Universidade Federal da Fronteira Sul
Curso de Ciência da Computação
Organização de Computadores
Professor Luciano L. Caimi
Débora Rebelatto - 1721101034

Relatório Batalha Naval

Problema

O problema proposto é um jogo de Batalha Naval para dois jogadores desenvolvido utilizando o conjunto de instruções do processador RISC-V RV32IM, no qual o objetivo é afundar os navios do adversário que estão dispostos em uma matriz 10x10.

O jogo recebe as informações sobre os navios do jogador 1, verifica a possibilidade de posicionar o navio dentro da matriz e insere ele, salvando essa matriz como gabarito.
Após isto, o jogador 2 deve disparar tiros que ficarão salvos em uma matriz de disparos, que servirá para fazer comparações com a de navios.

Solução

A partir da descrição do problema, foi necessário levantar quais seriam as funções para organizar as funcionalidades do jogo, assim foi feito um código em C para auxiliar na estruturação do código e desenho de um fluxograma da aplicação.

Após isso, é necessário dar início ao código em Assembly. Precisamos da entrada de usuário sobre número de navios e quais serão as informações de disposição, comprimento, linha inicial e linha final. Além disso, antes de fazer a inserção do navio, é necessário fazer verificações sobre se a partir das informações é possível posicionar o navio.

As situações propostas pelo professor seriam as seguintes:
a) A posição do navio é inválida. Exemplo: 0 3 11, 7
b) O navio extrapola as dimensões da matriz. Exemplo: 0 4 9 2
c) Ocorre sobreposição nos navios. Exemplo 0 4 2 2 e 1 3 0 3

Na função pergunta_linha é verificado se o número inserido para a linha inicial da embarcação ultrapassa de 9, se for, ela mostra a mensagem de erro e repete a pergunta. Mesma coisa ocorre para pergunta de coluna.

A função de menu aloca os registradores temporários e faz a comparação com a entrada de usuário para fazer os branches respectivos de cada função.

```
menu:
    addi t0, zero, 1    # T0 = 1
    addi t1, zero, 2    # T1 = 2
    addi t2, zero, 3    # T2 = 3
    addi t3, zero, 4    # T3 = 4

    la a0, msg_menu     # Menu
    li a7 4
    ecall

    li a7, 5
    ecall

    # Transições de menu
    beq t0, a0, reiniciar_jogo  # branch if equal to reiniciar_jogo
    beq t1, a0, estado_matriz       # branch if equal to estado_matriz
    beq t2, a0, nova_jogada         # branch if equal to nova_jogada
    beq t3, a0, exit                        # branch if equal to exit
```

Ao reiniciar o jogo, todos os registradores alocados ao longo do jogo são zerados para evitar qualquer conflito ao longo do novo jogo.

```
reiniciar_jogo:
    addi    a0, zero, 0
    ...

    addi    t0, zero, 0
    ...

    addi    s0, zero, 0
    ...

    # Reiniciar matrizes
    j get_info_embarcacoes
```

A função que mostra a matriz e estatísticas de jogo é dividida em três partes: Matriz, jogo atual e recordes. A função de imprimir matriz inicia os contadores e seta registradores

necessários para comparações e matrizes do jogo. Após, carrega numeração na parte de cima da matriz pula para a função de mensagem lateral.

```
imprime_matriz:
    li  t0, 0               # Min Contador
    li  t1, 10              # Max Contador
    la  t2, msg_lateral     # t2 = Mensagem Lateral
    li  t6, -1              # t6 = -1

    la  s0, matriz_batalha  # S0 = Matriz Batalha
    la  s1, matriz_tiros    # S1 = Matriz Tiros
    mul t3, t1, t1          # t3 = Calcula o tamanho da matriz

    la a0, matriz_header
    li a7, 4
    ecall

    la  a0, msg_cima    # Carrega a mensagem das marcaÃ§Ãµes que
vÃ£o na parte de cima da matriz
    li  a7, 4           # Imprime a mensagem
    ecall

    j   imprime_lateral
```

Carrega o primeiro valor para a coluna lateral

```
imprime_lateral:
    lw a0, 0(t2)    # Carrega o primeiro valor da mensagem
que vai na lateral da matriz para marcar as posições
    li a7, 1        # Imprime o valor
    ecall

    la  a0, msg_espaco  # Carrega a mensagem
    li  a7, 4           # Imprime a mensagem
    ecall

    addi    t2, t2, 4   # Acessa o próximo valor

    j   imprime
```

A função imprime faz as transições necessárias caso a matriz tenha terminado de ser imprimida ou se imprime as linhas de posições vazias na matriz.

```
imprime:
    beq t0, t3, standings   # Repete a função e após mostra
as estatísticas do jogo
    beq t0, t1, pula        # Se o contador for igual a 10,
pula para a função pula

    lw      a0, 0(s1)       # Carrega o valor da matriz

    beq a0, t6, arruma
    li  a7, 1               # Imprime o valor
    ecall

    la  a0, msg_espaco      # Carrega a mensagem espaço
    li  a7, 4               # Imprime a mensagem
    ecall
```

Arruma as posições na matriz como linhas e atualiza contadores.

```
arruma:
    la  a0, msg_traco   # Carrega a mensagem
    li  a7, 4           # Imprime a mensagem
    ecall

    la  a0, msg_espaco  # Carrega a mensagem
    li  a7, 4           # Imprime a mensagem
    ecall

    addi    s0, s0, 4 # Acrescenta 4 no endereço da matriz
    addi    t0, t0, 1   # Acrescenta 1 no contador
    j   imprime
```

