

## **Guilherme Cardozo (1711100024) e Gustavo G. Zanella (1621101059)**

Este relatório contém informações precisas para o desenvolvimento e implementação do nosso programa do jogo batalha naval utilizando o conjunto de instruções do processador **RISC-V RV32IM** com base em estudos e pesquisas sobre o assunto e etc. O desenvolvimento do programa foi feito com todo o aprendizado dos componentes da dupla, e também conhecimentos de fora, tudo resultou em um circuito que executa corretamente o que foi proposto. Este trabalho foi muito importante para o conhecimento dos integrantes da dupla.

### **1. Problema**

Batalha naval é basicamente um jogo de tabuleiro para 2 jogadores, no qual os mesmos terão que adivinhar em que quadrados estão os navios/barcos do adversário. O principal objetivo é derrubar os barcos do adversário, vence quem derrubar primeiro todos os navios do oponente.

O trabalho a ser implementado é o controle do jogo de batalha naval em uma matriz 10x10. 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.

### **2. Solução**

Primeiramente, criamos uma matriz de 400 de espaço na parte .data, referente ao tamanho da matriz 10x10 (10x10x4bytes = 400). Então, após iniciar as variáveis necessárias no main (FIGURA 1), usamos a instrução jump para a função “zera\_matriz”, exibida na FIGURA 2.

**FIGURA 1 - MAIN**

```
main:
    la    s0, matriz          # endereço inicial da matriz carregada em s0
    la    s1, navios          # endereço inicial da string navios carregada em s1
    la    s2, numeros         # endereço inicial da string numeros carregada em s2
    lw    t0, (s0)            # carrega o primeiro valor da matriz s0 em t0
    lb    t1, (s1)            # carrega a primeira letra da string navios s1 em t1
    lb    t2, (s2)            # carrega o primeiro número da string navios s2 em t2
    addi   a1, a1, 100         # tamanho total da matriz
    addi   a2, zero, 0         # auxiliar para o for
    addi   a3, zero, 0         # auxiliar para o for
    addi   a4, zero, 10        # auxiliar para imprimir a matriz e quebrar linha
    addi   a5, zero, 0         # auxiliar para saber qual número da string navios
    addi   a6, zero, 8         # valor de cada navio na string navios
    addi   a7, zero, 2         # valor para saber se o navios será inserido na vertical ou horizontal
    addi   s4, zero, 4         # valor para saber o tamanho do navio que será inserido
    addi   s5, zero, 0         # registrador responsável por armazenar o tamanho do navio que será inserido
    addi   s6, zero, 6         # valor para saber a linha que o navio será inserido
    addi   s7, zero, 0         # registrador responsável por armazenar a linha que o navio que será inserido
    addi   s8, zero, 8         # valor para saber a coluna que o navio será inserido
    addi   s9, zero, 0         # registrador responsável por armazenar a coluna que o navio que será inserido
    addi   s10, zero, 0        # registrador para o loop
    addi   t3, zero, 1         # valores das embarcacoes
    addi   t4, zero, 0
    j      zera_matriz        # preenche toda matriz com zero.
```

**FIGURA 2 - FUNÇÃO ZERA\_MATRIZ**

```
#####
#
# zera_matriz
# função responsável por inserir o número zero(0) na matriz
#
#####
zera_matriz:
    beq    a2, a1, qnt_embarcacoes    # se a2 (0) = a1 (100), acaba o laço de repetição.
    sw     zero, (s0)                  # preenche com o valor zero a matriz
    addi   s0, s0, 4                   # vai para a próxima posição da matriz
    addi   a2, a2, 1                   # incremento de um no registrador a2
    j      zera_matriz                  # continua o laço de repetição
```

A função “**zera\_matriz**”, ela escreve o número zero (0) em todas as posições da matriz.

