Arquitetura de Processadores na Prática – TP2

1 FORMAÇÃO DOS GRUPOS

Formação dos grupos: Os grupos devem ser de 2 ou 3 alunos. Não há trabalhos individuais ou de grupos com mais de 3 alunos.

2 TRABALHO A SER DESENVOLVIDO E REGRAS DO JOGO

Pesquisa Binária em Vetores Ordenados

1. Contexto: Procurar um elemento específico em um número imenso de objetos (digamos milhões ou bilhões) pode ser uma tarefa demorada. Caso os objetos tenham sido previamente ordenados em uma sequência linear segundo algum critério, a tarefa pode ser facilitada sobremaneira.
2. Uma técnica muito usada, devido a sua eficiência, para buscar um elemento em uma sequência ordenada de objetos é conhecida como pesquisa binária. Suponha-se, sem perda de generalidade, que os objetos estão sequenciados em ordem crescente. Parte-se do meio da sequência e verifica-se se o objeto que lá se encontra é o procurado. Se não for, procede-se à pesquisa na metade inferior ou na metade superior da sequência, caso o objeto do meio seja, respectivamente, maior ou menor que o procurado. O processo segue até encontrar o elemento, ou se descobrir que ele não existe na sequência inteira.
3. Implementem uma função recursiva para realizar pesquisa binária em um vetor ordenado usando a linguagem de montagem do MIPS **(vale 4 pontos)**. O algoritmo básico de pesquisa binária a usar é dado abaixo, como uma função escrita em C. Note que o que o algoritmo retorna é o índice do elemento do vetor que contém o elemento procurado, ou -1 (um índice inválido, pois índices de um vetor são sempre valores naturais).

```
// ---------------------------------------------------------
// Dado um vetor A, pesquisa entre elementos, de A[Prim]
//       até A[Ult] pelo valor chave, usando pesquisa binária.
//
// Pré-condições: 0<=Prim, Ult<=Tam-1, onde Tam é o tamanho
// total do Vetor A e A[Prim]<=A[Prim+1]<=...<=A[Ult],
// ou seja, o vetor já está ordenado em ordem crescente.
//
// Pós-condição: Se chave existe no vetor A, retorna
// o índice do vetor igual a chave, senão retorna -1.
//
// Lembre-se:    1) Assume-se A como um vetor de inteiros
//               2) Prim e Ult são índices, valores naturais
//               3) chave é um inteiro.
// ---------------------------------------------------------
int binary_search(int A[], int Prim, int Ult, int Chave)
{    int mid;

    if (Prim > Ult)
    { return -1; }
    mid = (Prim + Ult) / 2;
    if (A[mid] == Chave)
    { return mid;}
    else if (A[mid] > Chave)
    { return (binary_search(A, Prim, mid - 1, Chave));}

    else if (A[mid] < Chave)
```

```
        { return (binary_search(A, mid + 1, Ult, Chave));}
    }
```

Demonstrando a operação do algoritmo acima, assuma que o conteúdo de A (um array ordenado de inteiros) é, por exemplo: -5, -1, 5, 9, 12, 15, 21, 29, 31, 58, 250, 325 e teste as chamadas:

```
1. binary_search(A, 0, 11, 325);      # Retorna 11
2. binary_search (A, 11, 11, -1);     # Retorna -1
3. binary_search (A, 0, 5, -1);       # Retorna 1
4. binary_search (A, 4, 11, 31);      # Retorna 8
5. binary_search (A, 4, 11, 32);      # Retorna -1
```

4. Antes de implementar o algoritmo recursivo de pesquisa binária dado acima na linguagem de montagem do MIPS, elabore um programa principal na mesma linguagem de montagem (do MIPS) para testar a rotina binary_search **(vale 3 pontos)**. Lembre-se que este programa principal deve não apenas:
   4.1. **Chamar a rotina**, mas também:
   4.2. Passar argumentos para esta, como exigido no item 5 abaixo, **todos via pilha**;
   4.3. Receber o valor de retorno, **via pilha**;
   4.4. **Imprimir** este valor na console do simulador e;
   4.5. **Esvaziar a pilha** dos dados lá colocados antes da chamada ao final do processamento. Note que os elementos do vetor são inteiros e não somente naturais. Assim, use as instruções de comparação do MIPS adequadas para lidar com números com sinal. Defina uma área de dados adequada para o programa. Acrescente variáveis, se considerar necessário**.**
5. Respeite as seguintes convenções:
   5.1. Passagem de argumentos – **todos os argumentos para a função devem ser passados através da pilha**, apontada pelo registrador $sp. Isso obviamente vale também para argumentos que a função passa para uma chamada recursiva de si própria.
   5.2. O **retorno do valor resultante da execução de qualquer instância da função deve também ocorrer através da pilha** para quem a chamou.
6. Responda às seguintes questões no relatório final do trabalho **(vale 1 ponto)**: (1) Qual o número do registrador $sp no conjunto de registradores do MIPS e qual o seu valor inicial em hexadecimal (valor inicial atribuído pelo simulador MARS)? (2) Qual é o primeiro valor escrito na pilha pelos seu programa, e qual o significado dele? (3) Mostre o conteúdo da pilha ao entrar na **terceira** chamada aninhada de alguma recursão (use print screen ou outro método para gerar figuras do estado da pilha). (4) Qual o conteúdo do registrador $sp neste momento? (5) Isto implica ter quanto espaço alocado na pilha? (6) Observar o retorno do procedimento recursivo. O valor do registrador $sp volta ao valor original? (Se isto não ocorrer seu programa está incorreto, pois sua execução deixa lixo na pilha) (7) Em qual linha de código este valor é reestabelecido?
7. Formato do trabalho e entrega: O trabalho deverá ser entregue via sala do Moodle até a data de **14/10/2022 (sexta-feira)**, contendo o código fonte da aplicação **COMENTADO SEMANTICAMENTE**, as respostas das perguntas do item 6 acima e um relatório final, contendo uma explicação da solução, exemplos de telas do simulador MARS, mostrando alguns dos passos da execução do programa, devidamente comentados **(vale 2 pontos)**.
8. Valor do trabalho: Este trabalho vale **25% da nota de G1** da disciplina.