Conjunto Instruções MIPS

Luciano L. Caimi

lcaimi@uffs.edu.br

Introdução

- Linguagem de Máquina
- Mais primitiva que linguagens de alto nível
  i.e., controle de fluxo não sofisticado
- Muito restritiva

  operandos fixos

  modos de endereçamento
  ex. MIPS Instruções Aritméticas

- Nós trabalharemos com a arquitetura do conjunto de instruções do MIPS
  - similar a outras arquiteturas desenvolvidas após 1980's

Banco de Registradores

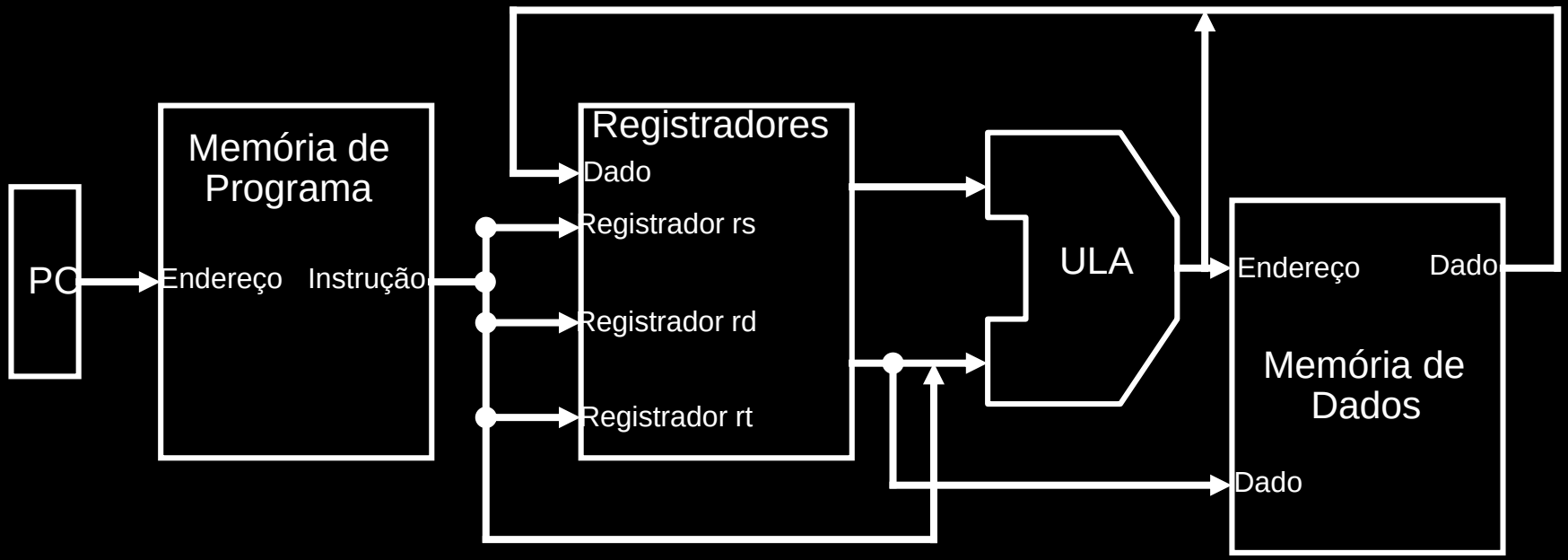
- O MIPS utiliza uma arquitetura LOAD/STORE, ou seja, apenas estas instruções fazem acesso dados na memória
- Dados das instruções são obtidos a partir de registradores
- Possui 32 registradores de 32 bits cada
- Nomeados através da notação: $??

Registradores do MIPS

Nome	Número	Utilização	Preservado
$zero	0	Valor de constante 0	---
$at	1	Reservado para o montador	---
$v0-$v1	2-3	Uso geral em 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
$k0-$k1	26-27	Reservado para o Sist. Oper.	---
$gp	28	Global Pointer	Sim
$sp	29	Stack Pointer	Sim
$fp	30	Frame Pointer	Sim
$ra	31	Retorno a procedimento	Sim

Visão Geral

Abstração / Vista Simplificada:



- Dois tipos de unidades funcionais:
  - elementos que operam com valores (combinatório)
  - elementos que contém estados (seqüenciais)

MIPS - Aritmética

- Todas instruções tem 3 operandos
- A ordem dos operandos é fixa(destino primeiro)

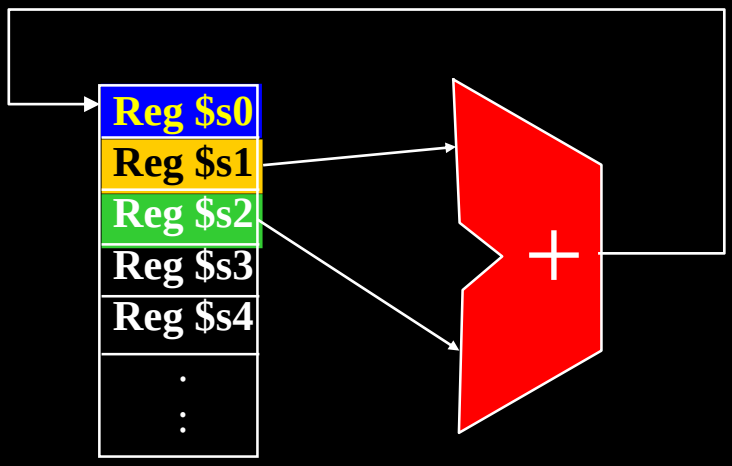
  Exemplo:

  Código C: `A = B + C`

  Código ASM: `ADD A, B, C`

  Código MIPS: `ADD $s0, $s1, $s2`

  rd ← rs operação rt



MIPS - Aritmética

- Princípio de projeto: simplicidade favorece a regularidade

  Ex: instruções aritméticas com formato único de 3 operandos

- Código C: A = B + C + D;
  E = F - A;

- Código ASM: add A, C, D
  add A, A, B
  sub E, F, A

Registradores

Princípio de Projeto: Menor é mais rápido

Ex: - banco de registradres com tamanho 32

- arquitetura RISC

Código C: A = B + C + D;

E = F - A;

Código MIPS: add $t0, $s1, $s2

add $s0, $t0, $s3

sub $s4, $s5, $s0

descrição	op	rs	rt	rd	shamt	funct
bits por campo	6	5	5	5	5	6
tamanho da instrução	32 bits					

Exemplo

- **Código C:**

  f = (g + h) – (i + j)

- **Código MIPS:**

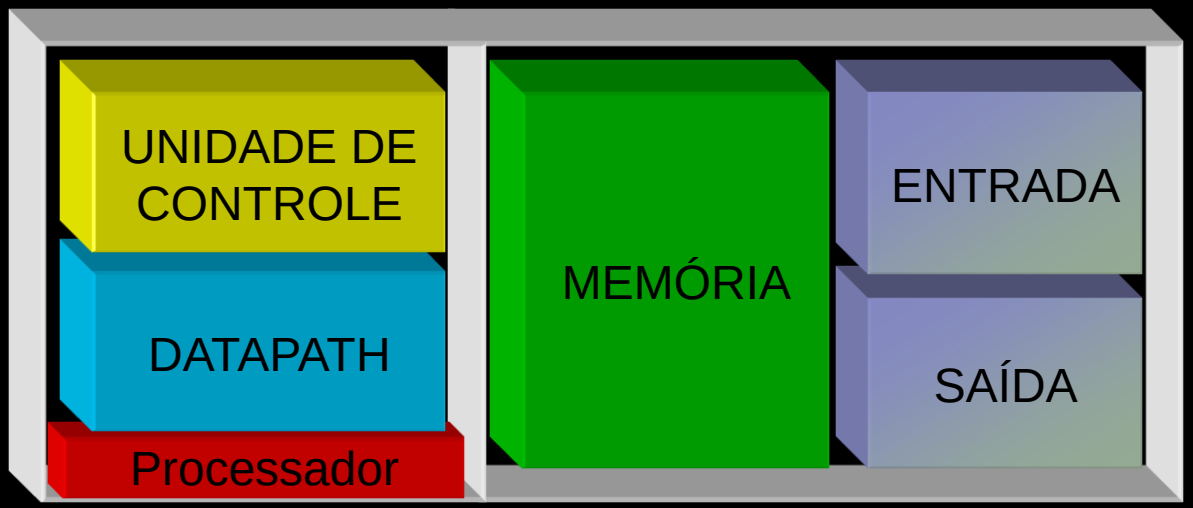
```
add $t0, $s0,$s1
add $t1, $s2,$s3
sub $s4, $t0,$t1
```

Mapeamento de registradores

f = $s4 g = $s0 h = $s1 i = $s2 j = $s3

Registradores X Memória

- Operandos de instruções Aritméticas devem ser registradores:
  - só existem 32 registradores
- Compilador associa variáveis com registradores
- Como fazer quando um programa tem muitas variáveis?
- Como fazer com matrizes e vetores?



Organização da Memória

- Memória é vista como um vetor (matriz unidimensional), com um endereço associado a cada célula
- Um endereço de memória é um índice em uma matriz
- Cada célula armazena um byte, assim cada endereço aponta para 1 byte na memória
- Chamamos isto de memória endereçada a byte

Organização da Memória

- Para o MIPS, uma palavra tem 32 bits ou 4 bytes
  Tamanho da palavra = 32 bits

- Espaço de endereçamento de 2^{32} células = 4Gcélulas com endereços desde 0 até 2^{32}-1

- 2^{30} palavras com endereços: 0, 4, 8, ... 2^{32}-4

- As instruções que referenciam a memória endereçam palavras, assim para obtermos o endereço da célula devemos multiplicar por 4

- As palavras são alinhadas na memória
  Quais são os dois bits menos significativos do endereço?

Instruções de Transf. de Dados

MIPS – Transferência de Dados

- Instruções load e store

  LOAD: leitura de dados na memória

  lw $reg_destino, deslocamento($reg_base);
  $reg_destino <− ($reg_base + deslocamento)

  lw $rt, deslocamento($rs);
  $rt <− ($rs + deslocamento)

lw $t0, 0($t1); $t0 <− ($t1 + 0)

descrição	op	rs	rt	offset
bits por campo	6	5	5	16
tamanho da instrução	32 bits			

MIPS – Transferência de Dados

STORE: armazenamento de dados na memória.

sw $reg_origem, deslocamento($reg_base);
($reg_base + deslocamento) <− $reg_origem

sw $rt, deslocamento($rs);
($rs + deslocamento) <− $rt

sw $t0, 0($t1); ($t1 + 0) <− $t0

descrição	op	rs	rt	offset
bits por campo	6	5	5	16
tamanho da instrução	32 bits			

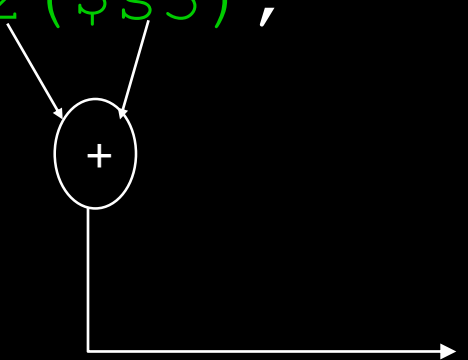
MIPS – Transferência de Dados

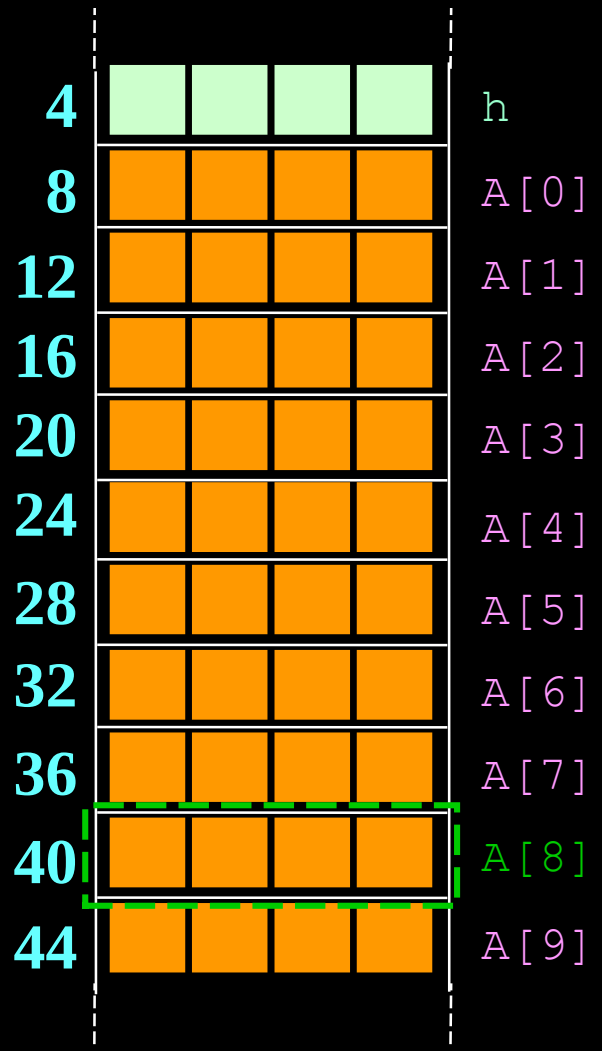
Código C:

```
A[8] = h + A[5];
```

Código MIPS:

```
lw $t0, 20($s3); *
add $t0, $s2, $t0
sw $t0, 32($s3); *
```





* real: palavra não byte

Continuamos a Conhecer o MIPS:

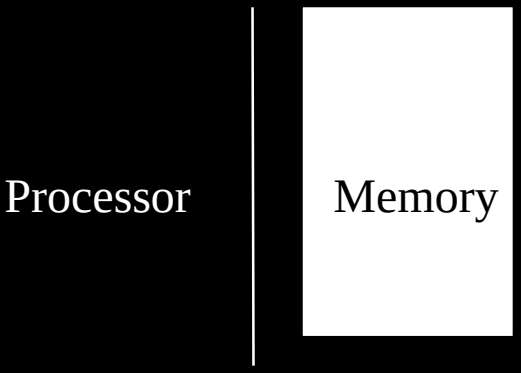
Princípio de Projeto: Um bom projeto demanda compromisso

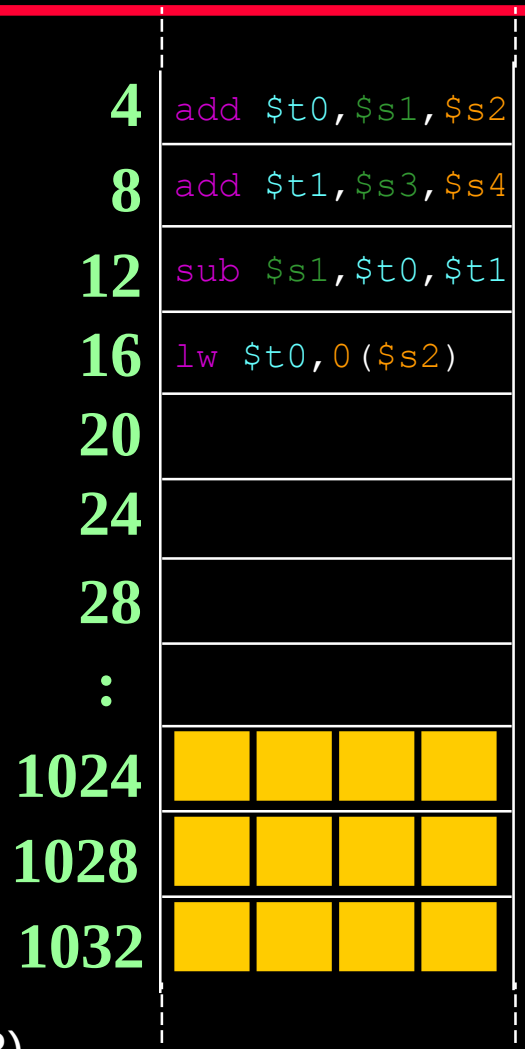
Ex: MIPS
- carrega palavras mas endereça bytes
- aritmética somente em registradores

Instrução		Resultado
add $s1, $s2, $s3	;	$s1 <– $s2 + $s3
sub $s1, $s2, $s3	;	$s1 <– $s2 – $s3
lw $s1, 100($s2)	;	$s1 <– Memory[$s2+100]
sw $s3, 100($s4)	;	Memory[$s4+100] <– $s3

Conceito de Programa Armazenado

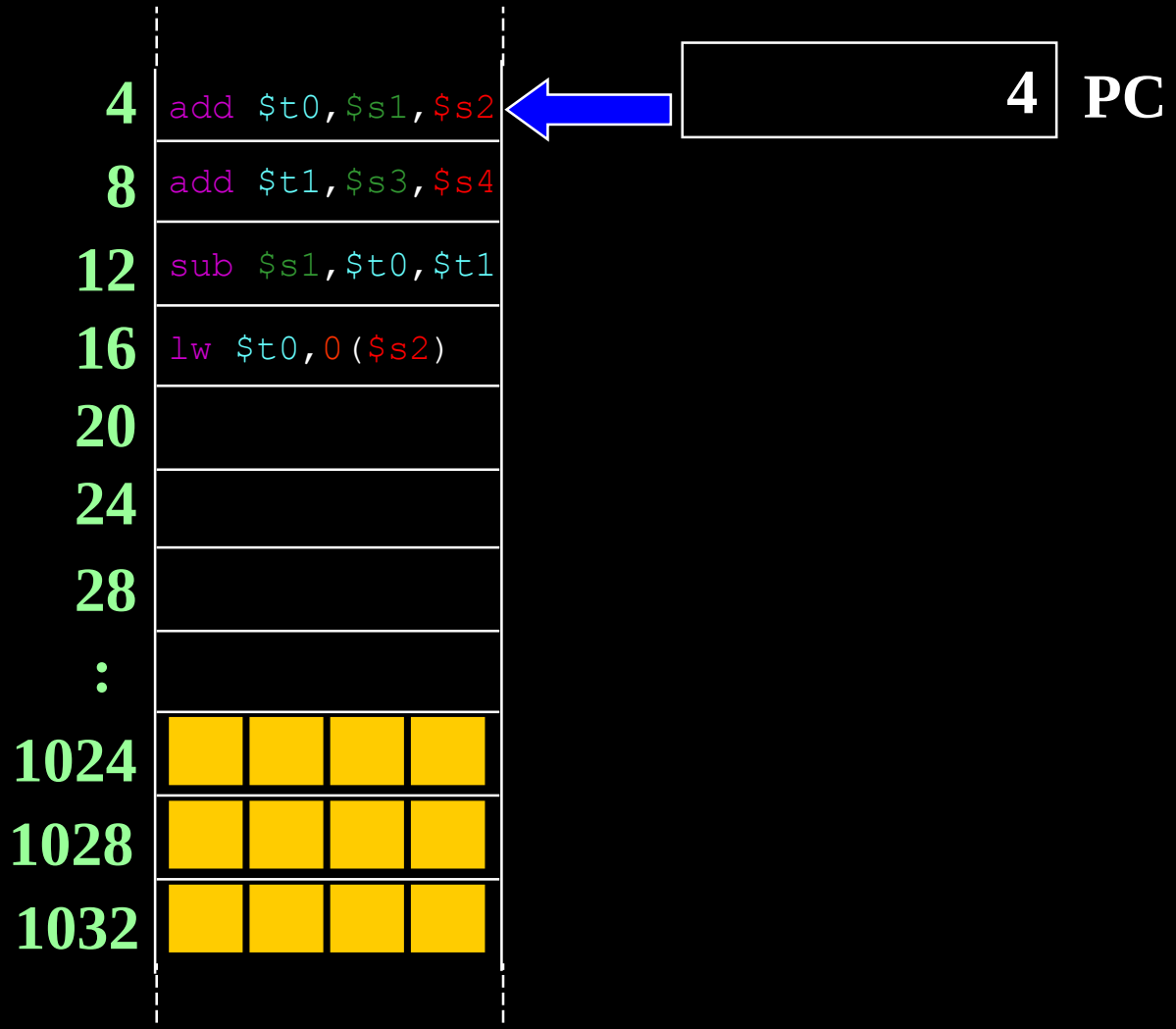
- **Instruções são bits**
- Programas são armazenados na memória



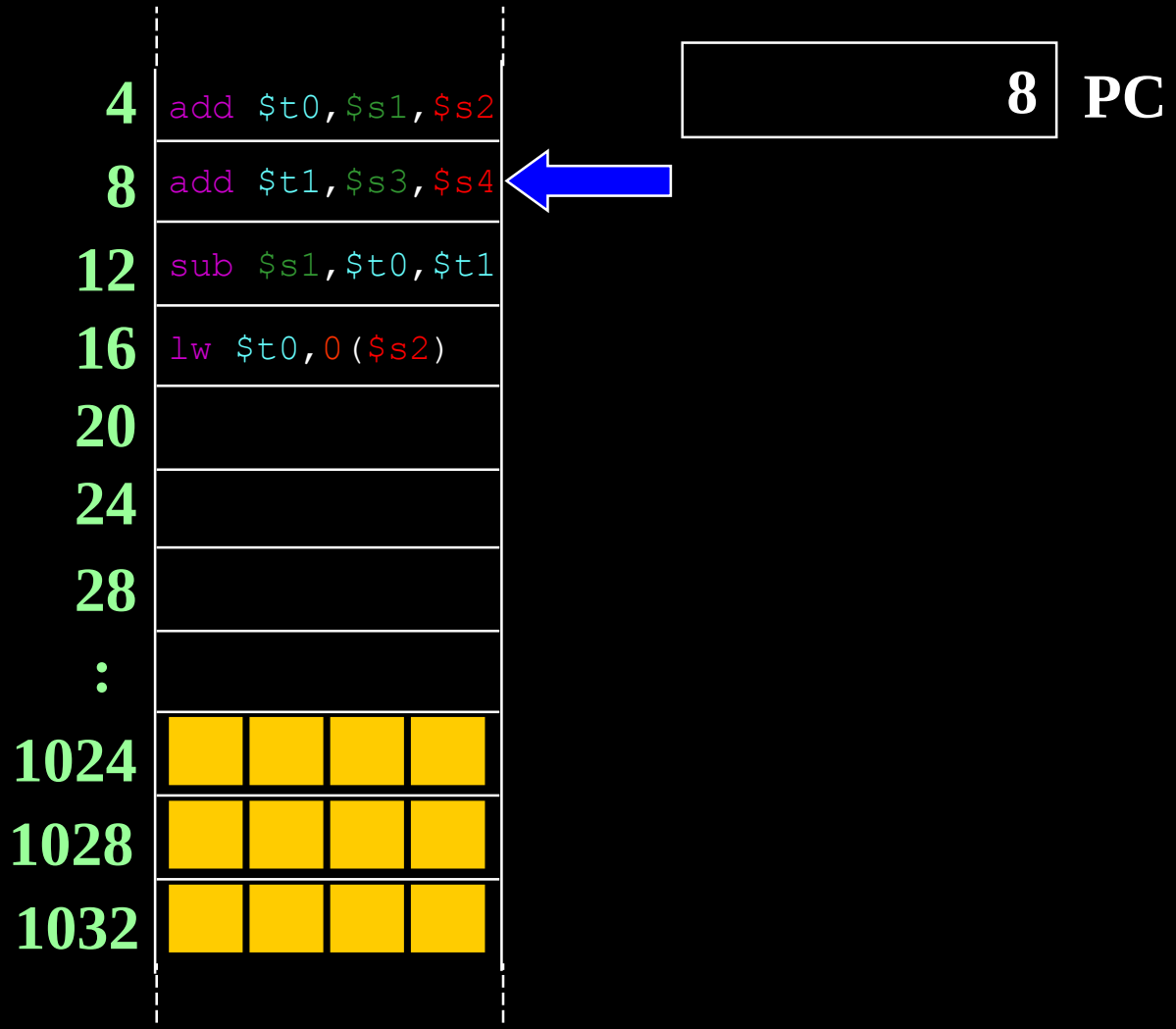


- Ciclo de busca & execução
  - Busca de instruções controlada por um registrador (contador de programa – PC)
  - Instruções são armazenadas e colocadas em um registrador especial (Registrador de instruções – IR)
  - Bits da instrução “controlam” as ações subseqüentes
  - Busca a próxima instrução e continua