Após a matriz estiver com zeros em todas as posições, seguimos para a função “**qnt\_embarcacoes**” (FIGURA 3). Essa função tem como objetivo ler a quantidade de embarcações que serão inseridas. Essa leitura é feita na string navios (FIGURA 4) que está no .data

**FIGURA 3 - FUNÇÃO QNT\_EMBARCACOES**

```
qnt_embarcacoes:
    addi    a2, zero, 0           # zera a2, pois usaremos no for usado na função percorre_string_navios
    lb      t1, (s1)              # atualiza da string navios s1 em t1
    beq     t1, a4, calc_qnt_embarcacoes # se t1(valor atual da string navios) for = a4 (10 é \n na tabela ascii), significa que
                                   # já lemos a qnt_navios
    beq     s7, zero, primeiro_caracter # s7 = 0, então está no primeiro caracter da string navios

    addi    s7, s7, 1             # incrementa s7
    beq     t1, t2, reinicia_t2   # vai para o próximo número da string números
    addi    s2, s2, 1             # vai para o próximo número da string números
    lb      t2, (s2)              # carrega número da string navios s2 em t2
    addi    s9, s9, 1             # registrador responsável por armazenar o valor da quantidade de navios que serão inseridos.
    j       qnt_embarcacoes
```

**FIGURA 4 - STRING NAVIOS**

```
navios: .asciz "2\n1 5 1 1\n0 1 2 2"
```

Ao obter a quantidade de navios, irá para a função “**percorre\_string\_navios**” (FIGURA 5). Nessa função, percorremos a string navios 2 posições por vez. Dessa forma, pulamos os espaços em branco e os \n.

**FIGURA 5 - FUNÇÃO PERCORRE\_STRING\_NAVIOS**

```
percorre_string_navios:
    la      s2, numeros           # endereço inicial da string numeros carregada em s2
    lb      t2, (s2)              # carrega o primeiro número da string números s2 em t2
    la      s11, qnt_navios       # carrega o valor da memória do .word qnt_navios
    addi    a5, a5, 1             # incrementar 1 na qnt_navios para o laço for da função verifica_se_navio_afundou
    sw      a5, (s11)             # salva a qnt_navios que serão inseridos de a5 para o .word qnt_navios
    addi    a5, a5, -1            # decrementa em 1 o valor de a5
    beq     a2, a5, imprime_matriz # verifica se inseriu todos os navios a2(0) = a5 (qnt_navios)
    beq     a3, a7, orientacao_navio # se a3 = a7 (2). Significa que estamos pegando o número da orientação do navio
    beq     a3, s4, tamanho_navio # se a3 = s4 (4). Significa que estamos pegando o tamanho do navio que será inserido
    beq     a3, s6, linha_navio   # se a3 = s6 (6). Significa que estamos pegando a linha que o navio será inserido
    beq     a3, a6, coluna_navio  # se a3 (0) = a6 (8)
    addi    s1, s1, 2             # vai para o próximo número da string navios. Pula de 2 em 2 para pular os espaços e \n
    lb      t1, (s1)              # atualiza valor de t1
    addi    a3, a3, 2             # atualiza valor de t1
    j       percorre_string_navios
```

Dependendo da posição obtida da string navios, obtemos a orientação do navio, tamanho do navio, linha do navio e coluna do navio. Quando estivermos na posição 2 da string navios, pulamos para a função **orientacao\_navio**, onde logicamente obtemos a orientação do navio (0 - navio horizontal, 1 - navio vertical). Quando estivermos na posição 4 da string navios, pulamos para a função **tamanho\_navio**, onde obtemos o tamanho do navio que será inserido. Já na posição 6 da string navios, obtemos a linha em que o navio será inserido e na posição 8, obtemos a coluna do navio que será inserido.

Para saber a posição que o navio será inserido, na função “**calc\_posicao\_navio**” (FIGURA 7), basicamente fazemos o seguinte cálculo, exibido na FIGURA 6.

