Universidade de Brasília
Campus Universitário Darcy Ribeiro

Disciplina: Organização e Arquitetura de Computadores
Turma: A
Prof.: Marcus Vinicius Lamar

Laboratório 1
Assembly RISC-V

Abdullah Zaiter - 15/0089392
Daniel Bauchspiess - 15/0078901
Danielle Almeida Lima - 14/0135740
Jose Reinaldo da Cunha - 14/0169148
Lucas dos Santos Schiavini - 14/0150749

Brasília, 30 de abril de 2018.

1) Compilador/Montador Rars

1.1) A quantidade de instruções utilizadas foram:

- Por estatística: 635 ALU, 27 jump, 11 branch, 0 memória e 130 em “outros”;
- Por tipo: 188 do tipo R, 376 do tipo I, 139 do tipo S e 100 do tipo U;

Resultando num total de 803 instruções. O código executável possui 296 bytes.

1.2) Considerando o mesmo clock e a mesma CPI, a comparação entre o desempenho do mesmo algoritmo para RISC-V e MIPS se dará pelo número de instruções (qual tiver menos instruções, completa o código em menor tempo, e, assim, tem o melhor desempenho). O código em MIPS possui um total de 802 instruções, tendo, então, um desempenho maior que o código em RISC-V (de 803 instruções).

É possível aumentar o desempenho do RISC-V otimizando o código (eliminando comandos desnecessários, fazendo alterações que não afetam o algoritmo em si).

Antes da otimização, o fator de desempenho obtido foi de 0.99875 (o RISC-V é 0.99875 vezes mais eficiente que o MIPS).

1.3) O desempenho de um algoritmo pode ser avaliado testando-se o melhor caso e o pior caso, sendo que, para o caso do sort.s, o melhor caso é a organização de um vetor já ordenado, e no pior, o vetor está invertido. O resultado pode ser visto na figura 1:

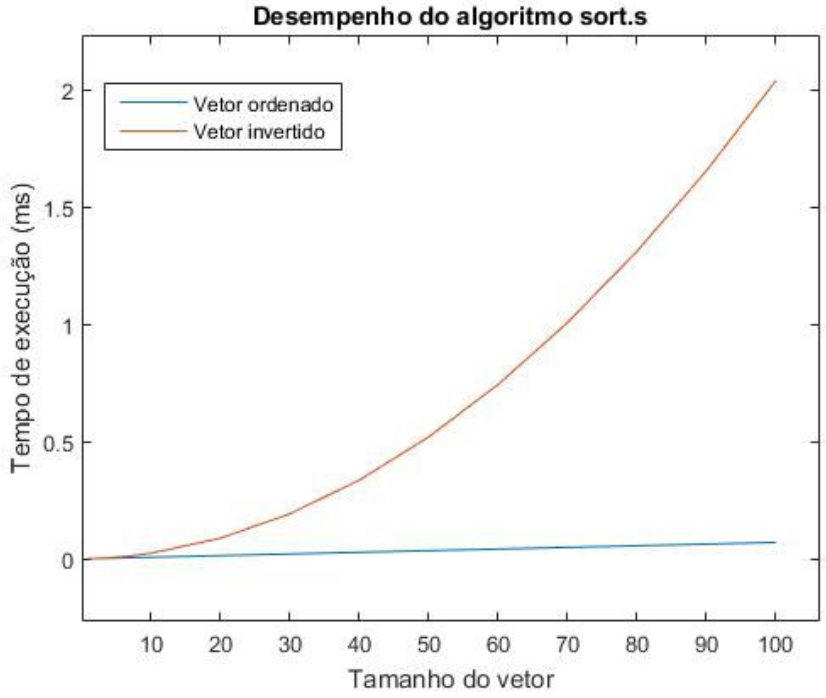


Figura 1: Comparação de tempo de execução do sort.s para um vetor já ordenado e um vetor invertido, para tamanhos variados do vetor

Pelo gráfico, observa-se um bom desempenho (baixo tempo de execução) do código para os vetores mais simples de se ordenar, enquanto que, para vetores relativamente grandes, tal código apresenta um aumento exponencial do tempo de execução.

