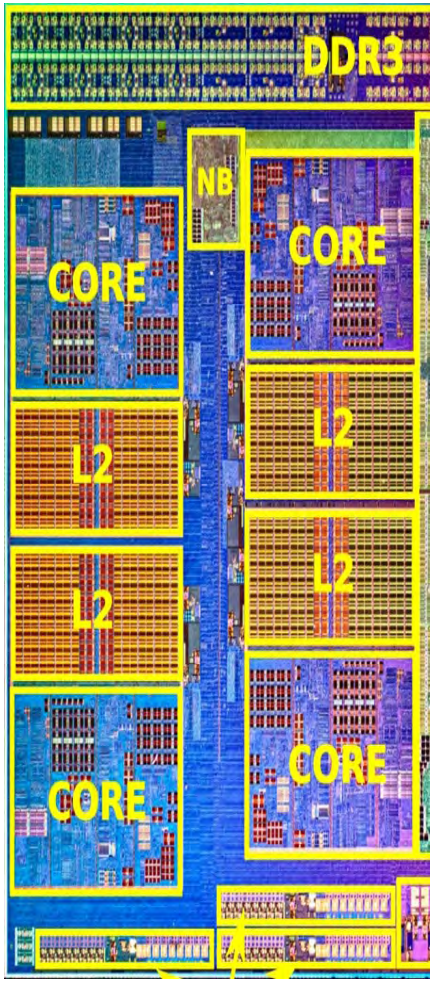
0273359 – Arquitetura e Organização de Computadores 1



Fonte: http://www.techspot.com/article/904-history-of-the-personal-computer-part-5/

Arquitetura MIPS

Luciano de Oliveira Neris
luciano@dc.ufscar.br

Adaptado de slides do prof. Marcio Merino Fernandes

Departamento de Computação
Universidade Federal de São Carlos

MIPS:Histórico

MIPS:Histórico

- MIPS, acrônimo para Microprocessor without Interlocked Pipeline Stages (Microprocessador sem estágios interligados de pipeline)

- Surgiu nos anos 80 a partir de um trabalho realizado por John Hennessy, na Universidade de Stanford.

MIPS:Histórico

- O trabalho tinha como objetivo explorar o padrão RISC, e é ainda considerado o mais elegante neste contexto
- O conceito introduzido por Hennessy foi tão bem sucedido que em 1984 foi formada a MIPS Technologies, Inc, a fim de comercializar os microprocessadores MIPS (R2000)
- Quase 100 milhões de processadores MIPS fabricados em 2009
- Atualmente esta presente em diversos produtos: Utilizada pela NEC, Nintendo, Cisco, Silicon Graphics, Sony, impressoras HP e Fuji, etc.

MIPS:Histórico

- MIPS é bastante utilizado para estudo da arquitetura de processadores por ser de notória simplicidade e clareza

- É a arquitetura base para construção de vários simuladores: R10k, ProcSIM, WinMIPS, MARS, WebSimple, SPIM, SimDE, entre outros.

- *Possui versões de 32 e 64 bits*

ISA MIPS

ISA MIPS

- A arquitetura MIPS é baseada em registrador, ou seja, a CPU utiliza apenas registradores para realizar as suas operações aritméticas e lógicas

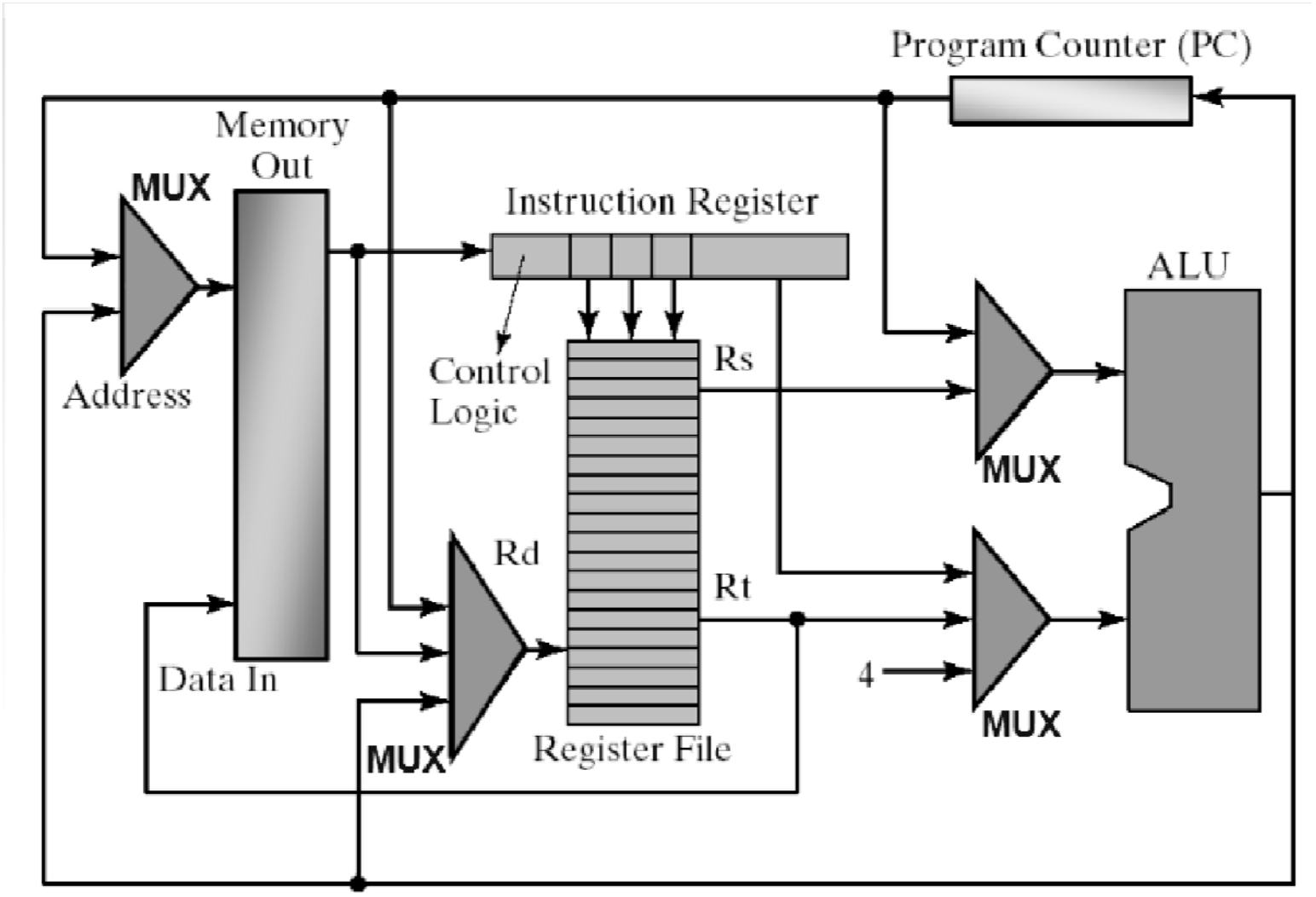
- É uma arquitetura do tipo load – store: as operações lógicas e aritméticas são executadas exclusivamente entre registradores da arquitetura ou entre constantes imediatas e registradores

- As operações de acesso à memória só executam ou uma leitura da memória (load) ou uma escrita na memória (store)

- O processador disponibiliza um conjunto relativamente grande de registradores para reduzir o número de acessos à memória externa

- As instruções possuem poucos formatos e são do mesmo tamanho

ISA MIPS



Fonte: Prof. Dr.-Ing. Frank Sill Torres http://www.cpdee.ufmg.br/~frank/

ISA MIPS

- Possui várias características/restrições
  - Todas as instruções possuem 32 bits: nenhuma instrução utiliza apenas dois ou três bytes de memória e nenhuma instrução pode ser maior do que 4 bytes
  - Todas as instruções possuem opcode de 6 bits
  - Todas as instruções aritméticas possuem 3 operandos
  - Bi-endian

ISA MIPS: Registradores

- MIPS possui um banco de 32 registradores
  - Cada registrador comporta valores de 32 bits
  - Os registradores são identificados unicamente através do símbolo do cifrão ($) seguido de um identificador
  - Adota-se uma convenção que especifica quais registradores devem ser utilizados em certas circunstâncias
  - Os registradores PC (contador de programas) e IR (registrador de instrução) não fazem parte do banco de registradores

