Laboratório 1

Matheus Olivera de Almeida Marques da Cruz* Pedro Henrique dos Santos † Nicole de Oliveira Sena ‡
João Pedro Carvalho de Oliveira Rodrigues §

20 de dezembro de 2023

RESUMO

Um compilador é o responsável por transformar um código de alto nível em assembly, para posteriormente ser transformado em código de máquina e executado pelo computador. Nesse laboratório fizemos uma análise do tempo de execução e do tamanho dos programas desenvolvidos diretamente em assembly e gerados pelo compilador. Além disso foi feito um jogo em que um canhão atira em uma espaço-nave, a trajetória da bola do canhão é calculada em tempo real considerando parâmetros de física pré-determinados.

Palavras-chave: OAC · Assembly RISC-V · Canhão · Compilador · GCC

1 SIMULADOR/MONTADOR RARS

Para essa seção queremos analisar o programa sort.s. Podemos calcular o tempo de execução para os vetores já ordenados e ordenados inversamente. O tempo de execução é determinado pelo número total de ciclos de clock necessários para concluir a execução do algoritmo para cada vetor.

1.1 VETOR ORDENADO

Nesse caso, o número de iterações do loop será n - 1, já que na primeira iteração o maior elemento é movido para a última posição, e na segunda iteração o segundo maior elemento é movido para a penúltima posição, e assim por diante.

O tempo de execução $t_o(n)$ é:

$$t_o(n) = (n-1) \cdot n$$

1.2 VETOR ORDENADO INVERSAMENTE

Aqui, o vetor está desordenado, o algoritmo vai percorrer o vetor trocando os elementos adjacentes em cada iteração. No pior caso, o número de iterações vai ser o mesmo que em $t_o(n)$.

O número total de comparações e trocas para cada iteração vai ser igual à soma dos primeiros n - 1 números inteiros, que é (n - 1) $\cdot n/2$

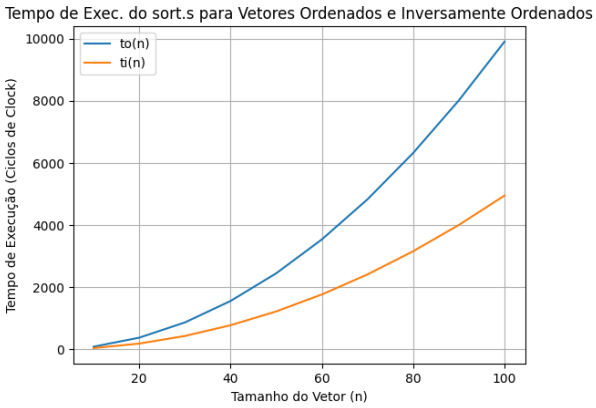
Então, o tempo de execução $t_i(n)$ é:

$$t_i(n) = (n-1) \cdot \frac{n}{2}$$

1.3 GRÁFICO

- À medida que n aumenta, tanto $t_o(n)$ quanto $t_i(n)$ aumentam, e isso é esperado, já que o tempo da execução do algoritmo de ordenação de bolha é diretamente proporcional ao tamanho do vetor.

Figura 1: *Curvas $t_o(n)$ e $t_i(n)$ em escala*



- Quando o vetor já tá ordenado, o tempo de execução é quase o dobro do tempo de execução quando o vetor tá desordenado ($t_i(n)$). Isso é porque quando o vetor tá ordenado, o algoritmo de ordenação de bolha percorre o vetor só uma vez, já no inversamente ordenado, ele faz mais iterações de comparações.

- A escala logarítmica no eixo y é serve pra visualizar melhor as diferenças entre $t_o(n)$ e $t_i(n)$ em um único gráfico. Como as curvas tem formas parecidas com a função quadrática, a escala logarítmica ajuda a gente a perceber as diferenças nos tempos de execução para os valores grandes de n.

2 COMPILADOR CRUZADO GCC

Nessa seção utilizaremos o GCC para compilar e analisar o código assembly resultante.

2.1 COMPILAÇÃO

Para rodarmos um programa compilado pelo GCC precisamos fazer algumas modificações:

- O GCC não faz uma separação de dados e programa, como fazemos no RARS. Então é necessário separar os dados com as diretivas *.text* e *.data.*

- Substituir quaisquer chamadas de sistema como o printf e outros, pelas syscalls disponíveis no rars.

- Mover o procedimento principal (main) para o início da memória de programa, garantindo que seja primeira a ser executada.

O resultado do procedimento acima pode ser encontrado no arquivo sortc.s, o programa faz a ordenação do vetor corretamente no rars. Com a ressalva de que o compilador salva o registrador ra na

*211055343@aluno.unb.br
†200026127@aluno.unb.br
‡190114860@aluno.unb.br
§221017032@aluno.unb.br

pilha para voltar a executar o programa anterior. Como o no RARS não tem nenhum programa rodando antes, ele para a execução com um erro ao tentarmos voltar para a execução do ra.

2.2 Comparação

Faremos nessa etapa, uma comparação dos programas gerados pelo compilador usando as diretivas de compilação definidas. Sendo elas: -O0, -O1, -O2, -O3, -Os.

Para calcular o tamanho em bytes do programa, basta seguir a formula:

$$(\text{último\_endereço\_da\_memória} \times 4) + 4 \quad (1)$$

Somamos 4 no final para também considerar a última instrução no calculo.

Executando o mesmo vetor de 32 itens, fizemos uma tabela comparando o tamanho do programa em bytes e o número de instruções executadas para ordenar o vetor. Os resultados estão na seguinte tabela:

Flag	Tamanho (bytes)	N Instruções
-O0	520	12508
-O1	364	4877
-O2	264	2627
-O3	264	2627
-Os	154	5135
Manual	276	5806

Tabela 1: *Comparação das diferentes diretivas de compilação.*

Analisando brevemente os resultados obtidos, observamos que exceto pela diretiva padrão (-O0), que é a menos otimizada, o código feito diretamente em assembly resultou em mais instruções executadas, isso reforça a ideia de que os compiladores são mais interessantes para essa tarefa. Curiosamente as diretivas -O2 e -O3 resultaram no mesmo código assembly, por isso ambos obtiveram os mesmos resultados na tabela.

3 Tiro de Canhão na Espaçonave

Os testes referentes ao tiro acertando tanto os limites da tela quanto a nave se encontram no link : https://www.youtube.com/watch?v=ZgJZTFNnxP0