	Universidade de São Paulo - USP Instituto de Ciências Matemáticas e de Computação – ICMC Departamento de Sistemas de Computação – SSC SSC0112 Organização de Computadores Digitais I 1º Trabalho – disponibilizado em 11/03/2014

Implementação do algoritmo *merge sort* recursivo em assembly ICMC e MIPS

O algoritmo de ordenação *merge sort* é um algoritmo do tipo "dividir para conquistar", onde um vetor de *n* elementos é recursivamente dividido em dois subvetores, até haver subvetores com tamanho 1 (n=1). Na volta da recursão os subvetores são ordenados e, após a ordenação dos mesmos, é feita a operação de *merge* em que tais subvetores ordenados são consolidados em um único vetor (ordenado). Como citado acima, na implementação recursiva padrão considera-se como caso base da recursão o subvetor de um único elemento. Alternativamente, pode-se considerar como caso base da recursão subvetores com tamanhos não superiores a um valor limite (*threshold*). Quando os subvetores atingem este limite aplica-se outro algoritmo de ordenação ao invés do *merge sort*.

O objetivo deste trabalho é implementar o algoritmo *merge sort* recursivo modificado que utiliza o algoritmo *bubble sort* para subvetores com tamanho menor ou igual a 4. A implementação deve ser feita tanto em *assembly* ICMC quanto em *assembly* MIPS (serão entregues dois códigos fonte, conforme especificado abaixo).

O vetor de inteiros a ser ordenado estará fixo no código fonte e deverá ser alocado na memória do computador quando o processo for carregado no respectivo simulador. O tamanho do vetor deverá ser definido na memória e apontado por um rótulo, o qual deverá ser utilizado para representar o tamanho do vetor durante a definição do mesmo e demais instruções que necessitarem desta informação. O vetor original (não ordenado) deverá ser mantido na memória até a finalização do processo.

Os algoritmos a serem implementados devem considerar vetores de qualquer tamanho entre 1 e 16 elementos. Vetores com mais de 16 posições devem ser detectados e a execução deve ser interrompida. Cada execução do processo ordena uma única vez o vetor desordenado e então finaliza. O algoritmo deve ser genérico e funcionar para qualquer tamanho de vetor, mesmo para vetores maiores (ou menores) que 16 elementos.

Como saída, a função *main* do processo deve imprimir no vídeo o vetor ordenado e o vetor não-ordenado em ordem crescente. Os vetores a serem impressos devem ser identificados no vídeo como "ordenado" e "nao-ordenado", respectivamente. Os elementos desses vetores devem ser separados por uma vírgula.

Os códigos fonte devem conter uma função *main*, a qual chama uma função *merge sort*, passando como argumentos de entrada o endereço da primeira posição do vetor, a posição inicial do vetor a ser considerada e a posição final do vetor. Ao detectar um subvetor de tamanho menor ou igual a 4, a função *bubble sort* deve ser utilizada para ordenação.

Quando a execução for correta, a função *merge sort* deve retornar o endereço de um novo vetor, este contendo os elementos ordenados. Quando a execução da função *merge sort* detectar algum erro (quantidade errada de elementos a ordenar, por exemplo), retornará 0 (zero). A função *merge sort* não poderá imprimir nada na tela. A função *main* será a responsável por imprimir os resultados finais do processo, conforme já detalhado.

Os vetores contendo os dados (ordenados e não ordenados) **não** podem estar na pilha. Devem estar em memória como dados estáticos. O algoritmo do *merge sort* recursivo não está detalhado nesta especificação. Este algoritmo deve ser pesquisado na literatura disponível que aborda estrutura de dados e métodos de ordenação. A detecção de plágio resultará em nota zero para os grupos envolvidos.

Referência para conceito e implementação:

- http://en.wikipedia.org/wiki/Merge_sort
- http://rosettacode.org/wiki/Sorting_algorithms/Merge_sort
- http://stackoverflow.com/questions/3542588/recursive-merge-sort-in-mips-using-stack

A correção levará em conta a execução correta dos algoritmos e a qualidade dos códigos fonte feitos (indentação, comentários corretos, nome significativos para rótulos, espaços entre porções do código, modularidade, entre outros).

Para verificar a execução correta dos algoritmos é necessário que a entrada e a saída sigam, ambas, um padrão rígido: (a) para a entrada será considerado um vetor fixo de inteiros positivos na memória do computador; (b) a saída deve ser impressa no vídeo pela função ***main***, seguindo o já determinado; (c) o programa encerra após a execução da operação solicitada.

O código desenvolvido deverá executar tanto no Linux quanto no Windows. Serão utilizados os simuladores PCSpim (ou QtSpim) para códigos em *assembly* MIPS e simulador ICMC para códigos em *assembly* ICMC.

Usem como casos de teste vetores de tamanho variado (entre 1 e 16 elementos), vetores totalmente ordenados e vetores totalmente/parcialmente desordenados. Usem também vetores de tamanho superior a 16 elementos. Nos casos de entradas inválidas, o seu algoritmo deve fornecer uma saída válida, indicando o erro e então finalizando.

Este trabalho deverá ser feito em grupo, o qual já foi determinado no início do semestre letivo. Envie apenas um arquivo por grupo contendo os códigos fonte compactados (padrão *zip*). Este arquivo ***.zip*** deve ter o nome: T***Y***-G***XX-nnnn.zip*** (onde ***Y*** indica a Turma (1, 2 ou 3), ***XX*** indica o número do grupo e ***nnnn*** indica o nome de um dos integrantes do grupo). Forneça, obrigatoriamente, como comentário no corpo do arquivo submetido o número da turma, o número do grupo e o nome de **todos** os integrantes do grupo que efetivamente participaram do desenvolvimento. A data máxima de entrega deste Trabalho Prático 1 já foi fixada com as turmas em sala de aula. A entrega será via Moodle da USP.

Você deve seguir rigorosamente toda a especificação deste texto a fim de validar a sua nota. Essa validação será feita por um fator de multiplicação à nota final do Trabalho: 0 (zero) por não seguir a especificação ou 1 (um) por seguir. Quaisquer dúvidas/erros/características omissas a esta especificação deverão ser reportadas ao professor e alunos PAE para orientação de como proceder na construção e entrega do algoritmo.