River Raid RISC-V

Pedro Henrique de Brito Agnes[1*] Luan Pignata Caldas[2†] Rafael Oliveira de Souza[3‡]
Gabriel Guimaràes Almeida de Castro[4§] Felipe da Graça Costa[5]

[1]Universidade de Brasília, Departmento de Ciência da Computação, Brasil

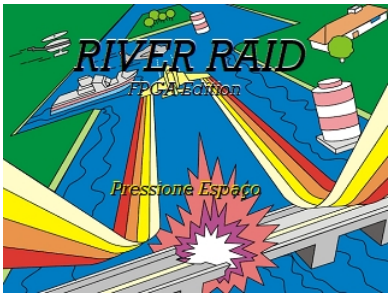

Figura 1: Tela inicial

RESUMO

O documento trata-se de um relatório do desenvolvimento de uma nova versão do jogo do Atari 2600, *River Raid*. Para o desenvolvimento em assembly RV32-IMF, foi utilizado o software *RISC-V assembler and Runtime Simulator (RARS)*, que permite a simulação de todo o processo de execução do código desenvolvido na linguagem e o transforma em linguagem de máquina. Desta forma, ele foi utilizado para exportar o arquivo de extensão `.mif` com o código em linguagem de máquina para o software Quartus, utilizado para a implementação do processador pipeline RISC-V, que vai passar o programa para o processador programado na FPGA onde se é conectado um monitor, teclado, joystick e dispositivo reprodutor de som.

Palavras-chave: River raid, jogo, RISC-V, RV32IMF, joystick.

1 INTRODUÇÃO

River Raid é um jogo clássico do estilo arcade com lançamento inicial no ano de 1982 para o console Atari 2600. Trata-se de um video-game em que o jogador controla um avião amarelo e tem como objetivo desviar dos obstáculos e destruir os inimigos para ganhar pontos e sobreviver o máximo que conseguir.

Neste projeto, o jogo foi recriado em assembly RV32IMF para ter seu funcionamento na FPGA DE1-Soc e permitir a jogabilidade com joystick e teclado com o vídeo sendo mostrado em um monitor com entrada VGA e audio sintetizado pela saída p2 da placa. Para isso, utilizamos o processador RISC-V com organização pipeline disponível no arquivo `.qar` do Quartus, que permite carregar o algoritmo e os dados convertidos pelo *RARS* diretamente na memória instruções e dados, respectivamente.

A nova versão do jogo possui quase todas as características do original, porém, devido a limitações de memória na FPGA, algumas características ficaram de fora, como o mapa dinâmico. Mas esta versão possui algumas características extras, como inimigos novos que se comportam de maneiras diferentes e outro tipo de tiro, além de ter dificuldade crescente.

2 FUNDAMENTAÇÃO TEÓRICA E TÉCNICA

Para o desenvolvimento do projeto, foi necessário conhecimento sobre a arquitetura RISC-V e sua *ISA* [1], que foi adquirido ao longo do curso de Organização e Arquitetura de Computadores na Universidade de Brasília em que foram desenvolvidos outros projetos envolvendo a utilização da ISA RV32IMF e a organização do processador RISC-V.

Os processadores são desenhados de forma que todas as instruções são codificadas em bits. Todos os dados são representados em bits. Programas são armazenados na memória para serem lidos da mesma forma que os dados. Os Ciclos de busca e execução são criados de forma que as instruções são buscadas na memória do endereço armazenado no registrador PC(Program Counter) e colocadas no registrador IR(Instruction Register) o Bits do registrador IR controlam as ações subsequentes necessárias à execução da instrução.