2) Compilador GCC

2.1) Os códigos foram compilados e analisados.
2.2) Todas as diretivas que apareceram nos dois arquivos compilados de assembly foram comentados e estão no arquivo sortc2_Comentado.s.

Figura 2.1 - Arquivo sortc_O0.s

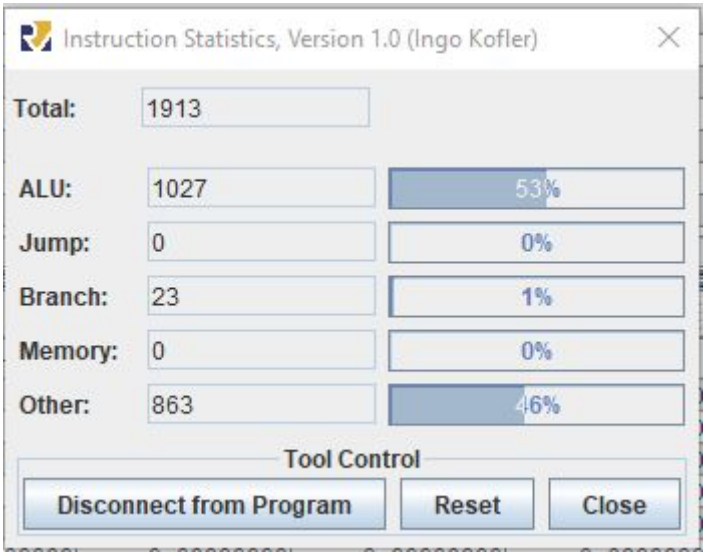


Figura 2.2 - sort2c_O0.s

2.3)Foi feito exatamente o que o professor M.V. Lamar tinha passado no email quando mandamos tirando dúvida

1) Defina no Settings para que use o main como ponto de entrada:

Initialize Program Counter to global main, assim não precisa mudar a ordem do código obtido do gcc.

2) O mais simples é definir as funções printf e putchar para fazer o que se quer, já que são chamadas apenas 1 vez no código.

3) Comentar apenas os .type e os .align que estiverem no .text

Para as diretivas de compilação -o1, -o2, -o3 e -os, o compilador mudava o nome do label vetor para .LANCHOR0 e era necessária a mudança para o nome do label certo (.LC0) para o código funcionar.

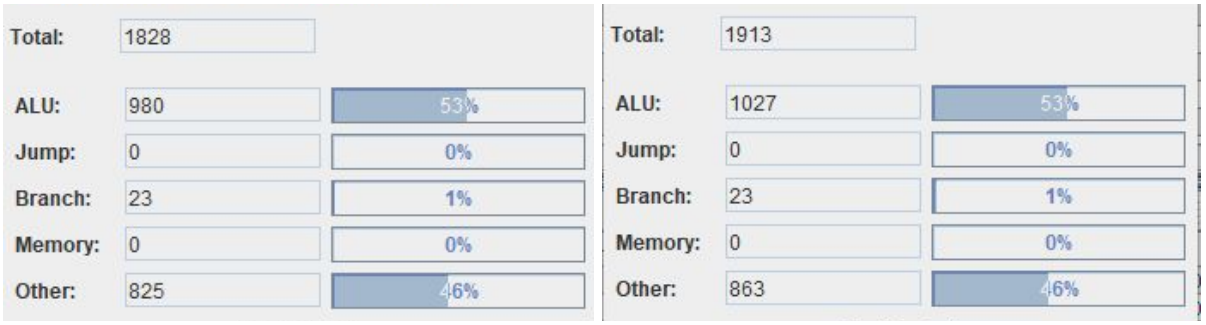
O código para printf e putchar :

```
printf: li a7,1
        mv a0,a1
        ecall
        li a7,11
        li a0,9
        ecall
        ret

putchar: li a7,11
         ecall
         ret
```

