UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

GEOVANO QUATRIN
RAFAELLE ARRUDA

TRABALHO 01: JOGO DE BATALHA NAVAL EM ASSEMBLY

CHAPECÓ
2021/09

SUMARIO

INTRODUÇÃO

É de todo conhecimento geral que, batalha naval é composta por um jogo de tabuleiro contendo dois jogadores, no qual ambos têm que adivinhar em que quadrados estão os navios das oponentes. Mas de acordo com este trabalho foi realizado da seguinte forma, criar o tabuleiro e contendo uma string com a seguintes informações como direcionamento dos navios (vertical ou horizonta), quantidade e o eixo x e y para alocar os navios, tendo pequenas validações e com isso inicia uma bela partida de game. Antes de tudo, para compreender este trabalho teve uma implementação de testes para compreender a elaboração do programa final, com isso teve uma programação de auto nível em c para compreender os passos para realização da implementação em assembly.

Como sabemos, a linguagem assembly é de baixo nível e segue uma sequência de instruções e métodos para elaboração de um simples programa, houve uma demanda grande em pesquisa de como usar cada instrução e ter uma boa noção sobre registradores e lógica de programação.

1. Problema Apresentado

Neste trabalho foi proposto a implementação de um jogo de batalha naval utilizando o conjunto de instruções do processador RISC-V RV32IM, com uma matriz 10 x 10. Como foi mencionado anteriormente o jogo de batalha naval é um jogo para dois jogadores cujo objetivo é afundar os navios (de diferentes tamanhos) do adversário, os quais são dispostos em uma matriz. Cada linha e coluna da matriz é identificada por um número entre 0 e 9. Na versão a ser implementada teremos apenas um jogador que fará os disparos contra os navios do inimigo, que será a própria “máquina” ou “sistema”. O programa a ser desenvolvido deverá fazer o controle do jogo e a interface com o usuário.

A cada jogada, uma coordenada (linha e coluna) é fornecida para que o programa verifique e diga se acertou algo. O jogo/rodada termina quando o jogador afunda todas as embarcações presentes na matriz.

Os navios do inimigo estão colocados em uma string chamada “navios” presente na área de dados (.data), que deve ser lida pela função insere_embarcacoes no início do jogo. A string navios possui o seguinte padrão. Na primeira linha é informado o número de navios inseridos. Cada uma das linhas seguintes possui um navio. As linhas que especificam navios possuem 4 valores, separados por um espaço, sendo: o primeiro valor é a disposição do navio sendo, 0 para navio na horizontal e 1 para navio na vertical; o segundo valor é o comprimento do navio; o terceiro valor é a linha inicial do navio e; o quarto valor é a coluna inicial do navio. Observe o Exemplo:

```
3
1 5 1 1
0 5 2 2
0 1 6 4
```

O posicionamento de navios mostrado acima resulta no seguinte posicionamento:

Figura 1. Matriz de navios

A princípio a função insere_embarcacoes deve verificar a validade do posicionamento dos navios, gerando uma mensagem de erro para as seguintes regras que são:

I. A posição do navio é invalida. Exemplo 0 3 11 7, o erro foi no eixo x da matriz, pois o mesmo é de 10x10.
II. O navio extrapola as dimensões da matriz. Exemplo 0 4 2 7, o erro foi a quantidade de navios horizontal.
III. Ocorre sobreposição dos navios. Exemplo 0 4 2 2e 1 3 0 3, erro quando o mesmo navio ocupa a posição que já esteja ocupada.

Com a partida iniciada, deve ser possível reiniciar o jogo, mostrar o estado atual da matriz de navios e fazer uma nova jogada. Caso queira terminar, poderá finalizar a partida. Entretanto, a interface vai ser exposta no próprio terminal do programa, com a matriz 10x10 gerada e seguindo as regras do posicionamento de cada navio, caso ao contrario emitirá uma mensagem de erro para o usuário arrumar a string de coordenadas.

A interface de jogo deve apresentar três grupos de informações:

I. A situação atual da matriz de jogo;
II. A quantidade de tiros já disparados, a quantidade de tiros que acertaram o alvo, a quantidade de barcos já afundados e a posição do último tiro disparado e;
III. Os recordes do jogo que consiste no menor número de tiros para afundar todas as embarcações de algum jogo anterior. Veja um exemplo:

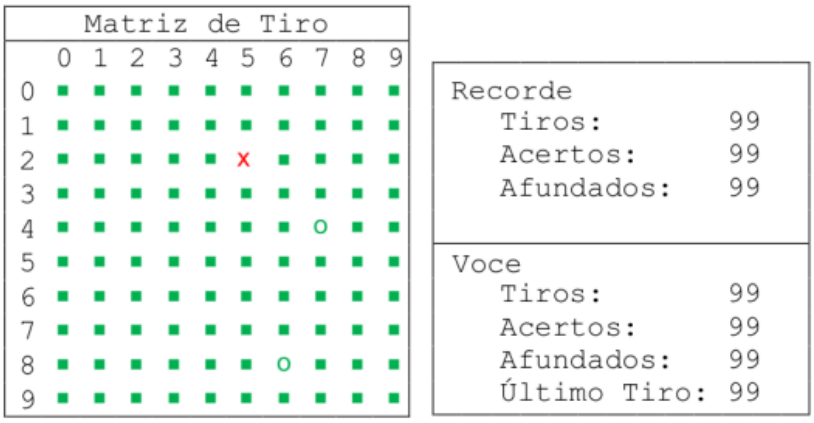

Figura 2. Matriz de tiros com painel de jogadas

2. Solução do desenvolvimento

De acordo com o desenvolvimento do programa, foi que separados por etapa são elas como imprimir a matriz, adicionar os navios de vetores, funções de verificações de dados e ainda tendo informação da linguagem C como base para implementação deste trabalho, pode-se afirmar que foi usado para aperfeiçoar os conhecimentos adquiridos para não houver algum bloqueio de ideia, incluso no projeto ainda tem a pasta de teses que contem conteúdos que foram realizados como base para conclusão do protótipo.

A princípio é a tela de configuração é mostrada quando o usuário seleciona a opção jogar no menu inicial. O programa deve apresentar uma matriz de navios (10x10). A matriz deve possuir cada linha e coluna identificada por um “-“ que está em asciz. Podemos ver que temos uma word com dados de informados no problema como direcionamento das embarcações, quantidade e a posição do eixo y, x, sendo lidos quando inicializa o programa, é valido lembrar que código de trabalho foi separados em partes para compressão do programa e visualização do funcionamento do game. Para isso, na main usamos o método include aonde importamos ou melhor usamos os métodos criados para fazer cada função determina e assim realizará seu objetivo. Observe abaixo o jeito que inicializamos um include na main, desta forma:

```
11   .include "PromptTexts.asm"
12   preenchimento1: .word 0x2d
13   char_tiro_certeiro: .word 0x78
14   char_tiro_nagua: .word 0x6f
15   inicio_maiusculas: .word 0x41
16   ordem: .word 10
17   vetor_xy: .word 0, 0
```

Figura 3. Início do Main do código fonte.

