TRABALHO PRÁTICO III

**Disciplina**: Arquitetura e Organização de Computadores

**Professor**: Saulo Henrique Cabral Silva

**Alunos**: Eduardo Octávio de Paula e Rodolfo Oliveira Miranda

Assembly – Troca de Vagões

1. **Introdução**

A linguagem Assembly MIPS é uma linguagem de baixo nível usada para programar diretamente microprocessadores MIPS. Esses processadores são amplamente utilizados em sistemas embarcados, dispositivos de redes, e até em alguns sistemas educacionais devido à sua simplicidade e eficiência. A arquitetura MIPS é baseada em um conjunto de instruções RISC, o que significa que ela possui um conjunto reduzido de instruções, facilitando a otimização do desempenho e a simplificação do design do processador.

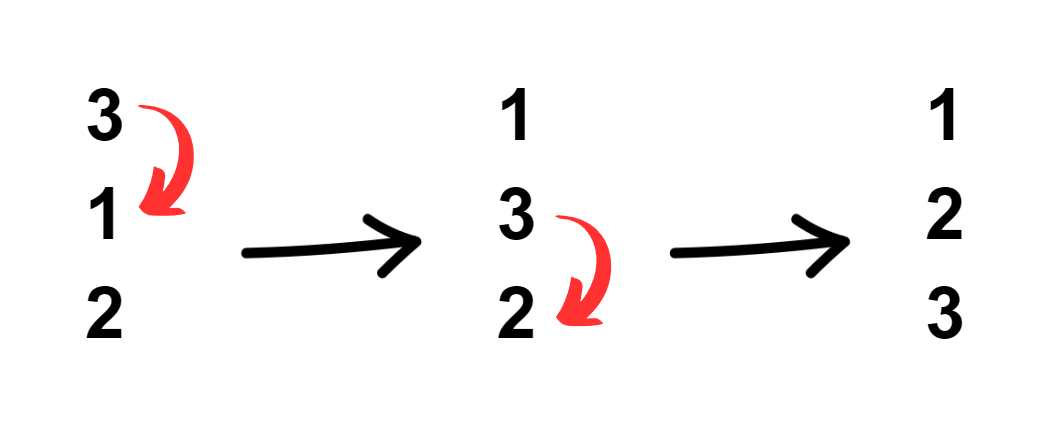
* 1. **Arrays em MIPS**

Em MIPS, arrays são estruturas de dados sequenciais que armazenam múltiplos elementos do memos tipo, como inteiros, em posições contíguas na memória. Cada elemento do array é acessado por meio de um índice, que representa o deslocamento em relação ao início do array. No MIPS, trabalhar com arrays envolve manipular endereços de memória diretamente.

Para acessar elementos em um array em MIPS, é preciso calcular o endereço do elemento desejado. Isso é feito somando o deslocamento do elemento (o índice multiplicado pelo tamanho de cada elemento, por exemplo, 4 bytes para inteiros) ao endereço base do array.

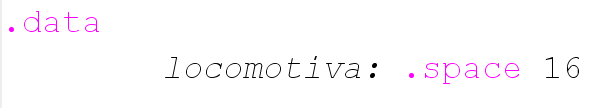
* 1. **Desafio Proposto**

Para este trabalho, foi proposto a ordenação de arrays, considerado no contexto como uma locomotiva, em que seus índices são os vagões. Cada vagão possui uma marcação (número inteiro) e o desafio é ordena-los de forma que as trocas devem ser feitas apenas com vagões adjacentes e que seja a menor quantidade de trocas possíveis.

Exemplo de caso: Dado uma locomotiva com vagões marcados da seguinte ordem: 3, 1 e 2. As trocas devem ocorrer da seguinte maneira:

1. **Implementação no MIPS**
   1. **“.data”**

No *.data* do algoritmo, foi definido um array na memória do tipo *.space* (utilizado para alocar *words*, que possuem tamanho de 4 bytes), chamado de “locomotiva”. No código, foi definido o tamanho como 16 bytes, entretanto, é possível aumenta-lo.

