UNIVERSIDADE FEDERAL DE SANTA MARIA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Relatório do Trabalho de Organização de Computadores

Desenvolvimento do jogo Connect 4 em assembly para MIPS

Nomes: Luan Tiago Streck, André Paulo Streck Renner

Turma: CC1

Professor: Giovani Baratto

Santa Maria, novembro de 2024.

SUMÁRIO:

1.INTRODUÇÃO.

Foi proposta uma avaliação em forma de trabalho para a disciplina de Organização de Computadores, na qual os alunos deveriam escolher entre fazer o jogo Connect 4 ou o Snake Game. No trabalho em questão, a decisão tomada pela dupla foi de fazer o Connect 4.

Assim, com os conhecimentos obtidos em aula e as pesquisas na internet, foi possível desenvolver um algoritmo em Assembly para MIPS, compilado no software Mars. Nesse algoritmo deve haver a lógica que faz o jogo funcionar, bem como o código que exibe o tabuleiro e as peças no display de bitmap do Mars.

Imagem 1- Exemplo de como é o jogo Connect 4.

2. OBJETIVOS.

O objetivo principal para esse trabalho foi criar um código em assembly capaz de executar toda a lógica do jogo escolhido e do funcionamento do display, sem que houvesse erros ou que o código “crashasse”. Dessa forma, oferecendo uma boa experiência a quem fosse usar, bem como cumprir com a proposta do trabalho.

Além disso, também era buscado obter mais conhecimentos a respeito da linguagem de programação Assembly para MIPS, que é uma linguagem nova e até então desconhecida para a maioria dos estudantes. Ademais, buscou-se desenvolver boas práticas e experiências com essa linguagem, lidando com seus diversos comandos e tornando os códigos mais rápidos e efetivos.

3.METODOLOGIA.

A metodologia para a realização do trabalho se baseou em duas etapas: criação do código do jogo em C (linguagem conhecida pelos estudantes) para comparação com a sua correspondência em Assembly e também a conversão e criação do código em Assembly, separando por partes.

Num primeiro momento, a ideia foi criar a lógica do jogo em uma linguagem conhecida e, assim, partir de associações para fazer a conversão, entender, criar e lidar com um código que funcione de forma equivalente. No entanto, tal ideia não se mostrou totalmente efetiva, visto que as duas linguagens ainda possuem uma grande diferença entre si. Além disso, a criação do código para o display de bitmap do software MARS exigiu um código exclusivo, que lidasse com as variáveis e criasse o desenho correspondente no display.

Durante o desenvolvimento do projeto, ele foi dividido em duas partes: o código que calculava e cuidava da parte de lógica do Connect 4, como a inserção, verificação de vitória, fim do jogo, etc. e a parte de exibir o tabuleiro com as peças no display, que desenhava o fundo do tabuleiro e então varria a matriz de peças e imprimia-as no seu devido lugar no display.

Nesse sentido, com o passar do tempo e do desenvolvimento do trabalho no MARS, a programação em Assembly começou a se tornar mais fácil, visto que foi possível adquirir experiência durante esse tempo e, dessa forma, aquele primeiro problema da diferença entre as linguagens foi superado pela capacidade de programar e entender melhor essa nova linguagem.

4.DETALHAMENTO.

O trabalho foi montado seguindo diversos algoritmos como loop e procedimentos, com cada um sendo responsável por uma parte da execução da lógica do jogo ou pela estética e representação do jogo no display.

O jogo funciona adicionando as entradas no terminal, já que na construção do trabalho optou-se por não utilizar a função de *keyboard* (teclado) e, assim, basta selecionar uma coluna válida, do valor 1 até o 7, sendo um inteiro. Como foi programado para obter o *char*, então não é necessário pressionar *enter*, apenas o número da coluna escolhida. Caso haja uma entrada diferente de 1-7, mas sem overflow, aparecerá uma imagem de erro no terminal. Dessa forma, o jogo funciona, a partir daí, basta jogar ou fazer os testes. O tabuleiro aparece no display do MARS e também uma matriz simples imprimida no terminal, com 0 para espaços vazios, 1 para as peças do jogador 1 e 2 para as peças do jogador 2.

