Universidade de Brasília
Departamento de Ciência da Computação

Disciplina: CIC 116394 – Organização e Arquitetura de Computadores – Turma A 2020/1
Prof. Marcus Vinicius Lamar
Alunos: Brenno Pereira Cordeiro - 19/0127465
João Victor Bohrer Munhoz - 16/0071101
Mariana Alencar do Vale - 16/0014522

Laboratório 1
- Assembly RISC V -

1) Simulador/Montador Rars

1.1)

Dado o vetor V[30]={9,2,5,1,8,2,4,3,6,7,10,2,32,54,2,12,6,3,1,78,54,23,1,54,2,65,3,6,55,31}, o ordenamos em ordem crescente com o programa de ordenamento *sort.s*, cujo output gerou 3.406 instruções, sendo elas 994 do tipo R, 1296 do tipo I, 312 do tipo S, 431 do tipo B, 1 do tipo U e 372 do tipo J.

O arquivo executável possui 172 bytes, sendo elas 43 instruções em Assembly com 4 bytes cada uma. A memória de dados usada possui 120 bytes, 4 bytes para cada posição do vetor.

1.2)

a) Para os vetores de entrada de **n elementos já ordenados** temos a seguinte equação do tempo de execução em função de n:

$$To(n) = \frac{n}{5}$$

Para os vetores de entrada de **n elementos ordenados inversamente** temos a seguinte equação do tempo de execução em função de n:

$$Ti(n) = 0.18n^2 - 0.08n + 0,1$$

PS: Para a contagem de instruções necessária para calcularmos as respectivas equações, não incluímos o procedimento SHOW, que printa os valores na tela, e nem as 2 instruções do ecall de saída, pois elas não fazem parte da função de ordenamento em si.

b) Para n = {10,20,30,40,50,60,70,80,90,100}, plotamos as duas curvas das funções $To(n)$ e $Ti(n)$ em um mesmo gráfico n X t

Podemos atestar logo de cara que a função de $Ti(n)$ cresce muito mais rapidamente do que a função $To(n)$. Isso se deve ao fato de que enquanto $To(n)$ cresce linearmente, $Ti(n)$ cresce de forma quadrática, como explicitado nas funções definidas na letra a.

1.3) --

2) Compilador cruzado GCC

2.2) Para que fosse possível rodar o código no RARS, foi preciso adicionar os indicadores de seção *.text* e *.data*, uma chamada para o procedimento MAIN no início do arquivo, adicionar *ecalls* para o fim da execução e substituir as chamadas para *printf* e *putchar* por *ecalls* apropriados. O arquivo editado está na pasta códigos do .rar, chamado *sortc.s.*

```
@@ -1,3 +1,4 @@
+.data
 v:
         .word   9

@@ -31,6 +32,11 @@ v:
 .LC0:
         .string "%d\t"
+
+.text
+       call main
+       li a7 10
+       ecall
 show:
         addi    sp,sp,-48

@@ -47,9 +53,12 @@ show:
         lw      a5,0(a5)
         mv      a1,a5
-        lui     a5,%hi(.LC0)
-        addi    a0,a5,%lo(.LC0)
-        call    printf
+        mv a0 a1
+        li a7 1
+        ecall
+        li a0 9
+        li a7 11
+        ecall
         lw      a5,-20(s0)

@@ -58,7 +67,8 @@ show:
         lw      a5,-40(s0)
         blt     a4,a5,.L3
         li      a0,10
-        call    putchar
+        li a7 11
+        ecall
         lw      s0,40(sp)
```

2.3)

Otimização	Tamanho em bytes (.text)	N° instruções
-O0	532B	9789
-O1	392B	3905
-O2	316B	2490
-O3	328B	2420
-Os	356B	4101
sort.s	172B	3406

O aumento do nível de otimização foi capaz de diminuir constantemente o número de instruções executadas como esperado, diminuindo também o tamanho do arquivo gerado. O nível de otimização -Os que deveria gerar o menor executável possível, foi pior que o nível de otimização -O2 nessa tarefa. O nosso arquivo foi o que gerou o menor executável, entretanto executado mais instruções que os níveis mais otimizados de compilação.

3) Polígonos Regulares Inscritos em uma Circunferência

3.1) Código presente na pasta *Códigos* dentro do .rar

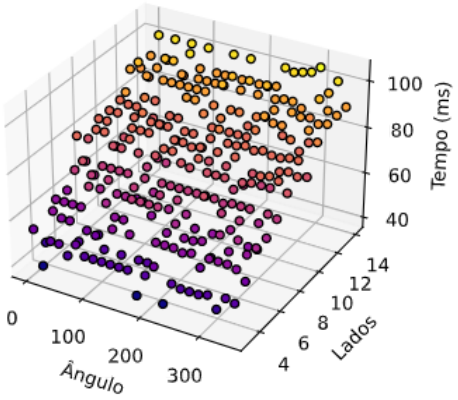
3.2) Código presente na pasta *Códigos* dentro do .rar

3.3) Código presente na pasta *Códigos* dentro do .rar

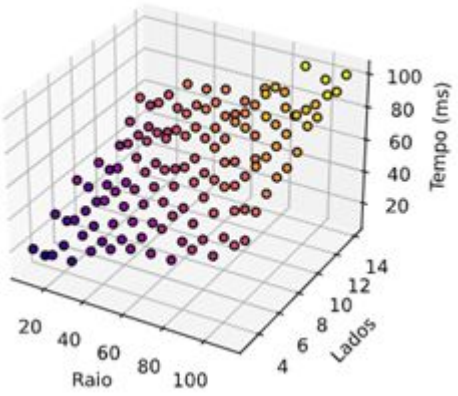
3.4) https://youtu.be/scjX3ZRwb0Y

3.5)

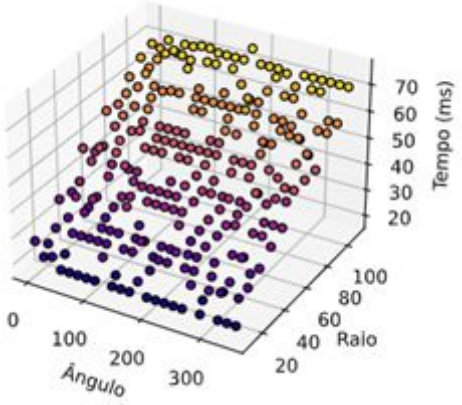
I) (100, [0:15:360], [3:1:15])



II) ([10:10:120], 90, [3:1:15]



III)([10:10:120], [0:15:360], 5)



O parâmetro Ângulo mantém o tempo de execução dos procedimentos quase constante. São os parâmetros raio e número de lados que influenciam mais no tempo de execução. Baseado em uma regressão linear desses parâmetros sobre os dados que coletamos, o raio tem um coeficiente angular de 0.45 em relação ao tempo de execução, enquanto o número de lados tem coeficiente de 3.75. Podemos concluir que o parâmetro que indica o número de lados influencia mais no tempo de execução do programa, e por consequência no seu desempenho.

3.6) Obtemos 10189 instruções executadas em 70 milissegundos. Com o CPI = 1, a frequência do processador RISC-V equivalente é f = 10189/0.070 = 145557 Hz ou 145KHz**.**