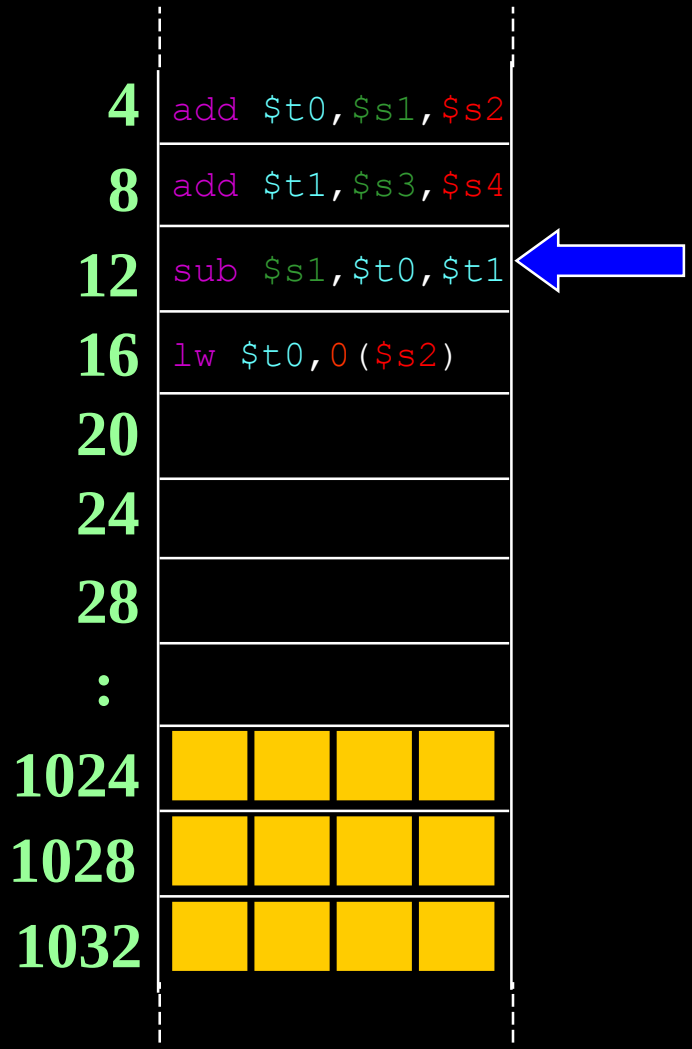
Controle


4
add $t0,$s1,$s2
8
add $t1,$s3,$s4
12
sub $s1,$t0,$t1
16
lw $t0,0($s2)
20
24
28
:
1024
1028
1032
4
PC

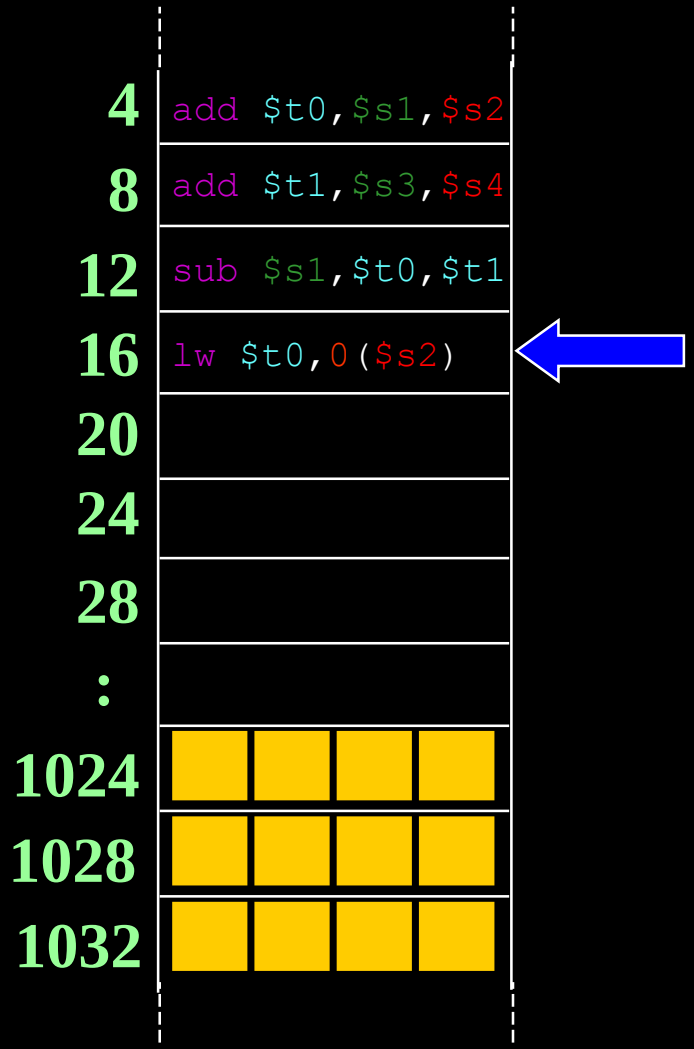
Controle


4
add $t0,$s1,$s2
8
add $t1,$s3,$s4
12
sub $s1,$t0,$t1
16
lw $t0,0($s2)
20
24
28
:
1024
1028
1032
8
PC

Controle


4
add $t0,$s1,$s2
8
add $t1,$s3,$s4
12
sub $s1,$t0,$t1
16
lw $t0,0($s2)
20
24
28
:
1024
1028
1032
12
PC

Controle



16 PC

MIPS – Desvios Condicionais

- Decisões dependentes da instrução
  - altera o fluxo de execução de instruções, ou seja, muda a próxima instrução a ser executada alterando o valor de PC
- Desvio condicional “branch”.

  Desvia se diferente:

```
bne rs, rt, Label
se(rs != rt)
   PC <- PC + [Label - PC]
        offset = (Label - PC) / 4
```

descrição	op	rs	rt	offset
bits por campo	6	5	5	16
tamanho da instrução	32 bits			

MIPS - Desvios

Desvia se igual:

```
beq rs, rt, Label
se(rs == rt)
    PC ← PC + [Label - PC]
    offset = (Label - PC) / 4
```

descrição	op	rs	rt	offset
bits por campo	6	5	5	16
tamanho da instrução	32 bits			

MIPS - Desvios

- Código C:

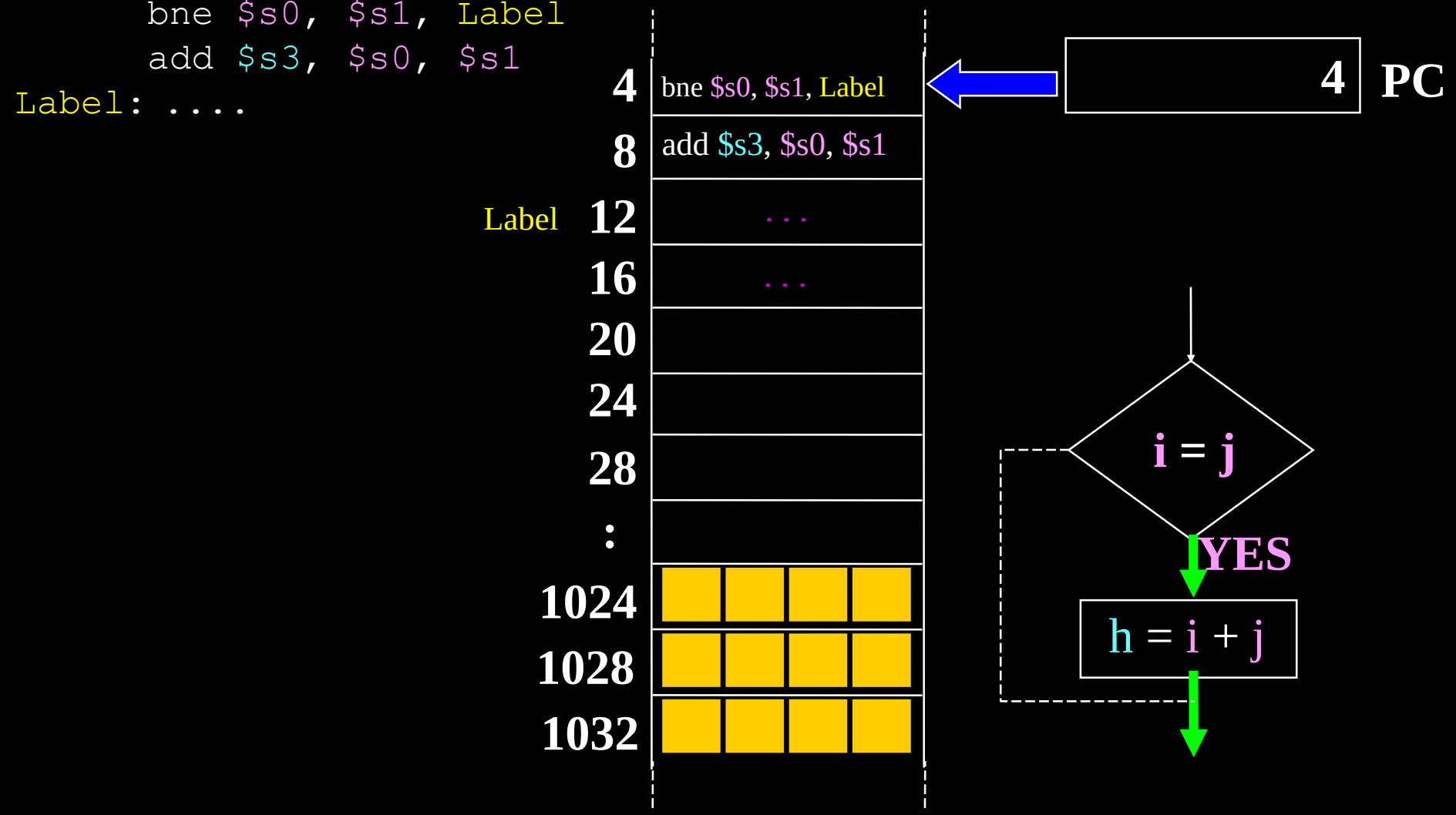
```
if (i==j)
    h = i + j;
```

- Código MIPS:

```
        bne $s0, $s1, segue
        add $s2, $s0, $s1
segue:  ....
```

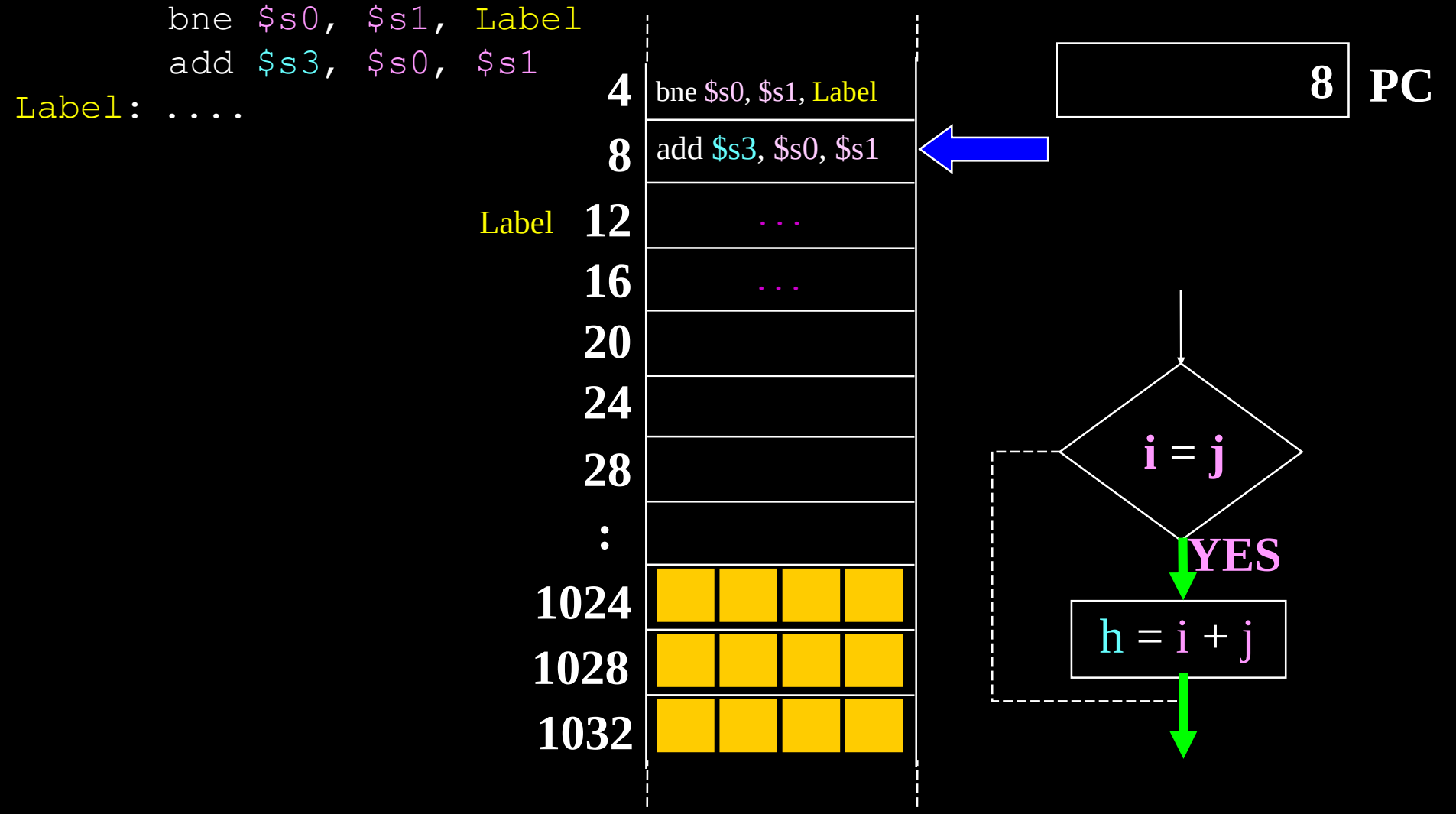
Controle

```
      bne $s0, $s1, Label
      add $s3, $s0, $s1
Label: ....
```