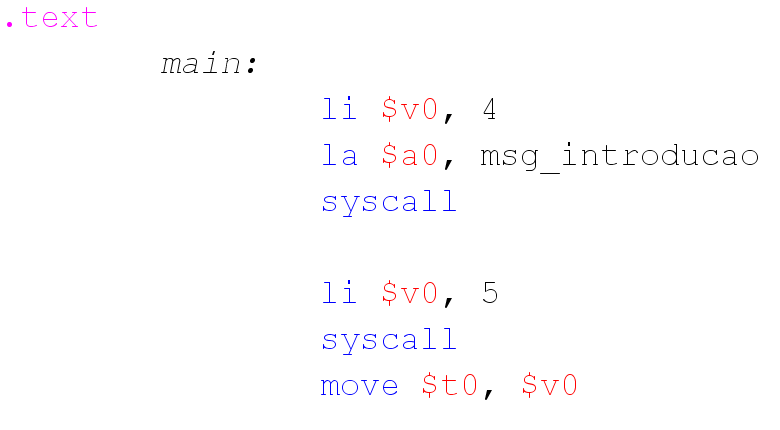


* 1. **“.text”**

No *.text* do algoritmo, foi definido algumas funções utilizadas para requisitar dados ao usuário, iniciar o array, verificar casos de trocas, executar as trocas e exibir o resultado.

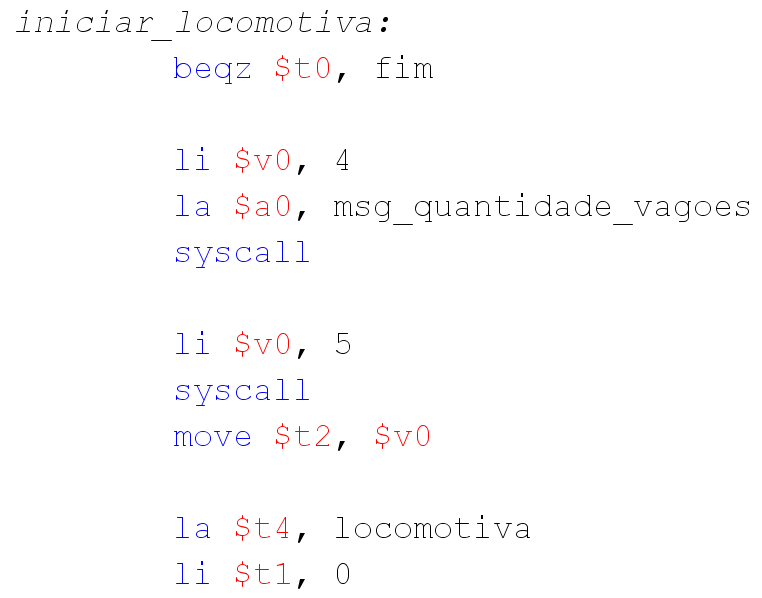
* + 1. **“main”**

Na função *main,* é solicitado ao usuário o primeiro input, que consiste na quantidade de casos que serão executados. No algoritmo, utilizamos o registrador *$t0* para armazenar a quantidade de casos e também como contador para a próxima função, sendo decrementado em um quando uma locomotiva ter sido ordenada. Quando chegar em zero, o programa é finalizado.



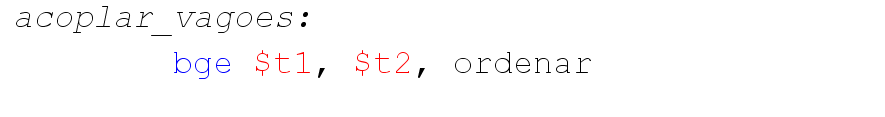
* + 1. **“iniciar\_locomotiva”**

A função *iniciar\_locomotiva,* é utilizada para iniciar o array (locomotiva), solicitando ao usuário o seu tamanho. Porém, antes disso, é verificado se a quantidade de locomotivas a serem iniciadas é igual a zero, e se for verdadeiro, o código é finalizado. Caso contrário, o algoritmo é continuado e se pede ao usuário a quantidade de vagões.