Note que, estamos incluindo o método ‘‘. include "PromptTexts.asm" ’’, que contém strings de interface com o usuário aonde exige a matriz (10x10), e perguntas do tipo aonde o usuário deseja dispara, qual linha e coluna e tendo mensagem de erro caso tenha já atirado em alguma posição da matriz e verificando as posições dos navios se não extrapolam a dimensão do tabuleiro, contendo informações de quantidade de tentativas, tiros, estatísticas e podendo adicionar navios no jogo para entretenimento do jogo.

O início de compilação do jogo batalha naval sempre começa com a main, e com isso vai executando cada função exigida pelo include e mostrando a matriz, e o menu com as informações para o usuário jogar. A princípio foi usado o cálculo de para transformar a posição de linha e coluna (L e C) em deslocamento na memória em relação à posição inicial da matriz:

Deslocamento = (L * QTD_colunas + C) * 4
Onde: L = linha; C = Coluna; QTD_colunas = quantidade de colunas da matriz; constante 4 em função de que cada inteiro ocupa 4 células de memória

Na interação do jogo (a cada jogada) é possível reiniciar o jogo, mostrar o estado atual da matriz de navios e ainda fazer uma nova jogada, com isso temos um menu repleto de funciolidades como verificar aonde está os navios que mostra uma matriz apenas dos navios, informações de total de tiros, acertos e fora (tiro na agua), observo a imagem abaixo do menu:

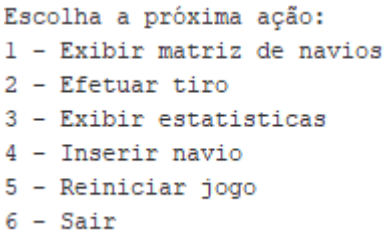

Figura 4. Menu do jogo.

3.Conclusão

A princípio foi uma experiência boa, pois nunca tinha trabalhado com linguagem de baixo nível. Isso nos ajuda a desenvolver algoritmos de alto nível melhores e tendo conhecimentos em maquinas e como são projetas desde do seu início para reta final. Ressaltando que, este projeto foi um trabalho conjunto e troca de conhecimento de ambos os membros deste protótipo, aonde foi crescendo a ideia de como melhorar o código e elabora a melhora performance possível e ensinamento que um poderia repassar para o outro.

4. Referências

Organização Básica de Computadores e linguagem de montagem, disponível em https://www.ic.unicamp.br/~edson/disciplinas/mc404/2019-2s/ab/slides/mc404_07.introducao_riscv.pdf.

Great Ideas in Computer Architecture More RISC-V Instructions and How to Implement Functions, disponível em https://inst.eecs.berkeley.edu/~cs61c/fa17/lec/06/L06%20RISCV%20Functions%20(1up).pdf

RISC-V Assembly Programmer's Manual, disponível em https://github.com/riscv-non-isa/riscv-asm-manual/blob/master/riscv-asm.md

Supported Instructions, disponível em https://github.com/TheThirdOne/rars/wiki/Supported-Instructions

Using the command line, disponível em https://github.com/TheThirdOne/rars/wiki/Using-the-command-line

5. Código fonte Anexado

Nessa secção o código fonte foi divido em nove partes, segue em anexo o manual dessa implementação.

1 . Arquivo, Main.asm

```
#####
                # 2021.2 - Organização de Computadores
                # Trabalho 1 - Batalha Naval
                # Professor Luciano L. Caimi
                # Estudantes:
                # Nome: Geovano Lago Quatrin Matrícula: 1721101007
                # Nome: Rafaelle Arruda      Matrícula: 1721101029
        #####
                .data
        #####
         # Definição das constantes e das variáveis de memória.
         # Legenda dos comentários:
                # V: variável de memória
                # C: constante armazenada em memória
                # Em: estrutura de dados do tipo matriz em memória
                # Ev: estrutura de dados do tipo vetor em memória
                # Et: estrutura de dados do tipo tupla em memória
        #####
        .include "PromptTexts.asm"                  # C: contém strings de interface com
        o usuário
        preenchimento1: .word 0x2d                  # C: código ASCII '-'
        char_tiro_certeiro: .word 0x78                      # C: código ASCII 'x'
        char_tiro_nagua: .word 0x6f                 # C: código ASCII 'o'
        inicio_maiusculas: .word 0x41               # C: caracter inicial da
        representação dos navios 'A'
        ordem: .word 10                                     # C: ordem da matriz para uso
        nas funções de 'MatrixFunctions.asm'
        vetor_xy: .word 0, 0                        # Et: vetor de coordenadas para
        input do usuário
        vetor_ultimo_tiro: .word 72,72                      # Et: vetor de coordenadas do
        último tiro disparado
        matriz_posicoes: .space 100                 # Em: matriz de caracteres de navios
        matriz_tiros: .space 100                    # Em: matriz de caracteres de tiros
```

```
#matriz_embarcacoes: .space 400
string_embarcacoes: .asciz "3\n1 5 0 0\n0 3 2 2\n1 1 3 4"        # C: string
representando as embarcações
string_embarcacoes2: .asciz "3\n1 5 0 0\n0 3 2 2\n1 1 3 4"              # C:
string representando as embarcações
matriz_embarcacoes: .word 3, 1, 5, 0, 0, 0, 3, 2, 2, 1, 1, 3, 4  # C: matriz
com as embarcações
vetor_tiros_acertados: .space 100          # Ev: vetor com os tiros acertados
pelo usuário
vetor_tiros_errados: .space 100            # Ev: vetor com os tiros errados
pelo usuário
buffer: .word '\0'                         # V: espaço temporário para saída de
processamento (buffer de output)
buffer_pointer: .space 400                 # V: buffer de ponteiros para
strings
buffer_words: .space 100                   # V: buffer de words


       .text
main:
       la a2, string_embarcacoes           # carrega endereço do string de
embarcações
       la a3, matriz_embarcacoes           # carrega endereço da matriz de
embarcações
       la a4, matriz_posicoes                      # carrega endereço da matriz
de caracteres de posições
       la a5, matriz_tiros                 # carrega endereço da matriz de
caracteres de tiros
       jal inicio_do_jogo                  # inicializa o conteúdo das matrizes
loop_jogo:                                 # loop de interação principal do
jogo
       la a0, separador_prompt             # caracteres para separação de
rodadas de output do prompt
       li a7, 4                            # carrega código ecall PrintString
       ecall                               #
       la a2, matriz_tiros                 # carrega endereço da matriz de
caracteres de tiros
       la a3, menu                         # carrega endereço do menu a ser
exibido para o usuário
       la a4, menu_footer                  # carrega endereço do rodapé
       la a5, vetor_ultimo_tiro            # carrega vetor do último tiro
       jal menu_e_tiros                    # exibe opções para o usuário
       li t0, 5                            # carrega número 5 para comparação
       beq a0, t0, fim_jogo                # caso opção digitada for 5, encerra
jogo
       li t0, 1                            # carrega número 1 para comparação
```

```
        beq a0, t0, exibe_matriz_navios     # caso opção digitada for 1, exibe a
matriz de navios
        li t0, 2                            # carrega número 2 para comparação
        beq a0, t0, input_tiro                      # caso opção digitada for 2,
solicita ao usuário coordenadas de tiro
        li t0, 3                            # carrega número 3 para comparação
        beq a0, t0, estatisticas_jogo       # caso opção digitada for 1, exibe a
matriz de navios
        li t0, 4                            # carrega número 6 para comparação
        beq a0, t0, reiniciar_jogo          # reinicializa as matrizes
        j loop_jogo                         # continua o loop do jogo
fim_jogo:
        li a7, 10
        ecall


.include "GameFunctions.asm"                # -----> Contém funções do menu
principal do jogo (da main) <-------
.include "PromptFunctions.asm"                      # Contém funções de interface
com o usuário
.include "MatrixFunctions.asm"                      # Contém funções de
manipulação de matrizes
.include "CharMatrixFunctions.asm"          # Contém funções para manipulação
das matrizes de caracteres
.include "VectorFunctions.asm"                      # Contém funções para
manipulação dos vetores de tiros
.include "StringFunctions.asm"                      # Contém funções para
manipulação de strings
.include "InsertionTests.asm"                       # x-x
```