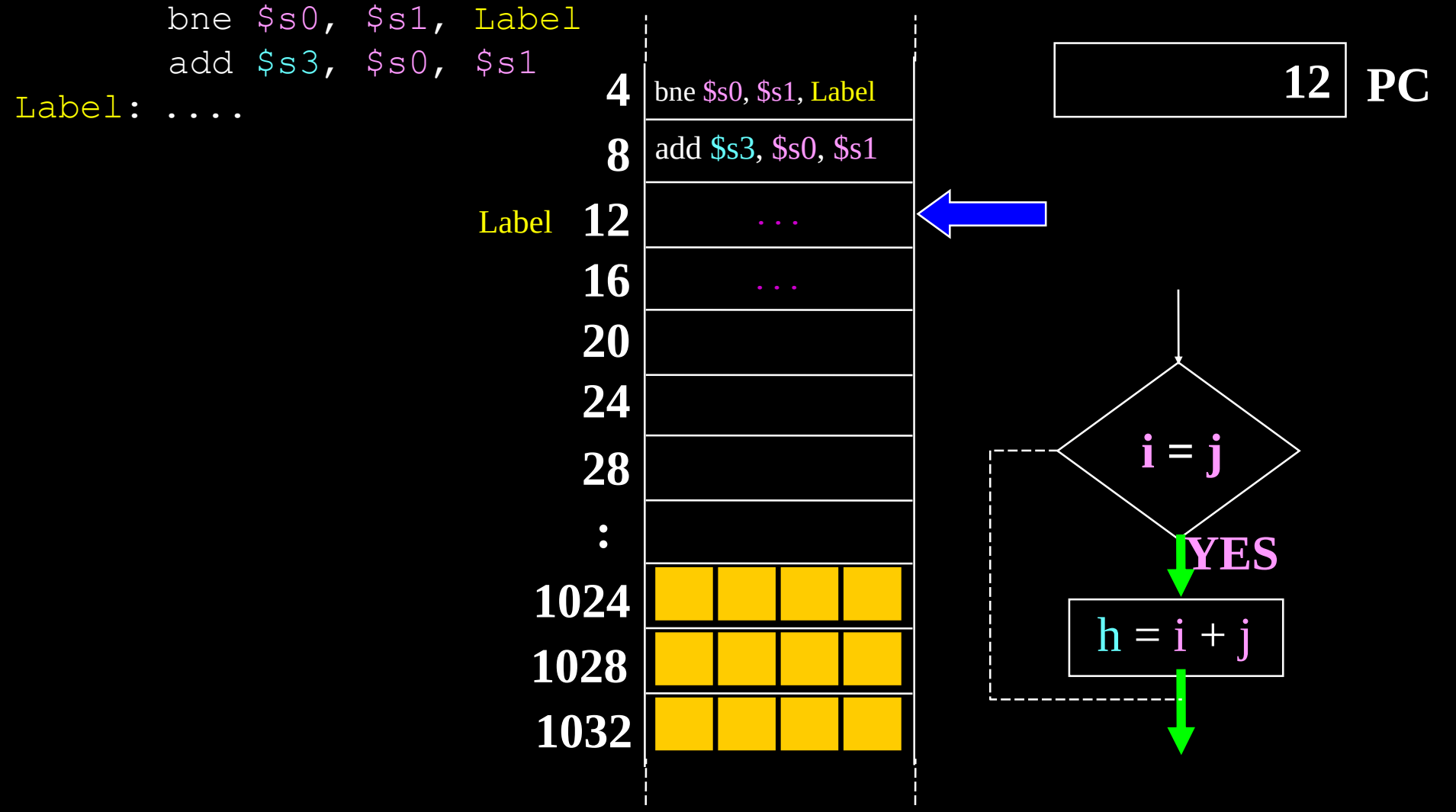
Controle

```
        bne $s0, $s1, Label
        add $s3, $s0, $s1
Label: ....
```



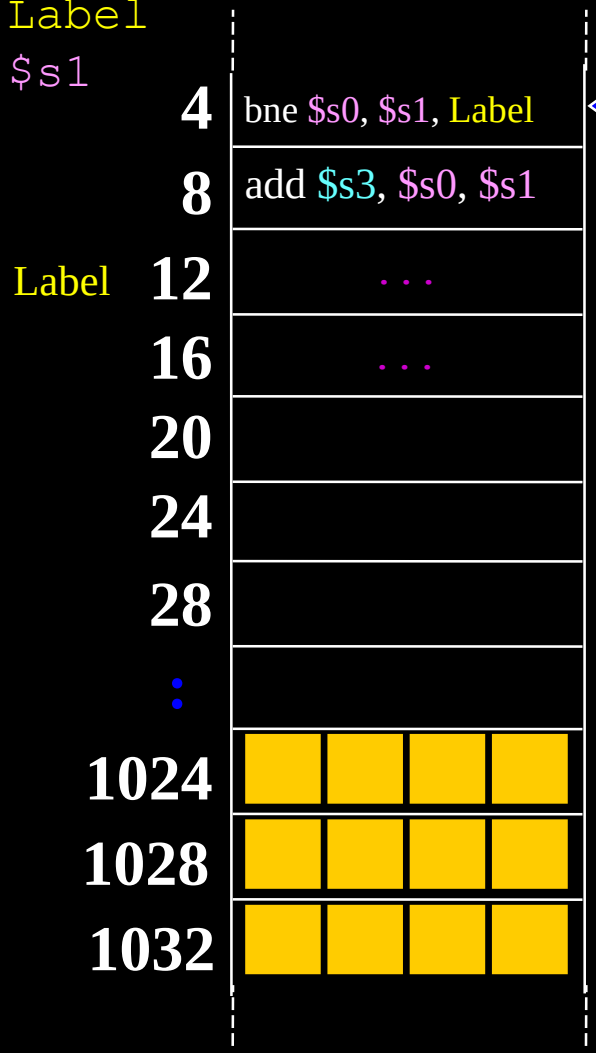
Controle

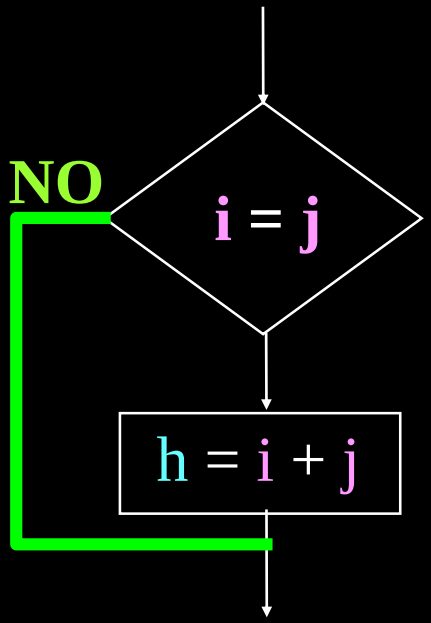
```
        bne $s0, $s1, Label
        add $s3, $s0, $s1
Label: ....
```



Controle

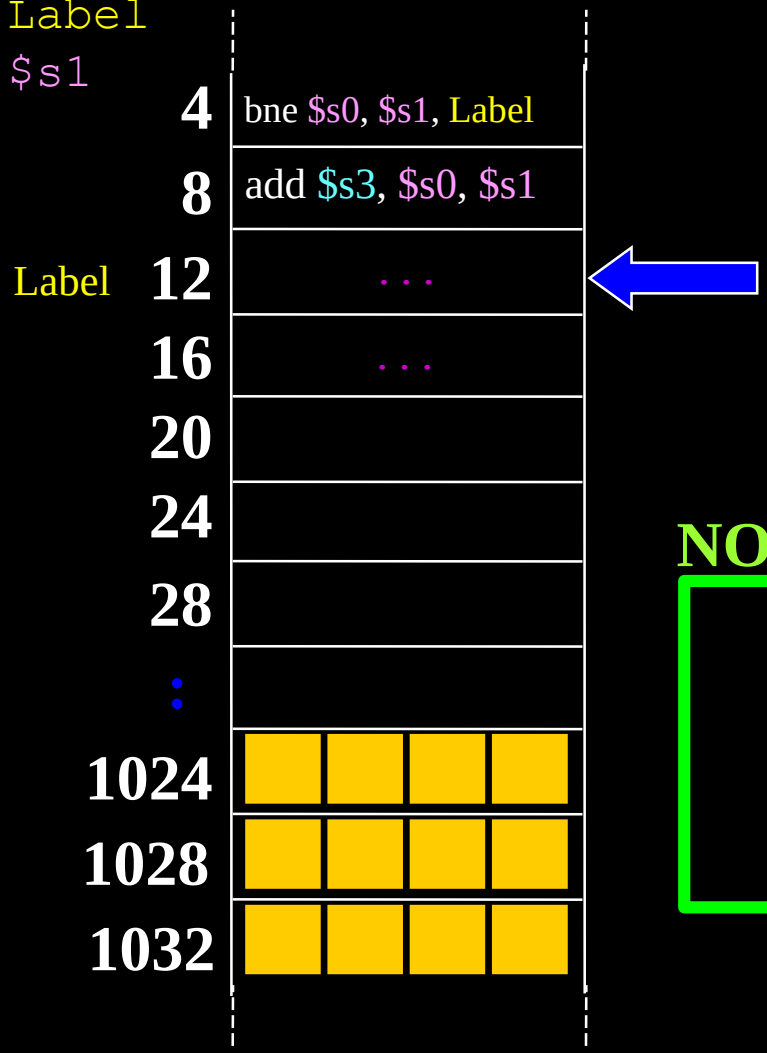
```
        bne $s0, $s1, Label
        add $s3, $s0, $s1
Label:  ....
```

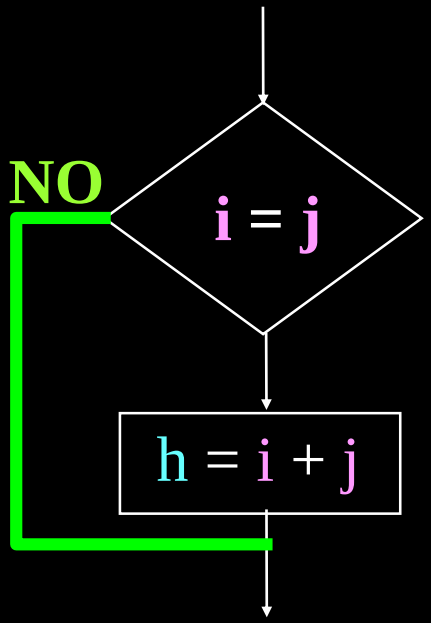




Controle

```
        bne $s0, $s1, Label
        add $s3, $s0, $s1
Label:  ....
```





MIPS - Desvios

- Instrução de desvio incondicional:

  `j label;` `PC <- PC[31-28]:label`

  O label é contado em palavras.

