Estradinha

Gabriel de Castro Dias Daniel da Cunha Pereira Luz

University of Brasília, Dept. of Computer Science, Brazil

Abstract

Estradinha é uma releitura do jogo Road Fighter feito em Assembly RISC-V utilizando o FPGRARS como emulador.O projeto foi feito para a disciplina de Introdução a Sistemas Computacionais.RISC-V é uma ISA (Instruction Set Architecture ou arquitetura de conjunto de instruções) desenvolvida originalmente na Universidade da California em Berkeley com o objetivo de ajudar os alunos da materia à aprenderem a linguagem de programação de baixo nível Assembly.

Palavras Chaves:Assembly,Road Fighter, RISC-V, FPGRARS, ISA,linguagem de programação

1 Introdução

Road Fighter é um jogo de arcade desenvolvido pela Konami e lançado em 1984. Também foi o primeiro jogo de corrida de carros da Konami. O objetivo do game é chegar ate o final da fase enquanto enfrenta varios obstáculos. Utilizando puramente de suas habilidades, o jogador deve desviar, prever movimentos padronizados e gerenciar sua quantidade de gasolina disponivel. O jogo termina quando o jogador fica incapacitado por falta de combustível ou passe a linha de chegada. Este artigo tem como objetivo documentar como foi feita a implementação do Road Fighter em assembly RISC-V, descrevendo a metodologia utilizada, resultados obtidos e conclusões finais.

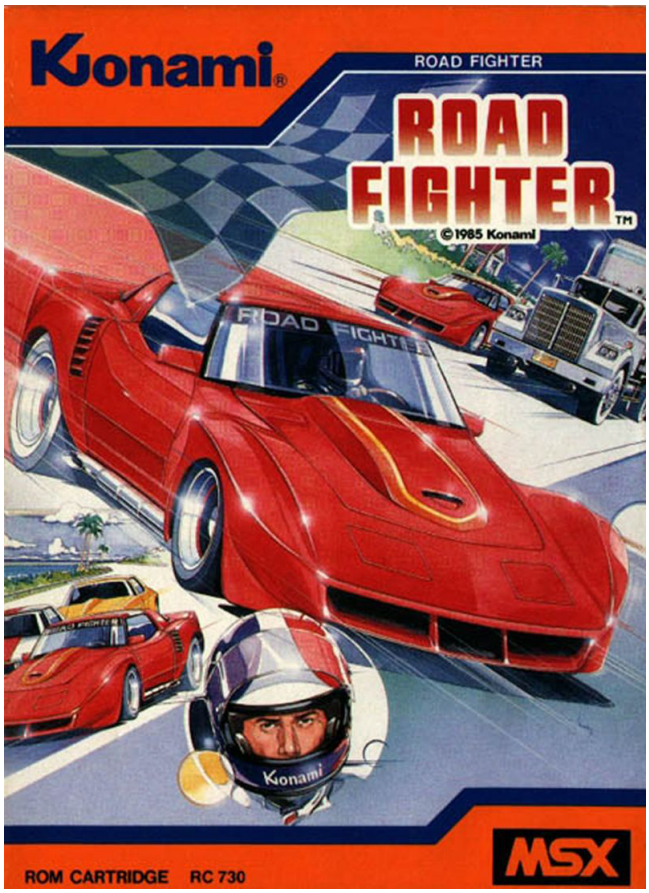


Figure 1: Capa do jogo Road Fighter

Na seção 2 será apresentada a metodologia utilizada. A seção 3 apresenta os resultados obtidos. A seção 4 conclui este trabalho.

2 Metodologia

O projeto monta no RARS, mas utiliza primariamente o FPGRARS (Fast Pretty Good RISC-V Assembly Rendering System) por ser mais rápido e um pouco mais tolerante com erros.

2.1 Reuniões

O projeto começou no dia x e terminou no dia 05/05, durante esse período fizemos reuniões diárias para comentar ideias e entender a lógica por trás dos códigos, fizemos as reuniões de chamada pelo Discord, discursões fora do horário de reunião pelo grupo do whatsapp e hospedamos nosso código no site do GitHub para o gerenciamento de arquivos e alterações feitas pelos membros, tornando assim o trabalho em equipe muito mais organizado e dinâmico.

