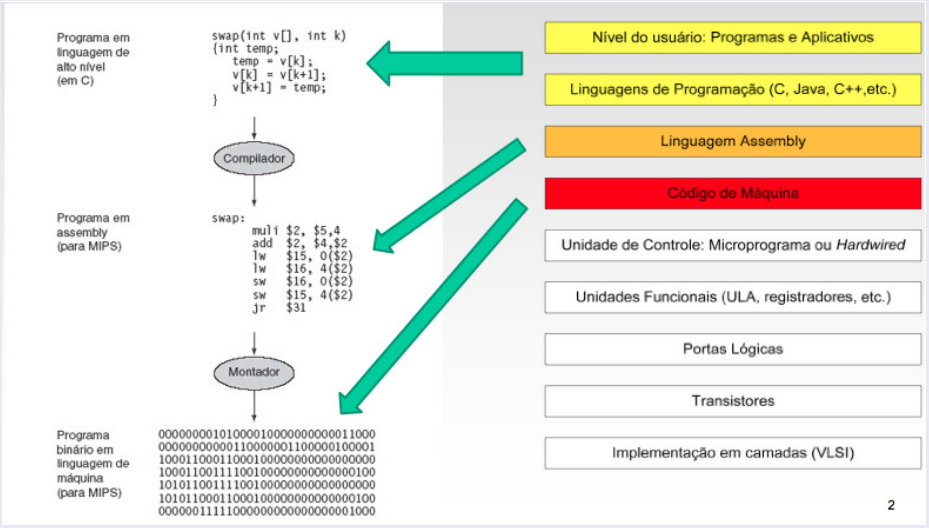
Fundamentos de Arquitetura de Computadores

Tiago Alves

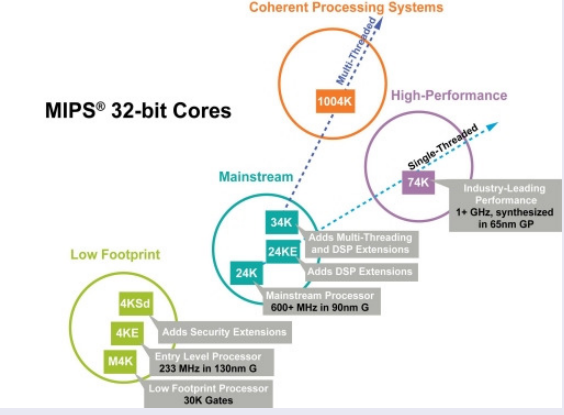
Faculdade UnB Gama
Universidade de Brasília

Abstração


Programa em linguagem de alto nível (em C)
swap(int v[], int k)
{int temp;
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
}
Compilador
Programa em assembly (para MIPS)
swap:
muli $2, $5,4
add $2, $4,$2
lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
jr $31
Montador
Programa binário em linguagem de máquina (para MIPS)
00000000101000010000000000011000
00000000000110000011000001000001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
Nível do usuário: Programas e Aplicativos
Linguagens de Programação (C, Java, C++,etc.)
Linguagem Assembly
Código de Máquina
Unidade de Controle: Microprograma ou Hardwired
Unidades Funcionais (ULA, registradores, etc.)
Portas Lógicas
Transistores
Implementação em camadas (VLSI)
2

Objetivo

Apresentar e construir a arquitetura do processador RISC MIPS R2000/3000 e exemplo de programação Assembly. (http://www.mips.com)



Objetivo

Por que Assembly para MIPS (Microprocessor Without Interlocked Pipeline Stages) ?

- Linguagem excelente para fins didáticos (extensamente coberta no livro-texto).
- Linguagem similar a ARMv7. Mais de 9 bilhões de chips com processadores ARM foram produzidos em 2011, o que tornou o conjunto de instruções mais popular do mundo!
- Ajudará no entendimento da linguagem de montagem para Intel x86, que controla a operação de dispositivos PC e da nuvem de dispositivos da era pós-PC.

