

**UNIVERSIDADE FEDERAL DA BAHIA**

**DEPARTAMENTO DE CIÊNCIA DE COMPUTAÇÃO**

**LUCAS MENEZES PEREIRA**

**MATHEUS OLIVER DE CARVALHO CERQUEIRA**

**RELATÓRIO DE ANÁLISE DE JOGO CONSTRUÍDO EM ASSEMBLY ARQUITETURA X86**

Salvador

2017

1. **APRESENTAÇÃO**

*Assembly* é uma notação legível por humanos para o código de máquina, utilizada para programar códigos entendidos por dispositivos computacionais. O código de máquina torna-se legível pela substituição dos valores em bruto por símbolos chamados mnemónicos. Cada mnemônico é associado a uma operação do computador.

A Linguagem *Assembly* é muito versátil, sendo constituinte de software básico e muito variada, devido às várias arquiteturas disponíveis, criadas por empresas como fabricantes de Microcontroladores, Microprocessadores, e outros Chips de Circuitos Integrados para garantir a exclusividade de seu Hardware através do Software.

Nesse cenário de diversas formas de *Assembly*, A arquitetura x86, da Intel® é uma das mais utilizadas, sendo possivelmente utilizada para criar, também, softwares de Jogos.

Será desenvolvida, portanto, neste relatório, uma análise do código *Assembly* de uma versão do clássico da década de 80, *Space Invaders*™, recriado por fãs, sem fins lucrativos.

O jogo, originalmente lançado para *Arcades*, consiste de uma nave, cujos canhões atiram sobre uma horda de alienígenas, que se movimentam horizontalmente, descendo uma linha da tela quando chegam à lateral do monitor. A versão escolhida não utiliza imagens *bitmap* similares, mas caracteres, reproduzindo a mesma mecânica (funcionamento do jogo definido pelo código escrito por desenvolvedores).

1. **MATERIAIS UTILIZADOS**
   1. **Sistema Operacional:**
   * Windows 10 64 bits, com processador em x64;
   * Linux Lubuntu 32 bits, 1,90GB de memória RAM.
   1. **Hardware:**
   * 4GB de memória RAM (Utilizável: 3,89GB);
   * Intel Core i3-3010M CPU @2.40GHz.
   1. **Software:**

* Oracle VM VirtualBox;
* Terminal (Linux);
* Jogo *Space Invaders*™(Coleção de arquivos);
* Nasm
* QEmu.

1. **OBJETIVOS**
   1. **Geral**

A partir do jogo escolhido, analisar como o jogo foi construído, e como a linguagem *Assembly* é utilizada para garantir o pleno funcionamento do jogo, associando cada bloco de código a uma ação ou recurso específico.

* 1. **Específicos**
* Verificar a execução do programa (garantir o funcionamento e validação do código);
* Analisar os códigos que compõem o jogo, separando os trechos de acordo com as ações que representam.

1. **METODOLOGIA**

Aqui são mencionados e discutido quais itens e como serão analisados neste relatório. As medidas de pré-execução serão esclarecidas na próxima seção. Dada a situação do jogo executado, os itens seguintes serão considerados na análise.

* 1. **Mecânica do Jogo**

A partir da apresentação do jogo, em que foi explicitado o conceito do jogo, podem ser destacados os seguintes funcionamentos:

* A nave pode se movimentar para a esquerda ou para a direita;
* A nave atira de acordo com a entrada do usuário;
* Os alienígenas se locomovem em uma única direção por vez, em conjunto;
* Os alienígenas atiram no jogador;
* Cada alienígena que toma um tiro, é abatido;
* Caso o jogador seja atingido, o jogo termina, e ele perde;
* Caso o jogador atinja todos os alienígenas, o jogo termina e ele ganha.

Considerando os itens apresentados, o teste de funcionamento deve comprovar que todos, sem exceção, são válidos. Após a confirmação, terá início a análise dos programas, ressaltando as mecânicas explicitadas.

* 1. **Análise do código**

A análise dos programas se dará individualmente, para cada arquivo. Para facilitar o acompanhamento da análise, os códigos foram ‘quebrados’ em blocos, caracterizados pelo comportamento geral, função em que desempenha no conjunto do jogo, e estes são divididos em partes, com uma ou mais procedures.

São os arquivos:

* Space-invaders.asm;
* Arena.asm;
* Bullets.asm;
* Display.asm;
* Game.asm;
* Invaders.asm;
* Keyboard.asm;
* Player.asm.
* Bootloader.asm

É importante lembrar que estes não são os únicos arquivos inclusos no Jogo, também havendo um LICENSE, um README.md, um .gitignore e dois S*hell Script*, create.sh e start.sh. Como a intenção no relatório é perceber como a Linguagem *Assembly* implica no funcionamento do jogo, apenas estes, .asm, serão abertos e, consequentemente, analisados.

1. **OPERACIONAL**
   1. **Medidas Pré-Execução**

Antes de executar o código do jogo baixado, é preciso atender também a certas condições. Para garantir a integridade e segurança do sistema operacional nativo, foi criada uma partição de disco por máquina virtual, através do *Software* da *Oracle*, *Virtualbox*, para instalação da distribuição *Linux,* *Lubuntu*. A escolha do SO tem razão pela interface gráfica amigável a desenvolvedores e por ser leve, bastando pouco menos de 2GB de RAM alocadas da máquina hospedeira para a virtual. Assim, a execução da máquina virtual fica mais otimizada, evitando lentidão de processos, ou até travamentos.

