

[Aula 02] Conjunto de instruções 1

Prof. João F. Mari
joaof.mari@ufv.br

Roteiro

- Introdução
- Operações no hardware do computador
- Operandos do hardware do computador
- Representando instruções no computador
- Operações lógicas
- Instruções para tomada de decisões
- Suporte para procedimentos no hardware do computador

Introdução

- Para controlar o hardware do computador, é necessário falar a sua linguagem.
- Palavras da linguagem do computador são chamadas de **instruções**
 - O vocabulário é chamado de conjunto de instruções.
- Apresentação das instruções através de uma abordagem *top-down*.
- Linguagens de computadores são semelhantes (entre diferentes arquiteturas)
 - Ao contrário da linguagem dos humanos.

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 sequência de operações... As considerações 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, junto com a velocidade com que tratam esses problemas”*
 - Burks, Goldstine e von Neumann, 1947

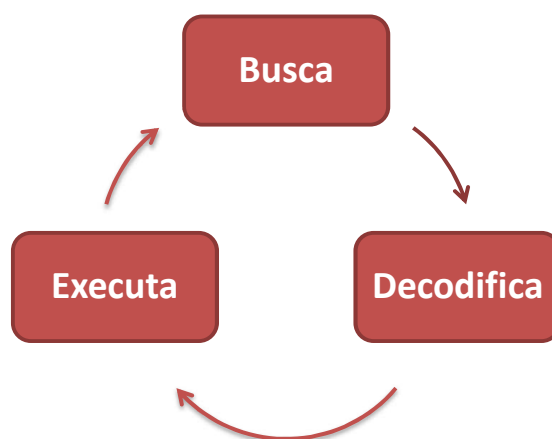
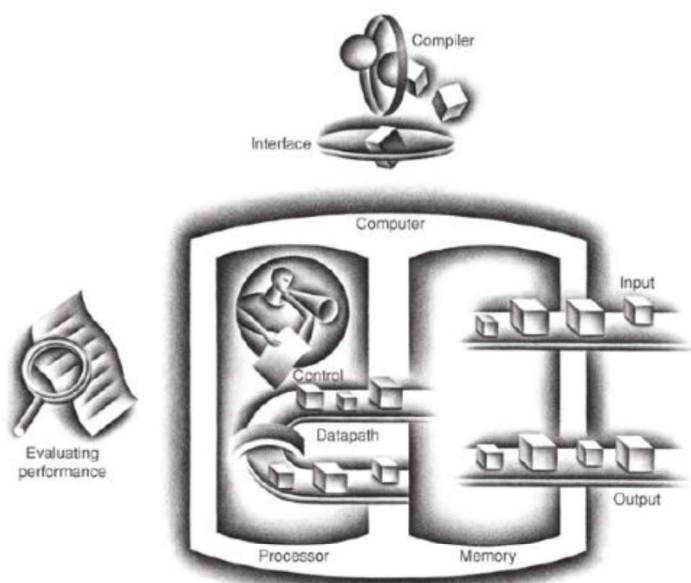
Introdução

- Conjunto de instruções (a ponta do iceberg)
- [EX] MIPS:
 - [https://pt.wikipedia.org/wiki/Arquitetura MIPS](https://pt.wikipedia.org/wiki/Arquitetura_MIPS)
 - Conjunto de instruções criado na década de 1980;
 - 100 milhões de processadores fabricados em 2002;
 - ATI, Broadcom, Cisco, NEC, Nintendo, Silicon Graphics, Sony, Texas Instruments e Toshiba.
- Processador RISC (conjunto reduzido de instruções):
 - Tamanho de instruções fixo (*fixed instruction lengths*);
 - Instruções de load-store (*load-store instruction sets*);
 - Modos de endereçamento limitado (*limited addressing modes*);
 - Operações limitadas (*limited operations*).



Introdução

- O ciclo básico de busca e execução de instruções.



MIPS R3000 ISA (Arquitetura do Conjunto de Instruções)

- Tipos de instrução:
 - Movimentação de Dados (*Load e Store*);
 - Aritmética e Lógica;
 - Saltos e Desvios (*Jump e Branch*);
 - Ponto flutuante;
 - Coprocessador.
 - Gerenciamento de memória;
 - Especiais.

