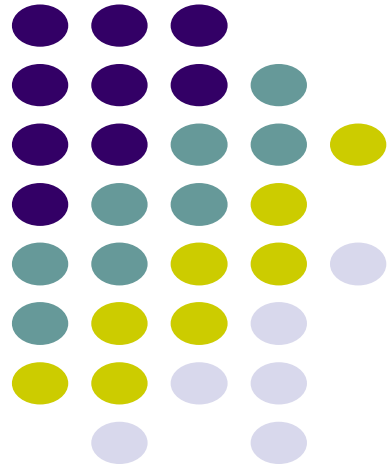
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

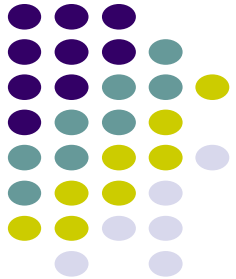
Arquitetura e Organização de Computadores

Instruções: a linguagem do computador
Parte III



Prof. Sílvio Fernandes

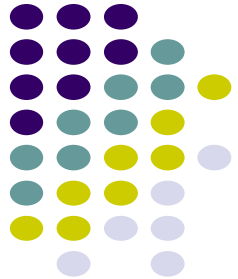
Instruções para tomada de decisões



“A utilidade de um computador reside na possibilidade de executar repetidamente uma sequência de instruções, sendo o número de repetições dependente do resultado de alguma computaçao anterior. Quando a iteração é concluída, uma sequência diferente de [instruções] deve ser seguida, de modo que devemos, em quase todos os casos, oferecer dois fluxos de [instruções] em paralelo precedidos por uma instrução indicando qual rotina deve ser seguida.”

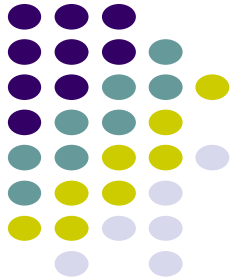
Burks, Goldstine and Von Neumann, 1947

Instruções para tomada de decisões



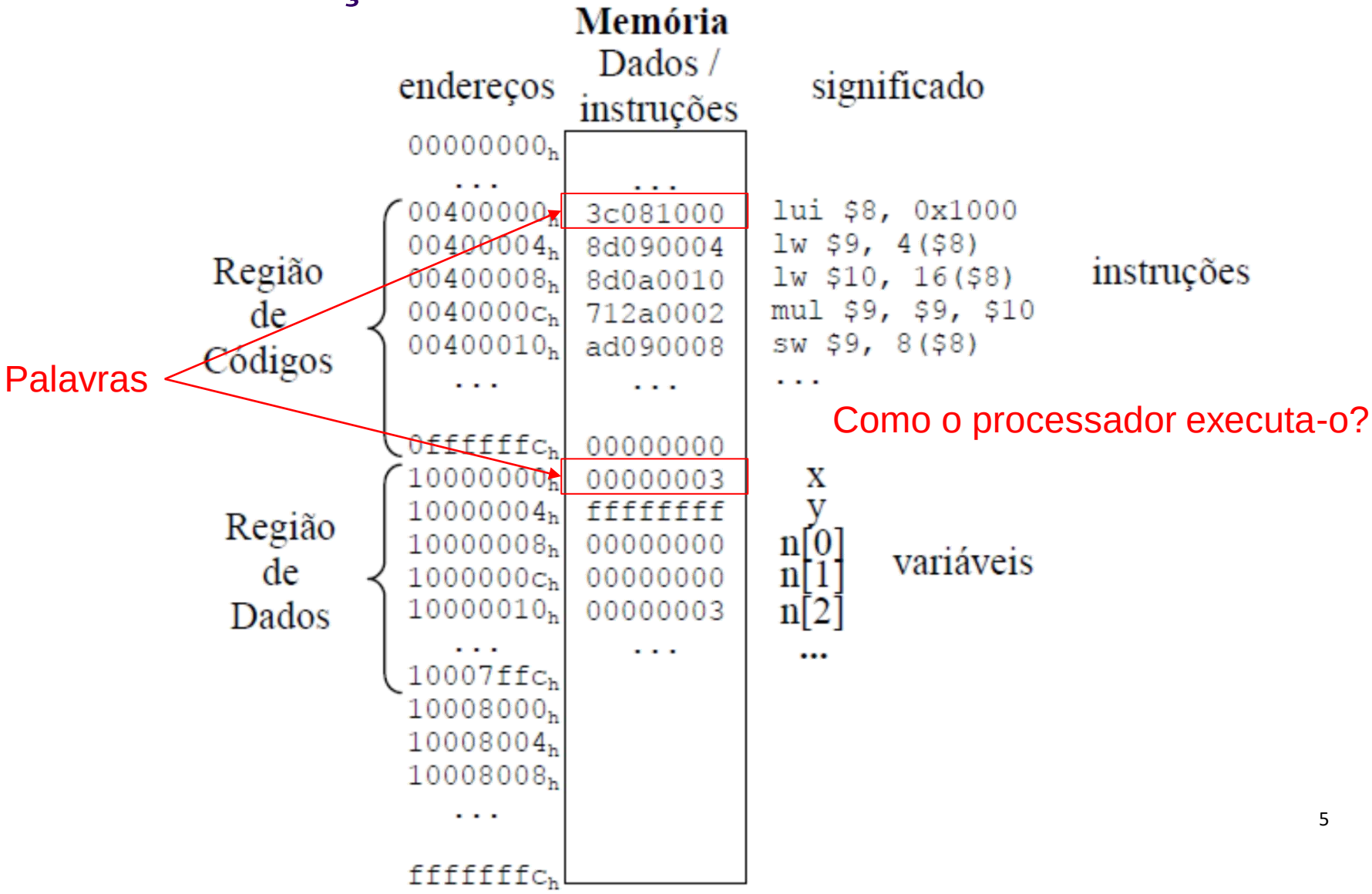
- Instruções para tomada de decisões
  - Desvio incondicional
- Suporte a procedimentos no hardware do computador
- Endereçamento no MIPS para operandos e endereços de 32 bits.
- Endereçamento em desvios condicionais e incondicionais (jumps).

