



UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO



Rodrigo Nunes de Castro
Rúben Barbosa da Paz
Thales Aguiar de Lima

RELATÓRIO: Implementação do Jogo da Velha

Natal/RN
Novembro/2014

Rodrigo Nunes de Castro

Rúben Barbosa da Paz

Thales Aguiar de Lima

RELATÓRIO: Implementação do Jogo da Velha

Relatório de projeto apresentado como requisito parcial para obtenção de aprovação na disciplina Introdução a Organização e Arquitetura de Computadores, no Curso de Bacharelado em Tecnologia da Informação, na Universidade Federal do Rio Grande do Norte.

Prof. Ms. Kayo Gonçalves e Silva

Natal/RN

Novembro/2014

SUMÁRIO

1. INTRODUÇÃO
2. DESENVOLVIMENTO
 - 2.1. Como executar
 - 2.2. Configurações
3. SOLUÇÕES IMPLEMENTADAS
 - 3.1. Desenhando o tabuleiro
 - 3.2. Imprimindo o tutorial
 - 3.3. Iniciando a jogada
 - 3.4. Desenhando as figuras
 - 3.5. Encontrando o vencedor
 - 3.6. Placar
 - 3.7. Jogar novamente
4. CONCLUSÃO
5. REFERÊNCIAS

1. INTRODUÇÃO

Este relatório é válido como parte da nota final da disciplina de Introdução a Organização e Arquitetura de Computadores. Aqui será explicado como foi o desenvolvimento do código para o trabalho de tema Jogo da velha; bem como todas as dificuldades, soluções, erros, instruções de execução e estado atual do projeto no momento de envio.

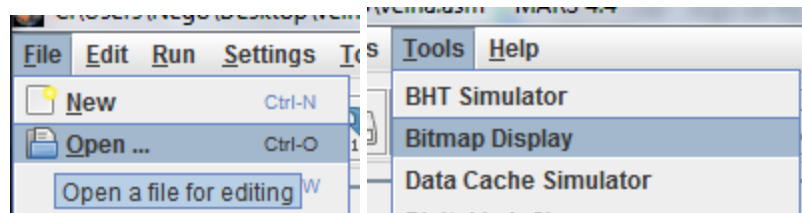
2. DESENVOLVIMENTO

2.1. Como executar

1. Abrir o código-fonte, através do software MARS;
 - a. Menu File > Open, ou simplesmente usando o atalho CTRL+O, e selecionar o arquivo [nomedoarquivo]

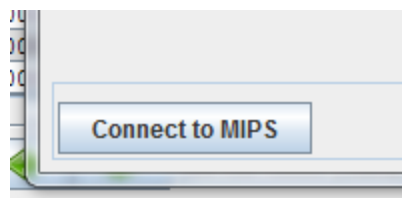
(Fonte: Autoria própria)

2. Abrir o Bitmap Display
 - a. Menu Tools > Bitmap Display



(Fonte: Autoria própria)

3. Inserir as configurações listadas no item Configurações;
4. Conectar o Bitmap ao Mips, através do botão “Connect to MIPS”;



(Fonte: Autoria própria)

5. Construir o código, através do botão “Assembly” (Botão com duas ferramentas cruzadas);



(Fonte: Autoria própria).

6. Rodar o código, através do botão “Run”;



(Fonte: Autoria própria).

2.2 Configurações

- Quantidade de pixels
 - Opção “Display Width in Pixels” = 128
 - Opção “Display Height in Pixels” = 128
- Tamanho dos pixels
 - Opção “Unit Width in Pixels” = 2

- Opção “Unit Height in Pixels” = 2
- Endereço base
 - Opção “Base adress for display” = 0x10010000

Unit Width in Pixels	2 ▼
Unit Height in Pixels	2 ▼
Display Width in Pixels	128 ▼
Display Height in Pixels	128 ▼
Base address for display	0x10010000 (static data) ▼

(Fonte: Autoria própria)

