3.6 Instruções Básicas: A Linguagem da Máquina

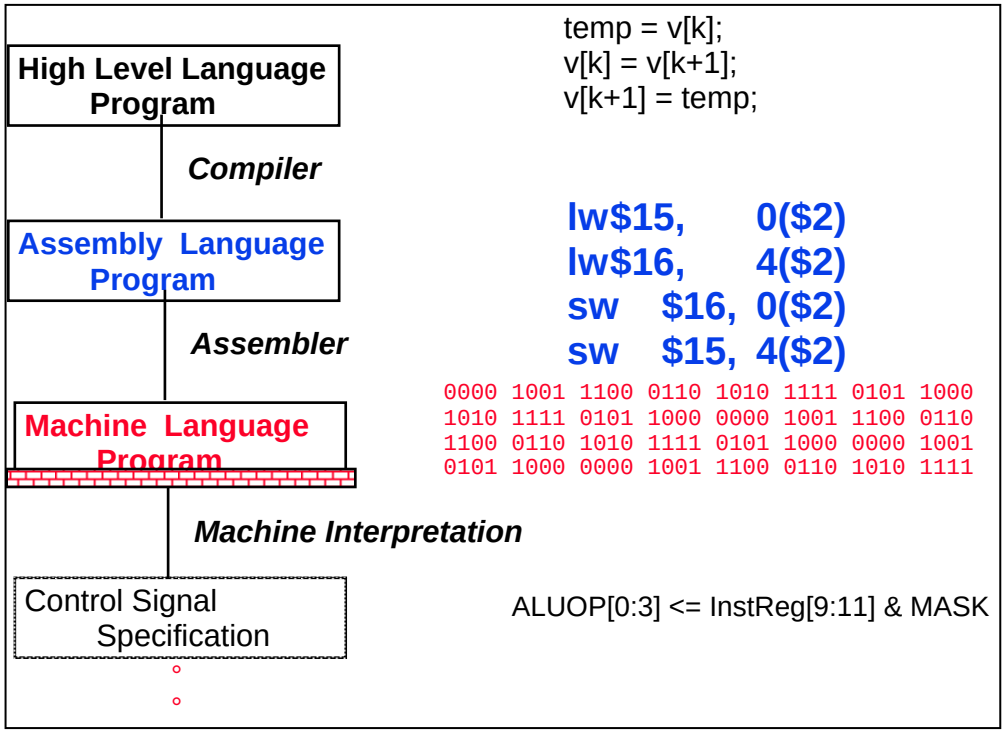
Objetivos: estudar as instruções de uma máquina real pro meio de códigos escritos em linguagem de montagem e de alto nível.

Metodologia: estudar a linguagem de máquina passo a passo até chegar a aspectos mais elaborados.

