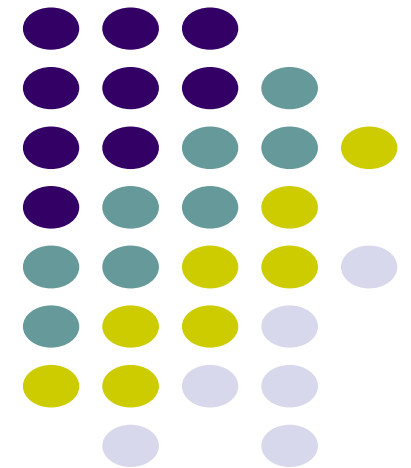
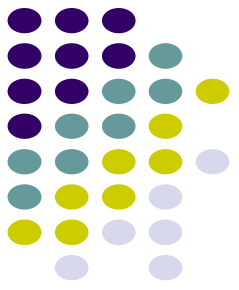


# Arquitetura e Organização de Computadores

## Instruções: a linguagem do computador Parte I

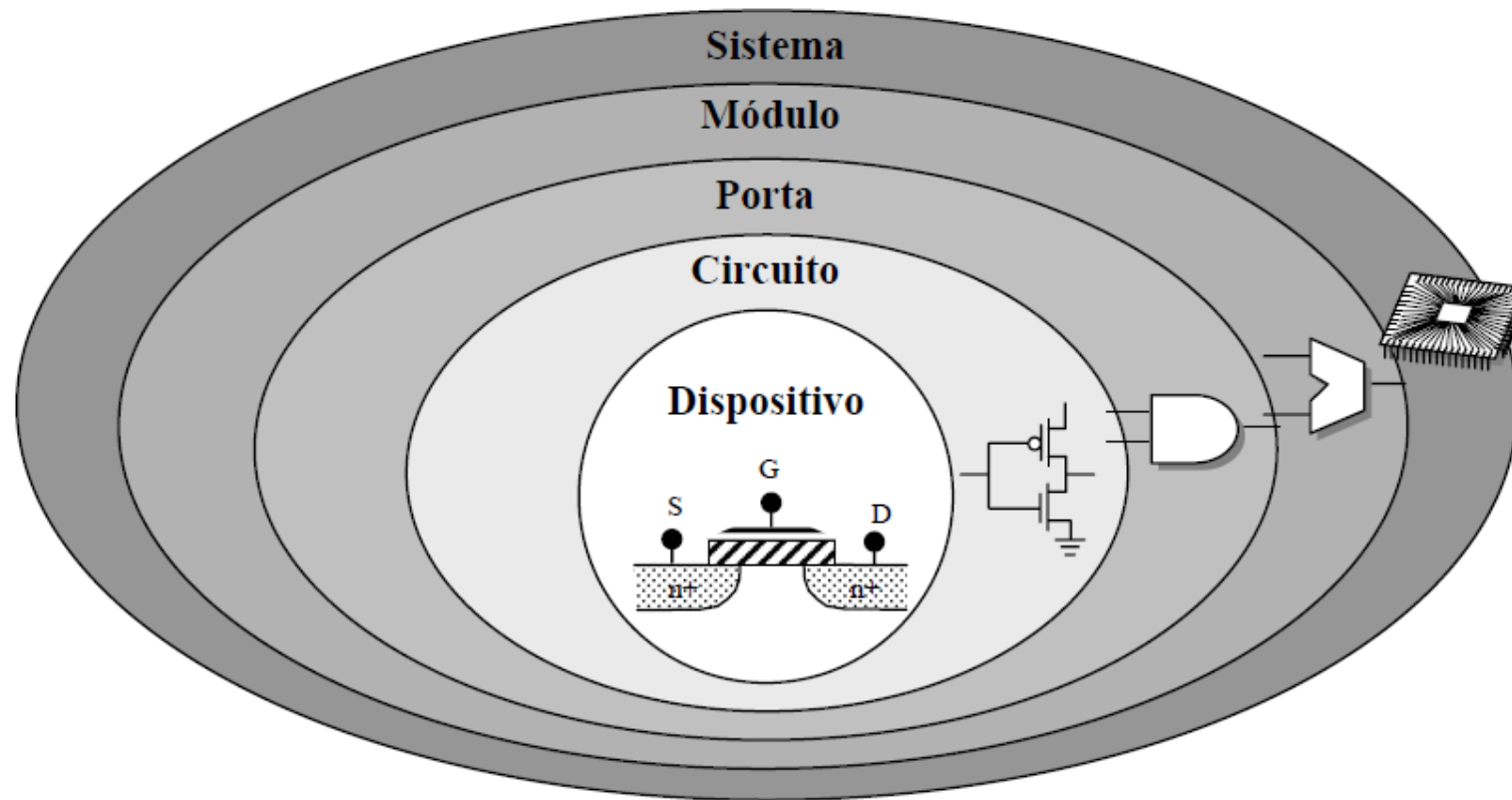
Prof. Sílvio Fernandes



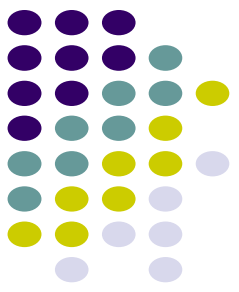


# Abstrações... reflexões

- Abstrações do hardware

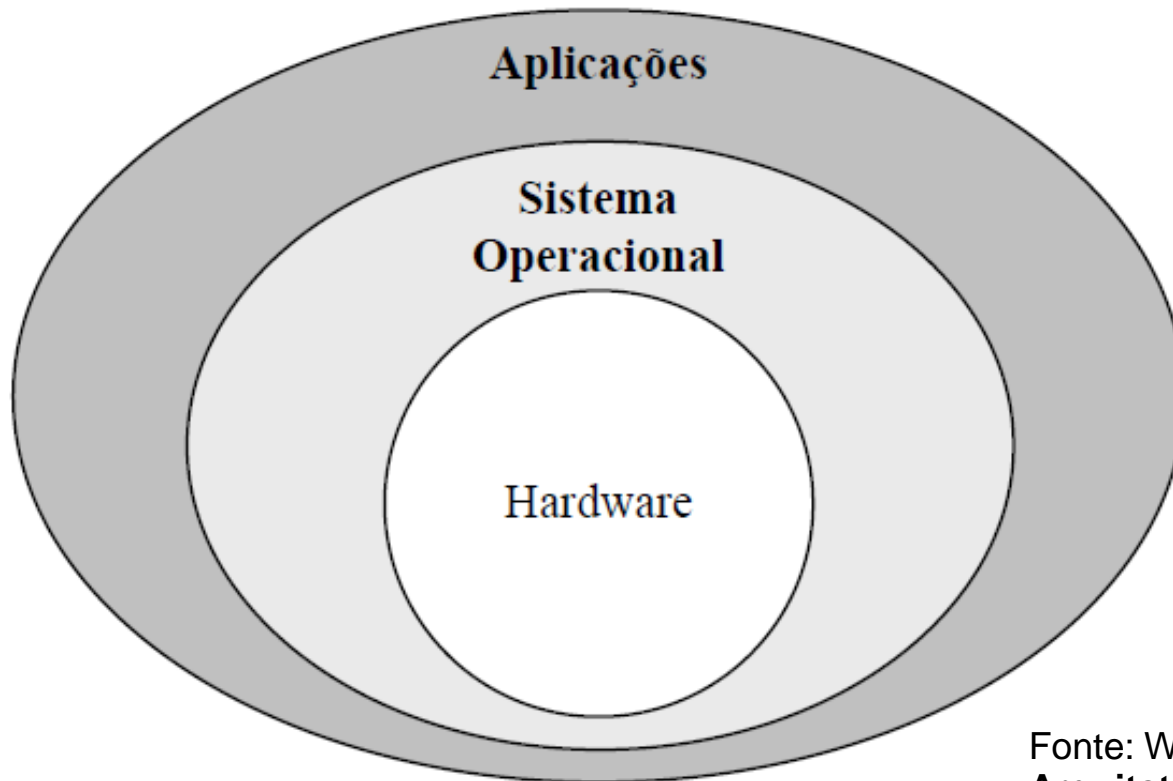


Fonte: WANDERLEY NETTO, Bráulio.  
**Arquitetura de Computadores:** A visão do  
software. Natal: Editora do CEFET-RN, 2005

