Laboratório 1
Assembly RISC - V

Atyrson Souto da Silva, 21/1010341
Breno Costa Avelino Lima, 21/1010280
Eduardo Quirino de Oliveira, 21/1010305
Henrique de Oliveira Ramos, 21/1036052
Pedro Rogrigues Diógenes Macedo, 211042739
Grupo 7

1. Simulador/Montador Rars

1.1.

Tipo de instrução	Nº instruções
Tipo R	1289
Tipo I	1358
Tipo S	415
Tipo B	503
Tipo U	0
Tipo J	442
TOTAL	4007
Programa todo	4677

Table 1. Número de instruções executadas no procedimento *sort* do programa *sort.s* / Número de instruções executadas no programa inteiro.

O tamanho total do programa *sort.s* foi de 276 Bytes.

1.2.

a) O fluxograma do procedimento *sort* pode ser visto abaixo.

Uma forma de analisar o algoritmo é perceber que ele ordena a subsequência de tamanho 0, então de tamanho 1, tamanho 2,tamanho 3...até que tenha ordenado o vetor inteiro. Por isso, a condição de que o elemento da posição do iterador j seja menor do que o elemento da posição i só precisa ser satisfeita uma vez para quebrar o *loop*.

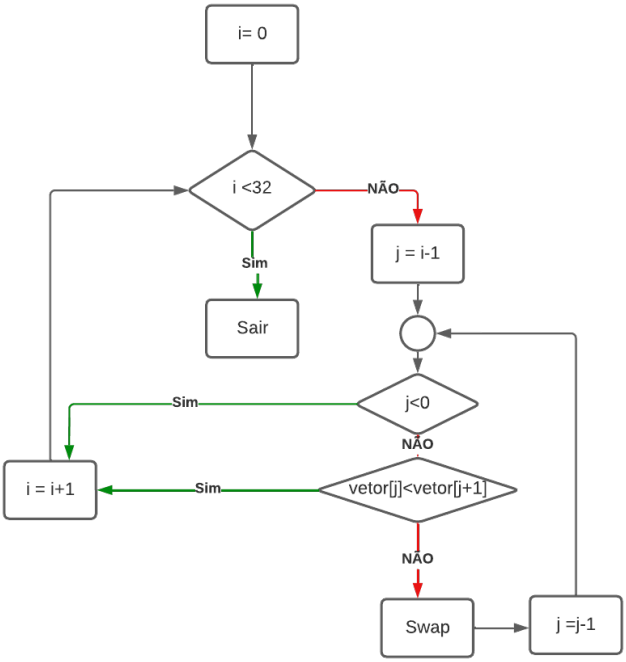

Figure 1. Fluxograma do procedimento

Assim, no caso da entrada do procedimento ser um vetor já ordenado, o laço de repetição sempre é rompido na primeira comparação, o que implica que, para este caso, a quantidade de instruções executadas nesse pode ser obtida por uma uma função afim. Pela contagem de instruções em cada etapa do processo, e pelos resultados obtidos em **b)**, pudemos constatar que $t_o(n) = 10n + 13$. Ademais, fazendo processo similar, encontramos que a expressão para o caso de um vetor inversamente ordenado de tamanho n: $t_i(n) = 9n^2 - 4n + 18$.

b) Coletando os resultados da contagem de instruções executadas no procedimento, considerando uma CPI de 1 ciclo/instrução e uma frequência de *clock* de 50MHz, podemos plotar o seguinte gráfico.