3. SOLUÇÕES IMPLEMENTADAS

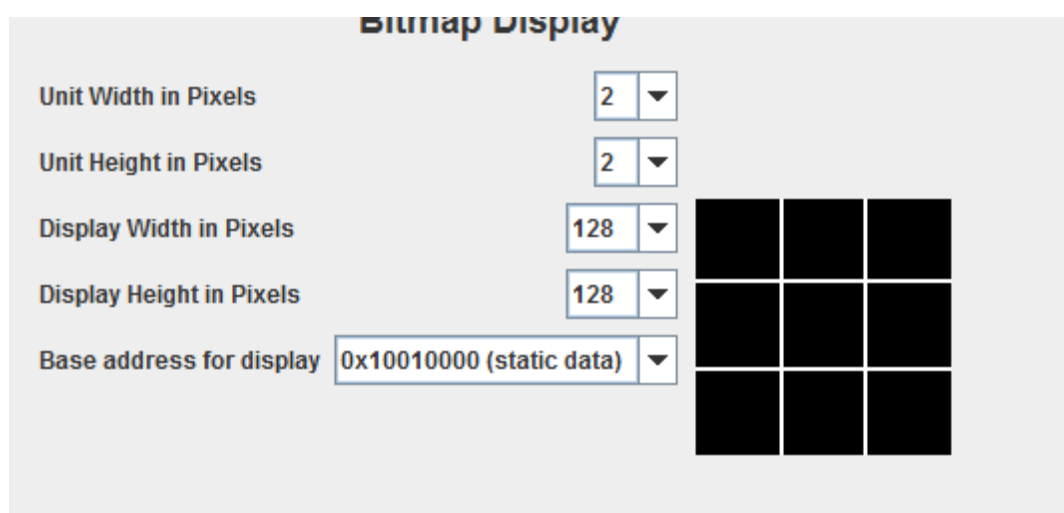
3.1. Desenhando o tabuleiro

Registradores usados: \$s2, \$s3, \$a3 e \$ra;

Esta é a primeira parte executada do código. Primeiramente são desenhadas as linhas horizontais. Para isso armazenamos no registrador \$s2 a posição do primeiro bit da linha, em \$s3 é armazenado o tamanho da linha e em \$a3 a cor. Após a inicialização desses registradores é feito um *jal*(Jump and Link) para um loop que começará armazenando a cor na posição \$s2 do bitmap, a partir daqui ele incrementará o endereço desse registrador em meio *byte* enquanto usa \$s3 como um contador. Dessa forma é desenhada uma linha horizontal de tamanho 64. Ao final do loop é feito o *jr*(Jump Register) para \$ra, voltando para a próxima instrução depois do *jal*. E por fim é feita uma nova inicialização dos registradores para uma nova posição de memória para assim desenhar a próxima linha horizontal.

Para desenhar as linhas verticais foi usado o mesmo processo das linhas horizontais. Entretanto, ao invés do endereço do registrador \$s2 ser incrementados em meio *byte*, é necessário incrementá-lo em 32 *bytes*, dessa forma o novo endereço será o *bit* logo abaixo do anterior. Assim, criamos as duas colunas do tabuleiro.

O tabuleiro foi a primeira parte feita durante o desenvolvimento do trabalho, e assim foi o que apresentou a maior dificuldade. O grupo ainda não estava acostumado a usar o bitmap display, isso dificultou bastante o andamento do projeto nesse período. Mesmo assim, após conseguirmos desenhar a primeira linha foi possível dar andamento as outras partes da implementação.

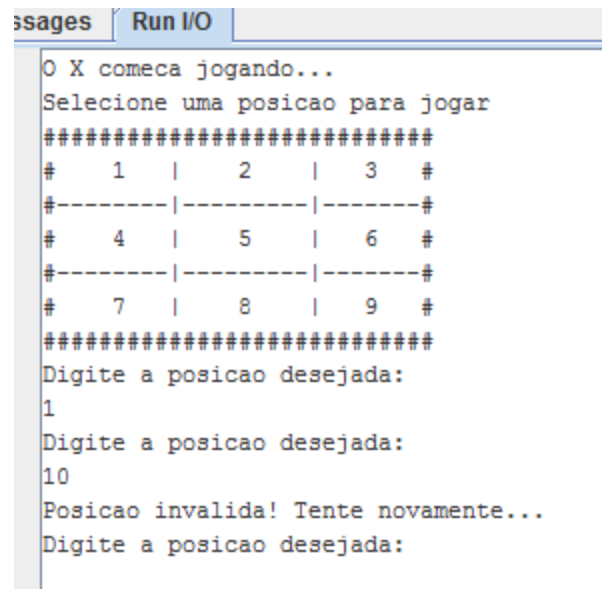


(Tabuleiro implementado pelo grupo; Fonte: Autoria própria).