- Exemplo:

```
if (i != j)                    beq $s4, $s5, Lab1
    h=i+j;                     add $s3, $s4, $s5
else
    h=i-j;                     j Lab2
              Lab1:            sub $s3, $s4, $s5
              Lab2:            ....
```

descrição	op	offset
bits por campo	6	26
tamanho da instrução	32 bits	

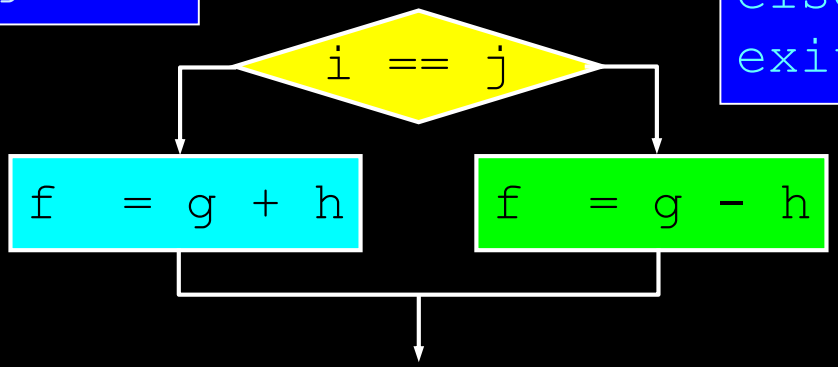
MIPS - Desvios

- Desvios condicionais (branches):
  - Baseados em relações de comparação

  Ex.: bne (branch on not equal), beq (branch on equal)
- Desvios incondicionais (jumps) : saltos no programa
- Modificam valor do PC (Program Counter)

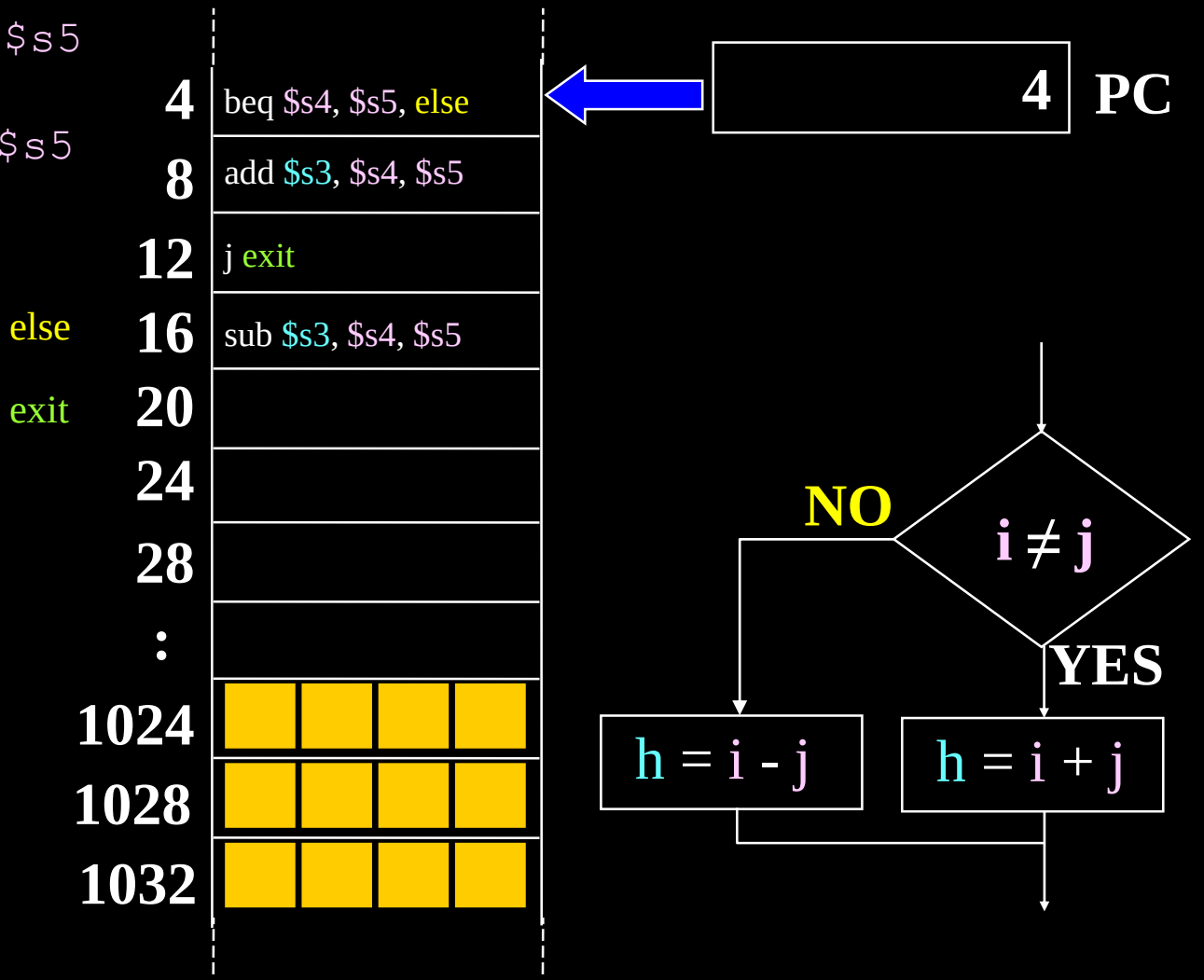
```
if (i == j)
    f = g + h;
else
    f = g - h;
```

```
        bne $19,$20,else
        add $16,$17,$18
        j   exit
else:   sub $16,$17,$18
exit:
```



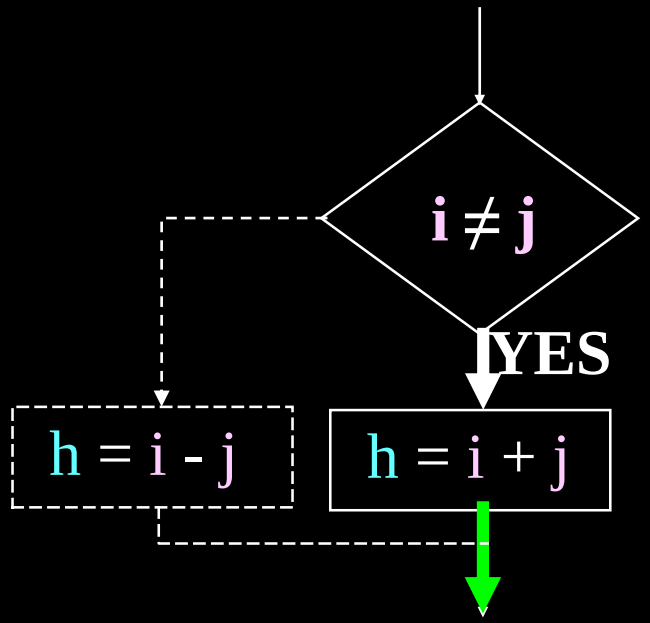
Controle

```
        beq $s4, $s5, else
        add $s3, $s4, $s5
        j exit
else:   sub $s3, $s4, $s5
exit:   ...
```



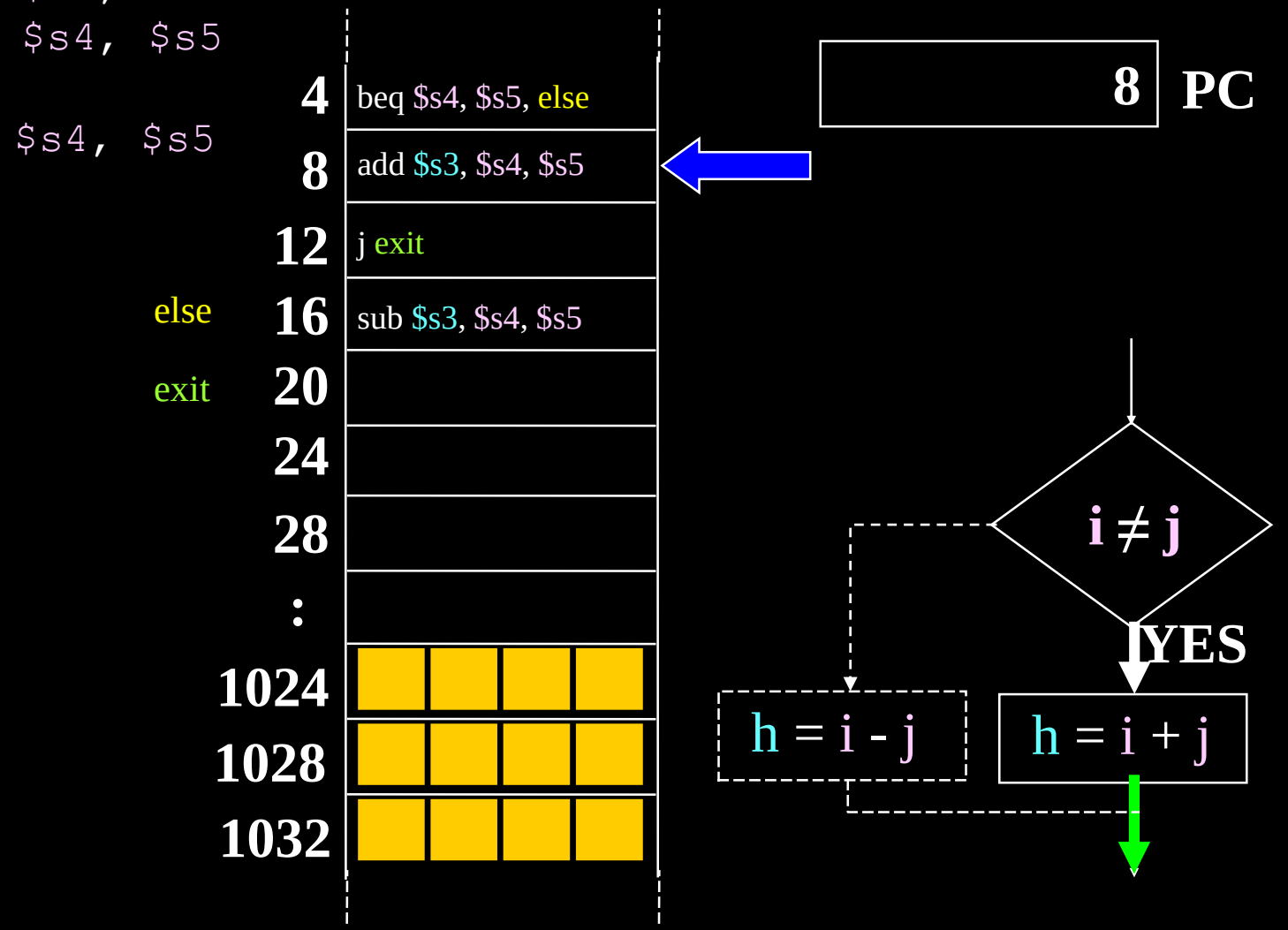
Controle

```
        beq $s4, $s5, else
        add $s3, $s4, $s5
        j exit
else:   sub $s3, $s4, $s5
exit:   ...
```



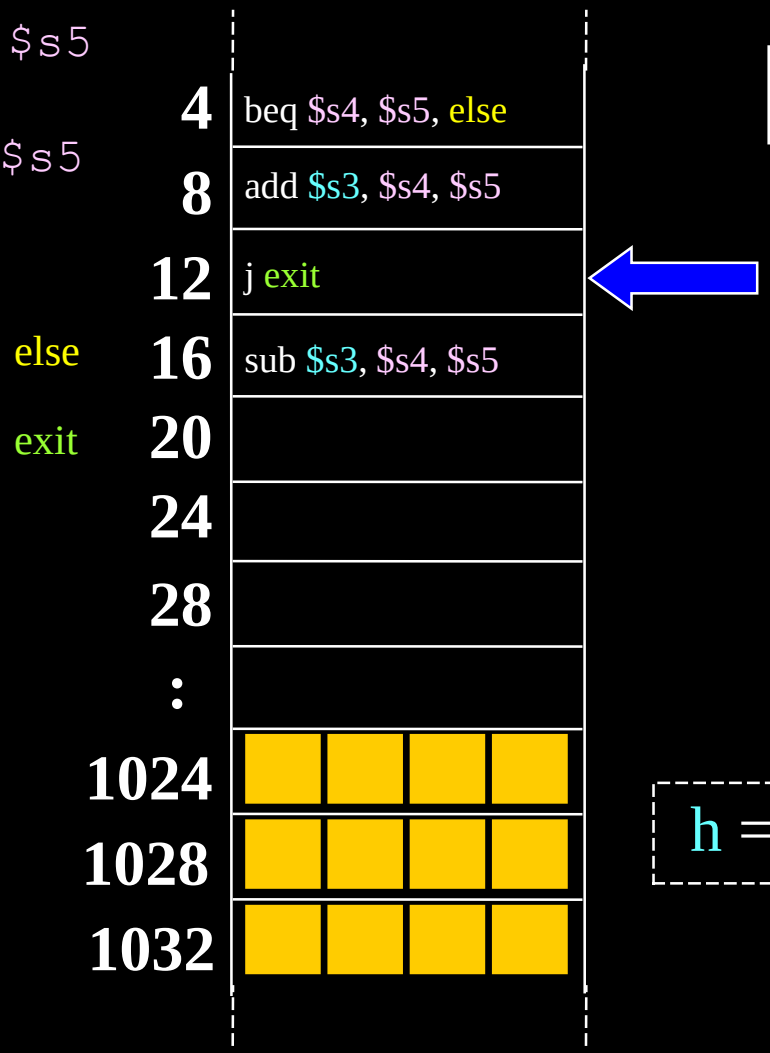
Controle

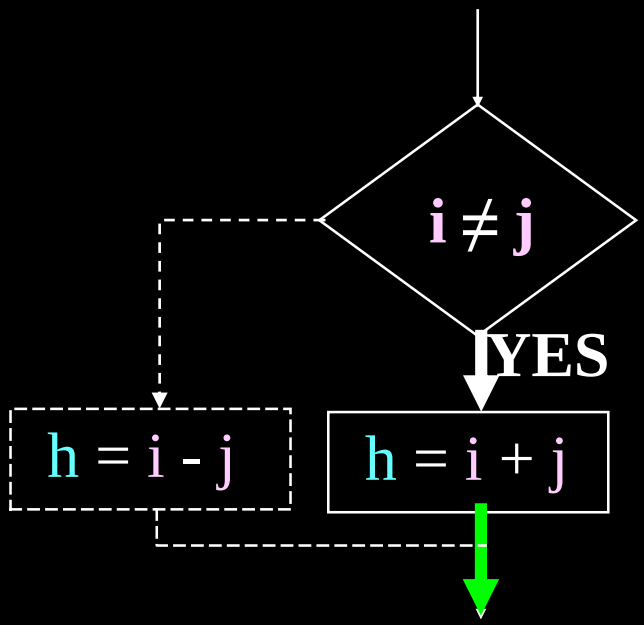
```
        beq $s4, $s5, else
        add $s3, $s4, $s5
        j exit
else:   sub $s3, $s4, $s5
exit:   ...
```