Como trabalho correlato, foi feito um algoritmo em assembly do RISC-V para sortear pontos aleatórios no bitmap display, ordená-los para que não se cruzem, traçar linhas

*e-mail: pedenite@gmail.com
†e-mail: luanpignat@gmail.com
‡e-mail: rafaelsoec@gmail.com
§e-mail: gabriel1997.castro@gmail.com

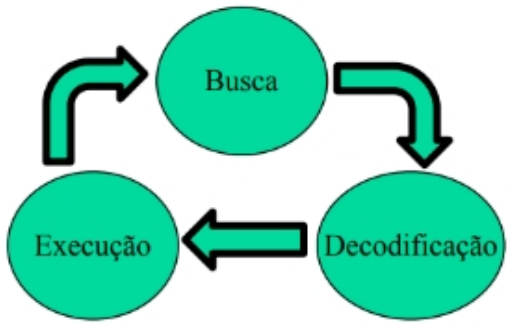

Figura 2: Ciclo de Instrução.

entre os pontos na ordem em que se encontram, pintar o polígono formado e imprimir na tela a área deste polígono. Este projeto permitiu uma melhor compreensão no desenvolvimento de algoritmos para o RISC-V.

Em outro projeto, foi requisitado que se adicionasse 6 novos registradores nos processadores RISC-V uniciclo, multiciclo e pipeline. Os três primeiros contendo os 32 bits menos significativos do contador de ciclos, de tempo e de instruções, respectivamente, enquanto os outros registradores armazenariam os 32 bits mais significativos de cada um deles. Isso permitiu um melhor entendimento de cada organização da arquitetura RISC-V.

3 METODOLOGIA

No projeto desenvolvido, foram implementadas diversas funções para o correto funcionamento do jogo, que apresenta tela inicial, movimentações e animações de acordo com o jogo original, inimigos, vidas, pontuação, entre outros. A tela inicial é carregada diretamente na `frame 0` do processador devido à limitação da memória, portanto não aparece no *RARS*. Durante a execução do jogo, após o usuário pressionar a barra de espaço, é inicialmente impresso o menu na tela e os valores dos registradores são inicializados.

Após a inicialização dos valores, entra no segmento chamado de *gameloop* que chama todas as funções que precisam ser atualizadas a cada *frame*, como a função que imprime o mapa na tela, que consiste em preencher a tela com duas cores definidas que representam a água e a terra que é usada para ”limpar”a tela no início do *frame* depois de todos os objetos terem sido impressos no *frame* anterior.

Ao final do *gameloop*, temos uma chamada do sistema usando `ecall 132` para pausar a execução por 20 milissegundos. Isso permite que o jogador tenha um tempo para enxergar os objetos na tela e limita a velocidade de execução a 50 *fps*, a depender do desempenho do processador. A seguir será descrito cada etapa do projeto.

3.1 Imagens no Bitmap

Um dos procedimentos mais importantes, que foi utilizada a todo momento no projeto é a função chamada `Set Pixels`, que foi utilizada para carregar qualquer imagem no formato `.data` no *bitmap* de acordo com a posição indicada nos parâmetros e suas dimensões indicadas no arquivo. Posteriormente, foi criado o procedimento `Set Reverse Pixels`, que faz o mesmo que o anterior, mas carrega a imagem invertida da esquerda para a direita.

Para o funcionamento dos procedimentos descritos acima, são enviados como argumento, no registrador `a0`, o endereço da imagem na memória de dados, a posição y no `a1` e a posição x no `a2`. Em seguida, é calculado o endereço dessas posições no *bitmap* usando o endereço base `0xFF000000` e assim, a imagem é pintada no endereço desejado.

3.2 Jogador

Figura 3: Jogador - avião.

O jogador controla um avião amarelo que pode se movimentar para os lados a uma velocidade de 3 *pixels* por segundo para desviar dos obstáculos e atirar para destruir os inimigos. Para a representação do avião na tela, foi utilizada a função descrita na secção 3.1, que tem sua posição atualizada por outro segmento de código conforme a tecla pressionada ou a direção do joystick, mudando a posição na tela no *frame* seguinte. O avião possui animações que funcionam de forma que é carregado um arquivo `.data` diferente dependendo do estado do *joystick*, em que permite uma melhor sensação de movimento para os lados.

