

“Abandone toda esperança aquele que por aqui entrar.”

Conjuntos de Instruções

Paulo Ricardo Lisboa de Almeida

Instruções de Máquina

- Para nos comunicar com o processador precisamos “falar a sua língua”
- Alguns exemplos:
 - O seu computador pessoal
 - x86, AMD64 (x64)
 - Seu Smartphone
 - ARM
 - Microcontroladores
 - MIPS, PIC instruction SET, ...

Instruções de Máquina

- O Conjunto de instruções está diretamente relacionado com o hardware
 - Como o hardware interpreta as instruções
 - O quão complexa é a interpretação
 - A quantidade de instruções disponíveis
 - Como as instruções são armazenadas e requisitadas da memória
 - ...

MIPS

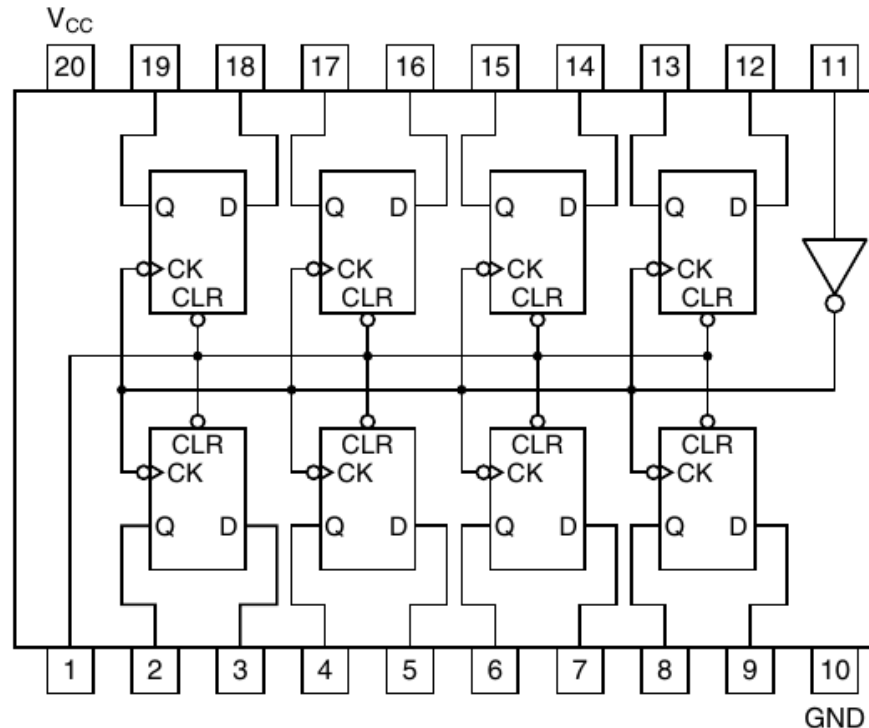
- Inicialmente abordaremos a arquitetura MIPS de 32 bits
 - Discutido em Patterson e Henessy (2014)
 - **Microprocessor without Interlocked Pipeline Stages**
- Desenvolvido por, entre outros pesquisadores, Patterson e Henessy
 - Turing Award de 2017
 - Ideias do MIPS da década de 80 possibilitaram a criação de processadores extremamente eficientes, como os do seu smartphone
 - Diversos processadores de hoje que utilizam a arquitetura MIPS atual, ou são baseados nela
- Conjunto de instruções relativamente simples
- Aprenda um conjunto/arquitetura e migrar para outro conjunto de instruções será (quase) fácil

■ Registradores

- A vasta maioria das arquiteturas atuais (x86-64, MIPS, ARM) operam somente na CPU
- Precisamos carregar os dados para os **registradores** da CPU
 - Porções de memória na CPU as quais podemos utilizar para realizar operações
- Os registradores **são visíveis** ao programador
 - Ao menos quando programamos em baixo nível
 - Existem registradores **não visíveis**, mas não trataremos deles agora

Registadores

- Registradores geralmente são construídos com flip-flops
 - Exemplo de um Circuito Integrado com 8 flip-flops, formando uma memória de 8 bits



Tanenbaum (2007)

Registadores

- Registradores são os dispositivos de memória **mais rápidos** disponíveis no computador
- Enquanto temos uma abundância relativa de memória principal, os **registradores são escassos**
 - Em MIPS temos **32 registradores de 32 bits cada**
 - O seu processador x86 tem apenas 8 registradores que usamos em nossos programas
 - 16 registradores no x86-64
 - Os microcontroladores PIC 16F62x possuem **um registrador** geral (W)
- Cada registrador precisa ter um endereço. Quantos bits são necessários para endereçar todos os registradores do MIPS?