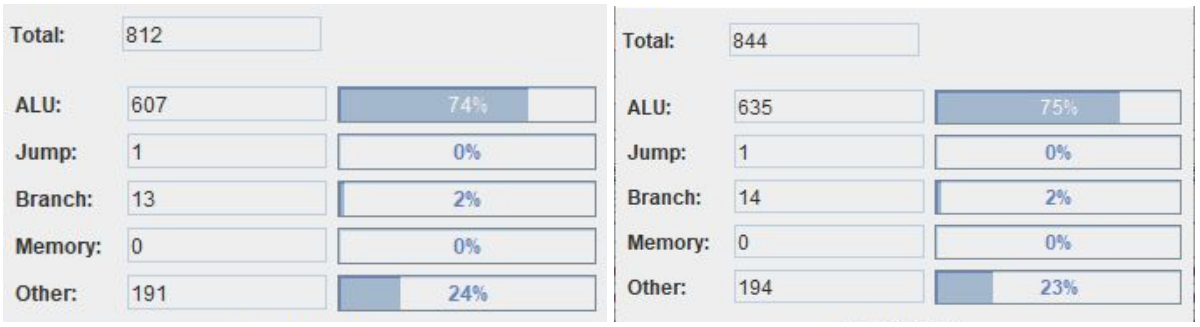
2.4) Para as diferentes diretivas de compilação(Sortc e Sortc2 respectivamente):

-O0:



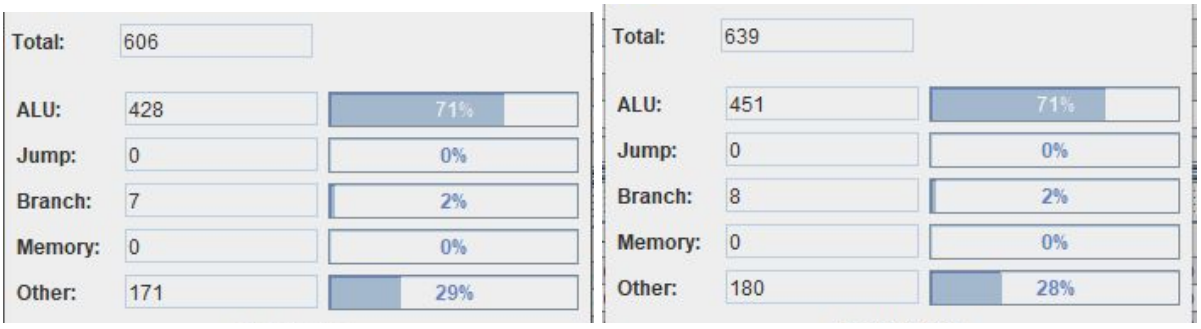
Podemos ver que nessa diretiva, há um acréscimo de instruções realizadas pelo sortc2, indicando uma performance pior para o segundo algoritmo. Diferença de número total de instruções de 85.

-O1:



