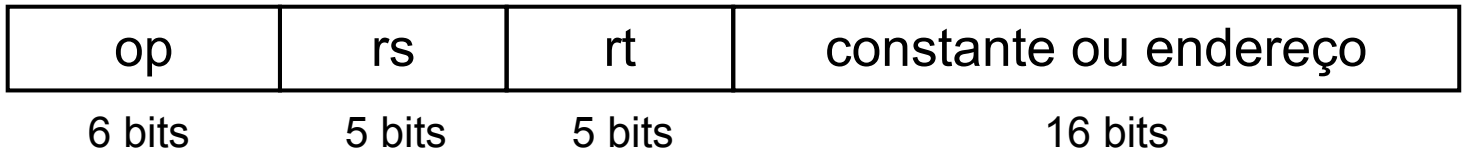
Capítulo 2

Instruções: A Linguagem de Máquina

Operações Condicionais - Formato I

- Desvia para instrução marcada se condição for verdade
  - Caso contrário, continue sequencialmente
- `beq rs, rt, L1 # branch if equal`
  - if (rs == rt) desvia para instrução marcada L1;
- `bne rs, rt, L1 # branch if not equal`
  - if (rs != rt) desvia para instrução marcada L1;

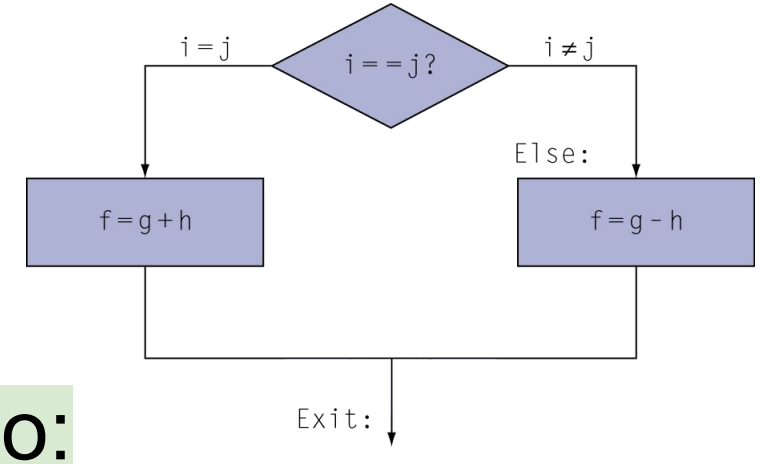


Compilando If-then-else

- Código em C:

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

  - f, g, ... em $s0, $s1, ...



- Código MIPS compilado:

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

Montador calcula endereços

Compilando Loops

- Código em C:

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

  - i em $s3, k em $s5, endereço de save em $s6

- Código MIPS compilado:

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

Mais Operações Condicionais

- Definir resultado 1 se uma condição é verdadeira
  - Caso contrário, defina como 0
- `slt rd, rs, rt # set on less than`
  - if (rs < rt) rd = 1; else rd = 0;
- `slti rt, rs, constante`
  - if (rs < constante) rt = 1; else rt = 0;
- Use em combinação com beq e bne

```
slt $t0, $s1, $s2  # if ($s1 < $s2)
bne $t0, $zero, L  #   desvia para L
```

Comparação Com Sinal e Sem Sinal

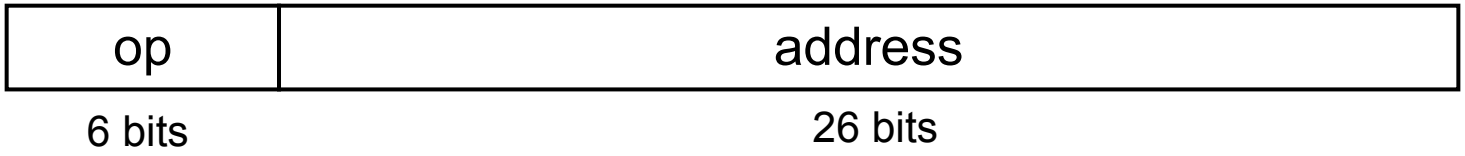
- Comparação com sinal (signed): `slt`, `slti`
- Comparação sem sinal (unsigned): `sltu`, `sltui`
- Exemplo
  - $s0 = 1111 1111 1111 1111 1111 1111 1111 1111
  - $s1 = 0000 0000 0000 0000 0000 0000 0000 0001
  - `slt  $t0, $s0, $s1  # signed`
    - –1 < +1 ⇒ $t0 = 1
  - `sltu $t0, $s0, $s1  # unsigned`
    - +4,294,967,295 > +1 ⇒ $t0 = 0

