Implementação de uma calculadora simples em linguagem de montagem

Elyabe Alves, *Ciência da Computação, UFES,* e Thayza Sacconi, *Ciência da Computação, UFES*

Abstract—The present article aims to show the methods, processes and tools used in the development of a low-level language calculator. It has been divided into sections whose purposes refer so much to describing the program as a means of orienting the user at the moment of execution, as well as exposing the ways and techniques used in each operation supported by the project in question.

Index Terms — Calculator, NASM, Assembly, Assembly Language, Computer Architecture.

Resumo — O presente artigo tem por objetivo apresentar os métodos, processos e ferramentas empregados no desenvolvimento de uma calculadora em linguagem de baixo nível. Foi divido em seções cujos propósitos remetem tanto a descrever o programa de forma simplificada como meio de orientação ao usuário em momento de execução, quanto expor as maneiras e técnicas utilizadas em cada operação suportada pelo projeto em questão.

Palavras-chave — Calculadora, NASM, Assembly, Linguagem de Montagem, Arquitetura de Computadores.

1 Introdução

MÁQUINAS de cálcular sempre fascinaram o homem. Desde muito cedo, sistemas de contagens foram criados com o propósito de facilitar nossas vidas. Com este objetivo surgiu o computador, uma máquina revolucionária presente em todos os setores e âmbitos da sociedade moderna. Não obstante a isso, formas de se comunicar e instruir este aparelho foram desenvolvidas. Inicialmente com comandos primitivos - cartões perfurados, por exemplo, -, passando pela linguagem de máquina até as linguagens dotadas de alto nível de abstração existentes atualmente. Ainda hoje a Assembly é empregada nas mais diversas áreas em sua essência. Visando estreitar esse contato com tal linguagem, foi proposto este projeto, ainda que simples em sua forma, no entanto, de grande valia para o aprendizado.

Dezembro 15, 2017

2 Descrição do programa

O ArchCalc é uma calculadora simples cujo objetivo é realizar operações matemáticas aritméticas básicas. É composto de dois arquivos, a saber, um de extensão .asm, o qual contém o código que efetivamente será executado; e um arquivo do tipo *make*, útil para agilizar o processo de compilação. Todo o código foi escrito em Assembly, e deve ser compilado usando o Netwide Assembler (NASM), um popular montador para arquitetura Intel x86*, bastante empregado em sistemas com kernel baseado em linux. Durante sua execução, chamadas às funções *printf*, e *scanf* da linguagem C são utilizadas a fim de tornar a implementação menos dispendiosa.

2.1 Organização estrutural

O código está modularizado em diferentes componentes cujas funcionalidades estão bem definidas. As entradas são todas em cadeias de caracteres que posteriormente serão convertidas em números do tipo ponto flutuante, desde que tal cadeia seja não vazia, isto é, que o usuário tecle *enter* antes que qualquer dado seja inserido fornecido. Caso isso ocorra em qualquer instância do programa, quer seja na entrada de operandos, quer seja na entrada do operador, o programa é então abortado e sua execução é finalizada. Vale ressaltar ainda que o resultado do último cálculo realizado é considerado como a primeira entrada da próxima operação, se esta existir. O projeto suporta as seguintes operações matemáticas: adição (+), subtração (-), multiplicação (*) e divisão (/).

**Código compilado em máquina com Sistema Operacional Ubuntu*

2.2 Conversão

Dada uma string representando um inteiro, a conversão é realizada percorrendo-se toda a string e convertendo cada caracter com sua potencia de base 10 equivalente utilizando seu valor decimal representado em ASCII. Com cada soma parcial obtemos o valor inteiro correspondente. No entanto, uma interrogação surge: Como converteremos uma string, que contém a representação de um número real em seu respectivo valor em ponto flutuante? A seção a seguir apresenta a abordagem adequada.