É importante ressaltar que o *Windows* não possuía desvantagens significativas para o cumprimento dos objetivos, sendo a alternativa considerada apenas para isolar a execução dos programas ao *Hardware* da máquina hospedeira.

* 1. **Conhecendo a Pasta projeto**

Ao extrair a pasta principal, raiz, tem-se dois arquivos *Assembly*, “space\_invaders.asm” (o arquivo principal, que chama o código contido nos outros .asm) “bootloader.asm”, que será detalhado mais à frente, um documento de texto, “.gitgnore”, um “LICENSE”, um “README.md” e duas pastas, “src” (contendo os códigos modulados do jogo, a serem usados por “space\_invaders.asm”) e “scripts”, que contém dois *Shell Scripts*, “create.sh” e “start.sh”. Estes dois últimos arquivos serão utilizados apenas na montagem do jogo dentro do sistema operacional.

* 1. **Montagem do Jogo**

Após inicializar o *Lubuntu*, e instalar os *Softwares* QEmu e Nasm (não será detalhado o processo de instalação neste relatório), deve-se abrir o *Terminal* (nativo do *Linux Lubuntu*) e digitar, em ordem, as seguintes linhas de comando:

*mkdir -p build*

*nasm bootloader.asm -f bin -o build/bootloader.bin*

*nasm space-invaders.asm -f bin -o build/space-invaders.bin*

*dd if=build/bootloader.bin of=build/image.img bs=512 count=1*

*dd if=build/space-invaders.bin of=build/image.img bs=512 count=4 seek=1*

O código acima, contido no arquivo "create.sh" na pasta "scripts", foi rodado para criar o arquivo de imagem emulável (.img) do jogo. A segunda e a terceira linhas acima utilizam o montador *Nasm* para criar os arquivos binários de execução de *Boot* a partir dos arquivos .asm.

Dos arquivos .bin recém-criados, foi extraído o arquivo de imagem "space.img".

* 1. **Emulação**

Após a extração do arquivo "space.img" foi preciso executá-lo usando o emulador QEmu, de funcionamento similar a uma máquina virtual comum, mas com a vantagem de simular o boot de um computador e assim permitir a execução dos programas assembly feitos para tal plataforma. Após a execução do comando "qemu-system-x86\_64 -fda space.img" a simulação do boot foi feita e a tela do jogo foi aberta, disponibilizando-o para ser testado.

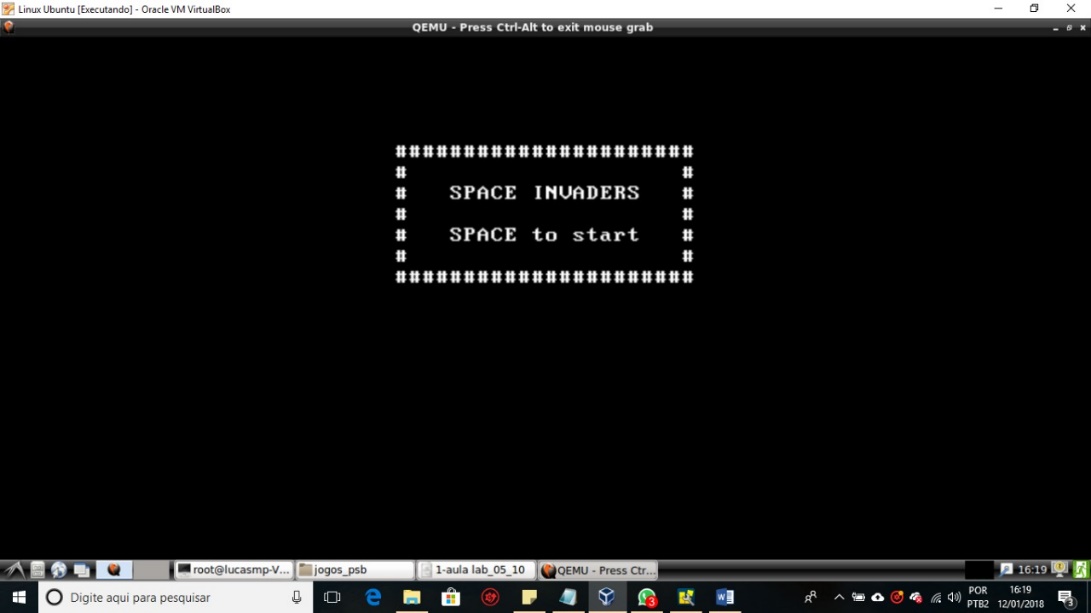


Imagem 1 – Tela inicial do Jogo em execução, rodando no QEmu, VurtualBox.

* 1. **Teste de Funcionamento**

O teste deve conferir se os itens citados em 4.1 estão de acordo. Assim deve-se avaliar, ao executar o jogo, a partir das entradas do usuário, ou avaliação do *Software*, se a ação ocorre corretamente. Em caso confirmativo, os códigos são analisados.