FORMATOS DE INSTRUÇÃO BÁSICOS (32 bits)

	31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
R	opcode	rs	rt	rd	shamt	funct
	31 – 26	25 – 21	20 – 16	15 – 0		
I	opcode	rs	rt	endereço/imediato		
	31 – 26	25 – 0				
J	opcode	endereço				

OPERAÇÕES NO HARDWARE DO COMPUTADOR

Operações no hardware do computador

- Todo computador precisa realizar operações aritméticas:
 - **[EX]** `add a, b, c`
 - Instrui o computador para realizar a soma entre as variáveis `b` e `c` e armazenar o resultado em `a`.
- **[EX]** Colocar a soma de `b, c, d, e` e na variável `a`:
 - Em linguagem de alto nível
 - `a = b + c + d + e;`
 - Em Assembly MIPS:
 - `add a, b, c` # soma `b+c` é colocada em `a`
 - `add a, a, d` # soma `b+c+d` está em `a`
 - `add a, a, e` # soma `b+c+d+e` está em `a`

Operações no hardware do computador

- O número natural de operandos para uma operação de adição é três
 - Exigir que cada instrução tenha exatamente três operandos nem mais nem menos,
 - Manter o hardware simples → o hardware para um número variável de operandos é mais complicado do que o hardware para um número fixo:
- **Princípio de projeto 1:**
 - *Simplicidade favorece a regularidade.*

Operações no hardware do computador

- Compilando duas instruções de atribuição C no MIPS:
 - $a = b + c;$
 - $d = a - e;$
- A tradução é realizada pelo compilador (MIPS):
 - `add a, b, c`
 - `sub d, a, e`

Operações no hardware do computador

- Compilando uma instrução complexa no MIPS
 - $f = (g + h) - (i + j);$
- O compilador precisa desmembrar essa instrução em várias instruções *assembly*, pois somente uma operação é realizada por instrução MIPS.
 - `add t0, g, h`
 - `# var. temp. t0 contém g+h`
 - `add t1, i, j`
 - `# var. temp. t1 contém i+j`
 - `sub f, t0, t1 # f recebe t0 - t1`
- Note que uma expressão na linguagem C gera 3 instruções *assembly* para o MIPS.

OPERANDOS NO HARDWARE DO COMPUTADOR

Operandos no hardware do computador

- Os operandos das instruções aritméticas precisam estar armazenados em um grupo limitado de locais especiais, embutidos diretamente no hardware, chamados registradores (*registers*)
 - Os registradores são os tijolos da construção do computador.
 - Primitivas usadas no projeto do computador.
 - Visíveis para o programador.
- O MIPS possui 32 registradores:
 - Cada registrador possui 32 bits.
 - Grupos de 32 bits ocorrem com tanta frequência no MIPS que recebem o nome de palavra (*word*).
 - Grupos específicos de registradores. Por enquanto:
 - $\$s0, \$s1, \dots, \$s7$
 - Correspondem as variáveis dos programas C e Java.
 - $\$t0, \$t1, \dots, \$t7$
 - Registradores temporários necessários para compilar o programa.

Operandos no hardware do computador

Nome	Número do registrador	Uso	Valor preservado
\$zero	0	Constante 0	n.a.
\$at	1	Reservado para o montador	n.a.
\$v0 – \$v1	2-3	Valores retornados	não
\$a0 – \$a3	4-7	Argumentos	sim
\$t0 – \$t7	8-15	Temporários	não
\$s0 – \$s7	16-23	Valores salvos	sim
\$t8 – \$t9	24-25	Temporários	não
\$k0 – \$k1	26-27	Temporários SO	sim
\$gp	28	Ponteiro global	sim
\$sp	29	Ponteiro para pilha	sim
\$fp	30	Ponteiro para quadro	sim
\$ra	31	Endereço de retorno	sim

Operandos no hardware do computador

- Uma diferença entre variáveis de um programa em linguagem de alto nível e os registradores é que o número de registradores é limitado
 - O computador MIPS tem 32 registradores:
 - Uma quantidade muito grande de registradores pode aumentar o tempo do ciclo de *clock* simplesmente porque os sinais eletrônicos levam mais tempo quando precisam atravessar uma distância maior
 - Deve-se equilibrar o “desejo” dos programas por mais registradores com o desejo do projetista de manter o ciclo de *clock* mais rápido.
 - Um número maior de registradores necessita de um número maior de bits para representação: influência no tamanho da instrução.
- **Princípio de projeto 2:**
 - Menor significa mais rápido

[EX] Compilando uma atribuição em C usando registradores

- Compilando uma atribuição em C usando registradores:
 - $f = (g + h) - (i + j);$
- As variáveis f, g, h, i e j estão associadas aos registradores $\$s0, \$s1, \$s2, \$s3$ e $\$s4$ respectivamente
- Código MIPS compilado:
 - `add $t0, $s1, $s2` # reg. \$t0 contém $g + h$
 - `add $t1, $s3, $s4` # reg. \$t1 contém $i + j$
 - `sub $s0, $t0, $t1`
 - # reg. \$s0 contém $\$t0 - \$t1,$
 - # que é $(g + h) - (i + j)$

Operandos no hardware do computador

- **Operandos na memória:**
 - Operações aritméticas só ocorrem com os registradores nas instruções MIPS:
 - O MIPS deve ter instruções que transferem os dados entre o processador e a memória: instruções de transferência de dados.
 - Para acessar uma palavra (*word*) na memória, é necessário passar o **endereço de memória** a ser utilizado.