Instruções para a tomada de decisões.

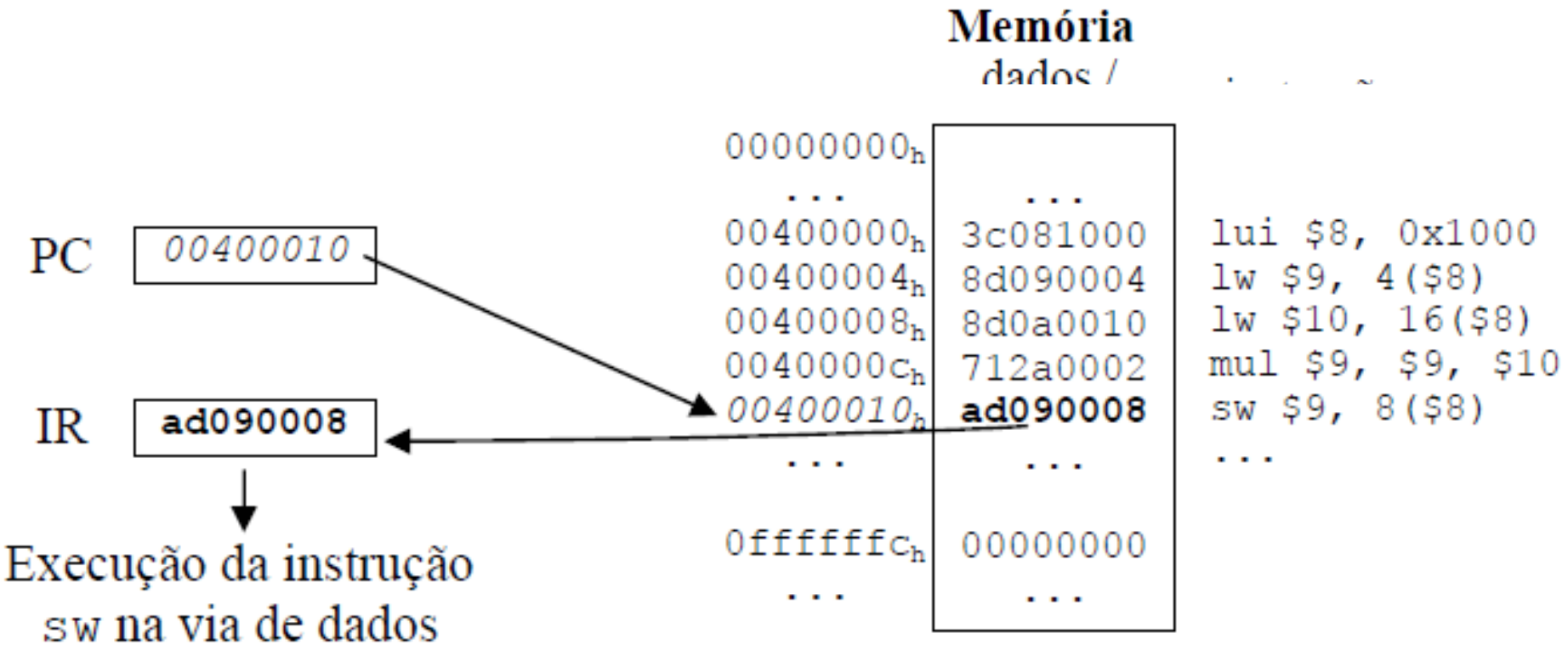


- O que distingue um computador de uma calculadora simples?
  - Capacidade de tomar decisões com base nos dados de entrada e valores calculados.
- Em linguagens de alto nível, estruturas típicas de decisão são as construções **if**, algumas vezes acompanhadas de **goto** e rótulos (labels).
- O assembly do MIPS possue duas instruções que dão suporte para tomada de decisões (**beq** e **bne**).