Arquitetura MIPS

Baseado na arquitetura RISC (Reduced Instruction Set Computer): Computador com conjunto de instruções reduzidas.

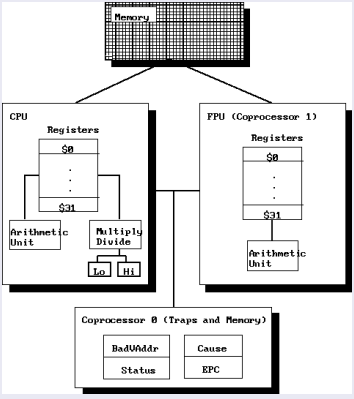
Arquitetura dos processadores MIPS R2000/R3000

Um processador MIPS consiste em uma unidade processadora de inteiros (CPU) e uma coleção de co-processadores que executam tarefas auxiliares ou operam sobre outros tipos de dados como números em ponto flutuante.

- O co-processador 0 gerencia traps (desvios e/ou chamadas de sistema), exceções e o sistema de memória virtual.
- O co-processador 1 é a unidade de ponto flutuante (FPU).

Arquitetura dos processadores MIPS R2000/R3000

- A CPU MIPS contém 32 registradores de uso geral numerados de 0 a 31. O registrador n é designado por $n.



Arquitetura dos processadores MIPS R2000/R3000

Alguns conceitos e boas práticas:

- O registrador $0 ($zero) contém sempre o valor 0 (hardwired).
- Os registradores $1 ($at), $26 ($k0) e $27 ($k1) são reservados para uso do montador e sistema operacional.
- Os registradores $2 e $3 ($v0, $v1) são utilizados para retornar valores de funções

Arquitetura dos processadores MIPS R2000/R3000

Alguns conceitos e boas práticas:

- Os registradores $4 ... $7 ($a0 ... $a3) são utilizados para passagem dos primeiros quatro argumentos para sub-programas/funções (os argumentos restantes são passados através da pilha).
- Os registradores $8...$15, $24, $25 ($t0...$t9) não são preservados pelo callee. Ou seja, convenientemente usado para receber dados temporários que não necessitam ser preservados durante as chamadas de funções/sub-rotinas.
- Os registradores $16...$23 ($s0...$s7) são callee-saved para dados que necessitam ser preservados durante as chamadas. Ou seja, o caller tem a garantia de que esses registradores não serão alterados entre chamadas de outras rotinas. (Assemelham-se funcionalmente às variáveis locais.)

Obs.: **Callee** = Função ou subrotina chamada pelo **Caller**.

Arquitetura dos processadores MIPS R2000/R3000

- O registrador $28 ($gp) é um ponteiro global que aponta para o meio de um bloco de memória de 64K, no segmento de dados estáticos.
- O registrador $29 ($sp) é o ponteiro de pilha, apontando sempre para o primeiro elemento da pilha.
- O registrador $30 ($fp) é o ponteiro de frame. Pode ser utilizado como registrador callee-saved $s8.
- registrador $31 ($ra) armazena o endereço de retorno quando é executada a instrução `jal`.

  `jal` = instrução de jump to subroutine label;

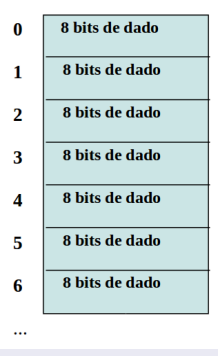
Arquitetura dos processadores MIPS R2000/R3000

Convenções de uso de registradores:

Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

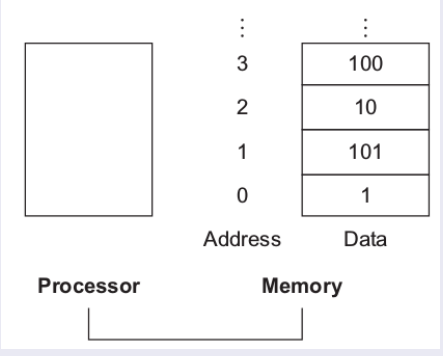
Organização da Memória

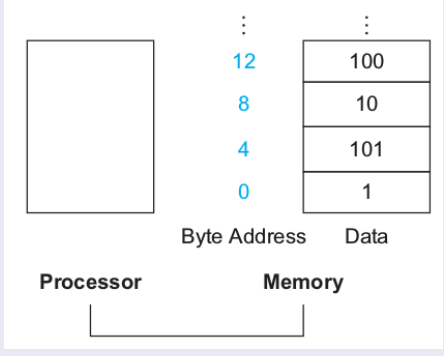
- Vista como um grande array unidimensional, com endereços sequenciais.
- Um endereço de memória é um índice no array.
- "Byte addressing" significa que o índice aponta para um byte na memória.



Organização da Memória

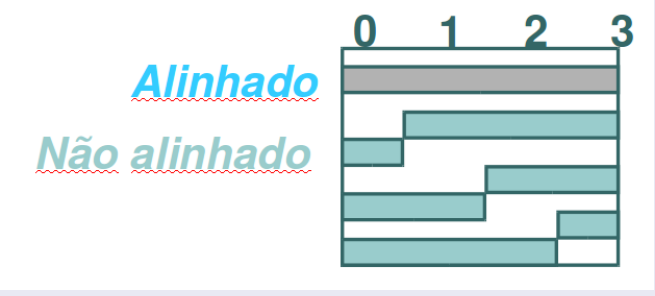
- Bytes são práticos, porém a maioria dos dados utiliza "words": unidade básica de referenciamento à memória. Definida pela arquitetura.
- Para MIPS, uma word tem 32 bits ou 4 bytes.
- Registradores armazenam 32 bits.





Organização da Memória

- Com 32 bits, é possível indexar 2^{32} bytes com endereços de byte de 0 a $(2^{32} - 1)$.
- Ou, 2^{30} words com endereços de byte de 0, 4, 8, ... $(2^{30} - 4)$
- Words são alinhadas (Restrição de Alinhamento). Quais são os valores dos 2 bits menos significativos do endereço de uma word?



Endianess or Byte Order ou Ordenamento dos Bytes

- Processadores MIPS podem operar tanto no esquema big-endian: (IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA)

- quanto little-endian: (Intel 80x86, MIPS, DEC Vax, DEC Alpha)

Modos de Endereçamento

- MIPS é uma arquitetura load/store, isto é somente instruções `load` e `store` têm acesso à memória.
- As instruções da ULA **operam somente com valores em registradores**.
- A máquina básica provê unicamente o modo de endereçamento imm (register) que utiliza como endereçamento a soma de um inteiro imediato e o conteúdo de um registrador.

Modos de Endereçamento

A máquina virtual (SPIM/MARS) oferece os seguintes modos de endereçamento para as instruções load e store:

Formato do endereço	Endereço calculado
(register)	Conteúdo do registro
imm	Inteiro imediato
imm (register)	Inteiro imediato + conteúdo do registro
symbol	Endereço de symbol
symbol +/- imm	Endereço de symbol +/- inteiro imediato
symbol +/- imm (register)	Endereço de symbol +/- (inteiro imediato + conteúdo do registro)

