Universidade de São Paulo
Laboratório de Processadores

ARM Invaders

Mini-jogo/simulador didático de registradores ARM no terminal

Relatório Final

Autor: Antonio Fassini

Professor: (preencher)

Data: Agosto de 2025

Abstract

O *ARM Invaders* é um mini-jogo/simulador didático, executado em terminal, que visualiza o estado dos registradores `r0..r7` e das flags `N/Z/C/V` enquanto o usuário executa operações aritméticas (`ADD`, `SUB`, `MUL`, `MOV`). O projeto busca reduzir a abstração típica da arquitetura ARM por meio de uma interface simples, colorida e com elementos lúdicos (*HUD*, barras de "vida" e "naves" em movimento). Foram implementados também comandos de utilidade (`rand`, `save`/`load`, `script`) para facilitar demonstrações e testes. O código é em C padrão (C11), portável para Linux/WSL e Raspberry Pi.

Contents

1 Introdução

Aprender arquitetura de processadores e manipulação de registradores pode parecer abstrato em estágios iniciais. O **ARM Invaders** nasce com o objetivo de tornar concreto — e divertido — o impacto de instruções básicas sobre registradores e flags, usando uma interface de terminal leve, colorida e interativa.

1.1 Objetivos

- Visualizar em tempo real alterações em `r0..r7` e nas flags `N/Z/C/V`;
- Relacionar comandos de alto nível a instruções ARM (representadas no *log* didático);
- Propor um artefato simples, de baixo risco e fácil execução em laboratório (PC/WSL ou Raspberry Pi);
- Servir de base para extensões (novas instruções, UI com `ncurses`, modo desafio etc.).

2 Relação com a Disciplina

Embora o foco da disciplina seja prática com ARM e Raspberry Pi, muitos estudantes esbarram na abstração inicial de registradores e flags. O *ARM Invaders* reforça:

- Conceitos de **arquitetura ARM**: registradores de propósito geral, flags `N/Z/C/V`, operações básicas;
- **Integração hardware–software**: como mudanças em dados internos se refletem em comportamento do programa;
- **Raciocínio passo a passo**: cada comando produz um estado observável, facilitando a depuração e o estudo.

3 Escopo e Decisões de Projeto

3.1 Simplicidade intencional

Optou-se por um escopo enxuto: operações aritméticas essenciais e uma UI ASCII com cores ANSI. Essa simplicidade:

- Reduz dependências (compila com `gcc` padrão);
- Favorece a compreensão do código por iniciantes;
- Diminui riscos de integração/ambiente e viabiliza a entrega no prazo.

3.2 Portabilidade e baixo custo

O código C (C11) roda tanto em WSL/Linux de PC quanto em Raspberry Pi (ARM). Evitou-se usar bibliotecas gráficas pesadas.

4 Arquitetura e Implementação

4.1 Visão geral

O projeto consiste em um único executável que mantém estado de CPU (registradores e flags) e oferece um REPL de comandos. A saída é reimpressa a cada operação com um *HUD* colorido (barras de 0–100, flags destacadas e "naves" com movimento lateral simples).

4.2 Principais componentes

- **Estrutura `CPU`**: `uint32_t r[8]`, `Flags`, posições/direções das "naves" e contador de turnos;
- **Atualização de flags**: `ADD/SUB` com modelagem realista; `MUL` com `C/V` didáticas;
- **REPL e parser**: comandos `add/sub/mul/mov/rand/save/load/script/show/reset/help/qui`
- **Persistência**: `save`/`load` em texto simples (fácil de inspecionar/versionar).

4.3 Estrutura de diretórios

```
arm_invaders/
          src/arm_invaders_sim.c
          doc/Relatorio.tex    (este arquivo; PDF gerado aqui)
          media/               (prints do terminal para o README/
   relat rio)
          scripts/demo.txt     (roteiro automatizado de comandos)
          Makefile
          README.md
          LICENSE
          .gitignore
```

5 Funcionalidades

- **Operações**: `add x k`, `sub x k`, `mul x y`, `mov x k`;
- **Utilidades**: `rand x a b`, `save`/`load`, `script arquivo.txt`, `show`/`reset`/`help`/`quit`;

- **UI**: cores ANSI, barras de vida (0–100), flags coloridas, “naves” animadas, explosão ASCII em `rX=0`;
- **Didática**: impressão de pseudo-`ASM` no terminal após cada operação.

