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

BIANCA GABRIELA FRITSCH
DIOGO BALTAZAR DO NASCIMENTO

RELATÓRIO DO JOGO DE BATALHA NAVAL

CHAPECÓ
2021

1. PROBLEMAS E SOLUÇÕES

- Ler string de navios caracter por caracter:
    - Solução - Ler word adicionando word no registrador, pegando cada posição do word (mais 4).
- Como navio seria escrito horizontal/vertical no vetor:
    - Solução - adicionar 2 registradores(0 e 1)como char, e verificar se a primeira posição do navio era 0 ou 1.
- Problema de como escrever na vertical:
    - Solução - quando escrever no sw adicionar 40 bytes para próxima posição.
- Leitura conjunta de toda a string do navio, precisava espaço a cada dígito, o que não acontecia
    - Foi preciso verificar char a char, vantagem que já resolvia outro problema que seria validar números maiores que 9, então seria fácil verificar se a próxima posição fosse um ‘ ’ pois caso contrário seria entrada inválida.
- Conversão de char para int:
    - Solução - subtrair 48 bytes do char, pois na tabela ASCII char - 48 = inteiro.
- Escrever segundo vetor sobrescrevendo o primeiro:
    - Solução: na hora da escrita ou leitura adicionar 400 bytes no segundo vetor.
- Não saber como funcionava o jogo por nunca ter jogado:
    - Solução - pedir para um colega o que fazer quando acertava o tiro.
- Escrever a localização do tiro acertado no segundo vetor:
    - Solução: pegar a posição do navio no vetor de navios e adicionar a mesma posição + 400 bytes para ficar na posição do vetor do jogo.

2. CONCLUSÃO

No desenvolvimento do projeto nos deparamos com diversas dificuldades, tais como: ler string de navios caracter por caracter, converter char para int, inserir navios na horizontal e vertical, sobrescrever vetores, assim como entender de fato como o jogo funciona. Dessa forma, aprendemos a manipular strings, vetores e funções em assembly e entender como manipular registradores, reaproveitar e interagir com um usuário para ter a jogabilidade.