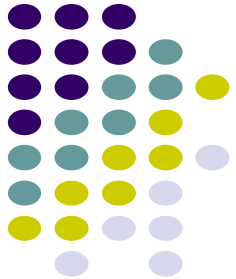
Variáveis e instruções na memória



Execução de cada instrução do programa

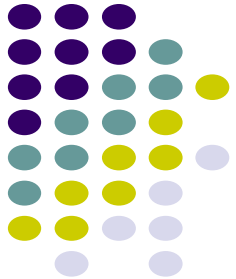


Instruções para a tomada de decisões.



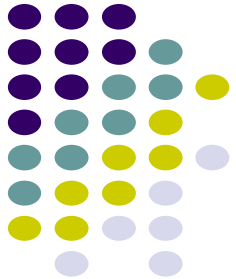
- beq
  - Instrução utilizada quando desejamos comparar se dois valores armazenados em registradores, são iguais.
  - Sintaxe:
    - beq $r1, $r2, L1,
      - Onde $r1, $r2 são os registradores cujos valores armazenados vão ser comparados.
      - Se os valores são iguais, a sequência de execução pula para a instrução que possui o rótulo L1.
      - Um rótulo é um "apelido" para um endereço de memória de uma outra instrução

Instruções para a tomada de decisões.



- bne
  - Instrução utilizada quando desejamos comparar se dois valores armazenados em registradores, são diferentes.
  - Sintaxe:
    - bne $r1, $r2, L1,
      - Onde $r1, $r2 são os registradores cujos valores armazenados vão ser comparados.
      - Se os valores são diferentes, a sequência de execução pula para a instrução que possui o rótulo L1.
- beq e bne são instruções conhecidas como ***desvios condicionais.***
  - ***Desvios condicionais*** são instruções que requerem a comparação de dois valores e que leva em conta uma transferência de controle subsequente para um novo endereço.

Desvio incondicional

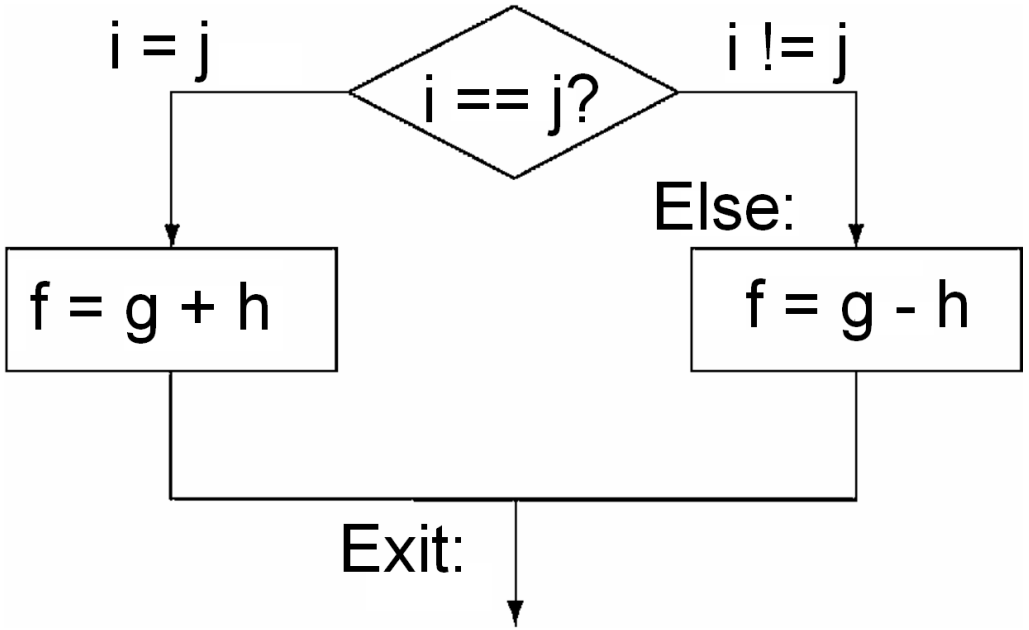


- O assembly do MIPS também dá suporte a instrução de desvio incondicional.
  - Basicamente, um desvio incondicional é uma instrução que ***sempre diz que o processador deverá seguir o desvio.***
  - A instrução para isso é a instrução j (jump).
  - Sintaxe:
    - *j rotulo #pula para a instrução precedida por rotulo*

Instruções para a tomada de decisões.

- Vamos ver estas instruções na prática.
  - Dada a seguinte construção C ou Java, qual o assembly obtido?