O jogador possui 3 vidas extras administradas pelo procedimento `Controla Vidas`, permitindo-o após morrer, caso tenha vida extra, pressionar espaço para usar uma vida e continuar jogando de onde parou. Caso não possua vidas extras, a função realizará um `jump` para o segmento chamado *game over*, que realizará a chamada do sistema `ecall 10`, parando a execução.

3.3 Controle de Inimigos

Existem dois procedimentos feitos para controlar os inimigos, a geração de posto de abastecimento e a dificuldade de acordo com a pontuação do jogador, que são chamados de `Enemy Behavior` e `Enemy Spawn`. Na função `Enemy Spawn`, é utilizada uma chamada do sistema para geração de números aleatórios, que vai definir o tipo do inimigo a ser gerado. Em seguida, vai usar o número aleatório para gerar a posição do inimigo na tela e por fim, alocar espaço para duas *words* na *stack pointer*, uma para o tipo do inimigo e a direção que ele se movimentará e outra para as coordenadas x e y do inimigo na tela.

Já na função `Enemy Behavior`, é buscado na pilha o tipo do inimigo e para cada um, tem-se um comportamento diferente definido pelos diferentes procedimentos criados dentro deste. Desta forma, é controlada a movimentação do inimigo de acordo com sua velocidade definida, a direção

em que ele deve seguir, como ele é destruído de acordo com a posição do tiro ou do jogador, quantos pontos ele vale, entre outros. O processo é repetido várias vezes até que toda a pilha seja percorrida.

Quando o inimigo é destruído, ou seja, um tiro o acertou ou ele atingiu o jogador ou chegou aos limites do mapa, é chamada uma rotina criada que realizará um *swap* dele com o objeto no topo da pilha se ele não for o elemento do topo e desaloca o espaço de duas words que era ocupado por ele.

3.3.1 Navio

Figura 4: Inimigo 1.

O primeiro inimigo, mostrado na figura acima, é o mesmo existente no jogo original. Ele possui dimensões 32 x 8 *pixels* e se destruído, o jogador ganha 30 pontos. Seu comportamento funciona de forma que ele vai se locomover para um dos lados com velocidade de 1 *pixel* por *frame* e para baixo na mesma velocidade. Se sua posição x somada à sua largura alcançar a parede, o bit que define a direção que ele se moverá é invertido e ele passa a seguir a direção contrária e será pintado na tela com a função `Set Reverse Pixels` mencionada acima 3.1.

3.3.2 Helicóptero

Figura 5: Inimigo 2.

O inimigo mostrado acima também está presente no jogo original. Ele possui dimensões 16 x 10 *pixels* e vale 60 pontos. O helicóptero se comporta de forma parecida com o navio, em que ele se moverá para os lados em uma velocidade de 2 *pixels* por *frame* e para baixo de 1 *pixel* e se chegar à parede, sua imagem inverte e ele passa a seguir a direção contrária.

Este inimigo possui animações, em que será carregada uma imagem diferente entre as duas existentes dele a cada *frame* de forma a passar sensação de movimento da hélice.

3.3.3 Avião

Figura 6: Inimigo 3.

O avião inimigo pode ser encontrado também no *River Raid* original, possui dimensões de 16 x 6 *pixels* e dá ao jogador 100 pontos. Ao ser gerado, ele terá sua posição y gerada aleatóriamente dentro de um intervalo, enquanto sua posição x inicial é 304, que o deixará no limite direito da tela.

Assim, seu comportamento sempre será de se locomover da direita para a esquerda com velocidade de 4 *pixels* por *frame* e para baixo em uma mesma velocidade que os inimigos anteriores.

3.3.4 Nave Alienígena

Figura 7: Inimigo 4.