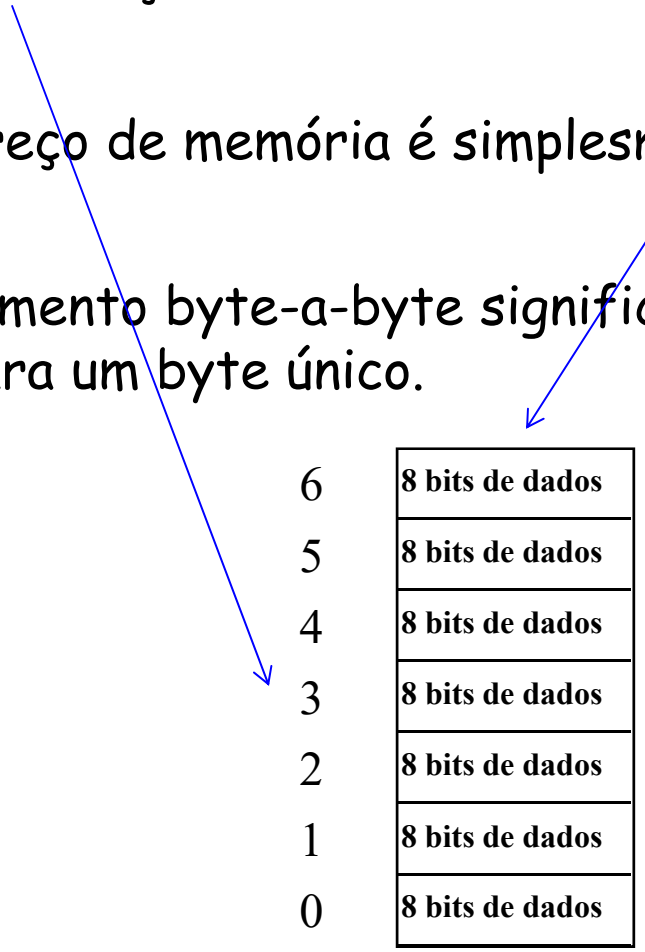
ISA MIPS: Registradores

#Registrador Físico	Nome Registrador no pgm MIPS	Convenção de uso (software)
$0	**$zero**	Constante = 0
$1	**$at**	Reservado p/ assembler
$2 - $3	**$v0 - $v1**	Retorna resultado de funções
$4 - $7	**$a0 - $a3**	Argumentos p/ funções
$8 - $15	**$t0 - $t7**	Temporários, (não preservados entre chamadas de funções)
$16 - $23	**$s0 - $s7**	Temporários, (preservados entre chamadas de funções)
$24 - $25	**$t8 - $t9**	Temporários, (não preservados entre chamadas de funções)
$26 - $27	**$k0 - $k1**	Reservado p/ OS kernel
$28	**$gp**	Global pointer
$29	**$sp**	Stack pointer
$30	**$fp**	Frame pointer
$31	**$ra**	Endereço de retorno de função (salvo na pilha pela instrução call)
$hi	**$hi**	Bits mais altos de resultado de 64 bits (remainder/div, high word/mult)
$lo	**$lo**	Bits mais baixos de resultado de 64 bits (quotient/div, low word/mult)

Organização da Memória

MIPS: Memória

- A memória pode ser visa como um grande vetor de bytes, cada um com um endereço hexadecimal.
- Um endereço de memória é simplesmente um índice desse vetor.
- Endereçamento byte-a-byte significa que cada índice do vetor aponta para um byte único.



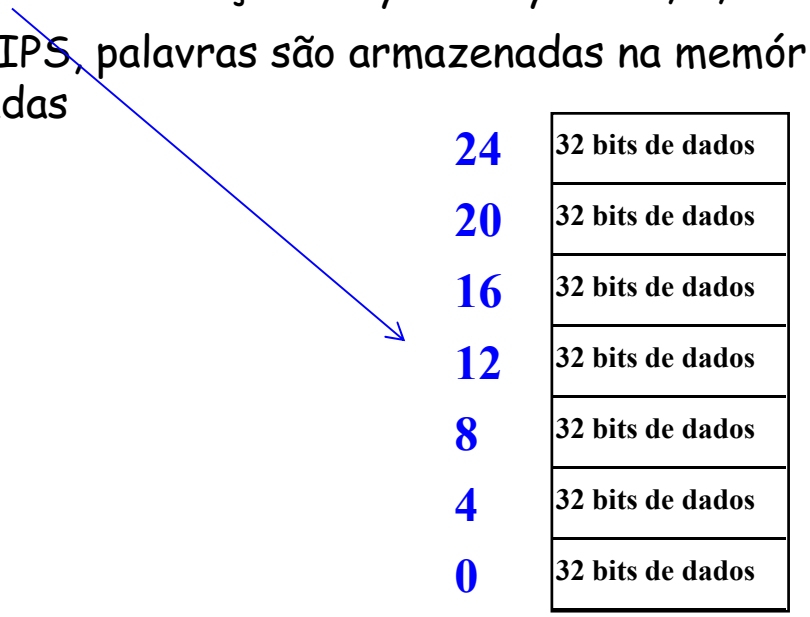
MIPS: Memória

- A maior parte dos dados utilizados em programas são maiores do que bytes.
- MIPS fornece as instruções lw/lh/lb and sw/sh/sb
  - w= word (32 bits), h= half-word (16 bits), b= byte (8 bits)

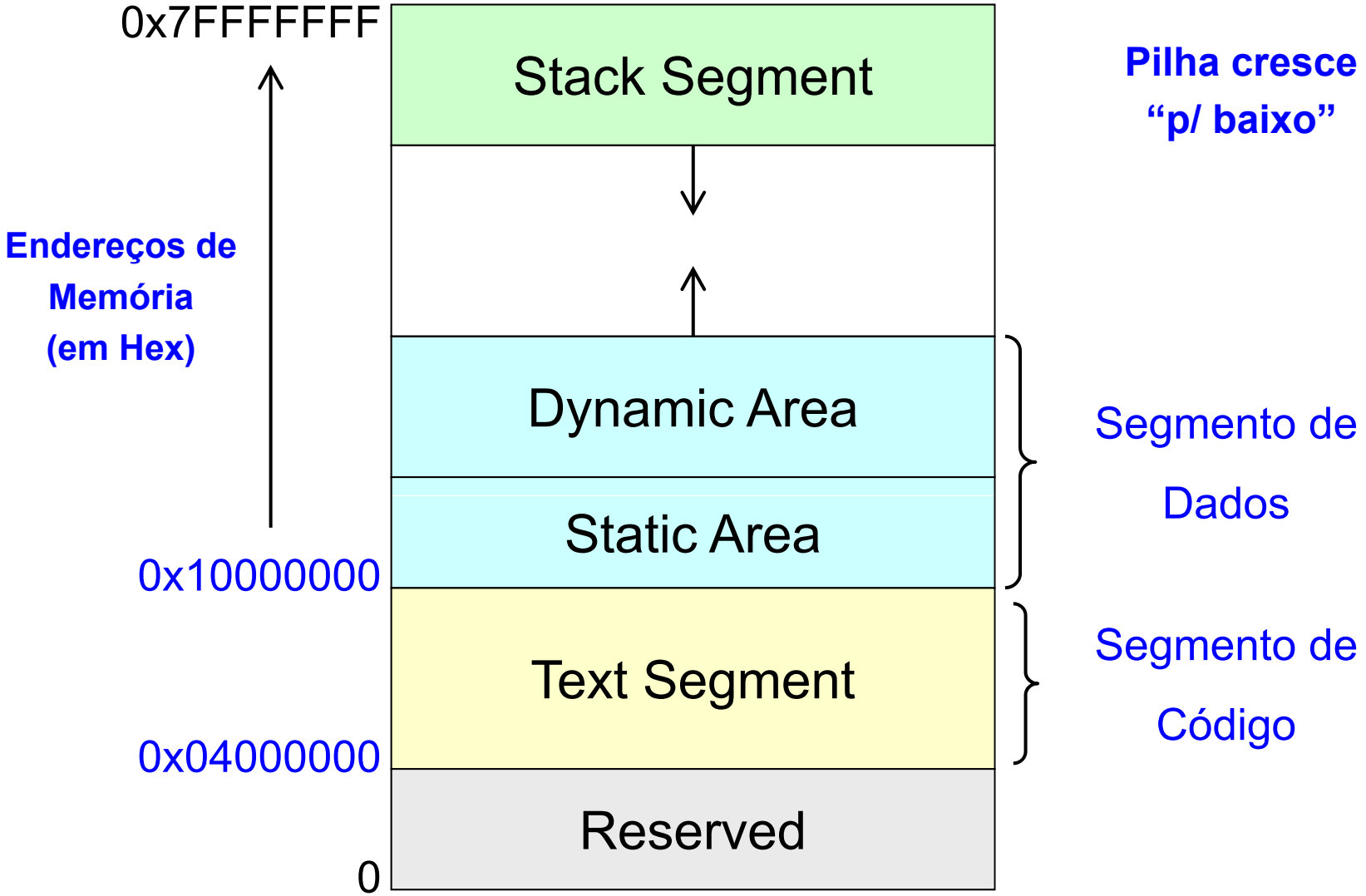
2^{32} bytes endereçados byte-a-byte = 0, 1, 2….2^{32}-1 endereços = 4GByte