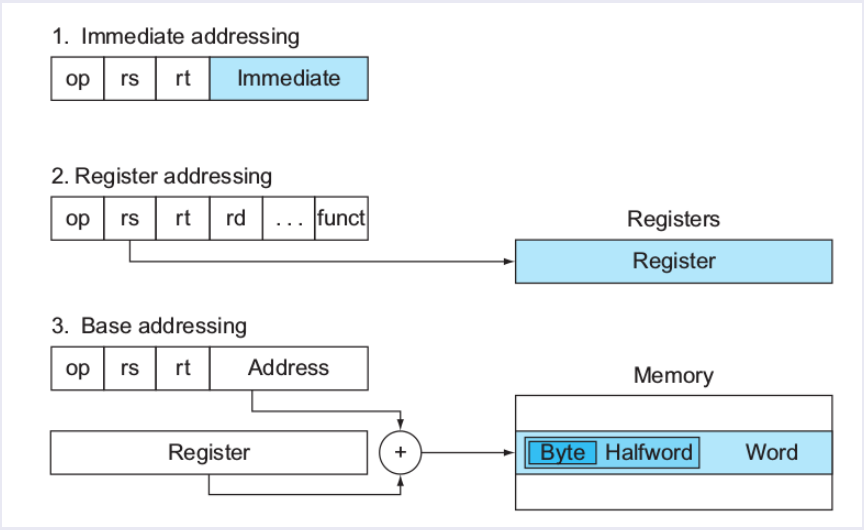
Modos de Endereçamento

Operandos constantes ou Imediatos

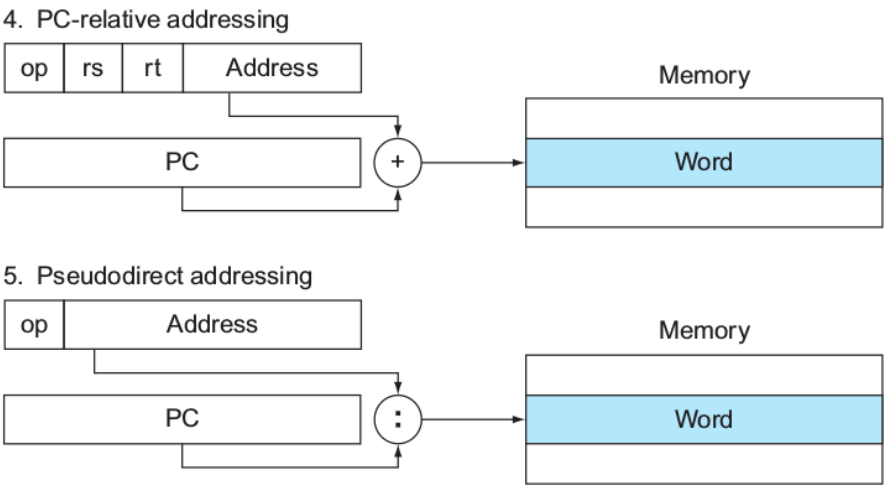
- É recorrente a necessidade de se usar uma constante em uma operação.
- Simplicidade arquitetural: evitar a carga da constante na memória para posterior transferência da mesma antes para um registrador antes de seu uso.

  Commom case fast!
- A arquitetura provê instruções que operam diretamente com constantes (imediatos)

Formato de Instruções



Formato de Instruções



Estrutura Básica de um Programa

Duas áreas distintas: `.text` e `.data`

- `.text`: área do programa (instruções) em si;
- `.data`: área para declarações de variáveis estáticas;

Áreas Independentes da ordem: Montador responsável pela colocação

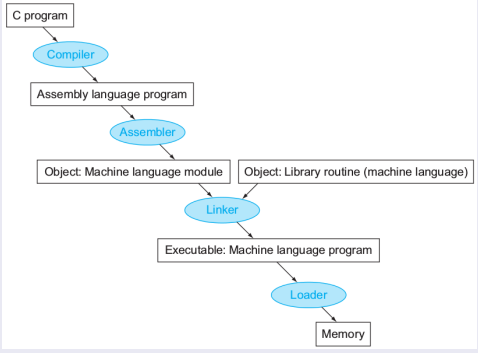
`.globl`: Declaração para rótulos globais.

Instruções

Linguagem da Máquina

- Mais primitiva que linguagens de alto nível. Sem controle de fluxo sofisticado.
- Muito restritiva: MIPS Arithmetic Instructions.

Instruções (Abstrações)



Código em Linguagem de Alto Nível

```
void
main() {
    int M, K, L, A[10];
    M = A[0];
    K = A[1];
    L = M + K;
    A[9] = L;
}
```

Código em *Assembly*

```
0: LD R0, 0
1: LD R1, 1
2: ADD R2, R0, R1
3: ST 9, R2
```

```
0: RF[0]=D[0]
1: RF[1]=D[1]
2: RF[2]=RF[0]+RF[1]
3: D[9]=RF[2]
```

Código de Máquina

Programa gravado na MI

```
0: 0000 0000 00000000
1: 0000 0001 00000001
2: 0010 0010 0000 0001
3: 0001 0010 00001001
```

Instruções (Abstrações)

Modelo de programação: MIPS ISA (Instruction Set Architecture)

- Similar a outras arquiteturas desenvolvidas desde os anos 80.

