Equipe: Emanuel Ricardo Yano e Kléber Oliveira da Silva

Inicialmente, na parte .data, definimos o tamanho do nosso vetor, logo abaixo, os números do nosso vetor, iniciando assim, a parte .text, que vai cuidar das instruções a serem executadas. Na main, salvamos o endereço do vetor em um registrador e o valor do tamanho em outro, para usarmos posteriormente. Salva o índice inicial (0), índice final (2, pois já foi decrementado acima, garantindo a exatidão do vetor) e endereço do vetor nas variáveis de argumentos, na próxima instrução já inicia de fato nossa recursão de ordenação. Ao entrar no método ordenaRapido, criamos as 4 pilhas e salvamos todos as nossas variáveis e nosso registrador de retorno. Salvamos o índice inicial de retorno, verificamos se ele é menor que nosso índice final, caso verdadeiro, chama o método separa para fazer as quebras em sub vetores, depois chama-se recursivamente até que tudo esteja corretamente ordenado e caia no encerrar, o qual vai restaurar os valores da pilha e voltar ao $ra inicial. No método separa, inicializamos os registradores na pilha, configuramos os que iremos usar, avançamos 1 endereço a frente do índice e carregamos o novo endereço, para realizar o carregamento do elemento pivô, que definimos como o ultimo elemento do nosso vetor. Decrementamos o índice inicial, criamos o novo índice j, para apoio, e decrementamos 1 do índice final, visto que a ultima posição agora é a anterior ao nosso pivô, partimos assim para a repetição. Nessa parte, fazemos o processo de verificação do índice j, se for menor que nosso índice final, encerra o laço. Após isso calculamos o endereço do próximo valor do vetor para pertencer ao índice j, salvamos ele em um registrador. Próximo passo e comparar o pivô com o nosso vetor no índice j, sendo menor nossas instruções continuam, caso maior, encerra a condição, incrementa 1 em j e volta para o laço. Continuando, nosso índice inicial, terá 1 incrementado, para que no momento da troca, seja pego o vetor de índice i e j, adiciono nos registradores de argumentos ambos os valores e chamo o método troca, após a execução de troca ele retorna para a próximo instrução após chama-la, sendo assim incrementa o j e retorna para o laço, somente se encerrará o laço assim quando nosso índice j, for maior que nosso índice final. O método troca inicialmente configuramos a pilha com nossos registradores sendo salvos na memória, calculamos o endereço do vetor no índice i e j, após isso fazemos a devida troca direto na memória, restauramos a pilha e voltamos para o endereço do $ra.

Algortimo

O algoritmo implementado é o algoritmo Quicksort, que é um algoritmo de ordenação eficiente baseado em separa o vetor em partes, dividir e conquistar.

Funções:

- OrdenaRapido

  A função ordenaRapido é a função principal que implementa o algoritmo Quicksort de forma recursiva. Ela verifica se há elementos suficientes para ordenar (comparando os índices inicial e final). Usa a função separa para escolher um pivô e particionar o vetor em dois subvetores ao redor do pivô. Recursivamente chama a si mesma para ordenar os subvetores esquerdo e direito. Ao final, realiza a limpeza da pilha e retorna.

- Separa

A função separa seleciona um pivô e rearranja o vetor de forma que todos os elementos menores que o pivô fique à esquerda dele e todos os elementos maiores fiquem à direita. Usa índices a1 (inicial) e a2 (final) para delimitar o subvetor atual. Usa a função repeticao para iterar sobre os elementos e realizar a troca quando necessário.

- Repeticao

  Repeticao é um laço que percorre o subvetor, comparando elementos com o pivô e realizando trocas conforme necessário até que todos os elementos sejam processados. Usa índices $t4 (j) para percorrer o vetor da esquerda para a direita.

- Troca

  Função auxiliar para trocar dois elementos de posição no vetor.

- main:

  Inicializa o vetor e chama a função ordenaRapido para ordená-lo.

Parâmetros:

Os parâmetros são passados através dos registradores $a0, $a1 e $a2.$a0 é usado para o endereço do vetor. $a1 e $a2 são usados para os índices inicial e final do vetor, respectivamente.

Variáveis:

Variáveis locais são armazenadas na pilha usando $sp (ponteiro de pilha). Registros como $t0, $t1, $t2, $t3, $t4, $t5, $s0, $s1, $s2, $s3, $s4 são usados para armazenar valores temporários, endereços de memória e índices durante as operações.

As dificuldades encontradas no momento do desenvolvimento do algoritmo, se deu mais na parte de transcrever o QuickSort da forma de programação em alto nível para linguagem de instrução. Inicialmente, não estávamos conseguindo compreender direito como faríamos as recursões, os envios das várias, os loops, etc., porém após analisar vídeos de implementação de recursividade em assembly, foi verificado a forma com que faríamos esse processo e implementado no nosso algoritmo. A implementação do para pular para o próximo endereço, no inicio não conseguimos identificar exatamente como realizar esse procedimento, acreditávamos que era apenas incrementar mais 4 ao endereço, no entanto, analisando alguns vídeos e sites, verificamos que como cada palavra tem 4 bytes, era necessário multiplicar 4 vezes para obter o endereço do próximo valor, sendo assim colocamos no codigo.