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

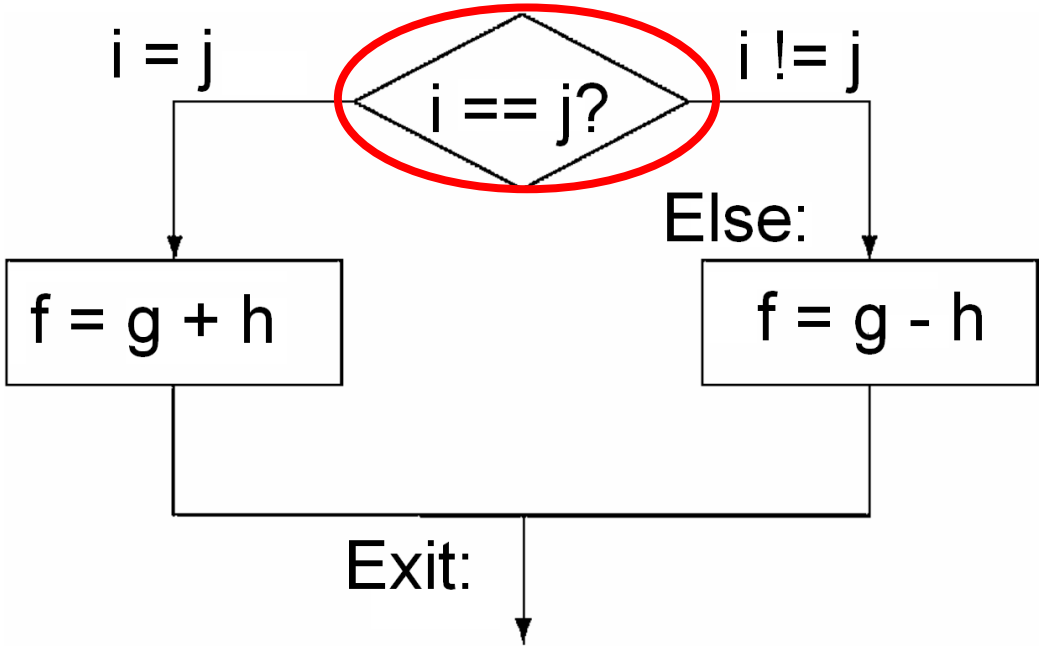


- Calma isso ainda não é o assembly ☺

Instruções para a tomada de decisões.

i está em $s3
j está em $s4
g está em $s1
h está em $s2
f está em $s0

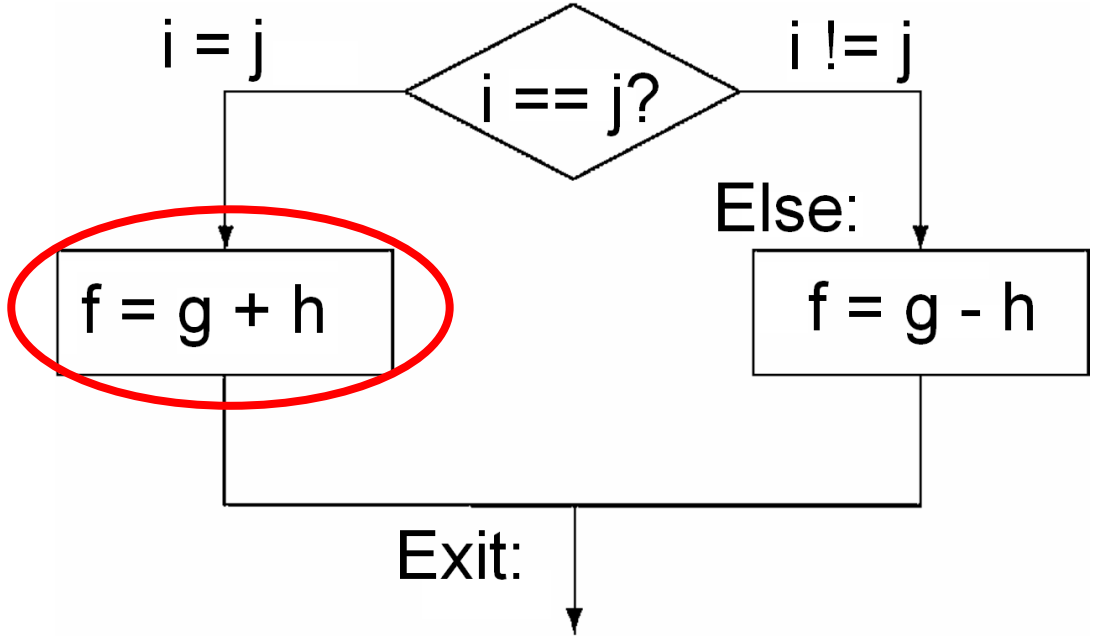
```
bne $s3, $s4, Else
```



Instruções para a tomada de decisões.