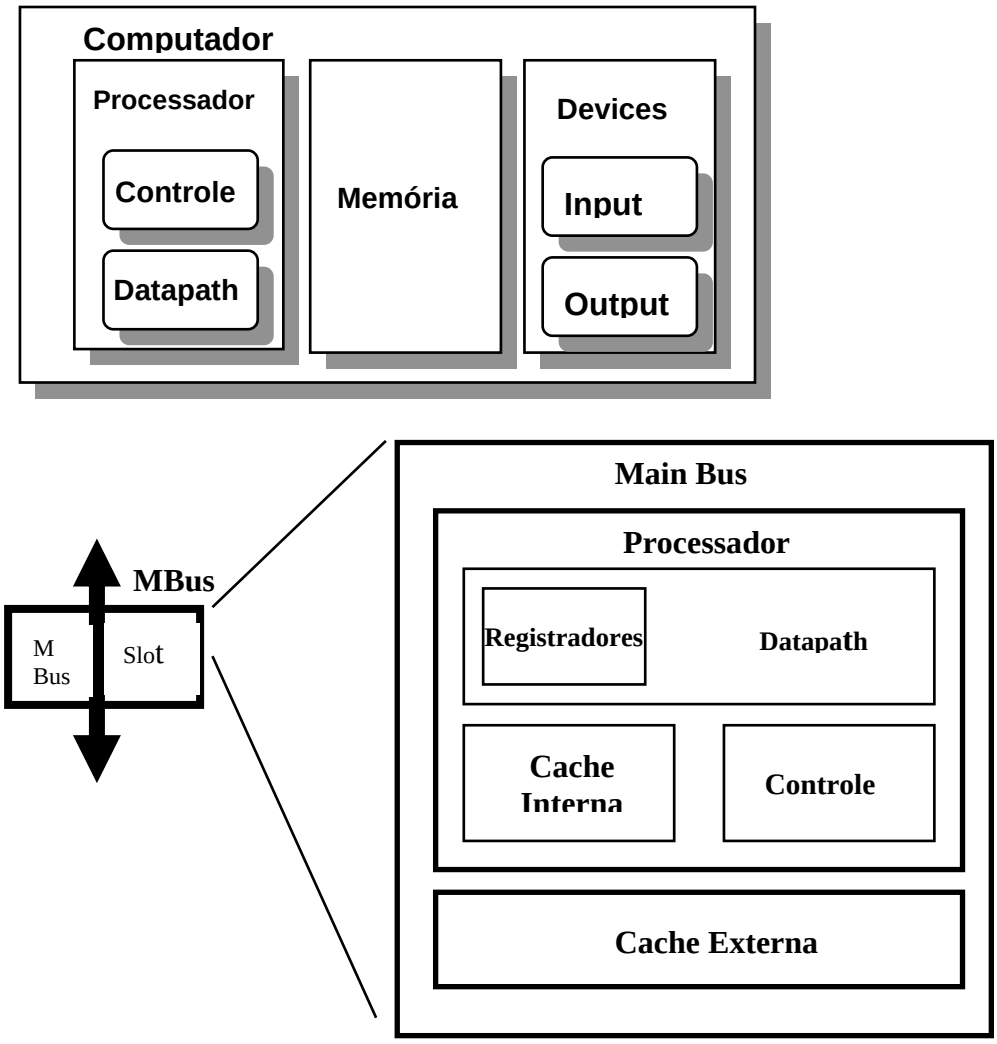
Justificativa: construir um conjunto de instruções que facilite o projeto tanto de hardware quanto de software e que maximize a performance e minimize os custos.

Arquitetura do Computador = Conjunto de Instruções (ISA) + Organização da Máquina

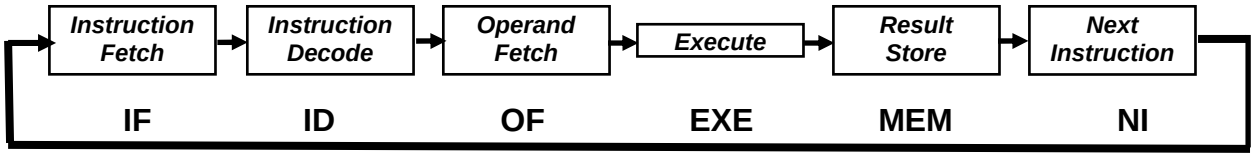
Níveis de Representação:



Níveis de organização da máquina:



Ciclo de execução:



A linguagem de alto nível a ser utilizada será a linguagem *C* e o conjunto de instruções a ser estudado será a do processador MIPS.

3.7 MIPS

Histórico:

MIPS Computer Systems, Inc. foi fundada em 1984. pela Universidade de Stanford a qual pesquisou e produziu o primeiro chip MIPS. Essa companhia foi comprada pela Silicon Graphics INC. em 1992, sendo criada a MIPS Technologies Inc. em 1998. Hoje a MIPS produz também outros dispositivos eletrônicos.

Características:

- Implementa um pequeno conjunto de instruções simples.
- Cada instrução é executada em um único ciclo de clock.
- Utiliza a técnica de pipeline.
- Utiliza 32 registradores de 32 bits.

Conjunto de Instruções: Consiste de 111 instruções, representadas em 32 bits.

O conjunto de instruções é dividido em uma variedade de instruções básicas mostradas abaixo:

- 21 Instruções aritméticas (+, -, *, /, %).
- 8 Instruções lógicas (&, |, ~).
- 8 bit Instruções de manipulação.
- 12 Instruções de comparação (>, <, =, >=, <=, ¬).
- 25 Instruções branch/jump.
- 15 Instruções de load.
- 10 Instruções de store.
- 8 Instruções de move.
- 4 Instruções diversas (miscellaneous).