As representações da nave e dos aliens são dados pelos caracteres “M” e “T”, respectivamente.

* + 1. **Movimento da Nave**

Segundo a interface gráfica do jogo, as teclas para movimentar a nave são: “A” para mover para a esquerda e “D” para a direita. Logo ao iniciar o jogo, as teclas foram testadas e corresponderam ao informado.

* + 1. **Tiro da Nave**

Para atirar com a nave, deve-se apertar a tecla espaço. Também funcionou corretamente.

* + 1. **Movimento dos Alienígenas**

Os alienígenas estão organizados em fileiras alternados como no xadrez (diferindo um pouco da versão clássica, mas tal fato é meramente estético, não comprometendo a execução do programa). Foi verificado o movimento, controlado pelo sistema, de uma direção por vez, descendo na tela ao chegar ao limite lateral e refletir o movimento horizontalmente (direita, desce, esquerda, desce...).

* + 1. **Tiro dos Alienígenas**

Os tiros dos alienígenas também são controlados pelo sistema, ocorrendo por fileira, 1 tiro para cada alienígena, a cada unidade regular de tempo. A diferença em relação ao clássico se dá pela multiplicidade de tiros, que no original era de um por nave aleatória, sendo, novamente, apenas uma mudança devido ao gosto do desenvolvedor, sendo a versão presente, a mecânica que deve ser analisada.

* + 1. **Morte dos Alienígenas**

Os aliens morrem caso acertado o tiro da nave, como era previsto anteriormente.

* + 1. **Morte do Jogador**

O jogador morre se for atingido por um tiro de alien, validando o caso de perda. Para reiniciar o jogo, basta apertar a tecla “R”.

* + 1. **Vitória do Jogador**

O jogador vence se todos os aliens forem abatidos, como era esperado.

* 1. **Análise dos Códigos**
     1. **Space-invaders.asm**

Esse é o arquivo principal, que funciona como a ‘placa-mãe’ do *Software*. Nele, todos os arquivos são incluídos, desempenhando suas funções quando chamadas.

Tem-se, portanto:

*1)*

*; constants*

*%define NUM\_INVADERS 12*

*%define GAME\_WIDTH 27*

*%define INAVDERS\_MOVE\_CYCLES 40*

*%define INVADERS\_SHOOT\_CYCLES 4*

*%define BULLETS\_MOVE\_CYCLE 28*

*2)*

*; clear the cursor blinking*

*mov ah, 0x01*

*mov cx, 0x2000*

*int 0x10*

*3)*

*; calculate game screen position*

*mov ah, 0x0F*

*Int 0x10 ; load the number of columns*

*sub byte ah, 1*

*sub ah, GAME\_WIDTH*

*sar ah, 1*

*mov [display\_offset], ah*

As funções desempenhadas pelos blocos acima são, respectivamente: 1) define constantes necessárias através do parâmetro %define; 2) retiram o piscar do cursor; 3) calculam a posição para a tela do jogo. As constantes são usadas ao longo de todo o jogo, definidas em 1):

* Há 12 invasores no (desenho de cada invasor é a letra “T”);
* A largura do cenário (limitada pela fileira de símbolos #) é igual a 27 espaços, sendo cada espaço corresponde à largura na tela em que caiba um caractere alfanumérico comum, como a letra T;
* “INVADERS\_MOVE\_CYCLES” é o ciclo que engloba um conjunto de movimentos de cada invader. O ciclo é constantemente reiniciado quando o contador atinge seu valor de 40, para que o movimento continue;
* “INVADERS\_SHOOT\_CYCLES” tem valor 4, pois é a quantidade de “passos” dados por cada invader entre dois momentos distintos de disparo. Os passos são no formato 1-2-4-3, no plano cartesiano.
* “BULLETS\_MOVE\_CYCLE” é um valor limite que cada bala disparada por invader ou jogador pode percorrer. 28 é o seu valor constante e equivale à distância máxima em passos. A distância percorrida pelo disparo do invader mais alto até o ponto mais baixo da tela é de 28 espaços.

*4)*

*jmp main*

*5)*

*; include dependencies*

*%include "./src/keyboard.asm"*

*%include "./src/display.asm"*

*%include "./src/game.asm"*

*%include "./src/bullets.asm"*

*%include "./src/invaders.asm"*

*%include "./src/player.asm"*

*%include "./src/arena.asm"*

*6)*

*; main loop*

*main*

*mov ah, [program\_state*

*cmp ah, 1*

*je .game*

*cmp ah, 2*

*je .end*

*.intro*

*call intro*

*jmp main*

*.game*

*call game*

*jmp main*

*.end*

*call end*

*jmp main*

Para estes blocos, tem-se: 5) Aqui, estão inclusas as dependências (outros arquivos .asm) criadas pelo mesmo autor, que são complementares ao arquivo principal. Nestes arquivos, estão definidos parâmetros para os elementos do cenário, como balas (disparos), invasores e jogador, além das definições para a entrada pelo teclado e da exibição na tela. Há também uma linha (item 4) )que faz o salto obrigatório para o rótulo “main”.

Em 6), o rótulo main, checa o estado do programa (“program\_state”) e mantém o jogo funcionando ou o encerra, a depender do valor 1 ou 2 para ele.