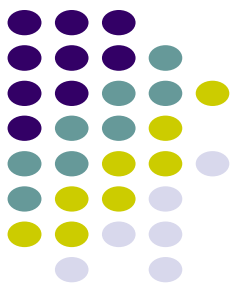


# Abstrações... reflexões

- Abstrações do software

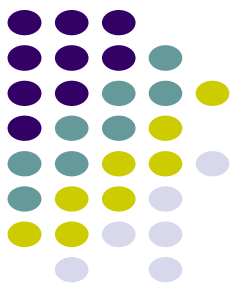


Fonte: WANDERLEY NETTO, Bráulio.  
**Arquitetura de Computadores:** A visão do  
software. Natal: Editora do CEFET-RN, 2005



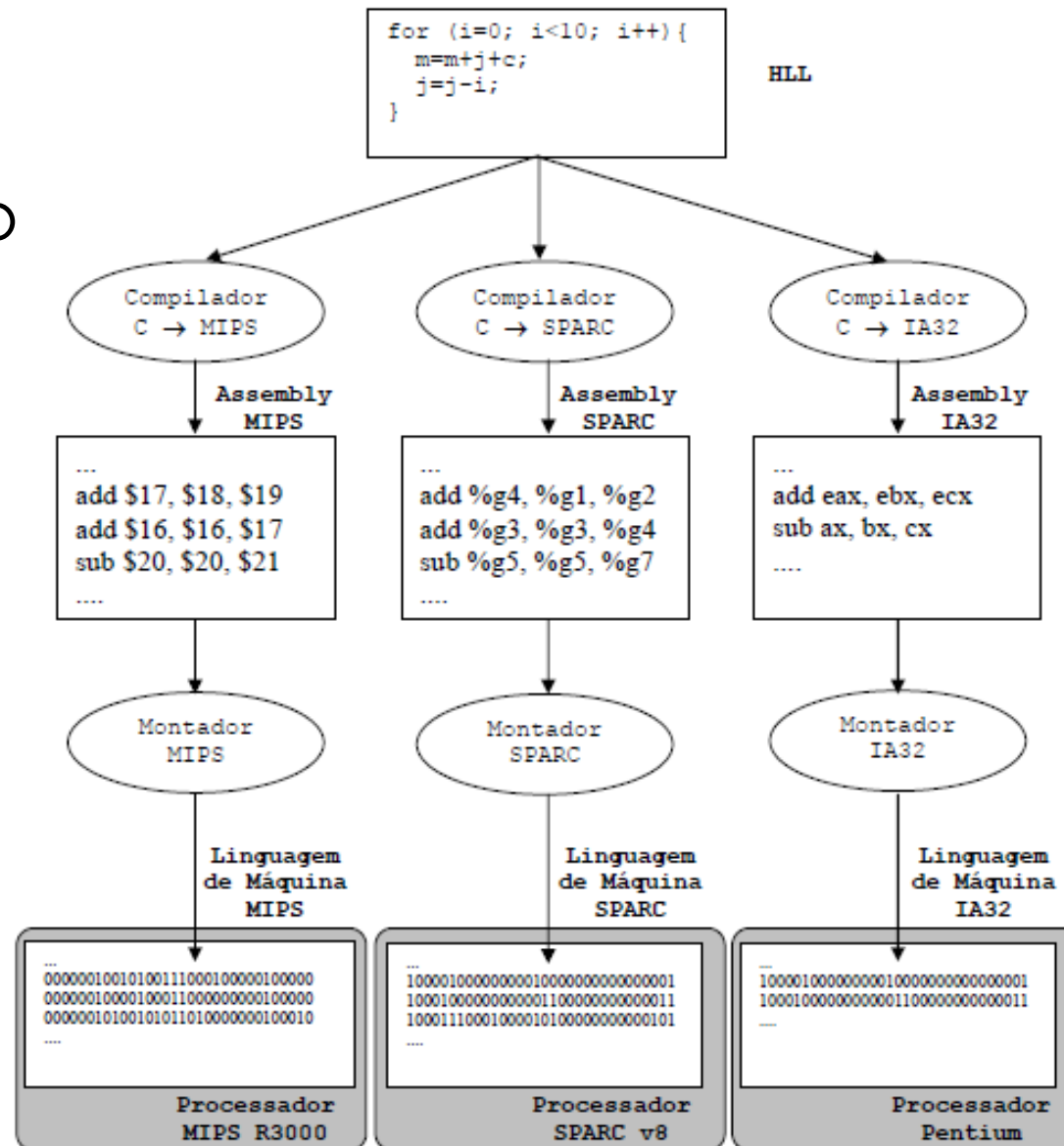
# Abstrações... reflexões

- É preciso entender abstrações como:
  - Software de aplicações
  - Software de sistemas
  - Linguagem assembly
  - Linguagem de máquina
  - Aspectos de arquitetura, como caches, memória virtual, canalização
  - Lógica sequencial, máquinas de estado finito
  - Lógica combinatória, circuitos aritméticos
  - Lógica booleana, 1s e 0s
  - Transistores usados para construir portas lógicas (CMOS)
  - Semicondutores/silício usados para construir transistores
  - Propriedades dos átomos, elétrons e dinâmica quantitativa
- Muito o que aprender!

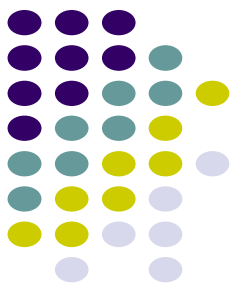


# Abstrações... reflexões

- Compilação e Montagem de um código em HLL



Fonte: WANDERLEY NETTO, Bráulio.  
**Arquitetura de Computadores:** A visão do software. Natal: Editora do CEFET-RN, 2005



# Motivação

- O que é isso?

```
.text
# carregando os valores da memoria
la $t0, g      # carrega o endereco de g em $t0
lw $s1, 0($t0) # carrega o valor de g em $s1
la $t0, h      # carrega o endereco de h em $t0
lw $s2, 0($t0) # carrega o valor de h em $s2
la $t0, i      # carrega o endereco de i em $t0
lw $s3, 0($t0) # carrega o valor de i em $s3
la $t0, j      # carrega o endereco de j em $t0
lw $s4, 0($t0) # carrega o valor de j em $s4

#Realizando o calculo
add $t0, $s1, $s2 #soma g e h
add $t1, $s3, $s4 #soma i e j
sub $s0, $t0, $t1 #(g+h)-(i+j)

#Armazena o resultado na memoria
la $t0, f      # carrega o endereco de f em $t0
sw $s0, 0($t0) # armazena o valor
```

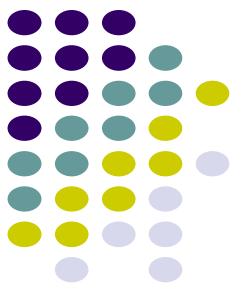
```
00111100000000010001000000000001
00110100001010000000000000000000
10001101000100010000000000000000
```

```
10000000000001
000000000000100
000000000000000
10000000000001
000000000001000
000000000000000
10000000000001
000000000001100
000000000000000
00000000100000
00100000100000
00000000100010
10000000000001
00000000010000
000000000000000
000000000001010
000000000001100
```

$f = (g + h) - (i + j);$

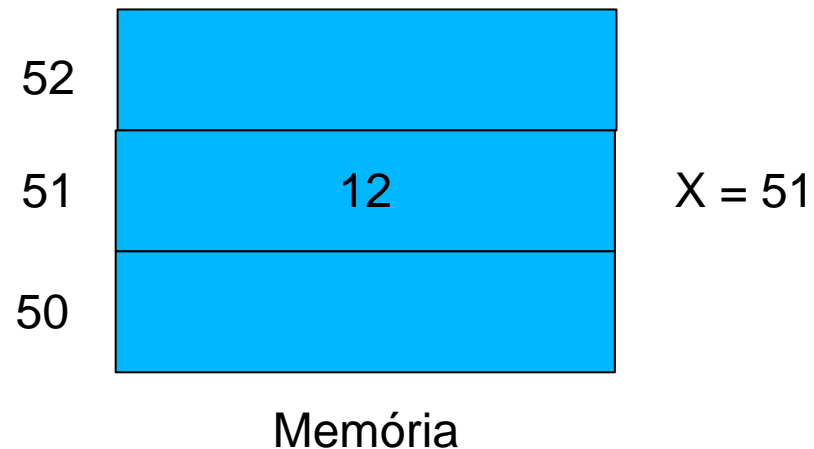
Assim é ainda  
mais fácil

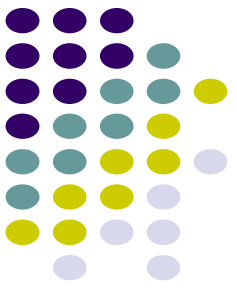
Assim fica melhor?



