



Laboratório 1 **- Assembly RISC-V -**

Objetivos:

- Familiarizar o aluno com o Simulador/Montador Rars;
- Desenvolver a capacidade de codificação de algoritmos em linguagem Assembly;
- Desenvolver a capacidade de análise de desempenho de algoritmos em Assembly;

(3.0) 1) Compilador cruzado GCC

Um compilador cruzado (*cross compiler*) compila um código fonte para uma arquitetura diferente daquela da máquina em que está sendo utilizado. Você pode baixar gratuitamente os compiladores gcc para todas as arquiteturas (RISC-V, ARM, MIPS, x86, etc.) e instalar na sua máquina, sendo que o código executável gerado apenas poderá ser executado em uma máquina que possuir o processador para qual foi compilado. No gcc, a diretiva de compilação `-s` faz com que o processo pare com a geração do arquivo em assembly e a diretiva `-march` permite definir a arquitetura a ser utilizada.

```
Ex.: riscv64-unknown-elf-gcc -S -march=rv32imf -mabi=ilp32f # RV32IMF
    arm-eabi-gcc -S -march=armv7 # ARMv7
    gcc -S -m32 # x86
```

Para fins didáticos, o site [Compiler Explorer](#) disponibiliza estes (e vários outros) compiladores C (com diretiva `-s`) *on-line* para as arquiteturas RISC-V, ARM, x86 e x86-64. (usar C e compilador RISC-V rv32gc 10.2).

(0.0) 2.1) **Teste a compilação para Assembly RISC-V com programas triviais em C disponíveis no diretório 'ArquivosC', para entender a convenção do uso dos registradores e memória utilizada pelo gcc para a geração do código Assembly, usando as diretivas de otimização `-O0` e `-O3`. Ver exemplos com os limitantes da ISA RV32I `-march=rv32i` (mul, div e float).**

(1.0) 2.2) **Dado o programa `sortc.c`, compile-o com a diretiva `-O0` e obtenha o arquivo `sortc.s`. Indique as modificações necessárias no código Assembly gerado para que possa ser executado corretamente no Rars.**

Dica: Uso de Assembly em um programa em C. Use a função `show` definida no `sort.s` para não precisar implementar a função `printf`, conforme mostrado no `sortc_mod.c`

(2.0) 2.3) **Compile o programa `sortc_mod.c` e, com a ajuda do Rars, monte uma tabela comparativa com o número total de instruções executadas pelo **programa todo**, e o tamanho em bytes dos códigos em linguagem de máquina gerados para cada diretiva de otimização da compilação `{-O0, -O1, -O2, -O3, -Os}`. Compare ainda com os resultados obtidos no item 1.1) com o **seu** programa `sort.s` que foi implementado diretamente em Assembly. Analise os resultados obtidos.**

(0.0) 2.4) Exemplos de uso da linguagem C para acesso às ferramentas KDMIO e BITMAP DISPLAY (`teste10.c`).

(7.0) 3) Solução de Labirintos

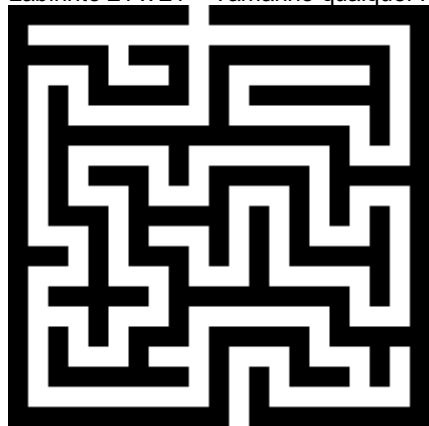
Dado um labirinto de tamanho arbitrário entre 5x5 e 319x239, com caminhos de cor branca e paredes de cor diferente de branca, ambos com tamanho de 1 pixel.

Encontre o caminho que liga a entrada (na parte superior) à saída (na parte inferior).