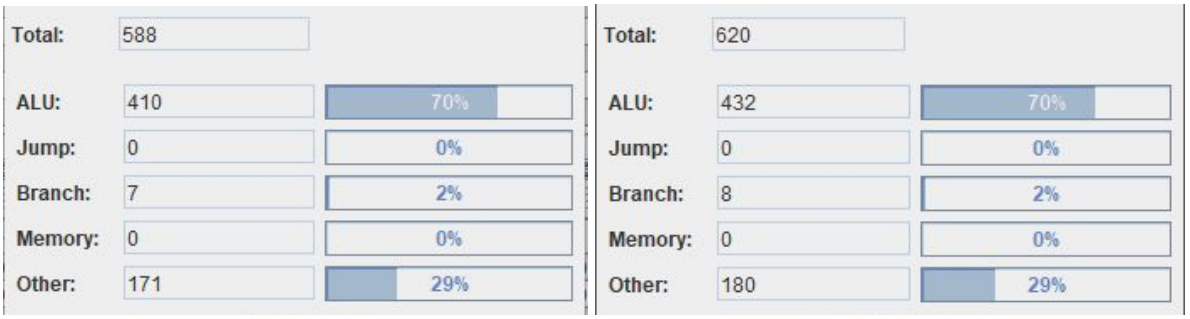
Novamente vemos uma quantidade maior de instruções para o sortc2. E, assim como acima, os dois códigos têm, relativamente com seu próprio número total de instruções, tipos de instruções em mesma proporção. Diferença de 32 instruções.

-O2:



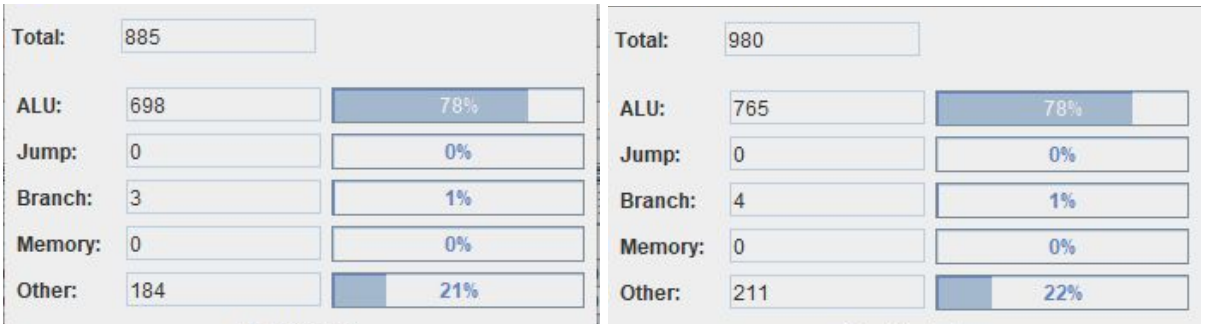
O resultado segue como acima, mesma proporcionalidade de instruções no código. Diferença de 33 instruções.

-O3:



Número próximo de instruções, com diferença de 32 instruções, mesma proporção de utilização da ULA e demais instruções.

-Os:



Diferença no número total de instruções de 95 instruções. Sortc2 continua tendo a maior quantidade. Quase a mesma proporção de usos de diferentes instruções.

TABELA	COMPILAÇÃO					
	Diretivas	-O0	-O1	-O2	-O3	-Os
SORTC						
Número Instruções		1828	812	606	588	885
Tamanho bytes		2.503 bytes	2.047 bytes	1.854 bytes	1.840 bytes	1.987 bytes
SORTC2						
Número Instruções		1913	844	639	620	980
Tamanho bytes		3.012 bytes	2.286 bytes	2.101 bytes	2.087 bytes	1.978 bytes

Temos assim, 3 possíveis bases para comparar os códigos. Em relação às diretivas de compilação temos que em quantidade de instruções, SORTC tem vantagens em relação a SORTC2, com menor quantidade.

Em relação ao tamanho final do arquivo executável, temos que na diretiva -Os, SORTC2 tem tamanho menor em quantidade de bytes.

Com análise de estatística, vemos que os códigos são semelhantes e um não possui maior utilização de recursos que o outro não utiliza. Sendo assim, os códigos têm, em base estatística, comportamentos similares quanto ao uso do processador, memória, etc.

No geral, SORTC tem comportamento mais otimizado que SORTC2.