Projeto de Instruções Desvio

- Por que não `blt`, `bge`, etc?
  - blt (**b**ranch on **l**ess **t**han)
  - bge (**b**ranch on **g**reater or **e**qual)
- Hardware para <, ≥, … mais lento que =, ≠
  - Combinando com desvio envolve mais trabalho por instrução, exigindo um clock mais lento
  - Todas as instruções são penalizadas!
- `beq` e `bne` são um caso comum
- Este é um bom compromisso de projeto

Intruções MIPS Formato J

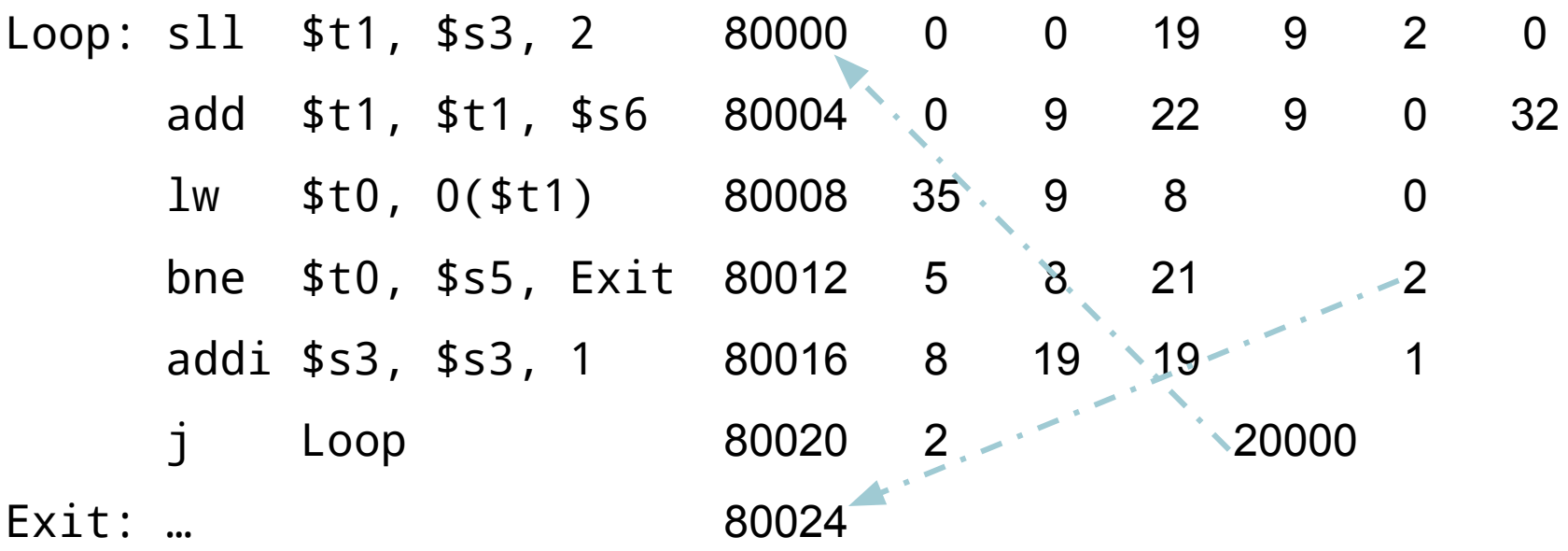
- Jump (`j` e `jal`) poder ser qualquer local no segmento text (código programa)
  - Desvio incondicional
  - Codificado o endereço completo na instrução