Sobre os procedimentos de estética:

No código do jogo, existem alguns procedimentos responsáveis pela estética, como:

- *init2_graph_test:* é um procedimento copiado do arquivo de exemplo de display do professor Giovani Baratto, que inicializa o display do Mars.
- *desenha_tabuleiro:* ele inicializa a matriz do tabuleiro no display, pintando o fundo e cortando ele com linhas para formar a matriz 6x7.
- *desenha_numeros:* é um procedimento simples que faz os números das colunas na parte inferior do display de forma manual, abaixo do tabuleiro, para ajudar na decisão da jogada.
- *desenha_circulo:* é um procedimento auxiliar criado exclusivamente para desenhar as peças na posição desejada, ela recebe uma coordenada de x e y na sua chamada e faz a impressão do círculo (peça do jogo) naquele lugar desejado.
- *entrada_pecas:* é a função responsável por mostrar a inserção das peças no tabuleiro, que vai atualizando-o conforme as jogadas na parte de lógica do código. Ela decide a cor das peças de acordo com o

preenchimento da matriz do tabuleiro, que é percorrido em um loop. Caso seja 1, imprime uma peça vermelha e vai para o próximo, caso seja 2, imprime a peça amarela e avança e caso seja 0 somente avança, sem imprimir.

Sobre os procedimentos de lógica:

Agora, sobre a parte da lógica do código, que precisou de vários procedimentos para contemplar todas as necessidades:

- *main:* nele, há um laço de repetição do-while que é executado e apenas será interrompido em caso de vitória ou empate. Dentro do laço são chamadas as funções de inserir peças, exibir tabuleiro, verificar empate e verificar vitória. Além disso, a troca entre vez do jogador a cada jogada ocorre dentro do laço.
- *imprimeTabuleiro:* ele é construído com um laço for duplo que acessa cada célula do tabuleiro e a imprime na tela. O procedimento não recebe nada como parâmetro, já que o vetor tabuleiro é uma variável global, bem como o número de linhas e colunas do mesmo. Ele tem retorno void, já que apenas exibe o tabuleiro na tela para melhor localização dos jogadores.
- *verificaEmpate:* esse procedimento acessará todos os elementos da última coluna com um laço for, e caso nenhum deles seja 0, é sinal que todas as colunas estão cheias com peças dos jogadores e não há mais jogadas disponíveis, ocasionando no empate (retorno 0).
- *jogada*: é responsável pelo cálculo da linha de uma peça jogada a partir de sua coluna, em que cairá até chegar na linha 0 ou haver uma peça abaixo dela. Após isso, a célula é marcada na posição da coluna inserida e da linha calculada com o número do jogador que jogou. A função também verifica se a jogada é válida (se a coluna já não está cheia), retornando 0 caso a jogada seja inválida e 1 caso seja válida.
- *verificaVitoria*: é um laço triplo, em que, a partir de cada célula do tabuleiro, andará 4 casas em todas as 4 direções (horizontal, vertical, diagonal principal e secundária) e verificará se essas 4 casas tem a mesma marcação que a do último jogador que jogou. Caso tenha, a

função retorna 1 e o programa é finalizado, caso contrário, prossegue normalmente.

Dessa maneira foi idealizado e montado o código responsável por fazer o código de Connect 4 funcionar tanto em sua parte lógica quanto na estética do display. Assim, com essa explicação detalhada, é mais fácil de entender o que acontece em cada parte do código, visto que ele ficou razoavelmente extenso.

5.EXPERIMENTOS.

Partindo da construção dos procedimentos e das ideias listadas no tópico anterior (4.DETALHAMENTO), começaram os testes e experimentos sobre o jogo.