**FIGURA 6 - CÁLCULO PARA OBTER POSIÇÃO PARA INSERIR NAVIO**

$$\text{Posicao} = ((\text{linha} * 10) + 10) * 4.$$

coluna

bytes

linha

**FIGURA 7 - FUNÇÃO CALC\_POSICAO\_NAVIO**

```
calc_posicao_navio:
    addi    a2, a2, 1           # incrementa de 2 em 2
    mul     s11, s7, a4         # S11 (POSICÃO INICIAL NAVIO) = S7 (LINHA) * A4 (10 QNT COLUNA)
    add     s11, s11, s9        # S11 (POSICÃO INICIAL NAVIO) = S11 + S9 (COLUNA)
    mul     s11, s11, s4        # S11 (POSICÃO INICIAL NAVIO) = s11 * s4 (4)
    la      s0, matriz         # endereço inicial da matriz carregada em s0
    add     s0, s0, s11         # adiciona a posição inicial do vetor com s11 (posição que o navio será inserido)

    add     t5, s7, s5          # t5 = linha (s7) + tamanho navio (s5) e salva em t5
    blt     a4, t5, erro_navio_grande # a4 (10) < t5, significa que o navio é maior do que pode ser inserido

    add     t6, s9, s5          # t6 = coluna (s9) + tamanho navio (s5) e salva em s9
    blt     a4, t6, erro_navio_grande # a4 (10) < t6, significa que o navio é maior do que pode ser inserido

    j       insere_embarcacoes
```

Após isso, inserimos as embarcações na função “insere\_embarcacoes” (FIGURA 8), se passar por 3 validações: a posição do navio é inválida, ou seja, ao tentar inserir o navio em uma linha ou coluna superior a 9; o navio extrapola as dimensões da matriz, isto é, se eu tentar inserir um navio de tamanho 4 na posição 9x9 ele não deve permitir, pois o navio é maior que a matriz; ocorre sobreposição nos navios, ou seja, se a posição do navio que será inserido já está ocupada por outro navio.

**FIGURA 8 - FUNÇÃO INSERE\_EMBARCACOES**

```
insere_embarcacoes:
    beq     s10, s5, atualiza_registradores # se s10 (0) = s5 (tamanho navio)
    lw      t0, (s0)                   # carrega o valor da matriz s0 em t0 na respectiva posição
    bne     t0, zero, erro_posicao_ocupada # t0 != 0, é uma posição ocupada por outro navio
    beq     s3, zero, insere_embarcacoes_horizental # se o valor lido da string navios for zero, insere a embarcação na horizontal
    sw      t3, (s0)                   # salva o valor da embarcação na posição vertical
    addi    s0, s0, 40                 # incrementa o s0 em 1 e vai para a próxima posição vertical do navio
    addi    s10, s10, 1               # incrementa o s10 em 1
    addi    s7, s7, 1                 # incrementa em 1 o valor da linha
    j       insere_embarcacoes
```

Concluindo a inserção de todos os navios, vai para a função “atualiza\_registradores” (FIGURA 9), na qual zeramos alguns registradores para serem

utilizados novamente. Em seguida, retornamos para a função “percorre\_string\_navios” e imprimimos a matriz com os navios inseridos.

**FIGURA 9 - FUNÇÃO ATUALIZA\_REGISTRADORES**

```
atualiza_registradores:
    addi    a3, zero, 0           # atualiza o valor de a3 para zero
    addi    s10, zero, 0         # atualiza o aux do for que compara o tamanho do vetor
    addi    s5, zero, 0          # atualiza o valor de s5 (tamanho navio) para zero
    addi    s7, zero, 0          # atualiza o valor de s7 (linha do navio) para zero
    addi    s9, zero, 0          # atualiza o valor de s7 (coluna do navio) para zero
    addi    t3, t3, 1            # add 1 no registrador t3 (responsavel pelo numero do navio)
    la      s0, matriz           # endereço inicial da matriz carregada em s0

    jal     final_string_navios   # verifica se chegou no final da string navios

    j       percorre_string_navios
```

Somente agora, exibimos o seguinte menu (FIGURA 10)

**FIGURA 10 - MENU**

```
Digite a opcao que deseja:
1-Nova jogada
2-Mostrar estado atual da matriz
3-Mostrar matriz com as posições dos navios
4-Reiniciar jogo
5-Sair do jogo
```

Se selecionarmos a opção “**1 - Nova jogada**”, precisamos fornecer o valor da linha e da coluna em que efetuaremos o tiro. Se esse tiro for bem sucedido, ele salva o número 88 na posição, o qual é equivalente ao caractere “**X**” na tabela ASCII. Caso o tiro acerte somente o mar, ele salva o número 45 na posição, o qual é equivalente ao caractere “**-**” na tabela ASCII. Após isso, exibe o menu novamente.