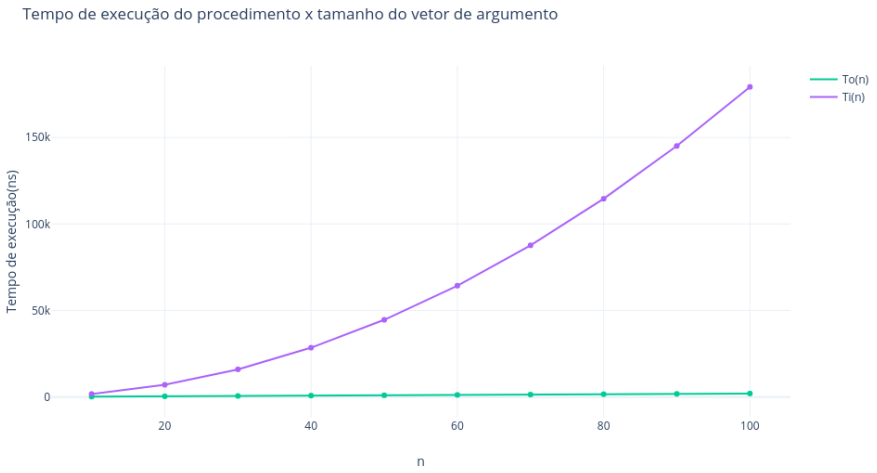

Figure 2.

Percebemos que a discrepância entre $t_o(n)$ e $t_i(n)$ fica maior conforme o n. Além disso, analisando o cenário de pior caso, podemos deduzir que a complexidade do algoritmo utilizado é $O(n^2)$.

2. Compilador cruzado GCC

2.1.

2.2.

Ao compilar o programa *sort.c* com as diretivas *-S -O0* obtivemos o seguinte código em assemble RISC V:

```
v :
        .word   9
        .word   2
        .word   5
        .word   1
        .word   8
        .word   2
        .word   4
        .word   3
        .word   6
        .word   7
        .word   10
        .word   2
        .word   32
        .word   54
        .word   2
        .word   12
        .word   6
        .word   3
```

```
        .word   1
        .word   78
        .word   54
        .word   23
        .word   1
        .word   54
        .word   2
        .word   65
        .word   3
        .word   6
        .word   55
        .word   31
.LC0:
        .string "%d\t"
show:
        addi    sp,sp,-48
        sw      ra,44(sp)
        sw      s0,40(sp)
        addi    s0,sp,48
        sw      a0,-36(s0)
        sw      a1,-40(s0)
        sw      zero,-20(s0)
        j       .L2
.L3:
        lw      a5,-20(s0)
        slli    a5,a5,2
        lw      a4,-36(s0)
        add     a5,a4,a5
        lw      a5,0(a5)
        mv      a1,a5
        lui     a5,%hi(.LC0)
        addi    a0,a5,%lo(.LC0)
        call    printf
        lw      a5,-20(s0)
        addi    a5,a5,1
        sw      a5,-20(s0)
.L2:
        lw      a4,-20(s0)
        lw      a5,-40(s0)
        blt     a4,a5,.L3
        li      a0,10
        call    putchar
        nop
        lw      ra,44(sp)
        lw      s0,40(sp)
        addi    sp,sp,48
```

```
        jr      ra
swap:
        addi    sp,sp,-48
        sw      s0,44(sp)
        addi    s0,sp,48
        sw      a0,-36(s0)
        sw      a1,-40(s0)
        lw      a5,-40(s0)
        slli    a5,a5,2
        lw      a4,-36(s0)
        add     a5,a4,a5
        lw      a5,0(a5)
        sw      a5,-20(s0)
        lw      a5,-40(s0)
