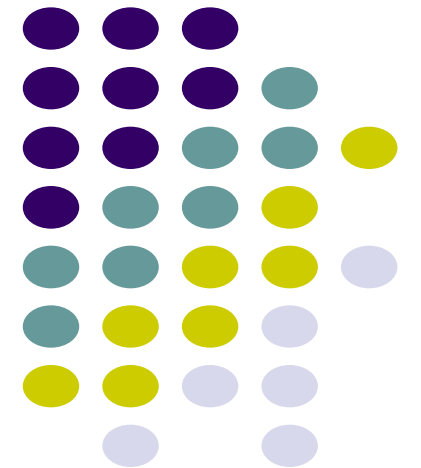


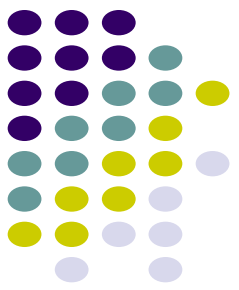
# Arquitetura e Organização de Computadores

Instruções: linguagem de máquina

Prof. Sílvio Fernandes

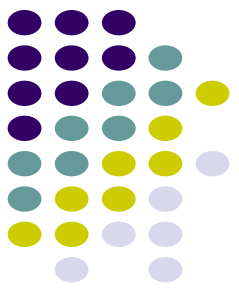


# Representação das instruções em linguagem de máquina.



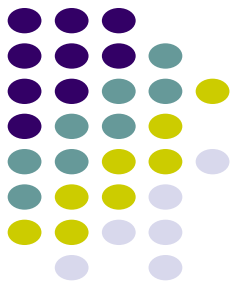
- Humanos aprendem a pensar na base 10, mas os números podem ser representados em qualquer base
  - 123 base 10 = 1111011 base 2
- No HW do computador, os números são mantidos como uma série de sinais eletrônicos altos e baixos, por isso são considerados base 2
- Toda informação é composta por **dígitos binários** ou bits
- As instruções também são mantidas como uma série de bits e podem ser representadas como números

# Representação das instruções em linguagem de máquina.

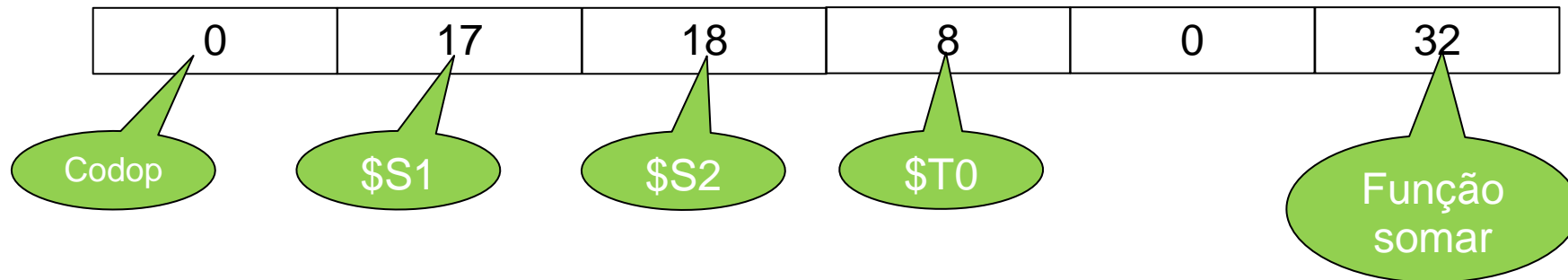


- Vamos abordar a conversão da instrução
  - **add \$t0, \$s1, \$s2**
- Os 6 bits mais significativos (mais a esquerda) formam um campo chamado **opcode**
  - Para **add** o *opcode* vale  $000000_2$  (como todas as inst. lógicas e aritméticas)
  - Para diferenciá-la há um campo função (**funct**)
    - Vale 32 ( $100000_2$ ) para instrução add
- Os operandos (registradores) são chamados genericamente de:
  - **rd**: destino
  - **rs**: fonte
  - **rt**: alvo (pode ser destino ou fonte) – no **add** é fonte

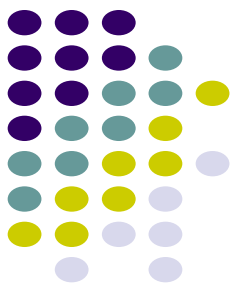
# Representação das instruções em linguagem de máquina.