Objetivos de projeto:

- maximizar o desempenho;
- minimizar o custo e
- reduzir o tempo de projeto.

Instruções (Abstrações)

Primeiro princípio de projeto:

- **Simplicidade favorece regularidade.**

Aritmética MIPS

- Todas as instruções possuem 3 operandos
- A ordem dos operandos é fixa.

Exemplo:

- código C: `A=B+C;`
- código MIPS: `add $s0, $s1, $s2`
  (associado às variáveis pelo compilador).

Aritmética MIPS

Algumas coisas ficam mais complicadas

- código C:

```
A=B+C+D;
E=F-A;
```

- código MIPS:

```
add $t0, $s1, $s2
add $s0, $t0, $s3
sub $s4, $s5, $s0
```

Aritmética MIPS

- Operandos devem ser registradores.
- 32 registradores disponíveis.

Instruções (Abstrações)

Segundo princípio de projeto:

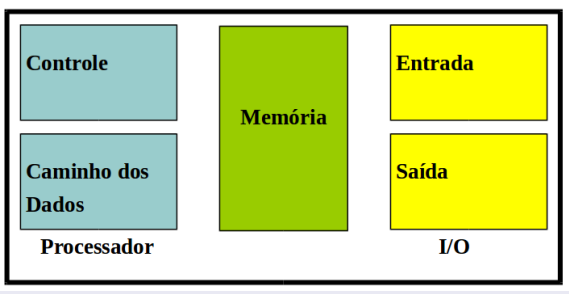
- **Menor significa mais rápido.**

Geralmente uma quantidade grande de registradores pode aumentar o tempo de ciclo de Clock

Aumenta a distância física entre sinais eletrônicos.

Registradores vs. Memória

- Operandos de instruções aritméticas devem ser registradores (32 registradores disponíveis).
- Compilador associa variáveis a registradores.
- E programas com várias variáveis, arrays, structs,...?



Manipulação de Memória MIPS

Instruções `load` e `store`

- código C: (variáveis inteiras de 32 bits)

```
A[12] = h + A[8];
```

- código MIPS:

```
lw $t0, 32($s3)     # Temporary reg $t0 gets A[8]
add $t0, $s2, $t0   # Temporary reg $t0 gets h + A[8]
sw $t0, 48($s3)     # Stores h + A[8] back into A[12]
```

`store word` tem o destino no final do enunciado.

Atenção: operandos aritméticos são registradores, não memória!

Manipulação de Memória MIPS

MIPS: Dados armazenados na memória.

- se quiséssemos somar $s3=$s3+4:

```
lw $t0, EnderecoConstante4($s1)     # $t0=constante inteira 4
add $s3,$s3,$t0
```

Porém: SPEC2000 mais da metade das operações são com constantes!

Instruções (Abstrações)

Terceiro princípio de projeto:

- **Agilize os casos mais comuns.**

Soma com imediato: `addi $s3,$s3,4 # $s3=$s3+4`

Instruções (Abstrações)

Compilar o código:

```
swap(int v[], int k);
{
   int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}

swap:
   muli $2, $5, 4
   add $2, $4, $2
   lw $15, 0($2)
   lw $16, 4($2)
   sw $16, 0($2)
   sw $15, 4($2)
   jr $31
```

Instruções (Abstrações)

Compilar o código:

```
swap(int v[], int k);
{
  int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}

swap:
    muli $v0, $a1, 4
    add $v0, $a0, $v0
    lw $t7, 0($v0)
    lw $s0, 4($v0)
    sw $s0, 0($v0)
    sw $t7, 4($v0)
    jr $ra
```