UNIVERSIDADE DE EVORA
Escola de Ciências e Tecnologias

Engenharia Informática
Arquitetura de Sistemas e Computadores I
2017/2018

Calculadora em Notação Polaca Inversa

Docente: Miguel Barão
Discentes: Cláudia Dias - 35308
Ana Ferro - 39872

1 Introdução

Pretende-se desenvolver um programa em assembly MIPS para correr no simulador MARS. Este programa consiste em recriar uma calculadora que opere na notação polaca inversa, onde os operandos são introduzidos em primeiro, seguidos dos operadores.

O programa deverá receber strings com a expressão que se pretende calcular, retornando o seu resultado através da pilha na consola. Os dados podem ser introduzidos um a um ou numa só linha, sendo estes separados por espaços ou por quebras-de-linha. Dependente da maneira como são introduzidos os dados, assim será o funcionamento do programa, caso seja introduzido um a um, cada vez que é dada uma quebra-de-linha deve-se mostrar o estado da pilha na consola, ao contrário de quando é introduzida a expressão, que só deverá mostrar o resultado no final da execução da mesma.

Input: String composta por operandos e/ou operadores (o utilizador pode decidir introduzir um a um ou numa só linha)

Output: Estado da memória da calculadora na forma de uma pilha (a quando existe de uma quebra-de-linha)

2 Funções

2.1 *MENSAGEM()*

NOME

MENSAGEM - Imprime a mensagem na consola

DESCRIÇÃO

Esta função tem como objetivo imprimir na pilha uma mensagem com os autores, a disciplina e o ano letivo 2017/2018.

Será executada uma única vez no início do programa não fazendo parte do loop principal, uma vez que não é do nosso intuito imprimir esta mensagem na consola cada vez que é introduzido um novo dado na calculadora. Esta função tem como argumento uma string, ou seja, a mensagem descrita anteriormente. Como esta mensagem vai permanecer inalterada desde o inicio ao fim do programa será tratada como uma variável global, sendo definida no data segment onde teremos acesso a ela através da label: "Msg:".

2.2 *MAIN()*

NOME

MAIN - Repetição do programa

DESCRIÇÃO

Esta função é a responsável pela execução do loop principal. Nela está contida todas as outras funções(à exceção da função MENSAGEM()), sendo a "chave" para execução de um programa contínuo.

2.3 *PEDIR_INPUT()*

NOME

PEDIR_INPUT - Permite a introdução do input

DESCRIÇÃO

Esta função possibilita a introdução do input pelo utilizador. Para tal aloca espaço na pilha e de seguida executa-se um syscall que irá gerar uma exceção, e que durante a qual se poderá escrever o input. Quando se carregar no enter ou quando excedermos o tamanho da pilha, o sistema operativo retorna.

2.4 *VAL_INPUT()*

NOME

Val_input - Avalia o input introduzido

DESCRIÇÃO

Esta função tem como argumento o input introduzido pelo utilizador e tem como objetivo avaliar a sua validade. Para tal iremos recorrer a diversos branches que nos permitirão mudar o rumo de execução do nosso programa dependendo de qual condição é satisfeita. Pode existir três situações diferentes:

- Não cumprir notação polaca inversa;
- Erros de sintaxe (operadores mal escritos);
- Código válido;

Para identificar a primeira situação teremos que confirmar os seguintes requisitos:

-Se os operadores são escritos antes dos operandos;

-Se existem operandos suficientes para a operação pretendida. É de notar que existem dois tipos de operadores, os operadores unários que têm só um argumento e os operadores binários que necessitam de dois argumentos.

Para identificar a segunda situação, teremos que comparar o primeiro caractere com o código identificador dos operadores (códigos esses já definidos no data segment). Se estas situações se confirmarem, serão impressas na consola da pilha da calculadora as seguintes mensagens de erro:

- Primeiro caso: "Input inválido: Não cumpre as normas da notação polaca inversa"
- Segundo caso: "Input inválido: Operador desconhecido"

Se não se verificar as situações anteriores, verificamos que o código introduzido é válido. Ao chegarmos a esta conclusão o programa pode continuar para a função seguinte, função LER_INPUT().