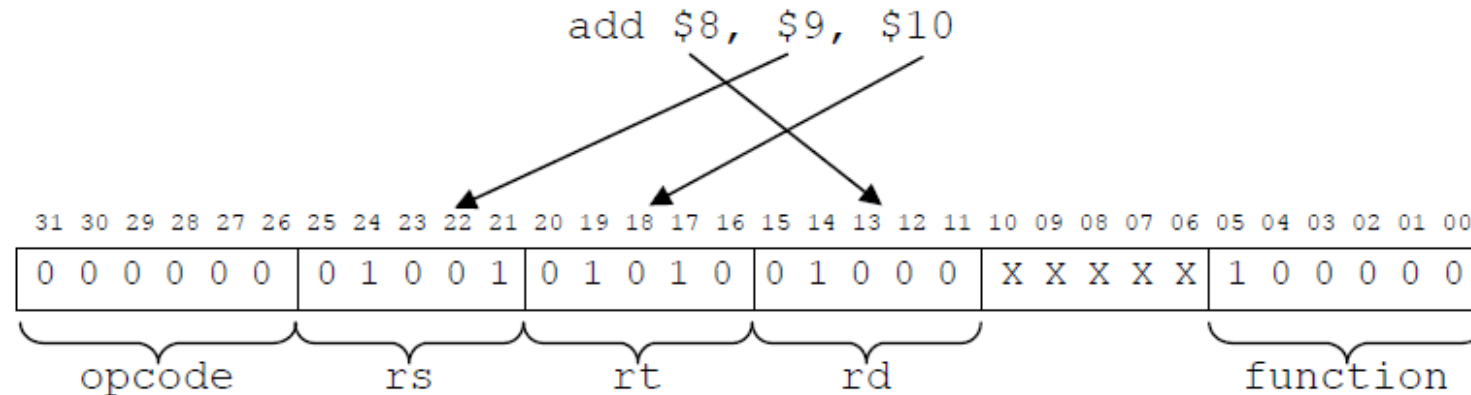
- No assembly do MIPS, os registradores tem endereços
  - \$t0 a \$t7: 8 a 15
  - \$s0 a \$s7: 16 a 23
- Por exemplo a instrução **add \$t0, \$s1, \$s2** possui a seguinte representação (decimal):



# Representação das instruções em linguagem de máquina.



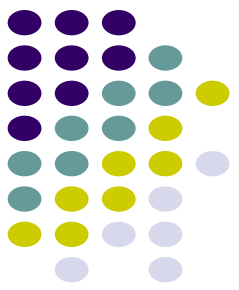
- Para a instrução **add \$8, \$9, \$10**



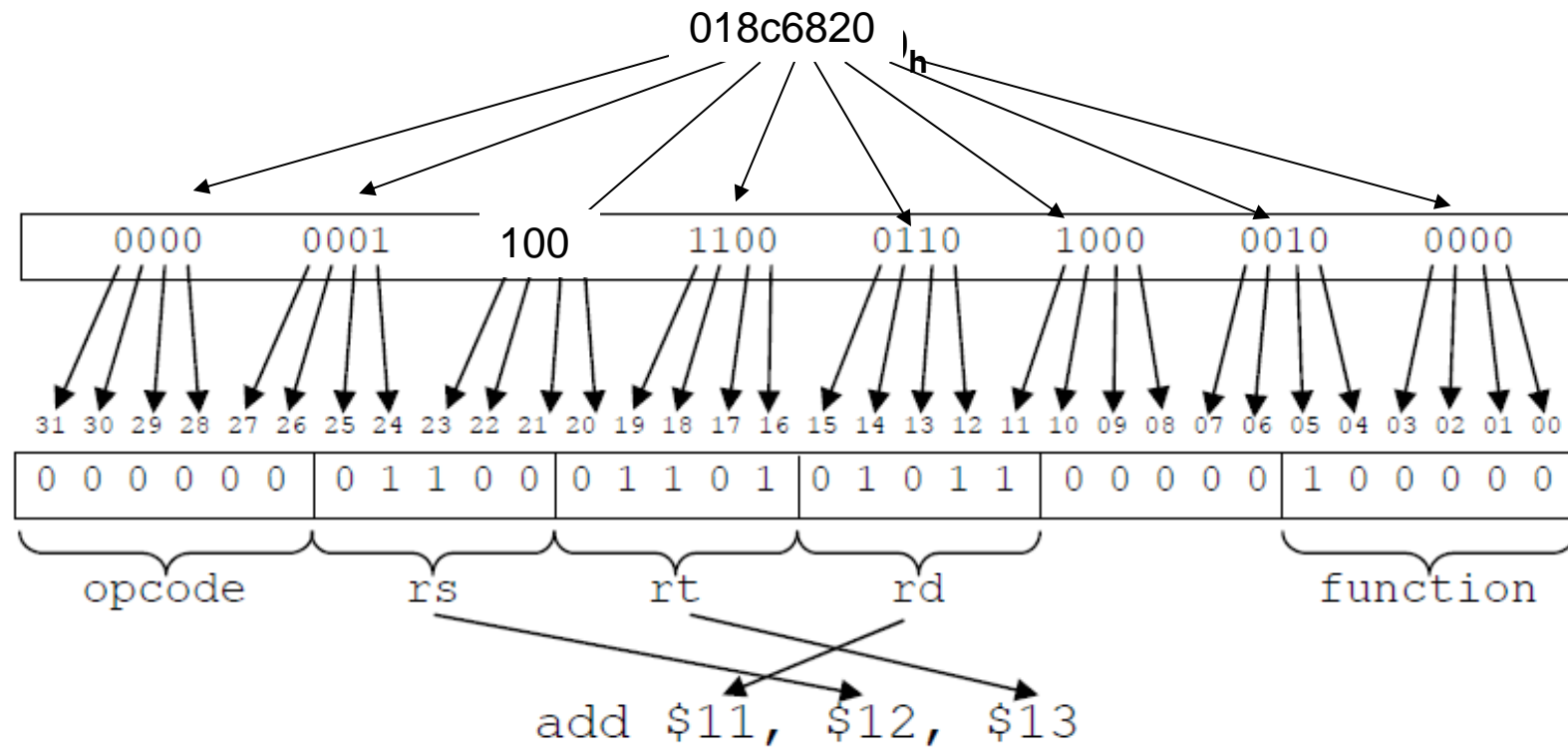
Fonte: WANDERLEY NETTO, Bráulio. **Arquitetura de Computadores**: A visão do software. Natal: Editora do CEFET-RN, 2005

