

Grupo 12 - OAC - 2021-2

Eduardo Ferreira Marques Cavalcante - 202006368

Gabriel Mendes Ciriatico Guimarães - 202033202

Gustavo Lopes Dezan - 202033463

2.1) Considere a execução deste algoritmo em um processador RISC-V com frequência de clock de 50MHz que necessita 1 ciclo de clock para a execução de cada instrução (CPI=1). Para os vetores de entrada de n elementos já ordenados $Vo[n] = \{1, 2, 3, 4, \dots, n\}$ e ordenados inversamente $Vi[n] = \{n, n-1, n-2, \dots, 2, 1\}$:

a) Para o procedimento sort, escreva as equações dos tempos de execução em função de n , $to(n)$ e $ti(n)$, (1.0)

b) Para $n=\{10,20,30,40,50,60,70,80,90,100\}$, plote (em escala!) as duas curvas em um mesmo gráfico $n \times t$. Comente os resultados obtidos

a) $to(n) = lo * CPI * T \Leftrightarrow to(n) = (10*n + 13) * (2*10^{-8})$

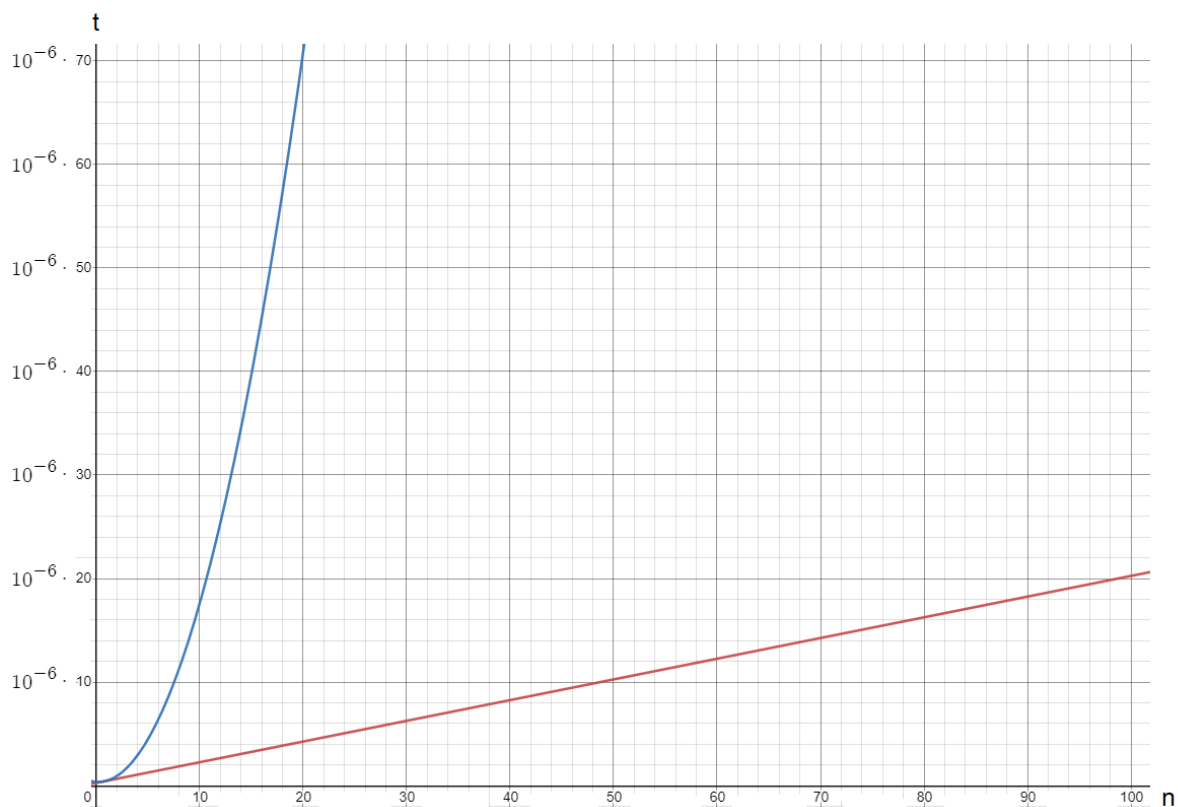
$ti(n) = li * CPI * T \Leftrightarrow ti(n) = (9*n^2 - 4*n + 18) * (2*10^{-8})$

$T = 2.10^{-8}$ e $CPI = 1$ para ambas as equações;

$lo = (10*n + 13)$

$li = (9*n^2 - 4*n + 18)$

b) Aqui vemos que o tempo para execução do método sort para um vetor inversamente ordenado (curva azul) cresce bem rápido que a curva do tempo de execução do mesmo programa para um vetor já ordenado (curva vermelha). Com n sendo o tamanho do vetor e t o tempo de execução em segundos.



