2º Trabalho Prático
CIC 116432 – Software Básico
Prof. Bruno Macchiavello
1° Semestre de 2016

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 e IA-32.

2 Objetivo

Fixar o funcionamento de um processo de tradução de código. Na parte final os alunos podem aperfeiçoar o seu conhecimento sobre formato de arquivos.

3 Especificação

3.1 Tradutor

O programa tradutor (`tradutor.c`) deve receber um arquivo (`arquivo.asm`) como argumento (para facilitar, o nome do arquivo deve vir com a extensão na linha de comando!). Este arquivo deve estar na linguagem Assembly hipotética vista em sala de aula (com algumas modificações, a serem descritas posteriormente), sendo que o código deve estar separado em seções de dados e texto. O tradutor deve ser capaz de avaliar EQU e IF, e a diretiva CONST deve aceitar números positivos, tanto em decimal como em hexadecimal (mediante o formato 0xNUMERO). Não será avaliado detecção de erros léxicos, semânticos ou sintáticos; porém, é fortemente recomendado que os alunos utilizem como base o trabalho realizado anteriormente. A linguagem hipotética é formada por um conjunto de apenas 18 instruções, conforme Tabela 1, e algumas diretivas.

O programa deve entregar uma saída. A saída deve ser um arquivo em formato texto (`arquivo.s`) que deve ser a tradução do programa de entrada em Assembly IA-32. Observe que as seções de texto e dados da linguagem de montagem hipotética devem ser convertidas para o novo formato de forma a conservar o comportamento

correto do programa. A diretiva SPACE pode ser traduzida dentro da seção BSS, ou dentro da seção DATA iniciando os valores com zero.

Para entrada e saída de dados, a linguagem hipotética possui, como sempre, a leitura/escrita de números inteiros mediante as instruções INPUT e OUTPUT. Porém, agora existem, para ler/escrever caracteres, as instruções C_INPUT e C_OUTPUT; e para trabalhar com STRINGS, as instruções S_INPUT e S_OUTPUT. As instruções com caracteres funcionam exatamente igual às de inteiros, com a diferença que o valor lido deve estar em ASCII; já as instruções com string possuem 1 operando, e podem ler mais de um caractere. O argumento é a posição de memória, nome da variável, onde a string será lida/escrita. No seu programa, deve existir então 6 sub-rotinas: *LerInteiro*, *EscreverInteiro*, *LerChar*, *EscreverChar*, *LerString*, *EscreverString*. As funções devem estar em Assembly IA-32. No arquivo de saída (`arquivo.s`), as instruções de INPUT, OUTPUT, C_INPUT, C_OUTPUT, S_INPUT e S_OUTPUT devem ser trocadas por chamadas às sub-rotinas equivalentes mediante o comando CALL, como visto em sala de aula. As funções *LerChar* e *EscreverChar* devem ler ou escrever um único digito/letra. A função *LerString* deve ler todos os caracteres digitados pelo usuário e deve devolver em AX (ou EAX) o tamanho da STRING. As funções *LerInteiro* e *EscreverInteiro* devem ler inteiros de vários dígitos até o usuário digitar ENTER. Assumir que inteiros sempre terão 32 bits e são números positivos. As funções NÃO podem ser cópias da `io.mac`.

Recapitulando: a saída do seu tradutor é um arquivo Assembly IA32 que pode ser montado utilizando o NASM em Linux e quando montado deve ser executável, caso contrário sua tradução é considerada incorreta.

3.2 Gerador de Executável ELF

Realizar um programa gerador de arquivo objeto em formato ELF (`gerador_elf.c`) que recebe como entrado um arquivo em IA-32 (por exemplo o arquivo de sáida da parte 1). A saída do programa deve ser um arquivo executável em formato ELF32, capaz de ser executado em qualquer máquina INTEL 386 ou superior, rodando **SÓ NO LINUX**. Para isso recomenda-se o uso da biblioteca "libelf" (linguagem C) ou "ELFIO" (linguagem C++) para a criação do arquivo ELF32. Somente é necessário verificar os OPCODES das 18 instruções equivalentes em IA-32 do Assembly hipotético e das instruções utilizadas nas funções de ler e escrever inteiro e das instruções utilizadas nas sub-rotinas de I/O. Para verificar OPCODES, acessar uma Referência ASM[1] ou verificar o manual da INTEL. Uma outra alternativa é o uso do programa *objdump*, que funciona em Linux e Mac OS X. Outra ferramenta para auxílio é o *readelf* Lembre que as diretivas são avaliadas durante a montagem/ligação e não geram código objeto. Para as sub-rotinas de entrada de dados, o programa pode já ter as sub-rotinas pré-compiladas e somente copiar o código quando forem chamadas.

Recapitulando: seu programa vai compilar um arquivo em Assembly IA-32 para

[1] http://www.mathemainzel.info/files/x86asmref.html#call

um executável ELF32 que será então testado em Linux. Seu programa deve executar corretamente, sem dar falha de segmentação ou similares.

4 Avaliação

O prazo de entrega do trabalho é 26 de Junho de 2016. A entrega consistirá em:

- Código-fonte completo e comentado com instruções de compilação dos programas de tradução e simulação;

A forma de entrega é pelo Moodle. O trabalho pode ser feito individualmente ou em dupla.

Tabela 1: Intruções e diretivas.

Instruções				
Mnemônico	Operandos	Código	Tamanho	Descrição
ADD	1	1	2	ACC <- ACC + MEM[OP]
SUB	1	2	2	ACC <- ACC - MEM[OP]
MULT	1	3	2	ACC <- ACC * MEM[OP]
DIV	1	4	2	ACC <- ACC / MEM[OP]
JMP	1	5	2	PC <- OP
JMPN	1	6	2	Se ACC < 0, PC <- OP
JMPP	1	7	2	Se ACC > 0, PC <- OP
JMPZ	1	8	2	Se ACC = 0, PC <- OP
COPY	2	9	3	MEM[OP2] <- MEM[OP1]
LOAD	1	10	2	ACC <- MEM[OP]
STORE	1	11	2	MEM[OP] <- ACC
INPUT	1	12	2	MEM[OP] <- STDIN
OUTPUT	1	13	2	STDOUT <- MEM[OP]
STOP	0	14	1	Encerrar execução.
C_INPUT	1	15	2	MEM[OP] <- STDIN
C_OUTPUT	1	16	2	STDOUT <- MEM[OP]
S_INPUT	1	17	2	MEM[OP] <- STDIN
S_OUTPUT	1	18	2	STDOUT <- MEM[OP]
Diretivas				
SECTION	1	-	0	Marcar início de seção de código (TEXT) ou dados (DATA).
SPACE	0/1	-	variável	Reservar 1 ou mais endereços de memória não-inicializada para armazenamento de uma palavra.
CONST	1	-	1	Reservar memória para armazenamento de uma constante inteira de 16 *bits* em base decimal ou hexadecimal.
EQU	1	-	0	Cria um sinônimo textual para um símbolo
IF	1	-	0	Instrue o montador a incluir a **linha seguinte** do código somente se o valor do operando for 1