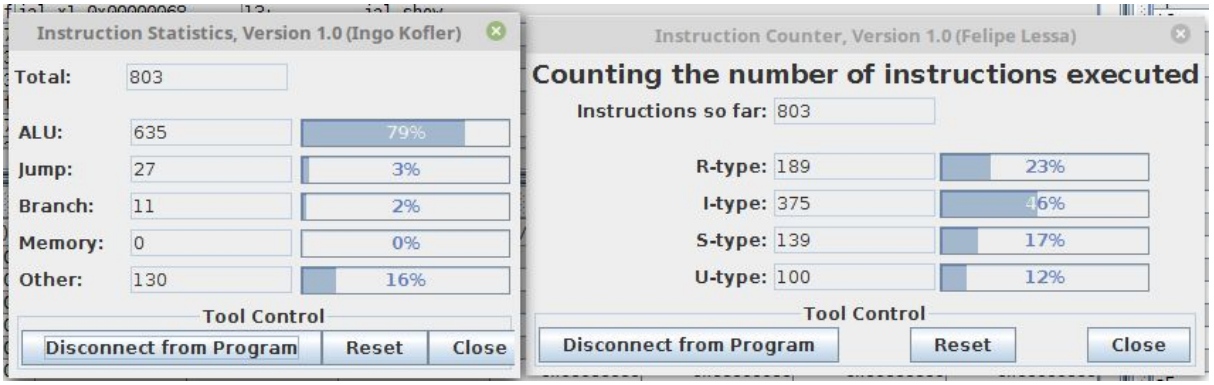
Laboratório 1

Alunos:

- Caio Couto Moreira (15/0155433)
- Gabriel Alcantara(15/0126701)
- Breno Felipe(16/0003318)
- João Pedro Ramos do Amaral(14/0056394)

1 - Simulador/Montador Rars

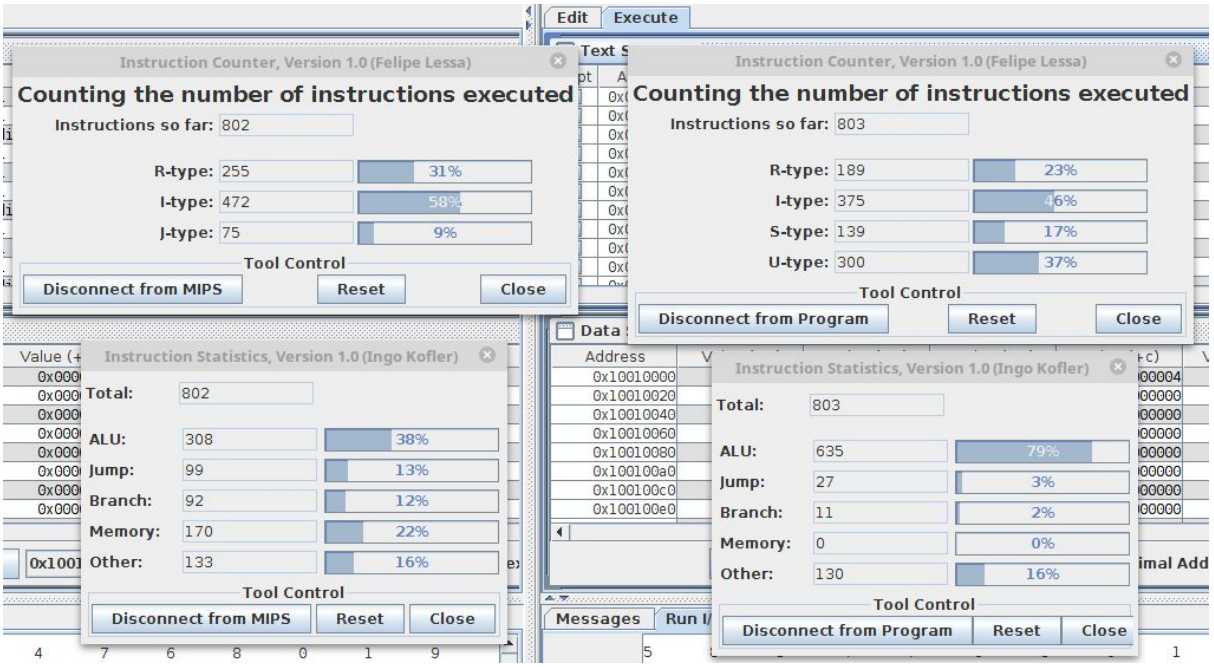
1.1)



Tamanho em bytes:
Endereço de início: 0x00400000
Endereço final: 0x00400128
Total: 296 bytes

1.2)

MIPS X RISC



Melhorando desempenho:

Fator de desempenho (considerando as quantidade de instruções):
RARS: 802
MIPS: 802
Obs: ignorado a instrução nop.