- Entre os bits 6 e 10 serão utilizados apenas em instruções de deslocamento
  - Assim, nas demais instruções sempre será  $00000_2$
- Essa instrução em hexadecimal é: **016c820<sub>h</sub>**

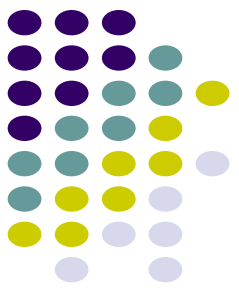
# Representação das instruções em linguagem de máquina.



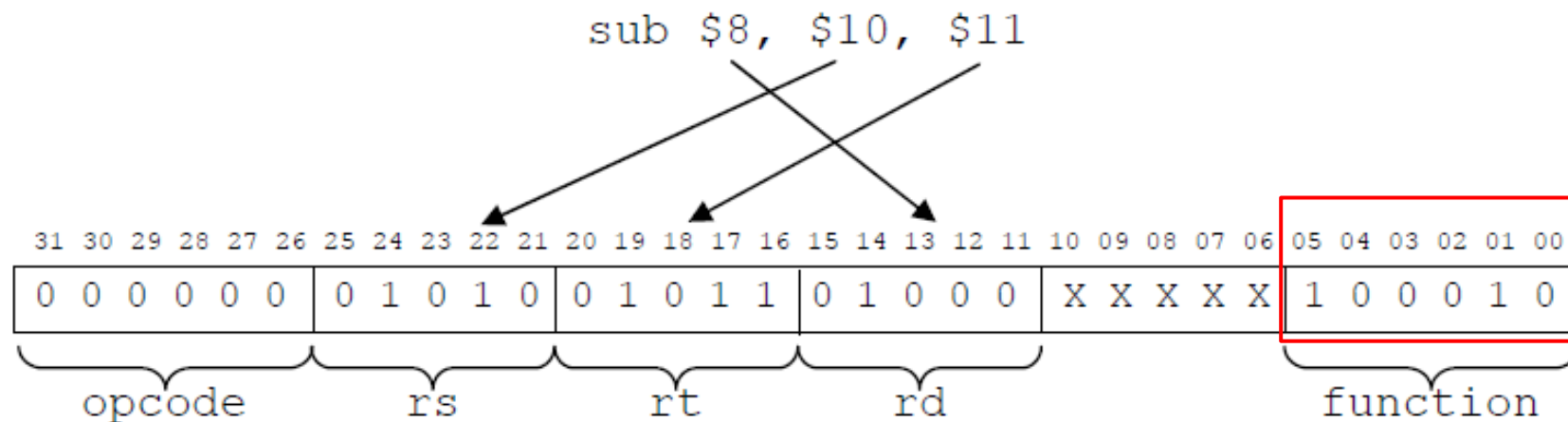
- Para a instrução **add \$11, \$12, \$13**