*7)*

*; intro screen*

*intro:*

*call clear\_screen*

*mov ax, intro\_string\_t*

*mov bx, intro\_string\_o*

*call print\_window*

*.wait:*

*call get\_key*

*mov al, [key\_pressed]*

*cmp al, ' '*

*je .game*

*jmp .wait*

*.game:*

*mov byte [program\_state], 1*

*ret*

Este bloco , 7), é o responsável pela exibição da primeira tela do jogo (tela de menu), onde se aguarda pelo botão “espaço” para inicializar o cenário. Os procedimentos chamados pelo “call” e definidos nas dependências, tem as seguintes funções:

“clear\_screen” (em “*display.asm*”, Anexo 1): move valores específicos para os registradores, deixando a tela limpa, com fundo preto.

“print\_window” (em “*display.asm*”): faz imprimir, nesse momento, os caracteres da tela de menu do jogo. Várias chamadas são feitas dentro do arquivo para imprimir a mensagem, além das linhas em branco. Esse procedimento recebe os valores de “intro\_string\_t” e intro\_string\_o cujo conteúdo é, respectivamente "# SPACE INVADERS #" e "# SPACE to start #".

- get\_key (em “*keyboard.asm*”): recebe a tecla pressionada pelo usuário e a move para o registrador al. Al é então comparado a ‘ ‘ e caso igual, vai para o rótulo .game, caso contrário, permanece aguardando (rótulo .wait) até que se pressione espaço.

*8)*

*; game loop*

*game:*

*call init\_game*

*9)*

*.loop:*

*; check the current program state*

*cmp byte [program\_state], 1*

*jne .done*

*10)*

*; get key if available*

*call check\_key*

*11)*

*; check the game state*

*cmp word [player\_pos], 0x0000*

*je .invaders*

*12)*

*; check whether the player wins*

*cmp byte [num\_invaders\_alive], 0*

*je .player*

*13)*

*; execute a game step*

*jmp .execute*

*14)*

*.invaders:*

*mov byte [winner], 1*

*jmp .done*

*15)*

*.player:*

*mov byte [winner], 0*

*jmp .done*

*16)*

*.execute:*

*; move*

*call move\_bullets*

*call move\_player*

*call move\_invaders*

*; render*

*call clear\_screen*

*call render\_arena*

*call render\_bullets*

*call render\_player*

*call render\_invaders*

*; render controls*

*mov dx, 0x0000*

*mov si, left\_string*

*call print\_string*

*inc dh*

*mov si, right\_string*

*call print\_string*

*inc dh*

*mov si, shoot\_string*

*call print\_string*

Este bloco é um dos maiores responsáveis pelo funcionamento do jogo, dando o sentido visual da execução e permite a interação do usuário. 8) chama o rótulo “init\_game” (definido em game.asm, Anexo 2), que inicializa o personagem do jogador e dos adversários através da movimentação de valores para a variáveis/espaços de memória “player\_pos” e “invaders”. 9) verifica o valor de estado do programa, que é para fins de controle de execução. Caso 1, permanece com o jogo em funcionamento. Caso contrário, encerra.

O item 10) verifica se as teclas de controle do jogo: A, D e espaço foram deixadas disponíveis pelo computador para uso no jogo. 11) e 12) checam condições para que, respectivamente, os adversários (invaders) ou o jogador, tenha vencido o jogo. Para o adversário vencer (jogador perde), basta que a comparação da posição do player com o valor zero resulte verdadeira, para ir para o rótulo .invaders. Para que o player vença, basta que o número de invasores vivos (num\_invaders\_alive) seja 0, indo para o rótulo .player.

O 13) é o loop de execução do jogo, fazendo o papel similar à “atualização dos frames”. O rótulo .execute, definido mais ao fim do código, renderiza os elementos do cenário e os controles.

14) e 15) , nos rótulos .invaders e .player, respectivamente, são definidos, para cada caso, o bit 1 ou 0 para a variável de memória “winner”. Esse valor será usado mais adiante para imprimir na tela a mensagem do vencedor.

O bloco 16) usa o registrador ponteiro para índice de origem (registrador SI) para receber as strings de legenda do jogo: left\_string, right\_string e shoot\_string que tem conteúdo “A = move left”, “D = move right” e “SPACE = shoot”, respectivamente.

*17)*

*.continue:*

*mov cx, 0x0000 ; 0.05 seconds (cx:dx)*

*mov dx, 0x1388 ; 0x00001388 = 5000*

*call sleep*

*jmp .loop*

*18)*

*.done:*

*mov byte [program\_state], 2*

*ret*

Em 17), o rótulo trata de uma “pausa” temporal muito pequena (0,05 segundos) antes de checar o valor de estado do programa pelo rótulo .loop, descrito no item 9)., enquanto em 18), o rótulo .done, quando acionado, joga 2 no valor de estado do programa. Quando for necessário checar esse valor através do rótulo .loop, o programa se preparará para encerrar (pois o valor encontrado foi diferente de 1).