RARS/MIPS: 1

1.3)

Fórmula: Texec = I * CPI * 1 / F
CPI = 1
T = 1 / 50 * (10 ^ 6).

 # N = 1
Melhor Caso/Pior caso:
81 instruções

 # N = 2
Melhor Caso(vetor já ordenado):
116 instruções

Pior caso(vetor reverso)
130 instruções

 # N = 3
Melhor Caso(vetor já ordenado):
151 instruções
Pior caso(vetor reverso)
199 instruções

 # N = 4
Melhor Caso(vetor já ordenado):
186 instruções
Pior caso(vetor reverso)
288 instruções

 # N = 5
Melhor Caso(vetor já ordenado):
221 instruções
Pior caso(vetor reverso)
397 instruções

 # N = 6
Melhor Caso(vetor já ordenado):
256 instruções
Pior caso(vetor reverso)
526 instruções

 # N = 7
Melhor Caso(vetor já ordenado):
291 instruções
Pior caso(vetor reverso)
675 instruções

 # N = 8
Melhor Caso(vetor já ordenado):
326 instruções
Pior caso(vetor reverso)
844 instruções

 # N = 9
Melhor Caso(vetor já ordenado):
361 instruções
Pior caso(vetor reverso)
1033 instruções

 # N = 10
Melhor Caso(vetor já ordenado):
395 instruções

Pior caso(vetor reverso)
1242 instruções

 # N = 20
Melhor Caso(vetor já ordenado):
745 instruções

Pior caso(vetor reverso)
4431 instruções

 # N = 30
Melhor Caso(vetor já ordenado):
1095 instruções

Pior caso(vetor reverso)
9621 instruções

 # N = 40
Melhor Caso(vetor já ordenado):
1445 instruções

Pior caso(vetor reverso)
16811 instruções

 # N = 50
Melhor Caso(vetor já ordenado):
1795 instruções

Pior caso(vetor reverso)
26001 instruções

 # N = 60
Melhor Caso(vetor já ordenado):
2145 instruções
Pior caso(vetor reverso)
37191 instruções

 # N = 70
Melhor Caso(vetor já ordenado):
2495 instruções

Pior caso(vetor reverso)
50381 instruções

 # N = 80
Melhor Caso(vetor já ordenado):
2845 instruções

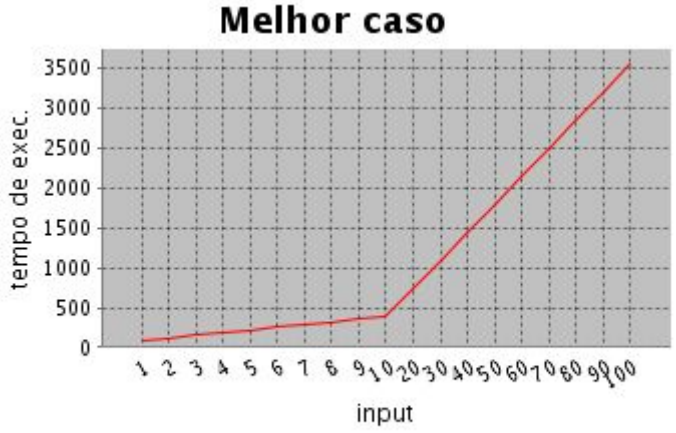
Pior caso(vetor reverso)
65571 instruções

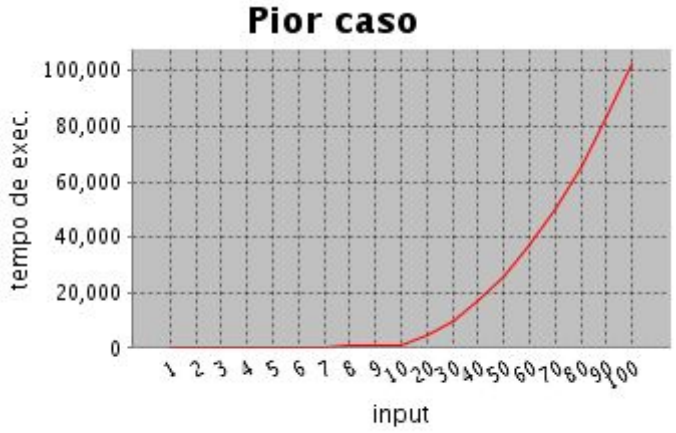
 # N = 90
Melhor Caso(vetor já ordenado):
3195 instruções

Pior caso(vetor reverso)
82761 instruções

 # N = 100
Melhor Caso(vetor já ordenado):
3545 instruções

Pior caso(vetor reverso)
101951 instruções





Obs.: nos gráficos acima consideramos a quantidade de instruções em relação ao tempo de execução.