Registadores

- São necessários **5 bits** para endereçar os registradores do MIPS ($2^5 = 32$).

Número (Decimal)	Nome Registrador	Descrição
0	\$zero,\$r0	Sempre contém zero
1	\$at	Utilizado para o assembler (montador)
2 e 3	\$v0 e \$v1	Valores de retorno
4,...,7	\$a0,...,\$a3	Argumentos de função
8,...,15	\$t0,...,\$t7	Para cálculos temporários (não salvos)
16,...,23	\$s0,...,\$s7	Registradores salvos (entre chamadas de função)
24 e 25	\$t8 e \$t9	Mais registradores temporários
26 e 27	\$k0 e \$k1	Reservados para o Kernel (S.O.)
28	\$gp	Apontador de memória global
29	\$sp	Ponteiro de pilha
30	\$fp	Ponteiro de quadro
31	\$ra	Endereço de retorno

Registadores

- No momento vamos focar nos registradores gerais 8 a 15 (não salvos), e 16 a 23 (salvos)
- **A máquina entende somente zeros e uns (Linguagem de Máquina)**
 - Difícil enxergar que o valor 10001_2 em uma instrução se referencia ao registrador 17_{10}
- Por essa razão programamos em **linguagem de montagem – Assembly**
 - Nos referenciamos aos registradores (e operações) **por seus nomes**
- Os nomes dos registradores em assembly do MIPS **começam com \$**
 - Exemplo: o registrador \$s0 é o registrador 16_{10} , ou 10000_2
- O **montador (Assembler)** simplesmente traduz de \$s0 para 10000_2 em linguagem de máquina

Número	Nome	Descrição
0	\$zero,\$r0	Sempre contém zero
1	\$at	Utilizado para o assembler
2 e 3	\$v0 e \$v1	Valores de retorno
4,...,7	\$a0,...,\$a3	Argumentos de função
8,...,15	\$t0,...,\$t7	Para cálculos temporários
16,...,23	\$s0,...,\$s7	Registradores salvos
...

Tamanho da palavra – Word size

- O tamanho “natural” dos dados que um processador lida é denominado **word (palavra)**
- O tamanho da palavra (word) do MIPS32 é de 32 bits
- No MIPS32, os registradores suportam 32 bits, e as operações geralmente lidam com 32 bits
- Processadores diferentes possuem palavras de tamanhos diferentes
 - x86-64 possui uma word de 64 bits
 - Os PICs da família 16F62x possuem uma word de 8 bits

Instruções

- Todas instruções **no MIPS** ocupam **32 bits**
- **A consistência facilita o projeto**
- x86 por exemplo possui instruções de tamanhos variados
 - Mais flexível, mas o hardware se torna muito mais complexo (e muitas vezes lento)

Instruções

- Um exemplo de uma instrução no MIPS então poderia ser

00000010001100100100000000100000



32 bits

Instruções

- Um exemplo de uma instrução no MIPS então poderia ser

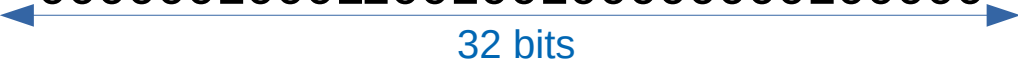
00000010001100100100000000100000



32 bits

- Um tanto difícil interpretar e criar um programa utilizando diretamente as instruções de máquina

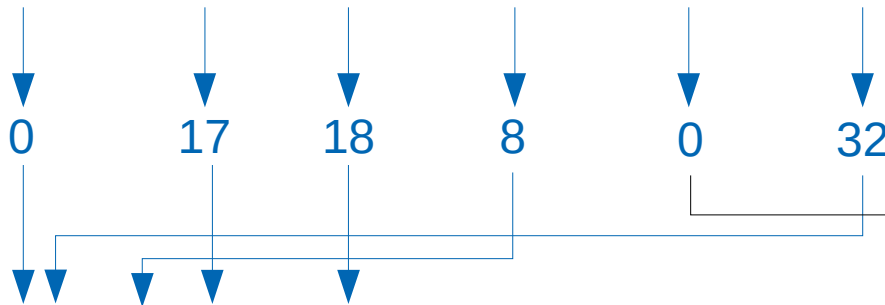
Instruções

- Um exemplo de uma instrução no MIPS então poderia ser
00000010001100100100000000100000