Controle

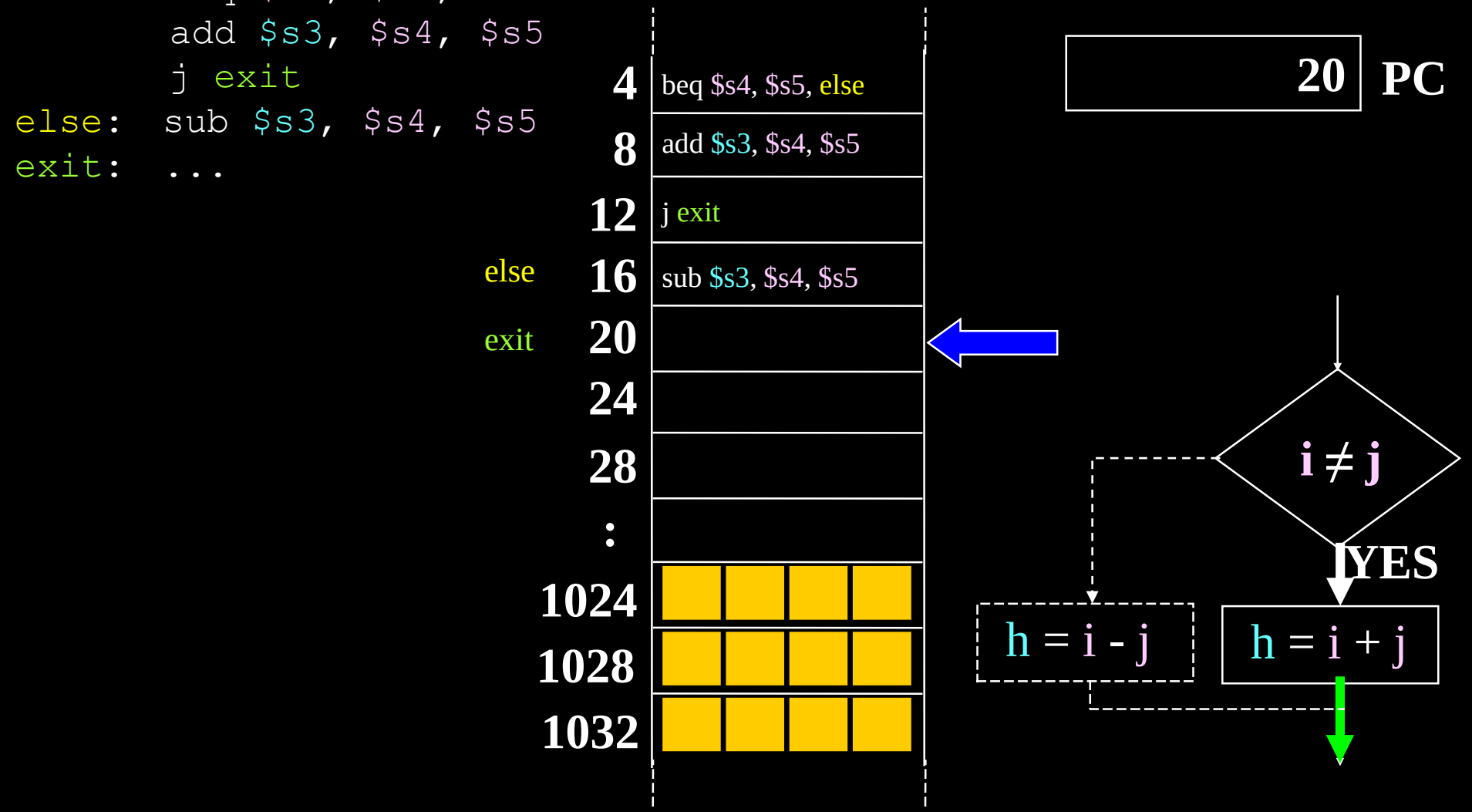
```
        beq $s4, $s5, else
        add $s3, $s4, $s5
        j exit
else:   sub $s3, $s4, $s5
exit:   ...
```





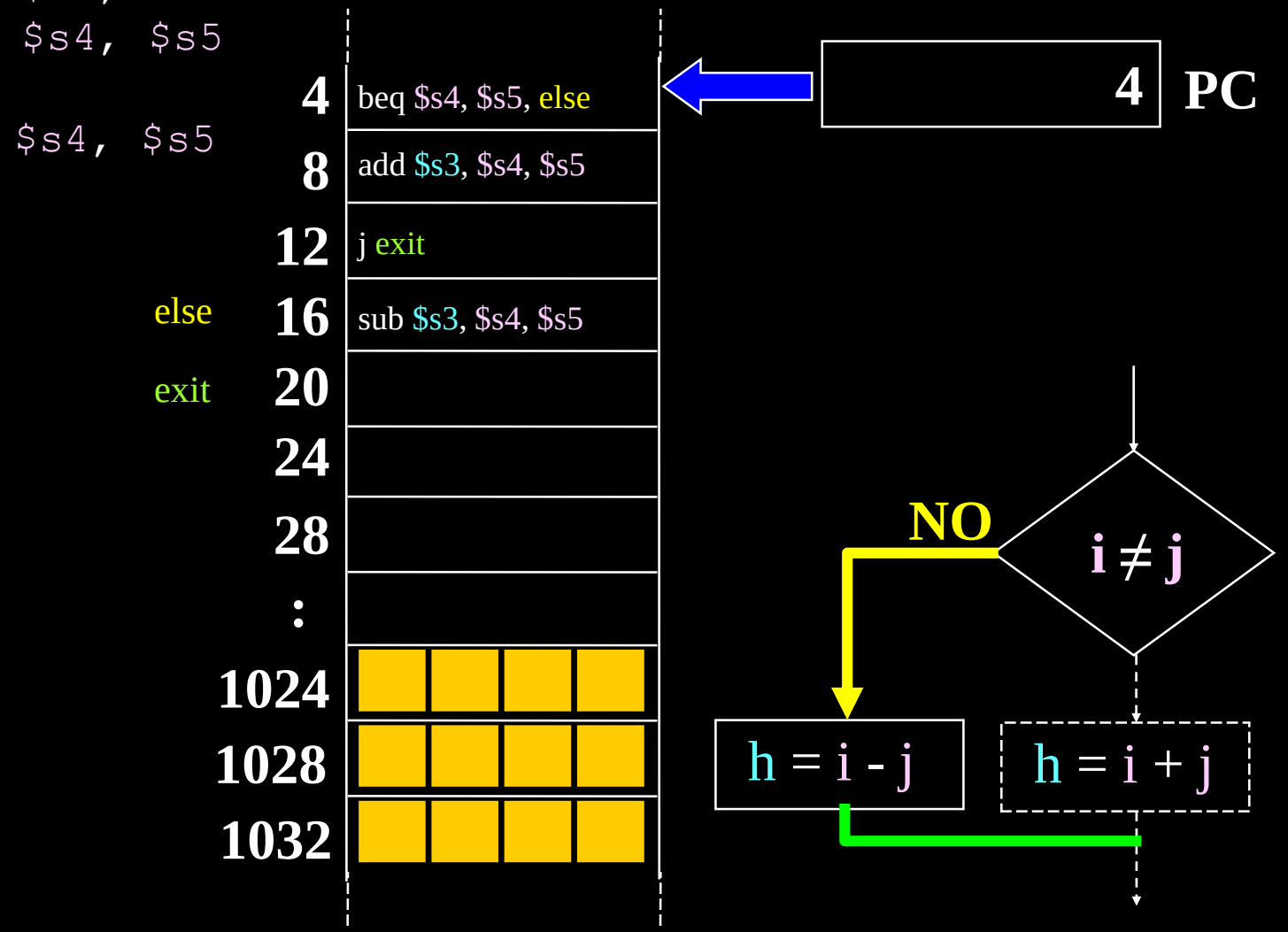
Controle

```
        beq $s4, $s5, else
        add $s3, $s4, $s5
        j exit
else:   sub $s3, $s4, $s5
exit:   ...
```



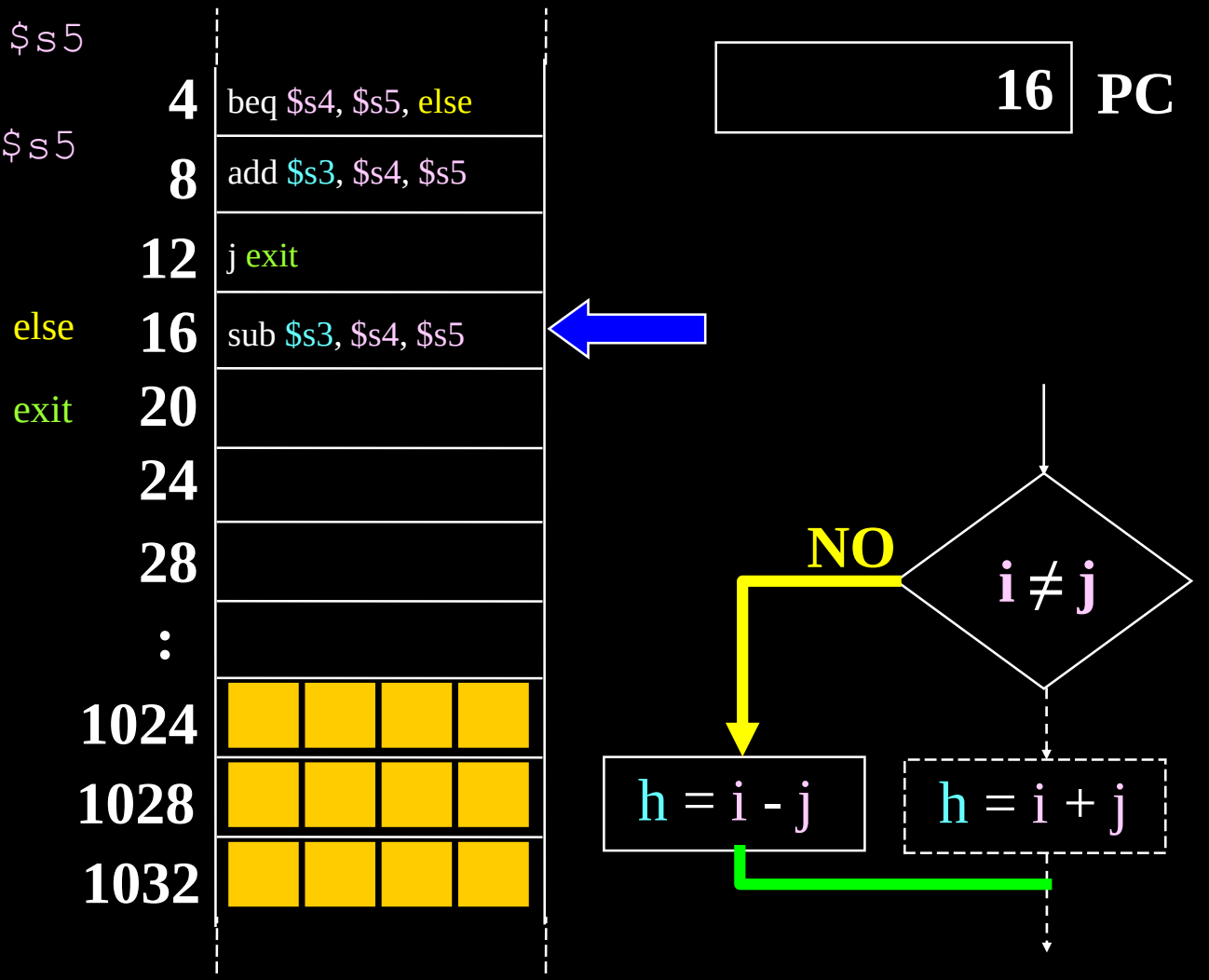
Controle

```
        beq $s4, $s5, else
        add $s3, $s4, $s5
        j exit
else:   sub $s3, $s4, $s5
exit:   ...
```