* + 1. **“acoplar\_vagoes”**

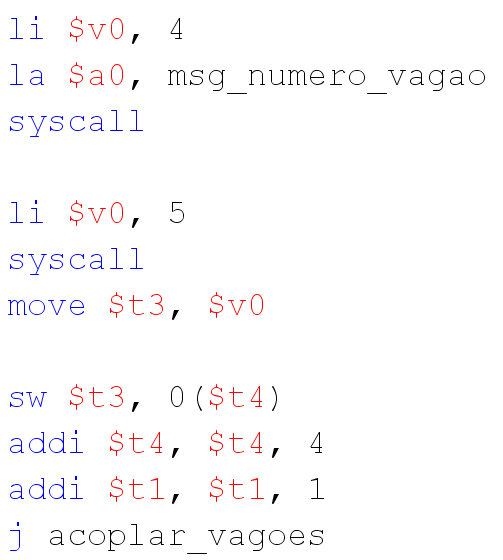
Nesta função, primeiramente, é verificado se todos os vagões já foram preenchidos. Caso verdadeiro, a função de ordena-los é chamada, e caso contrário, mais um vagão é acoplado.



Para acoplar um vagão, é solicitado ao usuário sua marcação (número inteiro). Caso seja o primeiro vagão, o endereço de memória onde será registrado é o inicio da locomotiva. Caso contrário, este endereço será a soma entre o endereço que está no registrador e 4 bytes.

Por exemplo, considerando que já foi adicionado um vagão, o endereço do segundo a ser adicionado será o endereço de memória da locomotiva mais 4 bytes. Para ser adicionado o terceiro, o endereço que este será adicionado será o a última soma com mais 4 bytes, e assim por diante.

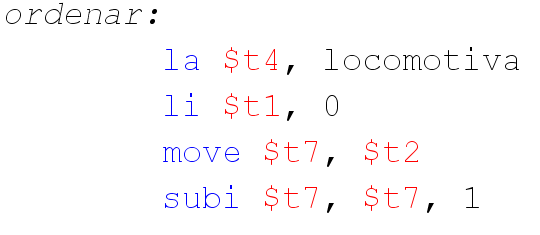
Finalmente, é incrementado um no contador de vagões e recomeça o loop para acoplar o vagão seguinte ou não.



* + 1. **“ordenar”**

Na função ordenar, recarregamos o inicio da locomotiva, instanciamos um novo contador, começado em zero, e guardamos em um novo registrador o tamanho do array subtraído de um.

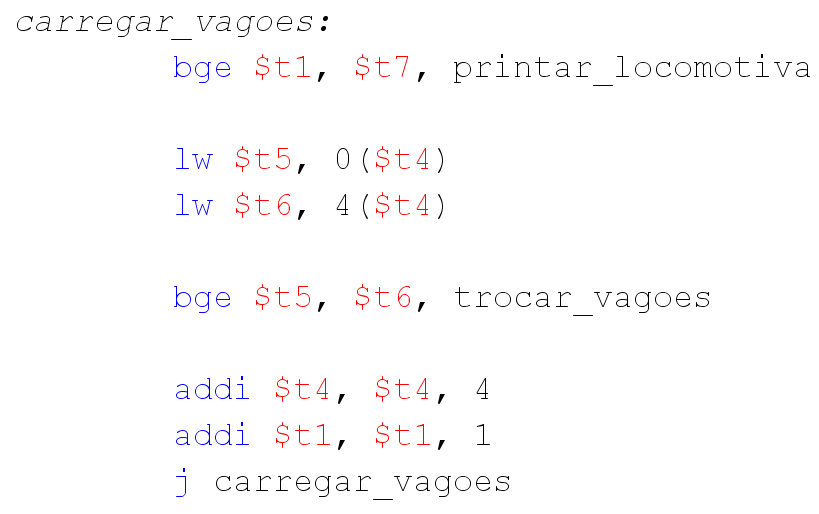
Em seguida, a função de carregar os vagões a serem trocados ou não é iniciada.