32 bits
- Um tanto difícil interpretar e criar um programa utilizando diretamente as instruções de máquina
 - Esse é um dos motivos de programarmos em **Assembly**
 - O **montador (assembler)** consegue traduzir diretamente de Assembly para a linguagem de máquina, e vice-versa
- Diferente de um compilador, que precisa fazer uma “reinterpretação do código” para transformá-lo em linguagem de máquina

Instruções

- No assembly, utilizamos **mnemônicos** ao invés dos bits diretamente para representar uma instrução
- Exemplo:

000000 10001 10010 01000 00000 100000 ← **Linguagem de Máquina**



→ Não usado nessa instrução.
Problema de toda instrução
ter o mesmo tamanho!

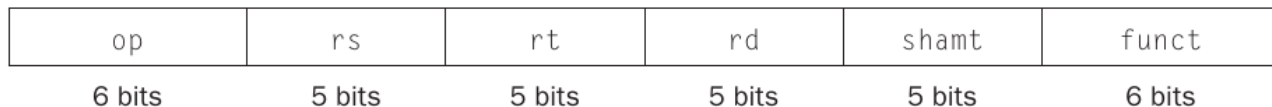
assembly → **add \$t0, \$s1, \$s2** #somar \$s1 com \$s2 e armazenar o resultado em \$t0

Instruções

- Para entender como a CPU interpreta a instrução, e como podemos transformar de assembly para linguagem de máquina (e vice-versa), vamos começar a entender a arquitetura MIPS
- A instrução MIPS possui campos com larguras pré-definidas
 - Quais campos são utilizados em quais instruções depende do formato da instrução

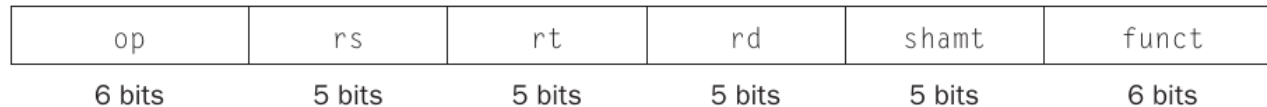
Instruções

- op:** código básico da instrução, tradicionalmente chamado de **opcode**
- rs:** registrador do primeiro operando (fonte)
- rt:** registrador do segundo operando (fonte)
- rd:** registrador destino
- shamt:** “Shift Ammount” (quantidade de deslocamento) → veremos adiante
- funct:** variante da operação



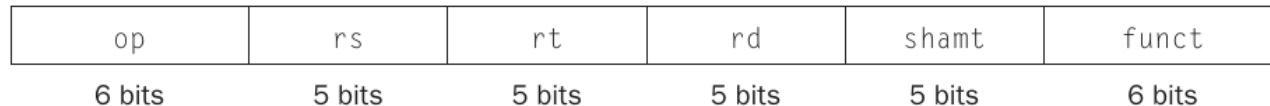
Instruções

- Qual seria o problema no MIPS se tivéssemos mais de 32 registradores?



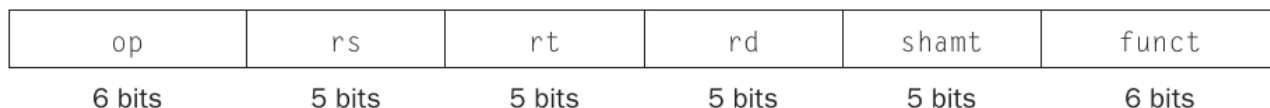
Instruções

- Qual seria o problema no MIPS se tivéssemos mais de 32 registradores?
 - Os campos rs, rt e rd precisariam de mais bits
 - Sacrificaríamos outros campos, ou então ocuparíamos mais bits com as instruções