*19)*

*; end screen*

*end:*

*cmp byte [winner], 0*

*je .player*

*mov ax, end\_string\_l*

*jmp .continue*

*.player:*

*mov ax, end\_string\_w*

*.continue:*

*mov bx, end\_string\_o*

*call print\_window*

*.wait:*

*call get\_key*

*mov al, [key\_pressed]*

*cmp al, 'r'*

*je .game*

*jmp .wait*

*.game:*

*mov byte [program\_state], 1*

*ret*

19) Bloco responsável por imprimir a última tela do Space Invaders, a qual informa o vencedor e oferece a opção de reiniciar. O bit na variável “winner” é testado: Caso 0: então o usuário venceu e a string end\_string\_w, de conteúdo "# PLAYER wins #" é movida para ax, através do rótulo .player; Caso não seja 0 (obrigatoriamente 1): a string end\_string\_l, contendo "# INVADERS win #", é movida para ax. Em qualquer situação, ainda bx receberá end\_string\_o, contendo "# Press R to restart #". Após isso, o procedimento print\_window será chamado para imprimir, de fato, a mensagem no console. Aguarda-se então no teclado por “r”, para que o bit de estado de jogo volte a ser 1 (jogo executando) para que o game se reinicie.

*20)*

*; window*

*window\_bar db "######################", 0*

*window\_space db "# #", 0*

*; intro*

*intro\_string\_t db "# SPACE INVADERS #", 0*

*intro\_string\_o db "# SPACE to start #", 0*

*; end*

*end\_string\_w db "# PLAYER wins #", 0*

*end\_string\_l db "# INVADERS win #", 0*

*end\_string\_o db "# Press R to restart #", 0*

*; controls*

*left\_string db "A = move left", 0*

*right\_string db "D = move right", 0*

*shoot\_string db "SPACE = shoot", 0*

*; program state*

*; 0: start screen*

*; 1: game screen*

*; 2: end screen*

*program\_state db 0*

O bloco 20 cria as strings que serão utilizadas, com nome, tipo e conteúdo.

*21)*

*segment .bss*

*; display properties*

*display\_offset resb 1*

*; keyboard*

*key\_pressed resb 1*

*; game*

*; 0: player wins*

*; 1: invaders win*

*winner resb 1*

*; player*

*player\_pos resw 1*

*; invaders*

*invaders resw NUM\_INVADERS*

*num\_invaders\_alive resb 11*

*invaders\_move\_direction resb 1*

*invaders\_move\_cycle resb 1*

*invaders\_shoot\_cycle resb 1*

*; bullets: 0x PY PX STATUS*

*; STATUS == 0: end of list*

*; STATUS == #: explosion*

*; STATUS == p: player bullet*

*; STATUS == i: invader bullet*

*bullets\_move\_cycle resb 1*

*bullet\_list\_end resw 1*

*bullet\_list resb 1*

Finalizando o programa, tem-se em 21), o bloco de criação de variáveis não inicializadas (segment .bss) com seus respectivos nomes e tamanhos em bytes. Elas são criadas, em geral, para controle dos elementos móveis do cenário.

* + 1. **Arena.asm**

O arquivo com menor código. Desempenha a função de criar o cenário, consistente de um fundo preto, delimitado de “#” por todos os lados, com largura definida no próprio “space\_invaders.asm” (ver “%define”).

*1)*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* render*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*render\_arena:*

*push ax*

*push cx*

*push dx*

*2)*

*mov al, '#'*

*mov dh, 0*

*mov cl, 25*

*.loop:*

*mov dl, 0*

*call print\_object*

*mov dl, GAME\_WIDTH*

*call print\_object*

*inc dh*

*dec cl*

*jnz .loop*

*.done:*

*pop dx*

*pop cx*

*pop ax*

*ret*

* + 1. **Bullets.asm**

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* move*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*move\_bullets:*

*cmp byte [bullets\_move\_cycle], BULLETS\_MOVE\_CYCLE ; every 24 frames*

*je .move*

*inc byte [bullets\_move\_cycle] ; increase the counter*

*jmp .ret*

*.move:*

*push di*

*mov byte [bullets\_move\_cycle], 0 ; reset the counter*

*; delete bullets that are exploded or out of the screen*

*mov di, \_check\_and\_delete\_bullet*

*call \_iterate\_bullets*

*; move all bullets*

*mov di, \_move\_bullet*

*call \_iterate\_bullets*

*pop di*

*.ret:*

*Ret*

*; move a single bullet*

*; SI bullet pointer*

*\_move\_bullet:*

*push ax*

*push dx*

*mov al, [si] ; load status*

*mov dx, [si + 1] ; load position*

*cmp al, 'p'*

*je .player*

*cmp al, 'i'*

*je .invader*

*jmp .done*

*.player:*

*mov al, 0*

*jmp .move*

*.invader:*

*mov al, 2*

*.move:*

*call move*

*mov [si+1], dx ; save new position*

*.done:*

*pop dx*

*pop ax*

*ret*

*; delete bullets that are out of the frame or explosions*

*; SI bullet pointer*

*\_check\_and\_delete\_bullet:*

*push ax*

*push dx*

*mov al, [si] ; load status*

*cmp al, '#'*

*je .remove*

*mov dx, [si + 1] ; load position*

*cmp dh, 0*

*je .remove*

*cmp dh, 24*

*je .remove*

*jmp .done*

*.remove:*

*call \_remove\_bullet ; remove the bullet*

*sub si, 3 ; reset loop to former bullet -> next loop is the next*

*.done:*

*pop dx*

*pop ax*

*ret*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* render all bullets*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; render all bullets*