2.Arquivo,MatrixFunctions.asm

```
#####
 # insere_matriz_embarcacoes(a4,a5,a6,a7): insere
um elemento em uma posicao da matriz
 #      a4: ponteiro do inicio da matriz (P)
 #      a5: elemento a ser armazenado (E)
 #     a6: linha da matriz (L)
 #     a7: coluna da matriz (C)
insere_matriz_embarcacoes:
       lw t1, 0(a4)                # ponteiro para
a posição inicial da matriz de embarcações
       li t2, 4                    # número de
colunas na matriz (Q)
       mul t3, a6, t2              # L * Q
       add t3, t3, a7              # L * Q + C
       li t2, 4                    # número de
bytes de deslocamento = 4 bytes
       mul t3, t3, t2              # deslocamento
= (L * Q + C) * 4
       add t1, a4, t3              # P +
deslocamento
       sw a5, 0(t1)                # armazena E em
na posição (P + deslocamento)
       ret

#####
 # busca_matriz_embarcacoes(a4,a6,a7): busca o
endereço de um elemento na matriz de embarcações
 #      a4: ponteiro do inicio da matriz (P)
 #     a6: linha da matriz (L)
 #     a7: coluna da matriz (C)
 #     -> a0: endereço do elemento
busca_matriz_embarcacoes:
       lw t1, 0(a4)                # ponteiro para
a posição inicial da matriz de embarcações
       li t2, 4                    # número de
colunas na matriz (Q)
       mul t3, a6, t2              # L * Q
       add t3, t3, a7              # L * Q + C
       li t2, 4                    # número de
bytes de deslocamento = 4 bytes
       mul t3, t3, t2              # deslocamento
= (L * Q + C) * 4
```

```
       add t1, a4, t3               # P +
deslocamento
       addi a0, t1, 0               # retorna
endereço de memória
       ret


#####
 # insere_matriz_posicoes(a4,a5,a6,a7): insere um
caracter em uma posicao da matriz
 #      a4: ponteiro do inicio da matriz (P)
 #      a5: caracter a ser armazenado (E)
 #     a6: linha da matriz (L)
 #     a7: coluna da matriz (C)
insere_matriz_posicoes:
       lw t1, 0(a4)                 # ponteiro para
a posição inicial da matriz de caracteres
       lw t2, ordem                 # número de
colunas na matriz (Q)
       mul t3, a6, t2               # L * Q
       add t3, t3, a7               # L * Q + C
       add t1, a4, t3               # P +
deslocamento
       sb a5, 0(t1)                 # armazena E em
na posição (P + deslocamento)
       ret


#####
 # busca_matriz_posicoes(a4,a6,a7): retorna um
caracter da matriz de posições
 #      a4: ponteiro do inicio da matriz (P)
 #     a6: linha da matriz (L)
 #     a7: coluna da matriz (C)
 #     -> a1: caracter da posição (L,C)
busca_matriz_posicoes:
       lw t1, 0(a4)                 # ponteiro para
a posição inicial da matriz de caracteres
       lw t2, ordem                 # número de
colunas na matriz (Q)
       mul t3, a6, t2               # L * Q
       add t3, t3, a7               # L * Q + C
       add t1, a4, t3               # P +
deslocamento
```

```
        lb a1, 0(t1)                # carrega E na
posição (P + deslocamento) no registrador de
retorno
        ret

#####
 # busca_matriz_posicoes_endereco(a4,a6,a7):
retorna um caracter da matriz de posições
 #      a4: ponteiro do inicio da matriz (P)
 #      a6: linha da matriz (L)
 #      a7: coluna da matriz (C)
 #      -> a1: caracter da posição (L,C)
busca_matriz_posicoes_endereco:
        lw t1, 0(a4)                # ponteiro para
a posição inicial da matriz de caracteres
        lw t2, ordem                # número de
colunas na matriz (Q)
        mul t3, a6, t2              # L * Q
        add t3, t3, a7              # L * Q + C
        add t1, a4, t3              # P +
deslocamento
        lb a1, 0(t1)                # carrega E na
posição (P + deslocamento) no registrador de
retorno
        ret
```