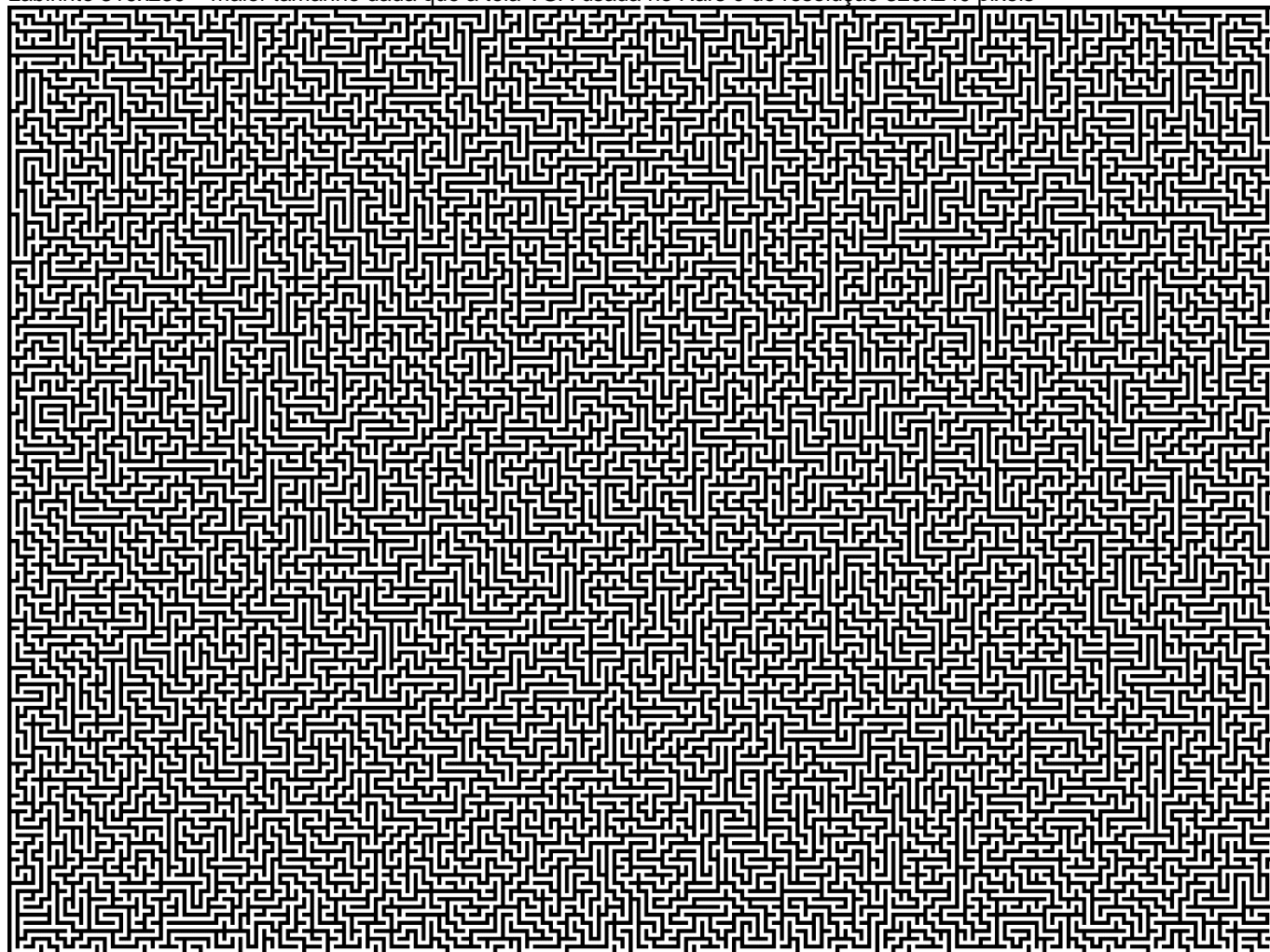
Labirinto 5x5 – Menor tamanho considerado



Labirinto 21 x 21 – Tamanho qualquer AxB onde A é um número ímpar entre 5 e 319 e B um número ímpar entre 5 e 239.



Labirinto 319x239 – maior tamanho dada que a tela VGA usada no Rars é de resolução 320x240 pixels



(0.0) 3.1) Crie um programa principal `main()` que importe no segmento de dados `(.data)` um arquivo de nome `'MAZE.s'`, criado através da ferramenta `oac2bmp2.exe`, desenhe o labirinto no centro da tela, encontre a solução e faça uma animação da solução encontrada.

```
.data
.include "MAZE.s"
CAMINHO: .space 153600      # Estimativa de pior caso: 4x 320x240/2 tamanho do maior labirinto

.text
MAIN:  la a0,MAZE
      jal draw_maze
      la a0,MAZE
      la a1,CAMINHO
      jal solve_maze
      la a0,CAMINHO
      jal animate
      li a7,10
      ecall
```

(0.0) 3.2) Escreva uma função `void draw_maze(int *labirinto)` que receba o ponteiro para a estrutura da imagem importada no segmento `.data` e desenhe o labirinto no centro da tela.

Ex.:

```
.data
MAZE: .word 21,21          # número de colunas, número de linhas do labirinto
      .byte 0,0,0,0,...
```

```
.text
la a0,MAZE
jal draw_maze      # desenha no centro da tela
```

(0.0) 3.3) Escreva um procedimento `void solve_maze(int *maze, int *caminho)` que receba o ponteiro para a estrutura da imagem do labirinto e o ponteiro para a estrutura que conterá o caminho que resolve o labirinto na memória de dados.

Sugestão: A estrutura da solução pode ser uma matriz $N \times 2$, onde N é o número de passos para se chegar da entrada até a saída, e cada passo é caracterizado pelas coordenadas (coluna,linha) que deve ser seguida.

Ex.:

```
.data CAMINHO: .word 100, 160,120, 160,121, 160,122, 161,122, ...  # onde 100 inicial é o número de
passos e as duplas (160,120) (160,121)... o caminho a ser seguido.
```

(0.0) 3.4) Escreva um procedimento `void animate(int *caminho)` que receba o ponteiro para a estrutura do caminho e faça uma animação de um pixel de cor vermelha (ou a sua escolha) indo da entrada até a saída.