- Endereçamento pseudodireto
  - Endereço destino = $PC_{31\ldots28}$: (endereço × 4)
  - O endereço de jump são os 26 bits da instrução concatenados com os bits mais altos do PC

Exemplo de endereçamento de destino

- Código de Loop
  - Suponha o Loop na posição 80000



Loop:	sll	$t1, $s3, 2	80000	0	0	19	9	2	0
	add	$t1, $t1, $s6	80004	0	9	22	9	0	32
	lw	$t0, 0($t1)	80008	35	9	8		0	
	bne	$t0, $s5, Exit	80012	5	8	21		2	
	addi	$s3, $s3, 1	80016	8	19	19		1	
	j	Loop	80020	2			20000		
Exit:	…		80024						

Desvio para mais Longe

- Se o destino dos desvio é muito longe para codificar em 16 bits, o montador reescreve o código
- Exemplo

```
      beq $s0,$s1, L1
            ↓
      bne $s0,$s1, L2
   j L1
   L2: …
```

Chamada de Procedimento

- Etapas necessárias
  1. Colocar parâmetros nos registradores ($a0-$a3)
  2. Transferir o controle para o procedimento (jal)
  3. Salvar registradores $s que usar na pilha
  4. Realizar a tarefa desejada
  5. Colocar o resultado nos registradores de retorno ($v0,$v1)
  6. Restaurar registradores $s e retornar ao local da chamada (jr $ra)

Registradores Utilizados

- $a0 – $a3: argumentos (reg's 4 – 7)
- $v0, $v1: valores de resultado (reg's 2 and 3)
- $t0 – $t9: temporários
  - Pode ser substituído pelo procedimento chamado
- $s0 – $s7: salvo
  - Deve ser salvo/restaurado pelo procedimento chamado
- $gp: global pointer para dados estáticos (reg 28)
- $sp: stack pointer (reg 29)
- $fp: frame pointer (reg 30)
- $ra: return address (reg 31)

Instruções - Chamada de Procedimento

- Chamada procedimento: jump and link

  `jal EndereçoProcedimento`

  - Salva o enderço da instrução seguinte no $ra
  - Salta para o endereço de destino
- Retorno procedimento: jump register

  `jr $ra`

  - Copia $ra para pc (program counter)
  - Também pode ser usado para saltos computados
    - ex., para declarações case/switch

Exemplo Procedimento Folha

- Código em C:

```
int folha (int g, h, i, j)
{ int f;
  f = (g + h) - (i + j);
  return f;
}
```

  - Argumentos g, …, j em $a0, …, $a3
  - f em $s0 (portanto, precisa salvar $s0 na pilha)
  - Resulado em $v0

Exemplo Procedimento Folha

- Código MIPS:

```
folha:
  addi $sp, $sp, -4
  sw   $s0, 0($sp)
  add  $t0, $a0, $a1
  add  $t1, $a2, $a3
  sub  $s0, $t0, $t1
  add  $v0, $s0, $zero
  lw   $s0, 0($sp)
  addi $sp, $sp, 4
  jr   $ra
```

Salva $s0 na pilha

Corpo procedimento

Resultado em $v0

Restaura $s0

Retorna

Procedimento Não Folha

- Procedimentos que chamam outros procedimentos
- Para chamadas aninhadas, o chamador precisa salvar na pilha:
  - Seu endereço de retorno
  - Quaisquer argumentos e temporários necessários após a chamada
- Restaurar da pilha após a chamada

Exemplo Procedimento Não Folha

- Código C:

```
int fatorial(int n)
{
  if (n < 1) return f;
  else return n * fact(n - 1);
}
```

