Laboratório 1
RISC-V

Thiago V Machado, 16/0146682
Gabriel B Vieira, 16/0120811
Lucas Mafra Chagas, 12/0126443

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CiC 116394 - Organização e Arquitetura de Computadores - Turma A

1. Simulador/Montador Rars

1.1. Número de instruções por tipo

Estatísticas somente da função sort (implementação de bubble sort) em Assembly RISC-V. Retiradas com auxílio das ferramentas *Instruction Counter* e *Instruction Statistics* do MARS.

Baseado em um vetor não ordenado de tamanho n = 10.

- 803 instructions : R - 23%; I - 46%; S - 17%; U - 12
- 803 instructions : ALU - 79% ; Jump - 2% ; Branch - 3% ; Memory - 0% ; Other - 16%.
- 1147 bytes de memória.

1.2. Desempenho dos processadores RISC-V e MIPS

Sabendo que o desempenho pode ser definido pela equação

$$Texec = IxCPIxT$$

Onde o Tempo de Execução da CPU para um programa **(Texec)** é a multiplicação da Contagem de Instruções **(I)**, Ciclos de clock por instrução **(CPI)** e o tempo do ciclo de clock **(T)**, podemos definir que para melhorar o desempenho do RISC-V teríamos que aumentar a frequência de clock, melhorar a organização do processador que diminuem a CPI, algumas melhorias no compilador que diminui a CPI e a contagem de instruções.

Considerando que o algoritmo contém 803 instruções, rodando em um processador i7 com frequência de 4GHz, com tempo de ciclo de clock igual a 250ps. Considerando também que a CPI tenha uma arquitetura multiciclo, onde é calculada por meio da formula $(0,79 * 4) + (0,05 * 3) + (0,16 * 5) = 4,11$, tendo a ALU realizadando 4 ciclos de clock, o Branch e Jump 3 ciclos de clock e os outros 5 ciclos de clock. Fazendo a equação supracitada, temos que o Tempo de Execução (texec) é igual a $825082,5$

1.3. Execução Bubble Sort em um processador RISC-V

Ao executar o algoritmo de Bubble Sort, foi possível obter o seguinte gráfico que possui o melhor e pior caso.