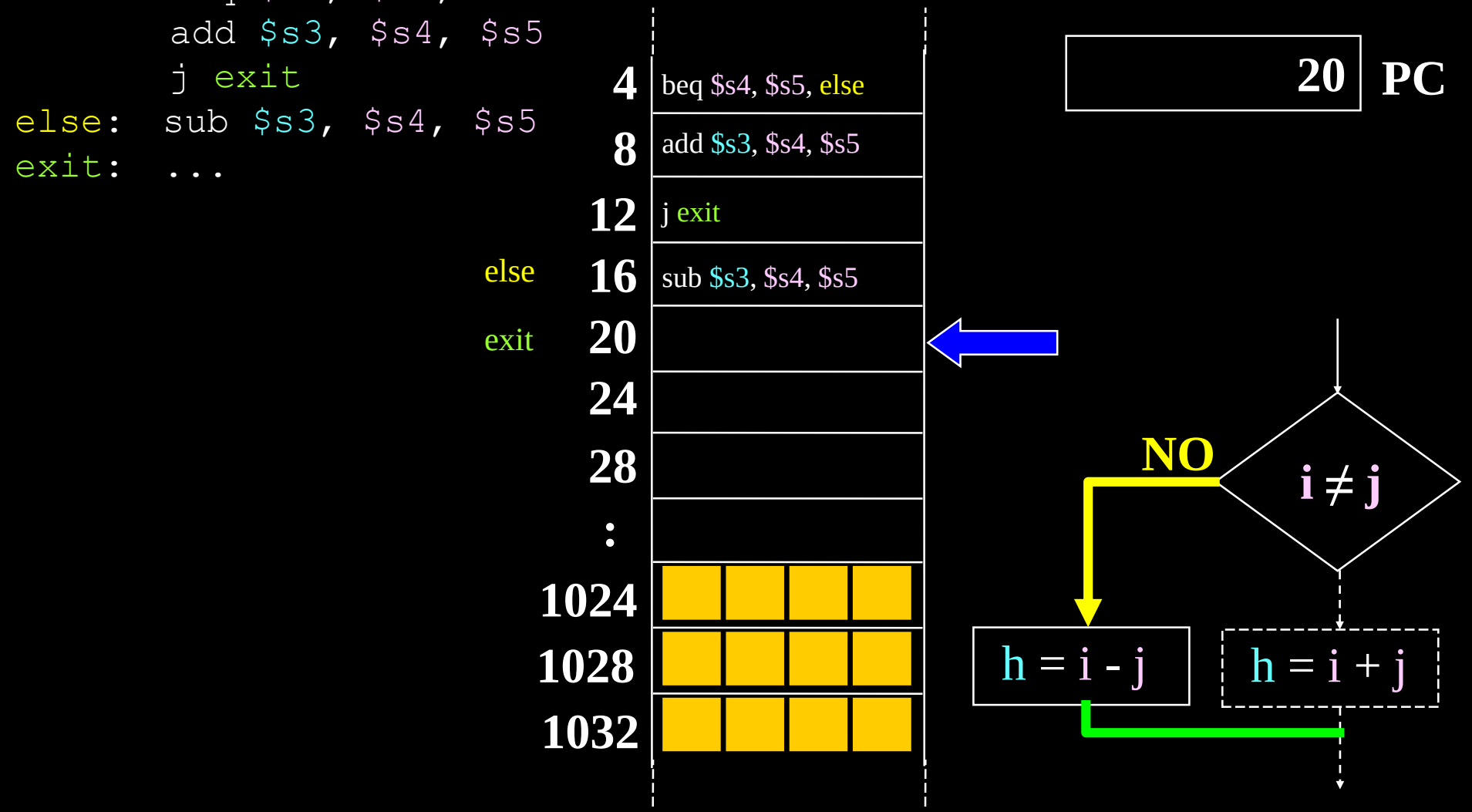
Controle

```
        beq $s4, $s5, else
        add $s3, $s4, $s5
        j exit
else:   sub $s3, $s4, $s5
exit:   ...
```



Controle

```
        beq $s4, $s5, else
        add $s3, $s4, $s5
        j exit
else:   sub $s3, $s4, $s5
exit:   ...
```

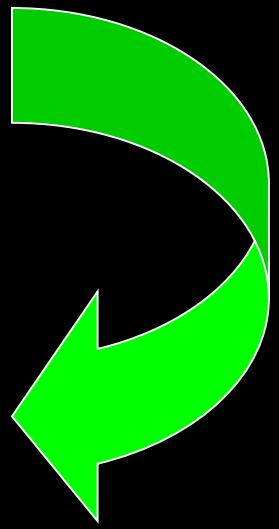


Desvios

- Temos: beq, bne;
- Como produzir Branch-if-less-than?
- Nova Instrução:

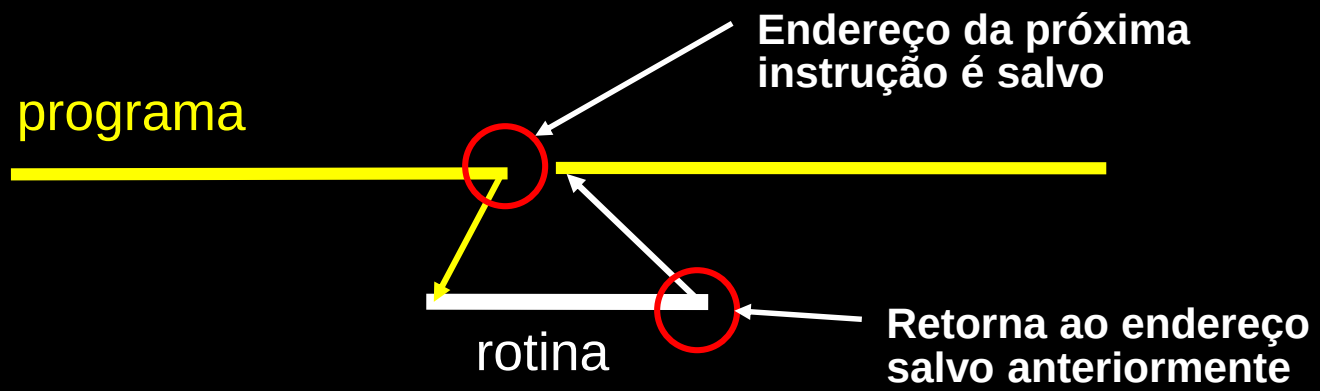
```
if  $s1 < $s2 then
 $t0 = 1
else
 $t0 = 0

 slt  $t0, $s1, $s2
```



Chamada de Procedimentos

- Caso especial de desvio (com retorno)
  - Uso da instrução jal (jump and link)
- Endereço do “chamador”
  - Armazenado em área especial (pilha) até o retorno
- Endereço do “chamado”
  - Alvo do desvio
- Empregado em funções, procedimentos, interrupções de hardware



Chamada de Procedimentos

- Instrução de chamada de procedimento:

```
jal   label   ; $ra ← PC + 4
              ; PC ← PC[28-31]:label
```

  O label é contado em palavras.

- Instrução de retorno da chamada de procedimento

```
jr   $ra     ; PC ← $ra
```

  Para procedimentos aninhados devemos salvar $ra na pilha.

descrição	op	offset
bits por campo	6	26
tamanho da instrução	32 bits	

Constantes

- Pequenas constantes são usadas freqüentemente

  A = 0; X = 5; B++; Y = Z + 3;

  Soluções?
  - criar registradores hard-wired para constantes (como $zero)

Princípio de Projeto: torne o caso comum mais rápido

```
MIPS - Instruções:
        addi $t0, $t0, 4
        slti $t0, $s0, 10
        andi $a1, $a1, 6
        ori  $t0, $t0, 15
```

descrição	op	rs	rt	offset
bits por campo	6	5	5	16
tamanho da instrução	32 bits			

Constantes Grandes

- Gostaríamos de ser capazes de carregar uma constante de 32 bit em um registrador.

  Por exemplo: carregar 700.000 no registrador $t0

  700.000 = 000AAE60h
- Deve ser usada uma nova instrução:

  "load upper immediate" - **lui**

  Esta instrução carrega a parte alta do registrador (16 bits mais significativos) com a constante especificada na instrução. A parte baixa é carregada com zero:

```
lui $reg,constante ; $reg[31:16] <- constante
lui $t0, 000Ah   ; $t0 <- 000Ah
```

$t0 ← 00000000000010100000000000000000

Constantes Grandes

- Então concatenamos os bits de baixa ordem na direita usando:

```
ori  $t0, $t0, 0F0Fh
```

ou então

```
addi $t0, $t0, 0F0Fh
```

Mostrando detalhadamente:

```
lui $t0, 000Ah
ori $t0, $t0, AE60h
```

	0000000000001010	0000000000000000
	0000000000000000	1010111001100000
resultado →	0000000000001010	1010111001100000

Assim, para carregarmos uma constante de 32 bits devemos utilizar uma seqüência de 2 instruções

Formato Básico de Instruções

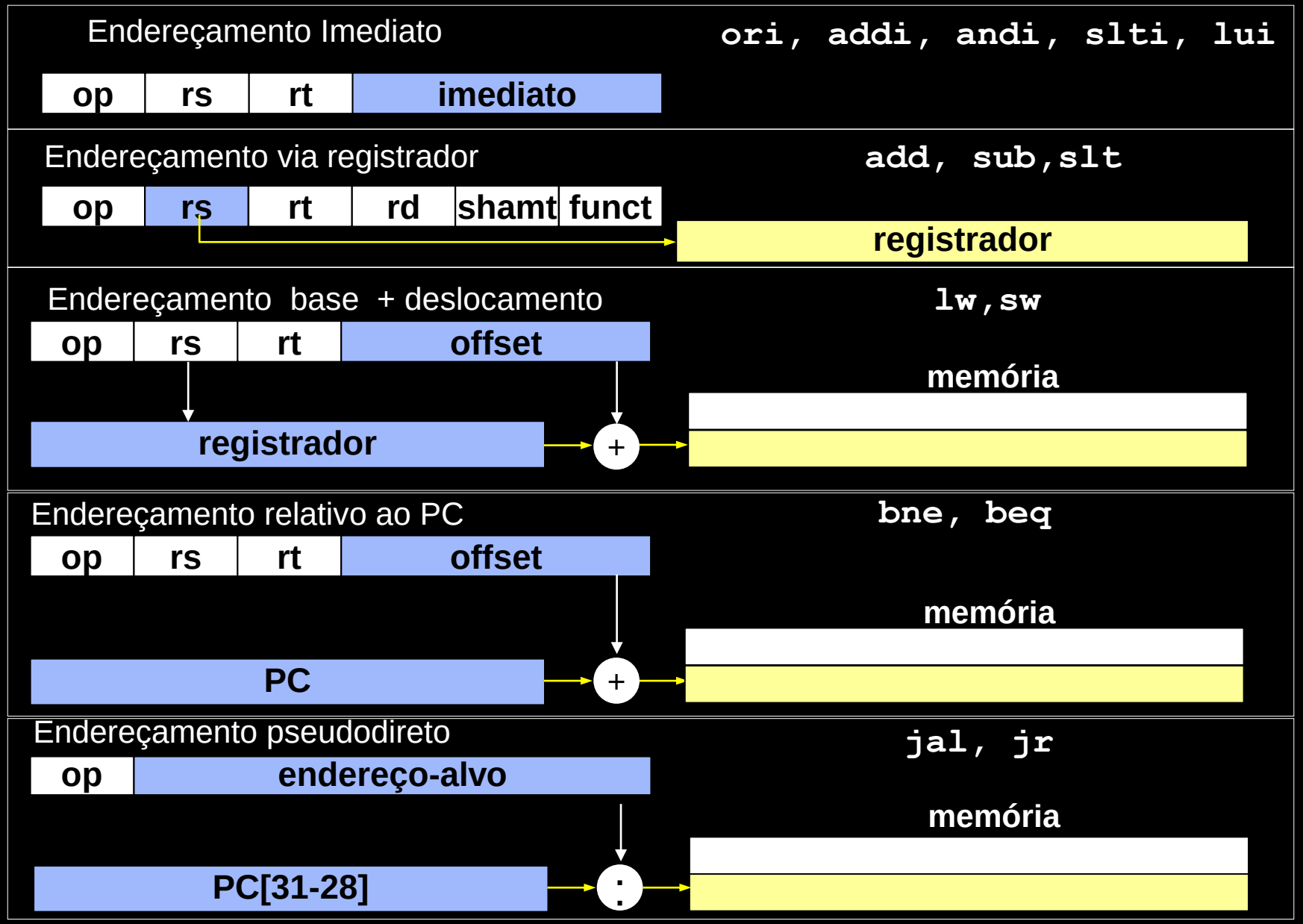
- Toda instrução é formada por campos

  Como vimos os 32 bits de uma instrução MIPS são separados em campos específicos.