# Representação das instruções em linguagem de máquina.

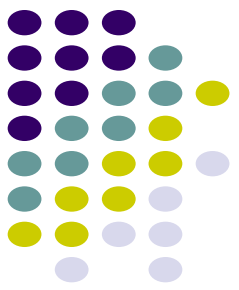


- Para a instrução **sub \$8, \$10, \$11**

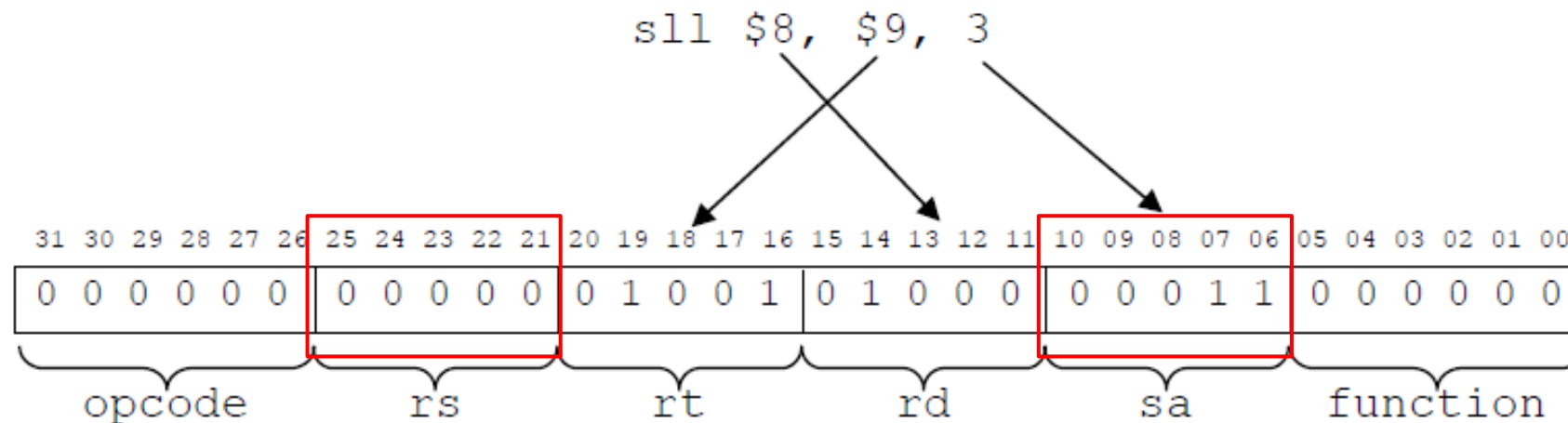


Fonte: WANDERLEY NETTO, Bráulio. **Arquitetura de Computadores**: A visão do software. Natal: Editora do CEFET-RN, 2005

# Representação das instruções em linguagem de máquina.



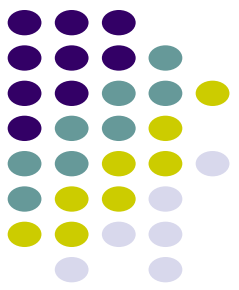
- Para a instrução **sll \$8, \$9, 3**



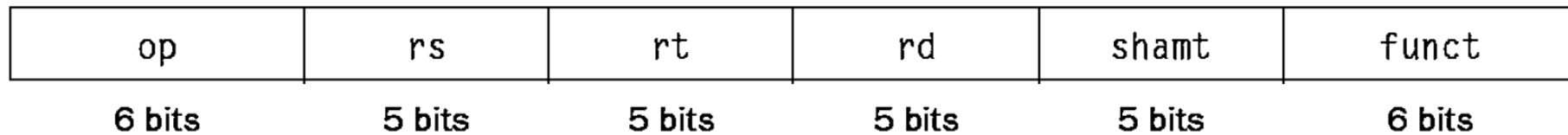
Fonte: WANDERLEY NETTO, Bráulio. **Arquitetura de Computadores**: A visão do software. Natal: Editora do CEFET-RN, 2005



# Representação das instruções em linguagem de máquina.

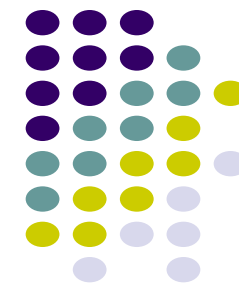


- Campos de uma instrução MIPS:
  - Instruções **tipo-R** (de registradores) ou **formato R**.



- **op**: Operação básica da instrução (opcode)
- **rs**: registrador do primeiro operando de origem
- **rt**: registrador do segundo operando de origem
- **rd**: registrador do operando de destino
- **shamt**: “shift amount” (quantidade de deslocamento).
- **funct**: seleciona a variante específica do campo op (código de função)

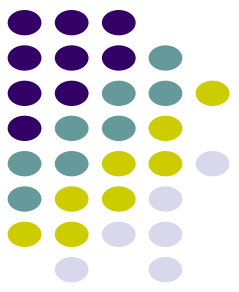
# Representação das instruções em linguagem de máquina.