Operandos no hardware do computador

- A instrução que copia dados da memória para um registrador tradicionalmente é chamada de *load*
 - A instrução *load* é representada pelo mnemônico `lw`
- O formato da instrução *load* é:
 - o nome da operação;
 - seguido pelo registrador a ser carregado;
 - depois uma constante e
 - o registrador usado para acessar a memória.
- A soma da parte constante da instrução com o conteúdo do segundo registrador forma o endereço de memória

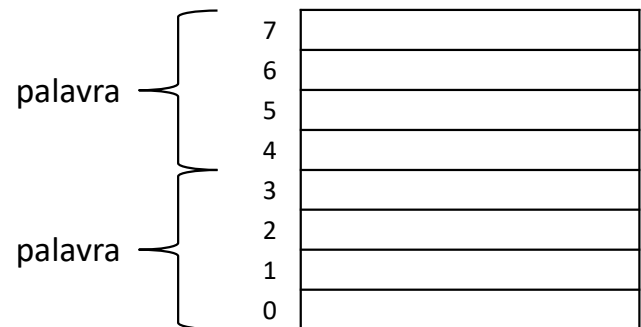
[EX] Compilando quando um operador está na memória

- *A* é uma sequência de 100 palavras e o compilador associou as variáveis *g* e *h* aos registradores `$s1` e `$s2`.
- O endereço inicial da sequência está no endereço armazenado em `$s3` (endereço base)
 - `g = h + A[8];` // código em linguagem C
- Embora haja uma única operação nessa instrução de atribuição, um dos operandos está na memória
 - Precisamos transferir `A[8]` para um registrador:
 - `$s3` contém elemento base – o endereço de `A[0]`.
 - `lw $t0, 8($s3)` #registrador temporário
 - `$t0` recebe `A[8]`
 - Constante → *offset*
 - Registrador somado ao *offset* → registrador de base
- A instrução seguinte (`add`) pode operar sobre o valor em `$t0`, já que é um registrador:
 - `add $s1, $s2, $t0`

Operandos no hardware do computador

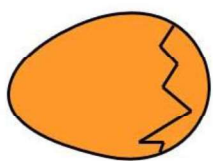
- Além de associar variáveis a registradores, o compilador aloca estrutura de dados, como vetores, em locais na memória:
 - Bytes de 8 bits são úteis em muitos programas:
 - A maioria das arquiteturas endereça bytes individuais.
 - Endereços de palavras combinam os endereços dos 4 bytes dentro da palavra (32 bits)
- No MIPS, palavras precisam começar em endereços que sejam múltiplos de 4:

– Restrição de Alinhamento.

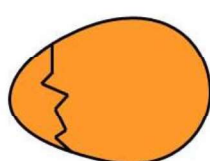


Endianness (Extremidade ou ordenação)

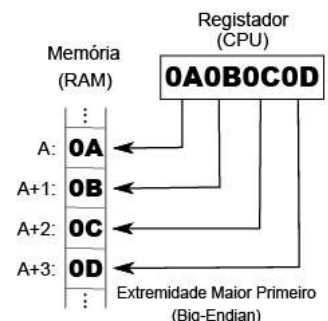
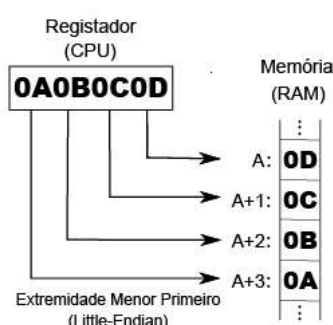
- Big-endian Vs. Little-endian*
 - Big-endian*
 - O **byte mais significativo** é armazenado no menor endereço.
 - Little-endian*
 - O **byte menos significativo** é armazenado no maior endereço.
- O nome é inspirado na história 'As viagens de Gulliver'
- Processadores DEC e IBM são *little-endian*. Motorola e SUN são *big-endian*.
- MIPS podem ser configurados para funcionar como *big-endian* ou *little-endian*.
 - [https://pt.wikipedia.org/wiki/Extremidade_\(ordenação\)](https://pt.wikipedia.org/wiki/Extremidade_(ordenação))



BIG ENDIAN - The way people always broke their eggs in the Lilliput land



LITTLE ENDIAN - The way the king then ordered the people to break their eggs



Operandos no hardware do computador

- Restrição de alinhamento:
 - No MIPS, palavras precisam começar em endereços que sejam múltiplos de 4.
- O endereçamento em bytes afeta o índice do *array*:
 - Para obter o endereço em bytes de maneira apropriada o offset necessita ser igual a $4 \times 8 = 32$
- A instrução complementar ao *load* chama-se *store*:
 - O *store* copia dados de um registrador para a memória;
 - A instrução *store* é representada pelo mnemônico *sw*.