3.2 Imprimindo o tutorial

Registradores usados: \$a0 e \$v0;

Nesta parte do código o grupo decidiu exibir um tutorial na entrada e saída padrão do MARS4_4. Nele é explicado como o jogo funciona, guiando o usuário para usar o jogo de maneira mais simples. Para exibir o tutorial criamos um total de 14 strings (m0 à m13), dentre as quais existem mensagens de erro, fim de jogo e qual foi o vencedor. Cada mensagem é armazenada em \$a0 e logo após é feito um *syscall* para exibir a mensagem ao usuário. Enquanto isso, é carregado o valor inteiro 4 em \$v0 indicando que será impresso uma string no sistema. Ao final é feito um *jump* para o label *depois_do_tutorial*, que se encontra no main.



```
Messages Run I/O
O X começa jogando...
Selecione uma posicao para jogar
#####
#  1  |  2  |  3  #
#-----|-----|-----#
#  4  |  5  |  6  #
#-----|-----|-----#
#  7  |  8  |  9  #
#####
Digite a posicao desejada:
1
Digite a posicao desejada:
10
Posicao invalida! Tente novamente...
Digite a posicao desejada:
```

(Exibição do tutorial implementado pelo grupo; Fonte: Autoria própria).

3.3 Iniciando a jogada

Registradores usados: \$t0, \$t1, \$t2, \$t3, \$t4, \$t5, \$t6, \$t5, \$t8, \$t9, \$a0, \$zero

Nesta parte do código é onde o jogador escolhe a jogada. No main, é armazenado no registrador \$a0 um valor inteiro correspondente a posição escolhida pelo usuário. Logo em seguida é feito um *jump* para o label *posicao*. Nesse label é verificado qual posição foi escolhida usando condicionais *beq* testando valores de 1 a 9, caso nenhum desses valores seja válido será feito um *jump* para o label *outro* onde será

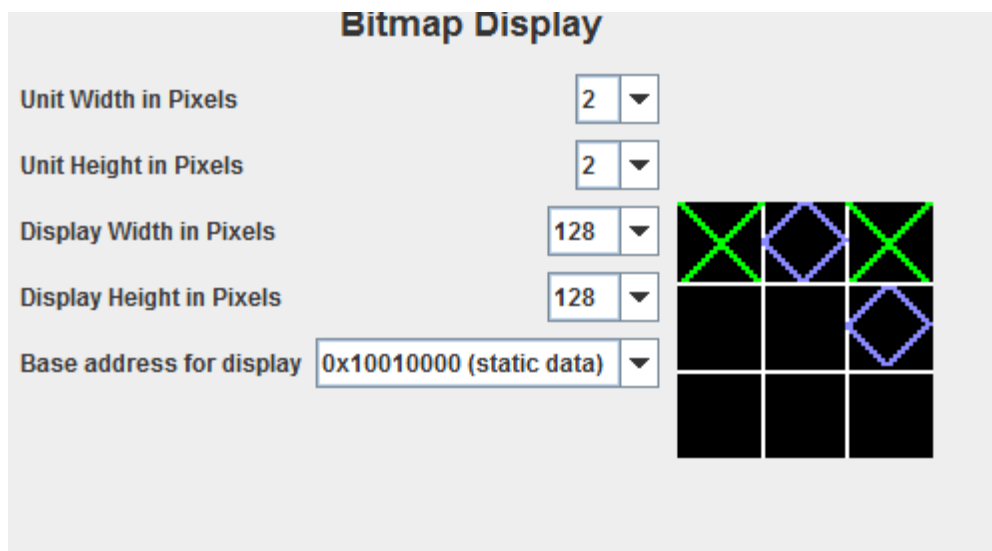
exibida uma mensagem de posição inválida e pedirá ao usuário para digitar novamente uma posição.