2.2) Dado o programa `sortc.c`, compile-o com a diretiva `-O0` e obtenha o arquivo `sortc.s`. Indique as modificações necessárias no código Assembly gerado para que possa ser executado corretamente no Rars:

Foram necessárias 5 mudanças no script “`sortc.s`” gerado pelo compilador cruzado:

1. A separação entre `.data` e `.text` foi inserida, separando as linhas de código Assembly para serem executadas dos dados salvos no início do programa;
2. Os rótulos dos procedimentos foram alterados. No arquivo original, os rótulos se aproximam dos rótulos em C (“`show(int*, int)`”, por exemplo), com parênteses indicando o tipo do dado. Os parênteses foram eliminados para que fosse um rótulo entendível para o RARS (“`show`”, por exemplo);
3. Procedimento “`show`” foi alterado, com a versão do código original sendo apagada e substituída pelo procedimento implementado no arquivo dado “`sort.s`”;
4. O ordenamento dos procedimentos foi alterado. No código original, o procedimento “`main`” era o último do script, mas no RARS deve vir na primeira posição;
5. Foi necessário colocar no fim do procedimento MAIN o `ecall` para terminar o programa.

2.3) Compile o programa `sortc_mod.c` e, com a ajuda do Rars, monte uma tabela comparativa com o número total de instruções executadas pelo programa todo, e o tamanho em bytes dos códigos em linguagem de máquina gerados para cada diretiva de otimização da compilação {`-O0`, `-O1`, `-O2`, `-O3`, `-Os`}. Compare ainda com os resultados obtidos no item 1.1) com o programa `sort.s` que foi implementado diretamente em Assembly. Analise os resultados obtidos usando o mesmo vetor de entrada.

O script “`sortc_mod.s`”, resultante da compilação cruzada de “`sortc_mod.c`”, pode ter seu número de instruções e tamanho em bytes analisado na tabela abaixo:

| Diretiva de otimização | Número de instruções | Tamanho (em bytes) |
|------------------------|----------------------|--------------------|
| -O0 | 9940 | 524 |
| -O1 | 3886 | 372 |
| -O2 | 2174 | 280 |
| -O3 | 2174 | 272 |
| -Os | 4097 | 348 |

Veja que há uma diminuição progressiva no número de instruções e no tamanho do código entre `-O0` e `-O3`, enquanto com a diretiva `-Os` chega-se no segundo maior número de instruções e no terceiro maior código em número de bytes.

É possível comparar ainda com os dados obtidos ao rodar o programa “`sort.s`” (com o mesmo vetor de valores). Veja na tabela abaixo a comparação:


| Programa | Diretiva de | Número de | Tamanho (em |
|----------|-------------|-----------|-------------|
|----------|-------------|-----------|-------------|

| | otimização | instruções | bytes) |
|-------------|------------|------------|--------|
| sort.s | | 4677 | 276 |
| sortc_mod.s | -O0 | 9940 | 524 |
| sortc_mod.s | -O1 | 3886 | 372 |
| sortc_mod.s | -O2 | 2174 | 280 |
| sortc_mod.s | -O3 | 2174 | 272 |
| sortc_mod.s | -Os | 4097 | 348 |

Observe que não houve mudança no número de instruções de “sortc_mod.s”, bem como, obviamente, no tamanho do código (que não foi alterado), mesmo que tenha acontecido uma mudança no vetor ordenado.

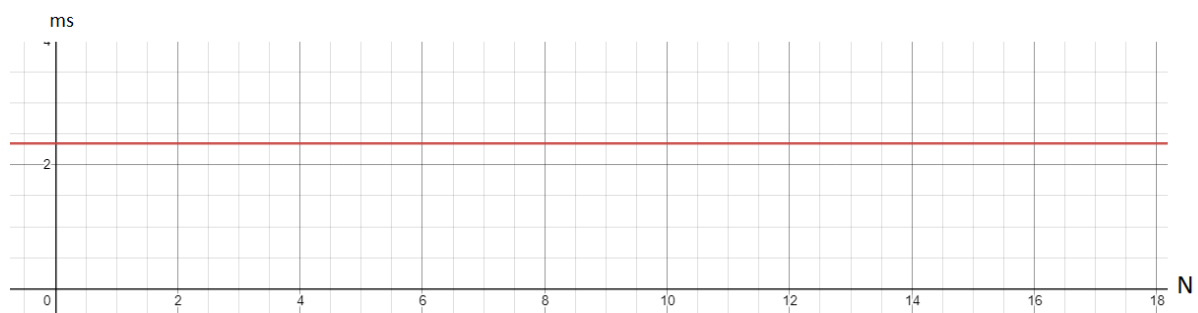
A única mudança que houve no vetor de “sortc_mod.s” foi a remoção dos valores 4 e -4, o que não teve impacto no número de instruções executadas no código. Observe também que os dois scripts executam a mesma tarefa, embora com instruções diferentes. A otimização -O3 de “sortc_mod.s” permite que esse processo de ordenamento e exibição do vetor seja feita com o menor número de instruções e tamanho de código - menor ainda que o programa “sort.s”, que não é resultado de compilação cruzada.