i está em $s3
j está em $s4
g está em $s1
h está em $s2
f está em $s0

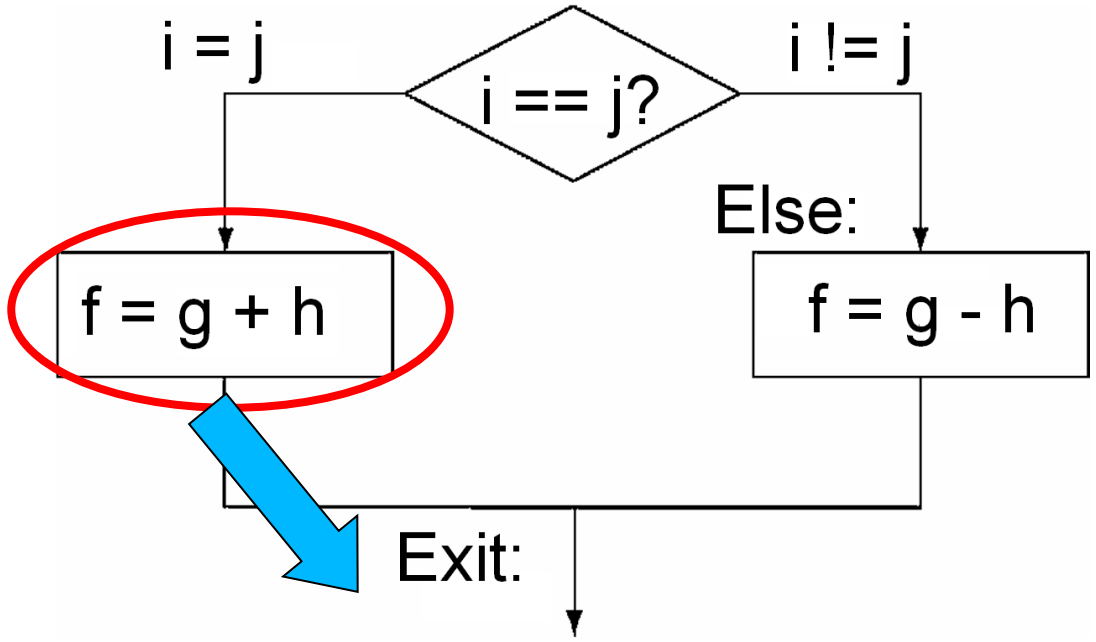
```
bne $s3, $s4, Else
add $s0, $s1, $s2
```



Instruções para a tomada de decisões.

i está em $s3
j está em $s4
g está em $s1
h está em $s2
f está em $s0

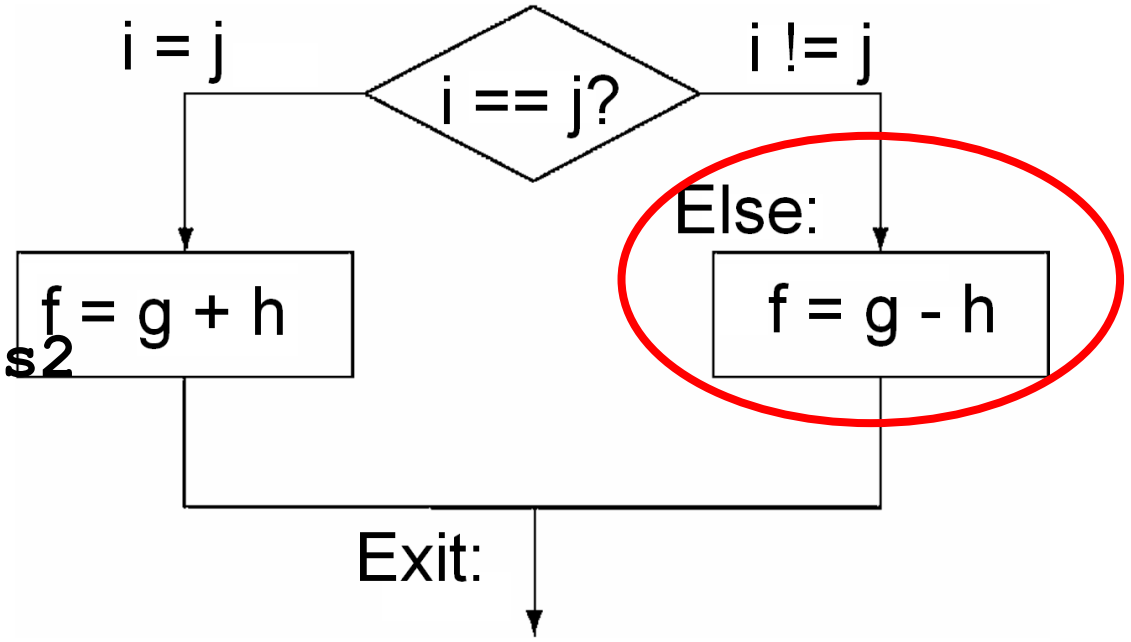
```
bne $s3, $s4, Else
add $s0, $s1, $s2
j Exit;
```



Instruções para a tomada de decisões.

