# Arquitetura de Computadores

## Memória

A memória é um espaço usado para armazenar dados que são usados em operações pelo processador.

Existem vários tipos de memórias e conforme aumenta a capacidade de armazenamento, tamanho e distância do processador, menor será a velocidade de acesso.

Registradores: espaço de memória no processador usado para armazenar o resultado de operações.

**Cachê:** memória usada para armazenar os dados mais acessados pelo processador.

**RAM:** armazena dados de programas em execução de forma temporária.

**BIOS:** armazena dados de inicialização do sistema e parâmetros como a data e a hora.

**Disco rígido, pendrive, CD/DVD:** memórias secundárias usadas para armazenar os dados de forma permanente.

## Sistemas numéricos

**Binário:** representado por 2 símbolos {0,1} e é o sistema numérico usado na linguagem de máquina, circuitos lógicos, sinais elétricos, etc.

**Decimal:** representado por 10 símbolos {0,1,2,3,4,5,6,7,8,9} e é o sistema numérico utilizado no dia-a-dia.

**Octal:** representado por 8 símbolos {0,1,2,3,4,5,6,7,8}.

**Hexadecimal:** representado por 16 símbolos {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}.

### Conversão de qualquer base para decimal

Basta fazer a soma da multiplicação de cada algarismo pelo número da base elevado à posição do algarismo.

**Por exemplo:** 10111(2) = 1 x 2^4 + 0 x 2^3 + 1 x 2^2 + 1 x 2^1 + 1 x 2^0 = 23(10)

### Conversão de decimal para qualquer base

Divide o número decimal consecutivas vezes pelo número da base até o quociente 1. Depois disso, agrupa o último quociente e os restos da divisão ao contrário e obtém-se o número na base desejada.

### Representação binária de ponto flutuante

Cada dígito decimal vira um expoente negativo de base 2.

**Por exemplo:** 0,5(2) = 1 x 2^(-1) = 0.1(2)

## MIPS

Arquitetura de conjunto de instruções usado para criar códigos mais próximos da linguagem de máquina.

O MIPS é baseado na arquitetura de John Von Neumann que é dividida em 4 partes: entrada de dados, processamento, memória e saída de dados.

Cada instrução do MIPS é transformada em código binário (0 e 1). Um dos formatos de conversão é o formato do tipo R que consiste em 5 campos (opcode, primeiro registrador fonte, segundo registrador fonte, registrador de destino, shift, function).

**Observação:** Toda instrução no MIPS tem tamanho de 32 bits, ou seja, 4 bytes.

O MIPS possui 32 registradores que são identificados por nomes que começam com $ (parecido com PHP).

O nome do registrador identifica o tipo de uso que deve ser feito com ele, conforme descrito abaixo.

**$v:** passa valores como parâmetro para chamada de sistema e recebe o resultado de funções.

**$a:** passa valores como argumento

**$t:** recebe o resultado de operações

**$s:** armazena valores de forma temporária e que são usados como operandos em operações aritméticas.

**$f:** armazena valores do tipo float (registradores ímpares) e double (registradores pares).

**Lo:** armazena os bits menos significativos de uma multiplicação e o quociente de uma divisão.

**Hi:** armazena os bits mais significativos de uma multiplicação e o resto de uma divisão.

## Assembly

Linguagem de programação de baixo nível.

Um código em assembly é dividido em duas partes data e text. Em data são declaradas as variáveis, tipos e valores. Enquanto em text são declaradas todas as instruções do programa.

Overflow

Ocorre quando um tipo de dado não possui tamanho suficiente para armazenar o resultado de uma operação, resultando em uma exceção (erro).

Por exemplo: float possui tamanho de 32 bits e double 64 bits. Caso o resultado de uma operação com valores do tipo double seja armazenado em um registrador do tipo float, a execução incorrerá em erro.

### Tipos de dados

Cada tipo de dado no MIPS possui um tamanho de bits próprio para o dado que será armazenado.

**Bit:** boolean (1 bit).

**Half-word:** short (16 bits ou 2 bytes).

**Word:** inteiro (32 bits ou 4 bytes).

**Double-word:** long (64 bits ou 8 bytes).

**Byte:** caracteres (8 bits ou 1 byte).

**Space:** aloca um espaço fixo de bytes na memória.

**Float:** ponto flutuante de precisão simples (32 bits ou 4 bytes).

**Double:** ponto flutuante de precisão dupla (64 bits ou 8 bytes).

### Principais instruções

Segue abaixo as principais instruções do MIPS Assembly.

