Arquitetura de Processadores na Prática – TP2

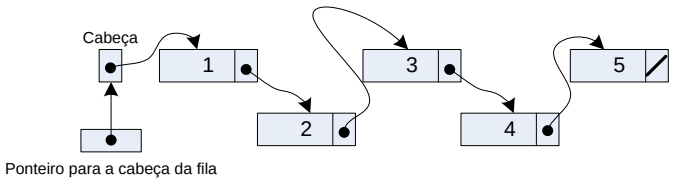
1 FORMAÇÃO DOS GRUPOS

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

2 TRABALHO A SER DESENVOLVIDO E REGRAS DO JOGO

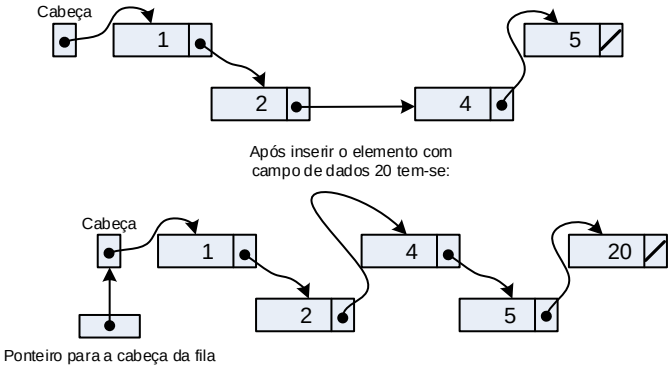
Manipulação de Listas Encadeadas usando Linguagem de Montagem no MIPS

1. Contexto: Listas encadeadas são estruturas de dados muito importantes em computação. Compiladores de linguagens de alto nível devem ser capazes de criar tais listas a partir de código fonte escrito em linguagem de alto nível. A Figura abaixo ilustra através de um exemplo a estrutura de listas encadeadas. Existe um nodo especial, denominado **`cabeça`** da fila, que contém o endereço (de memória) do primeiro elemento desta. Cada elemento da lista em si é formado por dois campos: um campo de conteúdo (que no exemplo contém apenas um número inteiro) e um campo com um ponteiro para o próximo elemento da lista, também chamado de *enlace* ou (em inglês) *link*, que deve ser um endereço de memória. O último elemento da lista possui um ponteiro "nulo", que normalmente se representa com o valor 0, pois o endereço 0 de memória raramente é usado na prática para conter dados de usuário.



2. Como auxílio para realizar o trabalho é dado um método de criação e um formato natural de armazenamento de listas encadeadas através de um programa denominado **`Criacao lista encadeada.asm`**. O programa usa uma estrutura comum em compiladores, denominada em inglês de *heap*. Trata-se de uma estrutura para dar apoio à alocação dinâmica de memória. O *heap* estrutura funciona de forma similar à pilha apontada pelo registrador $sp. Contudo, o *heap* deve ocupar, por convenção do montador MARS, os endereços do mapa de memória a partir de 0x10040000. O programa usa o registrador $fp como o apontador da "heap". O objetivo deste trabalho é partir do programa **`Criacao lista encadeada.asm`** e desenvolver funções para manipular a lista dada, conforme descrito a seguir.
3. No trabalho, cada grupo deve criar duas funções **recursivas**. Estas devem ser chamadas a partir de um programa principal, que pode ser uma modificação do programa de base fornecido junto com a subrotina de criação de listas (ver item 4 abaixo). A especificação das funções recursivas é a seguinte:
   3.1. A primeira função recursiva (de nome **`ins_ult_el`**) deve receber como parâmetro um ponteiro para o início da lista encadeada alvo e um número inteiro. Ela deve criar um nodo de lista (alocando memória no *heap*) que conterá o número no seu campo de valor. Deve também inserir o novo nodo no final da lista (o final é onde a fila acaba, este nodo deve ter o campo de enlace com o valor nulo, ver desenho abaixo). A função não precisa retornar nenhum valor (assume-se que a criação e inserção do novo nodo sempre é realizada). Notem que a lista encadeada é simplesmente encadeada (existem listas duplamente encadeadas). Assim, ajuda se a função receber como parâmetro não um ponteiro para o primeiro elemento da lista, mas um ponteiro para

este ponteiro, para permitir por exemplo criar uma lista se o ponteiro inicial for vazio (=0). Cuidado com os casos especiais, tais como tentar inserir o primeiro elemento da lista ou criar a lista a partir de um ponteiro inicial vazio (3 pontos).



3.2. A segunda função recursiva (de nome **`soma_val`**) deve receber como parâmetro o ponteiro para a lista e retornar o somatório de todos os valores armazenados nos nodos desta. Cuidado com os casos especiais, tais como lista vazia. A função retorna o valor da soma (3 pontos). Por exemplo, se a rotina for chamada para a lista acima antes da inserção, retornará o valor 12 (1+2+4+5), e se for chamada depois da inserção sobre a mesma lista, retornará 32 (1+2+4+5+20).

4. A partir do programa principal exemplo citado acima e da função **`gera_lista`** contida nele, definam um novo programa principal (2 pontos) que chama as funções que vocês criaram e realize várias chamadas destas para demonstrar suas funcionalidades.
5. Respeite as convenções de Passagem de argumentos – todos os parâmetros devem ser passados através da pilha, apontada pelo registrador $sp. Todos os valores retornados devem voltar também através da pilha (não usem registradores aqui, **é regra!!**).
6. Pesquise na documentação disponível (Internet, livros de programação, etc.) uma solução em pseudo-código para funções recursivas de inserção, remoção e pesquisa de conteúdos em listas encadeadas Um exemplo é o site https://www.ics.uci.edu/~pattis/ICS-21/lectures/llrecursion/lecture.html, onde se descrevem pseudo-códigos de vários algoritmos recursivos de manipulação de listas encadeadas. Note que:
   6.1. A lista usa encadeamento simples e não duplo;
   6.2. As listas criadas por **`gera_lista`** não são ordenadas por conteúdo.
7. Responda: (1) Qual o número do registrador $sp no conjunto de registradores do MIPS e qual o seu valor inicial (atribuído pelo simulador MARS)? (2) Qual é o primeiro valor escrito na pilha, e qual o significado do mesmo? (3) Mostre o conteúdo da pilha ao entrar na terceira chamada aninhada de alguma recursão (use a opção File→Dump Memory do simulador MARS). (4) Qual o conteúdo do registrador $sp neste momento? (5) Isto implica 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 re-estabelecido?
8. Formato do trabalho e entrega: O trabalho deverá ser entregue via sala do Moodle até a data de 10/05/2018 (quinta-feira), contendo o código fonte da aplicação **COMENTADO SEMANTICAMENTE**, as respostas das perguntas do 7 acima e exemplos de telas do simulador MARS, mostrando alguns dos passos da execução do programa, com comentários.
9. Valor do trabalho: Este trabalho vale 25% da nota de Trabalho Prático da disciplina. Lembrando que TP corresponde a 40% da composição da nota de G1, este TP2 corresponde a 10% do G1.