i está em $s3
j está em $s4
g está em $s1
h está em $s2
f está em $s0

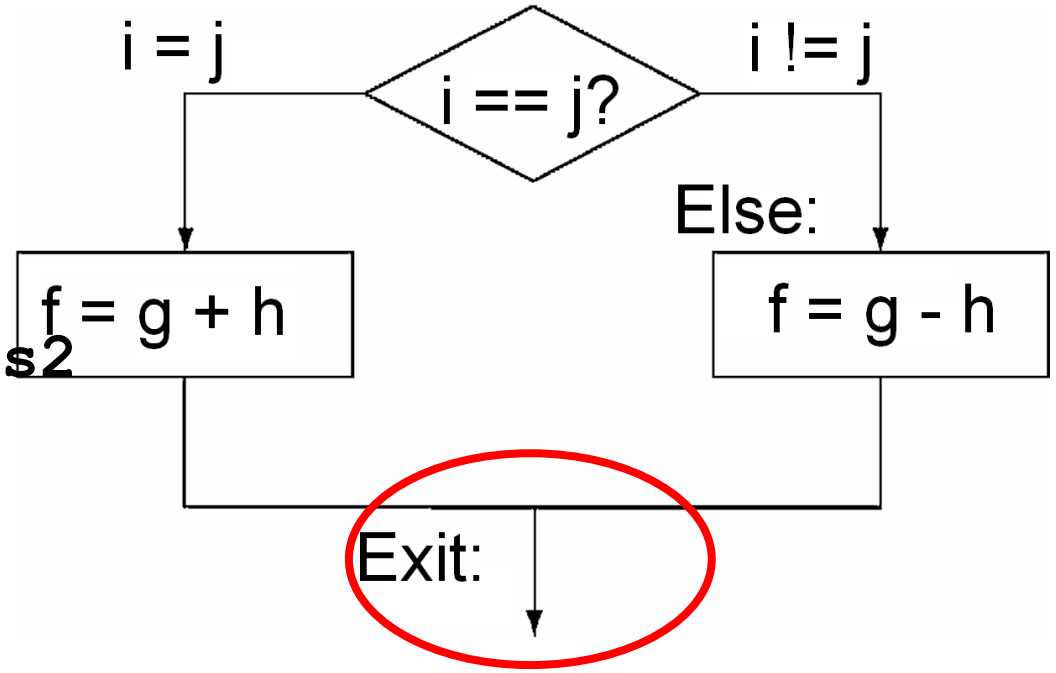
```
bne $s3, $s4, Else
add $s0, $s1, $s2
j Exit;
Else:sub $s0, $s1, $s2
```



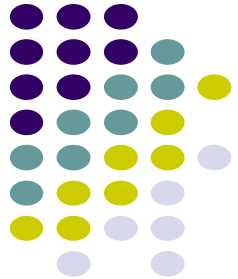
Instruções para a tomada de decisões.

i está em $s3
j está em $s4
g está em $s1
h está em $s2
f está em $s0

```
bne $s3, $s4, Else
add $s0, $s1, $s2
j Exit
Else:sub $s0, $s1, $s2
Exit:
```

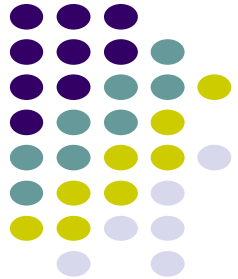


Instruções para a tomada de decisões.



- Observem que o programador assembly não precisa se preocupar com o cálculo do endereço utilizado nos desvios.
- E um laço, como seria?
- Nos restringiremos ao laço while. Os demais laços são bastante semelhantes.

Instruções para a tomada de decisões.

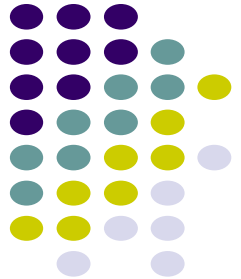


- Seja o seguinte código C ou Java

```
while (save[i] == k)
   i += 1;
```

- Qual seria o assembly correspondente, supondo que os valores de i e k estão nos registradores $s3 e $s5, e a base do array save em $s6?

Instruções para a tomada de decisões.



- 1. Realizar a leitura de save[i]

```
Loop: sll $t1, $s3, 2     # $t1 = 4 * i
add $t1, $t1, $s6         # $t1 = endereço de save[i]
lw $t0, 0($t1)            # $t0 = save[i]
```

- 2. Teste do loop, terminando se save[i] != k

```
Bne $t0, $s5, Exit     #vá para exit se save[i] != k
```

- 3. Senão, adiciona 1 a i e volta para o início.

```
addi $s3, $s3, 1          # i = i + 1
j       Loop
Exit:
```

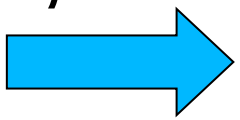
i: $s3
k: $s5
save: $s6

```
while (save[i] == k)
    i += 1;
```

Instruções para a tomada de decisões.

- Código fonte

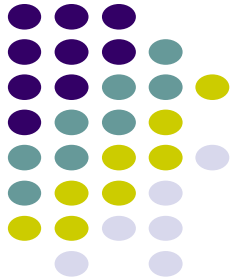
```
while (save[i] == k)
   i += 1;
```



- Resultado Final

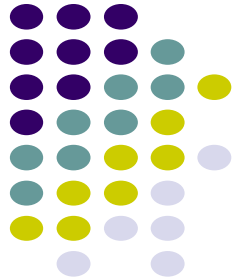
```
Loop: sll $t1, $s3, 2
      add $t1, $t1, $s6
      lw $t0, 0($t1)
      bne $t0, $s5, Exit
      addi $s3, $s3, 1
      j     Loop
Exit:
```

Instruções para a tomada de decisões.