2^{30} words endereçados byte-a-byte = 0, 4, 8 … 2^{30}-4 endereços= 1GWord

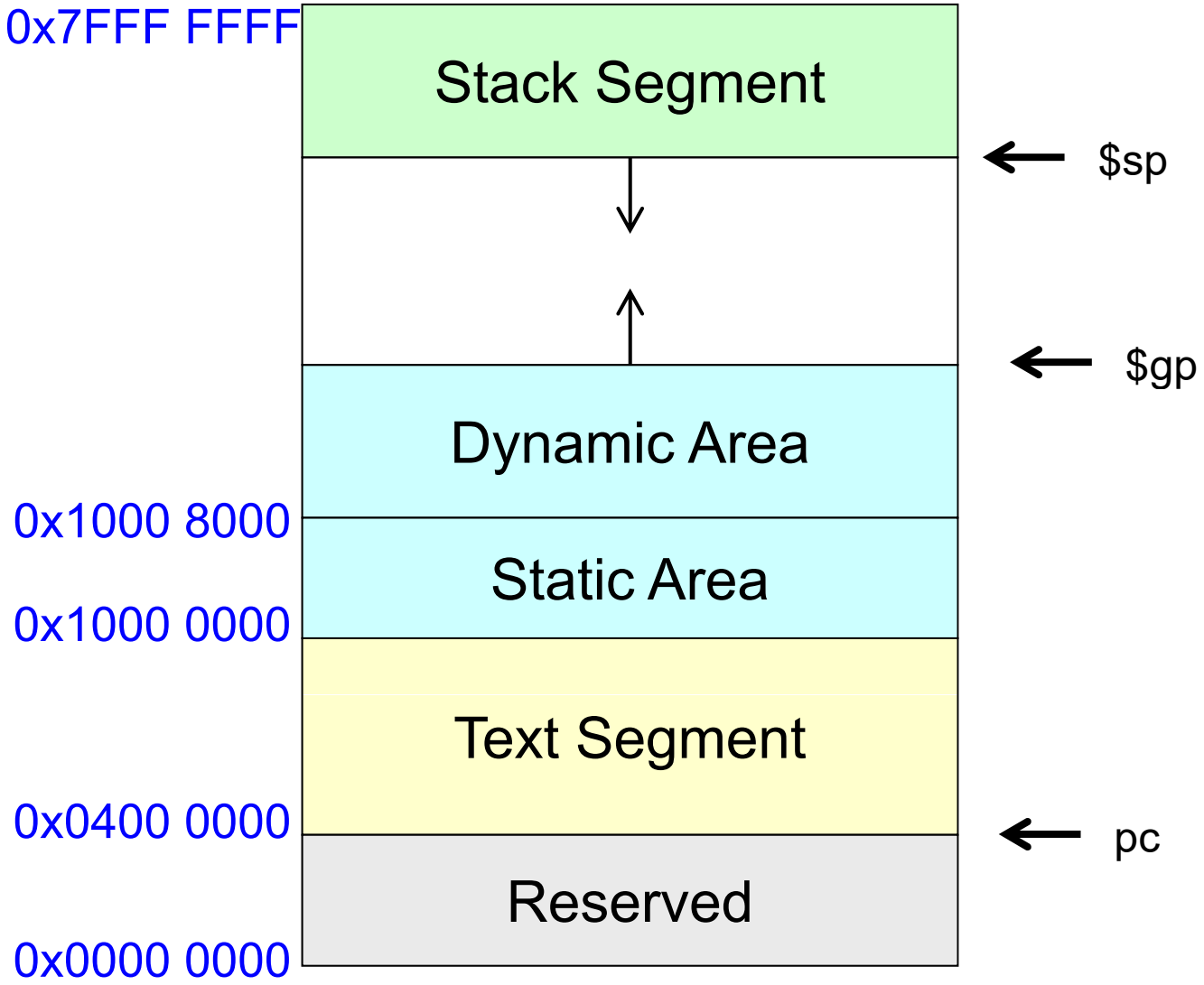
Em MIPS, palavras são armazenadas na memória em BIG-ENDIAN. Palavras são alinhadas