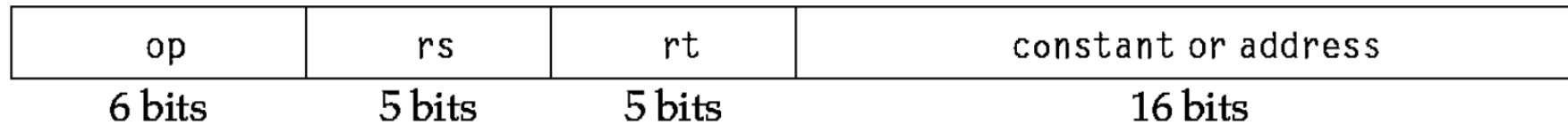
- Instruções **tipo-R**

Nome	Formato	Exemplo	Codificação					
			opcode	rs	rt	rd	sa	function
sll	R	sll \$8, \$9, 3	0	0	10	8	3	0
srl	R	srl \$8, \$9, 3	0	0	10	8	3	2
jr	R	jr \$8	0	8	0	0	0	8
mfhi	R	mfhi \$8	0	0	0	8	0	16
mflo	R	mflo \$8	0	0	0	8	0	18
mult	R	mult \$9, \$10	0	9	10	0	0	24
multu	R	multu \$9, \$10	0	9	10	0	0	25
div	R	div \$9, \$10	0	9	10	0	0	26
divu	R	divu \$9, \$10	0	9	10	0	0	27
add	R	add \$8, \$9, \$10	0	9	10	8	0	32
addu	R	addu \$8, \$9, \$10	0	9	10	8	0	33
sub	R	sub \$8, \$9, \$10	0	9	10	8	0	34
subu	R	subu \$8, \$9, \$10	0	9	10	8	0	35
and	R	and \$8, \$9, \$10	0	9	10	8	0	36
or	R	or \$8, \$9, \$10	0	9	10	8	0	37
slt	R	slt \$8, \$9, \$10	0	9	10	8	0	42
sltu	R	sltu \$8, \$9, \$10	0	9	10	8	0	43
mul	R	mul \$8, \$9, \$10	28	9	10	8	0	2

# Representação das instruções em linguagem de máquina.

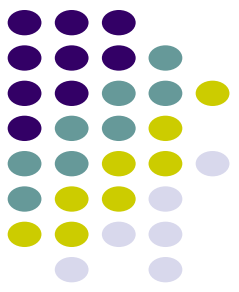


- Campos de uma instrução MIPS:
  - Instruções **tipo-I** (de imediato) ou **formato I**.

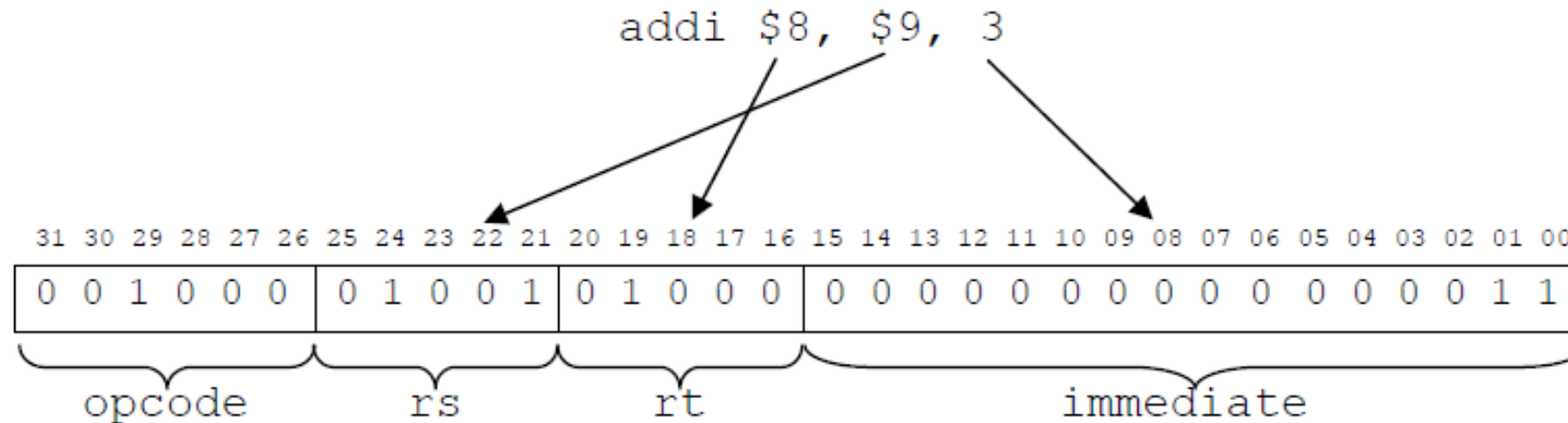


- **op**: Operação básica da instrução (opcode)
- **rs**: registrador do operando de origem
- **rt**: registrador de destino
- **constant or address**: constante ou endereço de memória.

# Representação das instruções em linguagem de máquina.

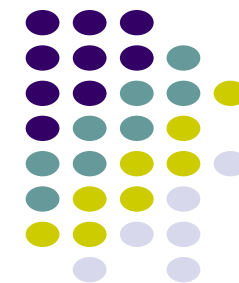


- Considere **addi \$8, \$9, 3**



- O campo *immediate* contém 16 bits, o que implica na limitação de um dos operandos à faixa entre -32768 e +32767
  - Uma instrução **addi \$8, \$9, 128256** não é possível ser codificada

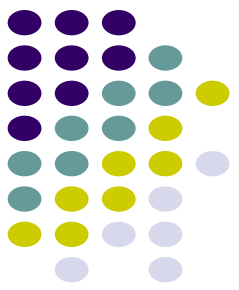
# Representação das instruções em linguagem de máquina.