3.8 Instruções Aritméticas

Soma:

```
add a, b, c # A soma do conteúdo dos registradores b e c é
            # colocada no registrador a.
```

Subtração:

```
sub a, b, c # A subtração do conteúdo dos registradores b e c
            # é colocada no registrador a.
```

Exemplos

Seja o seguinte segmento de código escrito em C que contém as seguintes variáveis `a, b, c, d e e:`

```
a = b + c  ➔ add a, b, c
d = a –e   ➔ sub d, a, e
```

Outras variáveis f, g, h, i e j. para o segmento de código:

```
f = (g + h) – (i + j)
```

Código no MIPS:

```
add t0, g, h  #t0 é uma variável temporária.
add t1, i, j  #t1 é uma variável temporária.
sub f, t0, t1 #f recebe (g + h) – (i + j)= (t0 – t1)
```

3.9 Operandos do hardware

As variáveis das linguagens de alto nível são suportadas pelo hardware através de um conjunto de localidades de memória chamados de REGISTRADORES.
O número de registradores é limitado em 32, pois um grande número de registradores poderia fazer o ciclo de clock crescer e tornar a máquina mais lenta.
O MIPS possui 32 registradores de 32 bits como projetado para muitos outros processadores..
Os registradores são numerados da seguinte forma:

- Registradores para **variáveis** de dos programas em C: **$s0, $s1, $s2, ...**
- Registradores **temporários**: **$t0, $t1, $t2, ...**

Usando o exemplo anterior:

```
f = (g + h) – (i + j)
```

Código no MIPS:

```
add $t0, $s1, $s2  #$t0 contém g + h.
add $t1, $s3, $s4  #$t1 contém i + j.
sub $s0, $t0, $t1 #f recebe $t0 - $t1, ou (g + h) – (i + j).
```

O MIPS realiza as operações aritméticas somente entre **registradores**, dessa forma ele necessita de instruções que transfiram dados entre a memória e os registradores.
Podemos enxergar a memória como um vetor unidirecional que é acessada por meio de endereços. O endereço de memória é formado por uma constante associada ao conteúdo de um registrador.
A função de transferência de dados da **memória para o registrador** é conhecida como **load**, no MIPS essa instrução é conhecida como **load word** ou simplesmente **lw.**

Exemplo: Suponha que A seja um vetor de 100 palavras, e que o compilador tenha associado as variáveis g e h aos registradores $s1 e $s2. O endereço inicial do vetor, ou endereço base, está armazenado em$s3. Escreva o código na linguagem do MIPS.

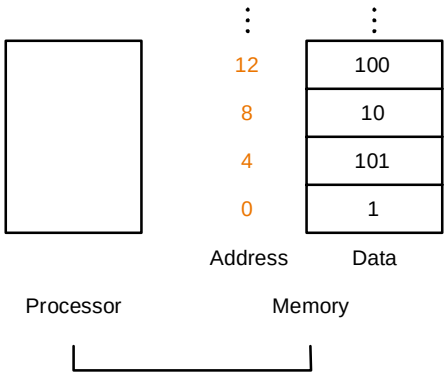
```
g = h + A[8];
```

O valor de A[8] está na memória e deve ser transferido para um registrador:

```
lw  $t0, 8($s3)  # registrador $t0 recebe A[8]
add $s1, $s2, $t0 # g recebe h + A[8]
```

A constante que aparece na instrução é denominada de deslocamento, e o registrador cujo valor armazenado é somado a essa constante é chamado de registrador base.

No MIPS o endereçamento de memória é sempre múltiplo de 4, já que cada registrador é de 32 bits, e o endereçamento é feito por bytes.



Então para obter o correto deslocamento no registrador base $s3, deveríamos 8X4 = 32, dessa forma seria deslocado para o vetor a[8] e não A[8/4].

```
lw  $t0, 32($s3)  # registrador $t0 recebe A[8]
add $s1, $s2, $t0 # g recebe h + A[8]
```

3.10 Operações de Load e Store

A transferência dos dados no **registrador para a memória** é realizada pela instrução store.

