Trabalho Prático 2 CIC 116432
Software Básico
Prof. Bruno Macchiavello
1 o Semestre de 2021

1 Introdução
O trabalho consiste em implementar em C/C++ um método de tradução de uma linguagem de montagem simples para uma representação de código objeto. O tradutor a ser implementado será um Assembler da linguagem hipotética vista em sala de aula.

2 Objetivo
Fixar o funcionamento de um processo de tradução. Especificamente as etapas de carregar o programa em memória e programação IA-32.

3 Especificação

3.1 Montador

Realice modificações no MONTADOR do trabalho 1 de forma que o arquivo objeto indique um cabeçalho no seguinte formato:

H: PROG1
H: 12
H: 010010101010
T: 12 10 15 11 5 0 12 9 14 0 0 2

A seção T é a seção de texto onde deve ser indicado o trabalho traduzido a código máquina exatamente como era no trabalho 1. A primeira linha do cabeçalho indica o nome do programa, a segunda o tamanho dele em memória (em quantidade de espaços de memória) e a terceira a informação de realocação.

O montador da hora de executar agora deve receber um parâmetro “-r” antes do nome do programa de entrada (ex.: ./montador -r 1 myprogram.asm”). Se for indicado “-r 0” a informação de realocação deve ser mapa de bits. Se for indicado “-r 1” a informação de realocação deve ser lista de endereços.

Desta vez não será verificado se o montador consegue detetar erros (todos os programas de testes estarão livre de erros) e a seção de dados nos testes sempre virá depois da seção de texto.

3.2 Carregador

Fazer um programa chamado CARREGADOR. Que deve receber por linha de comando o nome do 1 a 3 arquivos objetos a serem carregados, a quantidade de chunks de memória disponível, o tamanho de cada chunk e o endereço inicial de cada chunk.

Por exemplo:

> carregador myprogram1.obj myprogram2.obj 3 15 10 12 1000 2500 5000

Indica que tem 2 programas a serem carregados “myprogram1.obj” e “myprogram2.obj”; e neste momento o computador tem 3 chunks de memória disponíveis cada um com tamanho 15, 10 e 12 respectivamente. E o primeiro chunk começa no endereço 1000, o segundo no 2500 e o terceiro no 5000. Sempre serão usados números decimais (não hexadecimais).

O programa deve alocar o programa em “memória”. Isso de forma simulada. O programa deve dar como saída um arquivo TEXTO sem cabeçalho para cada programa com extensão .saída (mantendo o mesmo nome). Nela deve estar o código objeto do programa com os endereços relativos modificados para o lugar (lugares) onde foi carregado o programa. No formato a seguir (onde a primeira coluna indica o endereço):

1000 12 1010
1002 15 1011
1004   5 1000
1006 12 1009
1008 14
1009   0
1010   0
1011   2

Seguindo o exemplo anterior, suponhamos que “myprogram1.obj” ocupa 22 espaços de memória e “myprogram2.obj” ocupa 11 espaços de memória. O carregador deve primeiro verificar cada programa na ordem indicada. Ou seja, começar a alocar pelo primeiro. Então ele deve começar por “myprogram1.obj”. Se é possível alocar ele num ÚNICO chunk então o primeiro CHUNK disponível onde caiba o programa deve ser usado. No caso do exemplo, não tem nenhum CHUNK de 22 espaços então não tem como alocar o programa num único CHUNK. O carregar então deve verificar se juntando CHUNKS consegue alocar. Neste caso sim, juntando os 2 primeiros ele consegue. Depois verifica se a alocação do próximo programa. No exemplo, estão sobrando 3 espaços de memória do segundo CHUNK (já que 7 espaços foram utilizados para o primeiro programa) e o terceiro CHUNK de 12 espaços. Logo, o segundo programa é alocado no terceiro CHUNK. O carregador deve mostrar na tela:

> PROG1 utilizando 2 CHUNKS. Endereços iniciais: 1000 2500
> PROG2 utilizando 1 CHUNK. Endereços iniciais: 5000

O arquivo de saída do “myprogram2.obj” deve estar corrigido para o endereço 5000. E o do “myprogram1.obj” para o 1000 nos primeiros 12 espaços de memória, e para 2500 nos seguintes. PROG1 e PROG2 sao respectivamente os nomes indicados nos cabeçalhos dos arquivos objeto.

Caso não seja possível alocar um programa, então deve aparecer:

> PROG3 NAO FOI POSSIVEL ALOCAR

E não gerar saída.

3.3 Calculadora

Deve realizar um terceiro programa em Assembly IA-32 (linux). Este programa deve ser chamado calculadora.asm e deve ser capaz de ser montado utilizando o nasm e ligado utilizando o ld.

Ao iniciar o programa deve aparecer um menu de opções perguntado se deseja:

1 - soma,
2 - subtração,
3 - multiplicação,
4 - dividir,
5 - potenciação,
6 - fatorial
7 - soma (concatenaçao) de strings
8 - multiplicação (repetição) de strings
9 - sair.

A opção 9 termina o programa. Depois, cada opção deve pedir dois operandos (sem pedir mensagem, após digitar as opções de 1 a 5 o programa fica esperando digitar um número+ENTER e depois outro número+ENTER). Os números podem ser positivos ou negativos sempre em decimal. Após a opção 6 deve esperar só um número. Os números podem ter vários dígitos. A calculadora deve trabalhar com números de 16 bits (deve ser verificado que os números estejam na faixa de 16 bits com sinal). A calculadora então deve: 1 - mostrar a soma dos 2 números, ou 2 - mostrar a subtração do primeiro pelo segundo (podendo dar negativo), ou 3 - mostrar a multiplicação dos 2 números, usando MUL em IA-32. Deve verificar se deu overflow (se a saída é maior a 16 bits - indicando o resultado obtido e a mensagem DEU OVERFLOW), ou 4 - mostrar a divisão inteira do primeiro pelo segundo número e o resto, ou 5 - elevar o primeiro número a potência do segundo . Deve verificar se

deu overflow (se a saída é maior a 16 bits - indicando o resultado obtido e a mensagem DEU OVERFLOW), ou 6 - apresentar o fatorial do número. Deve verificar se deu overflow (se a saída é maior a 16 bits - indicando o resultado obtido e a mensagem DEU OVERFLOW). Não pode usar a MAC.O. A leitura e saída de dados deve ser feita por funções. Ter uma função para ler números com sinal e outra para escrever números com sinal. Cada uma das operações deve ser feita dentro de uma função. A passagem por parâmetros das funções DEVE ser feita pela pilha.

As opções 7 e 8 são para trabalhar com strings. Deve ter uma função para cada opção, assim como as funções para ler e escrever strings. A opção 7 deve esperar o usuário digitar um STRING + ENTER e depois digitar um segundo STRING+ENTER. A opção 8, o usuário vai digitar primeiro um STRING+ENTER e depois um número+ENTER. Assumir que os strings nunca vão ser maiores a 20 caracteres. A opção 7 deve concatenar os 2 strings (SEM adicionar espaço entre eles) num único string e mostrar isso na tela. A opção 8 deve repetir o string como se fossem vários concatenando os strings até às vezes indicadas pelo número. Exemplo: Se o usuário digitar “Assembly” e “3”, então a saída deve ser um string “AssemblyAssemblyAssembly”. Assumir que no máximo o número será 9 (ou seja um único dígito, para poder reservar no máximo um string de tamanho 20x9 para a saída).