# Abstrações... reflexões

- O que é um programa?
- O que é uma variável?
- Como uma variável é referenciada?
- Como abstraímos uma variável?





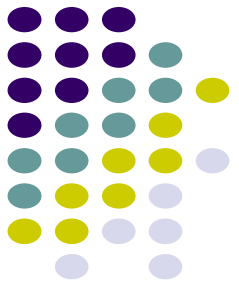
# Como os computadores funcionam

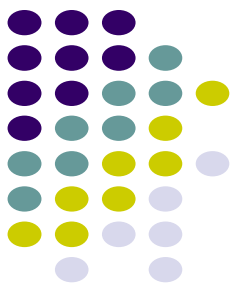
- O que é um programa?
  - Uma série de instruções
- O que é um computador?
  - máquina que pode manipular dados, resolver problemas e tomar decisões, SEMPRE sob controle de um programa.



# Conceitos

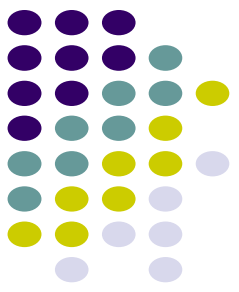
- **Organização** de computador
  - Refere-se às unidades operacionais e suas interconexões
  - Tem a ver quais componentes existem e como estão conectados





# Conceitos

- **Arquitetura** de computador
  - Refere-se aos atributos de um sistema visíveis a um programador ou, em outras palavras aqueles atributos que possuem um impacto direto sobre a execução lógica de um programa
  - Como a visão que um programador de baixo nível tem sobre a máquina para qual ele está codificando. Este programador enxerga então uma abstração da máquina chamada Arquitetura do Conjunto de Instruções, ISA (*Instruction Set Architecture*). Ela define como é possível fazer a nossa máquina funcionar.
- Muitos fabricantes de computador oferecem uma família de modelos de computador, todos com a mesma **arquitetura**, mas com diferença na **organização**

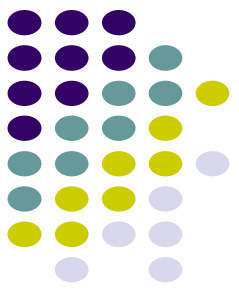


## Analogia: Agente secreto 89

- O assassinato de um líder mundial está para acontecer
- O agente secreto 89 deve descobrir quantos dias faltam para o assassinato
- Ele tem um contato com essa informação
- Para ninguém descobrir essa informação foi espalhada por uma série de 10 caixas postais
- Há 10 chaves e um conjunto de instruções para decifrar informação

# Agente secreto 89

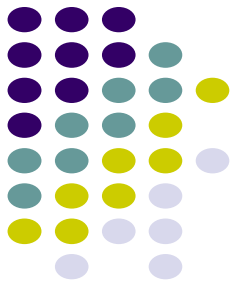
## Conjunto de Instruções



1. A informação em cada uma das caixas está escrita em código.
2. Abra a caixa 1 primeiro e execute a instrução localizada lá.
3. Continue pelas caixas restantes, em sequência, a menos que seja instruído do contrário.
4. Uma das caixas está preparada para explodir quando for aberta.

# Agente secreto 89

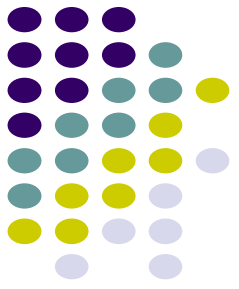
## Caixas postais com mensagens codificadas



Caixa 1	Caixa 2
Caixa 3	Caixa 4
Caixa 5	Caixa 6
Caixa 7	Caixa 8
Caixa 9	Caixa 10

# Agente secreto 89

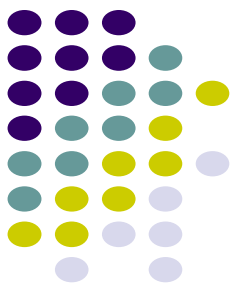
## Caixas postais com mensagens codificadas



1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	Caixa 2
Caixa 3	Caixa 4
Caixa 5	Caixa 6
Caixa 7	Caixa 8
Caixa 9	Caixa 10

# Agente secreto 89

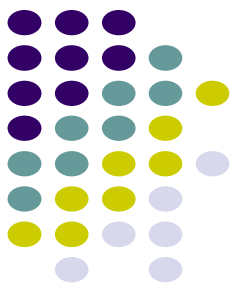
## Caixas postais com mensagens codificadas



1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	Caixa 2
Caixa 3	Caixa 4
Caixa 5	Caixa 6
Caixa 7	Caixa 8
9. 11	Caixa 10

# Agente secreto 89

## Caixas postais com mensagens codificadas



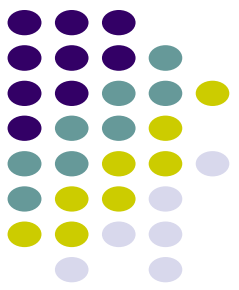
1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	Caixa 2
Caixa 3	Caixa 4
Caixa 5	Caixa 6
Caixa 7	Caixa 8
9. 11	Caixa 10

No. Agente = 89  
No. Armazenado em (9) = 11  
 $89 + 11 = 100$



# Agente secreto 89

## Caixas postais com mensagens codificadas



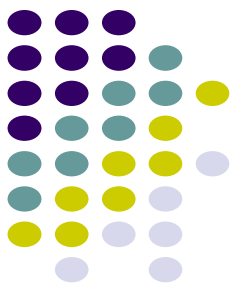
1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	2. Divida o resultado anterior pelo número armazenado na caixa (10).
Caixa 3	Caixa 4
Caixa 5	Caixa 6
Caixa 7	Caixa 8
9. 11	Caixa 10

Resultado anterior = 100  
No. Armazenado em (10) = ?

$$100 / ? = ?$$

# Agente secreto 89

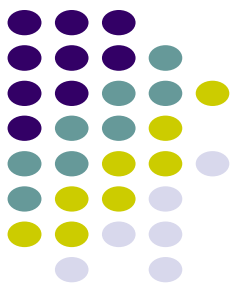
## Caixas postais com mensagens codificadas



1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	2. Divida o resultado anterior pelo número armazenado na caixa (10).	Resultado anterior = 100 No. Armazenado em (10) = 2  $100 / 2 = 50$
Caixa 3	Caixa 4	
Caixa 5	Caixa 6	
Caixa 7	Caixa 8	
9. 11	10. 2	

# Agente secreto 89

## Caixas postais com mensagens codificadas



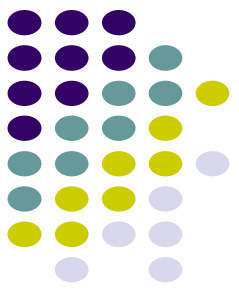
1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	2. Divida o resultado anterior pelo número armazenado na caixa (10).
3. Subtraia o número armazenado na caixa (8).	Caixa 4
Caixa 5	Caixa 6
Caixa 7	Caixa 8
9. 11	10. 2

Resultado anterior = 50  
No. Armazenado em (8) = ?

$$50 - ? = ?$$

# Agente secreto 89

## Caixas postais com mensagens codificadas



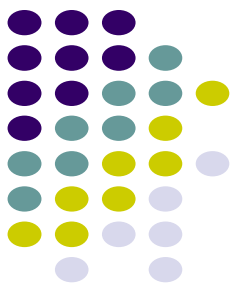
1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	2. Divida o resultado anterior pelo número armazenado na caixa (10).
3. Subtraia o número armazenado na caixa (8).	Caixa 4
Caixa 5	Caixa 6
Caixa 7	8. 20
9. 11	10. 2

Resultado anterior = 50  
No. Armazenado em (8) = 30

$$50 - 20 = 30$$

# Agente secreto 89

## Caixas postais com mensagens codificadas

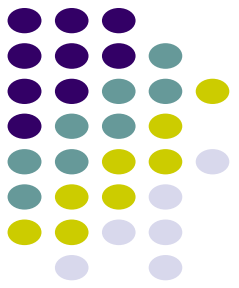


1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	2. Divida o resultado anterior pelo número armazenado na caixa (10).
3. Subtraia o número armazenado na caixa (8).	4. Se o resultado anterior não for igual a 30, vá para a caixa (7). Caso contrário continue para a próxima caixa.
Caixa 5	Caixa 6
Caixa 7	8. 20
9. 11	10. 2