Neles, foram testados as entradas de peças, alocação da memória no tabuleiro, impressão das peças e matriz tanto no terminal quanto no display, entre outros. Isso pode ser visto nas imagens a seguir:

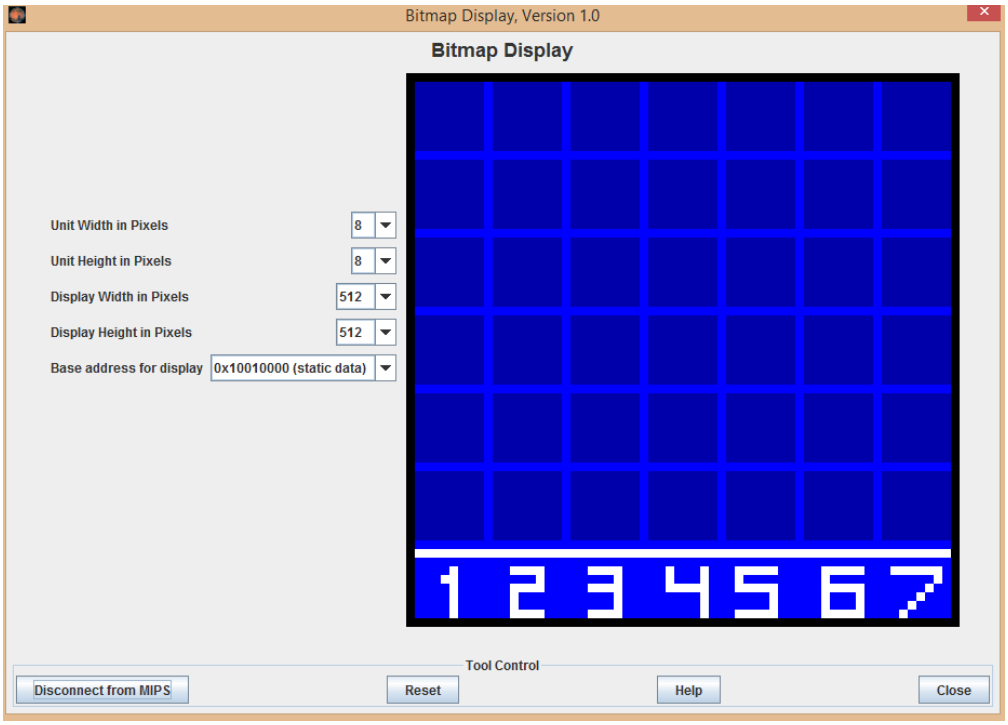

Imagem 2- Inicialização do tabuleiro no display.

Nesse caso, o procedimento de inserção de peças ainda não estava ativo na execução do programa, era puramente a inicialização do tabuleiro com o seu desenho e números.