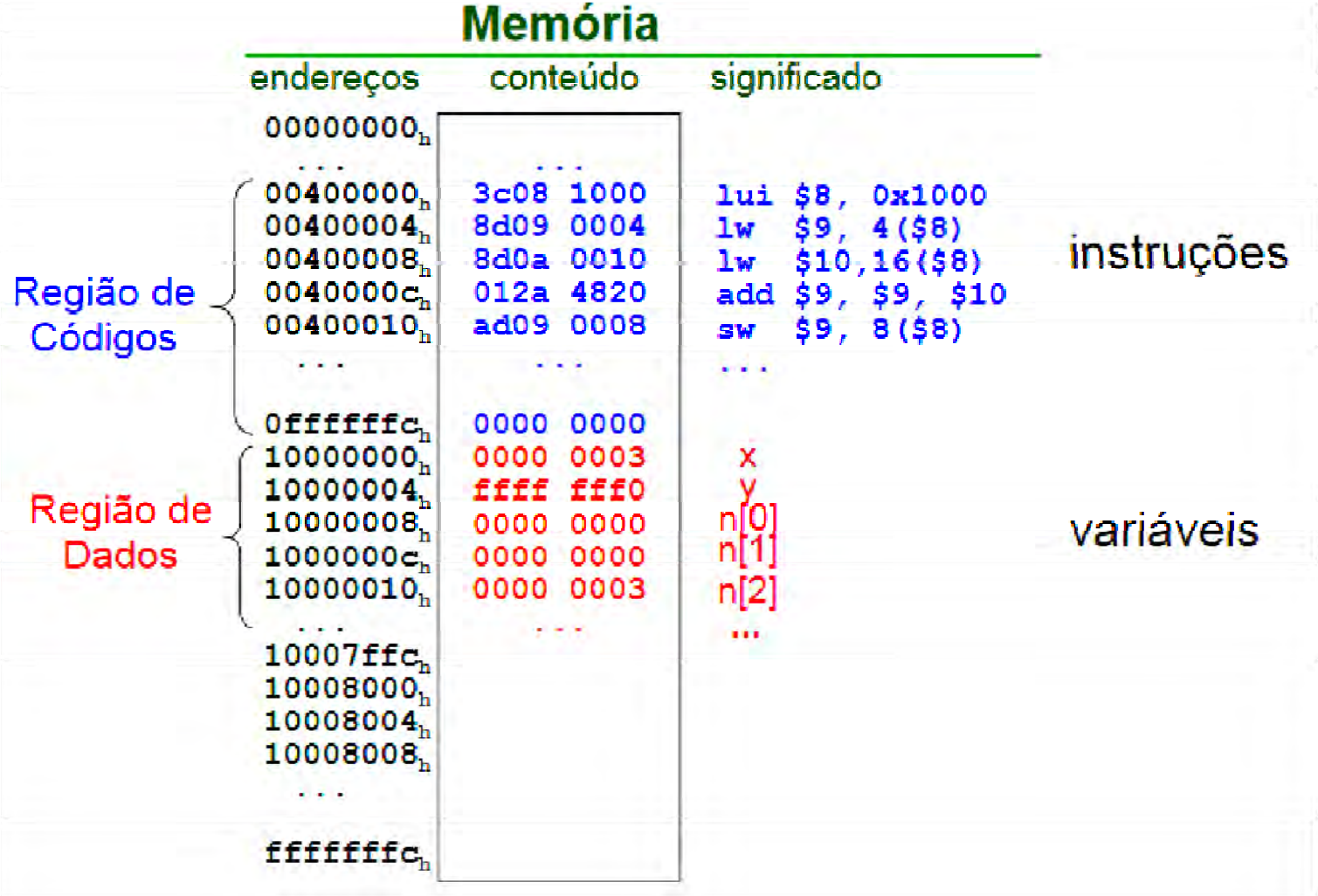
MIPS: Memória



MIPS: Memória

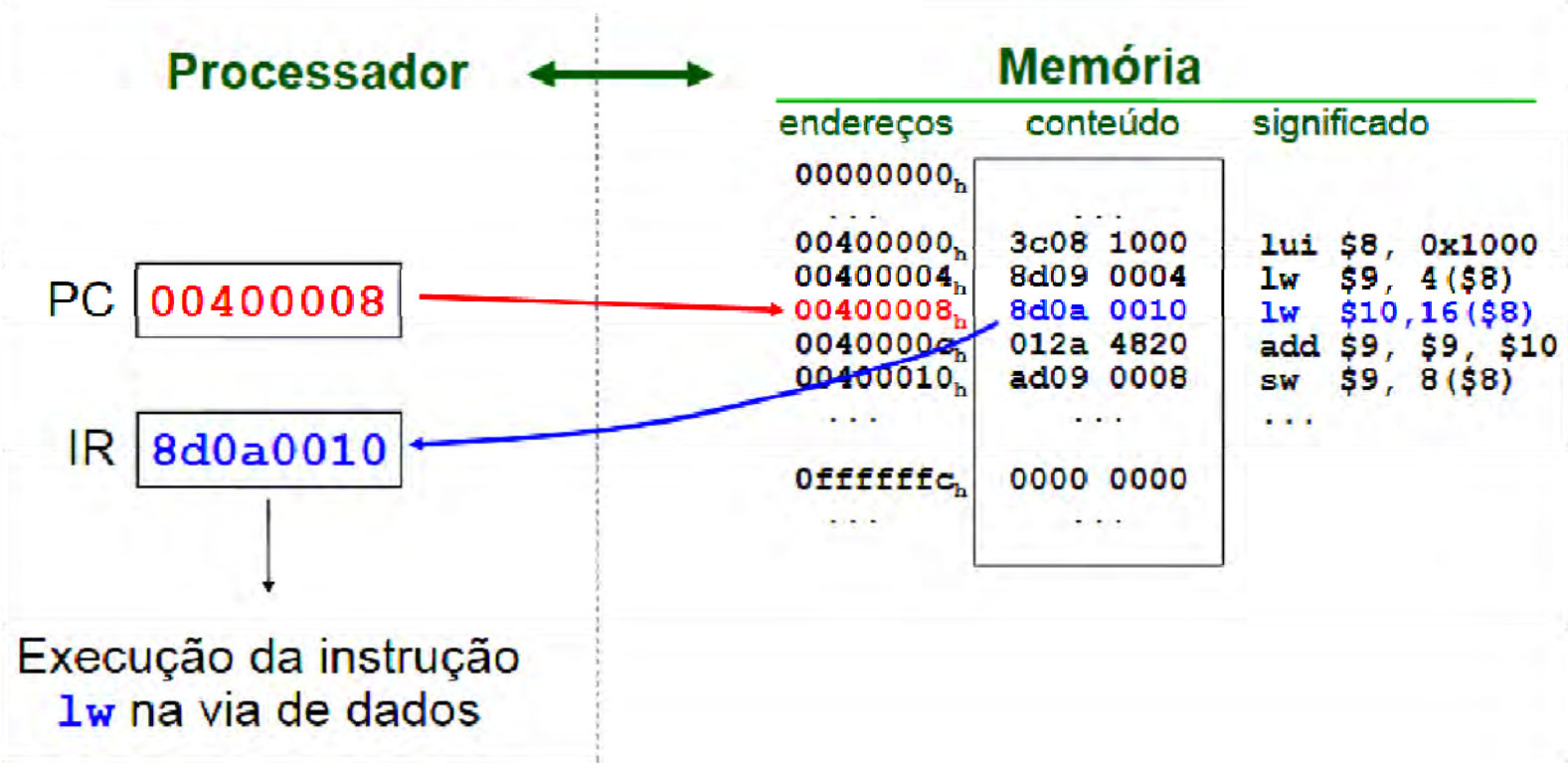


MIPS: Memória



Fonte: Slides do Prof. Frank Sill Torres DELT – Escola de Engenharia UFMG

MIPS: Memória



Fonte: Slides do Prof. Frank Sill Torres DELT – Escola de Engenharia UFMG

Formato de Instruções

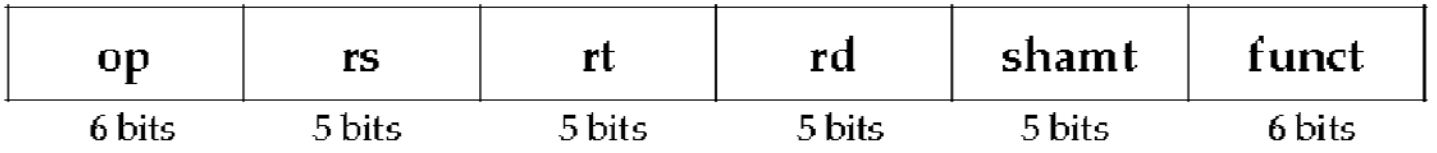
MIPS: Formato de Instruções

- Instruções
  - Todas as instruções possuem 32 bits
  - Todas as instruções possuem opcode de 6 bits
  - O modo de endereçamento é codificado no opcode

- As instruções podem ser classificadas em:
  - R-Type
  - I-Type
  - J-Type

MIPS: Formato de Instruções

- R-Type
  - Uso de registradores na instrução
  - Instruções registrador-registrador
  - As instruções do tipo R possuem opcode igual a 0.



op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

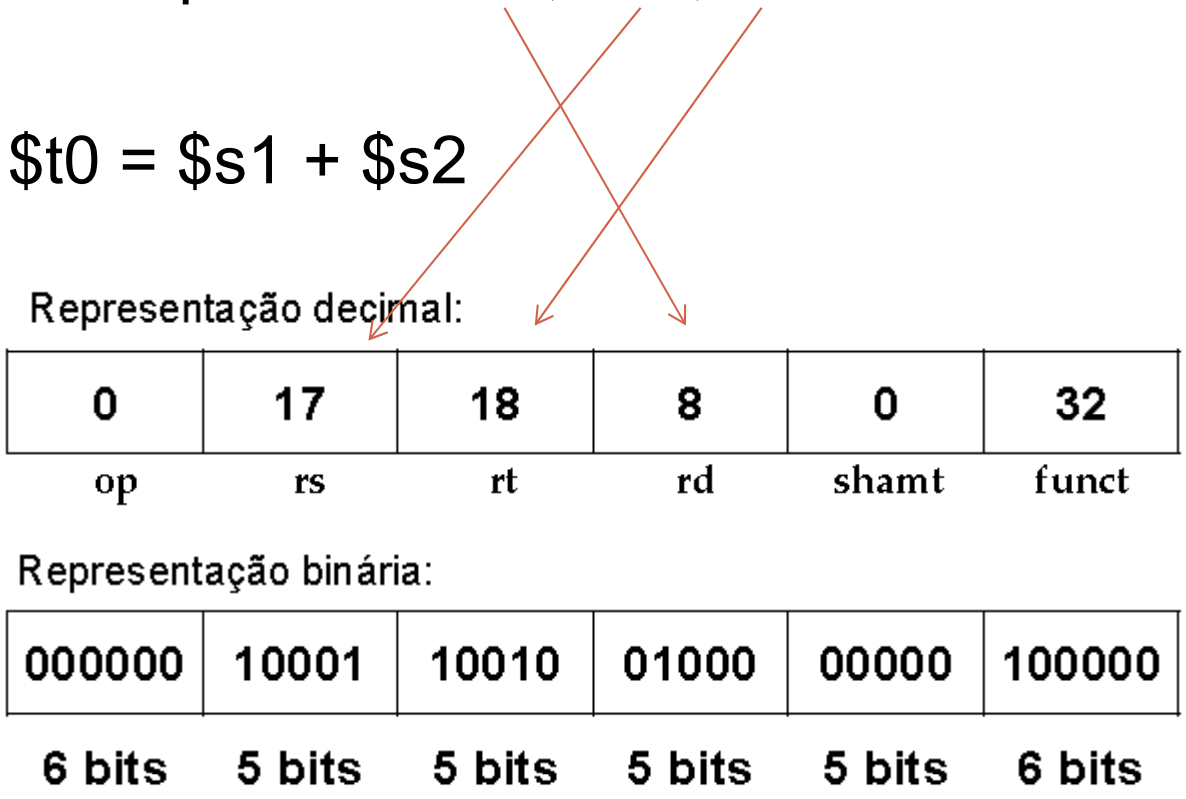
op – *opcode*: identifica a operação básica.
rs – primeiro operando (registrador).
rt – segundo operando (registrador).
rd – registrador de destino (resultado).
shamt – *shift amount*: define o tamanho do deslocamento.
funct – *function*: especifica a versão ou variante da operação indicada em op.