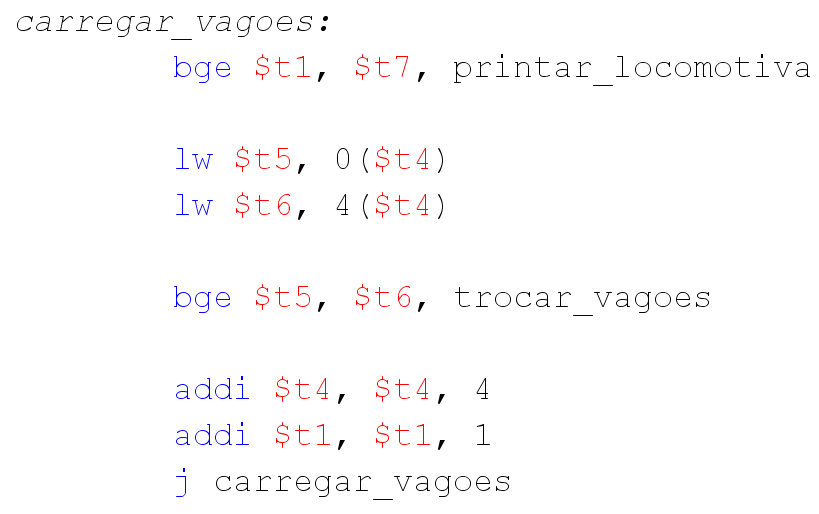


* + 1. **“carregar\_vagoes”**

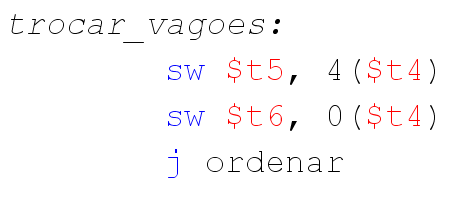
Nessa função, primeiro verifica se o contador ultrapassou a quantidade de vagões. Caso verdadeiro, significa que a locomotiva já foi ordenada e já pode ser exibida. Caso contrário, a lógica de trocar vagões é continuada.

Inicialmente, carregamos o primeiro valor que está no início do endereço do array daquele ciclo no registrador *$t5* e o segundo no registrador *$t6.* Depois, é verificado se o primeiro é maior que o segundo. Caso seja falso, significa que os valores já estão ordenados e não precisam ser trocados, portanto, apenas é adicionado mais 4 bytes ao endereço do array, passando pros próximos valores a serem comparados, e incrementado um no contador de vagões. Finalmente, o loop reinicia-se.



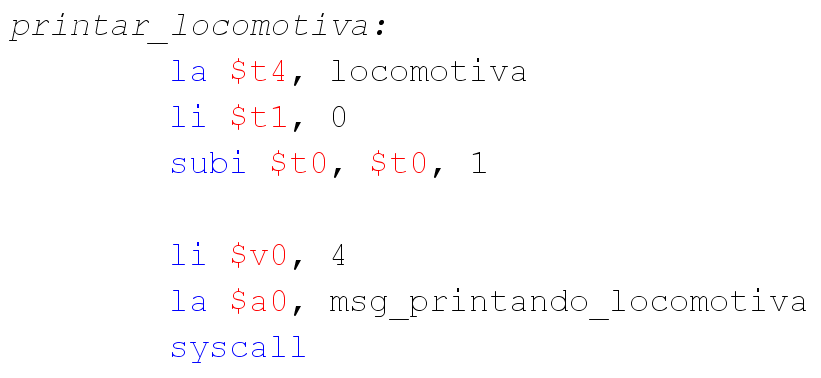


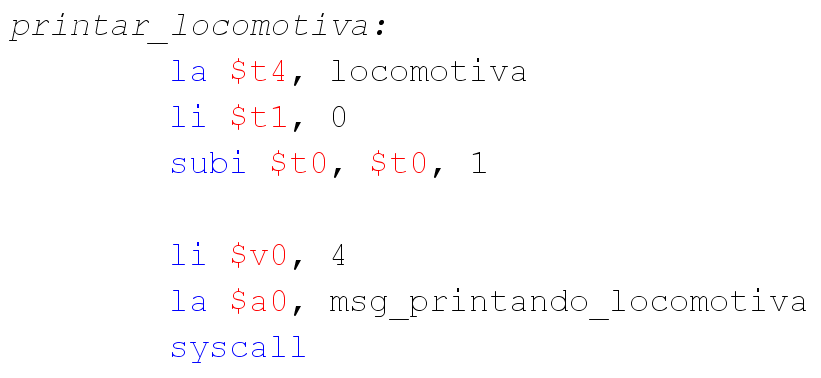
Caso seja verdadeiro, é chamada a função que faz as trocas dos vagões. Para fazer essa troca, simplesmente, copiamos o valor que está no endereço de memória do clico incrementado de 4 bytes para o registrador $t5, que é o primeiro elemento, e depois copiamos o valor que está no endereço de memória do inicio do array do ciclo ao registrador $t6, que é o segundo elemento. Por fim, recomeçamos o loop de ordenação.