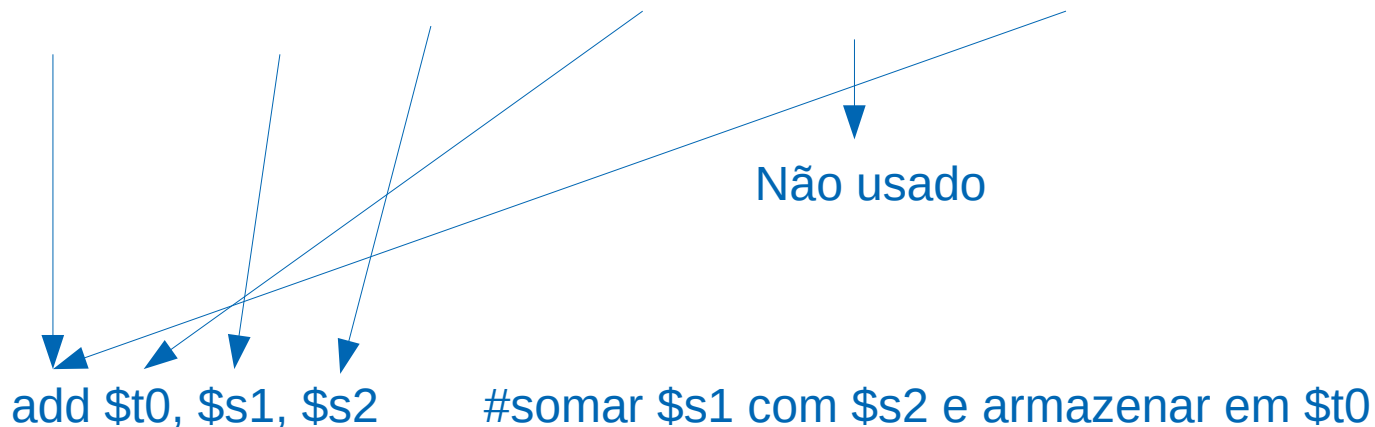


Instruções do tipo-R

- O formato de instrução que vimos anteriormente é chamado **tipo-R**
 - tipo-Registrador**



Exemplo: 000000 10001 10010 01000 00000 100000



Instruções do tipo-I

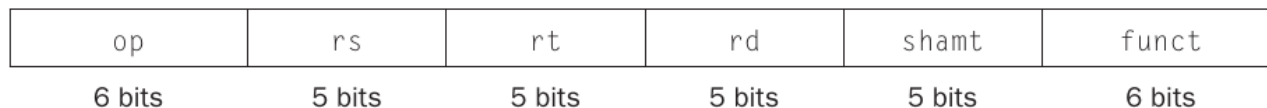
- Instruções do tipo-R são fundamentais para lidarmos diretamente com registradores
- Mas e se precisarmos carregar um valor “fixo” para dentro de um registrador?
 - Ex.: colocar o valor 2855_{10} em \$s0
 - Poderíamos utilizar um opcode diferente para especificar que rs ou rt se referem ao valor a ser carregado, e não o endereço do registrador
 - Qual o problema?



Tipo-R

Instruções do tipo-I

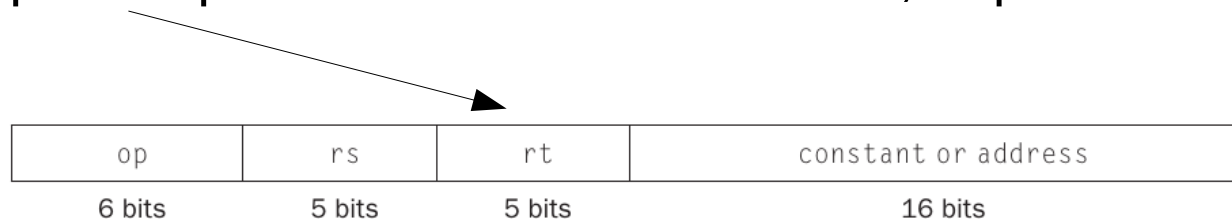
- Poderíamos utilizar um opcode diferente para especificar que rs ou rt se referem ao valor a ser carregado, e não o endereço do registrador
 - Problema
 - Temos apenas 5 bits nesses campos
 - A maior constante que podemos especificar seria 32_{10}
 - Se considerarmos valores com sinal em complemento a 2, nosso alcance cai para valores entre -16 e +15



Tipo-R

Instruções do tipo-I

- Instruções do **tipo-I** servem para (dentre outras coisas) **carregar constantes**, denominadas **valores imediatos**, e para acessar a memória
 - **tipo-Imediato**
- Não temos rd, shamt e func
 - Esses campos viram um único campo de 16 bits, onde colocamos o imediato
 - Agora podemos inserir constantes de $\pm 2^{15}$ (complemento a dois)
- op e rs possuem os mesmos significados do tipo-R
- No tipo-I, o campo rt especifica o destino ou a fonte, dependendo da instrução