2.5 *LER_INPUT()*

NOME

LER_INPUT - Identifica os operandos e operadores

DESCRIÇÃO

Esta função tem como objetivo descodificar o que o input pretende representar, ou seja identificar os operandos e os operadores. Para tal vai usar um mecanismo em cadeia composto por diversos branches que vai avaliar cada caracter um a um.

O argumento desta função é uma string (input do utilizador), que pode ser

constituída por operandos (números constituídos pelos algarismos de 0-9), operadores, espaços e quebras-de-linha.

Primeiro, começa por avaliar se se trata dos comandos off e clear, saltando para a função respetiva se tal se confirmar. De seguida, averigua se se trata de um espaço ou de uma quebra-de-linha(símbolo de que a string chegou ao fim).

Para confirmar que se trata de um espaço(Tab), compara-se ao código identificador de espaço que foi anteriormente definido no data segment. Se tal se confirmar avançamos um caracter na string e saltamos para o início da função LER_INPUT().Este mecanismo repete se para a quebra-de-linha. Ao deparar-se com uma quebra-de-linha, saltamos para a função PILHA().

Uma vez que tanto os operadores como os operandos, nesta primeira fase, ainda se encontram codificados de acordo com o código ascii, o modo de concluir que se trata de um operador é descobrir se esse carater é maior que 0x60 (ou seja, segundo a tabela ascii de a-DEL) e menor que 0x30(ou seja, de NUL-/). Se tal se confirmar faz-se um jump para a função OPERADORES().

Se nenhuma destas condições se confirmar o programa salta para a função CONVERSAO().

2.6 *CONVERSAO()*

NOME

CONVERSAO - Converte código ascii para código binário(representativo do algarismo inicial)

DESCRIÇÃO

Esta função recebe como argumento o código ascii correspondente a um algarismo. O seu objetivo é converter esse código para o código binário do verdadeiro valor do algarismo.

Para chegar a esse resultado temos que subtrair 0x30(ou seja 48) a esse código. Por exemplo, o algarismo 1 está codificado como sendo 0x31, ao subtrairmos 0x30 ficamos com o código a 1. Este mecanismo repete-se para os restantes algarismos.Depois de executada a conversão guarda o número na pilha(stack). Retorna à função LER_INPUT().

Para números com dois ou mais algarismos é necessário converte-lo num só código, representando assim um único número. Para tal criou-se um loop que só é executado se o caracter seguinte da string for também um algarismo.

Nesse loop multiplicamos por 10 o algarismo anteriormente convertido, de seguida avançamos na string e convertemos também esse caracter, somando-os no fim e guardando o número na pilha. Se o caracter seguinte não for um algarismo retornamos ao ínicio da função LER_INPUT(). Se for um algarismo saltamos para a label "Algarismos_plus:"(localização do ínicio do loop).

2.7 *OPERADORES()*

NOME

OPERADORES - Identifica qual é a operação pretendida

DESCRIÇÃO

A função recebe como argumento o código ascii de um operador. O objetivo desta função é identificar qual operador representa.

Para descobrir qual operador se trata recorre-se a um mecanismo em cadeia composto por diferentes branches que nos permitirão mudar o rumo de execução do nosso programa.

Dependente de qual condição é satisfeita, assim será o jump para a função seguinte. Por exemplo, se o programa identificar o operador como sendo o da adição, o programa salta para a função da operação correspondente, função ADD().

2.8 *ADD()*

NOME

ADD - Realiza a adição

DESCRIÇÃO

Esta função recebe como argumentos dois operandos e tem como objetivo executar a soma entre eles. Esta função vai buscar os seus argumentos à pilha(últimos dois valores a serem introduzidos) e guarda o resultado da operação também na pilha. No final retorna à função LER_INPUT().