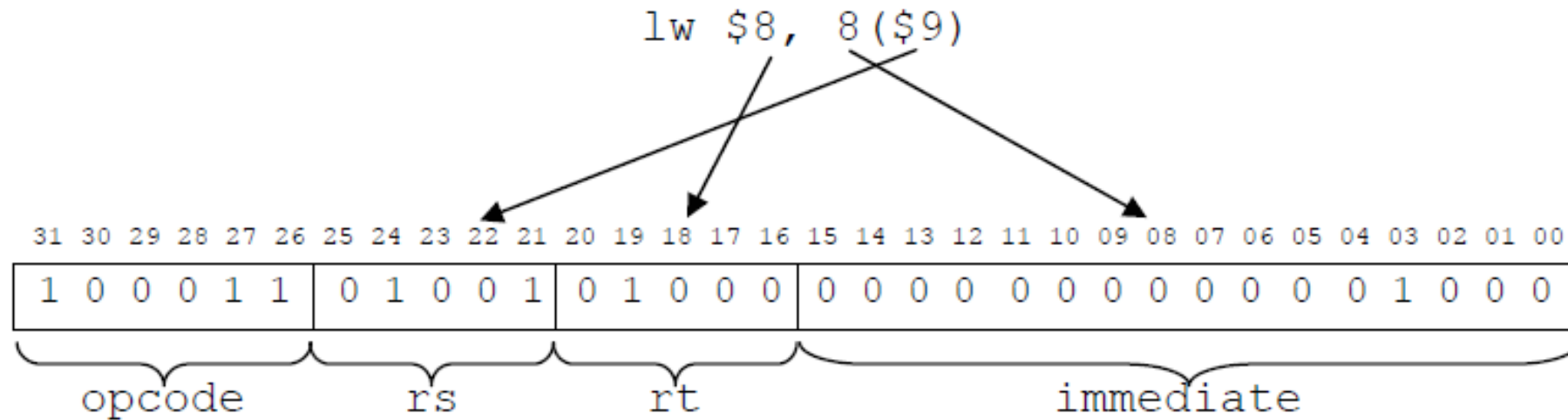
- Instruções **tipo-I**

Nome	Formato	Exemplo	Codificação			
			opcode	rs	rt	immediate
beq	I	beq \$8, \$9, 3	4	8	9	3
bne	I	bne \$8, \$9, 3	5	8	9	3
addi	I	addi \$8, \$9, 3	8	9	8	3
addiu	I	addiu \$8, \$9, 3	9	9	8	3
slti	I	slti \$8, \$9, 3	10	9	8	3
sltiu	I	sltiu \$8, \$9, 3	11	9	8	3
andi	I	andi \$8, \$9, 3	12	9	8	3
ori	I	ori \$8, \$9, 3	13	9	8	3
lui	I	lui \$8, 3	15	0	8	3
lw	I	lw \$8, 4(\$9)	35	9	8	4
sw	I	sw \$8, 4(\$9)	43	9	8	4

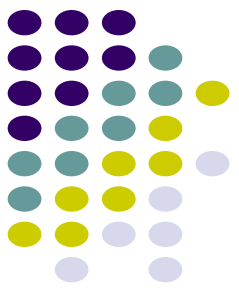
# Representação das instruções em linguagem de máquina.



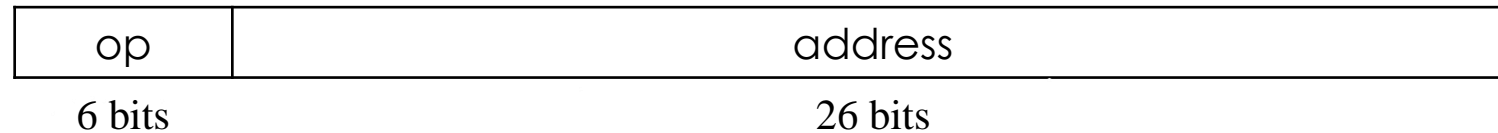
- Instruções LW e SW também são do tipo I



# Representação das instruções em linguagem de máquina.

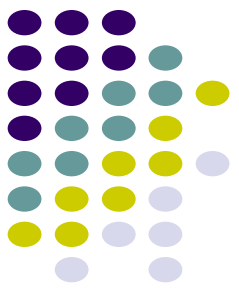


- Campos de uma instrução MIPS:
  - Instruções **tipo-J** (de salto ou jump) ou **formato J**.



- **op**: Operação básica da instrução (opcode)
- **address**: endereço de memória.

# Representação das instruções em linguagem de máquina.

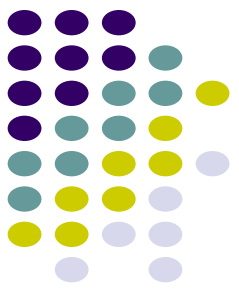


- Instruções **tipo-J**

Nome	Formato	Exemplo	Codificação	
			opcode	endereço
j	J	j 1000	2	1000
jal	J	jal 1000	3	1000

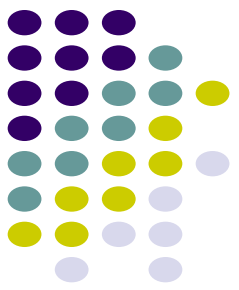


# Representação das instruções em linguagem de máquina.



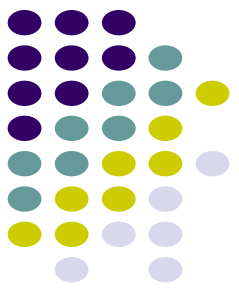
- Embora MIPS possua três diferentes tipos de formato de instruções, percebam que a quantidade de bits é a mesma.
- Isso caracteriza o Princípio de Projeto 4, que diz:
  - “Um bom projeto exige bons compromissos”.