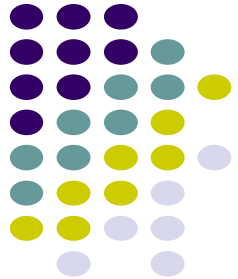
- Estas sequências de instruções sem desvios (exceto, possivelmente, no final) e sem destinos de desvio ou rótulos de desvio (exceto, possivelmente, no início) são conhecidas como ***blocos básicos.***
- Blocos básicos são muito importante e constituem uma das etapas do projeto de compiladores, basicamente, através deles, podemos realizar algumas otimizações no programa.

Instruções para a tomada de decisões.



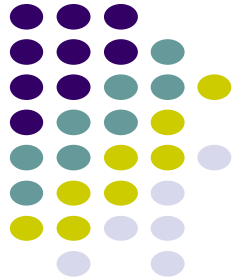
- As instruções slt e slti
  - Embora o teste de igualdade e desigualdade seja bastante popular, precisamos também de um outro tipo de comparação.
  - A instrução slt (*set on less than*) é usada quando desejamos verificar se o valor armazenado em um registrador é menor que o valor armazenado em um outro registrador.
  - A instrução slti é usada quando desejamos verificar se o valor armazenado em um registrador é menor que o valor de uma constante literal.
  - Um terceiro registrador recebe 1 quando o primeiro valor for menor que o segundo

Instruções para a tomada de decisões.



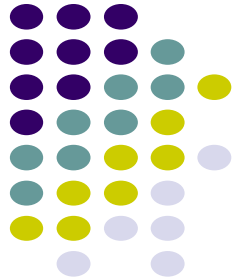
- Sintaxe de uso
  - slt $t1, $r1, $r2
    - Basicamente, se o valor em $r1 for menor que o valor em $r2, $t1 recebe o valor 1. Caso contrário, $t1 recebe o valor 0.
  - slti $t1, $r1, constante
    - Basicamente, se o valor em $r1 for menor que o valor da constante literal, $t1 recebe o valor 1. Caso contrário, $t1 recebe o valor 0.
- E se quisermos fazer algo do tipo se i > j faça, tem como?
- Senão tiver como fazer, porque o MIPS é assim?

Instruções para a tomada de decisões.



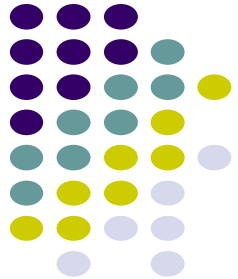
- Realmente não temos como fazer isso, mas há uma razão bastante lógica.
- Se o MIPS tivesse uma instrução que já desempenhasse ambos os papéis ele estaria ferindo sua restrição de projeto (no qual, o processador seria dotado de apenas instruções simples).
- Logo é preferível quebrar esta instrução mais complexas em duas mais simples e, caso tenhamos necessidade de reproduzirmos um se i < 4 então, utilizamos as instruções apresentadas anteriormente.

Instruções para a tomada de decisões.



- O registrador $zero
  - No MIPS temos um registrador especial que sempre armazena o valor 0.
  - O registrador $zero em conjunto com slt, slti, beq, bne criam todas as condições relativas: igual, diferente, menor que, maior que, menor ou igual e maior ou igual.

Instruções para a tomada de decisões.



- Instruções Case/Switch
  - O modo mais simples de implementá-las é através de uma cadeia de if-then-else.
  - Outra forma de implementá-las é através de uma ***tabela de endereços de desvio***.
    - Para apoiar tais situações, o processador MIPS possui a instrução de ***desvio incondicional*** jr (jump register) que pula para o endereço armazenado pelo registrador.
    - Sintaxe:
      - jr $r1

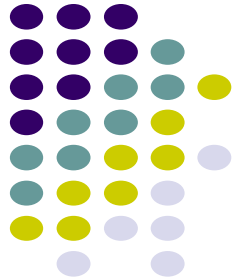
Operandos MIPS

Nome	Exemplo	Comentário
32 registradores	$s0, $s1, ..., $s7 $t0, $t1, ..., $t7	Locais rápidos para dados. No MIPS, os dados precisam estar em registradores para a realização de operações aritméticas. Os registradores $s0-$s7 são mapeados para 16-23; $t0-$t7 são mapeados para 8-15. O registrador $zero sempre é igual a 0.
2^{30} words na memória	Memória[0], Memória[4], ... Memória[4294967292]	Acessadas apenas por instruções de transferência de dados no MIPS. O MIPS utiliza endereços em bytes, de modo que os endereços em words sequenciais diferem em 4 vezes. A memória contém estruturas de dados, arrays e spilled registers.