Formato R

MIPS: Formato de Instruções

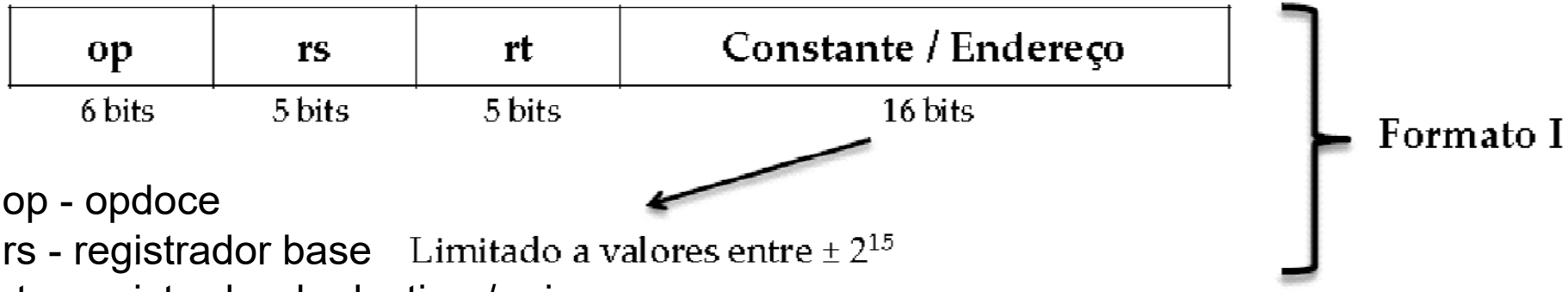
- R-Type
  - Exemplo: add $t0, $s1, $s2

$t0 = $s1 + $s2



MIPS: Formato de Instruções

- I-Type
  - Instruções envolvendo valor imediato
  - Um dos operandos pode ser representado na forma de um valor constante ou o endereço da palavra a ser acessada mantido dentro da própria instrução



MIPS: Formato de Instruções

- I-Type:
  - Exemplo: Transferência de Dados

A[300] = h + A[300]

↓

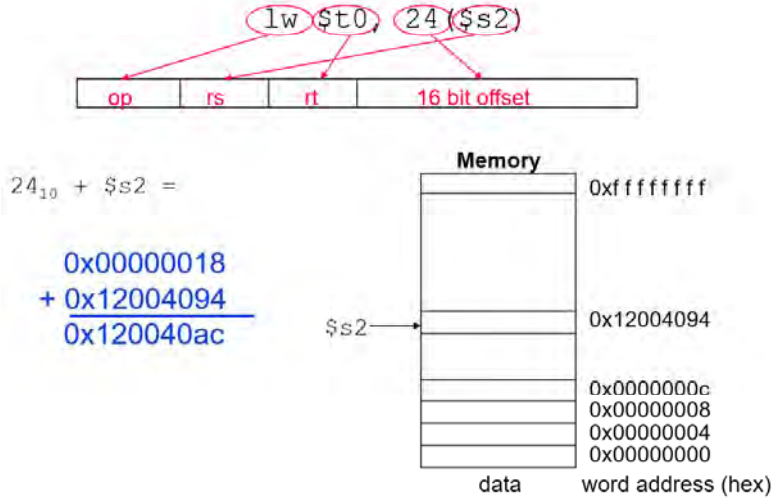
lw $t0, 1200 ($t1) # $t0 = A[300]
add $t0, $s2, $t0   # $t0 = h + A[300]
sw $t0, 1200 ($t1) # A[300] = $t0

↓

op rs rt endereço
35 9 8 1200

op rs rt rd shmat func
0   18 8 8   0   32

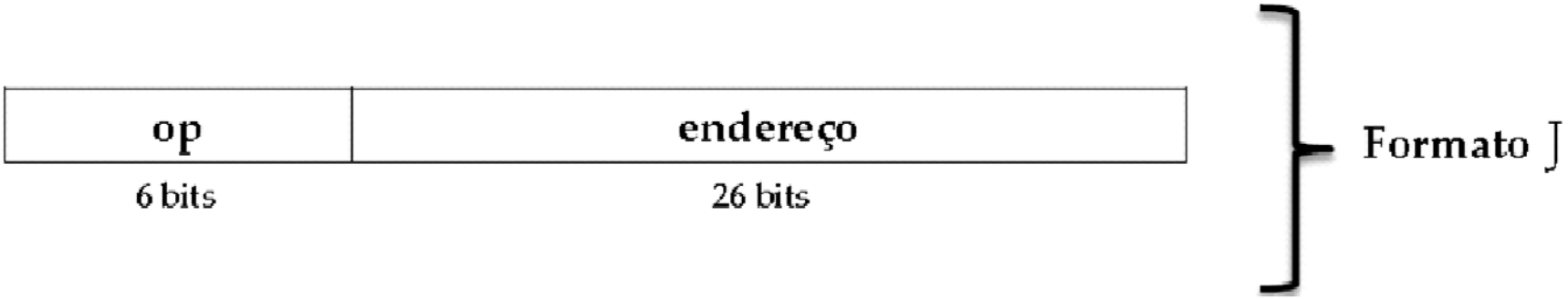
op rs rt endereço
43 9   8 1200



100011	01001	01000	0000 0100 1011 0000
000000	10010	01000	01000 00000 100000
101011	01001	01000	0000 0100 1011 0000

MIPS: Formato de Instruções

- J-Type
  - Instruções de desvio incondicional (jump)
  - O desvio sempre é realizado

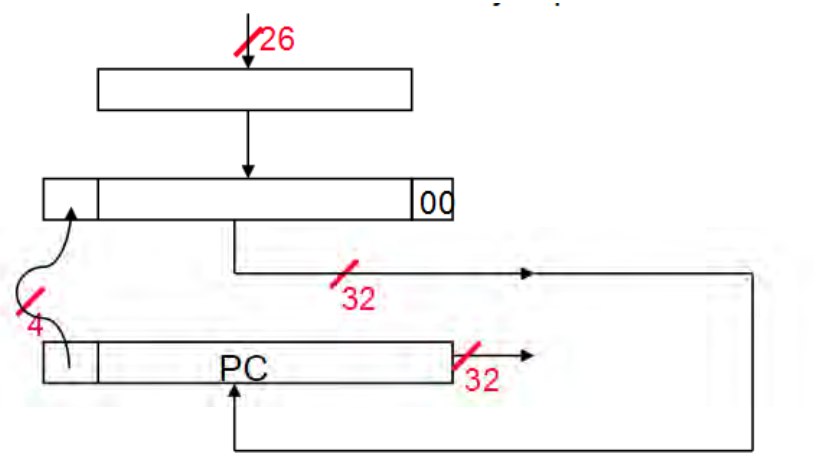


op - opcoce
endereço (target) - local da memória a saltar (onde esta a próxima instrução a ser executada).

MIPS: Formato de Instruções

- J-Type
  - Exemplo

```
            add $s0, $s1, $s2
            j Exit
            sub $s0, $s1, $s2
Exit:
```



MIPS: Formato de Instruções

- Machine Codes

Mnemonic	Meaning	Type	Opcode	Funct
`add`	Add	R	0x00	0x20
`addu`	Add Unsigned	R	0x00	0x21
`and`	Bitwise AND	R	0x00	0x24
`div`	Divide	R	0x00	0x1A
`divu`	Unsigned Divide	R	0x00	0x1B
`jr`	Jump to Address in Register	R	0x00	0x08
`mfhi`	Move from HI Register	R	0x00	0x10
`mflo`	Move from LO Register	R	0x00	0x12
`mfc0`	Move from Coprocessor 0	R	0x10	NA
`mult`	Multiply	R	0x00	0x18
`multu`	Unsigned Multiply	R	0x00	0x19
`nor`	Bitwise NOR (NOT-OR)	R	0x00	0x27
`xor`	Bitwise XOR (Exclusive-OR)	R	0x00	0x26
`or`	Bitwise OR	R	0x00	0x25
`slt`	Set to 1 if Less Than	R	0x00	0x2A
`sltu`	Set to 1 if Less Than Unsigned	R	0x00	0x2B
`sll`	Logical Shift Left	R	0x00	0x00
`srl`	Logical Shift Right (0-extended)	R	0x00	0x02