3. Arquivo, CharMatrixFunctions.asm.

```
#####
         # gera_matriz_vazia(a3): preenche matriz de caracteres
         #      a3: ponteiro para matriz de caracteres
        gera_matriz_vazia:
               addi sp, sp, -4                    # adiciona espaço na pilha
               sw ra, 0(sp)                # armazena endereço de retorno na pilha
               addi a4, a3, 0              # copia o endereço da matriz para A4
               lw a5, preenchimento1       # caracter vazio
               lw s1, ordem                # ordem da matriz
               li a6, 0                    # contador de linha
               li a7, 0                    # contador de coluna
        loop_vazio:
               jal insere_matriz_posicoes  # insere elemento na matriz de posições
               addi a7, a7, 1              # incrementa índice de coluna
```

```
        bne a7, s1, loop_vazio              # caso não tenha chegado até a
última coluna, repetir
        srli a7, a7, 15                     # zera o índice de coluna
        addi a6, a6, 1              # incrementa índice de linha
        bne a6, s1, loop_vazio              # caso não tenha chegado até a
última linha, repetir
        lw ra, 0(sp)                # retorna o valor de RA salvo na pilha de
execução
        addi sp, sp, 4              # retorna para a posição anterior da pilha
        ret

#####
 # imprime_matriz_posicoes(a3, a4): preenche matriz de caracteres
 #      a3: ponteiro para matriz de caracteres
 #     a4: cabeçalho da matriz
imprime_matriz_posicoes:
        addi sp, sp, -8                     # adiciona espaço na pilha
        sw ra, 0(sp)                # armazena endereço de retorno na pilha
        addi a0, a4, 0                      # carrega cabecalho da matriz de
navios
        li a7, 4                    # carrega ecall PrintString
        ecall                       #
        addi a4, a3, 0              # copia o endereço da matriz para A4
        li s0, 0x20                 # caracter de espaço
        lw s1, ordem                # ordem da matriz
        li a6, 0                    # contador de linha
        addi a0, a6, 0              # carrega indice da linha
        li a7, 1                    # carrega valor de ecall PrintInt
        ecall                       #
        li a7, 0                    # contador de coluna
loop_vazia_linha:
        jal busca_matriz_posicoes   # insere elemento na matriz de posições
        sw a7, 4(sp)                # salva o valor de a7(índice de coluna) na
pilha de execução para ecall
        addi a0, s0, 0              # carrega valor ASCII de espaço
        li a7, 11                   # carrega ecall PrintChar
        ecall                       #
        addi a0, a1, 0              # carrega elemento da matriz para impressão
        ecall                       #
        lw a7, 4(sp)                # retorna valor de A7 salvo na pilha
        addi a7, a7, 1              # incrementa índice de coluna
        bne a7, s1, loop_vazia_linha # caso não tenha chegado até a última
coluna, repetir

        li a0, 10                   # carrega valor ASCII de Nova Linha
        li a7, 11                   # carrega valor de ecall PrintChar
```

```
        ecall                       #
        srli a7, a7, 15                     # zera o índice de coluna
        addi a6, a6, 1              # incrementa índice de linha
        addi a0, a6, 0              # carrega indice da linha
        li a7, 1                    # carrega valor de ecall PrintInt
        ecall                       #
        li a7, 0                    # zera o valor do contador de colunas
        bne a6, s1, loop_vazia_linha # caso não tenha chegado até a última
linha, repetir
        la a0, rodape_matriz_navios
        li a7, 4
        ecall
        lw ra, 0(sp)                # retorna o valor de RA salvo na pilha de
execução
        addi sp, sp, 8              # retorna para a posição anterior da pilha
        ret


#####
 # gera_matriz_posicoes(a2,a3): preenche matriz de caracteres
 #      a2: ponteiro para matriz de embarcacoes
 #      a3: ponteiro para matriz de caracteres
gera_matriz_posicoes:
        addi sp, sp, -4                     # adiciona espaço na pilha
        sw ra, 0(sp)                # armazena endereço de retorno na pilha
        lw s0, 0(a2)                # número de embarcações
        lw a4, inicio_maiusculas    # número ASCII inicial das maiúsculas
        li s2, 0                    # contador de embarcações
        addi sp, sp, -12            # salvar variáveis
        addi a2, a2, 4              # avança 1 word (4 bytes) no endereço de
memória
loop_embarcacoes:
        sw a2, 0(sp)                # salva a2 na pilha
        sw a3, 4(sp)                # salva a3 na pilha
        sw a4, 8(sp)                # salva a4 na pilha
        jal insere_embarcacao_posicao# chama função
        lw a2, 0(sp)                # recupera a2 da pilha
        lw a3, 4(sp)                # recupera a3 da pilha
        lw a4, 8(sp)                # recupera a4 da pilha
        addi s2, s2, 1              # incrementa contador de embarcações
        addi a4, a4, 1              # incrementa número de letra para
representar a próxima embarcação
        addi a2, a2, 16                     # avança 1 word (4 bytes) no
endereço de memória
        bne s0, s2, loop_embarcacoes # caso ainda existam embarcações a inserir,
atualizar
```

```
        addi sp, sp, 12             # retorna o ponteiro da pilha para RA
        lw ra, 0(sp)                # retorna o valor de RA salvo na pilha de
execução
        addi sp, sp, 4              # retorna para a posição anterior da pilha
        ret

#####
 # insere_embarcacao_posicao(a2,a3,a4): insere embarcação na matriz de posições
 #      a2: ponteiro para matriz de embarcacoes
 #     a3: ponteiro para matriz de caracteres
 #      a4: caracter representando o navio
insere_embarcacao_posicao:
        addi sp, sp, -4                     # adiciona espaço na pilha
        sw ra, 0(sp)                # armazena endereço de retorno na pilha
        lw s3, 0(a2)                # carrega orientação da embarcação
        lw s4, 4(a2)                # carrega comprimento da embarcação
        lw a6, 8(a2)                # carrega número da linha
        lw a7, 12(a2)               # carrega número da coluna
        li s5, 1                    # contador de itens
        addi a5, a4, 0              # copia registrador para chamada da função
        addi a4, a3, 0              # copia registrador para chamada da função
        bnez s3, insere_embarcacao_posicao_vertical # se orientação não for 0
(vertical), pula
        beqz s3, insere_embarcacao_posicao_horizontal      # se orientação for 0
(horizontal), pula
insere_embarcacao_posicao_vertical:
        jal insere_matriz_posicoes   # chama função para inserir na matriz de
posições
        addi s5, s5, 1              # incrementa contador de comprimento
        addi a6, a6, 1              # incrementa posição da linha
        ble s5, s4, insere_embarcacao_posicao_vertical # enquanto não tiver o
comprimento, continua colocando chars
        j fim_insere_posicao         # pula para o final da função
insere_embarcacao_posicao_horizontal:
        jal insere_matriz_posicoes   # função no arquivo "MatrixFunctions.asm",
insere caracter representando o navio
        addi s5, s5, 1              # incrementa contador de comprimento
        addi a7, a7, 1              # incrementa posição da coluna
        ble s5, s4, insere_embarcacao_posicao_horizontal # enquanto não tiver o
comprimento, continua colocando chars
fim_insere_posicao:
        lw ra, 0(sp)                # retorna o valor de RA salvo na pilha de
execução
        addi sp, sp, 4              # retorna para a posição anterior da pilha
        ret
```