Tipo-I

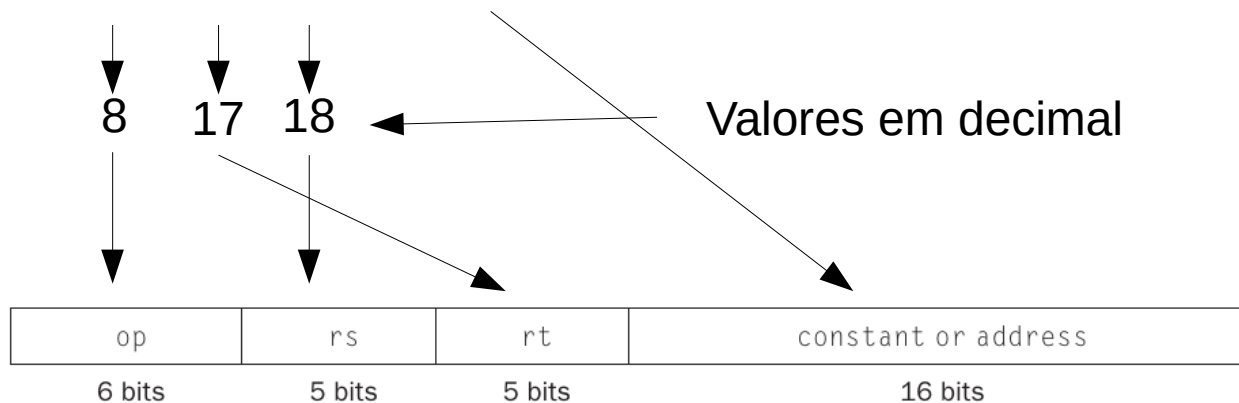
Instruções do tipo-I - Exemplo

addi reg1,reg2,imediato #some reg2 + imediato e armazene em reg1

Exemplo concreto

#some o **conteúdo** de \$s2 com o **imediato** (constante) 100₁₀ e armazene em \$s1

addi \$s1, \$s2, 100



Tipo-I



Exercícios

1. Procure na internet pela especificação das instruções MIPS, contendo os valores de opcode e func para cada instrução e registrador do MIPS

- Exemplos
 - github.com/MIPT-ILab/mipt-mips/wiki/MIPS-Instruction-Set
 - opencores.org/projects/plasma/opcodes
- Considerando que as instruções de máquina a seguir estão em big-endian (e representadas em hexa), indique o assembly para cada instrução
- Valores com sinal são representados com complemento a 2
- Leve isso em consideração se as operações são unsigned ou não
- Exemplo: addiu é uma soma sem sinal, enquanto addi utiliza o complemento a 2

a) 02 32 40 22

b) 02 50 58 21

c) 26 51 FF AA

d) 22 73 FF AA

- Compare suas respostas com www.eg.bucknell.edu/~csci320/mips_web/

Exercícios

2. Mostre o código de máquina para as instruções a seguir e encaixe os bits nos campos das instruções do tipo-R ou tipo-I (dependendo da instrução). Obs.: Os valores imediatos nas instruções estão em decimal

- a) sub \$s0, \$t3, \$t4
- b) addi \$s1, \$s2, -8
- c) addiu \$t6, \$t6, 555

Compare suas respostas com

www.eg.bucknell.edu/~csci320/mips_web/

<http://www.kurtm.net/mipsasm/index.cgi>

3. Considere que precisamos subtrair 256_{10} do registrador \$t0, e armazenar o resultado no registrador \$s0. Como essa instrução vai ficar em assembly do MIPS? E em código de máquina do MIPS? Note que **não temos uma instrução para subtrair um imediato** no Assembly do MIPS.

Referências

- D. Patterson; J. Henessy. **Organização e Projeto de Computadores**. 4a Edição: Interface Hardware / Software. Elsevier Brasil, 2014.
- Andrew S. Tanenbaum. **Organização estruturada de computadores**. 5. ed. São Paulo: Pearson, 2007.
- Ronald Tocci, Neal Widmer, Greg Moss. **Digital Systems**. 12 ed. Pearson Education. 2016.
- James Bignell, Robert Donovan. **Eletrônica digital**. Cengage Do Brasil, 2010.
- MELO, M. **Eletrônica Digital**. Makron Books.2003.