Mnemonic	Meaning	Type	Opcode	Funct
`sra`	Arithmetic Shift Right (sign-extended)	R	0x00	0x03
`sub`	Subtract	R	0x00	0x22
`subu`	Unsigned Subtract	R	0x00	0x23
`j`	Jump to Address	J	0x02	NA
`jal`	Jump and Link	J	0x03	NA
`addi`	Add Immediate	I	0x08	NA
`addiu`	Add Unsigned Immediate	I	0x09	NA
`andi`	Bitwise AND Immediate	I	0x0C	NA
`beq`	Branch if Equal	I	0x04	NA
`bne`	Branch if Not Equal	I	0x05	NA
`lbu`	Load Byte Unsigned	I	0x24	NA
`lhu`	Load Halfword Unsigned	I	0x25	NA
`lui`	Load Upper Immediate	I	0x0F	NA
`lw`	Load Word	I	0x23	NA
`ori`	Bitwise OR Immediate	I	0x0D	NA
`sb`	Store Byte	I	0x28	NA
`sh`	Store Halfword	I	0x29	NA
`slti`	Set to 1 if Less Than Immediate	I	0x0A	NA
`sltiu`	Set to 1 if Less Than Unsigned Immediate	I	0x0B	NA
`sw`	Store Word	I	0x2B	NA

https://en.wikibooks.org/wiki/MIPS_Assembly/Instruction_Formats

MIPS: Princípios de Projeto

- Simplicidade favorece a regularidade
  - -Instruções fixas de 32 bits
  - -Pequeno número de formato de instruções
  - -O opcode está sempre nos 6 bits mais significativos
- Bons projetos requerem compromissos
  - -3 formatos de instruções
- •Menor é melhor
  - -Conjunto de instruções limitado
  - -Número de registradores no banco de registradores limitado
  - -Número de modos de endereçamento limitado
- Torne o caso comum rápido
  - -Operandos aritméticos estão no banco de registradores (máquina load-store)
  - -Permitir que as instruções tenham operandos imediatos

Instruções

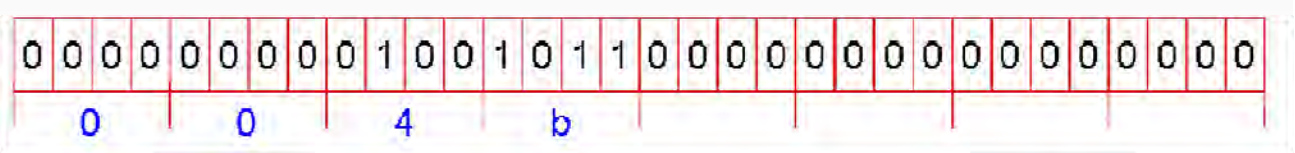
MIPS: Instruções

- Transferência de Dados
  - Como carregar constantes de 32 bits?
    - Utilizar a instrução lui (*load upper immediate*) para os MSBs
    - Utilizar a instrução ori (*or immediate*) para os LSBs

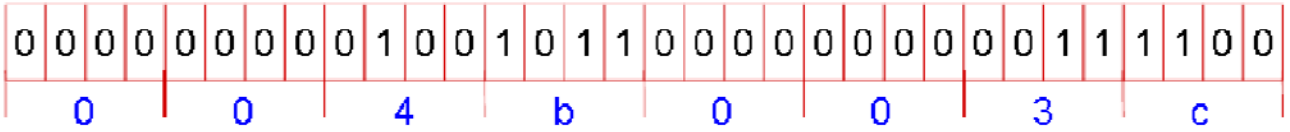
MIPS: Instruções

- Transferência de Dados
  - Ex: $t0 0x004b003c

lui $t0, 0x004b



ori $t0,$t0, 0x003c



MIPS: Instruções

- Desvio condicional (branch)
  - Em linguagens de programação de alto nível o comando de tomada de decisão é geralmente implementado através do comando if
  - O MIPS possui duas principas instruções de desvio:
    - beq r1, r2, L1 (branch if equal)
      - se os valores de r1 e r2 são iguais, desvia para a linha identificada pelo label L1
    - bne r1, r2, L1 (branch if not equal)
      - se os valores de r1 e r2 são diferentes, desvia para a linha identificada pelo label L1

if cond then goto label

MIPS: Instruções

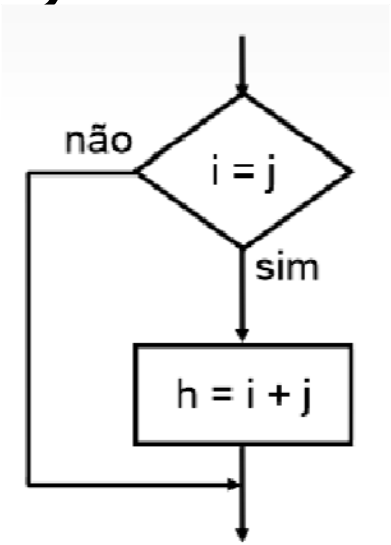
- Desvio condicional (branch)

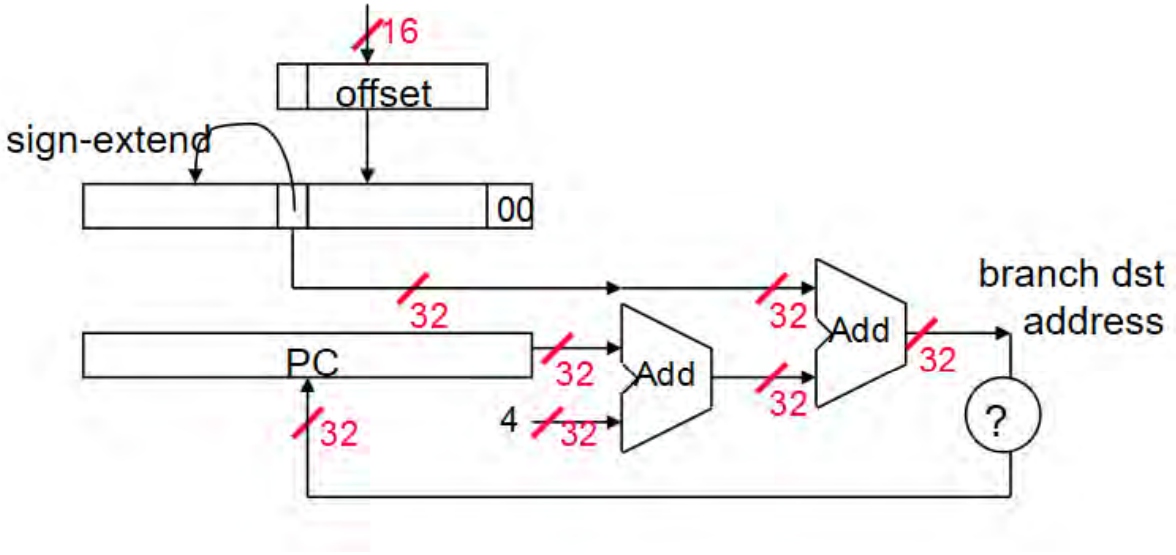
```
beqz $s0, label          #if $s0==0 goto label

bnez $s0, label          #if $s0!=0 goto label

bge  $s0, $s1, label     #if $s0>=$s1 goto label

ble  $s0, $s1, label     #if $s0<=$s1 goto label

blt  $s0, $s1, label     #if $s0<$s1 goto label

beq  $s0, $s1, label     #if $s0==$s1 goto label

bne  $s0, $s1, label     #if $s0!=$s1 goto label

bgez $s0, label          #if $s0>=0 goto label
```

MIPS: Instruções

- Desvio condicional (branch)

```
        bne $8, $9, exit
        add $10, $8, $9
exit: ...
```

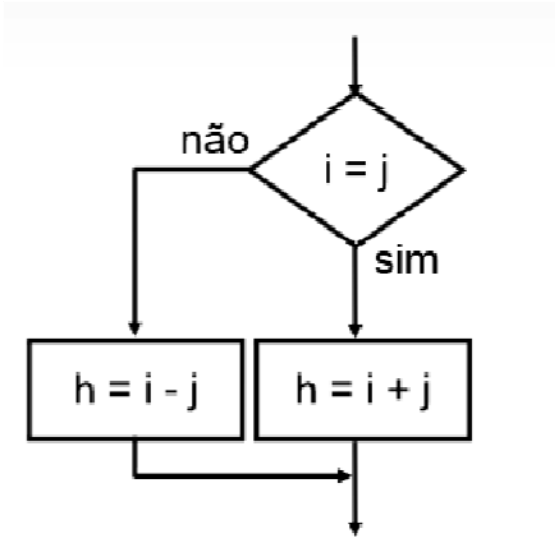




MIPS: Instruções

- Desvio condicional (branch)

```
        bne  $8, $9, else
        add  $10, $8, $9
        j exit
else: sub  $10, $8, $9
exit : ...
```