Este inimigo foi criado para esta nova versão do *River Raid* e possui dimensões de 24 x 10 *pixels* e tem o valor de 150 pontos. Assim como o avião 3.3.3, ele vai se movimentar apenas para o lado e 1 *pixel* para baixo por *frame* ignorando as paredes até desaparecer chegando ao limite lateral ou vertical ou até sendo destruído. A diferença dele é que ele tem sua posição x inicial à esquerda e se movimenta à direita com velocidade de 5 *pixels* por *frame*.

O inimigo 4 só começa a aparecer a partir do momento em que o jogador obtiver pelo menos 1000 pontos.

3.3.5 Barco

Figura 8: Inimigo 5.

O barco é um inimigo novo que possui dimensões de 14 x 7 *pixels* e dá ao jogador 90 pontos. Ele se comporta de forma parecida com o navio, trocando de direção ao alcançar as paredes, porém ele é mais rápido, tendo velocidade de 3 *pixels* por *frame* e se movimentando sempre para baixo assim como os outros inimigos à mesma velocidade.

O inimigo 5 começa a aparecer apenas quando o jogador obtiver, pelo menos, 2000 pontos.

3.3.6 Porta-Aviões

Figura 9: Inimigo 6.

O inimigo final é um porta-aviões, que foi criado com o intuito de surpreender pelo seu tamanho, tendo dimensões de 20 x 40 pixels e dá ao jogador a quantidade de 500 pontos. Ele apresenta o comportamento diferente dos outros inimigos, de forma que ele se movimenta apenas para baixo na velocidade de 3 *pixels* por *frame*.

O inimigo 6 começa a aparecer apenas quando o jogador obtiver, pelo menos, 5000 pontos.

3.4 Combustível

Figura 10: Posto de abastecimento.

Assim como no jogo original, esta versão possui combustível, que, inicialmente será atribuído o valor 2000 a um registrador salvo que representará a quantidade de combustível restante. A cada frame, um procedimento é chamado para subtrair 1 do registrador do combustível, o que permite que ele dure por, aproximadamente, 40 segundos.

Para recuperar o combustível, o jogador deve passar por cima do objeto mostrado na imagem acima, que é o posto. Com ele, o jogador recupera 20 de combustível por segundo, mas caso seja identificado um tiro na mesma posição que ele, é destruído e o jogador ganha 80 pontos.

Diferente do jogo original, que havia uma barra marcando a quantidade de combustível, a quantidade de combustível restante é calculada pelo procedimento e mostrada em porcentagem na tela.

3.5 Dificuldade

No início do jogo, o jogador possuirá 0 pontos, portanto a dificuldade do jogo estará definida a mais fácil em que só serão gerados os 3 primeiros tipos de inimigos 3.3 e o posto para abastecer o combustível, sendo apenas 1 deles gerado a cada 60 *frames*, tendo ainda a possibilidade de não gerar nenhum inimigo de acordo com o número randômico gerado. Porém, como o jogo possui dificuldade crescente, ao chegar em 500 pontos, os objetos serão gerados a partir do momento, a cada 40 *frames*.

Na marca de 1000 pontos, o inimigo 4 passa a ter a possibilidade de ser gerado e aos 2000 pontos, o inimigo 5, além de agora, os objetos surgirem a cada 20 *frames*. Por fim, isso se mantém constante até a marca dos 5000 pontos, que permitirá que o inimigo final seja gerado e assim, facilitar o alcance da marca final de 15000 pontos que pode gerar 1 objeto a cada 10 *frames*.

3.6 Tiro

Como mencionado anteriormente, o jogador pode atirar para derrotar os inimigos, bastando pressionar a barra de espaço. Desta forma, as coodenadas do tiro são armazenadas em um registrador salvo para a utilização pela função que controla os inimigos e pela função que atualiza a posição do tiro, que se movimenta para cima a 5 *pixels* por *frame* e se chegar ao topo da tela ou atingir um inimigo, é destruído, que para isso, foi definido que se o valor no registrador da posição do tiro for 300, não o imprime na tela.