*render\_bullets:*

*push di*

*mov di, \_render\_bullet*

*call \_iterate\_bullets*

*pop di*

*ret*

*; render a single bullet*

*; SI bullet pointer*

*\_render\_bullet:*

*push ax*

*push dx*

*mov al, [si] ; load status*

*mov dx, [si + 1] ; load position*

*cmp al, '#'*

*je .explosion*

*.bullet:*

*mov al, '|'*

*jmp .print*

*.explosion:*

*mov al, '#'*

*.print:*

*call print\_object*

*pop dx*

*pop ax*

*ret*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* check for collisions*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; check for collisions*

*; DX position of the object*

*check\_bullet\_collisions:*

*mov di, \_check\_bullet\_collision*

*call \_iterate\_bullets*

*.done:*

*Ret*

*; check for a collision between a bullet and an object*

*; DX object position*

*; SI bullet pointer*

*\_check\_bullet\_collision:*

*push ax*

*mov ax, [si + 1] ; load position*

*cmp ax, dx*

*jne .done*

*mov dx, 0x0000 ; set position to invalid state*

*mov byte [si], '#' ; set bullet status to explosion*

*.done:*

*pop ax*

*ret*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* create new bullets*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; let the player shoot a bullet*

*create\_player\_bullet:*

*push ax*

*mov al, 'p'*

*call \_create\_bullet*

*pop ax*

*ret*

*; let an invader shoot a bullet*

*create\_invader\_bullet:*

*push ax*

*mov al, 'i'*

*call \_create\_bullet*

*pop ax*

*ret*

*; create a new bullet*

*; DX position of creator*

*; AL status*

*\_create\_bullet:*

*push dx*

*push di*

*cmp al, 'p'*

*je .player*

*.invader:*

*inc dh ; adjust the creator position*

*jmp .create*

*.player:*

*dec dh ; adjust the creator position*

*.create:*

*mov di, [bullet\_list\_end]*

*mov [di], al ; save the status*

*mov [di + 1], dx ; save the position*

*add di, 3*

*mov byte [di], 0x00 ; set the end of the list*

*mov [bullet\_list\_end], di ; save the list end*

*.done:*

*pop di*

*pop dx*

*ret*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* remove a bullet*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; delete the bullet at SI*

*\_remove\_bullet:*

*push ax*

*push si*

*.loop:*

*cmp si, [bullet\_list\_end]*

*je .done*

*mov al, [si+3] ; copy the status*

*mov [si], al*

*inc si*

*mov ax, [si+3] ; copy the position*

*mov [si], ax*

*add si, 2 ; set SI to the next bullet*

*jmp .loop*

*.done:*

*sub word [bullet\_list\_end], 3 ; adjust end of list*

*pop si*

*pop ax*

*ret*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* iterate bullets*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; cycle through bullets*

*; DI address of the loop functions*

*; calls the function in DX with:*

*; SI bullet pointer*

*\_iterate\_bullets:*

*push si*

*mov si, bullet\_list*

*.loop:*

*cmp si, [bullet\_list\_end]*

*je .done*

*call di*

*add si, 3*

*jmp .loop*

*.done:*

*pop si*

*ret*

* + 1. **Display.asm**

*clear\_screen:*

*mov ax, 0x0700*

*mov bh, 0x0f*

*xor cx, cx*

*mov dx, 0x1950*

*int 0x10*

*ret*

*; print the window*

*; AX: title*

*; BX: options*

*print\_window:*

*push dx*

*push bx*

*push ax*

*mov dl, [display\_offset]*

*add dl, 3*

*; top*

*mov dh, 0x05*

*mov si, window\_bar*

*call print\_string*

*; space 1*

*inc dh*

*mov si, window\_space*

*call print\_string*

*; title*

*inc dh*

*pop si*

*call print\_string*

*; space 2*

*inc dh*

*mov si, window\_space*

*call print\_string*

*; options*

*inc dh*

*pop si*

*call print\_string*

*; space 3*

*inc dh*

*mov si, window\_space*

*call print\_string*

*; bottom*

*inc dh*

*mov si, window\_bar*

*call print\_string*

*; return*

*pop dx*

*ret*

*; DX position of char*

*; AL character*

*print\_object:*

*add byte dl, [display\_offset]*

*call move\_cursor*

*sub byte dl, [display\_offset]*

*call print\_char*

*ret*

*; DX cursor position*

*move\_cursor:*

*mov ah, 0x02*

*xor bh, bh*

*int 0x10*

*ret*

*; AL character*

*print\_char:*

*mov ah, 0x0E ;t ell BIOS that we need to print one charater on screen*

*mov bh, 0x00 ; page number*

*mov bl, 0x07 ; text attribute 0x07 is lightgrey font on black background*

*int 0x10*

*ret*

*; DX position*

*; SI string pointer*

*print\_string:*

*push ax*

*call move\_cursor*

*.loop:*

*mov al, [si]*

*cmp al, 0x00*

*je .done*

*call print\_char*

*inc si*

*jmp .loop*

*.done:*

*pop ax*

*ret*

* + 1. **Game.asm**

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* initialize the game state*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*init\_game:*