2.2 Menu

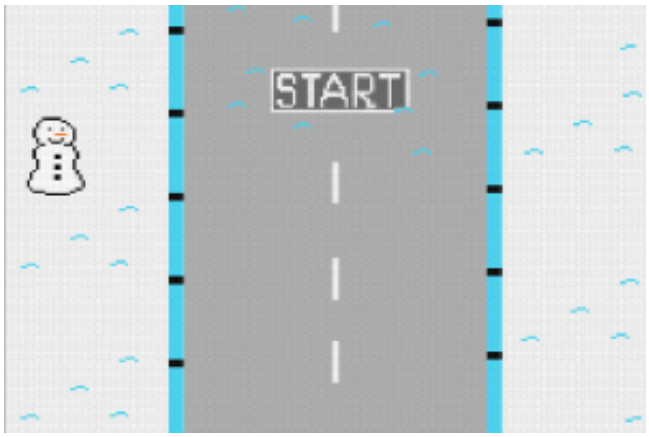
O menu foi o primeiro tópico da produção do game, é um dos itens mais fáceis, porém sem dúvidas o mais importante, além de deixar os membros com gostinho de quero mais ao verem o seu trabalho printado em uma tela apenas usando assembly. O menu foi feito por imagens editadas pelo paint.net, e logo em seguida transformadas para .DATA. Com a tela printada utilizamos os sprites,MIDI,teclado MMIO e o swap de frames para desenvolver uma opção de menu selecionável e com efeitos sonoros a cada pressionada das teclas.

2.3 Música

Para a música utilizamos o aplicativo 'AnthemScore' para obtermos as notas e durações utilizadas no arquivo, com as notas em mãos, desenvolvemos um programa python para agilizar o processo de migração para o Rars. O método que utilizamos é composto de nota, duracao,e um digito pertencente a [0,1], se for 0, utilizamos a syscall 31, que não espera a duração da nota passada acabar, caso seja 1 chamaremos a syscall 33, que começa apenas quando a anterior terminar. Dessa forma, conseguimos chamar mais de uma nota por vez, ampliando as nossas opções de músicas e efeitos.

2.4 Mapas

No total foram feitos 2 mapas, um com tema x e outro com tema y. Cada mapa foi editado pixel a pixel pelos membros do grupo, assim deixando os mapas mais agradáveis e limpos.

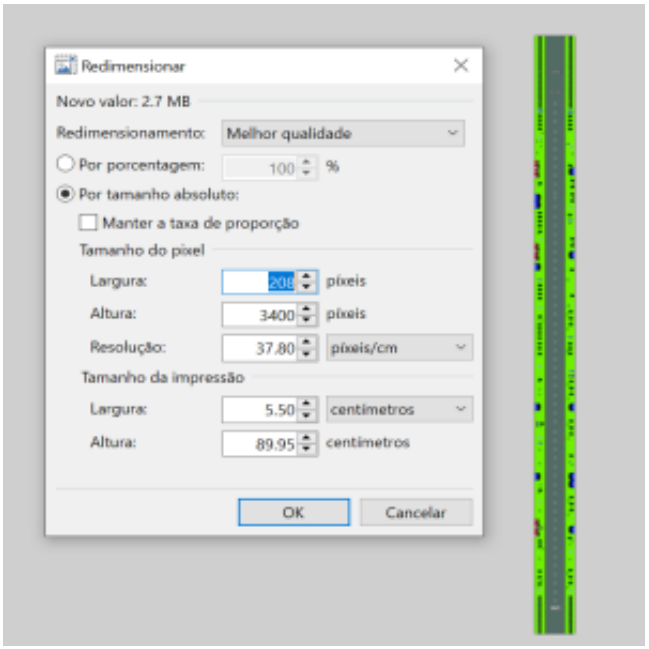


2.5 Sprites

Os sprites foram baixados na internet e depois editados no paint.net, utilizamos alguns cálculos básicos sobre resolução para providenciar um tamanho adequado em relação ao mapa, em seguida desenvolvemos um código para printar os sprites de acordo com as coordenadas dos píxeis, assim obtivemos mais controle sobre os registradores, o que acabou facilitando nas manipulações nos prints de sprites.