Resultado anterior = 30

# Agente secreto 89

## Caixas postais com mensagens codificadas



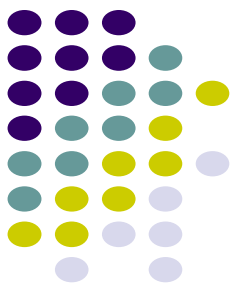
1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	2. Divida o resultado anterior pelo número armazenado na caixa (10).
3. Subtraia o número armazenado na caixa (8).	4. Se o resultado anterior não for igual a 30, vá para a caixa (7). Caso contrário continue para a próxima caixa.
5. Subtraia 13 do resultado anterior	Caixa 6
Caixa 7	8. 20
9. 11	10. 2

Resultado anterior = 30

$$30 - 13 = 17$$

# Agente secreto 89

## Caixas postais com mensagens codificadas




1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	2. Divida o resultado anterior pelo número armazenado na caixa (10).
3. Subtraia o número armazenado na caixa (8).	4. Se o resultado anterior não for igual a 30, vá para a caixa (7). Caso contrário continue para a próxima caixa.
5. Subtraia 13 do resultado anterior	6. Retorne para o quartel-general para receber mais instruções.
Caixa 7	8. 20
9. 11	10. 2

Resultado anterior = 17

# Agente secreto 89

## Caixas postais com mensagens codificadas



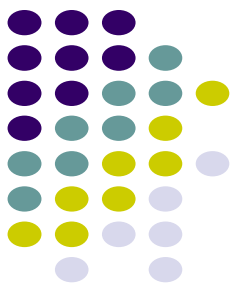
1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	2. Divida o resultado anterior pelo número armazenado na caixa (10).
3. Subtraia o número armazenado na caixa (8).	4. Se o resultado anterior não for igual a 30, vá para a caixa (7). Caso contrário continue para a próxima caixa.
5. Subtraia 13 do resultado anterior	6. Retorne para o quartel-general para receber mais instruções.
7. 	8. 20
9. 11	10. 2

Resultado anterior = 17



# Agente secreto 89

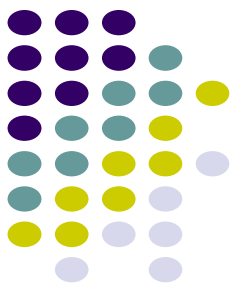
## Analogias




- Quem ou que é o agente secreto?
  - Computador que executa as instruções
- E o conjunto das caixas postais?
  - Memória contendo instruções (caixas de 1 a 6) e dados (8 a 10).
  - Caixa 7 sem equivalentes nos computadores
- Os números de cada caixa?
  - Endereços das posições de memória
- Onde são armazenados os resultados intermediários?
  - Memória interna do processador (registradores)

# Agente secreto 89

## Analogias

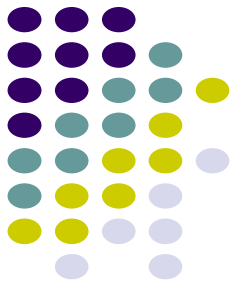


- Três classes de instruções
  - Caixas 1, 2, 3 e 5: instruções aritméticas
  - Caixa 4: instrução de decisão ou desvio condicional
  - Caixa 6: instrução de controle

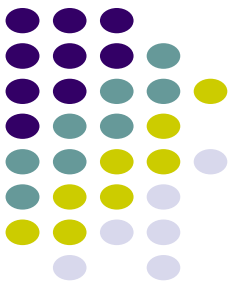
1. Some o número armazenado na caixa (9) ao seu número de código de agente secreto.	2. Divida o resultado anterior pelo número armazenado na caixa (10).
3. Subtraia o número armazenado na caixa (8).	4. Se o resultado anterior não for igual a 30, vá para a caixa (7). Caso contrário continue para a próxima caixa.
5. Subtraia 13 do resultado anterior	6. Retorne para o quartel-general para receber mais instruções.
7. 	8. 20
9. 11	10. 2

# Agente secreto 89

## Analogias



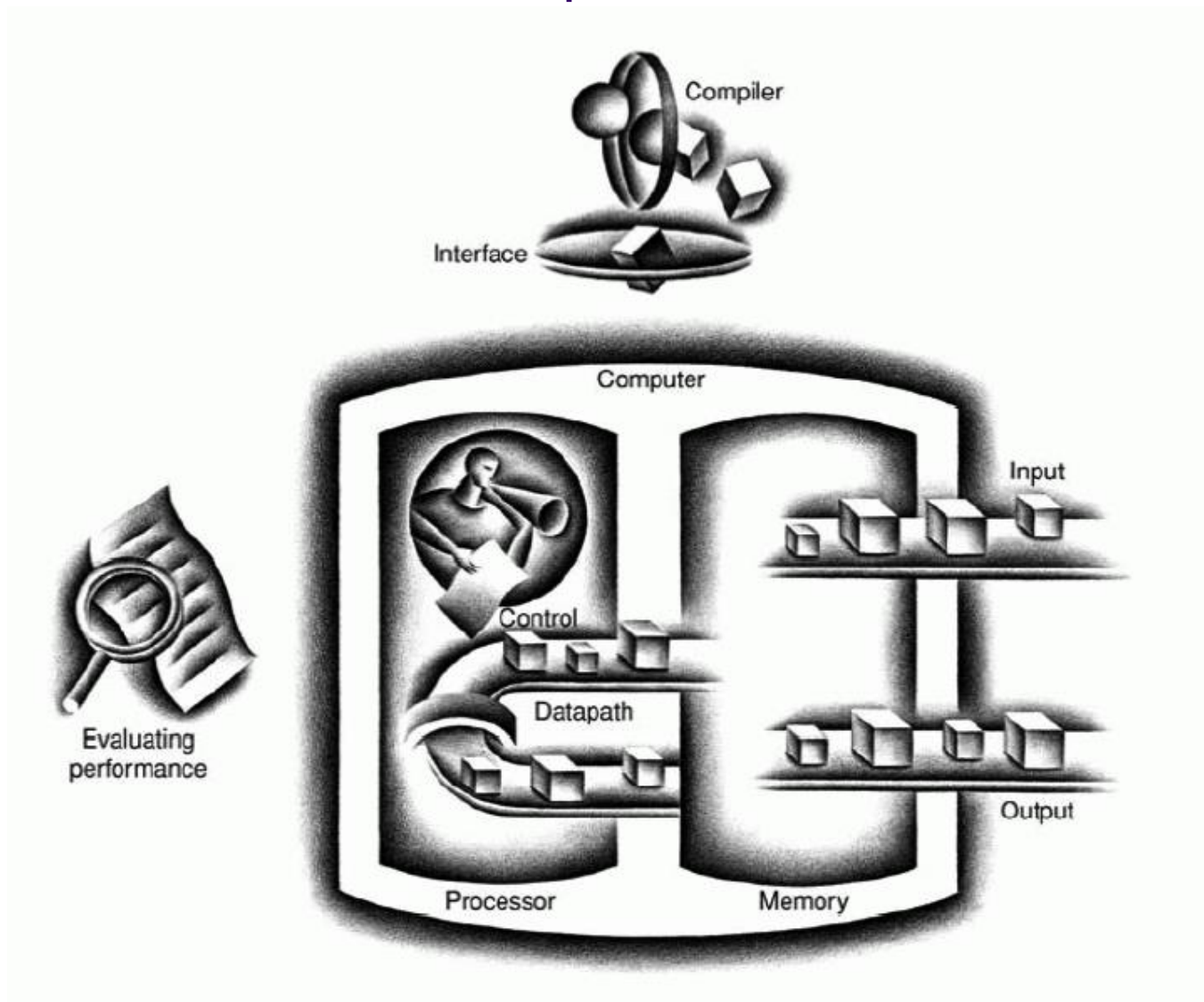
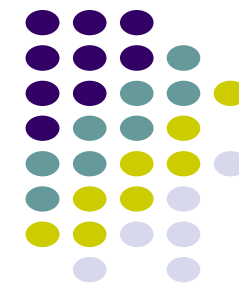
- Computador decodifica e executa as instruções armazenadas na memória
  - Sequencialmente desde que não haja desvio
  - Após o desvio são executadas sequencialmente a partir do novo endereço



