UFMG

Trabalho Prático 1 (TP1)
Organização de Computadores I

Ítalo Leal Lana Santos | Matrícula: 2024013893 | italolealanasantos@ufmg.br
Universidade Federal de Minas Gerais (UFMG) | Belo Horizonte/MG | outubro de 2025

---

1. Introdução

Este relatório descreve o desenvolvimento de dois programas em Assembly RISC-V: implementação do algoritmo de criptografia One-Time Pad e multiplicação de matrizes quadradas. Ambos os problemas foram implementados utilizando o Venus Simulator, respeitando as convenções de chamada de procedimento e gerenciamento de pilha do RISC-V.

2. Problema 1: Cálculo de One-Time Pad

2.1 Decisões de Implementação

O procedimento onetime_pad foi implementado utilizando um loop simples que itera sobre cada elemento dos vetores de mensagem e chave. A estratégia adotada foi:

- **Contador de iteração:** Utilizei o registrador t0 como contador do loop, iniciando em 0 e incrementando até atingir o tamanho do vetor
- **Cálculo de offset:** Para acessar elementos individuais, multipliquei o índice por 4 (usando slli t1, t0, 2) devido ao tamanho de cada word ser 4 bytes
- **Operação XOR:** Apliquei xor diretamente nos valores da mensagem e da chave
- **Modificação in-place:** Os resultados foram armazenados de volta no vetor de mensagem, conforme especificação

2.2 Dificuldades Enfrentadas

A principal dificuldade foi compreender o sistema de endereçamento e offset de memória no RISC-V. Inicialmente, tentei acessar os elementos sem multiplicar por 4, o que resultava em sobreposição de dados. Após revisar a documentação sobre o tamanho de words (32 bits = 4 bytes), consegui corrigir o problema.

2.3 Testes Realizados

- Vetores de tamanho 1 (caso mínimo)
- Vetores de tamanho 10 (caso maior)
- Valores extremos (0 e valores máximos de 32 bits)
- Aplicação dupla do XOR para verificar recuperação da mensagem original

3. Problema 2: Multiplicação de Matrizes

3.1 Decisões de Implementação

A implementação foi dividida em dois procedimentos conforme especificado:

Procedimento mat_mult:

- **Gerenciamento de pilha:** Salvei 7 registradores na pilha (ra, s0-s5) para preservar o estado durante chamadas recursivas
- **Loops aninhados:** Implementei dois loops (linhas e colunas) para iterar sobre todos os elementos da matriz resultado
- **Cálculo de endereços:** Para matrizes armazenadas linearmente, calculei o offset como (i * n + j) * 4 para acessar o elemento C[i][j]
- **Preservação de argumentos:** Copiei os argumentos para registradores salvos (s2-s5) para não os perder durante as chamadas a dot_product

Procedimento dot_product:

- **Acesso à coluna:** A maior complexidade foi acessar elementos de uma coluna da matriz B, que está armazenada por linhas. Usei stride de n * 4 bytes para pular linhas e acessar elementos da mesma coluna
- **Acumulação:** Utilizei o registrador t0 para acumular o resultado do produto interno
- **Retorno:** O resultado foi retornado em a0 conforme convenção

3.2 Dificuldades Enfrentadas

O maior desafio foi entender o acesso a colunas em matrizes armazenadas linearmente por linhas. Inicialmente, estava acessando elementos consecutivos, o que funcionava apenas para a primeira linha. Após desenhar a memória no papel, compreendi que precisava usar um stride multiplicativo para acessar os elementos corretos da coluna. Outra dificuldade foi o gerenciamento correto da pilha. Esqueci inicialmente de salvar o registrador ra, o que causava retornos incorretos, e resolvi revisando as convenções de chamada do RISC-V, salvando os registradores necessários.

3.3 Testes Realizados

- Matrizes 2×2 (fornecidas)
- Matrizes 3×3 com valores variados
- Matrizes identidade (verificando se A × I = A) e Matriz zero (verificando se A × 0 = 0)
- Teste isolado do dot_product com vetores conhecidos

4. Principais Aprendizados

Durante o desenvolvimento, compreendi a importância do uso correto da pilha para salvar registradores em chamadas de procedimentos, bem como o cálculo de offsets para acessar vetores e matrizes em memória linear. Também aprimorei a depuração no Venus Simulator e percebi como instruções simples, como slli, são fundamentais para otimizar operações. Além disso, ficou evidente a relevância de seguir as convenções RISC-V para garantir código funcional e modular.

5. Conclusão

Os dois problemas foram implementados com sucesso, respeitando as especificações propostas. O uso do Venus foi essencial para depurar e validar o código. A experiência reforçou a compreensão da arquitetura do processador e do papel das convenções em Assembly, contribuindo para um entendimento mais sólido do funcionamento interno dos computadores.