Assim como no jogo original, foi feito nesta versão com que o jogador só possa atirar 1 tiro de cada vez. Então será possível atirar apenas quando o tiro anterior for destruído, seja por acertar um inimigo ou alcançar o topo da tela.

3.7 Joystick

Para a movimentação do jogador, foi utilizado um controle analógico conectado à interface com conversor analógico-digital da *FPGA*. Ela possui um pino terra e um de 5V, além de outros 8 pinos para serem usados para sinais analógicos. Para o projeto, foi usado o pino `CH0` para a posição x do joystick e o `CH1` para a posição y.

Deste modo, os sinais são convertidos em 12 bits e se tornam acessíveis no endereço de memória `0xFF200200`, que é acessado pelo procedimento chamado de `Get Axis`, que vai verificar o valor nesse endereço e comparar com um valor definido para assim, fazer com que o avião esteja parado ou movimentando para os lados.

3.8 Teclado

O teclado foi utilizado para permitir ao jogador atirar com a barra de espaço e, se desejar, movimentar com ”a”e ”d”, porém não recomendado. Para isso, foi criado um procedimento que verifica a cada *frame* se há alguma tecla pressionada na *MMIO* e compara o *scancode* com o código ascii da tecla desejada.

3.9 Música e Efeitos Sonoros

O processador *risc-v* usado para executar o jogo possui um sintetizador de áudio *MIDI*. O instrumento implementado no processador é o piano.

Os serviços implementados são *MidiOut*, *MidiOutSync*, *Sleep*. O *MidiOut* sintetiza uma nota com uma duração e volume, o *MidiOutSync* faz o mesmo mas pausa o programa até a execução e *Sleep* pausa a execução do programa.

A música implementada usou apenas *MidiOut* já que a música toca enquanto o jogo roda. Além da música também foi implementado efeitos sonoros para os tiros com notas agudas, explosões com notas mais graves e um efeito de *game over*.

O *MidiSyncOut* foi usado apenas na hora dos efeitos de *game over*.

3.10 Algumas imagens do Jogo

Nesta imagem pode ser visto em o jogo em uma dificuldade média com vários inimigos ao redor e o efeito de mostrando a nave em movimento.

Figura 11: Início do jogo.

4 Resultados Obtidos

Figura 12: Início do jogo.

Ao analisar os resultados obtidos neste trabalho, sem dúvida o mais importante foi analisar o compromisso do projeto da arquitetura RISC-V e como a simplicidade das instruções são importantes na curva de aprendizado sobre o processador foram fundamentais na atualização do *River raid*.

Outra característica importante a ser destacada é a maneira como o desenvolvimento em uma linguagem de mais baixo nível requer muita atenção, principalmente no gerenciamento de memória dos dispositivos.

Link do vídeo: https://youtu.be/67Au6zkLdYo

5 Conclusões

Neste projeto pôde-se concluir que a filosofia do projeto RSIC-V é realmente bastante efetiva na construção de projetos mais complexos como o desenvolvimento de um jogo. A simplicidade favorece a regularidade.O menor é (quase sempre) mais rápido. Um bom projeto demanda compromissos. E é muito melhor tornar uma instrução que é usada 90% do tempo 10% mais rápida do que fazer com que uma instrução usada em 10% das vezes torne-se 90% mais rápida, regra que é baseada na ”Lei de Amdahl”.

O projeto possibilitou também aos membros do grupo uma visão mais global do comportamento do processador na arquitetura RISC- V, as etapas que são executadas em sequência e perceber a importância de cada processo para a execução de uma tarefa por completo.

Todo o conhecimento prévio sobre computação foi, exaustivamente, testado no desenvolvimento deste projeto desde o simples processo de atribuição de variáveis ao gerenciamento de estrutura de dados

Referências

[1] D. A. P. e John L. Hennessy. *Computer Organization and Design: The Hadware/Software Interface*. Morgan Kaufmann Publishers, risc-v edition, 2017.