Assembly do MIPS

Categoria	Instrução	Exemplo	Significado	Comentário
Aritmética	Add	add $s1, $s2, $s3	$s1 = $s2+$s3	3 operandos; dados registradores
	Subtract	sub $s1, $s2, $s3	$s1 = $s2-$s3	3 operandos; dados registradores
	Add Immediate	dddi $s1, $s2, 100	$s1 = $s2+100	Usada para somar constantes
Lógica	And	and $s1, $s2, $s3	$s1 = $s2 & $s3	3 operandos em reg; AND bit a bit
	Or	or $s1, $s2, $s3	$s1 = $s2 | $s3	3 operandos em reg; OR bit a bit
	Nor	nor $s1, $s2, $s3	$s1 = ~($s2 | $s3)	3 operandos em reg; NOR bit a bit
	And Immediate	andi $s1, $s2, 100	$s1 = $s2 & 100	AND bit a bit entre reg. e constante
	Or Immediate	ori $s1, $s2, 100	$s1 = $s2 | 100	OR bit a bit entre reg. e constante
	Shift left logical	sll $s1, $s2, 10	$s1 = $s2 << 10	Deslocamento à esquerda por const.
	Shift rigth logical	srl $s1, $s2, 10	$s1 = $s2 >> 10	Deslocamento à direita por const.
Transferência de dados	Load word	lw $s1, 100($s2)	$s1 = Mem[$s2+100]	Dados da mem. para o registrador
	Store Word	sw $s1, 100($s2)	Mem[$s2+100] = $s1	Dados do registrador para a mem.

Assembly do MIPS



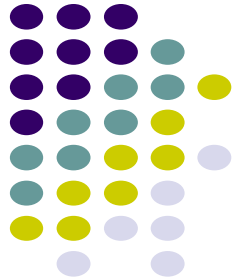
Categoria	Instrução	Exemplo	Significado	Comentário
Desvio Condicional	Branch on equal	beq $s1, $s2, L	If($s1 == $s2) go to L	Testa igualdade e desvia
	Branch on not equal	bne $s1, $s2, L	If($s1 != $s2) go to L	Testa desigualdade e desvia
	Set on less than	stl $s1, $s2, $s3	If($s2 < $s3) $s1 = 1; Else $s1 = 0	Compara menor que
	Set on less than immediate	stli $s1, $s2, 100	If($s2 < 100) $s1 = 1; Else $s1 = 0	Compara menor que imediato
Desvio incondicional	Jump	j L	go to L	Desvia para endereço de destino
	Jump register	jr $s1	go to [$s1]	Devia para o end. no registrador

Questões

- Escrever um programa em assembly e testar no MARS.
  - Considere duas variáveis A e B inicializadas
  - Leia as 2 variáveis
  - Teste se A é maior que B
    - Se sim, imprima “O valor A é maior que B” (onde A e B são os valores das variáveis)
    - Caso contrário, “O valor A é menor ou igual a B” (onde A e B são os valores das variáveis)

Questões

- Escrever um programa em assembly com 4 variáveis (n1, n2, n3 e n4) e teste no MARS.
  - Leia as 3 primeiras variáveis
  - Faça um somatório delas
  - Teste se o valor é maior ou igual a 21
    - Se for, imprima "O valor X atingiu o esperado com 3 valores" (onde X é o seu resultado)
    - Caso contrário, leia a 4ª. variável e some ao resultado anterior e faça um novo teste:
      - Teste se o somatório é maior ou igual a 24
        - Se for imprima "O valor X atingiu a meta com 4 valores"
        - Senão imprima "Não foi dessa vez. O valor X está abaixo do esperado"



Questões

- A linguagem C possui muitas instruções para decisões e loops, enquanto MIPS possui poucas. Quais dos seguintes itens explicam ou não explicam esse desequilíbrio? Por quê?
  1. Mais instruções de decisão tornam o código mais fácil de ler e entender
  2. Menos instruções de decisão simplificam a tarefa da camada inferior responsável pela execução.
  3. Mais instruções de decisão significam menos linhas de código, o que geralmente reduz o tempo de codificação.
  4. Mais instruções de decisão significam menos linhas de código, o que geralmente resulta na execução de menos operações.

Questões

- Por que a linguagem C oferece dois conjuntos de operadores para AND (& e &&) e dois conjuntos de operadores OR (| e ||), enquanto MIPS não faz isso?
  1. As operações lógicas AND e OR implementam & e |, enquanto os desvios condicionais implementam && e ||.
  2. A afirmativa anterior é o contrário: && e || correspondem a operações lógicas, enquanto & e | são mapeadas para desvios condicionais.
  3. Elas são redundantes e significam a mesma coisa: && e || são simplesmente herdados da linguagem B, a antecessora do C.

Referências

- PATTERSON, D. A. ; HENNESSY, J.L. Organização e projeto de computadores – a interface hardware software. 3. ed. Editora Campus, 2005.
- WANDERLEY NETTO, Bráulio. **Arquitetura de Computadores**: A visão do software. Natal: Editora do CEFET-RN, 2005
- Notas de aula do Prof. André Luis Meneses Silva