4. Arquivo, GameFunctions.asm

```
#####
         # inicio_do_jogo(a2,a3,a4,a5): inicialização do jogo
         #      a2: ponteiro para string representando a matriz de embarcações
         #      a3: ponteiro para matriz de embarcações
         #      a4: ponteiro para matriz de caracteres de posições
         #      a5: ponteiro para matriz de caracteres de tiros
        inicio_do_jogo:
               addi sp, sp, -20
               sw ra, 0(sp)
               sw a2, 4(sp)                        # salva valor de A2 na pilha para
        chamada de função
               sw a3, 8(sp)                        # salva valor de A3 na pilha para
        chamada de função
               sw a4, 12(sp)                       # salva valor de A4 na pilha para
        chamada de função
               sw a5, 16(sp)                       # salva valor de A5 na pilha para
        chamada de função
               #TODO conversão da string de navios para a matriz de embarcações
               lw a3, 12(sp)                       # carrega ponteiro da matriz de
        caracteres de posições
               jal gera_matriz_vazia               # insere caracteres de preenchimento
        na matriz
               lw a3, 16(sp)                       # carrega ponteiro da matriz de
        caracteres de tiros
               jal gera_matriz_vazia               # insere caracteres de preenchimento
        na matriz
               lw a2, 8(sp)                        # carrega ponteiro da matriz de
        embarcações
               lw a3, 12(sp)                       # carrega ponteiro da matriz de
        caracteres de navios
               jal gera_matriz_posicoes            # insere navios na matriz de navios
               lw ra, 0(sp)                        # retorna o valor de RA salvo na
        pilha de execução
               addi sp, sp, 20                             # retorna para a posição
        anterior da pilha
               ret

        #####
         # exibe_matriz_navios(): exibe a matriz de navios
        exibe_matriz_navios:
               la a3, matriz_posicoes              #
               la a4, cabecalho_matriz_navios              # carrega label do arquivo
        "PromptTexts.asm" no reg. A4
```

```
        jal imprime_matriz_posicoes         # função do arquivo
"CharMatrixFunctions.asm"
        la a0, prompt_continuar             # carrega prompt de pausa
        li a7, 4                            # carrega código ecall PrintString
        ecall                               #
        li a7, 8                            # carrega código ecall
        ecall                               #
        j loop_jogo                         # retorna ao loop principal do jogo


#####
 # input_tiro(): realiza tiro na matriz
input_tiro:
        la a2, vetor_xy                             # carrega ponteiro do vetor
de coordenadas de input
        jal solicita_input_tiro             # função do arquivo
"PromptFunctions.asm", retorna em a0
        addi a2, a0, 0                              # carrega ponteiro do vetor
de coordenadas de input
        la a3, matriz_posicoes                      # carrega ponteiro da matriz
de caracteres de navios
        la a4, matriz_tiros                 # carrega ponteiro da matriz de
caracteres de tiros
        la t0, vetor_ultimo_tiro            # carrega ponteiro do vetor de
último tiro
        lw t1, 0(a2)                        #
        sw t1, 0(t0)                        #
        lw t1, 4(a2)                        #
        sw t1, 4(t0)                        #
        jal testa_e_marca_se_acertou        # testa se o tiro acertou algum
navio
        j loop_jogo                         # retorna ao loop de jogo


#####
 # testa_e_marca_se_acertou(a2,a3,a4): realiza tiro na matriz
 #      a2: ponteiro do vetor de coordenadas
 #     a3: ponteiro da matriz de caracteres de navios
 #     a4: ponteiro da matriz de caracteres de tiros
testa_e_marca_se_acertou:
        addi sp, sp, -16                    # adiciona espaço na pilha
        sw ra, 0(sp)                        # armazena endereço de retorno na
pilha
        sw a3, 4(sp)                        # armazena A3 na pilha (ponteiro
matriz de navios)
```

```
        sw a4, 8(sp)                        # armazena A4 na pilha (ponteiro
matriz de tiro)
        sw a2, 12(sp)                       # armazena A2 na pilha (ponteiro
vetor de input)
        addi a4, a3, 0                      # carrega A4 com ponteiro de matriz
de caracteres de navios
        lw a6, 0(a2)                        # carrega A6 com coordenada de linha
para chamada de função
        lw a7, 4(a2)                        # carrega A7 com coordenada de
coluna para chamada de função
        jal busca_matriz_posicoes           # função do arquivo
"MatrixFunctions.asm", retorna valor em A1
        lw t0, preenchimento1               # carrega o caracter '-' de
preenchimento
        beq t0, a1, errou_o_tiro            # caso o tiro seja em um local com
caracter de preenchimento, errou
        li t1, 0x5A                         # carrega código ASCII da letra Z
        ble a1, t1, acertou_o_tiro          # caso o caracter da matriz esteja
no interfalo de letras maiusculas, acertou
        j fim_teste_tiro
acertou_o_tiro:
        la t2, contabiliza_tiro             # carrega endereço
        jalr t2                             # chama a função contabiliza tiro
        lw a2, 12(sp)                       # carrega ponteiro da pilha
        addi a5, a2, 0                      # carrega ponteiro do vetor de
coordenadas
        la a4, vetor_tiros_acertados        # carrega vetor de tiros acertados
para chamada da função
        jal insere_tupla2_vetor             # função do arquivo
"VectorFunctions.asm"
        lw a2, 12(sp)                       # carrega ponteiro do vetor de
coordenadas
        lw a4, 4(sp)                        # carrega ponteiro da matriz de
caracteres em A4
        lw a6, 0(a2)                        # carrega indice da linha
        lw a7, 4(a2)                        # carrega indice da coluna
        addi a5, a1, 32                             # adiciona valor 31 para
tornar letra minúscula
        jal insere_matriz_posicoes          # função do arquivo
"MatrixFunctions.asm"
        lw a4, 8(sp)                        # carrega valor de A4 (ponteiro
matriz de tiro)
        lw a5, char_tiro_certeiro           # carrega letra 'x'
        jal insere_matriz_posicoes          # função do arquivo
"MatrixFunctions.asm"
        j fim_teste_tiro
```

```
errou_o_tiro:
	addi a5, a2, 0			# carrega ponteiro do vetor de
coordenadas
	la a4, vetor_tiros_errados	# carrega vetor de tiros acertados
para chamada da função
	jal insere_tupla2_vetor		# função do arquivo
"VectorFunctions.asm"
	lw a4, 4(sp)			# carrega ponteiro da matriz de
caracteres em A4
	lw a5, char_tiro_nagua			# carrega letra 'o'
	jal insere_matriz_posicoes	# função do arquivo
"MatrixFunctions.asm"
	lw a4, 8(sp)			# carrega valor de A4 (ponteiro
matriz de tiro)
	jal insere_matriz_posicoes	# função do arquivo
"MatrixFunctions.asm"
fim_teste_tiro:
	lw ra, 0(sp)			# retorna o valor de RA salvo na
pilha de execução
	addi sp, sp, 4			# retorna para a posição anterior da
pilha
	ret

#####
 # contabiliza_tiro(a1): mostra as estatisticas para o usuário
	# a1: caracter simbolizando a embarcação
contabiliza_tiro:
	addi sp, sp, -4				# adiciona espaço na pilha
	sw ra, 0(sp)			# armazena endereço de retorno na
pilha
	addi t2, a1, -65		# subtrai o código ASCII para obter
o índice da embarcação
	la a4, matriz_embarcacoes	# carrega endereço da matriz de
embarcações
	addi a6, t2, 0			# carrega indice de linha para
chamada
	li a7, 2			# carrega indice de coluna para
chamada
	jal busca_matriz_embarcacoes	# função no arquivo
"MatrixFunctions.asm", retorna ponteiro em A0
	lw t3, 0(a0)			# carrega "comprimento" da
embarcação
	addi t3, t3, -1				# subtrai 1 do comprimento da
embarcação
	sw t3, 0(a0)			# salva novo "comprimento"
	lw ra, 0(sp)			# recupera RA da pilha
```

```
        addi sp, sp, 4                      # restaura ponteiro da pilha
        ret

#####
 # conta_afundados(): conta o número de navios afundados
        # -> a0: número de navios afundados
conta_afundados:
        ebreak
        addi sp, sp, -4                             # adiciona espaço na pilha
        sw ra, 0(sp)                        # armazena endereço de retorno na
pilha
        la t0, matriz_embarcacoes           # carrega endereço da matriz de
embarcacoes
        lw s2, 0(t0)                        # carrega contador de elementos da
matriz de embarcações
        li s0, 0                            # contador local
        li s1, 0                            # contador embarcações
        # addi t0, t0, 4                    # avança para início da matriz de
embarcações
        addi a4, t0, 0                              # carrega endereço da matriz
de embarcações
        li a7, 2                            # coluna correspondente ao
comprimento da embarcação
loop_afundados:
        bge s0, s2, fim_loop_afundados              # se ainda não percorreu
todas as linhas, repetir
        addi a6, s0, 0                      # carrega número da linha
        addi s0, s0, 1                      # incrementa número da linha
        jal busca_matriz_embarcacoes        #
        lw t6, 0(a0)                        #
        bnez t6, loop_afundados             # se comprimento não for zero, a
embarcação não está afundada
        addi s1, s1, 1                      # incrementa contador de embarcações
afundadas
        j loop_afundados
fim_loop_afundados:
        #addi s1, s1, 48                            # adiciona 48 para completar
o código ASCII
        addi a0, s1, 0                      # carrega valor no registrador de
retorno
        lw ra, 0(sp)                        # retorna o valor de RA salvo na
pilha de execução
        addi sp, sp, 4                      # retorna para a posição anterior da
pilha
        ret
```

```
#####
 # estatisticas_jogo(): mostra as estatisticas para o usuário
estatisticas_jogo:
	addi sp, sp, -4				# reserva espaço na pilha de
execução
	sw ra, 0(sp)				# salva RA na pilha de execução
	lw s0, vetor_tiros_acertados		# recupera total de tiros acertados
	lw s1, vetor_tiros_errados		# recupera total de tiros errados
	add s2, s0, s1				# total = acertados + errados
	#
	la a2, separador_prompt			#
	jal adiciona_ao_buffer_pointers		#
	# msg total
	la a2, divisor				# carrega endereço do divisor
	jal adiciona_ao_buffer_pointers		# adiciona ao buffer de ponteiros
	la a2, msg_qtd_total_tiros		# carrega ponteiro da mensagem
	jal adiciona_ao_buffer_pointers		# adiciona ao buffer de ponteiros
	# nro total
	addi a2, s2, 0				# carrega nro tiros acertados
	jal adiciona_int_ao_buffer_words	# adiciona ao buffer de conteudos
	addi a2, a0, 0				#
	jal adiciona_ao_buffer_pointers		# adiciona ao buffer de ponteiros
	# msg acertados
	la a2, divisor				# carrega endereço do divisor
	jal adiciona_ao_buffer_pointers		# adiciona ao buffer de ponteiros
	la a2, msg_qtd_tiros_certeiros		# carrega ponteiro da
mensagem
	jal adiciona_ao_buffer_pointers		# adiciona ao buffer de ponteiros
	# nro acertados
	addi a2, s0, 0				# carrega nro tiros acertados
	jal adiciona_int_ao_buffer_words	# adiciona ao buffer de conteudos
	addi a2, a0, 0				#
	jal adiciona_ao_buffer_pointers		# adiciona ao buffer de ponteiros
	# msg errados
	la a2, divisor				# carrega endereço do divisor
	jal adiciona_ao_buffer_pointers		# adiciona ao buffer de ponteiros
	la a2, msg_qtd_tiros_nagua		# carrega ponteiro da mensagem
	jal adiciona_ao_buffer_pointers		# adiciona ao buffer de ponteiros
	# nro errados
	addi a2, s1, 0				# carrega nro tiros acertados
	jal adiciona_int_ao_buffer_words	# adiciona ao buffer de conteudos
	addi a2, a0, 0				# carrega para a variável de entrada
	jal adiciona_ao_buffer_pointers		# adiciona ao buffer de ponteiros
	# msg afundados
	la a2, divisor				# carrega endereço do divisor
```

```
        jal adiciona_ao_buffer_pointers     # adiciona ao buffer de ponteiros
        la a2, msg_qtd_afundados            # carrega endereço mensagem navios
afundados
        jal adiciona_ao_buffer_pointers     # adiciona ao buffer de ponteiros
        # nro afundados
        jal conta_afundados                 # conta nro de afundados
        addi a2, a0, 0                      # carrega para a variável de entrada
        jal adiciona_int_ao_buffer_words    # adiciona ao buffer de conteudos
        addi a2, a0, 0                      # carrega para a variável de entrada
        jal adiciona_ao_buffer_pointers     # adiciona ao buffer de ponteiros
        # print buffer
        jal print_buffer_pointers           # função do arquivo
"MatrixFunctions.asm"
        # limpa bufferes
        jal limpa_buffer_pointers           # limpa o buffer_pointers
        jal limpa_buffer_words                      # limpa o buffer_words
        # fim
        lw ra, 0(sp)                        # recupera o RA da pilha
        addi sp, sp, 4                      # reseta a pilha
        ret

#####
 # reiniciar_jogo(a2,a3): reinicializa as matrizes do jogo
reiniciar_jogo:
        la a2, string_embarcacoes           # carrega endereço do string de
embarcações
        la a3, matriz_embarcacoes           # carrega endereço da matriz de
embarcações
        la a4, matriz_posicoes                      # carrega endereço da matriz
de caracteres de posições
        la a5, matriz_tiros                 # carrega endereço da matriz de
caracteres de tiros
        jal inicio_do_jogo                  # inicializa o conteúdo das matrizes
        j loop_jogo


#####
 # insere_embarcacoes(a2,a3): insere a embarcação na matriz
 #      a2: ponteiro do vetor representando o navio
 #     a3: ponteiro do início da matriz
 #       0(a2): disposicao do navio (0 horizontal) (1 vertical)
 #       4(a2): comprimento do navio
 #       8(a2): linha inicial do navio
 #      12(a2): coluna inicial do navio
insere_embarcacoes:
        addi sp, sp, -4                     #
```

```
        sw ra, 0(sp)                        #
        jal testeAB_vetor                   #
        bnez a0, fim                        #
        #jal teste_sobreposicao             #
        addi a4, a3, 0                      #
        addi a4, a4, 4                      #
        lw a6, 0(a3)                        # números de linhas na matriz
        addi s0, a2, 0                      # contador de elementos do vetor
        li t2, 0                            # contador de colunas
        addi a7, t2, 0                      #
        li t4, 4                            # número de colunas da matriz de
navios
loop_insere:
        lw a5, 0(s0)
        jal insere_matriz_embarcacoes
        addi s0, s0, 4
        addi a7, a7, 1
        bne a7, t4, loop_insere
        addi a6, a6, 1
        sw a6, 0(a3)
fim:
        lw ra, 0(sp)
        addi sp, sp, 4
        ret
```