**LW registrador\_destino, RAM\_origem**

Carrega um valor da memória RAM para dentro do registrador.

Exemplo: lw $s0, numero1

**SW registrador\_origem, RAM\_destino**

Armazena o valor de um registrador em um espaço da memória RAM.

Exemplo: sw $t0, resultado

**LI registrador\_destino, valor**

Carrega de forma direta um valor para dentro do registrador.

Exemplo: li $v0, 10

**ADD registrador\_destino, registrador\_origem, registrador\_origem**

Adiciona os valores armazenados nos registradores de origem e carrega o resultado para dentro do registrador de destino.

Exemplo: add $t0, $s0, $s1

**ADDU registrador\_destino, registrador\_origem, registrador\_origem**

Adiciona dois números ignorando os sinais e guarda o resultado no registrador de destino.

Exemplo: addu $t0, $s0, $s1

**ADDI registrador\_destino, registrador\_origem, registrador\_origem**

Adiciona dois valores de um registrador de origem e um valor bruto e guarda o resultado em um registrador de destino.

Exemplo: addi $t0, $s0, 5

**SUB registrador\_destino, registrador\_origem, registrador\_origem**

Subtrai dois números e guarda o resultado em um registrador de destino. Caso ocorra overflow, não será armazenado valor algum no registrador de destino e incorrerá em erro.

Exemplo: sub $t0, $s0, $s1

**ABS registrador\_destino, registrador\_origem**

Determina o valor absoluto do número, ou seja, desconsidera o sinal e guarda o resultado no registrador de destino.

Exemplo: abs $s1, $s0

**MULT registrador\_origem, registrador\_origem**

Multiplica dois valores e guarda o resultado nos registradores LO e HI.

Exemplo: mult $s0, $s1

**MULTU registrador\_origem, registrador\_origem**

Multiplica dois valores ignorando os sinais e guarda o resultado nos registradores LO e HI.

Exemplo: multu $s0, $s1

**MUL registrador\_destino, registrador\_origem, registrador\_origem**

Multiplica dois valores e guarda o resultado no registrador de destino. Essa pseudoinstrução não trata o overflow.

Exemplo: mul $t0, $s0, $s1

**MULO registrador\_destino, registrador\_origem, registrador\_origem**

Multiplica dois valores e guarda o resultado no registrador de destino. Essa pseudoinstrução trata o overflow.

Exemplo: mul $t0, $s0, $s1

### Valores do tipo ponto flutuante

Operações aritméticas com dados do tipo ponto flutuante devem ser separadas pelo identificador “s” (precisão simples) ou “d” (precisão dupla).

Por exemplo:

add.s $f5, $f1, $f3 (adiciona dois números ponto flutuante de precisão simples)

mulo.d $f4, $f0, $f2 (multiplica dois números ponto flutuante de precisão dupla, tratando o overflow)

Estruturas de controle

Servem para mudar o fluxo de execução do programa baseado em condições e loops.

BEQ registrador\_origem, registrador\_origem, label

Compara os valores em dois registradores e, caso sejam iguais, pula a execução para a label especificada.

Exemplo: beq $s0, $s1, imprimeValoresIguais

J label

Pula a execução de forma incondicional para a label especificada.

Exemplo: j while

Endereçamento

O endereçamento de memória permite a criação de arrays.

LA $t0, var1

Carrega o endereço de memória de var1 para dentro do registrador $t0.

Exemplo: la $t0, numero1

Acesso ao endereço

LW $t1, ($t0)

Para fazer referência ao espaço da memória usando o seu endereço, o endereço deve ser passado como parâmetro dentro de parênteses.

Exemplo: lw $t1, ($t0)

Arrays

Sequência de espaços na memória usados para armazenar mais do que um valor.

Criação de array

Array1: .space 40

Aloca um espaço na memória de 40 bits (10 bytes) para armazenar 10 inteiros.

Indexação

LA $t0, array1

LI $t1, 0

LI $t2, 1

SW $t1, ($t0)

Guarda o valor do registrador $t1 na posição 0 do array1.

SW $t2, 4($t0)

Guarda o valor do registrador $t2 na posição 4 do array1.

Observação: a indexação é feita usando o número do bit onde o valor vai ser armazenado. Como estamos guardando valores do tipo inteiro, lembrando que um inteiro possui tamanho de 4 bits, a indexação é feita de 4 em 4. Caso estivéssemos trabalhando com um array de double, a indexação seria de 8 em 8.