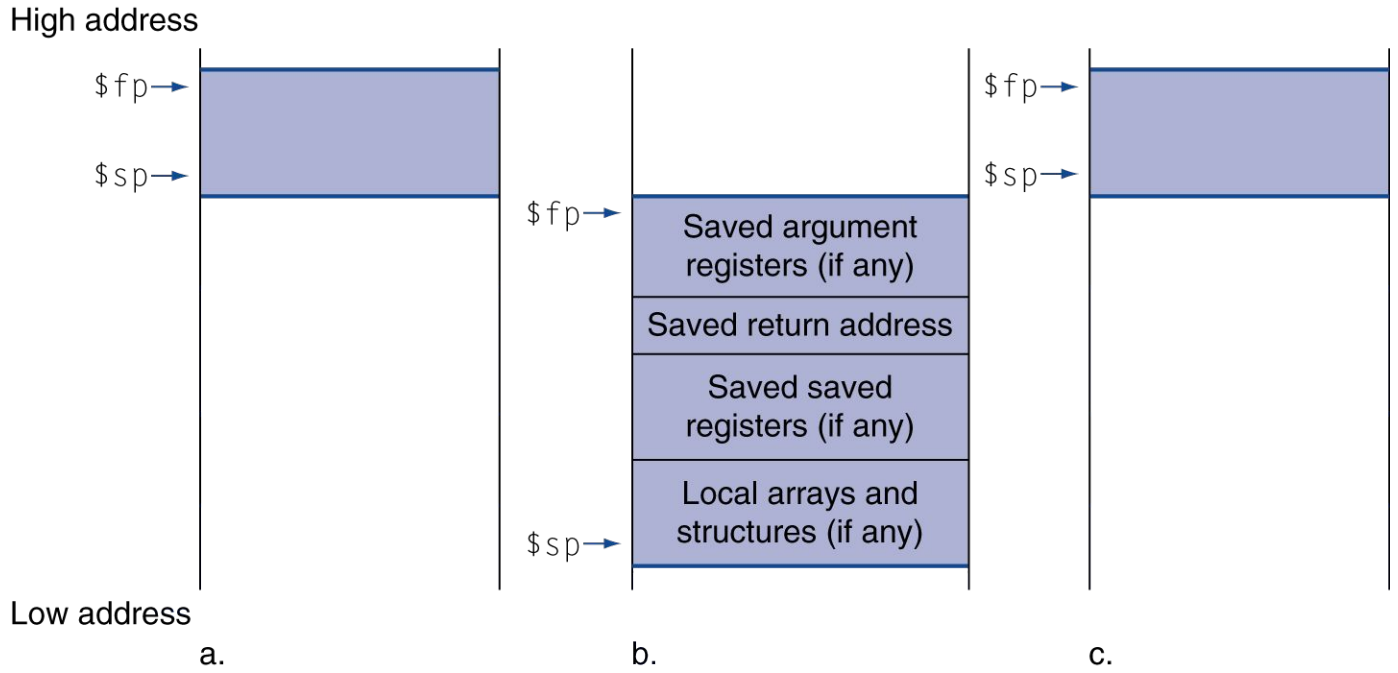
  - Argumento n em $a0
  - Resultado em $v0

Exemplo Procedimento Não Folha

- Código MIPS:

```
fatorial:
    addi $sp, $sp, -8      # ajusta pilha  +2 itens
    sw   $ra, 4($sp)       # salva endereço retorno
    sw   $a0, 0($sp)       # salva o argumento n
    slti $t0, $a0, 1       # teste para n < 1
    beq  $t0, $zero, L1    # se n>=1, vai para L1
    addi $v0, $zero, 1     # retorna 1
    addi $sp, $sp, 8       # retira 2 itens da pilha
    jr   $ra               # retorna para depois jal
L1: addi $a0, $a0, -1      # n>=1: argumento (n-1)
    jal  fact              # chama fatorial com (n-1)
    lw   $a0, 0($sp)       # restaura argumento n
    lw   $ra, 4($sp)       # restaura endereço retorno
    addi $sp, $sp, 8       # ajusta pilha -2 itens
    mul  $v0, $a0, $v0     # retorna n*fatorial(n-1)
    jr   $ra               # retorna para chamador
```