# Endereçamento em desvios condicionais e jumps



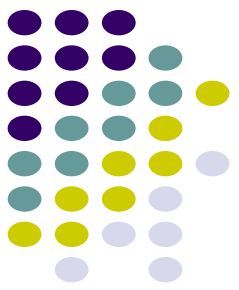
- A instrução de desvio incondicional jump utiliza o terceiro formato de instrução (tipo j).
- Nessas instruções temos 26 bits para especificar o local para onde desejamos desviar.

# Endereçamento em desvios condicionais e jumps



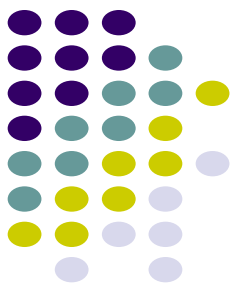
- No entanto, para desvios condicionais temos um problema:
  - Nosso campo de desvio só possui 16 bits.
- Fazendo os cálculos,  $2^{16} = 65536$ ;
- E agora pessoal, será que no MIPS só podemos fazer programas de no máximo 64 kbytes?

# Endereçamento em desvios condicionais e jumps

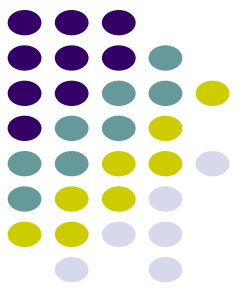


- Para evitar este problema, os processadores trabalham com uma técnica denominada de endereçamento relativo.
- Basicamente a ideia é associar o endereço expresso em uma instrução condicional a um outro elemento (registrador).
  - Qual seria este registrador?
    - O registrador utilizado é o PC. Lembrem-se o PC a todo momento se atualiza, armazenando o endereço da instrução atual, sendo executada.
    - $PC = \text{Registrador} + \text{Endereço de desvio}$

# Endereçamento em desvios condicionais e jumps



- Porque isso resolve?
  - Isso resolve porque, em geral, os endereços de desvio não ficam muito longe dos locais onde é solicitado o desvio.
- Mas e para chamada de funções? Tem algum sentido ela ficar próxima do local do desvio? Como isso é resolvido?
  - Lembrem-se que para funções utilizamos j ou jr que possuem um espaçamento de bits ainda maior (26 bits).
  - Adicionalmente, o MIPS otimiza isso tudo. Seu endereçamento é feito por palavras (endereçamento alinhado), logo o espaço de endereçamento aumenta em mais 4 vezes.



# Endereçamento em desvios condicionais e jumps

- Um loop while foi compilado para o código assembly

```
Loop: sll $t1, $s3, 2      # $t1 = 4 * i
      add $t1, $t1, $s6    # $t1 = endereço de save[i]
      lw $t0, 0($t1)       # $t0 = save[i]
      Bne $t0, $s5, Exit   # vá para exit se save[i] != k
      addi $s3, $s3, 1     # i = i + 1
```





Bne \$t0, \$s5, Exit



j Loop

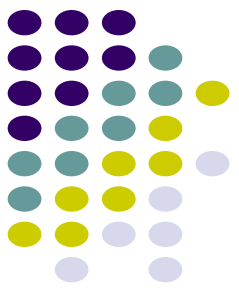
Exit:

Acrescenta  
2 words (8 bytes)

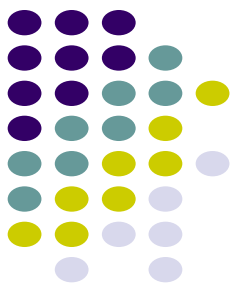
80000	0	0	19	9	4	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
 80012	5	8	21	2		
80016	8	19	19	1		
 80020	2	20000				
80024	...	Utilize a 32-bit address completely (20000*4 = 80000)				

Utiliza o endereço completo ( $20000 * 4 = 80000$ )

# Endereçamento em desvios condicionais e jumps



- Desviando para um lugar mais distante
    - Dada um desvio onde o reg. \$s0 é igual ao reg. \$s1  
    beq    \$s0, \$s1, L1
    - Substitui-o por um par de instruções que ofereça uma distância de desvio muito maior  
    bne    \$s0, \$s1, L2  
    j       L1
- L2:

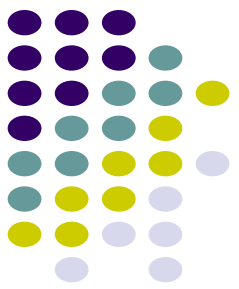


# Formas de endereçamento.

- Endereçamento em Registrador
  - Onde o operando é um registrador
    - add e sub, por exemplo.
- Endereçamento imediato
  - Onde o operando é uma constante dentro da própria instrução
    - addi e subi, por exemplo.
- Endereçamento de base
  - Onde o operando está no local de memória cujo endereço é a soma de um registrador e uma constante.
    - lw, sw
- Endereçamento relativo ao PC
  - Onde o endereçamento é a soma do PC e uma constante na instrução
    - bne, beq, por exemplo.
- Endereçamento pseudodireto.
  - Onde o endereço de jump são os 26 bits da instrução concatenados com os bits mais altos do PC.
    - j e jr.