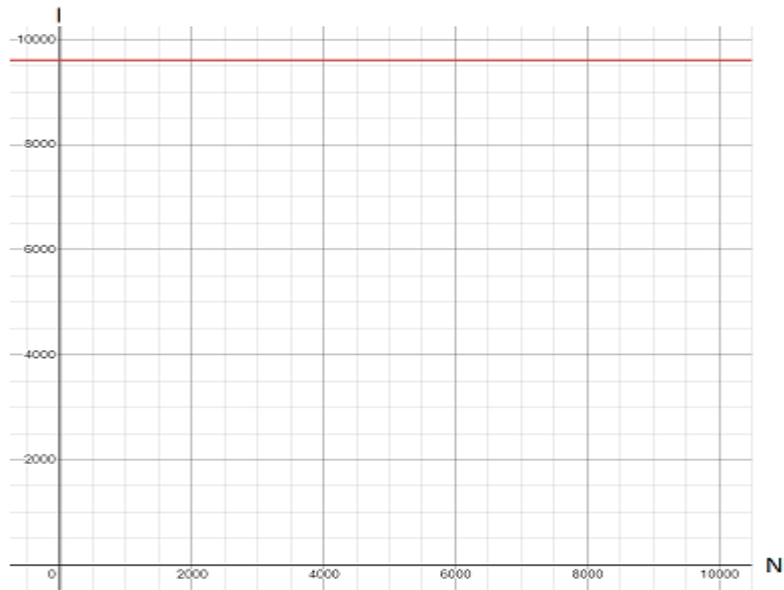
3.5) Filme vc jogando até o máximo N que seu grupo conseguir. (Não vale usar cheat!!!):

 3) Senha (Mastermind) em assembly RISC-V

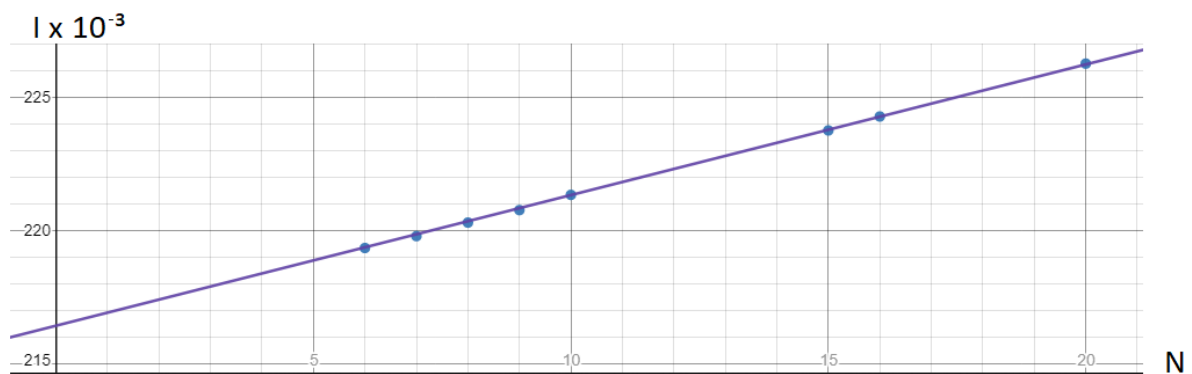
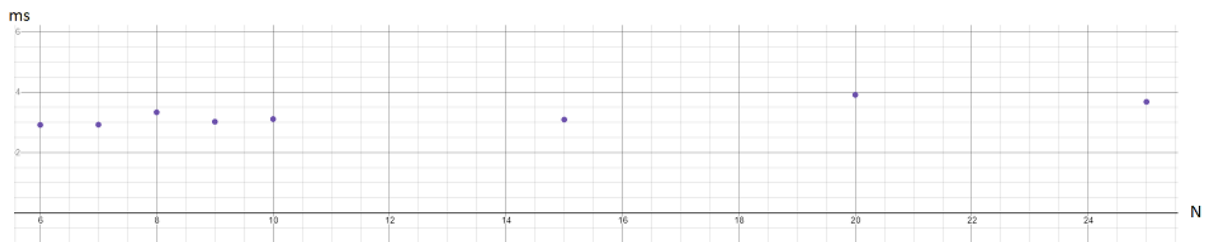
3.6) Faça os gráficos $N \times \text{texec}$ e $N \times I$, onde texec é o tempo de execução e I o número de instruções requeridas pelo seu algoritmo para a análise do pior caso da tentativa do jogador. Sabendo que o Rars simula uma CPU RISC-V com CPI=1, qual a frequência de clock da CPU?

Considerando apenas a análise após a tentativa do jogador:





Isso ocorre por conta da análise da tentativa do jogador não ser relacionada com a dificuldade N. Ela sempre faz 4 iterações pelo vetor da senha. Já considerando o cálculo feito para todo o programa (desconsiderando apenas o tempo de espera pela entrada do input), com isso analisando o tempo levado para gerar a lista de cores, gerar a senha e renderizar a tela.



O cálculo do tempo sofre influência de diversos fatores, por conta disso não foi possível encontrar uma fórmula que se aproxime dos valores obtidos. Considerando um caso base com relação ao início, com Texec = 3506 e número de instruções (I) = 219360, encontra-se uma frequência de clock para a CPU de 0,061130 MHz.