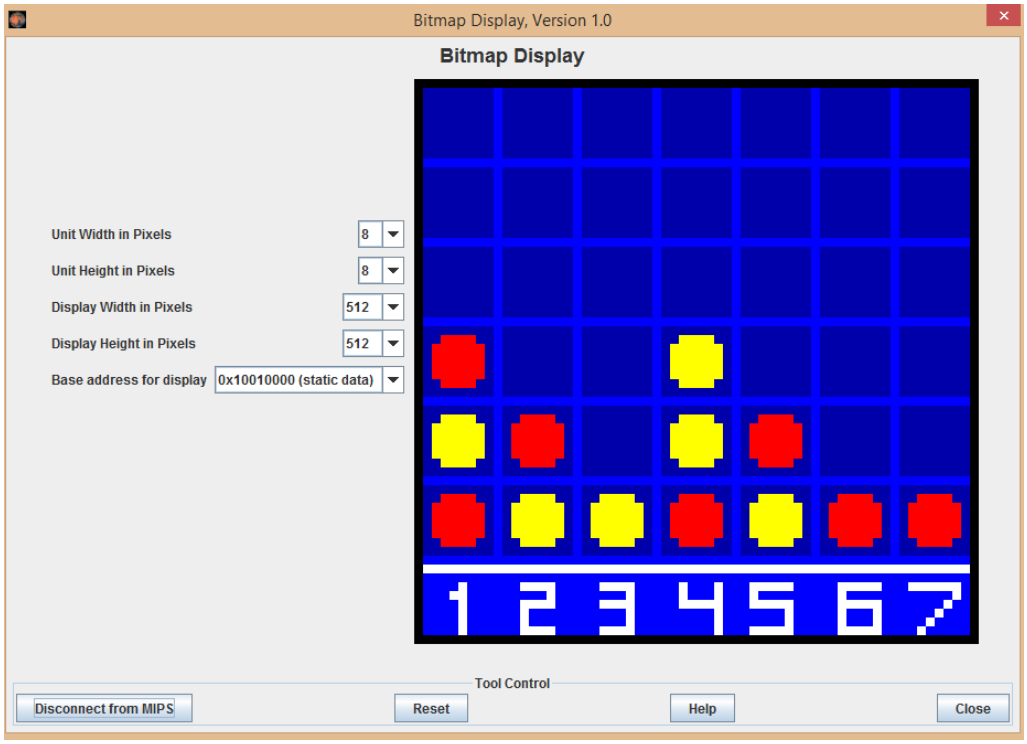

Imagem 3- Teste de inserção de peças com visualização no display.

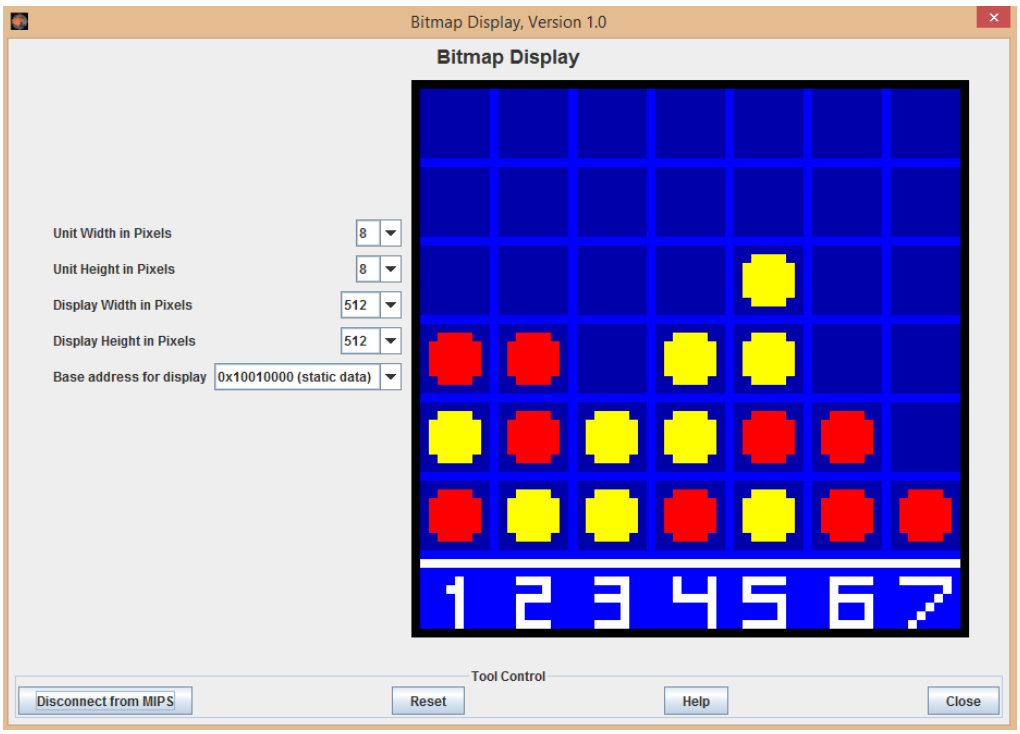

Imagem 4- Teste de vitória com visualização no display.