# Organização de um computador

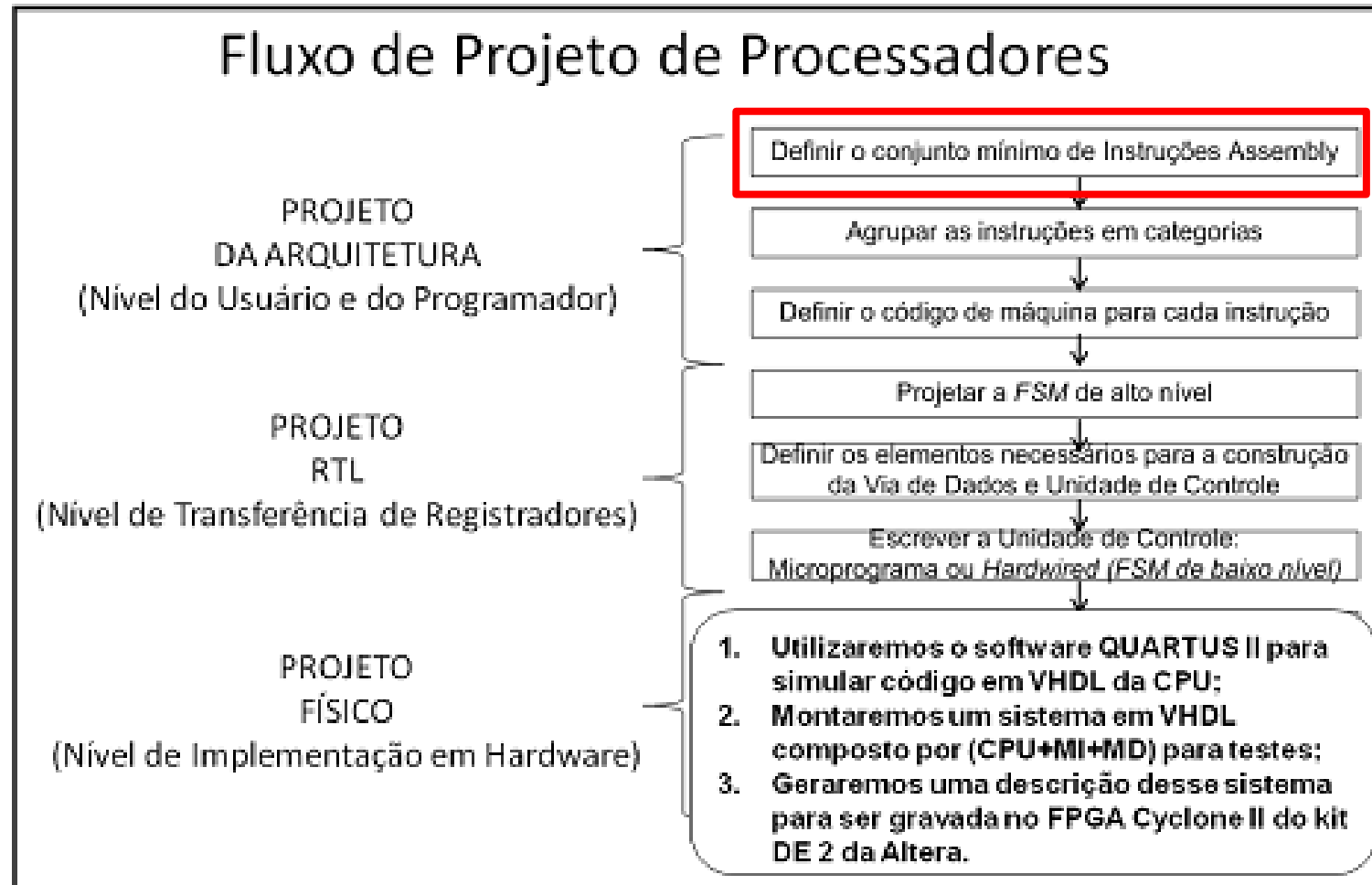
- Cinco componentes clássicos:
  - Entrada
  - Saída
  - Memória
  - Caminho de Dados
  - Controle

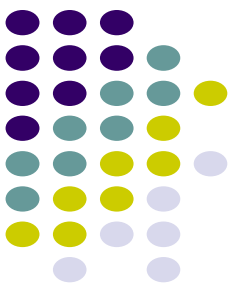
# Organização de um computador





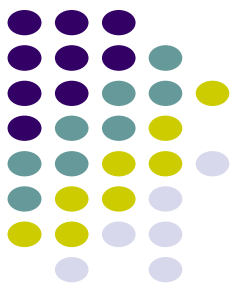
# Projeto de um processador





# Introdução

- Para controlar o hardware de um computador é preciso falar da sua linguagem
- As palavras da linguagem são chamadas de instruções
- Seu vocabulário é denominado conjunto de instruções ou ISA (*Instructions Set Architecture*)
- Objetivos dos projetistas de computador
  - Encontrar uma linguagem que facilite o projeto do hardware e do compilador enquanto maximiza o desempenho e minimiza o custo
  - Esse objetivo é antigo...

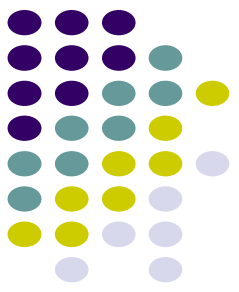


# Introdução

*“É fácil ver, por métodos lógicos formais, que existem certos **[conjuntos de instruções]** que são adequados para controlar e causar a execução de qualquer sequencia de operações... As considerações realmente decisivas, do ponto de vista atual, na seleção de um **[conjunto de instruções]**, são mais de natureza prática: a simplicidade do equipamento exigido pelo **[conjunto de instruções]** e a clareza de sua aplicação para os problemas realmente importantes, juntos com a velocidade com que tratam esses problemas.”*

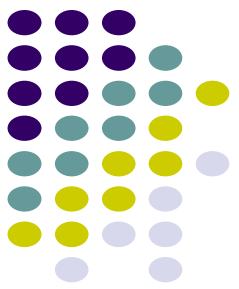
*Burks, Goldstine e von Neumann, 1947*





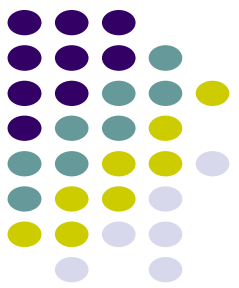
# Princípios de Projeto

- Princípio 1:
  - Simplicidade favorece a regularidade
- Princípio 2:
  - Menor significa mais rápido
- Princípio 3:
  - Agilize os casos mais comuns
- Princípio 4:
  - Um bom projeto exige bons compromissos



# Elementos de uma instrução

- Código de operação (Op code):
  - Faça isto.
- Referência a operando fonte:
  - Nisto.
- Referência a operando de destino:
  - Coloque a resposta aqui.
- Referência à próxima instrução:
  - Quando tiver feito isso, faça isto...



# Elementos de uma instrução

**Tabela 10.1** Utilização de endereços de instrução (instruções sem desvio)

Número de endereços	Representação simbólica	Interpretação
3	OP A, B, C	$A \leftarrow B \text{ OP } C$
2	OP A, B	$A \leftarrow A \text{ OP } B$
1	OP A	$AC \leftarrow AC \text{ OP } A$
0	OP	$T \leftarrow (T - 1) \text{ OP } T$

AC = acumulador

T = topo da pilha

(T - 1) = segundo elemento da pilha

A, B, C = locais de memória ou registradores

**Fonte:** STALLINGS, W. Arquitetura e organização de computadores: projeto para o desempenho. 8. ed. Prentice Hall, 2009.



# Números de endereços

- Programas para executar  $Y = \frac{A - B}{C + (D \times E)}$

Instrução	Comentário
SUB Y, A, B	$Y \leftarrow A - B$
MPY T, D, E	$T \leftarrow D \times E$
ADD T, T, C	$T \leftarrow T + C$
DIV Y, Y, T	$Y \leftarrow Y / T$

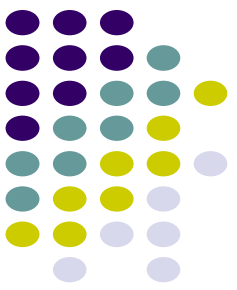
Instruções com 3 endereços

Instrução	Comentário
MOVE Y, A	$Y \leftarrow A$
SUB Y, B	$Y \leftarrow Y - B$
MOVE T, D	$T \leftarrow D$
MPY T, E	$T \leftarrow T \times E$
ADD T, C	$T \leftarrow T + C$
DIV Y, T	$Y \leftarrow Y / T$

Instruções com 2 endereços

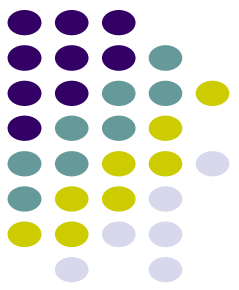
Instrução	Comentário
LOAD D	$AC \leftarrow D$
MPY E	$AC \leftarrow AC \times E$
ADD C	$AC \leftarrow AC + C$
STOR Y	$Y \leftarrow AC$
LOAD A	$AC \leftarrow A$
SUB B	$AC \leftarrow AC - B$
DIV Y	$AC \leftarrow AC / Y$
STOR Y	$Y \leftarrow AC$