3. PROGRAMA

```
.data

ships:                  .asciz   "3 \n 1 5 1 1 \n 0 5 2 2 \n 0 1 6 4 "
error:                  .asciz   "Entrada invalida"
error_sobreposicao:     .asciz  "Sobreposição nos navios"
error_tam:              .asciz  "O navio extrapola as dimensões
da matriz"
error_tiro:             .asciz  "\n\nVoce errou seu tiro\n"
acertou_navio:          .asciz  "\n\nVoce acertou um navio\n"
columns:                .asciz   "0 1 2 3 4 5 6 7 8 9\n"
msg_Linha:              .asciz  "Digite o numero da linha pro tiro:
"
msg_Coluna:             .asciz  "Digite o numero da coluna pro
tiro: "
matriz:                 .word   100 # matriz q vai ser escrita os
navios
matriz_jogo:            .word   100 # matriz q vai aparecer
durante jogo

.text
main:
```

```
        la      a1, ships                       # lê endereço do vetor
        add     s3, s3, zero                    # autoincrement para for de printar
        addi    s4, s4, 10                      # valor para adicionar \n
        addi    s5, s5, 100                     # tamanho da matriz
        addi    t6, t6, 4                       # para coluna

        add     s1, s1, zero                    # contador para linha
        lbu     s2, (a1)            # pego valor primeira posicao = quantidade de navios

        addi    a1, a1, 4                       # pula 2 caracter pois informacao
necessaria ja adqurida na primeira linha

        li      t1, '\n'                        # grava \n
        li      t2, '\0'                        # grava eof
        li      t3, '0'                         # horizontal
        li      t4, '1'                         # vertical
        li      s11, 'A'
        j       insere_embarcacoes              # chama função insere_embarcacoes:

verifica_vazio:
        addi    a6, zero, 32


        addi    a1, a1, 1                       # pula 1 endereco de memoria, aqui por
ser string cada posicao tem 8 bits
        lbu     t0, (a1)            # estende endereco de memoria atual de a1 para t0
        bne     t0, a6, fim_error               # verifica se proxima posicao n for ' '
printa erro
        addi    a1, a1, 1                       # pula 1 endereco de memoria, se for
espaco na posicao atual
        lbu     t0, (a1)            # estende endereco de memoria atual de a1 para t0
```

```
        ret


verifica_tam:
        addi    a6, zero, 10
        add     s7, a2, a4
        bge     s7, a6, fim_error_tam
        add     s7, a2, a3
        bge     s7, a6, fim_error_tam
        ret


insere_embarcacoes:
        la      s9, matriz                      # le matriz para ser escrita
        addi    s9, s9, 4                       # evitando erro de lixo na memoria
        lbu     t0, (a1)            # estende endereco de memoria atual de a1 para t0

        add     s1, zero, zero                  # reseto contador para linha
        beq     t0, t1, loop_find_eof           # verifica se possui \n para proxima
interacao
        beq     t0, t2, inicia_game             # verifica se endereço atual é \0 vai
printar a matriz
        beq     t0, t3, insere_na_horizontal    # insere navio na horizontal
        beq     t0, t4, insere_na_vertical      # insere navio na vertical

        addi    a1, a1, 1                       # pula 1 endereco de memoria, aqui por
ser string cada posicao tem 8 bits
        lbu     t0, (a1)                # estende endereco de memoria atual de a1 para
t0
        j       insere_embarcacoes



loop_find_eof:
```

```
        addi    s11, s11, 1
        addi    a1, a1, 1                       # pula 1 endereco de memoria, aqui por
ser string cada posicao tem 8 bits
        lbu     t0, (a1)                # estende endereco de memoria atual de a1 para t0
        j       insere_embarcacoes



insere_na_horizontal:

        jal     verifica_vazio
        add     a2, a2, zero                    # tamanho do navio
        addi    a2, t0, -48                     # coloca o tamanho do navio lido na linha
converte pra int e grava

        jal     verifica_vazio
        add     a3, a3, zero                    # linha do navio
        addi    a3, t0, -48                     # coloca a linha inicial do navio lido na
linha converte pra int e grava

        jal     verifica_vazio
        add     a4, a4, zero                    # coluna do navio
        addi    a4, t0, -48                     # coloca o coluna inicial do navio lido na
linha converte pra int e grava

        mul     t5, a3, s4                      # multiplico pra ter a coluna inicial
        add     a5, t5, a4                      # somo a coluna com a linha pra saber
posicao na matriz
        mul     a5, a5, t6                      # multiplco * 4 para ter posicao na
memoria correta

        jal     verifica_vazio
```

```
        add     s9, s9, a5
        jal     verifica_tam
        j       preenche_vetor_horizontal

insere_na_vertical:
        jal     verifica_vazio

        add     a2, a2, zero                    # tamanho do navio
        addi    a2, t0, -48                     # coloca o tamanho do navio lido na linha
converte pra int e grava

        jal     verifica_vazio
        add     a3, a3, zero                    # linha do navio
        addi    a3, t0, -48                     # coloca o linha inicial do navio lido na
linha converte pra int e grava

        jal     verifica_vazio
        add     a4, a4, zero                    # coluna do navio
        addi    a4, t0, -48                     # coloca o coluna inicial do navio lido na
linha converte pra int e grava

        mul     t5, a3, s4                      # multiplico pra ter a coluna inicial
        add     a5, t5, a4                      # somo a coluna com a linha pra saber
posicao na matriz
```

```
        mul     a5, a5, t6                      # multiplco * 4 para ter posicao na
memoria correta



        jal     verifica_vazio
        add     s9, s9, a5
        jal     verifica_tam
        j       preenche_vetor_vertical


preenche_vetor_vertical:
        beq     s1, a2, insere_embarcacoes      # se o tamanho do navio ja tiver completo
        addi    s1, s1, 1

        lw      a6, 0(s9)                       # pega valor em s9
        bne     a6, zero, fim_error_choque      # se valor diferente de 0 ja existe navio
entao choque de navios

        sw      s11, 0(s9)
        addi    s9, s9, 40                      # se for vertical escrevo na mesma
posicao a cada 10 colunas

        j       preenche_vetor_vertical


preenche_vetor_horizontal:
        beq     s1, a2, insere_embarcacoes      # se o tamanho do navio ja tiver completo
        addi    s1, s1, 1                       # auto incremento

        lw      a6, 0(s9)                       # pega valor em s9
```

```