(1.5) 3.5) Para 20 labirintos aleatórios de tamanho 51×51 (Columns 25 Rows 25), faça um histograma do número de instruções (I) necessárias à execução do procedimento `solve_maze`. Analise o histograma obtido (média, desvio padrão, tipo de distribuição etc.).

(1.5) 3.6) Para 20 labirintos aleatórios de tamanhos $N \times N$: 5×5 , 11×11 , 17×17 ... 119×119 , faça um gráfico de $N \times I$, sendo I o número de instruções necessárias à execução do procedimento `solve_maze`. Analise o gráfico obtido.

(4.0) 3.7) No seu computador e dado o workload definido pela execução do procedimento `main` com um labirinto de tamanho 319×239 , a) Qual a contagem de Instruções I ? b) Qual o tempo de execução t_{exec} ? c) Considerando que o processador que o Rars simula possui $CPI=1$, qual a frequência do processador RISC-V equivalente? d) Qual o tamanho da pilha (em bytes) foi necessária à execução do programa? e) Filme a execução do programa colocando o link do vídeo no relatório.

Dica: Para gerar novos labirintos e de outros tamanhos:

- 1) Acesse o site <https://keesiemeijer.github.io/maze-generator/>
- 2) Para um labirinto de tamanho máximo (319×239) use os seguintes parâmetros

Wall thickness:	<input type="text" value="1"/>
Columns:	<input type="text" value="159"/>
Rows:	<input type="text" value="119"/>
Maze entries:	<input type="button" value="top and bottom"/>
Bias:	<input type="button" value="none"/>

- 3) Salve (ou copie) a imagem e abra no paint.net. Salve como `MAZE.bmp` com 24 bits/pixel.
- 4) Execute: `oac2bmp2 MAZE` que criará o arquivo `MAZE.s` que deverá ser incluído (`.include`) no `.data` do seu programa

Dicas: o RISC-V possui um banco de registradores de Status e Controle (visto mais tarde) no qual armazena continuamente diversas informações úteis, e que podem ser lidos pela instrução:

```
csrr t1, fcsr #Read control and status register
```

onde `t1` é o registrador de destino da leitura e `fcsr` é um imediato de 12 bits correspondente ao registrador a ser lido.

Os registradores abaixo são registradores de 64 bits que contém as informações:

`{timeh, time}` = tempo do sistema em ms

`{instreth, instret}` = número de instruções executadas

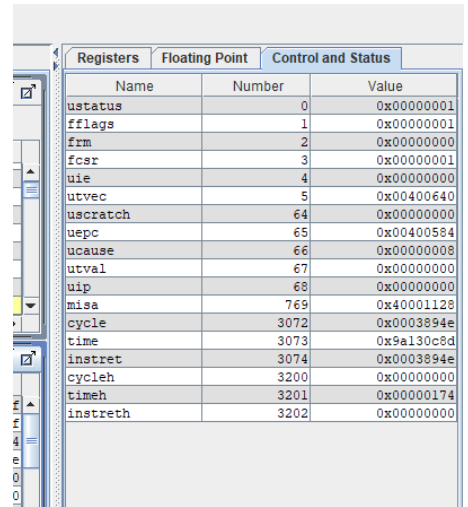
`{cycleh, cycle}` = número de ciclos executados (se `CPI=1` é igual ao `instret`)

Geralmente nossos programas não precisarão dessa precisão de 64 bits. Podemos usar então apenas os 32 bits menos significativos.

Ex.: Para medir o tempo e o número de instruções do procedimento PROC para os registradores `s0` e `s1` respectivamente.

```
Main: ...
...
csrr s1,3074 # le o num instr atual
csrr s0,3073 # le o time atual
jal PROC
csrr t0,3073 # le o time atual
csrr t1,3074 # le o num instr atual
sub s0,t0,s0 # calcula o tempo
sub s1,t1,s1 # calcula o numero de instruções
...
```

Note que terá um erro de 2 instruções na medida do número de instruções. Por quê?



Name	Number	Value
ustatus	0	0x00000001
fflags	1	0x00000001
fpm	2	0x00000000
fcsr	3	0x00000001
uie	4	0x00000000
utvec	5	0x00400640
uscratch	64	0x00000000
uepc	65	0x00400584
ucause	66	0x00000008
utval	67	0x00000000
uip	68	0x00000000
misa	769	0x40001128
cycle	3072	0x0003894e
time	3073	0x9a130c8d
instret	3074	0x0003894e
cycleh	3200	0x00000000
timeh	3201	0x00000174
instreth	3202	0x00000000

Para a apresentação da verificação dos laboratórios (e projeto) nesta disciplina, crie um canal para o seu grupo no YouTube e poste os vídeos dos testes (sempre com o nome 'UnB – OAC Turma A - 2020/2 – Grupo Y - Laboratório X - <palavras-chaves que identifiquem este vídeo em uma busca>'), coloque os links clicáveis no relatório.

Passos do vídeo:

- i) Apresente o grupo e seus membros;
- ii) Explique o projeto a ser realizado;
- iii) Apresente os testes solicitados;
- iv) Apresente suas conclusões.