Se um dos valores válidos forem digitados, então haverá um salto para o label correspondente a cada número. O grupo usou os registradores de \$t0 a \$t9 para representar as posições do tabuleiro, de forma que eles armazenam zero quando não há nada naquela posição, ou seja, é uma posição válida. Para o valor um, o registrador estará armazenando um X e para o valor dois o registrador está armazenando um O. Então, após a jogada ser validada o registrador \$t9 é incrementado em um. Esse registrador é o responsável por contar o número de jogadas, ou seja, a cada jogada ele será incrementado até chegar a um limite de 9. Ao atingir esse limite o jogo será encerrado. Além disso, é armazenado em \$a0 a posição onde será desenhada a figura. Foram criadas nove posições, cada uma com o valor do primeiro pixel de algum quadrado do tabuleiro.

Em seguida, é necessário decidir qual figura será desenhada. Para isso, o registrador \$s7 é usado, ele começa com o valor zero o qual indica que será a vez de desenhar o X. Ao final da etapa de desenhar a figura, o valor de \$s7 será trocado entre 1 e 0. Por exemplo, quando o X é desenhado o valor de \$s7 passa a ser 0. Dessa forma é feito um desenho de cada figura alternadamente.

Após desenhar a figura, ainda no label referente a posição escolhida pelo jogador, é feita a verificação do número de jogadas. Como já dito antes, o limite é 9. Seguindo esses passos, quando a execução chegar a essa verificação e o registrador \$t9 tiver armazenado o valor 9, já implica que o jogo terminou sem vencedores. Logo, a execução pula para o label *end_posicao* que se encontra no main onde ele irá exibir a mensagem de empate e terminar o jogo.

Na etapa de iniciar a jogada se encontra praticamente toda a lógica do jogo. No desenvolvimento desta etapa o grupo não apresentou dificuldades e construiu ela de forma rápida e simples.



(Tabuleiro do jogo com algumas figuras desenhadas implementado pelo grupo; Fonte: Autoria própria)

3.4 Desenhando as figuras

Registradores usados: \$s2, \$a0, \$s3, \$a3, \$ra, \$s7;

Para desenhar o X, o grupo usou a mesma logica para desenhar as linhas e colunas. De certa forma, elas foram unidas. O endereço onde será pintado, que se encontra armazenado em \$s2 o próximo bit será incrementado em meio *byte*, passando para o bit logo à direita dele, e depois será incrementado em 32 *bytes*, passando para o bit logo abaixo. Isso é feito 20 vezes, usando o registrado \$s3 como contador, de forma que será desenhado uma parte do X. A segunda parte é desenhada da mesma forma, apenas mudando o endereço inicial, o qual é o mesmo usado na parte anterior acrescido de 84(inteiro), e ao invés de incrementar meio *byte* nos decrementamos.

O desenho do O é feito de maneira similar ao X, entretanto foi dividido em quatro etapas. O grupo decidiu que seria mais simples desenhar um quadrilátero, do que desenhar uma bola em si. As duas primeiras desenhavam da esquerda para direita, de forma que o endereço inicial é o ponto médio do quadrado do tabuleiro, que pode ser encontrado acrescentando 40(inteiro) ao endereço do ponto inicial. Começando do ponto médio, são usados os mesmos passos do desenho do X, mas apenas 11 vezes. Logo após, o registrador \$s2 possui o endereço do ponto médio de um lado adjacente ao lado inicial. Dessa forma, ao subtrair 84(inteiro) de \$s2 obtemos o endereço do ponto médio

do lado oposto. A partir daqui usamos a mesma função para desenhar o resto do quadrilátero.

3.5 Encontrando o vencedor

Registradores usado: \$t0, \$t1, \$t2, \$t3, \$t4, \$t5, \$t6, \$t5, \$t8, \$s2, \$s3, \$a3, \$zero, \$s, \$a0, \$v0.

Para encontrar o vencedor, o código analisa o tabuleiro imediatamente após cada jogada. Nessa análise, ocorrem ao todo 10 casos distintos.

Oito desses dez casos ocorrem quando um vencedor é encontrado. Três dessas possibilidades ocorrem quando uma linha vertical é formada por um mesmo jogador. Outras três possibilidades ocorrem quando uma linha horizontal é formada por um mesmo jogador, e as outras duas possibilidades ocorrem quando é formado linhas diagonais.

Para se verificar a formação dessas linhas, primeiramente são verificados através de comandos condicionais aninhados as primeiras posições dessas linhas, depois, se a posição ao lado é igual a essa primeira posição, e por último se a última posição é igual as outras duas. Quando enfim é confirmado que há um vencedor, o código desenha uma linha onde ocorreu a jogada vencedora e exibe o jogador vencedor. Caso alguma das condições de vitória não seja satisfeita, o código abandona o teste atual, e segue para o próximo teste.