5. Arquivo, InsertionTests.asm

```
#####
        # testeAB_vetor(a2): testa caso A e B de inclusão na matriz de navios
        #  Caso A: As coordenadas são maiores que o permitido
        #  Caso B: A coordenada inicial + o comprimento do navio é maior que o tamanho
da matriz
        #      a2: ponteiro do vetor representando o navio
        #     -> a0: codigo de erro
testeAB_vetor:
        li a0, 10                   # carrega ordem da matriz
        li t0, 1                    #
        lw t1, 0(a2)                #
        lw t2, 4(a2)                #
        lw t3, 8(a2)                #
        srli t6, t6, 16                     #
        addi t6, t6, 1              #
```

```
        bgt t3, a0, fim_erro         #
        lw t3, 12(a2)                #
        bgt t3, a0, fim_erro         #
        beqz t1, testa_horizontal    #
testa_vertical:
        lw t4, 8(a2)                 #
        j testa_estouro
testa_horizontal:
        lw t4, 12(a2)
testa_estouro:
        add t5, t4, t2
        addi t6, t6, 2
        bge t5, a0, fim_erro
fim_normal:
        srli a0, a0, 31
        addi a0, a0, 0
        ret
fim_erro:
        addi a0, t6, 0
        ret


#####
 # teste_sobreposicao(a2,a3): testa a sobreposição entre a embarcação a ser
inserida e as já existentes
 #      a2: ponteiro do vetor representando o navio
 #     a3: ponteiro do início da matriz
 #     -> a0: código de erro ou "zero" = exito
teste_sobreposicao:
        addi sp, sp, -4
        sw ra, 0(sp)
        jal gerar_vetor_logico
        addi s0, a0, 0

        lw ra, 0(sp)
        addi sp, sp, 4
        ret


#####
 # gerar_vetor_logico(a2): gera vetor de bits representando a embarcação
 #  O MSB do vetor de bits é a posição de início (coordenada inicial vertical ou
horizontal)
 #      a2: ponteiro do vetor representando a embarcação
 #     -> a0: vetor de bits
gerar_vetor_logico:
```

```
        li t0, 1                    # contador do loop de adição de bits
        li t1, 1                    # registrador vetor de bits
        lw t2, 4(a2)                # carrega comprimento da embarcação
loop_gerador:
        slli t1, t1, 1              # rola à esquerda para abrir espaço para
novo bit à direita
        addi t1, t1, 1              # adiciona bit ao vetor
        addi t0, t0, 1              # contador ++
        bne t0, t2, loop_gerador    # caso vetor não possua o comprimento da
embarcação, continua adição
        li t2, 10                   # ordem da matriz
        lw t3, 4(a2)                # tamanho da embarcação
        lw t4, 0(a2)                # orientação da embarcação
        beqz t4, ajusta_horizontal
        bnez t4, ajusta_vertical
ajusta_horizontal:
        lw t4, 8(a2)                # usa coordenada horizontal como posição
inicial (MSB)
        j alinha_vetor
ajusta_vertical:
        lw t4, 12(a2)               # usa coordenada vertical como posição
inicial (MSB)
alinha_vetor:                       # alinha os bits adicionados no loop com a
posição inicial
        sub t5, t2, t3              # (ordem) - (comprimento)
        sub t5, t5, t4                      # (ordem) - (comprimento) - MSB
        sll a0, t1, t5              # rola todos os bits do vetor até a posição
inicial, à esquerda
        ret
```