Descrição	Tipo	6	5	5	5	5	6
Aritmética	R	op	rs	rt	rd	shamt	funct
Transferência	I	op	rs	rt	offset		
Jump	J	op	Endereço - alvo				

Modos de Endereçamento

- Número de modos - Complexidade
  - Vantagem: Flexibilidade
  - Desvantagem: baixo desempenho, custo de decodificação
- Modos de endereçamento do MIPS:
  - Por Registrador: operando em registrador
  - Base-Deslocamento: operando na memória
  - Imediato: operando é constante no corpo da instrução
  - Relativo ao PC: constante na instrução + PC
  - Pseudodireto: concatenação de constante com PC


Endereçamento Imediato
ori, addi, andi, slti, lui
op
rs
rt
imediato
Endereçamento via registrador
add, sub,slt
op
rs
rt
rd
shamt
funct
registrador
Endereçamento base + deslocamento
lw,sw
op
rs
rt
offset
memória
registrador
+
Endereçamento relativo ao PC
bne, beq
op
rs
rt
offset
memória
PC
+
Endereçamento pseudodireto
jal, jr
op
endereço-alvo
memória
PC[31-28]
:

Conjunto de Instruções MIPS

ADD $rd, $rs, $rt	; $rd ← $rs + $rt	00h | rs | rt | rd | 00h | 20h
SUB $rd, $rs, $rt	; $rd ← $rs - $rt	00h | rs | rt | rd | 00h | 22h
SLT $rd, $rs, $rt	; se($rs < $rt) $rd ← 1 ; senão $rd ← 0	00h | rs | rt | rd | 00h | 2Ah
BNE $rs, $rt, offset	; se($rs != $rt) ; PC ←PC+[offset – PC]	05h | rs | rt | offset
BEQ $rs, $rt, offset	; se($rs == $rt) ; PC ←PC+[offset – PC]	04h | rs | rt | offset
SLTI $rd, $rs, offset	; se($rs < $offset) $rd ← 1 ; senão $rd ← 0	0Ch | rs | rd | offset
LW $rt, offset($rs)	; $rt ← ($rs + offset)	23h | rs | rt | offset
SW $rt, offset($rs)	; ($rs + offset) ← $rt	2Bh | rs | rt | offset

Conjunto de Instruções MIPS

ADDI $rt, $rs, offset	; $rt ← $rs + offset	08h	rs	rt	offset
LUI $rt, offset	; $rt ← offset	0Fh	0	rt	offset
ORI $rt, $rs, offset	; $rt ← $rs OR offset	0Dh	rs	rt	offset
ANDI $rt, $rs, offset	; $rt ← $rs AND offset	0Ch	rs	rt	offset

J offset	; PC ← PC[31-28]:offset	02h	offset
JAL offset	; $ra ← PC ; PC ← PC[31-28]:offset	03h	offset

Legenda:

Aritméticas **Desvios Condicionais** **Carga/Armazenamento**

Constantes **Desvio Incondicionais**

Questões!!!

Classe de Instrução

vs Formato de Instrução

vs Modo de Endereçamento

Ling. Assembly X Ling. de Máquina

- Assembly provê representação simbólica conveniente
  - mais fácil que escrever números
- Linguagem de Máquina é a mais real
- Assembly pode prover 'pseudoinstruções'
  - “move $t0, $t1” existe somente em Assembly

    Pode ser implementado usando “add $t0,$t1,$zero”

Alternativas de Arquiteturas

- Alternativas de projeto:
  - prover operações mais potentes
  - reduzir o número de instruções executadas
  - Pode ter um tempo de ciclo lento e/ou um maior CPI
- As vezes referido como “RISC X CISC”
  - Em geral novos conjuntos de instrução depois de 1982 são RISC: ARM, RISC-V

Resumo

- A complexidade das instruções é somente uma das questões envolvidas no projeto
  - Pequena quantidade de instruções vs. alta CPI / freqüência de clock baixa
- Arquitetura do conjunto de instruções
  - Abstração verdadeiramente importante na implementação de um processador

Resumo

Projeto de uma arquitetura deve seguir ao menos 4 princípios básicos:

- **Simplicidade favorece a regularidade**
  - Busca de componentes mais simples no projeto
  - Desenvolvimento bottom-up
- **Menor é mais rápido**
  - Número de registradores impacta o ciclo de relógio
- **Um bom projeto implica em compromisso**
  - Decisões são baseadas em prejuízos x ganhos
- **Fazer o caso comum mais rápido**
  - Otimizando o caso genérico, como o uso de certas constantes, melhora desempenho geral

Apêndice A

```
#o valor default para o segmento de
#dados é 0x10010000 = 268500992
#(High=0x1001=4097 and Low=0x0000=0)

.data
a:          .word 36, 20, 27, 15, 1, 62, 41
n:          .word 7
max:        .word 0

.text
#O programa começa sua execução no rótulo "main"
main:
ori $8, $0, 0         # i está em $8
ori $16, $0, 0        # max está em $16
           lui $18, 4097
lw $17, 28($18)       # n está em $s1

m1:         slt $18, $8, $17
           beq $18, $0, m3     # se i >=n estão quit
           ori $18, $0, 4
mul $9, $8, $18      # multiplica i
lui $18, 4097
           add $18, $18, $9    # add 4.i ao endereço base de
           lw $10, 0($18)      #           a[i] em $10
           slt $18, $16, $10   # verifica se a[i] é maior que o atual
beq $18, $0, m2      # pula a parte"then" se a[i] <= max
add $16, $0, $10     # parte "then": max = a[i]
m2:         addi $8, $8, 1      # i++
j m1
m3:         nop              ....
```

- Estrutura básica de um programa para rodar no SPIM

Apêndice A

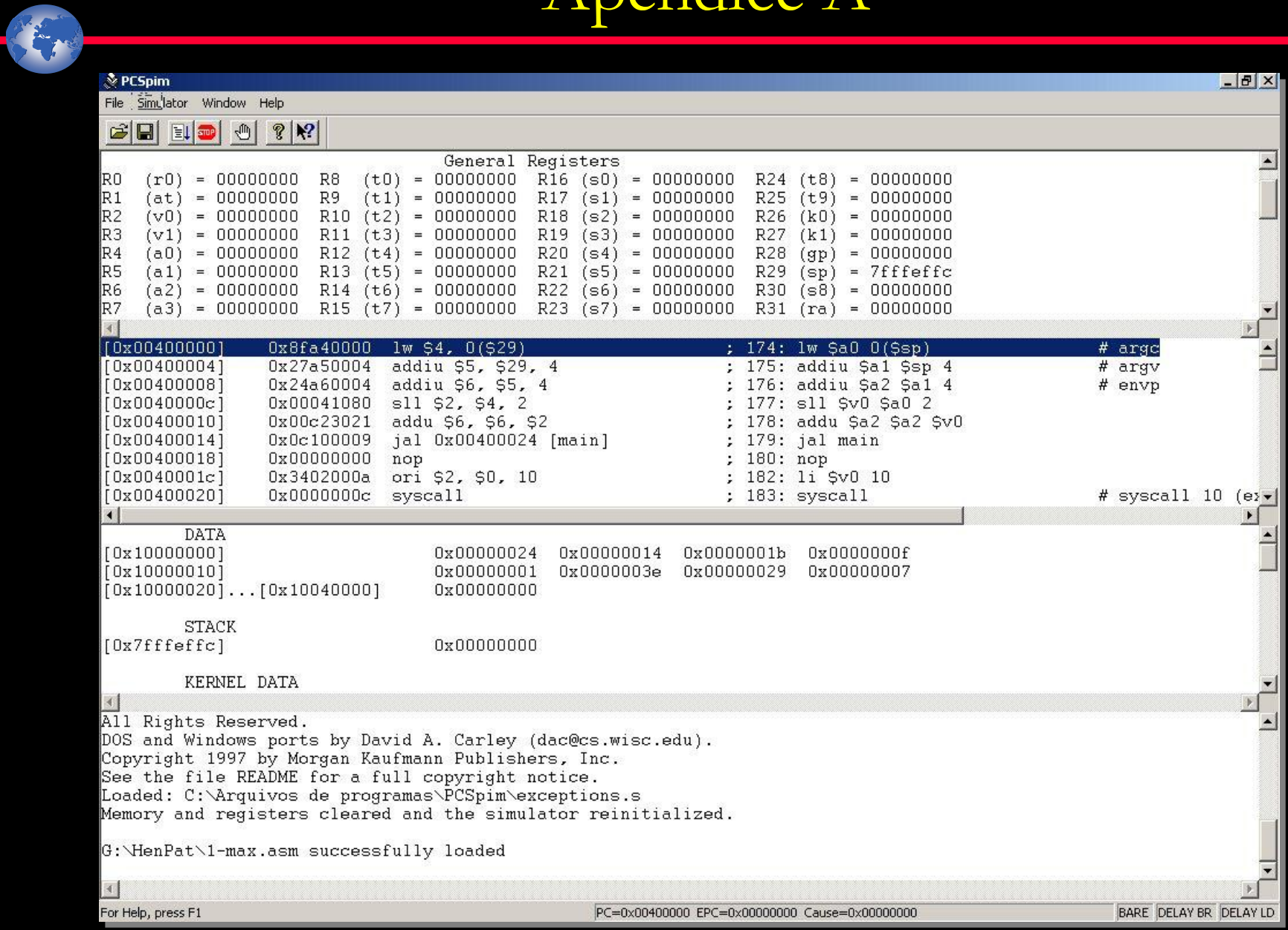
```
#o valor default para o segmento de
#dados é 0x10010000 = 268500992
#(High=0x1001=4097 and Low=0x0000=0)

          .data
a:        .word 36, 20, 27, 15, 1, 62, 41
n:        .word 7
max:      .word 0

          .text    #400000000h
#O programa começa sua execução no rótulo "main"
main:
          ori $8, $0, 0        # i está em $8
          ori $16, $0, 0       # max está em $16
          lui $18, 4097
          lw $17, 28($18)      # n está em $s1

m1:       slt $18, $8, $17
          beq $18, $0, m3    # se i >=n estão quit
          ori $18, $0, 4
          mul $9, $8, $18    # multiplica i
m2:       lui $18, 4097
          add $18, $18, $9   # add 4.i ao endereço base de
          lw $10, 0($18)     #           a[i] em $10
          slt $18, $16, $10  # verifica se a[i] é maior que o atual
          beq $18, $0, m2    # pula a parte"then" se a[i] <= max
          add $16, $0, $10   # parte "then": max = a[i]
          addi $8, $8, 1     # i++
          j m1
m3:       nop           ....
```

Apêndice A


PCSpim
File Simulator Window Help
General Registers
R0 (r0) = 00000000 R8 (t0) = 00000000 R16 (s0) = 00000000 R24 (t8) = 00000000
R1 (at) = 00000000 R9 (t1) = 00000000 R17 (s1) = 00000000 R25 (t9) = 00000000
R2 (v0) = 00000000 R10 (t2) = 00000000 R18 (s2) = 00000000 R26 (k0) = 00000000
R3 (v1) = 00000000 R11 (t3) = 00000000 R19 (s3) = 00000000 R27 (k1) = 00000000
R4 (a0) = 00000000 R12 (t4) = 00000000 R20 (s4) = 00000000 R28 (gp) = 00000000
R5 (a1) = 00000000 R13 (t5) = 00000000 R21 (s5) = 00000000 R29 (sp) = 7fffeffc
R6 (a2) = 00000000 R14 (t6) = 00000000 R22 (s6) = 00000000 R30 (s8) = 00000000
R7 (a3) = 00000000 R15 (t7) = 00000000 R23 (s7) = 00000000 R31 (ra) = 00000000
[0x00400000] 0x8fa40000 lw $4, 0($29) ; 174: lw $a0 0($sp) # argc
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 175: addiu $a1 $sp 4 # argv
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 176: addiu $a2 $a1 4 # envp
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 177: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 178: addu $a2 $a2 $v0
[0x00400014] 0x0c100009 jal 0x00400024 [main] ; 179: jal main
[0x00400018] 0x00000000 nop ; 180: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 182: li $v0 10
[0x00400020] 0x0000000c syscall ; 183: syscall # syscall 10 (ex
DATA
[0x10000000] 0x00000024 0x00000014 0x0000001b 0x0000000f
[0x10000010] 0x00000001 0x0000003e 0x00000029 0x00000007
[0x10000020]...[0x10040000] 0x00000000
STACK
[0x7fffeffc] 0x00000000
KERNEL DATA
All Rights Reserved.
DOS and Windows ports by David A. Carley (dac@cs.wisc.edu).
Copyright 1997 by Morgan Kaufmann Publishers, Inc.
See the file README for a full copyright notice.
Loaded: C:\Arquivos de programas\PCSpim\exceptions.s
Memory and registers cleared and the simulator reinitialized.
G:\HenPat\1-max.asm successfully loaded
For Help, press F1
PC=0x00400000 EPC=0x00000000 Cause=0x00000000
BARE DELAY BR DELAY LD