[EX] Compilando com load e store

- Suponha que a variável *h* esteja associada ao registrador *\$s2* e o endereço base do vetor *A* esteja armazenado em *\$s3*.
 - Qual código assembly do MIPS para a instrução de atribuição *C* a seguir?
 - $A[12] = h + A[8];$
- Embora seja uma instrução na linguagem C, dois operandos estão na memória:
 - São necessárias instruções para acessar os dois operandos da memória:
 - `lw $t0, 32($s3)` `# reg. temp. $t0`
 `recebe A[8]`
 - `add $t0, $s2, $t0` `# reg. temp. $t0`
 `recebe h + A[8]`
 - `sw $t0, 48($s3)` `# armazena resultado`
 `em A[12]`

Operandos no hardware do computador

- Interface hardware/software:
 - Muitos programas têm mais variáveis do que os computadores têm registradores
 - O compilador tenta manter as variáveis mais utilizadas nos registradores e coloca as restantes na memória
 - Usando *load* e *store* para movimentar os dados
 - *Spilling registers*:
 - O processo de colocar variáveis menos utilizadas, ou aquelas que só serão empregadas mais tarde, na memória.
 - Registradores, apesar de serem mais reduzidos e terem um tamanho menor que a memória principal, são mais rápidos:
 - Define a preocupação com a utilização correta dos registradores
- Constantes ou operandos imediatos
 - Muitas vezes, os valores que necessitam ser trabalhados são passados na instrução como constantes, e não como endereços de memória;
 - Quando os dados são passados dessa forma, como constantes, é utilizado o modo de **endereçamento imediato**.

Operandos no hardware do computador

- **Trabalhando com constantes:**
 - Usando apenas instruções, teríamos de ler uma constante da memória para utilizá-la:
 - `lw $t0, EndConstante4($s1)`
 - `add $s3, $s3, $t0`
 - Supondo que `EndConstante4` seja o endereço de memória para a constante 4
 - Oferecer instruções aritméticas em que o operando seja uma constante:
 - Evita o uso de uma instrução *load* (uma leitura em memória desnecessária)
 - Essa instrução (no caso de uma soma) é chamada de `add imediato`, ou `addi`:
 - `addi $s3, $s3, 4` $\# \$s3 = \$s3 + 4$

Operandos no hardware do computador

- Trabalhando com constantes (cont.)
 - Os operandos com constantes ocorrem com bastante frequência e, incluindo constantes dentro das instruções aritméticas, as operações tornam-se mais rápidas para serem executadas.

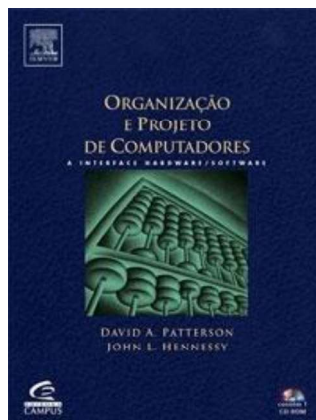
- **Princípio de projeto 3:**
 - Agilize os casos mais comuns

Assembly do MIPS

Assembly do MIPS				
Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	add	add \$s1, \$s2, \$s3	$\$s1 = \$s2 + \$s3$	3 operandos; Dados nos registradores.
	subtract	sub \$s1, \$s2, \$s3	$\$s1 = \$s2 - \$s3$	3 operandos; Dados nos registradores.
Tranf. de dados	load word	lw \$s1, 100(\$s2)	$\$s1 = \text{Memória}[\$s2 + 100]$	Dados da memória para o registrador.
	store word	sw \$s1, 100(\$s2)	$\text{Memória}[\$s2 + 100] = \$s1$	Dados do registrador para a memória.

BIBLIOGRAFIA

- PATTERSON, D.A; HENNESSY, J.L. **Organização e Projeto de Computadores: A Interface Hardware/Software**. 3a. Ed. Elsevier, 2005.
 - Capítulo 2



- Notas de aula do prof. Luciano J. Senger:
 - <http://www.ljsenger.net/classroom.html>
- Outras referências:
 - <http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/endian.html>

[FIM]

- FIM:
 - **[AULA 02]** Conjunto de instruções 1
- Próxima aula:
 - **[AULA 03]** Conjunto de instruções 2