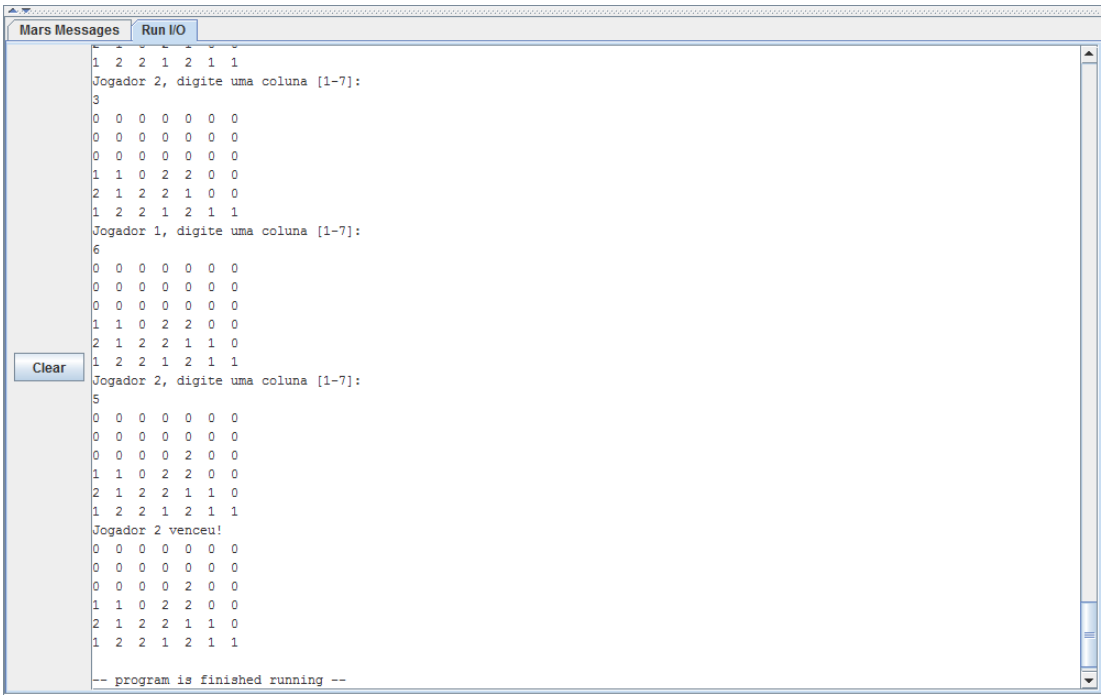

Imagem 5- Teste do jogo, com inserção e vitória das imagens acima no terminal.

As imagens 3, 4 e 5 mostram uma sequência de jogadas do Connect 4, sendo que a 3 e a 4 mostram no display do Mars e a 5 mostra no terminal do aplicativo. Ali é possível ver as jogadas acontecendo até a vitória e encerramento do programa, evidenciada pela mensagem na ultima linha da imagem 5. Assim, realizando esses experimentos, foram feitas as correções e adaptações necessárias até o pleno funcionamento do código.

6.RESULTADOS.

Os resultados obtidos atenderam as expectativas e objetivos desse projeto. Felizmente, assim como visto no tópico anterior, por meio daqueles experimentos e testes conseguimos alcançar um código com eficiência e excelência para o jogo Connect 4.

O jogo admite entradas possíveis e faz a entrada de peças no vetor (matriz) do tabuleiro, imprime elas no display e faz a verificação de empate e vitória do jogo, funcionando de forma plena e correta, como era previsto e buscado.