Caso, ao final desses oito testes, o jogo ainda não tenha encontrado nenhum vencedor, e ainda existam posições livres onde é possível realizar uma jogada, o jogo simplesmente segue, e possibilita ao próximo jogador a oportunidade de realizar sua jogada.

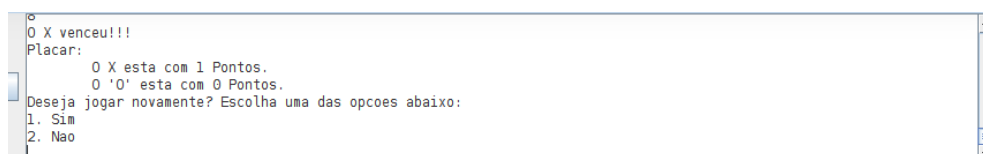
Quando o código encontra todas as posições jogáveis preenchidas, sem que tenha um vencedor, um aviso é exibido aos jogadores informando que o jogo “deu velha”.

3.6 Placar

Registradores utilizados: \$a1, \$a2, \$a0, \$v0.

Se for encontrar um vencedor, é incrementado o placar que está armazenado nos registradores \$a1 e \$a2 para os jogadores ‘X’ e ‘O’, respectivamente. Se o ‘X’ ganhar o \$a1 sofre incremento de 1. Se o ‘O’ ganhar o \$a2 sofre incremento de 1. Após o incremento é impresso o placar atualizado do jogo.

Se não for encontrado um vencedor o placar não sofre alteração e é impresso.



```
O X venceu!!!
Placar:
    O X esta com 1 Pontos.
    O 'O' esta com 0 Pontos.
Deseja jogar novamente? Escolha uma das opcoes abaixo:
1. Sim
2. Nao
```

(Placar ao ser impresso juntamente com o menu de jogar novamente implementado pelo grupo; Fonte: Autoria própria)

3.7 Jogar novamente

Registradores utilizados: \$t0, \$t1, \$t2, \$t3, \$t4, \$t5, \$t6, \$t5, \$t8, \$t9, \$s7, \$a0, \$v0.

Após imprimir o placar o jogador responderá se quer jogar novamente ou não. Isso é feito através de um menu de duas opções exibido para o jogador.

Se o jogador pressionar 1 (resposta afirmativa para a pergunta do menu “Deseja jogar novamente?”) serão resetados os seguintes registradores: \$t0, \$t1, \$t2, \$t3, \$t4, \$t5, \$t6, \$t5, \$t8, \$t9, \$s7. Do \$t0 até \$t8 são os registradores das posições que armazenam se tem um ‘X’ ou um ‘O’ no local, então eles são resetados para 0. O \$t9 armazena a quantidade de jogadas realizadas, então ele é resetado para 0. O \$s7 armazena de qual jogador é a vez, então ele é resetado para 0 representando que é a vez do ‘X’. O bitmap também será resetado, sendo pintado todo de preto para receber novamente o desenho das linhas e figuras.

Se o jogador pressionar 2 (resposta negativa para a pergunta do menu “Deseja jogar novamente?”) o jogo termina.

Se o jogador pressionar algum número que não seja nenhuma das opções acima o jogador recebe um aviso de opção inválida e é perguntado novamente.

4. CONCLUSÃO

Com a conclusão do projeto, o grupo obteve sucesso em implementar todos os requisitos exigidos na descrição do trabalho. Além disso, o projeto se tornou uma

atividade construtora, de forma que incitou o conhecimento dos integrantes do grupo para os assuntos relacionados a disciplina. Mesmo percebendo que o assembly MIPS é uma linguagem difícil de ser manuseada, seja na escolha de como os registradores e a memória será usada; seja na documentação o grupo concordou que é uma linguagem interessante e que serviu para entender melhor tópicos mostrados em aula.

5. REFERÊNCIAS

- FIGUEIREDO, Aline; SANTANA, Cristiano; GODOY, Jesileny. Jogo da Idosa. Disponível em: <<https://code.google.com/p/bsimips/wiki/IdosaGame>> Acesso em 1 de outubro.
- Material da disciplina