6. Arquivo, PromptFunctions.asm

```
#####
 # menu_e_tiros(a2,a3,a4): exibe menu
        # a2: ponteiro para matriz de caracteres
        # a3: ponteiro para cabeçalho do menu
        # a4: ponteiro para rodapé do menu
        # a5: ponteiro para vetor de coordenadas
        # -> a0: opção escolhida
menu_e_tiros:
        addi sp, sp, -12            # reserva espaço na pilha para 3 words (3 *
4)
        sw ra, 0(sp)                # salva RA na pilha
```

```
        sw a3, 4(sp)                # salva A3 na pilha
        sw a4, 8(sp)                # salva A4 na pilha
        la a4, cabecalho_matriz_tiro # carrega cabeçalho da matriz de tiros
(definido em "Main.asm")
        addi a3, a2, 0              # carrega ponteiro da matriz de caracteres
de tiros
        jal imprime_matriz_posicoes  # função do arquivo
"CharMatrixFunctions.asm"
        jal divisor_curto           # printa divisor
        la a0, msg_ultimo_tiro              #
        li a7, 4                    #
        ecall                       #
        la a2, vetor_ultimo_tiro    # carrega ponteiro do último tiro realizado
        jal string_tupla2           # função do arquivo "StringFunctions.asm" -
Gera a string do último tiro
        la a0, buffer               # carrega ponteiro para buffer
        ecall                       #
        jal divisor_n               # printa divisor
        lw a3, 4(sp)                #
        addi a0, a3, 0                      #
        li a7, 4                    #
        ecall                       #
        lw a4, 8(sp)                #
        addi a0, a4, 0              #
        ecall                       #
        li a7, 5                    #
        ecall                       #
        lw ra, 0(sp)                #
        addi sp, sp, 12                     #
        ret


#####
 # divisor_curto(): printa divisor curto de output
divisor_curto:
        la a0, divisor              # carrega ponteiro para string "divisor"
        li a7, 4                    # carrega código ecall PrintString
        ecall                       #
        ret
#####
 # divisor_n(): printa divisor com quebra de linha
divisor_n:
        la a0, menu_footer          # carrega ponteiro para string
"menu_footer"
        li a7, 4                    # carrega código ecall PrintString
        ecall                       #
```

```
        ret


#####
 # solicita_input_tiro(a2): solicita ao usuário o input de coordenadas do tiro
        # a2: ponteiro para vetor de coordenadas (x,y)
solicita_input_tiro:
        la a0, digite_linha_tiro     #
        li a7, 4                     #
        ecall                        #
        li a7, 5                     #
        ecall                        #
        sw a0, 0(a2)                 #
        la a0, digite_coluna_tiro    #
        li a7, 4                     #
        ecall                        #
        li a7, 5                     #
        ecall                        #
        sw a0, 4(a2)                 #
        addi a0, a2, 0               #
        ret




#####
 # input_navio(a2): prompt de input exibe menu
input_navio:
#       la t0, vetor_v
        la a0, digite_orientacao
        li a7, 4
        ecall
        li a7, 5
        ecall
        sw a0, 0(t0)
        addi t0, t0, 4

        la a0, digite_comprimento
        li a7, 4
        ecall
        li a7, 5
        ecall
        sw a0, 0(t0)
        addi t0, t0, 4

        la a0, digite_linha
```

```
        li a7, 4
        ecall
        li a7, 5
        ecall
        sw a0, 0(t0)
        addi t0, t0, 4

        la a0, digite_coluna
        li a7, 4
        ecall
        li a7, 5
        ecall
        sw a0, 0(t0)

        ret


###
 # mostra_erro(a2): exibe uma mensagem de erro para o usuário conforme o código
#mostra_erro:
```

7. Arquivo, PromptTexts.asm

```
###
        # Inclusão dos prompts para o usuário
buffer_vazio: .ascii "\0\0\0\0"
digite_orientacao: .asciz "\nEscolha a orientação: 0 - horizontal ; 1 - vertical
"
digite_comprimento: .asciz "\nEscolha o tamanho do navio: "
digite_linha: .asciz "\nDigite a coordenada vertical do navio: "
digite_coluna: .asciz "\nDigite a coordenada horizontal do navio: "
msg_erro_invalida: .asciz "A posição da embarcação é inválida.\n"
msg_erro_saliente: .asciz "A embarcação está saliente à dimensão da matriz.\n"
digite_linha_tiro: .asciz "\nDigite a linha onde será o disparo: "
digite_coluna_tiro: .asciz "\nDigite a coluna onde será o disparo: "
msg_ultimo_tiro: .asciz "\n Último tiro: "
 # Prompts do menu
menu: .asciz "Escolha a próxima ação:\n1 - Exibir matriz de navios\n2 - Efetuar
tiro\n3 - Exibir estatisticas\n4 - Reiniciar jogo\n5 - Sair"
menu_footer: .asciz "\n--------------------------------\n"
divisor: .asciz "\n--------------------------------"
 # Montagem da matriz
cabecalho_matriz_navios: .asciz " _____________________ \n|   MATRIZ DE NAVIOS
|\n|_____________________|\n  0 1 2 3 4 5 6 7 8 9\n"
```

```
cabecalho_matriz_tiro: .asciz " _____ \n|   MATRIZ DE TIRO
|\n|_____|\n  0 1 2 3 4 5 6 7 8 9\n"
rodape_matriz_navios: .asciz "10101010101010101010\n"
separador_prompt: .asciz
"\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n"
prompt_continuar: .asciz "\nSenta o dedo no Enter pra continuar..."
# Estatísticas:
msg_qtd_total_tiros: .asciz "\nTotal de tiros: "
msg_qtd_tiros_certeiros: .asciz "\nTiros certeiros ao alvo: "
msg_qtd_tiros_nagua: .asciz "\nTiros n'água: "
msg_qtd_afundados: .asciz "\nNavios afundados: "
```