Se selecionarmos a opção “**2 - Mostrar estado atual da matriz**”, irá exibir a matriz com os tiros que acertaram e erraram. As posições onde não foram efetuados os tiros, será exibida com o caractere “**?**”, conforme FIGURA 11. Note que na FIGURA 11, já é exibido os recordes e a pontuação atual do jogador.

FIGURA 11 - OPÇÃO 2 MENU

```

Recorde
Tiros: 0
Acertos: 0
Afundados: 0

Sua pontuacao
Tiros: 2
Acertos: 1
Afundados: 0
Último tiro: 00

Abaixo está a matriz atual

- ? ? ? ? ? ? ? ? ?
? X ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ? ? ?

Digite a opcao que deseja:
1-Nova jogada
2-Mostrar estado atual da matriz
3-Mostrar matriz com as posições dos navios
4-Reiniciar jogo
5-Sair do jogo
```

Se selecionarmos a opção **“3 - Mostrar matriz com posições dos navios”**, irá exibir a matriz com os tiros que acertaram e erraram. Além disso, exibe as posições dos barcos restantes (FIGURA 12). Nesse menu também é exibido os recordes e a pontuação atual do jogador.

**FIGURA 12 - OPÇÃO 3 MENU**

```

Recorde
Tiros: 0
Acertos: 0
Afundados: 0

Sua pontuacao
Tiros: 2
Acertos: 1
Afundados: 0
Último tiro: 00

Abaixo está a matriz atual

- 0 0 0 0 0 0 0 0 0 0
0 X 0 0 0 0 0 0 0 0
0 1 2 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

Digite a opcao que deseja:
1-Nova jogada
2-Mostrar estado atual da matriz
3-Mostrar matriz com as posições dos navios
4-Reiniciar jogo
5-Sair do jogo
```

Na opção **“4 - Reiniciar jogo”**, reinicia o jogo e mantém os recordes (se possuir) dos jogos anteriores.

Na opção **“5 - Sair do jogo”**, para a execução do programa.

Em relação às pontuações atuais e os recordes que são exibidos nas FIGURAS 11 e 12. O recorde só é contabilizado, caso o jogador afunda todos os barcos. Se o jogador afundar todos os barcos e reiniciar o jogo, o recorde fica armazenado até escolher a opção sair do jogo. Já a pontuação atual do jogador é reiniciada quando o jogador reinicia o jogo.

### 3. Conclusões

Contudo, com o trabalho já finalizado juntamente com as funções, validações e métodos implementados tivemos nosso conhecimento muito agregado na parte de programação em RISC-V e também em lógica propriamente dita, levando em questão também, obviamente, todo o estudo, empenho e dedicação que tivemos no decorrer dos

encontros virtuais que fizemos para juntamente chegar ao produto final, que o qual tivemos um pouco mais de dificuldade na parte que podemos dizer de “reinicialização de variáveis” como dizemos em programação de alto nível, pois no RISC-V temos uma espécie de “limite” digamos assim de registradores para armazenar e manipular valores para conseguirmos efetuar com sucesso as operações desejadas como operações de lógica, aritmética, comparação, desvios, controle, saltos, entre outros, mas conseguimos reutilizar os valores dos registradores vezes que utilizamos, tanto é que teve uma parte do código-fonte que criamos uma função específica para isso, que possibilita a reinicialização de determinados registradores para que não haja o problema de possíveis perdas ou até de sobrescrição de algum valor de algum registrador que possa acarretar em algum problema no decorrer dos testes que estão sendo feitos.

#### **4. Programa**

Segue link do repositório do código fonte do programa no github:

**<https://github.com/guilhermecardozodasilva/ORG-jogo-batalha-naval>**