Assembly - Quick Sort

O seguinte trabalho foi feito pelos alunos Pedro Augusto e Marcus Felipe, o algoritmo escolhido para escrevermos em assembly foi o algoritmo de ordenação quick sort.

Descrição da Implementação:
No .section .data foram definidos o vetor global V e o seu tamanho N
.section .data
V: .quad 2, 8, 7, 5, 1, 3, 4, 6, 9
N: .quad 9
str1: .string "%d\n"
.section .text
A passagem de parâmetros para os procedimentos segue o padrão do LibC. Sendo assim os parâmetros são passados por registradores, na qual evita acessos a memória.
partition:
rdi - Low
rsi - High
rdx - Endereco do vetor
Conservando a base do registro de ativação da função chamadora.
pushq %rbp
Conservando o valor contido em %rbx
pushq %rbx
movq %rsp, %rbp
Acessando o endereço da posição indicada pelo valor contido em %rsi no vetor.
%rbx - pivot
movq %rsi, %rbx
imulq $8, %rbx
addq %rdx, %rbx
%rax - i
movq %rdi, %rax
subq $1, %rax%rcx será usado como a variável “j” controladora do laço FOR
%rcx – j
movq %rdi, %rcx
LOOP:
cmpq %rcx, %rsi
je END_LOOP
Acessando o endereço da posição indicada pelo valor contido em %rcx no vetor.
%r8 – arr[j]
movq %rcx, %r8

imulq $8, %r8
addq %rdx, %r8
Acessando o endereço da posição indicada pelo valor contido em %rbx no vetor.
%r9 – arr[pivo]
movq (%rbx), %r9
cmpq (%r8), %r9
jl END_IF_2
incq %rax
Swap dos registradores %r9 e %r8
movq %rax, %r9
imulq $8, %r9
addq %rdx, %r9
movq (%r9), %r10
movq (%r8), %r11
movq %r11, (%r9)
movq %r10, (%r8)
END_IF_2:
incq %rcx
jmp LOOP
END_LOOP:
Acessando o endereço da posição indicada pelo valor contido em %rax + 1 no vetor. %r9 – arr[i + 1]
movq %rax, %r9
incq %r9
imulq $8, %r9
addq %rdx, %r9Acessando o endereço da posição indicada pelo valor contido em %rsi no vetor.
%r8– arr[high]
movq %rsi, %r8
imulq $8, %r8
addq %rdx, %r8
Swap dos regsitradores %r9 e %r8
movq
movq
movq
movq
(%r9), %r10
(%r8), %r11
%r11, (%r9)
%r10, (%r8)
Reestabelecendo o valor de %rbx

popq %rbx
Reestabelecendo o registro de ativação da função chamadora
popq %rbp
inc %rax
ret
quick_sort:
rdi - Low
rsi - High
rdx - Endereco do vetor
Conservando a base do registro de ativação da função chamadora.
pushq %rbp
movq %rsp, %rbp
Espaço alocado para receber o valor de retorno do procedimento partition. Poderiamos ter usado um registrador para evitar acesso a memória, porém no enunciado do trabalho pede para criar variáveis locais, então decidimos criar essa variável.
subq $8, %rsp
cmpq %rdi, %rsi
jle END_IF
Convervando os valores dos registradores %rdi, %rsi, %rdx
pushq %rdi
pushq %rsi
pushq %rdx
call partitionVariável local recebe o valor de retorno da procedimento partition
movq %rax, -8(%rbp)
Recuperando os valores dos registradores %rdi, %rsi, %rdx
popq %rdx
popq %rsi
popq %rdi
Convervando os valores dos registradores %rdi, %rsi, %rdx
pushq %rdi
pushq %rsi
pushq %rdx
movq -8(%rbp), %rsi
subq $1, %rsi
call quick_sort
Recuperando os valores dos registradores %rdi, %rsi, %rdx
popq %rdx
popq %rsi
popq %rdi
Convervando os valores dos registradores %rdi, %rsi, %rdx
pushq %rdi

```
pushq %rsi
pushq %rdx
movq -8(%rbp), %rdi
incq %rdi
call quick_sort
Recuperando os valores dos registradores %rdi, %rsi, %rdx
popq %rdx
popq %rsi
popq %rdi
END_IF:
Desalocando o espaço na pilha reservado para a variavel local
add $8, %rsp
Reestabelecendo o registro de ativação da função chamadora
popq %rbp
retvector_printing:
pushq %rbp
movq %rsp, %rbp
pushq %rbx
pushq %r12
movq $0, %rbx
movq %rdi, %r12
for:
cmpq %rsi, %rbx
jge end_for
pushq %rsi
pushq %rdi
movq (%r12), %rsi
movq $str1, %rdi
call printf
popq %rdi
popq %rsi
addq $8, %r12
incq %rbx
jmp for
end_for:
popq %r12
popq %rbx
popq %rbp
ret
.globl main
main:
movq %rsp, %rbp
```

Espaço alocado para receber o valor de retorno da função print
subq $8, %rsp
movq $0, %rdi
movq N, %rsi
subq $1, %rsi
movq $V, %rdx
call quick_sort
movq $V, %rdi
movq N, %rsi
call vector_printing
addq $32, %rsp
movq $60, %rax
syscall

fonte: https://www.geeksforgeeks.org/quick-sort/