        addi    a5,a5,1
        slli    a5,a5,2
        lw      a4,-36(s0)
        add     a4,a4,a5
        lw      a5,-40(s0)
        slli    a5,a5,2
        lw      a3,-36(s0)
        add     a5,a3,a5
        lw      a4,0(a4)
        sw      a4,0(a5)
        lw      a5,-40(s0)
        addi    a5,a5,1
        slli    a5,a5,2
        lw      a4,-36(s0)
        add     a5,a4,a5
        lw      a4,-20(s0)
        sw      a4,0(a5)
        nop
        lw      s0,44(sp)
        addi    sp,sp,48
        jr      ra
sort:
        addi    sp,sp,-48
        sw      ra,44(sp)
        sw      s0,40(sp)
        addi    s0,sp,48
        sw      a0,-36(s0)
        sw      a1,-40(s0)
        sw      zero,-20(s0)
        j       .L6
.L10:
        lw      a5,-20(s0)
```

```
        addi    a5,a5,-1
        sw      a5,-24(s0)
        j       .L7
.L9:
        lw      a1,-24(s0)
        lw      a0,-36(s0)
        call    swap
        lw      a5,-24(s0)
        addi    a5,a5,-1
        sw      a5,-24(s0)
.L7:
        lw      a5,-24(s0)
        blt     a5,zero,.L8
        lw      a5,-24(s0)
        slli    a5,a5,2
        lw      a4,-36(s0)
        add     a5,a4,a5
        lw      a4,0(a5)
        lw      a5,-24(s0)
        addi    a5,a5,1
        slli    a5,a5,2
        lw      a3,-36(s0)
        add     a5,a3,a5
        lw      a5,0(a5)
        bgt     a4,a5,.L9
.L8:
        lw      a5,-20(s0)
        addi    a5,a5,1
        sw      a5,-20(s0)
.L6:
        lw      a4,-20(s0)
        lw      a5,-40(s0)
        blt     a4,a5,.L10
        nop
        nop
        lw      ra,44(sp)
        lw      s0,40(sp)
        addi    sp,sp,48
        jr      ra
main:
        addi    sp,sp,-16
        sw      ra,12(sp)
        sw      s0,8(sp)
        addi    s0,sp,16
        li      a1,30
        lui     a5,%hi(v)
```

```
addi    a0,a5,%lo(v)
call    show
li      a1,30
lui     a5,%hi(v)
addi    a0,a5,%lo(v)
call    sort
li      a1,30
lui     a5,%hi(v)
addi    a0,a5,%lo(v)
call    show
li      a5,0
mv      a0,a5
lw      ra,12(sp)
lw      s0,8(sp)
addi    sp,sp,16
jr      ra
```