Exemplo: suponha que h está associado ao registrador $s2 e que o endereço base do vetor A esteja armazenado em $s3. Escreva o código de montagem para:

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

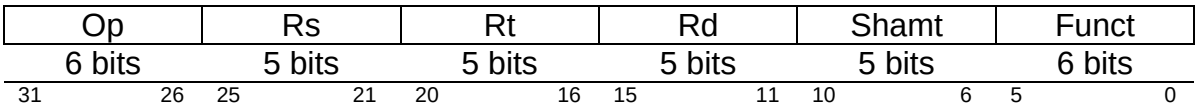
```
lw  $t0, 32($s3)  # registrador temporário $t0 recebe A[8]
add $t0, $s2, $t0 # registrador temporário $t0 recebe h + A[8]
sw  $t0, 48($s3)  # h + A[8] é armazenado de volta para a memória
                  # em A[12]
```

3.11 Tabela Resumo:

Categoria	Instrução	Exemplo	Significado	Comentário
Aritmética	add	`add $s1, $s2, $s3`	`$s1= $s2 + $s3`	3 operandos; dados em registradores
	sub	`sub $s1, $s2, $s3`	`$s1= $s2 - $s3`	3 operandos; dados em registradores
Transferência de dados	load word	`lw $s1, 100($s2)`	$s1= Memória[$s2 +100]	Dados transferidos da memória para registradores
	store word	`sw $s1, 100($s2)`	Memória[$s2 + 100] = $s1	Dados transferidos do registrador para a memória

3.12 Representação das Instruções

Tipo R – registrador:



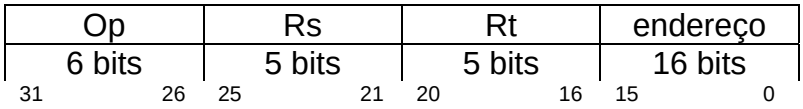
Op – operação básica a ser realizada pela instrução, conhecida como *Opcode*.
Rs – registrador que contém o primeiro operando fonte.
Rt - registrador que contém o segundo operando fonte.
Rd - registrador que guarda o resultado da operação, conhecido como registrador-destino.
Shamt – quantidade de bits deslocados.
Funct – especifica uma variação da operação apontada no campo **op**, conhecida como código de função.

Exemplo: `add Rd, Rs, Rd`

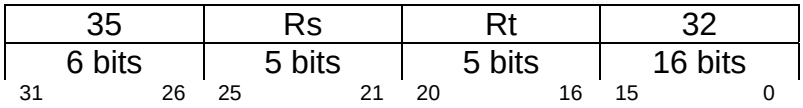
Os registradores são reservados de acordo com uma convenção:

- $s0 a $s7 → registradores de dados de16 a 23.
- $t0 a $t7 → registradores temporários de 8 a 15.

Tipo I – instrução:



Exemplo: `lw Rt, 32(Rs)`



3.13 Instruções de Desvio

Comandos condicionais usados na tomada de decisões, usando a combinação dos comandos *if,* um *go to* e *labels*.

Branch equal:

```
beq Rs, Rt, endereço
beq Registrador1, Registrador2, Label
```

Branch not equal:

```
bne Rs, Rt, endereço
bne Registrador1, Registrador2, Label
```

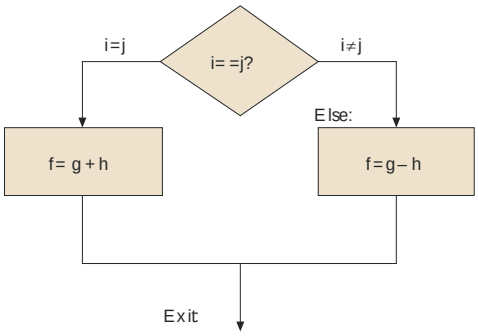
Exemplo: Seja o código C, os registradores de $s0 a $s4 se referem às variáveis de f até j.

```
          if (i == j) go to L1;
          f = g + h;
     L1: f = f – i;
```

```
beq $s3, $s4, L1; # desvia para L1 se i = j
add $s0, $s1, $s2; # f = g + h
L1: sub $s0, $s0, $s3; # f = f – i , e L1 Label com o endereço de
                       # desvio
```

Exemplo: Seja o código C abaixo:

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




```
bne $s3, $s4, Else; # desvia para Else se i <> j
add $s0, $s1, $s2; # f = g + h
j Exit # desvia para a saída
Else: sub $s0, $s1, $s2; # f = g – h , salta se i == j
                         # e Else label com o endereço de desvio
Exit:
```