3) Cálculo das Raízes da Equação de Segundo Grau

3.1) A solução desta questão está no arquivo 3_1 na pasta Lab1>Questao3. O programa contém o procedimento bhaskara, o qual a partir dos valores definidos para a,b e c calcula o valor de delta e retorna 1 caso as raízes sejam reais e 2 caso as raízes sejam complexas conjugadas. A partir da tomada de decisão, o programa salta para a função real ou complexa, em seguida, calcula as raízes e coloca esses valores na pilha.

3.2) A solução desta questão está no arquivo 3_2 na pasta Lab1>Questao3. A função print recebe como parâmetro o valor 1 ou 2 armazenado no registrador a2, que indica se as raízes são reais ou complexas conjugadas, respectivamente. Posteriormente, realiza uma comparação, resgata os valores das raízes na pilha e imprime os valores como foi especificado.

3.3) A versão final que atende a todos os requisitos anteriores e lê os valores dos coeficientes a (ft1), b (ft2) e c (ft3), bem como retorna para a função main e efetua a leitura de outros valores está no arquivo 3_3 na pasta Lab1>Questao3. Para este item, implementamos também uma função que faz o tratamento da exceção, a=0, na qual o programa é interrompido, uma vez que R(1)=NaN e R(2)= $-\infty$.

3.4) A Tabela 3.1 representa os valores das saídas obtidas a partir dos polinômios especificados.

Polinômio [a, b, c]	Saídas
[1, 0, -9.86960440]	$R(1) = 3.1415925$ $R(2) = -3.1415925$
[1, 0, 0]	$R(1) = 0.0$ $R(2) = 0.0$
[1, 99, 2459]	$R(1) = -49.5 + 2.95804i$ $R(2) = -49.5 - 2.95804i$
[1, -2468, 33762440]	$R(1) = 1234.0 + 5678.0i$ $R(2) = 1234.0 - 5678.0i$
[0, 10, 100]	$R(1) = NaN$ $R(2) = -\infty$

Tabela 3.1 - Raízes dos polinômios especificados.

Com o auxílio da ferramenta Instruction Counter, realizamos a contagem das instruções do procedimento bhaskara para os seguintes casos:

- Caso 1: raízes reais

São 30 instruções para o procedimento bhaskara, sendo 21 instruções de ponto flutuante e 9 instruções inteiras

$$t_{exe} = \frac{(21 \cdot 10 + 9 \cdot 1)}{10^9} = 219ns$$

- Caso 2: raízes complexas

São 28 instruções para o procedimento bhaskara, sendo 20 instruções de ponto flutuante e 8 instruções inteiras

$$t_{exe} = \frac{(20 \cdot 10 + 8 \cdot 1)}{10^9} = 208ns$$

4) Tradução de Programas

4.1) A solução para este item encontra-se no arquivo template.s (equivalente ao SYSTEMv1.s) na pasta Lab1>Questao4_final. Foram traduzidas do Assembly MIPS para a arquitetura RV32IMF as funções syscallException, endSyscall, printString, printChar, printHex, clsCLS, printInt, readChar, readString, time, midiOut, midiOutSync, random, sleep, readInt, printFloat, readFloat, endException. Toda a parte de tratamento de exceções e interrupções foram retiradas, assim como foi especificado.

4.2) Para testar o funcionamento das chamadas do sistema implementadas no template.s, executamos o programa testeSYSCALLv1.s da pasta Lab1>Questao4_final usando o Bitmap Display e o Keyboard and Display MMIO Simulator. O vídeo com a execução dos resultados obtidos encontra-se na pasta Lab1>Questao4_final, nomeado de Apresentacao_lab1_Questao4.

Bibliografia

[1] Site: http://web.mit.edu/gnu/doc/html/as_7.html

[2] Site: https://community.arm.com/processors/b/blog/posts/useful-assembler-directives-and-macros-for-the-gnu-assembler

[3] Site: https://developer.apple.com/library/content/documentation/DeveloperTools/Reference/Assembler/040-Assembler_Directives/asm_directives.html#//apple_ref/doc/uid/TP30000823-SW1