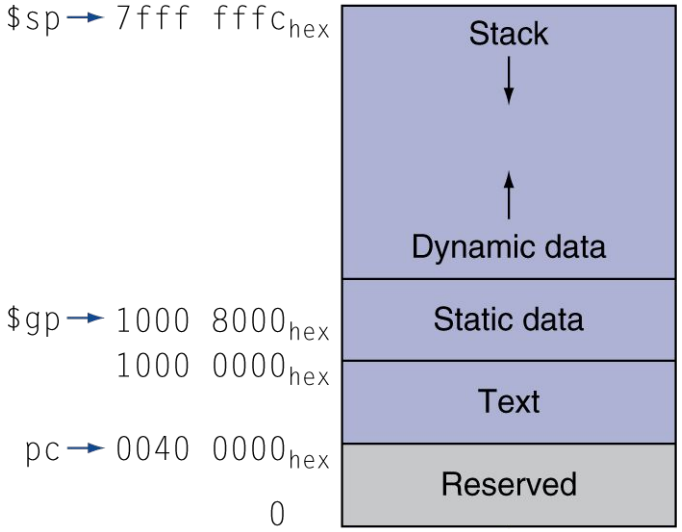
Dados Locais na Pilha



- Variáveis locais alocadas pelo procedimento chamado
  - Como array ou estruturas locais
- Frame de procedimento (registro de ativação)
  - Alguns compiladores utilizam o frame pointer ($fp) para apontar para a primeira word do registro de ativação

Layout da Memória

- Text: código programa
- Static data: variáveis globais
  - ex., variáveis static em C, arrays constantes e strings
  - $gp este endereço permite ±offsets dentro do segmento
- Dynamic data: heap
  - ex., malloc em C, new em Java
- Stack: Armazenamento automático



Dados e Caracteres

- Conjunto de caracteres codificados byte
  - ASCII: 128 characteres
    - 95 gráficos, 33 controle
  - Latin-1: 256 characteres
    - ASCII, +96 caracteres gráficos adicionais
- Unicode: conjunto de caracteres de 32 bits
  - Usado em Java, C++, …
  - A maioria dos alfabetos do mundo, mais símbolos
  - UTF-8, UTF-16: codificações de comprimento variável

Operações Byte/Halfword

- Poderia utilizar operações bit a bit (bitwise)
- MIPS byte/halfword load/store
  - Processamento de string é um caso comum

```
lb rt, offset(rs)      lh rt, offset(rs)
```

  - Sinal extendido para 32 bits em rt

```
lbu rt, offset(rs)     lhu rt, offset(rs)
```

  - Zero extendido para 32 bits em rt

```
sb rt, offset(rs)      sh rt, offset(rs)
```

  - Armazena somente byte/halfword mais à direita

Exemplo Cópia de String

- Código em C:
  - String terminada em null

```
void strcpy (char x[], char y[])
{ int i;
  i = 0;
  while ((x[i]=y[i])!='\0')
    i += 1;
}
```

  - Endereços x, y em $a0, $a1
  - i em $s0

Exemplo Cópia de String

- Cógigo MIPS:

```
 strcpy:
    addi $sp, $sp, -4       # ajusta pilha em 1 item
    sw   $s0, 0($sp)        # salva $s0
    add  $s0, $zero, $zero # i = 0
L1: add  $t1, $s0, $a1      # endereço y[i] em $t1
    lbu  $t2, 0($t1)        # $t2 = y[i]
    add  $t3, $s0, $a0      # endereço x[i] in $t3
    sb   $t2, 0($t3)        # x[i] = y[i]
    beq  $t2, $zero, L2     # se y[i] == 0 sai loop
    addi $s0, $s0, 1        # i = i + 1
    j    L1                 # próxima interação loop
L2: lw   $s0, 0($sp)        # restaura $s0 salvo
    addi $sp, $sp, 4        # retira 1 item da pilha
    jr   $ra                # retorna
```

Constantes 32-bit

- A maioria das constantes são pequenas
  - É suficiente campo com 16-bit (immediate)
- Para a constante de 32 bits ocasional

```
lui rt, constant
```

  - Copia 16 bits para os 16 bits da esquerda rt
  - Zera os 16 bits da direita rt

```
$s0 <- 0x007D0900
```