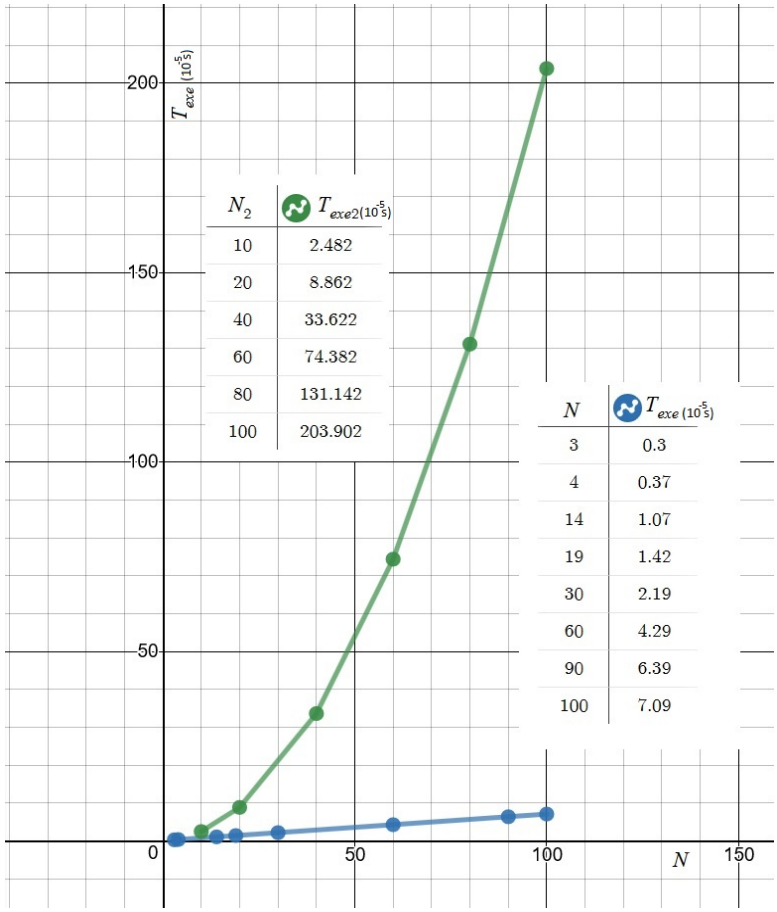

Figure 1. Gráfico com melhor caso e pior caso para o código de Bubble Sort

Observando o gráfico da figura 1, podemos perceber que o melhor caso(azul) possui tempo de execução linear e o pior caso(verde) possui tempo de execução limitado por O(n^2), sendo representado por uma parábola.

2. Compilador GCC

2.1. Desempenho entre os códigos sortc.c e sort2c.c

Comparando o desempenho dos códigos Assembly gerado pelo cross compiler, temos que o sortc2 apresenta um desempenho melhor, visto que realiza menos laços no algoritmo, apresentando menos instruções.

2.2. Modificações do código

Para funcionar no Rars, foram necessárias algumas modificações no código. Para o código de sortc.s, foi necessário alterar a ordem do algoritmo. Para o código sortc2.s, foi necessário alterar a ordem do algoritmo, além de retirar a função LC0 do .text e colocar no .data. Foi necessário alterar os `jr $ra` para `li a7, 1 ecall` e os calls de print como `li a7, 4 la a0, .LC0 ecall` e `li a6, 4 la a0, .LC0 ecall` Além dessas modificações, foi necessário rodar o código para processador 32bits, para não ter que fazer modificações das instruções.

2.3. Comparação do código gerado com o código sort.s

3. Cálculo das raízes da equação de segundo grau

a) [1, 0, -9.86960440]	
Raiz 1	R(1)=3.1415925
Raiz 2	R(2)=-3.1415925
Tempo de Execução Inteiros	$10^{-6} \times 17 \times 10 = 170 \times 10^{-6}$s
Tempo de Execução Ponto F.	$10^{-6} \times 4 \times 1 = 4 \times 10^{-6}$s
Tempo de Execução Total	0.174 ms

b) [1, 0, 0]	
Raiz 1	R(1)=0.0
Raiz 2	R(2)=0.0
Tempo de Execução Inteiros	$10^{-6} \times 17 \times 10 = 170 \times 10^{-6}$s
Tempo de Execução Ponto F.	$10^{-6} \times 4 \times 1 = 4 \times 10^{-6}$s
Tempo de Execução Total	0.174 ms

c) [1, 99, 2459]	
Raiz 1	R(1)=-49.5 + 2.95804 i
Raiz 2	R(2)=-49.5 - 2.95804 i
Tempo de Execução Inteiros	$10^{-6} \times 15 \times 10 = 150 \times 10^{-6}$s
Tempo de Execução Ponto F.	$10^{-6} \times 4 \times 1 = 4 \times 10^{-6}$s
Tempo de Execução Total	0.154 ms

d) [1, -2468, 33762440]	
Raiz 1	R(1)=1234.0 + 5678.0 i
Raiz 2	R(2)=1234.0 - 5678.0 i
Tempo de Execução Inteiros	$10^{-6} \times 15 \times 10 = 150 \times 10^{-6}$s
Tempo de Execução Ponto F.	$10^{-6} \times 4 \times 1 = 4 \times 10^{-6}$s
Tempo de Execução Total	0.154 ms

e) [0, 10, 100]	
Raiz 1	R(1)=NaN
Raiz 2	R(2)=-Infinity
Tempo de Execução Inteiros	$10^{-6} \times 17 \times 10 = 170 \times 10^{-6}$s
Tempo de Execução Ponto F.	$10^{-6} \times 4 \times 1 = 4 \times 10^{-6}$s
Tempo de Execução Total	0.174 ms

4. Tradução de Programas

4.1. Tradução do assembly MIPS para Assembly RISC-V

Para o funcionamento no RARS do systemv1.s foram necessárias várias modificações pois o arquivo principal estava em **Assembly MIPS** e tivemos de traduzir para **Assembly RISC-V**, a fim de obter os resultados finais.