2.6 Movimentação

Com a tela e os sprites printados, um próximo passo muito importante era a movimentação,para o carro criamos um sistema que printa o sprite no frame x. inverte e depois printa no outro frame, com a gerenciamento das coordenadas juntamente com as swaps de frames e comandos do teclado MMIO conseguimos manipular o print do carro de acordo com as verificações de teclas hexadecimais pressionadas pelo usuário. Já para o mapa decidimos criar um arquivo com as dimensões 208x3400, e com a utilizacao de cálculos matemáticos conseguimos medir as distâncias necessárias para printar nos 320x240 do Rars sem ter nenhuma influência do tamanho desproporcional do arquivo original. A movimentação do mapa foi feita subtraindo pouco a pouco do arquivo cada vez que a tecla de movimentação era pressionada, como 208 píxeis era menor que 320 píxeis do Rars, a tela ficaria torta e desproporcional, mas usamos esse 'problema' ao nosso favor e criamos o arquivo sem as bordas pretas, calculamos as dimensões e conseguimos ter o arquivo original,porém economizando muito mais na hora do print, o que deixou nosso código muito mais otimizado comparado as tentativas anteriores.




```
aumentarVelocidade:
        # s1 = velocidade pista
        li t0,6
        beq s1,t0,printRepeat

        addi s1,s1,2
        j printRepeat

moveCarroEsquerda:
        addi s0,s0,-2
        j printRepeat

moveCarroDireita:
        addi s0,s0,2
        j printRepeat

showFrameAtual:
        li t0,0xFF200604
        sw s3,0(t0)
        ret
```

```
.text
# s1 = velocidade pista
# s2 = posição pista
# s3 = frame atual

setInicioPista:
    li s2,3160          # s2 = posição pista

printPistaAtual:
    li t0,0xFF0         # t0 = 0x00000FF0
    add t0,t0,s3        # t0 = 0x00000FF0 ou 0x00000FF1 dependendo do frame
    slli t0,t0,20       # 0x00000FF0 -> 0xFF000000 ou 0x00000FF1 -> 0xFF100000

    la a0,graminha_teste2       # endereço da imagem
    li t3,0xC300        # contador de pixeis (tela)

    sub s2,s2,s1        # altura - velocidade atual

    li t2,208           # 208 largura e contador
    mul t1,s2,t2        # altura X posição = posição a ser mostrada (t1)

    add a0,a0,t1        # endereço imagem - posição

    addi t0,t0,32       # endereço print + 30

    addi a0,a0,8        # primeiros pixeis
```

```
    # loop de impressao: salva os pixels da imagem na memoria do monitor, de 4 em 4 pixels.
loop_printColunaPista:
    blez t2,loop_printLinhaPista       # se a quantidade de colunas == contador de colunas -> próxima linha
    lw t5,0(a0)                # coloca o a0(imagem) no t5
    sw t5,0(t0)                # guarda t5 no t0(endereço de print)
    addi t0,t0,4               # +4 pixeis ao endereço print
    addi a0,a0,4               # +4 pixeis ao endereço da imagem
    addi t2,t2,-4              # -4 pro contador
    addi t3,t3,-4              # -4 pixeis no contador total
    j loop_printColunaPista
loop_printLinhaPista:
    li t2,208                  # reseta contador
    addi t0,t0,112             # +60 pixeis == próxima linha ao endereço print
    blez t3,exit_printPista    # se o contador de pixeis == 0 -> fim
    j loop_printColunaPista

exit_printPista:
    ret
```

3 Resultados Obtidos

3.1 Problemas

- Definitivamente a linguagem Assembly foi o maior problema, nunca havíamos passado por algo parecido, foi uma experiência totalmente nova,registradores, memórias, word, byte,MIDI. Apesar de ser desafiador foi bem interessante, mantivemos o nosso interesse em aprender e continuamos até o fim.
- Os detalhes foram bem desafiadores, cada errinho pode quebrar o código todo, seja por registradores mal-utilizados, tamanhos de imagens não serem múltiplos de 4, descobrir o tamanho certo para o nosso sistema de movimentação, redesenhar os mapas píxel a píxel para não perder qualidade e etc.

3.2 Resultados

- Não estávamos com muitas expectativas, mas o resultado final nos surpreendeu, conseguimos fazer bastante coisa.

4 Conclusão

- Este trabalho acrescentou muita experiência para os membros da equipe,além de desenvolvimento social em grupo, cada um foi muito competente e o resultado foi bem satisfatório.

5 References

Wikipedia.com Aulas do Professor Lamar Vídeo aulas dos Monitores: Eduardo, Leandro, Ruan