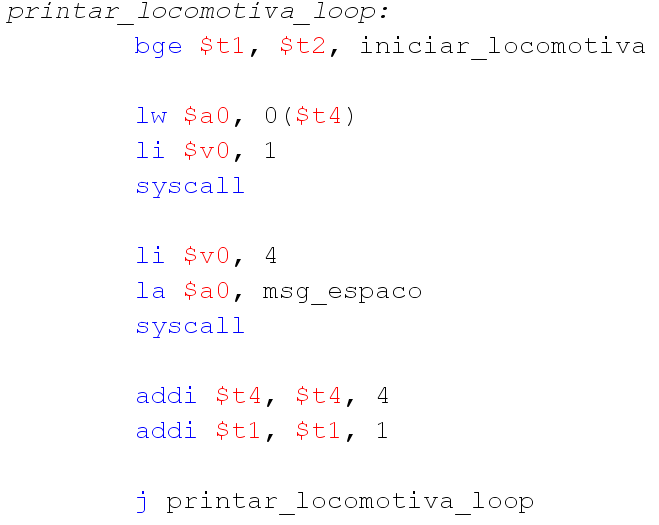
* + 1. **“printar\_locomotiva”**

Para exibir a locomotiva ordenada, primeiro recarregamos o início da locomotiva, instanciamos um novo contador, começado em zero, e decrementamos em um a quantidade de locomotivas a serem ordenadas.

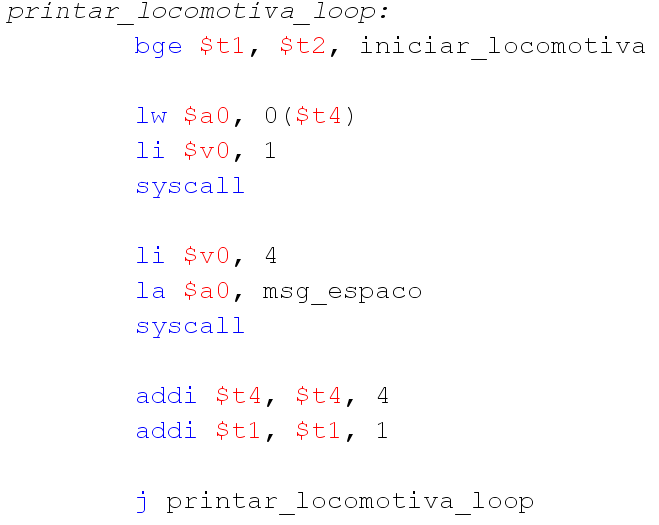




Quando é iniciado o loop para exibir os vagões da locomotiva, primeiro, verifica se o contador já excedeu o tamanho desta. Caso verdadeiro, é iniciado outra locomotiva para ser ordenada, e caso contrário, o próximo vagão é exibido.

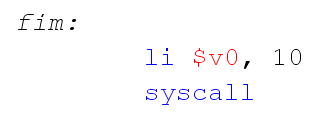


Após exibir o vagão, é incrementado mais 4 bytes ao endereço de memória da locomotiva e incrementado, também, mais 1 no contador de vagões. Por fim, é reiniciado o loop.



* + 1. **“fim”**

Como o nome já indica, a função *fim* termina a execução do programa.



1. **Conclusão**

Em suma, apesar das instruções simples da arquitetura MIPS, a implementação de um algoritmo para a ordenação de vagões foi um desafio significativo. A manipulação direta de endereços de memória, especialmente na gestão de arrays, exigiu uma compreensão aprofundada da organização e operação interna da linguagem.

O processo de implementar a ordenação por meio de trocas de elementos adjacentes, com foco em minimizar o número de trocas, demandou uma atenção cuidadosa à lógica e ao controle de fluxo do programa.

Esta experiência reforçou nossa habilidade de trabalhar com a linguagem e nos deu uma apreciação maior das complexidades envolvidas na programação de baixo nível, destacando a importância de uma boa compreensão dos conceitos de arquitetura e organização de computadores.