3º Relatório de Laboratório de Montagem

Bruno Cardoso Maciel RA: 141046279

Rafael Stefanini Carreira RA: 141040726

27/10/2015

---

1. Introdução

Notamos que o código-fonte de aplicações em Assembly é demasiadamente extenso, e conforme a complexidade do exercício proposto, o mesmo pode se tornar muito complicado de ser lido e compreendido. Dessa forma, podemos nos utilizar dos novos recursos aprendidos para escrever um código mais limpo e legível, sem comprometer seu funcionamento ou eficiência.

Um recurso muito útil e indispensável na programação em assembly é a utilização da pilha (stack). Sua utilização é muito simples, a partir do registrador esp, que sempre aponta para o ultimo elemento inserido na pilha, podemos realizar operações de inserção e remoção seguindo o padrão LIFO (last in first out), onde o ultimo elemento inserido é o primeiro a ser removido.

Através das instruções “call” e “ret”, que por sua vez utilizam a estrutura pilha, podemos criar funções na linguagem assembly. As funções criadas dessa maneira podem ser chamadas de qualquer lugar, e após a sua execução, o fluxo da execução segue imediatamente após a chamada da função. Dessa maneira, através de chamadas de funções, podemos tornar o código muito mais organizado e legível.

2. Objetivo

Através dos recursos aprendidos até o momento, escrever um programa que receba uma frase digitada pelo usuário, obtenha a quantidade de palavras presentes na frase e retorne este valor ao usuário.

3. Metodologia

Todas as funções utilizadas no exercício proposto foram criadas de tal forma que cada uma comece com uma label, que dá nome à função, e termine com a instrução “ret”. Dessa maneira, o inicio do programa é composto somente de instruções “call” com o nome da função desejada como parâmetro.

O funcionamento da instrução “call” é bem simples, ela salva o endereço da próxima instrução na pilha e em seguida, pula para o endereço da label informada. Após executar toda a função, a instrução “ret” faz um pulo para o primeiro valor da pilha, que no caso é o endereço da próxima instrução após a chamada da função.

Seguindo as chamadas das funções obtemos a seguinte execução:

- Imprime uma mensagem de entrada
    - Através da *syscall write* imprime uma mensagem de entrada na tela.

- Solicita uma frase ao usuário
    - Imprime uma mensagem solicitando uma frase ao usuário.

- Armazena a frase digitada
    - Através da *syscall read*, o valor informado é armazenado na forma de cadeia de caracteres na variável ‘string’.

- Conta a quantidade de palavras na frase.
    - Para realizarmos a contagem de palavras na frase digitada, usamos o método de contar a quantidade de espaços em toda a frase, e guardamos o valor no registrador ecx que é utilizado como um acumulador inicializado em 1.
    - Para realizar a contagem, usamos o registrador ebx como um contador iniciado em 0. São feitas comparação entre o caractere que se encontra na posição determinada por ebx e o caractere espaço (‘ ‘), caso sejam iguais, o acumulador ecx é incrementado. As comparações continuam até que o caractere na posição de ebx é igual a ‘\n’ (quebra de linha), sinalizando o fim da frase. Em seguida colocamos o caractere ‘\0’ após o ‘\n’, marcando assim o fim de uma *string*.
    - Após o processo a quantidade de palavras é armazenado na variavel ‘num’.

Converte o valor numérico obtido em ASCII

- O valor armazenado em ‘num’ não pode ser diretamente informado ao usuário pois se trata de um valor numérico, por isso é necessário realizar a conversão do número para caracteres em ASCII. Esse processo é feito através de divisões sucessivas por 10, e para tal, usamos novamente o ebx como um contador e o ecx como o divisor, com valor fixado em valor 10.
- A cada iteração, o valor em eax é divido por ecx e o resto da divisão é armazenado em edx, que por sua vez é convertido em caractere (ASCII) e salvo na variável “num”. O quociente fica armazenado em eax para as futuras iterações e ebx é incrementado para armazenar os restos de forma continua (sem sobreposições).
- Ao fim processo, os restos das divisões estarão armazenados em ‘num’, representando o número em forma de caracteres, porém, em ordem inversa, e a quantidade de caracteres a serem impressos estará em ebx.

- Imprime o valor convertido na tela
  - Para imprimir o valor na tela, nos utilizaremos a *syscall write,* que por sua vez, utiliza o registrador ebx. Dessa forma, antes de cada impressão, empilhamos o valor contido em ebx por meio do comando *“push”,* liberando assim o registrador a ser usado para imprimir cada caractere usuário, sem que houvesse perda do valor anteriormente armazenado nele. Feito isso, retornamos o valor à ebx utilizando o instrução *“pop”*(retorna o primeiro valor da pilha).
  - Durante esse processo, utilizamos o registrador ebx como contador e a cada impressão da variável “num” na posição ebx, o seu valor é decrementado. O processo de impressão dos elementos da variável chega ao fim quando o contador ebx chega a 0, dessa forma temos o número impresso na tela.

- Finaliza o programa.
  - Imprime uma linha indicando o fim da interação do usuário com o programa.
  - Finaliza a execução do programa através da *syscall exit*

4. Resultados

Alguns resultados obtidos com o programa.

Frase Digitada	Quantidade de Palavras
Quinta Feira, 22 de Outubro de 2015	7
Laboratório de Montagem	3
Bacharelado em Ciência da Computação	5
Informática	1

5. Discussão

Podemos observar claramente a contribuição das funções na organização de um programa. Esse recurso é amplamente utilizado em todas as linguagens e trazem diversos benefícios, tal como a reutilização de código e a melhora na legibilidade.

Outro fato a ser considerado é a necessidade da conversão do valor numérico em cadeia de caracteres, pois imprimir a quantidade de palavras contidas na variável “num”, logo que armazenada, nos traria um problema sempre que a quantidade de palavras da frase fosse maior que 9, pois não haveria um único caractere que representaria seu valor, e assim, seria mostrado pelo programa um caractere correspondente ao valor da variável na tabela ASCII. Por exemplo, se o programa encontrasse 10 palavras em determinada frase, o valor 10 + ’0’ estaria armazenado em “num”, porém não existe um único caractere que represente o número 10.

Tendo em vista disso, devemos sempre realizar a conversão de valores numérico antes de mostrá-los.

6. Conclusão

Obtemos resultados satisfatórios quanto ao objetivo proposto. Por meio das novas técnicas aprendidas, como procedimentos e pilhas, criamos procedimentos para cada função do programa, e dessa maneira, desenvolvemos um programa com o código legível e de fácil compreensão, sem que comprometesse o objetivo e/ou a eficiência do projeto.

Outro recurso interessante que a pilha nos traz é a possibilidade de salvar valores sem a necessidade de criar novas variáveis. Apesar desse recurso ser muito útil, devemos nos atentar quanto a problemas que isso pode nos trazer, como por exemplo, ao empilhar um valor e não desempilhá-lo antes da instrução “ret”, essa instrução tentará pular para a primeira posição da pilha, e caso esse valor não seja um endereço válido, o programa certamente travará.