Contudo ao colocar esse código no RARS ele não funciona como desejado. Para fazê-lo funcionar primeiro trocamos a função *show* desse código pela função *show* presente no código *sort.s* fornecido pelo professor. Essa troca foi realizada para evitar ter que implementar a função *printf*. Como trocamos a função podemos excluir o dado *.LCO* que era utilizado pela função trocada.

Contudo esse código ainda não funciona da maneira esperada, isso se deve porque a função *main* está no final do código, para resolver esse problema apenas mudamos a função *main* para o início do código.

Por fim apenas trocamos a última linha da *main* pelo ecall de encerrar programa (a7 = 10) para não ocorrer nenhum erro ao final do programa. O programa final que funciona no RARS ficou da seguinte forma:

```
.data
v:
        .word   9
        .word   2
        .word   5
        .word   1
        .word   8
        .word   2
        .word   4
        .word   3
        .word   6
        .word   7
        .word   10
        .word   2
        .word   32
        .word   54
        .word   2
        .word   12
```

```
        .word   6
        .word   3
        .word   1
        .word   78
        .word   54
        .word   23
        .word   1
        .word   54
        .word   2
        .word   65
        .word   3
        .word   6
        .word   55
        .word   31

.text
main:
        addi    sp,sp,-16
        sw      ra,12(sp)
        sw      s0,8(sp)
        addi    s0,sp,16
        li      a1,30
        lui     a5,%hi(v)
        addi    a0,a5,%lo(v)
        call    show
        li      a1,30
        lui     a5,%hi(v)
        addi    a0,a5,%lo(v)
        call    sort
        li      a1,30
        lui     a5,%hi(v)
        addi    a0,a5,%lo(v)
        call    show
        li      a5,0
        mv      a0,a5
        lw      ra,12(sp)
        lw      s0,8(sp)
        addi    sp,sp,16

        li  a7,10
        ecall

show:   mv  t0,a0
        mv  t1,a1
        mv  t2,zero
```

```
loop1:  beq  t2,t1,fim1
        li  a7,1
        lw  a0,0(t0)
        ecall
        li  a7,11
        li  a0,9
        ecall
        addi  t0,t0,4
        addi  t2,t2,1
        j  loop1

fim1:   li  a7,11
        li  a0,10
        ecall
        ret

swap:
        addi    sp,sp,-48
        sw      s0,44(sp)
        addi    s0,sp,48
        sw      a0,-36(s0)
        sw      a1,-40(s0)
        lw      a5,-40(s0)
        slli    a5,a5,2
        lw      a4,-36(s0)
        add     a5,a4,a5
        lw      a5,0(a5)
        sw      a5,-20(s0)
        lw      a5,-40(s0)
        addi    a5,a5,1
        slli    a5,a5,2
        lw      a4,-36(s0)
        add     a4,a4,a5
        lw      a5,-40(s0)
        slli    a5,a5,2
        lw      a3,-36(s0)
        add     a5,a3,a5
        lw      a4,0(a4)
        sw      a4,0(a5)
        lw      a5,-40(s0)
        addi    a5,a5,1
        slli    a5,a5,2
        lw      a4,-36(s0)
        add     a5,a4,a5
        lw      a4,-20(s0)
        sw      a4,0(a5)
```

```
        nop
        lw      s0,44(sp)
        addi    sp,sp,48
        jr      ra
sort:
        addi    sp,sp,-48
        sw      ra,44(sp)
        sw      s0,40(sp)
        addi    s0,sp,48
        sw      a0,-36(s0)
        sw      a1,-40(s0)
        sw      zero,-20(s0)
        j       .L6
.L10:
        lw      a5,-20(s0)
        addi    a5,a5,-1
        sw      a5,-24(s0)
        j       .L7
.L9:
        lw      a1,-24(s0)
        lw      a0,-36(s0)
        call    swap
        lw      a5,-24(s0)
        addi    a5,a5,-1
        sw      a5,-24(s0)
.L7:
        lw      a5,-24(s0)
        blt     a5,zero,.L8
        lw      a5,-24(s0)
        slli    a5,a5,2
        lw      a4,-36(s0)
        add     a5,a4,a5
        lw      a4,0(a5)
        lw      a5,-24(s0)
        addi    a5,a5,1
        slli    a5,a5,2
        lw      a3,-36(s0)
        add     a5,a3,a5
        lw      a5,0(a5)
        bgt     a4,a5,.L9
.L8:
        lw      a5,-20(s0)
        addi    a5,a5,1
        sw      a5,-20(s0)
.L6:
        lw      a4,-20(s0)
```

```
lw      a5,-40(s0)
blt     a4,a5,.L10
nop
nop
lw      ra,44(sp)
lw      s0,40(sp)
addi    sp,sp,48
jr      ra
```

2.3.

Para essa questão fizemos exatamente o que foi pedido no enunciado: compilamos o arquivo *sortc-mod.c* para diferentes diretivas de otimização da compilação e, fazendo as devidas mudanças para executar o código no **RARS**, coletamos o número de instruções executadas (utilizando a ferramenta *Instruction Counter*) e o tamanho em Bytes de cada programa compilado. Com isso, obtivemos a seguinte tabela:

Diretiva	Nº instruções	Tamanho (Bytes)
-O0	9786	520
-O1	3886	364
-O2	2174	272
-O3	2173	272
-Os	4097	340

Comparando com o programa *sort.s* da questão 1.1, é notável que ela executa mais instruções, porém possui tamanho menor que o código compilado na diretiva -Os.

3. Jogo da velha

link do vídeo

link 2, colocado depois de 23:55, apesar de ter sido gravado antes do limite de entrega (percebemos depois que havíamos feito o upload no youtube do arquivo errado): link 2 do vídeo

References

TOCCI, Ronald J.; WIDMER, Neal S.; MOSS, Gregory L..Sistemas Digitais: Princípios e Aplicações. 12ª ed. São Paulo: Pearson, 2019