Instruções com 1 endereço



## Por que MIPS?

- Típico conjunto de instruções da década de 1980
- Instruções simples, sempre realizando uma única operação
- As instruções possuem tamanho fixo de 32 bits.
- No total, o MIPS possui 32 registradores.
  - Cada um deles, de 32 bits
- Estava em quase 100 milhões de processadores em 2002
  - ARM é semelhante ao MIPS e em 2008 já estava em 3 bilhões de processadores
- Quem usa?
  - ATI Technologies, Broadcom, Cisco, NEC, Nintendo, Silicon Graphics, Sony, Texas Instruments e Toshiba, entre outros



# Operações do Hardware do Computador

- Todo computador precisa ser capaz de realizar aritmética

codop

## Notação assembly do MIPS

add a, b, c

- Instrui a somar as variáveis **b** e **c** e guardar em **a**

- A sequência de instruções soma 4 variáveis

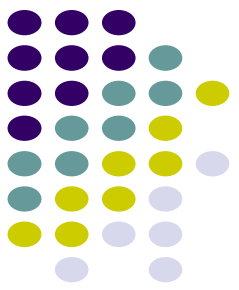
add a, b, c # A soma  $b+c$  é colocada em a

add a, a, d # A soma  $b+c+d$  é colocada em a

add a, a, e # A soma  $b+c+d+e$  é colocada em a

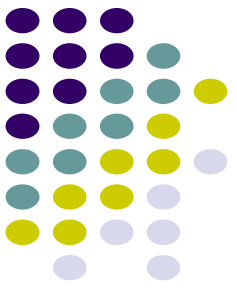
Operando destino

Operandos origem



# Operações do Hardware do Computador

- O número de operandos em operações como a adição é 3: os dois números sendo somados e um local para colocar a soma
- Exige que cada instrução tenha exatamente 3 endereços, de acordo com a filosofia de manter o hardware simples
- Princípio de projeto 1:
  - Simplicidade favorece a regularidade



# Instruções para soma e subtração

- E no caso de uma atribuição mais complexa, como por exemplo?
  - $f = (g + h) - (i + j);$ 
    - Neste caso, é necessário gerar mais de uma instrução.
    - Temos também de utilizar variáveis temporárias.

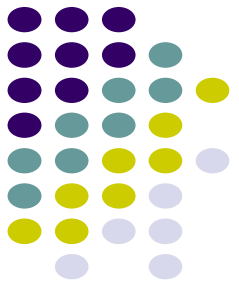
```
add t0, g, h
add t1, i, j
sub f, t0, t1
```

E aí pessoal, assembly é ou não é fácil? 😊

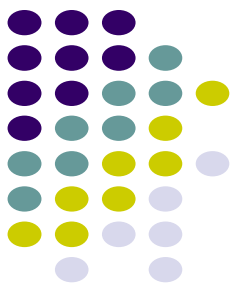
Isso é **quase** o assembly do MIPS



# Assembly do MIPS

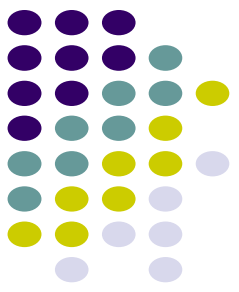


Categoria	Instrução	Exemplo	Significado	Comentário
Aritmética	Add	add a, b, c	$a = b + c$	Sempre 3 operandos
	Subtract	sub a, b, c	$a = b - c$	Sempre 3 operandos



# Registradores no MIPS

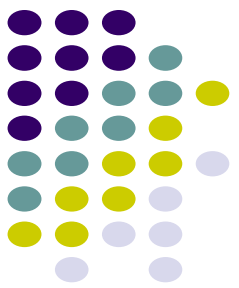
- Operandos do hardware de um computador
  - Ao contrário dos programas nas linguagens de alto nível, os operandos das instruções aritméticas são restritos.
    - Os operandos de uma instrução aritmética são os registradores.
  - Lembrem-se que no MIPS só temos 32 registradores.
  - Qual o motivo para termos uma quantidade tão pequena de registradores?
    - *Princípio de Projeto 2: “Menor significa mais rápido”*



# Registradores no MIPS

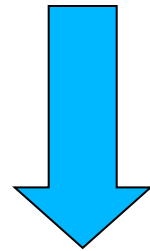
- Os nomes dos registradores MIPS obedecem ao seguinte padrão:
  - Utilizamos “\$” seguido por dois caracteres para representar um registrador.
  - \$s0, \$s1, \$s2,... representam os registradores que correspondem às variáveis dos programas em C e Java.
  - \$t0, \$t1, ... Representam os registradores que armazenam valores temporários.

Mais tarde veremos a convenção dos nomes e recomendação de uso dos registradores



# Registradores no MIPS

- Logo, para o programa anterior, teríamos:
  - $f = (g + h) - (i + j);$

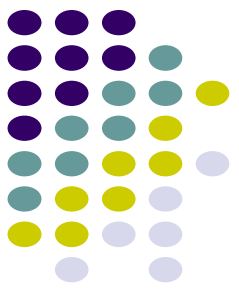


Assembly

```
add t0, g, h  
add t1, i, j  
sub f, t0, t1
```

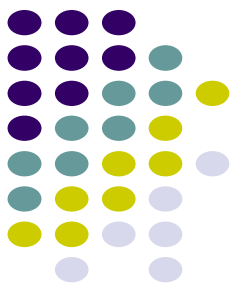


```
add $t0, $s1, $s2  
add $t1, $s3, $s4  
sub $s0, $t0, $t1
```



# Instrução para soma com constantes

- addi (Adição Imediata).
  - Para evitar o overhead de termos de ler uma constante da memória para depois utilizá-la em uma soma, MIPS dá suporte a uma instrução especial, o addi.
  - Basicamente, ela realiza uma soma com um valor constante.
    - Sintaxe:
      - `addi $r1, $r2, const # armazena o valor de $r2 + const no reg. $r1.`
  - Essa instrução obedece ao princípio de projeto 3:
    - *“Agilize os casos mais comuns”*
  - Realmente ele é um caso comum, pois soma com constantes representam um percentual bastante elevado do código.
    - Lembrem-se que grande parte dos laços fazem uso desta instrução.
    - A constante pode ter sinal



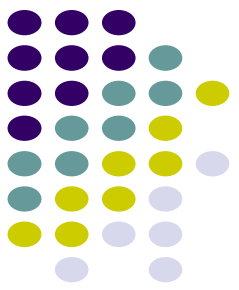
# Instruções Lógicas

- Operação OR

`or $8, $9, $10      # $8 = $9 or $10`

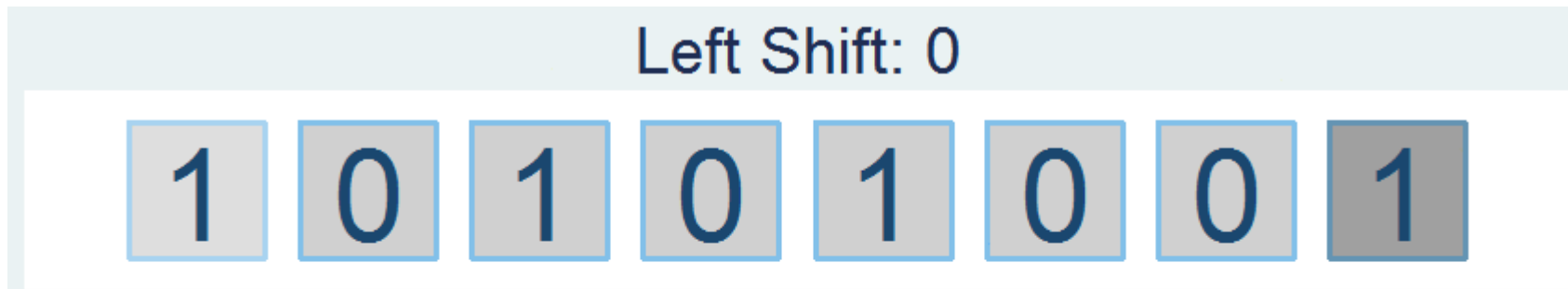
*se \$9 contém 0001 1001 1111 0000 0000 1111 0011 1100<sub>2</sub> e  
\$10 contém 1111 1101 1111 1110 0000 1111 1100 1100<sub>2</sub>, então o resultado  
será, em \$8, 1111 1101 1111 1110 0000 1111 1111 1100<sub>2</sub>.*