MIPS: Instruções

- Instrução de Comparação
  - Permite testar se uma variável é menor do que outra
  - Compara o conteúdo de dois registradores e atribui o valor 1 a um terceiro registrador se primeiro < segundo. Caso contrário, é atribuído o valor 0 ao terceiro registrador.
  - slt (set on less than)

```
slt $t0, $s3, $s4        # $t0 recebe 1 se $s3 < $s4
                         # $t0 recebe 0 se $s3 ≥ $s4
```

MIPS: Instruções

- Instruções de Controle
  - Com instruções simples como Branch e Jump é possível construir estruturas como:
    - Loops (For, While, Repeat Until)
    - If-Then-Else
    - Chamadas de funções

MIPS: Instruções

- Instruções de Controle

```
if ($t0==$t1)
then
    $t2= 25
else
    $t2= 77
$t3 = $t3 + $t2
```

```
            beq $t0, $t1, blockA
            j blocoB
blocoA:     li $t2, 25
            j exit
blocoB:     li $t2, 77
exit:       addu $t3, $t3, $t2
```

MIPS: Instruções

- Instruções de Controle

repeat … until $t0>$t1
  t0= t0 + 1

```
loop1:
      addi $t0, $t0, 1
      ble $t0, $t1, loop1   # if $t0<=$t1 goto loop1
```

MIPS: Instruções

- Instruções de Controle

```
switch (k) {
    case 0: f = i + j; break;
    case 1: f = g + h; break;
    case 2: f = g - h; break;
    case 3: f = i - j;
}
```

```
        slt $t3, $s5, $zero
        bne $t3, $zero, Exit
        slt $t3, $s5, $t2
        beq $t3, $zero, Exit
        add $t1, $s5, $s5
        add $t1, $t1, $t1
        add $t1, $t1, $t4
        lw $t0, 0 ($t1)
        jr $t0
L0:     add $s0, $s3, $s4
        J Exit
L1:     add $s0, $s1, $s2
        j Exit
L2:     sub $s0, $s1, $s2
        j Exit
L3:     sub $s0, $s3, $s4
Exit:
```

Sub-rotina

MIPS: Sub-rotina

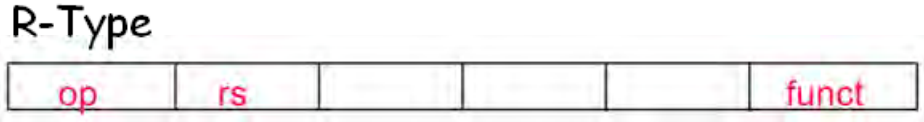
- Passos pra a execução
  1. Passagem de parâmetros
  2. Transferência do controle de execução
  3. Aquisição dos recursos de armazenamento necessários para o procedimento
  4. Realização da tarefa desejada.
  5. Armazenamento do resultado em local acessível ao programa que chamou a sub-rotina.
  6. Retorno à execução do programa original.

MIPS: Sub-rotina

- Convenção
  - $a0 - $a3: quatro registradores para a passagem de parâmetros.
  - $v0 - $v1: dois registradores para retorno de resultados.
  - $ra: registrador de endereço de retorno.
  - $t0 – t9: 10 registradores temporários.

- Instruções adicionais
  - jal ROTINA
    - Desvia a execução para o endereço indicado por ROTINA e automaticamente salva o endereço de retorno (PC + 4) em $ra.

  - jr $ra
    - Desvia para a posição de memória indicada



MIPS: Sub-rotina

- Convenção
  - Se forem necessários mais que quatro argumentos e/ou dois resultados:
    - $t0-$t9
      - Dez registradores temporários que NÃO SÃO preservados pelo chamador
    - $s0-$s7
      - Oito registradores que DEVEM ser preservados se utilizados no procedimento chamado

MIPS: Sub-rotina

- Convenção
  - Registradores "caller-saved": $t0-$t9
    - Chamador é responsável por salvá-los em memória caso seja necessário usá-los novamente depois que um procedimento é chamado
  - Registradores "callee-saved": $s0-$s7
    - Chamado é responsável por salvá-los em memória antes de utilizá-los e restaurá-los antes de devolver o controle ao chamador

MIPS: Sub-rotina

```
        li $a0,10
        li $a1,21
        li $a3,31
        jal silly       #Now the result of the function is in $v0.
        li $v0,10       #syscall code 10 is for exit.
        Syscall         #exit the program gracefully
silly: add $t0,$a0,$a1
        sub $v0,$a3,$t0
        jr $ra
```

Pilha

MIPS: Pilha

- Preservação de Contexto
  - Quando as chamadas a procedimentos forem recursivas ou aninhadas, é conveniente o uso de uma pilha
  - Qualquer registrador usado pelo chamador deve ter seu conteúdo restaurado para o valor original antes da chamada
  - O conteúdo dos registradores é salvo na memória. Depois da execução do procedimento, estes registradores devem ter seus valores restaurados

MIPS: Pilha

- Preservação de Contexto
  - $sp (stack pointer) - registrador utilizado para guardar o endereço do topo da pilha da chamada de procedimentos:
    - a posição de memória que contêm os valores dos registradores salvos na memória pela última chamada
    - a posição a partir da qual a próxima chamada de procedimento pode salvar seus registradores

MIPS: Pilha

- Manipulando a Pilha
  - MIPS não possui instruções específicas para manipular a pilha
  - As instruções lw e sw devem ser utilizadas para essa função
    - Ex.: inserir o conteúdo do registrador $s0 na pilha

```
addi  $sp, $sp, -4   # avança o stack pointer
sw    $s0, 0($sp)    # empilha o valor de $s0
```

    - Ex.: remover o topo da pilha e armazenar em $s0

```
lw    $s0, 0($sp)   # restaura $s0
addi  $sp,$sp,4     # desempilha o topo
```

Chamdas de Sistema

MIPS: Chamadas de Sistema

- Syscall
  - Instrução especial que suspende a execução do programa do usuário e transfere o controle para o sistema operacional
  - O sistema operacional acessa o conteúdo do registrador $V0 para determinar qual tarefa esta sendo solicitada e retorna o controle ao programa do usuário quando a tarefa é realizada
  - Possibilita chamar serviços fornecidos pelo sistema operacional como a leitura do teclado, a exibição de dados na tela, operações de acesso a disco, rede, etc…

MIPS: Chamadas de Sistema

- Syscall

Função	$v0	Arguments / Result
Print Integer	1	$a0 = integer value to print
Print Float	2	$f12 = float value to print
Print Double	3	$f12 = double value to print
Print String	4	$a0 = address of null-terminated string
Read Integer	5	$v0 = integer read
Read Float	6	$f0 = float read
Read Double	7	$f0 = double read
Read String	8	$a0 = address of input buffer $a1 = maximum number of characters to read
Exit Program	10	
Print Char	11	$a0 = character to print
Read Char	12	$a0 = character read

MIPS: Chamadas de Sistema

- Syscall

a) Carregar argumentos das funções em registradores pré-definidos
b) Carregar código das funções no registrador $v0
c) Executar a função syscall

```
a) li  $a0, 10      # argumento: $a0=10
b) li  $v0, 1       # código da função "print integer"
c) syscall          # exibe na tela conteúdo de $a0 (10)
```

MIPS: Chamadas de Sistema

- Syscall
  - Finalizar a execução do programa

```
li $v0, 10          # syscall code 10 is for exit.
syscall             # make the syscall.
```

  - Leitura de inteiros

```
li   $v0, 5         # load syscall read_int into $v0
syscall             # make the syscall
move $t0, $v0       # move the number read into $t0
```