2.2.1 O processo de conversão

Seja s uma string com $n+1$ caracteres representando um valor numérico em ponto flutuante a ser convertido. Para algum $1 \leq k \leq n$, se existir, o ponto flutuante aparecerá na seguinte forma

$$s: \ d_1 d_2 \ldots d_k . d_{k+1} d_{k+2} \ldots d_n \ .$$

Desse modo, desde que r seja o valor convertido em notação ponto flutuante, numericamente tem-se.

$$\begin{aligned} r &= d_1 d_2 \ldots d_k . d_{k+1} d_{k+2} \ldots d_n \times 10^0 \Longrightarrow \\ &= d_1 d_2 \ldots d_k . d_{k+1} d_{k+2} \ldots d_n \times 10^{n-k-(n-k)}. \end{aligned}$$

Deslocando-se o ponto no sentido da esquerda para direita, obtém-se

$$\begin{aligned} r &= d_1 d_2 \dots d_k d_{k+1} d_{k+2} \dots d_n \times 10^{-(n-k)} \implies \\ &= d_1 d_2 \dots d_k \dots d_n \times 10^{-(n-k)}. \end{aligned}$$

em que r, uma vez representado em uma string s, é um número inteiro exceto pela potência de 10 após o n-ésimo dígito, pelo que o algoritmo de conversão string para inteiro, representado no *figura 2* pode ser usado sem perda de generalidade. Finalmente, este valor é então normalizado fazendo-se a divisão do resultado da conversão inteira pela potência de base 10 conveniente, calculada tendo como base a posição do ponto flutuante na string correspondente. O fluxograma do módulo principal da calculadora é apresentado no A.

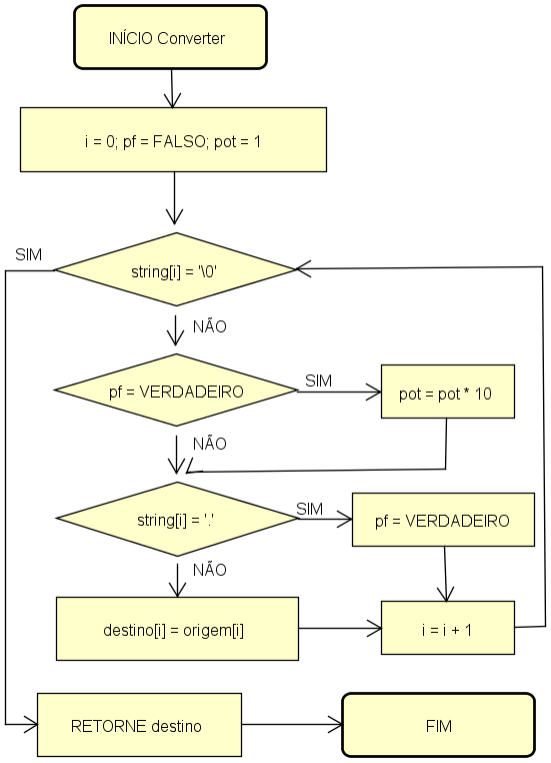

Fig. 1. Algoritmo de conversão parte 1: remoção do ponto flutuante. **pot** é uma variável usada na normalização e **string** é a cadeia a ser convertida.

Exemplo: Se $s = -145.678$, $p = 10$ uma constante aplicada no algoritmo de conversão $r = 0$ a variável onde o resultado numérico será armazenado, então:

$$\begin{aligned} s &= -145.678 \times 10^{3-3} \implies \\ &= -145678 \times 10^{-3} \end{aligned}$$

$$\begin{aligned} r \leftarrow r * p +' 1' &= 0 * 1 + 1 &&= 1 \\ r \leftarrow r * p +' 4' &= 1 * 10 + 4 &&= 14 \\ r \leftarrow r * p +' 5' &= 14 * 10 + 5 &&= 145 \\ r \leftarrow r * p +' 6' &= 145 * 10 + 6 &&= 1456 \\ r \leftarrow r * p +' 7' &= 1456 * 10 + 7 &&= 14567 \\ r \leftarrow r * p +' 8' &= 14567 * 10 + 8 &&= 145678 \end{aligned}$$

Normalizando...