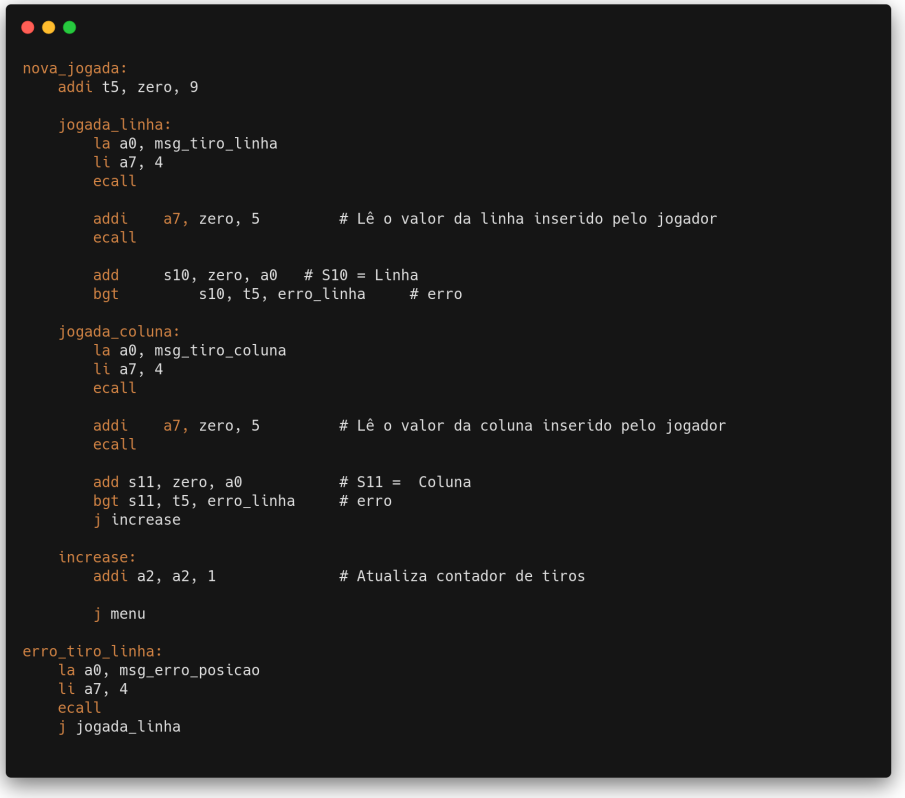
Carrega valores para a linha horizonal.

```
pula:
    la  a0, msg_enter   # Carrega a mensagem
    li  a7, 4           # Imprime a mensagem
    ecall

    addi    t1, t1, 10  # Acrescenta o número da próxima quebra de linha na matriz

    lw  a0, 0(t2)       # Carrega o primeiro valor da mensagem que vai na lateral da matriz
    li  a7, 1           # Imprime o valor
    ecall

    la  a0, msg_espaco  # Carrega a mensagem
    li  a7, 4           # Imprime a mensagem
    ecall

    addi    t2, t2, 4   # Acessa o próximo valor da mensagem
    j   imprime
```

Parte da função standings. Faz a impressão do jogo atual e recordes.

```
    # Tiros
    la a0, msg_currentTiros
    li a7, 4
    ecall
    addi a0, a2, 0
    li a7, 1
    ecall
    la a0, msg_enter
    li a7 4
    ecall

    # Acertos
    la a0, msg_currentAcertos
    li a7 4
    ecall
    addi a0, a3, 0
    li a7, 1
    ecall
    la a0, msg_enter
    li a7 4
    ecall

        # Linha
    la a0, msg_ultimoTiroLinha
    li a7 4
    ecall
    addi a0, s10, 0
    li a7, 1
    ecall

        # Coluna
    la a0, msg_ultimoTiroColuna
    li a7 4
    ecall
    addi a0, s11, 0
    li a7, 1
    ecall

    la a0, msg_enter
    li a7 4
    ecall
```

Faz uma nova jogada a partir da inserção de linhas e colunas por parte do jogador e atualiza valores dos contadores. Caso jogador insira valor maior que 9, mostra mensagem de erro.



```
nova_jogada:
    addi t5, zero, 9

    jogada_linha:
        la a0, msg_tiro_linha
        li a7, 4
        ecall

        addi    a7, zero, 5         # Lê o valor da linha inserido pelo jogador
        ecall

        add     s10, zero, a0   # S10 = Linha
        bgt         s10, t5, erro_linha     # erro

    jogada_coluna:
        la a0, msg_tiro_coluna
        li a7, 4
        ecall

        addi    a7, zero, 5         # Lê o valor da coluna inserido pelo jogador
        ecall

        add s11, zero, a0           # S11 =  Coluna
        bgt s11, t5, erro_linha     # erro
        j increase

    increase:
        addi a2, a2, 1              # Atualiza contador de tiros

        j menu

erro_tiro_linha:
    la a0, msg_erro_posicao
    li a7, 4
    ecall
    j jogada_linha
```

Conclusão

O desenvolvimento do trabalho de Batalha Naval foi de grande importância para o aprendizado da arquitetura que envolve o RISC-V e Assembly enquanto ferramentas essenciais para a disciplina de Organização de Computadores.

A partir da atividade, foi possível ter uma compreensão maior sobre o que envolve o conjunto de instruções e como utilizá-lo dentro de projetos de forma clara e objetiva utilizando a ferramenta RARS.

Programa

O código fonte do programa assim como outros auxiliares para o desenvolvimento deste projeto podem ser encontrados no GitHub - OrgBatalhaNaval.