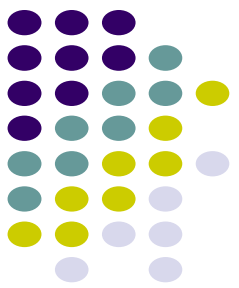
- Operações AND, XOR e NOR seguem a mesma sintaxe
- Operação ANDI
  - `andi $8, $9, 121   # $8 = $9 and 121.`
  - ORI segue a mesma ideia



# Instruções Lógicas

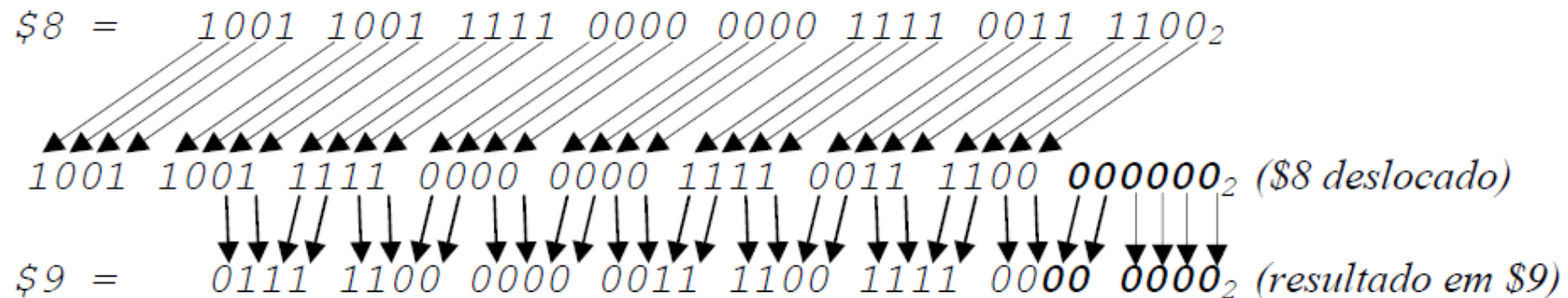
- Deslocamento lógico (Shift)



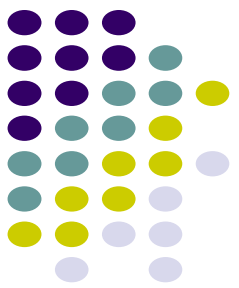


# Instruções Lógicas

- Deslocamento à direita e à esquerda
  - Instrução SLL (*shift left logical*)
    - Desloca n bits à esquerda e acrescenta zeros à direita
    - `sll $9, $8, 6`      #  $\$9 = \$8 \ll 6$
    - Se \$8 tem  $1001\ 1001\ 1111\ 0000\ 0000\ 1111\ 0011\ 1100_2$





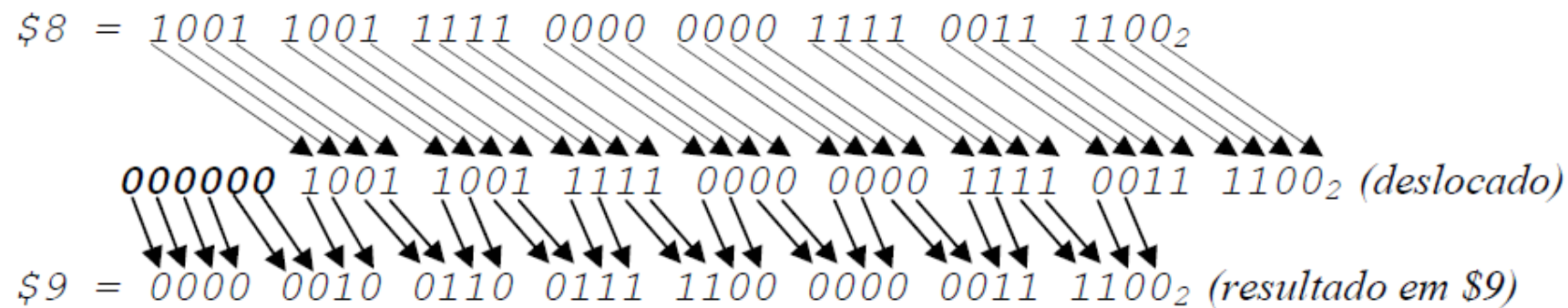


# Instruções Lógicas

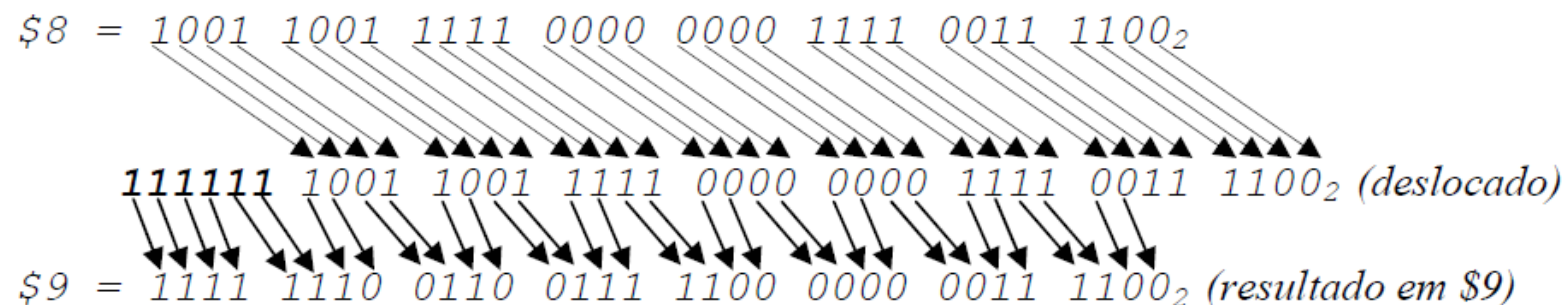
- Deslocamento à direita e à esquerda

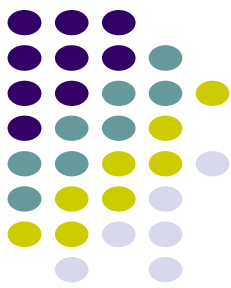
- Instrução SRL (*shift right logical*)

- srl \$9, \$8, 6      # \$9 = \$8 >> 6



- Instrução SRA (*shift right arithmetic*)





# Instruções Lógicas e Aritméticas

- Instruções aritméticas

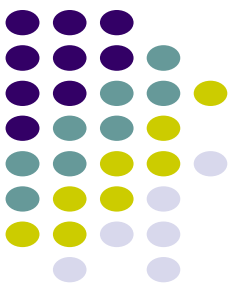
Categoria	Nome	Exemplo	Operação	Comentários
Aritmética	add	add \$8, \$9, \$10	$\$8 = \$9 + \$10$	<i>Overflow</i> gera exceção
	sub	sub \$8, \$9, \$10	$\$8 = \$9 - \$10$	<i>Overflow</i> gera exceção
	addi	addi \$8, \$9, 40	$\$8 = \$9 + 40$	<i>Overflow</i> gera exceção Valor do imediato na faixa entre -32.768 e +32.767
	addu	addu \$8, \$9, \$10	$\$8 = \$9 + \$10$	<i>Overflow</i> não gera exceção
	subu	subu \$8, \$9, \$10	$\$8 = \$9 - \$10$	<i>Overflow</i> não gera exceção
	addiu	addiu \$8, \$9, 40	$\$8 = \$9 + 40$	<i>Overflow</i> não gera exceção Valor do imediato na faixa entre 0 e 65.535
	mul	mul \$8, \$9, \$10	$\$8 = \$9 \times \$10$	<i>Overflow</i> não gera exceção HI, LO imprevisíveis após a operação
	mult	mult \$9, \$10	$HI, LO = \$9 \times \$10$	<i>Overflow</i> não gera exceção
	multu	multu \$9, \$10	$HI, LO = \$9 \times \$10$	<i>Overflow</i> não gera exceção
	div	div \$9, \$10	$HI = \$9 \bmod \$10$ $LO = \$9 \div \$10$	<i>Overflow</i> não gera exceção
	divu	divu \$9, \$10	$HI = \$9 \bmod \$10$ $LO = \$9 \div \$10$	<i>Overflow</i> não gera exceção