$$r \leftarrow r * 10^{-3} = 145.678$$

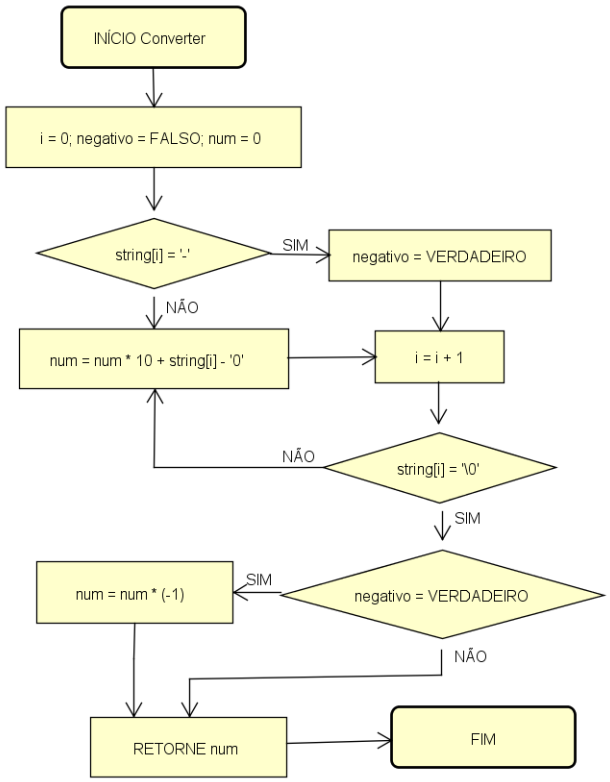

Fig. 2. Algoritmo de conversão *string* para inteiro. **num** é a variável que armazena o número resultado da conversão.

Como existe o sinal de '-' no início da string, então

$$r \leftarrow r * (-1) = -145.678.$$

Segue daí, que r está pronto para ser utilizado na operação em curso no programa. As operações são feitas basicamente em registradores, mais especificamente os registradores *xmm0, xmm1,. . . ,xmm8* para aritmética em ponto flutuante.

3 INSTRUÇÕES DE USO

De posse de um sistema computacional que atenda às restrições impostas, o usuário poderá executar o comando *make*, no diretório onde se encontram os arquivos do programa que será compilado e linkado com as biliotecas não nativas necessárias de forma automatizada. O usuário poderá optar por compilar de forma manual o código empregando os seguintes comandos:

nasm -l elf64 calculadora.asm
gcc -o calculadora calculadora.o

Em seguida, basta executar o arquivo gerado nas etapas anteriores. O programa então aguarda a primeira entrada numérica seguida da tecla *enter*. A partir de então, um operador aritmético e o segundo operando numérico, nesta ordem, ambos acompanhados da tecla enter imediatamente após as respectivas entradas. O resultado é exibido na tela com 10 casas decimais na parte fr e, doravante, o programa aguarda até que o usuário entre com o operador e o segundo operando, respectivamente a cada nova iteração, que será interrompida se, e somente se, uma entrada vazia é detectada, quando o programa será finalizado.

4 RESTRIÇÕES E CAPACIDADES

A arquitetura adotada no projeto foi a Intel x86, em particular, a de 64 bits. As entradas suportadas são valores reais com até 20 dígitos, e são consideradas entradas inválidas valores como "+<x>" e "<x>e10". Serão aceitas entradas nos formatos ".<x>" e "<x>.", onde x representa o valor numérico. O programa, tal como uma calculadora comum, não exibe mensagens ao usuário exceto em caso de erro aritmético, divisão com denominador nulo, por exemplo. Cabe ao usuário realizar as entradas corretas, respeitando as restrições de execução.

APPENDIX A
COMANDOS

Aqui estão reunidos os comandos nas *tabelas 1, 2* e *3*, por categoria de comando.

TABLE 1
Comandos de transferência de controle.