2.9 *SUB()*

NOME

SUB - Realiza a subtração

DESCRIÇÃO

A função recebe dois operandos como argumentos e tem como objetivo realizar a operação de subtração entre eles. Esta função descreve o mesmo comportamento que a função anterior, indo buscar os seus argumentos e guardando o resultado na pilha. Retorna à função LER_INPUT().

2.10 *DIV()*

NOME

DIV - Realiza a divisão

DESCRIÇÃO

Esta função recebe dois operandos como argumentos provenientes da pilha(stack) e tem como objetivo realizar a divisão dos mesmos, guardando o resultado da operação na pilha. No final, retorna à função LER_INPUT().

2.11 *MUL()*

NOME

MUL - Realiza a multiplicação

DESCRIÇÃO

Por se tratar de um operador binário, esta função recebe dois operandos como argumentos (provenientes da pilha) e pretende realizar a multiplicação dos mesmos, guardando o resultado na pilha. Retorna à função LER_INPUT().

2.12 *SWAP()*

NOME

SWAP - Troca a posição dos últimos dois valores guardados em memória

DESCRIÇÃO

Esta função recebe os últimos dois operandos provenientes da pilha como argumentos e tem como objetivo trocar a posição dos mesmos. Essa alteração é guardada na pilha e retorna à função LER_INPUT().

2.13 *NEG()*

NOME

NEG - Calcula o simétrico

DESCRIÇÃO

O objetivo desta função é calcular o simétrico. Para tal vai buscar o último valor introduzido na pilha como argumento e guarda o resultado na mesma. No final, retorna à função LER_INPUT().

2.14 *DUP()*

NOME

DUP - Duplica o último valor guardado em memória

DESCRIÇÃO

Por tratar-se de um operador unário, esta função possui apenas um argumento (operando do topo da pilha). Com esta função pretende-se duplicar o argumento, adicionando à pilha o seu duplicado. Retorna à função LER_INPUT().

2.15 *CLEAR()*

NOME

CLEAR - Apaga os valores guardados em memória

DESCRIÇÃO

Esta função limpa toda a pilha, ou seja, elimina todos os registos da pilha, ficando esta vazia. Após a sua execução, o programa salta para a função PILHA().

2.16 *DEL()*

NOME

DEL - Apaga o último valor guardado em memória

DESCRIÇÃO

Esta função tem como objetivo remover um operando do topo da pilha (em caso de engano), recebendo como argumento esse mesmo operando. Retorna à função PEDIR_INPUT().

2.17 *OFF()*

NOME

OFF - Termina o programa

DESCRIÇÃO

O objetivo desta função é desligar a calculadora. Para tal é executado um syscall($v0 igual a 10) onde se dará por terminado o programa.

2.18 *PILHA()*

NOME

PILHA - Imprime os valores guardados em memória

DESCRIÇÃO

Esta função tem como objetivo imprimir o estado atual da pilha na consola da calculadora, para tal vai buscar o operando do topo da pilha(stack). Se a pilha conter mais do que um número, vai-se executar um ciclo que imprime todos os outros números nela contidos.

É executada no início do programa, mostrando que a pilha está vazia; sempre que é identificado uma quebra de linha e quando se executa a função CLEAR().

3 Conclusão

Este trabalho foi bastante útil para a nossa aprendizagem e para a consolidação dos nossos conhecimentos.
Apesar de estar aqui mencionada e descrita a função VAL_INPUT() não a conseguimos implementar no nosso programa, no entanto, optamos por a incluir no relatório uma vez que é uma função importante para a execução correta do programa.
Para além disso, o programa ainda possui os seguintes erros:

- Quando inserido como input uma linha vazia o programa não duplica o último número introduzido na pilha;

- Quando, por exemplo, introduzimos a operação "1 2 3 4 + - -"é suposto realizar "1+2-3-4"no entanto está a realizar 3+4-2-1".

É também de notar que a calculadora não suporta a vírgula flutuante, o que significa que apenas funciona para números inteiros.