*; initialize the player*

*mov word [player\_pos], 0x1401*

*; initialize the invaders*

*mov byte [invaders\_move\_direction], 1*

*mov byte [invaders\_move\_cycle], 0*

*mov byte [invaders\_shoot\_cycle], 0*

*mov byte [num\_invaders\_alive], NUM\_INVADERS*

*mov word [invaders + 0], 0x0102*

*mov word [invaders + 2], 0x0304*

*mov word [invaders + 4], 0x0106*

*mov word [invaders + 6], 0x0308*

*mov word [invaders + 8], 0x010A*

*mov word [invaders + 10], 0x030C*

*mov word [invaders + 12], 0x010E*

*mov word [invaders + 14], 0x0310*

*mov word [invaders + 16], 0x0112*

*mov word [invaders + 18], 0x0314*

*mov word [invaders + 20], 0x0116*

*mov word [invaders + 22], 0x0318*

*; initialize the bullets*

*mov byte [bullets\_move\_cycle], 0*

*mov byte [bullet\_list], 0*

*mov word [bullet\_list\_end], bullet\_list*

*ret*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* check whether the game is finished*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; AL 0: continue*

*; 1: invaders win*

*; 2: player wins*

*check\_game\_state:*

*; check whether the player is destroyed*

*cmp word [player\_pos], 0x0000*

*je .invaders*

*; check whether the player wins*

*cmp byte [num\_invaders\_alive], 0*

*je .player*

*; continue*

*mov al, 0*

*jmp .done*

*.invaders:*

*mov al, 1*

*jmp .done*

*.player:*

*mov al, 2*

*.done:*

*Ret*

*; sleep*

*; DX Duration in micro-seconds*

*Sleep:*

*push ax*

*mov cx, 0x0000 ; Sleep for 0,05 seconds (cx:dx)*

*mov dx, 0x1388 ; 0x00001388 = 5000*

*mov ah, 0x86*

*int 0x15 ; Sleep*

*pop ax*

*ret*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* move an object*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; DX position to move*

*; AL direction*

*Move:*

*cmp al, 1*

*je .right*

*cmp al, 2*

*je .down*

*cmp al, 3*

*je .left*

*.up:*

*cmp dh, 0*

*jle .done*

*sub word dx, 0x0100*

*jmp .done*

*.down:*

*cmp dh, 24*

*jge .done*

*add word dx, 0x0100*

*jmp .done*

*.left:*

*cmp dl, 1*

*jle .done*

*sub word dx, 0x0001*

*jmp .done*

*.right:*

*cmp dl, GAME\_WIDTH – 1*

*jge .done*

*add word dx, 0x0001*

*.done:*

*Ret*

* + 1. **Invaders.asm**

Esse arquivo corresponde ao comportamento e exibição dos alienígenas, invasores.

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* move*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*move\_invaders:*

*cmp byte [invaders\_move\_cycle], INAVDERS\_MOVE\_CYCLES*

*je .move*

*inc byte [invaders\_move\_cycle]*

*jmp .done*

*.move:*

*push si*

*push cx*

*push dx*

*push ax*

*mov byte [invaders\_move\_cycle], 0 ; reset the cycle counter*

*mov si, invaders*

*mov cl, NUM\_INVADERS*

*.loop:*

*; load position*

*mov dx, [si]*

*; skip invader, if destroyed*

*cmp dx, 0x0000*

*je .continue*

*; move invader*

*mov al, [invaders\_move\_direction]*

*call move*

*; check collisions*

*call check\_bullet\_collisions*

*mov [si], dx*

*cmp dx, 0x0000*

*jne .shoot*

*; dev invader counter*

*dec byte [num\_invaders\_alive]*

*jmp .continue*

*.shoot:*

*; shoot, if necessary*

*cmp byte [invaders\_shoot\_cycle], INVADERS\_SHOOT\_CYCLES*

*jne .continue*

*call create\_invader\_bullet*

*.continue:*

*add si, 2*

*dec cl*

*jnz .loop*

*; update the shoot cycle*

*cmp byte [invaders\_shoot\_cycle], 4*

*jne .inc\_shoot\_cycle*

*mov byte [invaders\_shoot\_cycle], 0*

*jmp .update\_move\_direction*

*.inc\_shoot\_cycle:*

*inc byte [invaders\_shoot\_cycle]*

*.update\_move\_direction:*

*mov al, [invaders\_move\_direction]*

*inc al*

*cmp al, 4*

*jl .save\_move\_direction*

*xor al, al ; reset the move direction*

*.save\_move\_direction:*

*mov [invaders\_move\_direction], al*

*; restore registers*

*pop ax*

*pop dx*

*pop cx*

*pop si*

*.done:*

*Ret*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* render*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*render\_invaders:*

*push si*

*push ax*

*push cx*

*mov al, 'T'*

*mov si, invaders*

*mov cl, NUM\_INVADERS*

*.loop:*

*mov dx, [si]*

*cmp dx, 0x0000*

*je .continue*

*call print\_object*

*.continue:*

*add si, 2*

*dec cl*

*jnz .loop*

*.done:*

*pop cx*

*pop ax*

*pop si*

*ret*

* + 1. **Keyboard.asm**

Este programa é dos mais curtos. Sua função dentro do projeto é reconhecer as entradas dadas pelo usuário (jogador), através do teclado.