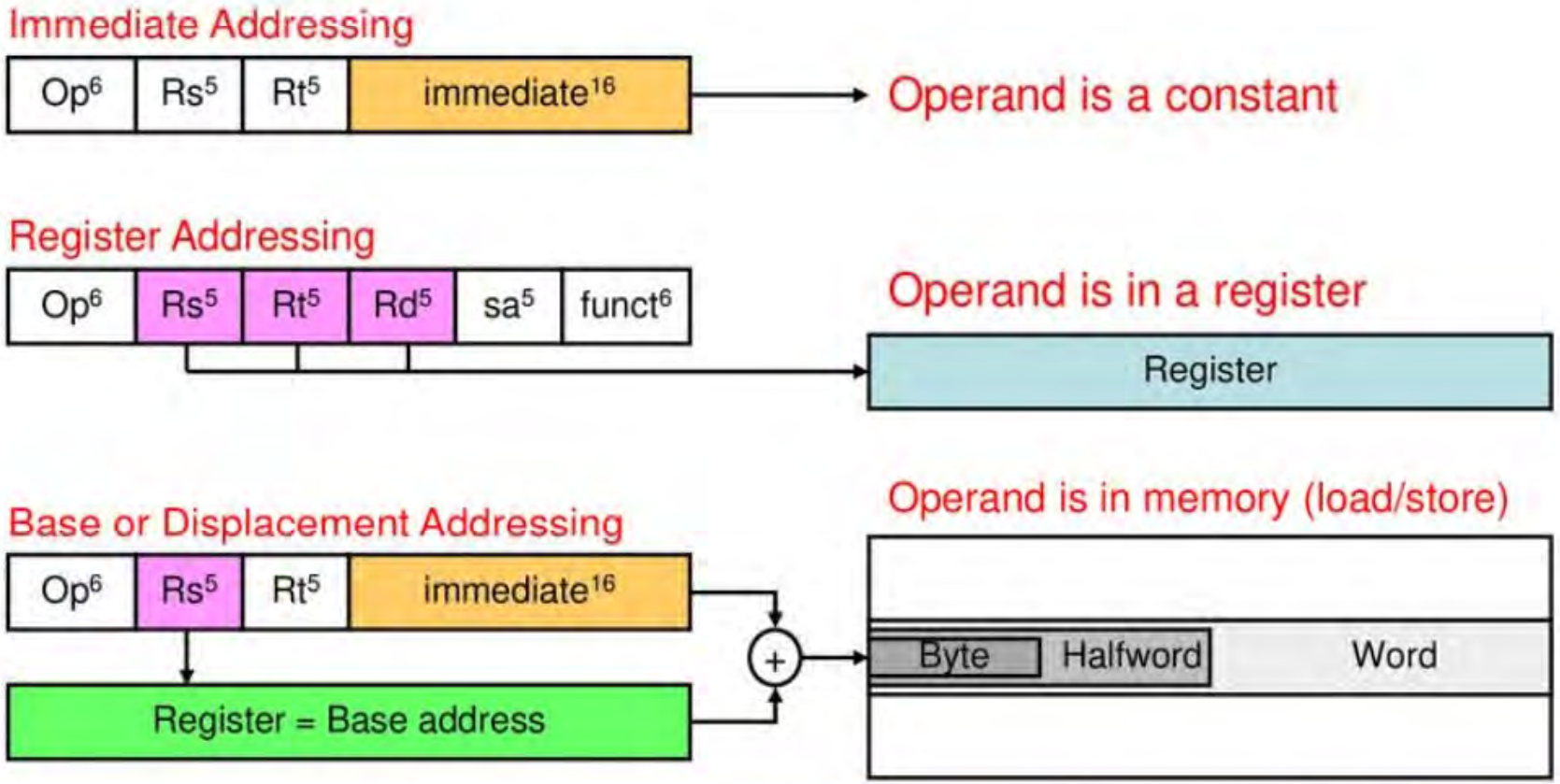
MIPS: Modos de Endereçamentos

MIPS: Modos de Endereçamentos

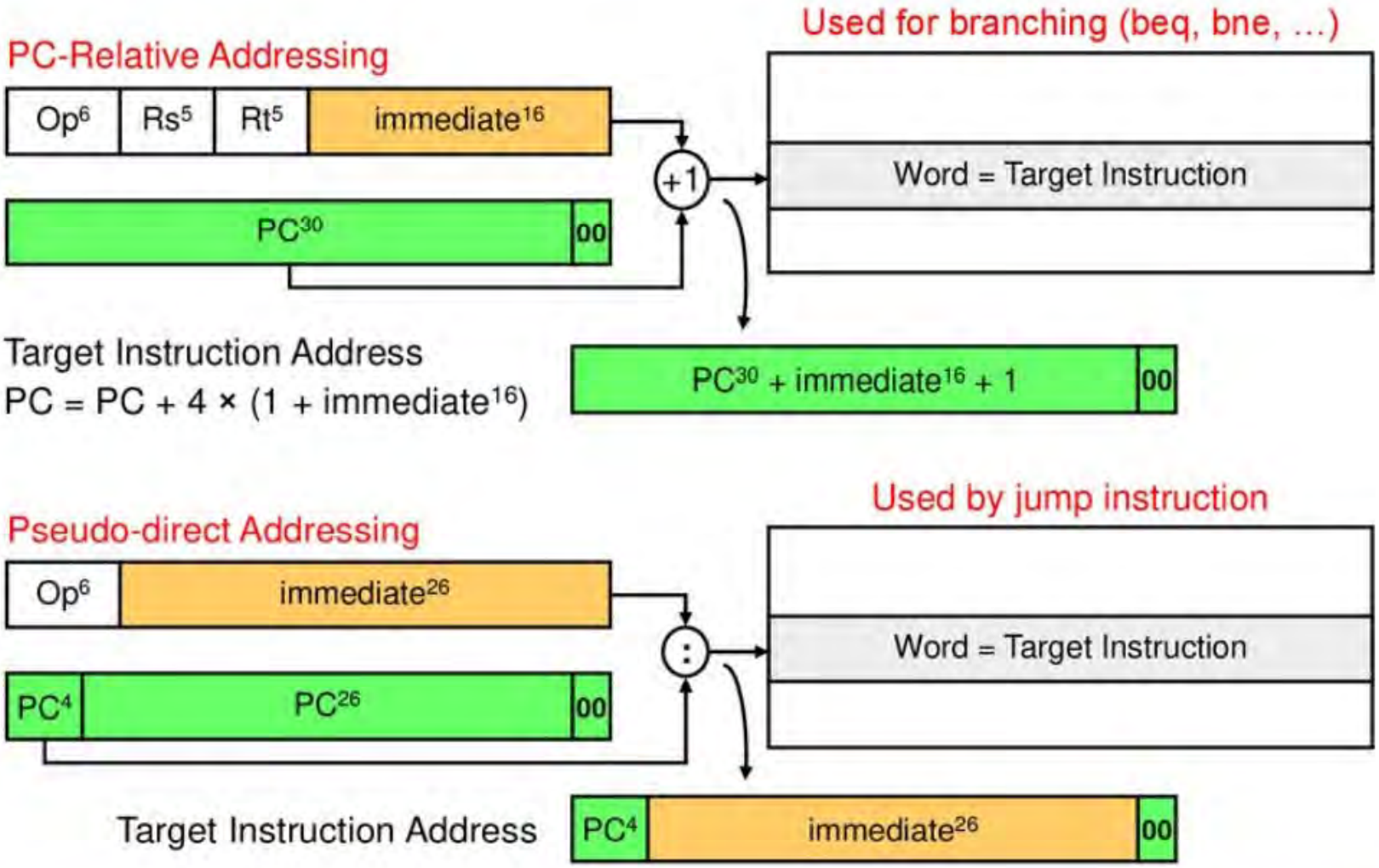
- Endereçamento imediato
  - addi $s2, $s1, 100 (Tipo I)
- Endereçamento por registrador
  - add $t2, $t1, $t0 (Tipo R)
- Endereçamento com base
  - lw $t0, 8($s1) (Tipo I)
- Relativo ao PC
  - beq $t0, $t1, 28 (Tipo I)
- Endereçamento pseudo-direto
  - j 300 (Tipo J)

Addressing Modes

- Where are the operands?
- How memory addresses are computed?



Branch / Jump Addressing Modes



Jump and Branch Limits

- Jump Address Boundary = 2^{26} instructions = 256 MB
  - Text segment cannot exceed 2^{26} instructions or 256 MB
  - Upper 4 bits of PC are unchanged

Target Instruction Address ^{4} 

- Branch Address Boundary
  - Branch instructions use I-Type format (16-bit immediate constant)
  - PC-relative addressing: 

    - Target instruction address = PC + 4×(1 + immediate^{16})
    - Count number of instructions to branch from next instruction
    - Positive constant => Forward Branch, Negative => Backward branch
    - At most ±2^{15} instructions to branch (most branches are near)