UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE SISTEMAS DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

# SSC0902 - ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

## TRABALHO 1

Prof. Sarita Mazzini Bruschi
Murilo Cury Pontes - 13830417
Thaís Laura Anício Andrade - 14608765
Gabriel Rosso Fernandes - 14572464

São Carlos
2024

# 1 Introdução

Assembly é uma linguagem de programação de baixo nível, cujas instruções possuem uma conexão direta com as instruções de máquina, de acordo com a arquitetura do computador. A partir do estudo e do uso dessa linguagem, houve uma compreensão mais profunda do funcionamento interno do processador, o qual permite interações entre o hardware e o software. O software utilizado para executar e testar o código foi o simulador RARS, RISC-V Assembler and Runtime Simulator, muito importante para visualizar os conteúdos dos registradores e das posições de memória e, assim, enfrentar os desafios da implementação.

Este trabalho possui como objetivo a implementação do jogo "Pedra, Papel, Tesoura", muito utilizado para seleções aleatórias, codificado em Assembly RISC-V. Nesse caso, o jogador enfrenta o computador, cujas escolhas são aleatórias, e pode decidir quando encerrar a partida. A proposta do projeto, portanto, permite tanto o aprendizado sobre manipulação de registradores e operações de memória, quanto o desenvolvimento da lógica de programação.

# 2 Funcionamento

O jogo Pedra, Papel, Tesoura se resume a uma comparação das escolhas dos jogadores, pedra, papel ou tesoura. Cada escolha tem um resultado específico em relação às outras, sendo que a pedra vence a tesoura, a tesoura vence o papel e o papel vence a pedra. No projeto, o jogador visualiza essas opções na tela, a partir do menu, e faz sua escolha, enquanto o computador seleciona aleatoriamente sua jogada (também entre 0 e 2). Uma cópia do menu pode ser vista a seguir:

0. Pedra
1. Papel
2. Tesoura
3. Sair

Após ambas as escolhas serem realizadas, comparam-se os resultados e determina-se o vencedor da rodada. Por fim, a cada jogada, o jogador pode decidir continuar ou encerrar o jogo. Quando o jogo acaba, o programa imprime o conteúdo da lista ligada em um formato gráfico, usando símbolos de borda para melhorar a visualização: há os pares de escolhas do computador e do jogador para todas as jogadas.

# 3 Estrutura do Código

Para a elaboração do menu, primeiro foram declaradas uma string, nomeada **menu_string** com o conteúdo "\nEscolha:\n0. Pedra \n1. Papel \n2.Tesoura \n3. Sair\n", gerando o menu descrito acima. Além disso, é declarada uma variável do tipo word para armazenar a escolha do jogador.

Então, é realizado um loop utilizando 2 branches e um jump incondicional ao label **menu:**. Nesse trecho, é dada a instrução ecall com o parâmetro 4, correspondente ao numero de chamada de sistema para imprimir a string declarada, assim como, em seguida, é realizado outro syscall com o código 5, que corresponde à leitura de um inteiro.

Em seguida, validamos se o número lido está no intervalo de escolha de jogada (entre 0 e 2). Se o número estiver fora do intervalo, o end_menu é executado, em que é feita uma instrução de syscall com o código 10, que termina a execução do código. Senão, o número é armazenado no endereço de memória declarado acima.

Para a escolha do computador, foi utilizado o serviço 42 do sistema para sortear um número entre zero e dois (esse limite é feito pelo registrador **a1**).

Em seguida, com a escolha do computador, armazenada no registrador **t4**, e a escolha do jogador, armazenada em **s1**, o código realiza uma operação XOR para comparar as duas escolhas e determinar o vencedor. Isso porque a operação retorna um valor distinto dependendo da diferença entre as escolhas: se as escolhas forem iguais, o resultado será zero (empate), e se forem diferentes, o valor resultante indicará qual combinação de jogadas ocorreu e, assim, determinando o vencedor.

Depois, são feitas quatro *branches* para determinar qual escolha permitiu a vitória ou o empate. O registrador **a1**, que contém o resultado da operação XOR, é comparado com os valores possíveis que indicam o resultado:

- Se **a1** for igual a 0 (beq a1, t6, win_tie), isso significa que as escolhas foram iguais, resultando em empate, e a execução salta para o *label* **win_tie**, onde a mensagem de empate é exibida.
- Se **a1** for igual a 1 (beq a1, t6, win_paper), isso indica que o papel venceu a

rodada, e o código direciona a execução para o *label* **win_paper**.

- Se **a1** for igual a 2 (beq a1, t6, win_rock), a pedra venceu a rodada, e o código salta para **win_rock**.
- Se **a1** for igual a 3 (beq a1, t6, win_scissors), a tesoura venceu, e a execução vai para **win_scissors**.

Dentro de cada *label* que trata a jogada vencedora, o código imprime a jogada correspondente (pedra, papel ou tesoura) e verifica quem fez essa escolha. Para isso, a escolha do jogador, armazenada em s1, é comparada com o valor correspondente à jogada vencedora (0 para pedra, 1 para papel e 2 para tesoura). Dependendo do resultado dessa condicional, o código imprime se o jogador ou o computador venceu a rodada, utilizando os *labels* **win_player** e **win_pc**.

Por fim, o código contém uma função auxiliar, **imprime_str**, que é responsável por imprimir qualquer string cujo endereço esteja no registrador **a0**. Dentro dela, utiliza-se o serviço 4 do sistema (ecall) para imprimir a string e retorna para a linha seguinte à chamada com o comando **jr ra**.