0000 0000 0111 1101 0000 1001 0000 0000

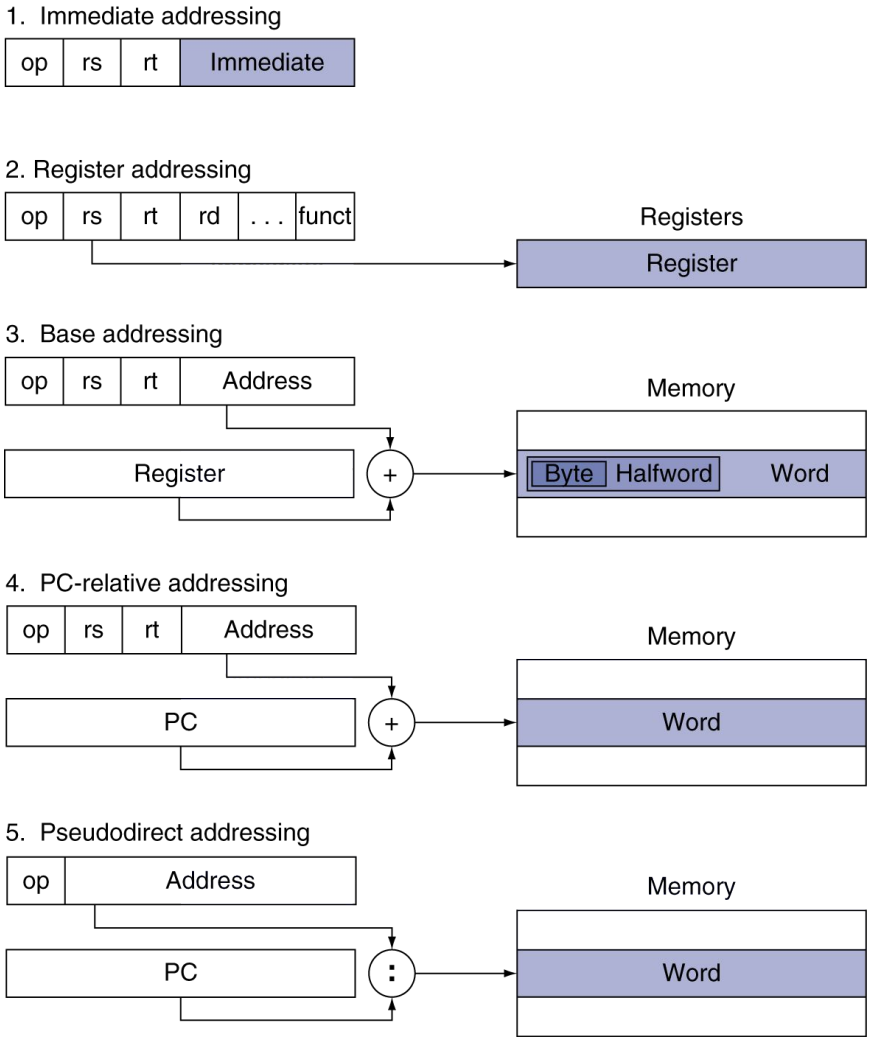
```
lui $s0, 0x007D
```

0000 0000 0111 1101 0000 0000 0000 0000

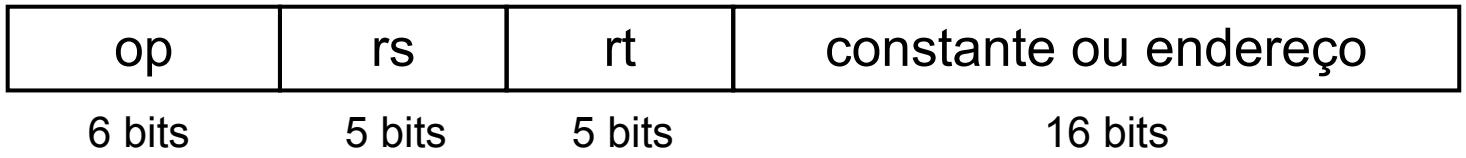
```
ori $s0, $s0, 0x0900
```

0000 0000 0111 1101 0000 1001 0000 0000

Resumo Modos Endereçamento



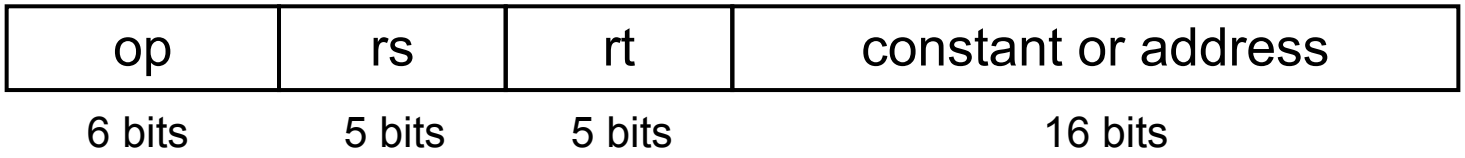
Endereçamento de Base



- Instruções load/store
  - rt: número do registrador de destino ou origem
  - Constante: -2^{15} até $+2^{15} - 1$
  - Endereço: endereço adicionado ao endereço base no registrador rs

Endereçamento Relativo ao PC

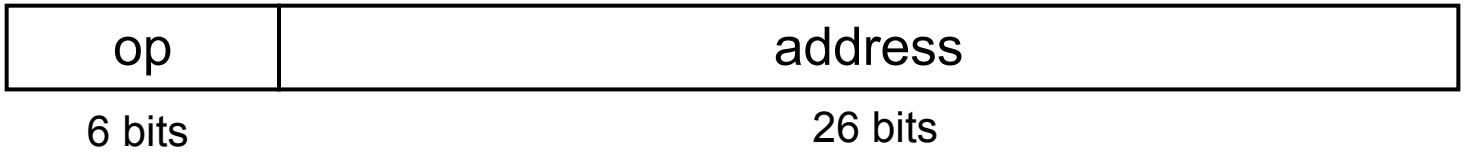
- Instruções de desvio beq/bne especificam
  - Opcode, dois registradores, endereço destino
- Na maioria o destino do desvio está próximo
  - Para frente ou para trás



- Endereçamento relativo ao PC
  - Endereço destino = PC + (endereço × 4)
  - Número de palavras até a próxima instrução

Endereçamento Pseudodireto

- Jump (`j` e `jal`) poder ser qualquer local no segmento text (código programa)
  - Desvio incondicional
  - Codificado o endereço completo na instrução



- Endereçamento pseudodireto
  - Endereço destino = $PC_{31\ldots28}$: (endereço × 4)