2) Compilador GCC

2.1) Feito em sala.

2.2)

.file: nome do arquivo local
.option:
.align: alinha os endereços de memória (0=byte, 1=half, 2=word, 3=double), no caso do 2, ele coloca o programa na memória alinhado em word ou seja os 2 primeiros bits do endereço são 0.
.globl: declara um label em um scopo global, um ponto de entrada do programa
.type: pode indicar por exemplo ".type main, @function" que main é uma função.
.size: gerada pelo compilador para auxiliar com informações na hora do debugging

.ident: usada por alguns assemblers para colocar rótulos em object files. as para fins de compatibilidade.
.data: Itens subsequentes armazenados no segmento de dados no próximo endereço disponível
.eqv: substitui o segundo operando pelo primeiro
.text: Itens subsequentes(instruções) armazenados no segmento de textos no próximo endereço disponível

Implementação com melhor desempenho:

Vetor Local x Vetor Global

```
50  .LC0:
51          .word   5
52          .word   8
53          .word   3
54          .word   4
55          .word   7
56          .word   6
57          .word   8
58          .word   0
59          .word   1
60          .word   9
61          .text
62          .align  2
63          .globl  main
64          .type   main, @function
```

```
 7  v:
 8          .word   9
 9          .word   2
10          .word   5
11          .word   1
12          .word   8
13          .word   2
14          .word   4
15          .word   3
16          .word   6
17          .word   7
18          .section        .rodata
19          .align  3
```

No vetor local temos o endereço de memória alinhado em word .align 2, enquanto que no vetor global temos que nosso endereço está alinhado em double .align 3, sendo assim o vetor global ocupa mais espaço para armazenamento dos valores do vetores, uma vez que seu endereço é alinhado em double, temos então que o vetor local tende a ter o melhor desempenho.

2.3) Tomando como exemplo o arquivo teste4.c dos ArquivosC do laboratório 1, foi gerado o seguinte código em assembly compilado com o código:
$ riscv64-unknown-elf-gcc -S -march=rv64imafd -O1 teste4.c

teste4.s*

```
1         .file   "teste4.c"
2         .option nopic
3         .text
4         .align  2
5         .globl  main
6         .type   main, @function
7   main:
8         addi    sp,sp,-16
9         sd      ra,8(sp)
10        sd      s0,0(sp)
11        li      s0,12288
12        addi    a1,s0,57
13        lui     a0,%hi(.LC0)
14        addi    a0,a0,%lo(.LC0)
15        call    printf
16        addi    a0,s0,59
17        ld      ra,8(sp)
18        ld      s0,0(sp)
19        addi    sp,sp,16
20        jr      ra
21        .size   main, .-main
22        .section        .rodata.str1.8,"aMS",@progbits,1
23        .align  3
24  .LC0:
25        .string "Numero: %d\n"
26        .ident  "GCC: (GNU) 7.2.0"
27
```

Nesse exemplo foi necessário:
- retirar todas as diretivas
- criar as diretivas **.data** e **.text**
- alterar as operações sd para sw
- retirar a linha 15: *call printf*, uma vez que essa função pertence a biblioteca <stdio.h>, nesse caso para imprimir uma string, foi declarado na diretiva .data uma label com a string que quero printar, e para imprimir o inteiro, usamos o ecall para printar o valor do registrador a0
- retirar as pseudo instrução da linha 20: jr ra

Resultando no seguinte código abaixo:

teste4.s*

```
1  .data
2
3          .LC0: .asciz "Numero: "
4
5  .text
6          addi    sp,sp,-16
7          sw      ra,8(sp)
8          sw      s0,0(sp)
9          li      s0,12288
10         addi    a1,s0,57
11         lui     a0,%hi(.LC0)
12         addi    a0,a0,%lo(.LC0)
13
14         li  a7, 4
15         la  a0, .LC0
16         ecall
17
18
19         addi    a0,s0,59
20         lw      ra,8(sp)
21         lw      s0,0(sp)
22         addi    sp,sp,16
23
24         li  a7, 1
25         ecall
```

3) Cálculo das raízes da equação de segundo grau

Feita em anexo.

4) Tradução de Programas

4.1) Feita em anexo

4.2) O vídeo se encontra no seguinte canal:
https://www.youtube.com/channel/UC-2AtGqtNboqsgExOfx8OZA?view_as=subscriber

Com o seguinte título: Laboratório 1 OAC - Questão 4.2