# Instruções Lógicas e Aritméticas

- Instruções lógicas

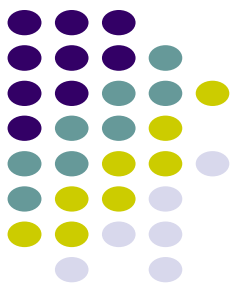
Categoria	Nome	Exemplo	Operação	Comentários
lógicas	or	or \$8, \$9, \$10	$\$8 = \$9 \text{ or } \$10$	
	and	and \$8, \$9, \$10	$\$8 = \$9 \text{ and } \$10$	
	xor	xor \$8, \$9, 40	$\$8 = \$9 \text{ xor } 40$	
	nor	nor \$8, \$9, \$10	$\$8 = \$9 \text{ nor } \$10$	
	andi	andi \$8, \$9, 5	$\$8 = \$9 \text{ and } 5$	Imediato em 16 bits
	ori	ori \$8, \$9, 40	$\$8 = \$9 \text{ or } 40$	Imediato em 16 bits
	sll	sll \$8, \$9, 10	$\$8 = \$9 \ll 10$	Desloc. $\leq 32$
	srl	srl \$8, \$9, 5	$\$8 = \$9 \gg 5$	Desloc. $\leq 32$
	sra	sra \$8, \$9, 5	$\$8 = \$9 \gg 5$	Desloc. $\leq 32$ , preserva sinal



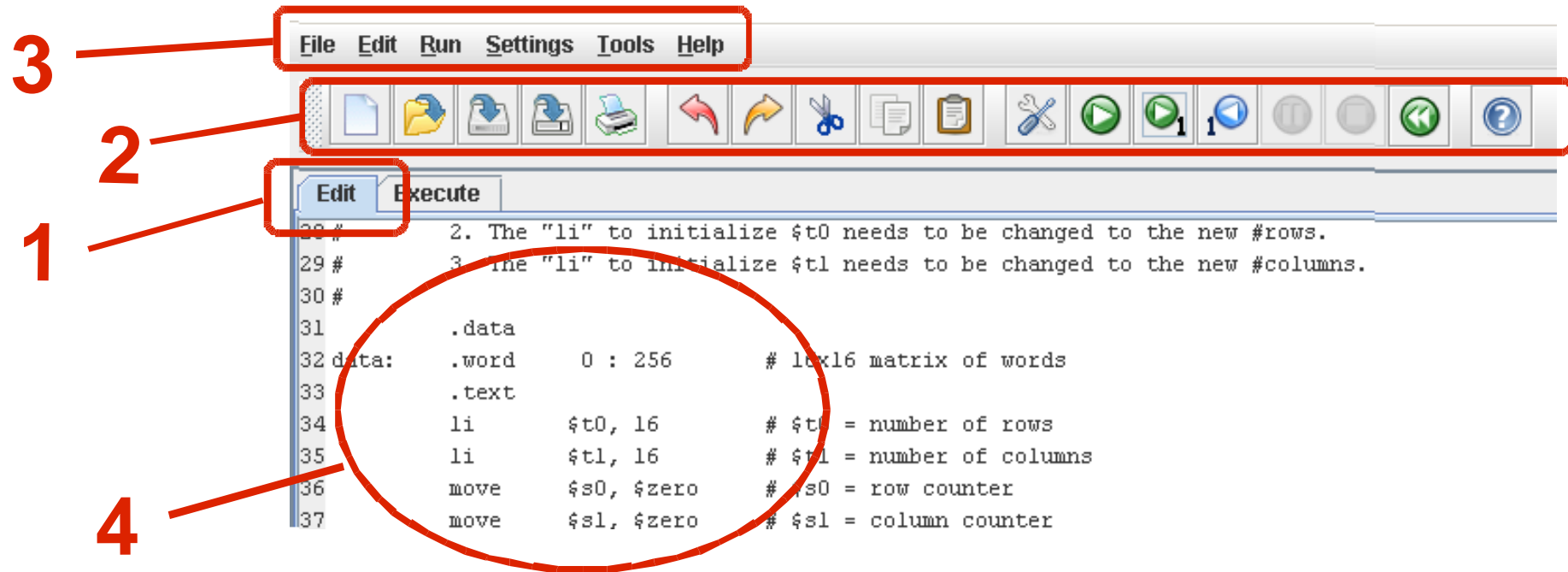
# Ferramenta

- MARS (MIPS Assembler and Runtime Simulator)
  - IDE para desenvolvimento de programas assembly MIPS
  - Compilação e verificação de erro sintático
  - Simulação de execução
  - Gratuita
    - <http://courses.missouristate.edu/kenvollmar/mars/>
  - Tutorial/Manual
    - <http://courses.missouristate.edu/KenVollmar/MARS/CCSC-CP%20material/MARS%20Tutorial.doc>

# MARS

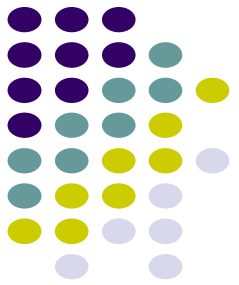


- Ambiente de Edição



1. Edit display is indicated by highlighted tab.
- 2, 3. Typical edit and execute operations are available through icons and menus, dimmed-out when unavailable or not applicable.
4. WYSIWYG editor for MIPS assembly language code.

# MARS



- Ambiente de Simulação

The screenshot shows the MARS MIPS simulator interface with several components and annotations:

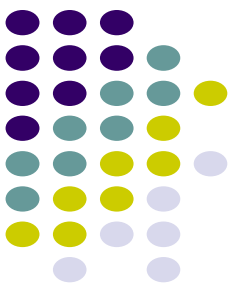
- 1**: Points to the **Execute** button in the toolbar.
- 2**: Points to the **Text Segment** tab.
- 3**: Points to the **Data Segment** tab.
- 4**: Points to the address selection dropdown menu showing **0x10010000 (.data)**.
- 5**: Points to the **Hexadecimal Addresses** checkbox.
- 6**: Points to the **Hexadecimal Values** checkbox.
- 7**: Points to the **Registers** panel on the right.
- 8**: Points to the **Labels** panel on the right.
- 9**: Points to the **Run speed at max (no interaction)** slider.

The **Text Segment** window displays assembly code with the following visible instructions:

```
0x00400000 0x3c010000 lui $t0, 0
0x00400004 0x34280010 ori $s0, $t0, 16
0x00400008 0x3c010000 lui $t1, 0
0x0040000c 0x34290010 ori $s1, $t1, 16
0x00400010 0x00008021 addu $t2, $t0, $t1
0x00400014 0x00008021 addu $t3, $t0, $t1
0x00400018 0x00008021 addu $t4, $t0, $t1
0x0040001c 0x02090018 mult $t5, $t0, $t1
0x00400020 0x00009022 mflo $t6
0x00400024 0x02519024 add $t7, $t0, $t1
0x00400028 0x00129080 sll $t8, $t0, 2
0x0040002c 0x3c011001 lui $t9, 4097
0x00400030 0x00320821 addu $t10, $t0, $t1
0x00400034 0xac2a0000 sw $t0, 0($t1)
0x00400038 0x214a0001 addi $t0, $t0, 1
0x0040003c 0x22100001 addi $t1, $t1, 1
```

The **Data Segment** window shows a memory dump with addresses and values in hexadecimal.

The **Registers** panel on the right lists registers \$zero through \$s31 with their current values.

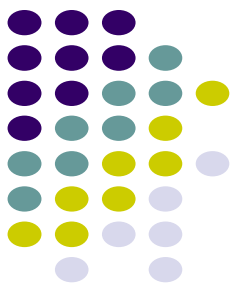


## Sobre o MARS

- Copie o código para o editor do MARS:

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

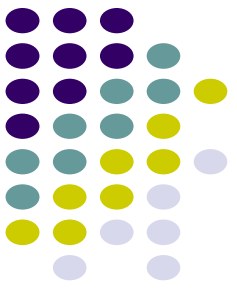
- Salve com extensão .asm
- Compile
- Atribua valores aos registradores \$s1, \$s2, \$s3 e \$s4
- Execute o código (uma instrução por vez) e verifique o resultado nos registradores de destino



## Exercícios – programas em assembly MIPS

1. Inicialize 2 registradores com valores quaisquer, assumindo que são variáveis inteiras **X** e **Y** inicializadas. Em seguida incremente em 1 o valor de **X**, subtraia o resultado pelo valor de **Y** e armazene em outro registrador. Verifique no MARS, na execução passo-a-passo os valores sendo atribuídos aos registradores.
  - a) Os valores das variáveis devem estar nos registradores \$s0 e \$s1
  - b) Para inicializar utilize **addi**





## Referências

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