  - O endereço de jump são os 26 bits da instrução concatenados com os bits mais altos do PC

Sincronização

- Dois processos compartilhando área na memória
  - P1 escreve, e P2 lê
  - Data race se P1 e P2 não sincronizarem
    - Resultado depende da ordem dos acessos
- Requer suporte do hardware
  - Operações atômicas leitura/escrita na memória
  - Nenhum outro acesso ao local permitido entre leitura e escrita
- Pode ser uma única instrução
  - Ex., swap atômico de registrador ↔ memória
  - Ou um par atômico de instruções

Sincronização no MIPS

- Load linked: `ll rt, offset(rs)`
- Store conditional: `sc rt, offset(rs)`
  - Sucesso se o local não for alterado desde o ll
    - Retorna 1 em rt
  - Falha se o local foi alterado
    - Retorna 0 em rt
- Exemplo: swap atômico

```
try: add $t0,$zero,$s4 ;copia valor da troca
     ll  $t1,0($s1)    ;load linked
     sc  $t0,0($s1)    ;store conditional
     beq $t0,$zero,try ;desvia store falha
     add $s4,$zero,$t1 ;coloca valor de load
                       ;em $s4
```

Referências

- Capítulo 2 - “Organização e Projeto de Computadores - A Interface Hardware/Software, David A. Patterson & John L. Hennessy, Campus, 4 edição, 2013.