Comando	Descrição
syscall	Chamada do sistema.
call <function>	Chamada à função/procedimento <function>.
ret	Encerra o procedimento e retorna seu resultado ao fluxo que o chamou.
int <código>	Chama uma interrupção com o <código> especificado.
cmp A, B	Compara se os operandos são iguais ou não. Não perturba os operandos de destino ou fonte. É usado junto com a instrução de salto condicional para a tomada de decisão.
jne <label>	Se não é igual vá para <label>.
jz <label>	Caso a flag indicadora seja 0 então o fluxo é desviado para o local indicado pelo rótulo <label>.
jnz <label>	Caso a flag seja não nula vá para a <label>.
jmp <label>	Vá para a label <label>.
jb <label>	Realiza comparação entre dois operandos sem sinal para (<) menor
ja <label>	Similar a JB, porém considerando a operação maior (>).

TABLE 2
Comandos de movimentação de dados.

Comando	Descrição
lea A, B	Carregar dados a partir do endereço efetivo em A.
movzx A, B	Copia o conteúdo do operando B para A e preenche o restante do espaço de A com zeros.
push X	Empilha X
pop Y	Desempilha o valor da pilha e salva em Y

TABLE 3
Comandos de conversão e reserva de memória.

Comando	Descrição
cvtsi2sd A, B	Converte um inteiro (tipo dword) para um escalar ponto flutuante com precisão dupla.
<label> resb X	Reserva X bytes na memória.
<label> resq Y	Reserva Y palavras quádruplas (qword) na memória, normalmente com 8 bytes.
<label> db X	Reserva um byte e o inicializa com X.
<label> dq X	Reserva uma quadword e o inicializa com X.

TABLE 4
Comandos lógico-aritméticos.

Comando	Descrição
test A, B	Realiza uma operação lógica AND bit a bit entre A e B sem alterá-los. Modifica apenas as flags.
cld	Controla as operações de cadeia incrementando os registros de índice.
xor X, Y	Realiza uma operação lógica OU EXCLUSIVO bit a bit entre X e Y e armazena o resultado em X.
inc X	O conteúdo de X é incrementado em uma unidade.
imul A, B	Multiplica A por B e armazena em A, sendo ambos inteiro.

APPENDIX B
FLUXO PRINCIPAL DO PROGRAMA

O fluxograma da *figura 3* descreve o funcionamento do módulo principal do programa, abstraindo-se os módulos secundários já apresentados.

AGRADECIMENTOS

Somos gratos a Deus por tudo.
Aos pais, pelo esforço e incentivo, cujos ensinamentos tornaram-se bases de nosso caráter.
Aos amigos e colegas pela parceria nos momentos difíceis.
Ao professor por transmitir seu conhecimento, orientando-nos a cada passo durante a trajetória desse trabalho.

REFERENCES

[1] X86-64 Reference Sheet (GNU assembler format). Disponível em: <https://www.systems.ethz.ch/sites/default/files/file/COURSES/2014%20FALL%20COURSES/2014_Fall_SPCA/lectures/x86_64_asm_cheat_sheet.pdf>. Acesso em: 01 dez. 2017.

[2] DOEPPNER, Tom. Introduction Computer Systems : x64 Cheat Sheet. Disponível em: <https://cs.brown.edu/courses/cs033/docs/guides/x64_cheatsheet.pdf>. Acesso em: 24 nov. 2017.

[3] NASM Manual Docs. Disponível em: <http://www.nasm.us/doc/>. Acesso em: 01 dez. 2017.
64-BIT NASM Notes. Disponível em: <http://dondi.lmu.build/share/sp/nasm64.pdf>. Acesso em: 08 dez. 2017. Baixar no formato Word
SAYFARTH, Ray. Introduction to 64 Bit Intel Assembly Language Programming . Hattiesburg, MS 39406 USA: [s.n.], 2011. 111-135 p.