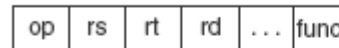
# Formas de endereçamento.



## 1. Endereçamento imediato

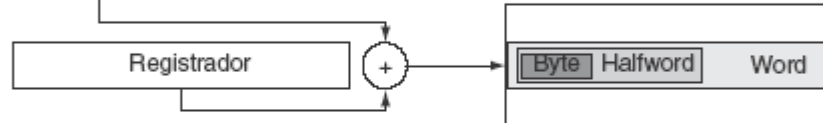
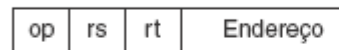


## 2. Endereçamento em registrador

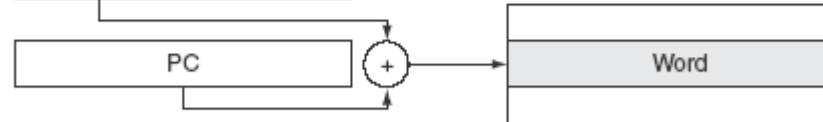
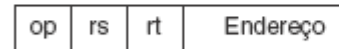


Registradores  
Registrador

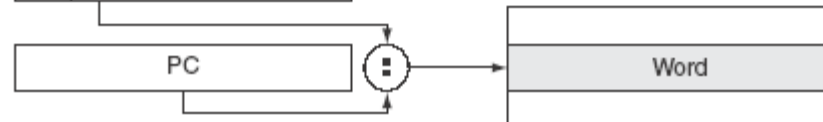
## 3. Endereçamento de base

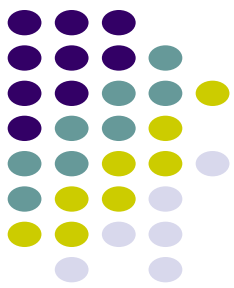


## 4. Endereçamento relativo ao PC



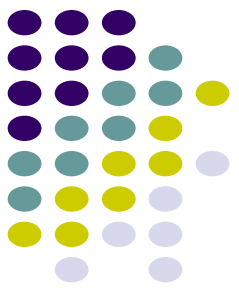
## 5. Endereçamento pseudodireto





# Linguagem de máquina do MIPS

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add \$s1,\$s2,\$s3
sub	R	0	18	19	17	0	34	sub \$s1,\$s2,\$s3
lw	I	35	18	17	100			lw \$s1,100(\$s2)
sw	I	43	18	17	100			sw \$s1,100(\$s2)
and	R	0	18	19	17	0	36	and \$s1,\$s2,\$s3
or	R	0	18	19	17	0	37	or \$s1,\$s2,\$s3
nor	R	0	18	19	17	0	39	nor \$s1,\$s2,\$s3
andi	I	12	18	17	100			andi \$s1,\$s2,100
ori	I	13	18	17	100			ori \$s1,\$s2,100
sll	R	0	0	18	17	10	0	sll \$s1,\$s2,10
srl	R	0	0	18	17	10	2	srl \$s1,\$s2,10
beq	I	4	17	18	25			beq \$s1,\$s2,100
bne	I	5	17	18	25			bne \$s1,\$s2,100
slt	R	0	18	19	17	0	42	slt \$s1,\$s2,\$s3
j	J	2	2500					j 10000 (see Section 2.9)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer, <a href="#">branch</a> format



## Verifique você mesmo

- Que instruções MIPS isto representa? Escolha dentre uma das quatro operações a seguir

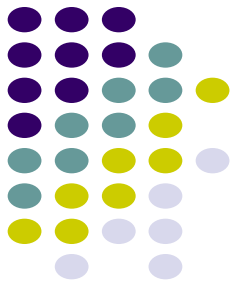
op	rs	rt	rd	shamt	funct
0	8	9	10	0	34

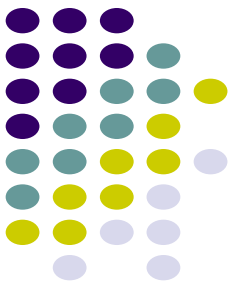
1. add \$t0, \$t1, \$t2
2. add \$t2, \$t01, \$t1
3. add \$t2, \$t1, \$t0
4. sub \$t2, \$t0, \$t1

# Verifique você mesmo

- Resposta:

4. `sub $t2, $t0, $t1`





## Referências

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