8. Arquivo, StringFunctions.asm

```
#####
 # string_tupla2(a2): gera a representação em string de uma tupla de 2
caracteres
       # a2: ponteiro para tupla (vetor de coordenadas (x,y))
string_tupla2:
       la t0, buffer               # carrega ponteiro do buffer (definido em
"Main.asm")
       li t2, 0                    # zera registrador de composição de tupla
       lw t3, 4(a2)                # carrega 1º elemento da tupla
       addi t3, t3, 48                     # adiciona 48 para completar o
código ASCII
       add t2, t2, t3              # adiciona caracter à string - SEGUNDO
       slli t2, t2, 8              # desloca 1 byte à esquerda
       addi t2, t2, 44                     # adiciona o caracter de virgula -
TERCEIRO
       slli t2, t2, 8              # desloca 1 byte à esquerda
       lw t3, 0(a2)                # carrega 1º elemento da tupla
       addi t3, t3, 48                     # adiciona 48 para completar o
código ASCII
       add t2, t2, t3              # adiciona caracter à string - QUARTO
       sw t2, 0(t0)                # descarrega para buffer
       ret

#####
 # adiciona_ao_buffer_pointers(a2): adiciona o ponteiro do conteúdo ao buffer de
ponteiros
       # a2: ponteiro para o conteúdo
adiciona_ao_buffer_pointers:
       la t0, buffer_pointer       # carrega endereço do buffer
```

```
        lw t1, 0(t0)                # carrega primeiro item do buffer
(contador)
        slli t2, t1, 2              # multiplica por 4
        add t2, t2, t0              # adiciona ao endereço de memória
        sw a2, 0(t2)                # armazena ponteiro
        addi t1, t1, 1              # incrementa contador do buffer
        sw t1, 0(t0)                # armazena conteúdo da variável
        ret

#####
 # limpa_buffer_pointers(a2): remove o conteúdo do buffer de caracteres
        # a2: ponteiro para o conteúdo
limpa_buffer_pointers:
        la t0, buffer_pointer               # carrega endereço de memória do
buffer
        lw t1, 0(t0)                # carrega valor do contador do buffer
        li t2, 0                    # contador de ponteiros
        lw t3, buffer_vazio         #
loop_limpa_buffer_ponteiros:
        addi t0, t0, 4              # endereço + 4
        addi t2, t2, 1              # incrementa contador de ponteiros
        sw t3, 0(t0)                # armazena string vazia no buffer
        ble t2, t1, loop_limpa_buffer_ponteiros     # se contador menor que nro
de ponteiros, continua
        la t0, buffer_pointer               # carrega endereço de memória do
buffer
        sw x0, 0(t0)                # zera valor do contador do buffer
        ret


#####
 # print_buffer_pointers(): imprime o conteúdo do buffer
print_buffer_pointers:
        addi sp, sp, -4                     # reserva espaço na pilha de
execução
        sw ra, 0(sp)                # salva RA na pilha de execução
        la t0, buffer_pointer       # carrega ponteiro do buffer
        lw t1, 0(t0)                # carrega contador do buffer de ponteiros
        li t2, 0                    # inicializa contador de loops
loop_print_bp:
        addi t0, t0, 4              # incrementa em 4 o ponteiro do buffer
        addi t2, t2, 1              # incrementa em 1 o contador do buffer
        lw a0, 0(t0)                # carrega o ponteiro do buffer
        beqz a0, fim_print_bp       # se o ponteiro for nulo, pula para o fim
        li a7, 4                    # carrega código da ecall PrintString
        ecall                       #
```

```
        ble t2, t1, loop_print_bp    # enquanto o número de ponteiros for menor
ou igual ao contador do buffer, continua
fim_print_bp:
        lw ra, 0(sp)                 # recupera o RA da pilha
        addi sp, sp, 4               # reseta a pilha
        ret


#####
 # adiciona_int_ao_buffer_words(a2): adiciona um inteiro ao buffer de conteudos
        # a2: numero inteiro
        # -> a0: endereço de armazenamento
adiciona_int_ao_buffer_words:
        la t0, buffer_words          # carrega endereço do buffer
        lw t1, 0(t0)                 # carrega contador de words
        addi t3, a2, 48                      # adiciona 48 para completar ASCII
#       slli t3, t3, 16                      # adiciona "\0\0"
        slli t2, t1, 2               # contador * 4
        addi t6, t0, 4
        add a0, t2, t6               # incrementa endereço
        sw t3, 0(a0)                 # salva int no buffer
        addi t1, t1, 1               # incrementa contador
        sw t1, 0(t0)                 # salva contador
        ret


#####
 # limpa_buffer_words(a2): remove o conteúdo do buffer de caracteres
        # a2: ponteiro para o conteúdo
limpa_buffer_words:
        la t0, buffer_words          # carrega endereço de memória do buffer
        lw t1, 0(t0)                 # carrega valor do contador do buffer
        li t2, 0                     # contador de ponteiros
        lw t3, buffer_vazio          #
loop_limpa_buffer_conteudo:
        addi t0, t0, 4               # endereço + 4
        addi t2, t2, 1               # incrementa contador de ponteiros
        sw t3, 0(t0)                 # armazena string vazia no buffer
        ble t2, t1, loop_limpa_buffer_conteudo      # se contador menor que nro
de ponteiros, continua
        la t0, buffer_words          # carrega endereço de memória do buffer
        sw x0, 0(t0)                 # zera valor do contador do buffer
        ret
```

9. Arquivo, VectorFunctions.asm

```
#####
         # insere_tupla2_vetor(a4,a5): insere uma tupla no vetor
         #      a4: ponteiro do inicio do vetor (P)
         #             primeiro elemento do vetor(F): contador de elementos do vetor
         #      a5: ponteiro do vetor de elementos a serem armazenados (E)
        insere_tupla2_vetor:
               lw t0, 0(a4)                # carrega F
               slli t0, t0, 3              # F * 2^3
               add t1, a4, t0              # P + (F * 2^3)
               addi t1, t1, 4              # P + (F * 2^3) + 4
               lw t2, 0(a5)                # carrega 1º elemento da tupla
               sw t2, 0(t1)                # salva no vetor 1º elemento da tupla
               lw t2, 4(a5)                # carrega 2º elemento da tupla
               sw t2, 4(t1)                # salva no vetor 2º elemento da tupla
               lw t0, 0(a4)                # carrega F
               addi t0, t0, 1              # F + 1
               sw t0, 0(a4)                # salva F
               ret

        #####
         # ultimo_elemento_vetor(a4): insere uma tupla no vetor
         #      a4: ponteiro do inicio do vetor (P)
         #             primeiro elemento do vetor(F): contador de elementos do vetor
         #      -> a0: ponteiro para o ultimo elemento
        ultimo_elemento_vetor:
               lw t0, 0(a4)                # carrega F
               slli t0, t0, 3              # F * 2^3
               add t1, a4, t0              # P + (F * 2^3)
               addi t1, t1, -4                     # P + (F * 2^3) - 8 (para apontar
        para a última tupla armazenada)
               addi a0, t1, 0              # salva t1 no registrador de retorno a0
               ret
```