução de uma instrução de desvio para o conteúdo de um registrador faz com que o programa passe a executar a instrução apontada na tabela de endereços de desvio, assim precisamos usar a instrução JR para que o desvio ocorra:

```
jr $t0                           # desvia com base no conteúdo de
$t0
```

Essa instrução vai forçar o desvio para o Label escolhido, por exemplo, se o endereço é de L2, ele vai desviar para lá exatamente! Então, notem que a dinâmica aqui é um pouco diferente de como acontece na programação de médio e alto nível, temos de pensar um pouquinho diferente do que estamos acostumados. A compilação feita até agora é

a parte mais difícil da tradução de um código *switch/case*, os *cases* por si só são bem fáceis de se traduzir, vejam:

```
L0:     add $s0, $s3, $s4       # Se K = 0 então f = i + j
j EXIT                          # fim deste case, desvia para EXIT

L1:     add $s0, $s1, $s2       # Se K = 1 então f = g + h
j EXIT                          # fim deste case, desvia para EXIT

L2:     sub $s0, $s1, $s2       # Se K = 2 então f = g - h
j EXIT                          # fim deste case, desvia para EXIT

L3:     sub $s0, $s3, $s4       # Se K = 3 então f = i – j
EXIT:                           # sai do Switch definitivamente
```

No último case podemos eliminar o desvio para a saída do comando SWITCH, pois as instruções deste case são as últimas, no entanto, um LABEL EXIT deve ser adicionado depois do último comando deste case para marcar o final do comando *switch*. Assim, o código na íntegra, para você testar no MARS, fica da seguinte forma:

```
1  # Definindo a Jump Table
2  .data
3  jTable: .word L0,L1,L2,L3
4
5  .text
6
7  # Definindo as variáveis
8  li  $s1, 15                  # g = $s1 = 15
```

```
 9  li  $s2, 20                 # h = $s2 = 20
10 li  $s3, 10                 # i = $s3 = 10
11 li  $s4, 5                  # j = $s4 = 5
12 li  $s5, -1                 # k = $s5 = 2
13 la  $t4, jTable                 # $t4 = base address of the jump table
14
15 # Verificando os limites de K
16 slt  $t3, $s5, $zero
17 bne  $t3, $zero, EXIT
18 slti $t3, $s5, 4
19 beq  $t3, $zero, EXIT
20
21 # Calculando o endereço correto do Label
22 sll $t1, $s5, 2
23 add $t1, $t1, $t4
24 lw  $t0, 0($t1)
25
26 # Seleção do Label
27 jr $t0
28
29 # Casos
30 L0:     add $s0, $s3, $s4
31 j EXIT
32 L1:     add $s0, $s1, $s2
33 j EXIT
34 L2:     sub $s0, $s1, $s2
35 j EXIT
36 L3:     sub $s0, $s3, $s4
37 EXIT:
38 #FIM
```

Para finalizar, vamos fazer a tradução para:

a) Linguagem de máquina:

```
.data
jTable: .word L0,L1,L2,L3
.text
li $17, 15
li $18, 20
li $19, 10
li $20, 5
li $21, 2
la $12, jTable
slt $11, $21, $zero
bne $11, $zero, EXIT
slti $11, $21, 4
beq $11, $zero, EXIT
sll $9, $21, 2
add $9, $9, $12
lw $8, 0($9)
jr $8
L0:     add $16, $19, $20
j EXIT
L1:     add $16, $17, $18
j EXIT
L2:     sub $16, $17, $18
j EXIT
L3:     sub $16, $19, $20
EXIT:
```

Exercício para entrega (moodle)

Utilizando a estrutura aprendida para a criação do switch, implemente a seguinte instrução em assembly do MIPS, com as condicionais aprendidas no laboratório anterior com a seguinte definição:

```
switch(x) {
        case 1: IF      # rodar uma rotina que implemente  um IF
                break;
        case 2: IF/Then/Else  # Implemente uma versão do IF/Else
                break;
        case 3: while   #implementar uma versão do while
                break;
        case 4: do/while  #implementar uma versão do do/while
                break;
}
```

Referencias

PATTERSON, D. A.; HENNESSY, J. L. “Organização e Projeto de Computadores: A Interface Hardware/Software”. 4.ª Edição, Rio de Janeiro: Elsevier, 2014.

GATTO, E.C. Compilando o comando switch/case no MIPS - Embarcados. Embarcados - Sua fonte de informações sobre Sistemas Embarcados. 2018. URL: <https://www.embarcados.com.br/compilando-switch-case-no-mips/> (acesso 14/03/19).