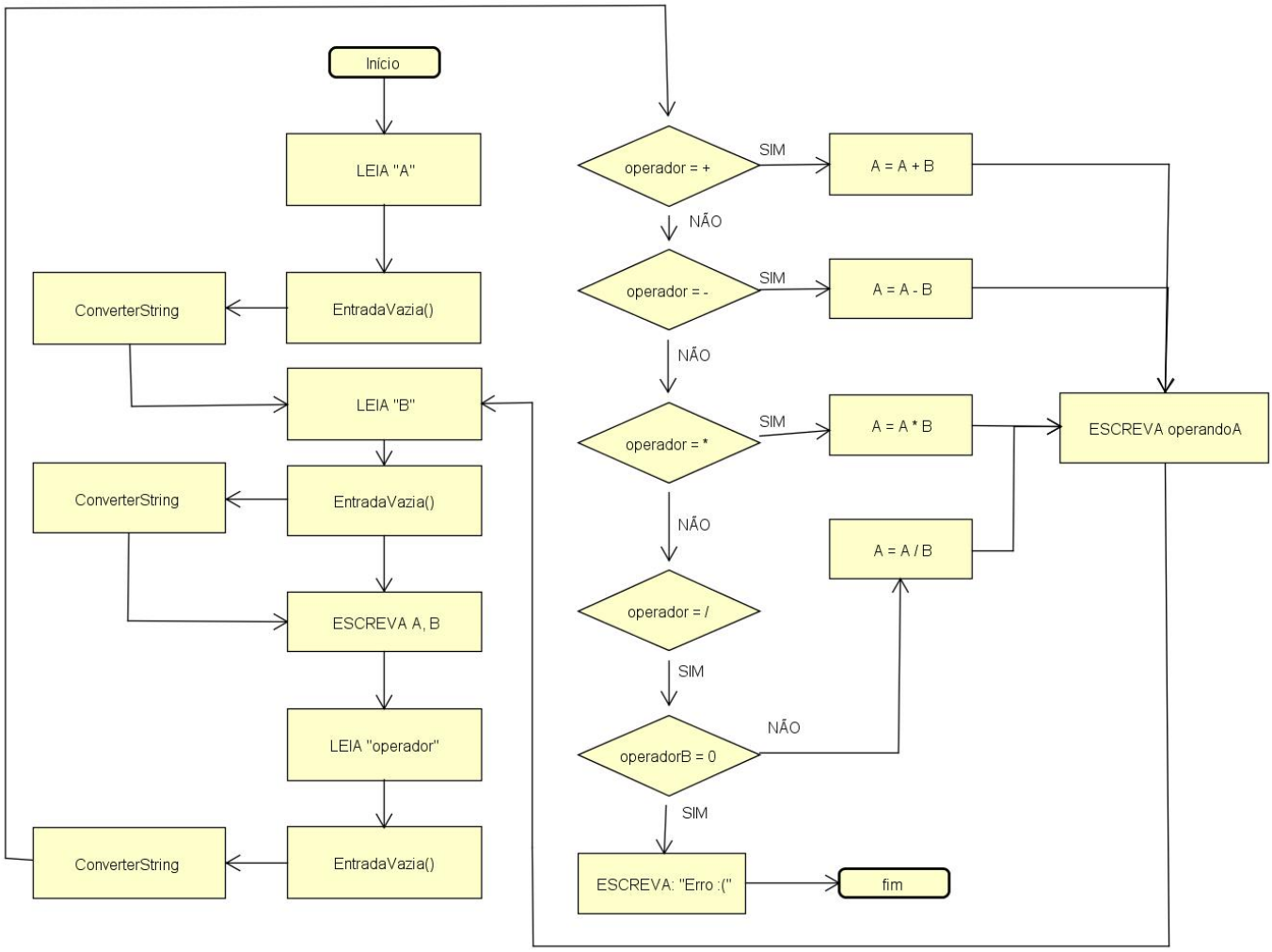

Fig. 3. Fluxograma do módulo principal da calculadora.

Elyabe Alves , 1996, Técnico em informática e graduando em Ciência da Computação pela Universidade Federal do Espírito Santo no campus do Centro Universitário Norte do Espírito Santo, Ceunes localizado em São Mateus, ES, Brasil.
e-mail: elyabe@outlook.com

Thayza Sacconi Thayza Sacconi Guarnier, nascida em 05 de novembro de 1996 na cidade de São Mateus - ES, é graduanda do curso de Ciência da Computação pela Universidade Federal do Espírito Santo - UFES campus São Mateus desde agosto de 2014 e possui formação como técnica no curso de Edificações desde 2015 pelo Instituto Federal do Espírito Santo - IFES campus Nova Venécia, ES, Brasil.
e-mail: thayzasaconi@gmail.com