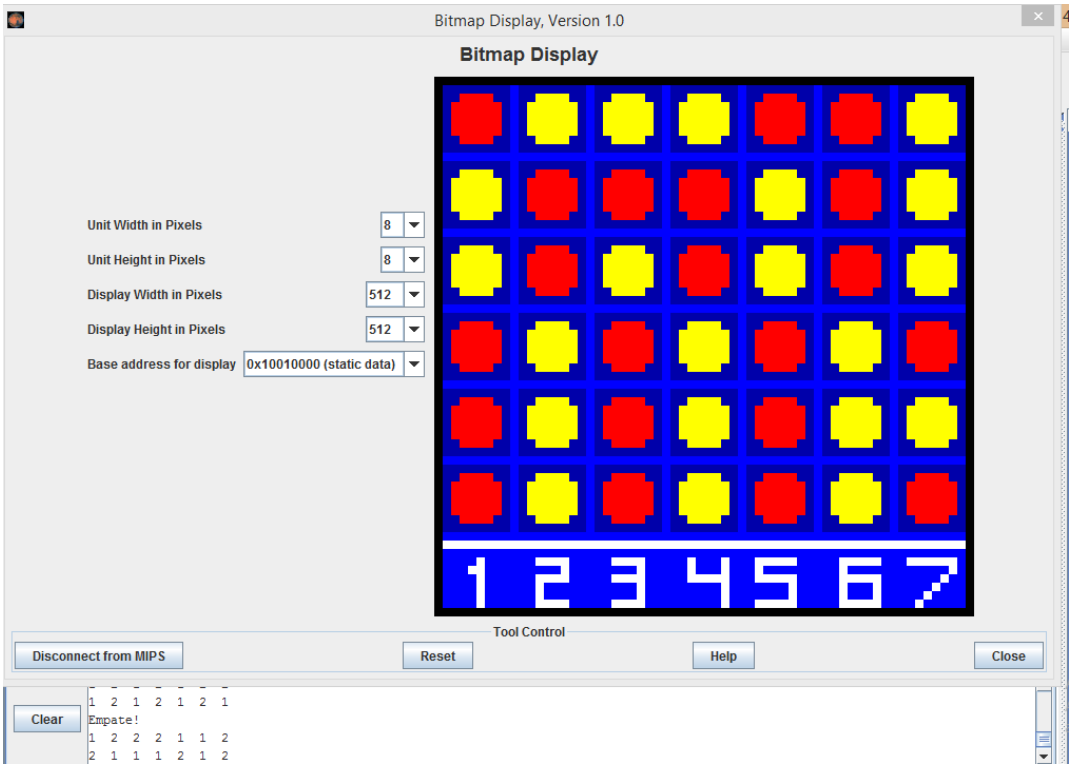

Imagem 6- Exemplo de verificação de empate.

Assim, esse e os exemplos do tópico anterior mostram os resultados do trabalho, que aborda as diferentes possibilidades e funciona para todas elas.

7.DISCUSSÃO.

Para a realização desse trabalho, como visto no tópico 2 sobre Metodologia, discutimos sobre a estratégia para montar o código correto e de forma mais tranquila, para que cada um conseguisse fazer sua parte sem interferir na do outro e tornar, assim, a montagem mais eficiente. Tivemos que fazer algumas alterações durante o trabalho, mas dentro do previsto.

Enquanto estávamos fazendo o jogo, surgiram as primeiras fontes de discussão, e entre elas, a principal foi de como lidar em um trabalho claramente mais complexo que o normal em uma linguagem de baixo nível como essa, visto que na linguagem C, que estamos acostumados, a disponibilidade de variáveis é infinita, já em Assembly tudo deve ser bem pensado utilizando apenas os registradores temporários e salvos, de forma que todos funcionem bem juntos, já que a disponibilidade dos temporários ($t) vai de $t0 até $t9 e a dos salvos ($s) vai de $s0 até $s7, totalizando 18 registradores. Entre esses tivemos que cuidar a utilização dos salvos nos procedimentos, porque eles não são exclusivos de cada procedimento e, portanto, podem ser sobrescritos em determinada chamada que também lidem com eles, causando *bugs* e erros.

Felizmente, foi possível lidar com eles, o que contribuiu muito para isso foi a possibilidade de guardar na pilha ($sp) os registradores temporários antes das chamadas de procedimentos com passagem de argumento e carrega-los de volta ao final da chamada. Assim, os registradores continuaram com seus valores originais e não foram alterados ou sobrescritos, problema que estava gerando erros anteriormente.

Dessa forma, conseguimos ultrapassar os problemas que apareceram, por meio da conversa, pesquisa e entendimento do software e da linguagem em questão.

8.REVISÃO BIBLIOGRÁFICA.

Os materiais utilizados para a resolução desse trabalho foram as aulas e os slides de aula do professor Giovani Baratto, disponíveis no Moodle, e o livro de apoio, bem como o uso de algumas inteligências artificiais para obter respostas mais diretas em determinados casos.

9.CONCLUSÕES E PERSPECTIVAS.

Ao final do trabalho, felizmente, foi possível passar por todas as dificuldades e obstáculos ao realizar o trabalho e entregar um código em Assembly para MIPS que funciona perfeitamente em sua lógica e comandos para o display do aplicativo.

De fato, foi uma tarefa complicada que exigiu bastante atenção e estratégia para pensar como fazer e então executar como o previsto, até fazer todas as mudanças e chegar em um código funcional. Foi uma atividade desafiadora, pelas limitações de uma linguagem antiga e de baixo nível, entretanto é mais eficiente pela maior proximidade à linguagem de máquina em comparação as outras linguagens. Ainda assim, o trabalho foi concluído e funcionou perfeitamente.