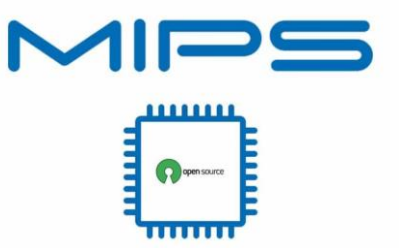
Faculdade de Computação

Arquitetura e Organização de Computadores 1

Laboratório de Programação Assembly 2

Prof. Cláudio C. Rodrigues

Programando a Arquitetura MIPS



MIPS, acrônimo para **Microprocessor without Interlocked Pipeline Stages** (microprocessador sem estágios intertravados de pipeline), é uma arquitetura de microprocessadores RISC desenvolvida pela MIPS Computer Systems.

Arquitetura

O MIPS é uma arquitetura baseada em registrador, ou seja, a CPU usa apenas registradores para realizar as suas operações aritméticas e lógicas. Existem outros tipos de processadores, tais como processadores baseados em pilha e processadores baseados em acumuladores. Processadores baseados no conjunto de instruções do MIPS estão em produção desde 1988. Ao longo do tempo foram feitas várias melhorias do conjunto de instruções. As diferentes revisões que foram introduzidas são MIPS I, MIPS II, MIPS III, MIPS IV e MIPS V. Cada revisão é um super conjunto de seus antecessores. Quando a MIPS Technologies saiu da Silicon Graphics em 1998, a definição da arquitetura foi alterada para definir um conjunto de instruções MIPS32 de 32 bits e um MIPS64 de 64 bits.

Projetos MIPS são atualmente bastante usados em muitos sistemas embarcados como dispositivos Windows CE, roteadores Cisco e consoles de games como: Nintendo 64, Playstation, Playstation 2 e Playstation Portable.

A Filosofia do Projeto do MIPS:

- A simplicidade favorece a regularidade.
  O menor é (quase sempre) mais rápido. Levando em conta que uma corrente elétrica se propaga cerca de um palmo por nanosegundo, circuitos simples são menores e portanto, mais rápidos.
- Um bom projeto demanda compromissos.
- O caso comum DEVE ser mais rápido.
  É muito melhor tornar uma instrução que é usada 90% do tempo 10% mais rápida do que fazer com que uma instrução usada em 10% das vezes torne-se 90% mais rápida. Esta regra é baseada na "Lei de Amdahl".

Instruções:

I. Apresentar as soluções usando a linguagem de montagem da Arquitetura de processadores MIPS.
II. O trabalho deve ser desenvolvido em grupo composto de 1 até 4 (um até quatro) estudantes e qualquer identificação de plágio sofrerá penalização;
III. Entrega dos resultados deverá ser feita por envio de arquivo zipado com os seguintes artefatos de software: Memorial descritivo das soluções em pdf; arquivo em txt com os códigos que solucionam os problemas propostos escritos em MIPS assembly.
IV. Submeter os documentos na plataforma MS Teams, dentro do prazo definida para a atividade.

Desafios de Programação MIPS:

P1. Converta o fragmento de código abaixo, escrito em linguagem C, para a linguagem do MIPS assembly.

```
/** Returns the number of bytes in S before, but not counting, the null
terminator. */
size_t string_length(char *s) {
    char *s2 = s;
    while(*s2++);
    return s2 - s - 1;
}
```

P2. Converta o fragmento de código abaixo, escrito em linguagem C, para a linguagem do MIPS assembly.

```
/** Converts the string S to lowercase */
void string_to_lowercase(char *s) {
    for(char c = *s; (c=*s) != '\0'; s++) {
         if(c >= 'A' && c <= 'Z') {
              *s += 'a' - 'A';
         }
    }
}
```

P3. Escreva em *MIPS assembly* uma função denominada ***senox***, que calcula o seno de ***x***. A função seno de x pode ser calculado utilizando a expansão da *Série de Taylor*:

$$\boldsymbol{seno\ x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots}$$

O elemento ***x*** da *Série de Taylor* deve ser em radianos. Portanto, a função ***senox*** deve ler o valor do ângulo em graus e converter para radianos. Para realizar a conversão utilize a regra de três, considerando que PI radianos é igual a 180° e que a constante PI assume valor de 3.141592.
A função deve somar os termos da série até aparecer um termo cujo valor absoluto seja menor que 0.00001 (**precisão**). Isto é, o termo $\left|\frac{x^k}{k!}\right|$ tende a zero quando **k** tende a **+∞**. Repetir o cálculo dos termos até um **k** tal que $\left|\frac{x^k}{k!}\right| < \mathbf{0.00001}$.
Dica: Use a convenção de registrador de ponto flutuante MIPS para passar o parâmetro **x** e retornar o resultado da função. Evite calcular o valor do fatorial, calcule um termo da série usando o termo anterior.

P4. Escreva em *MIPS assembly* uma função denominada ***cosex***, que calcula o *cosseno* de ***x***. A função cosseno de ***x*** pode ser calculado utilizando a expansão da série de Taylor:

$$\boldsymbol{cosseno\ x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots}$$

O elemento ***x*** da série de Taylor deve ser em radianos. Portanto, a função ***cosex*** deve ler o valor do ângulo em graus e converter para radianos. Para realizar a conversão utilize a regra de três, considerando que PI radianos é igual a 180° e que a constante PI assume valor de 3.141592.
A função deve somar os termos da série até aparecer um termo cujo valor absoluto seja menor que 0.00001 (**precisão**). Isto é, o termo $\left|\frac{x^k}{k!}\right|$ tende a zero quando **k** tende a **+∞**. Repetir o cálculo dos termos até um **k** tal que $\left|\frac{x^k}{k!}\right| < \mathbf{0.00001}$.
Dica: Use a convenção de registrador de ponto flutuante MIPS para passar o parâmetro **x** e retornar o resultado da função. Evite calcular o valor do fatorial, calcule um termo da série usando o termo anterior.

P5. Escreva em *MIPS assembly* uma função denominada ***quicksort***, que classifica um array recursivamente. Traduza a função a seguir, em código MIPS Assembly. Escreva uma função principal para chamar e testar a função *quicksort*.

```
void quicksort(int* array, int low, int high)
{
    int i = low, j = high;                      // low and high index
    int pivot = array[(low+high)/2];            // pivot = middle value
    while (i <= j)
    {
        while (array[i] < pivot) i++;
        while (array[j] > pivot) j--;
        if (i <= j)
        {
            int temp=array[i];
            array[i]=array[j];                  // swap array[i]
            array[j]=temp;                      // with array[j]
            i++;
            j--;
        }
    }
    if (low < j) quick_sort(array, low, j);     // Recursive call 1
    if (i < high) quick_sort(array, i, high);   // Recursive call 2
}
```