6 Como Compilar e Executar

6.1 Requisitos

Ubuntu/WSL ou Raspberry Pi com `gcc`. Em Ubuntu:

```
sudo apt update
sudo apt install -y build-essential
```

6.2 Makefile

```
make          # compila
make run      # compila e executa
make clean    # remove o bin rio
```

6.3 Execução direta

```
gcc -std=c11 -O2 -Wall -Wextra -o arm_invaders_sim src/
   arm_invaders_sim.c
./arm_invaders_sim
```

7 Roteiro de Testes Sugerido

```
add r2 10
sub r2 5
sub r2 105        # r2 -> 0 (explos o); Z=1
reset
mov 3 0
add 3 1073741824
add 3 1073741824 # overflow de sinal -> V=1
show
mov 4 70000
mov 5 70000
mul 4 5           # produto 32 bits; C/V did ticas
show
```

```
save estado.txt
load estado.txt
script scripts/demo.txt
```

8 Resultados (prints)

Inclua nesta seção capturas do terminal (`media/`) ilustrando:

- Tela inicial do HUD com barras e flags;
- Caso de `rX=0` com explosão;
- Exemplo de `V=1` em soma com overflow;
- Uso de `save`/`load` e `script`.

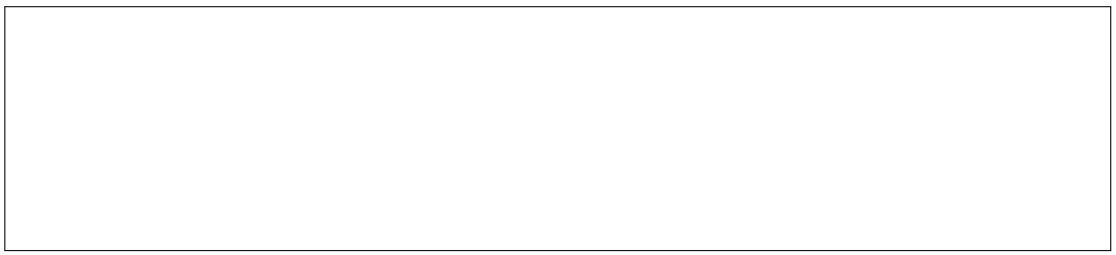

Figure 1: HUD do ARM Invaders (placeholder para captura real).

9 Limitações e Trabalhos Futuros

Limitações:

- Parser simples (imediatos negativos/hex não suportados);
- Modelagem didática de `C/V` em `MUL`;
- UI ASCII (sem `ncurses` por simplicidade).

Possíveis extensões:

- Instruções lógicas (`AND/ORR/EOR`), `LSL/LSR/ASR`;
- Interface com `ncurses` (janelas, teclado não bloqueante);
- Modo desafio/pontuação; exportação de `trace`; web dashboard simples.

10 Conclusão

O *ARM Invaders* cumpre o papel de ponte entre teoria e prática, tornando visível e intuitivo o efeito de instruções sobre registradores e flags ARM. A simplicidade foi uma escolha de engenharia para maximizar portabilidade, reduzir riscos e favorecer aprendizado colaborativo. O código está pronto para servir como base de extensões futuras.

Repositório

Código-fonte e documentação:
https://github.com/SEU_USUARIO/arm-invaders
(*atualize com a URL real do repositório*)

Referências

- ARM Architecture Reference Manual (conceitos de flags e instruções).
- Documentação GCC e *ANSI Escape Codes*.

Anexo A — Roteiro do Vídeo (5 min)

1. **Abertura (20s)**: apresentação e objetivo.
2. **Motivação (30s)**: reduzir abstração; conexão com a disciplina.
3. **Decisões (20s)**: simplicidade, portabilidade, didática.
4. **Demo (2–3 min)**: comandos do roteiro de testes.
5. **Estrutura (30s)**: pastas, Makefile, scripts.
6. **Fecho (20s)**: próximos passos e link do GitHub.

Anexo B — Exemplo de Script

```
# scripts/demo.txt
show
add r2 10
sub r2 5
sub r2 105
reset
mov 3 0
add 3 1073741824
add 3 1073741824
show
mov 4 70000
mov 5 70000
mul 4 5
show
quit
```