A cada rodada, as escolhas dos jogadores são inseridas numa lista ligada simples, a qual apresenta dois procedimentos: **New_Node** para criar os novos nós e **Print_List** para imprimir o conteúdo da lista.

A lista ligada é formada por nós que são alocados dinamicamente. Cada nó contém três campos:

- O valor da escolha do PC representado por **s2**.
- O valor da escolha do PLAYER representado por **s1**.
- O endereço do próximo nó

A lista começa com dois ponteiros:

- **head**: aponta para o primeio nó da lista
- **last**: aponta para o último nó da lista

O processo de inserção de um novo nó em uma lista encadeada começa com o carregamento do ponteiro que indica o último nó, chamado **last**. A partir desse ponto, o programa aloca 12 bytes de memória para criar o novo nó. Esses 12 bytes são divididos entre dois inteiros, que representam os dados a serem armazenados nos campos do nó, e o endereço do próximo nó da lista.

Se a lista estiver vazia, ou seja, se o ponteiro **last** estiver nulo, o programa reconhece que está criando o primeiro nó da lista. Nesse caso, o ponteiro **head**, que indica o início da lista, é ajustado para apontar para esse novo nó. Caso contrário, se a lista já contiver nós, o programa atualiza o ponteiro do nó anterior (o atual último nó) para que ele guarde o endereço do novo nó criado.

Após a criação do nó, os valores inteiros, denominados **s1** e **s2**, são armazenados nos primeiros 8 bytes do novo nó. O endereço de memória que aponta para o próximo nó (ou nulo, se for o último da lista) é armazenado nos últimos 4 bytes, completando assim a estrutura do novo nó.

Quando a função **Print_List** é chamada após a decisão de finalizar o programa por parte do jogador, inicializa-se a impressão carregando o endereço de **head**, o primeiro nó da lista. Foi criado, também, o topo do gráfico com a string armazenada em **top**, indicando os nomes das colunas correspondentes. Logo em seguida, para cada nó da lista, imprimem-se a borda esquerda, os dois valores armazenados no nó e as bordas do meio e da direita. Enquanto **next** não for nulo, continua-se percorrendo a lista ligada e imprimindo todos os nós. Finalizando, a borda inferior é impressa com a string **bottom**.

Encerrada a impressão das escolhas de ambos os jogadores, utiliza-se o serviço 10 do sistema para encerrar o programa.

# 4 Desafios e Soluções

É importante ressaltar que a linguagem Assembly possui alguns desafios intrínsecos à sua sintaxe: como não é possível atribuir nomes aos registradores para facilitar a sua manipulação, é necessário se organizar para entender o que cada variável significa naquele contexto. Então, a fim de não se perder entre os parâmetros, as variáveis salvas e temporárias, o código foi implementado carregado de vários comentários: seja explicando comandos (como chamadas ao sistema), seja anotando os conteúdos dos registradores para auxiliar na leitura do código.

Outro obstáculo enfrentado foi a necessidade de determinar o vencedor da partida com base nas escolhas do jogador e do computador. Uma abordagem trabalhosa seria verificar todas as possibilidades possíveis até obter o resultado (por exemplo, se a comparação for 0 e 1, pedra contra papel, então o papel ganha), cuja organização seria mais difícil de manter. Em vez disso, utilizou-se a operação XOR, uma solução simples: ela retorna valores distintos para cada combinação de escolhas e um valor igual (0) quando há empate. Portanto, a lógica de decisão é significativamente simplificada, permitindo uma verificação rápida e direta do resultado da partida.

Além disso, o principal desafio na implementação da lista encadeada foi integrá-la corretamente ao restante do código utilizando funções. Durante o processo, surgiram *bugs* causados por registradores que assumiam valores aleatórios, o que levou à necessidade de investigar como preservar seus estados ao longo das chamadas de função.

A lição aprendida foi a importância de sempre utilizar o Stack Pointer (**sp**) para armazenar o estado dos registradores, incluindo o **ra**, garantindo que os valores sejam mantidos intactos. Isso é essencial para evitar falhas e garantir a integridade do código, especialmente em projetos colaborativos, onde múltiplos desenvolvedores podem compartilhar e modificar o mesmo conjunto de recursos.

# 5 Conclusões

É notável que a realização do trabalho possibilitou praticar diversas habilidades: organização de um projeto em Assembly, desenvolvimento da comunicação entre os integrantes e do trabalho em equipe. Esses pontos são visíveis tanto pelo comprometimento em manter um código modularizado e bem comentado, quanto pela existência de uma divisão equilibrada das tarefas para os integrantes do grupo, o que permite aumentar a capacitação profissional dos discentes.

No que se refere ao código elaborado, obteve-se um resultado satisfatório, uma vez que o programa cumpre todos os requisitos solicitados: recebe a entrada do jogador, escolhe a jogada da máquina de forma aleatória e, então, verifica o ganhador com um algoritmo claro e conciso, devido à utilização do operador XOR. Por fim, é impressa a lista de escolhas de jogadas de ambos os jogadores antes do término do programa.

Finalmente, em relação às dificuldades enfrentadas durante a elaboração do programa, nota-se que foram atreladas ao uso da linguagem Assembly e à sua sintaxe. Isso ocorre por apresentar menor legibilidade, a qual dificulta a depuração do código, e por ser uma linguagem de baixo nível, dificultando o trabalho com algoritmos, visto que há a necessidade de lidar constantemente com alocação de memória, registradores e *stack* de maneira organizada e consciente.