*1)*

*; check whether a key has is pressed*

*; if yes, load it into the buffer*

*check\_key:*

*push ax*

*2)*

*; check whether there is a key available*

*mov ah, 0x01*

*int 0x16*

*jz .reset ; no key in the buffer*

*3)*

*; get key from the buffer*

*mov ah, 0x00*

*int 0x16*

*mov [key\_pressed], al*

*jmp .done*

*.reset:*

*mov byte [key\_pressed], 0*

*.done:*

*pop ax*

*ret*

*4)*

*; wait for a key to be pressed*

*get\_key:*

*push ax*

*mov ah, 0x00*

*int 0x16*

*mov [key\_pressed], al*

*pop ax*

*ret*

* + 1. **Player.asm**

Para este programa, o conteúdo se refere ao movimento, tiro e aparência da nave.

*1)*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* move*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*move\_player:*

*push ax*

*push dx*

*2)*

*; load data*

*mov dx, [player\_pos]*

*mov al, [key\_pressed]*

*3)*

*cmp al, 'a'*

*je .left*

*cmp al, 'd'*

*je .right*

*cmp al, ' '*

*je .shoot*

*jmp .check*

*.shoot:*

*call create\_player\_bullet*

*jmp .check*

*.left:*

*mov al, 3*

*call move*

*jmp .check*

*.right:*

*mov al, 1*

*call move*

*.check:*

*call check\_bullet\_collisions*

*mov [player\_pos], dx*

*.done:*

*pop dx*

*pop ax*

*ret*

*4)*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*; \* render*

*; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*render\_player :*

*push ax*

*push dx*

*mov dx, [player\_pos]*

*cmp dx, 0x0000*

*je .done*

*mov al, 'M'*

*call print\_object*

*.done:*

*pop dx*

*pop ax*

*ret*

* + 1. **Bootloader.asm**

O arquivo “bootloader.asm” torna o jogo capaz de ser carregado no boot de um computador. A execução no boot é uma habilidade especial de programas em A*ssembly*, pois, sendo linguagem de baixo nível e especificando diretamente os registradores com os quais trabalhará, não é preciso que o Sistema Operacional, que, por si, é mais robusto que a *BIOS* (esteja rodando para que a aplicação seja suportada.

*; initialize the environment*

*mov ax, 0x07c0*

*mov ds, ax*

*; the segment of the game*

*mov ax, 0x07e0*

*; setup the file-source*

*mov ch, 0x00 ; cylinder 0*

*mov cl, 0x02 ; sector 2 (skip first sector, which is the bootloader)*

*mov dh, 0x00 ; head 0*

*mov dl, 0x00 ; drive 0 (floppy disk)*

*; setup the destination*

*mov es, ax ; segment starts directly after the bootloader (7c00 - 7dff)*

*mov bx, 0x0000*

*; copy data into RAM*

*read:*

*mov al, 0x04 ; read four sectors*

*mov ah, 0x02 ; int 13h subfunction 2 -> read sectors (512 bytes) from disk*

*int 0x13 ; copy sectors to ES:BX*

*jc read ; carry-flag is set -> there was a read-error, retry*

*; rebase segments for game execution*

*mov ax, 0x07e0*

*mov ds, ax ; data segment*

*mov es, ax ; additional segments*

*mov fs, ax*

*mov gs, ax*

*mov ss, ax ; stack segment*

*; reset stack pointer*

*; mov sp, 0x0000*

*; enter the game code -> set CS:IP*

*jmp 0x07e0:0x0000*

*; spacing and signature*

*times 510 - ($ - $$) db 0*

*dw 0xaa55*

1. **RESULTADOS E DISCUSSÕES**

O Jogo funcionou perfeitamente, o que possibilitou a análise e serviu como base para o entendimento dos códigos (A ideia de como funciona o jogo visualmente possibilita prever, com conhecimento anteriormente adquirido, como foi desenvolvido o game).

1. **CONCLUSÃO**
2. **REFERÊNCIAS**

GitHub. Disponível em: <https://github.com/flxbe/asm-space-invaders>. Acesso em 08 de Janeiro de 2018.

Nasm. Disponível em:< http://www.nasm.us>. Acesso em 10 de Janeiro de 2018

Ubuntu. Disponível em: <https://www.ubuntu.com/download/desktop>. Acesso em 13 de Janeiro de 2018

University of Virginia Computer Science. Disponível em: <http://www.cs.virginia.edu/~evans/cs216/guides/x86.html>. Acesso em 12 de Janeiro de 2018.­­

VirtualBox. Disponível em <https://www.virtualbox.org>. Acesso em 13 de Janeiro de 2018.

Wikipedia. Disponível em: <https://pt.wikipedia.org/wiki/Assembly#Arquitetura>. Acesso em 12 de Janeiro de 2018.

Wikipedia. Disponível em: <https://pt.wikipedia.org/wiki/Boot>. Acesso em 09 de Janeiro de 2018

Wikipedia. Disponível em: <https://pt.wikipedia.org/wiki/Space\_Invaders>. Acesso em 12 de Janeiro de 2018.