        bne     a6, zero, fim_error_choque      # se valor diferente de 0 ja existe navio
entao choque de navios

        sw      s11, 0(s9)
        addi    s9, s9, 4                       # se for horizontal escrevo na proxima
posicao

        j       preenche_vetor_horizontal



header:
        la      s7, columns                     # le matriz para ser printada
        mv      a0, s7                          # imprime os valores
        li      a7, 4
        ecall

        addi    s3, s3, 1                       # autoincremento meu for
        j loop_matriz

loop_matriz:
        beqz    s3, header
        lw      a6, (s9)                        # carregando matriz de endereços
copiada

        mv      a0, a6                          # imprime os valores
        li      a7, 11
        ecall

        li      a0, ' '                         # imprime espaço para ficar melhor
visualmente
        li      a7, 11
```

```
        ecall


        addi    s9, s9, 4                       # vou para proxima posicao do vetor
copiado
        addi    s3, s3, 1                       # autoincremento meu for
        remu    s6, s3, s4                      # pego resto da divisão do
autoincrementro / 10
        beq     s3, s5, fim                     # vejo se o for precisa ser parado
        beqz    s6, print_n                     # vejo se o resto do autoincremento / 10
= 0 para adicionar \n
        j       loop_matriz                     # percorro o proximo elemento

print_n:

        li      a0, '\n'                        # printo \n
        li      a7, 11
        ecall
        j       loop_matriz                     # volto pra matriz


inicia_game:
        la      s8, matriz_jogo                 # le matriz para jogo
        beq     t2, s5, prepara_loop_matriz     # se foi preenchido vai printar matriz
para iniciar game
        j       preenche_vetor_jogo

preenche_vetor_jogo:
        addi    t2, t2, 1                       # autoincremento meu for

        add     s11, zero, zero
        addi    s11, zero, 42                   # valor padrao da matriz vai ser X
```

```
        sw      s11, 396, (s8)
        addi    s8, s8, 4                       # escrevo valor padrao em todas as
posicoes
        beq     t2, s5, inicia_game
        j       preenche_vetor_jogo


header_jogo:
        la      s7, columns                     # le matriz para ser printada
        mv      a0, s7                          # imprime os valores
        li      a7, 4
        ecall

        addi    t2, t2, 1                       # autoincremento meu for
        j       loop_matriz_jogo

print_n_jogo:

        li      a0, '\n'                        # printo \n
        li      a7, 11
        ecall
        j       loop_matriz_jogo                #volto pra matriz


loop_matriz_jogo:
        beqz    t2, header_jogo
        lw      a6, 396, (s8)                   # carregando matriz de endereços
copiada
```

```
        mv      a0, a6                  # imprime os valores
        li      a7, 11
        ecall

        li      a0, ' '                 # imprime espaço para ficar melhor visualmente
        li      a7, 11
        ecall

        addi    s8, s8, 4               # vou para proxima posicao do vetor copiado
        addi    s3, s3, 1               # autoincremento meu for
        remu    s6, s3, s4              # pego resto da divisão do autoincrementro / 10
        beq     s3, s5, jogo            # vejo se o for precisa ser parado
        beqz    s6, print_n_jogo        # vejo se o resto do autoincremento / 10 = 0 para adicionar \n
        j       loop_matriz_jogo        # percorro o proximo elemento

jogo:
        li      a0, '\n'                # printo \n
        li      a7, 11
        ecall

        la      a0, msg_Linha           # imprime mensagem
        li      a7, 4
        ecall

        addi    a7, zero, 5             #lê inteiro
        ecall
        add     a3, zero, a0            # carrega valor lido em a3
```

```
        la      a0, msg_Coluna          # imprime mensagem
        li      a7, 4
        ecall

        addi    a7, zero, 5             #lê inteiro
        ecall
        add     a4, zero, a0            # carrega valor lido em a4

        j       valida_tiro

valida_tiro:

        mul     t5, a3, s4              # multiplico pra ter a coluna inicial
        add     a5, t5, a4              # somo a coluna com a linha pra saber
posicao na matriz
        mul     a5, a5, t6              # multiplco * 4 para ter posicao na
memoria correta

        add     s9, s9, a5

        lw      a6, 4(s9)               # pega valor em s9
        mv      a0, a6                  # imprime os vetor de char
        li      a7, 1
        ecall
        beq     a6, s5, errou_tiro
        bgtz    a6, acertou_tiro

acertou_tiro:
        addi    s10, zero, 400
```

```
        bgt     a5, s10, fim_error              #verifica se foi pra posicao errada

        la      s7, acertou_navio               # le msg
        mv      a0, s7                          # imprime os vetor de char
        li      a7, 4
        ecall

        add     s3, zero, zero                  #contador
        add     t2, zero, zero                  #contador
        la      s8, matriz_jogo
        j       escreve_na_matriz_jogo          #procura e marca todo navio q foi
acertado nesse tiro

prepara_loop_matriz:
        add     t2, zero, zero
        la      s8, matriz_jogo                 # le matriz para jogo
        la      s9, matriz                      # le matriz para jogo
        j       loop_matriz_jogo



escreve_na_matriz_jogo:
        addi    s10, zero, 32
        beq     a6, s10, errou_tiro
        addi    s10, zero, 42
        beq     a6, s10, errou_tiro
        beqz    a6, errou_tiro

        add     s8, s9, a5
        sw      a6, 400,(s9)
        j       prepara_loop_matriz
```

```
errou_tiro:
        la      s7, error_tiro          # le erro
        mv      a0, s7                  # imprime os vetor de char
        li      a7, 4
        ecall
        add     s3, zero, zero

        addi    s10, zero, 120
        sw      s10, 400,(s9)
        j       prepara_loop_matriz
        j       prepara_loop_matriz

fim_error:
        la      s7, error               # le erro
        mv      a0, s7                  # imprime os vetor de char
        li      a7, 4
        ecall
        j       fim

fim_error_choque:

        la      s7, error_sobreposicao  # le erro
        mv      a0, s7                  # imprime os vetor de char
        li      a7, 4
        ecall
        j       fim

fim_error_tam:
```

```
        la      s7, error_tam                   # le erro
        mv      a0, s7                  # imprime os vetor de char
        li      a7, 4
        ecall
        j       fim

fim:
        nop
```