Projeto Programação Assembly no 8086/8088 “Conversor de Bases Numéricas”

José Arthur Pereira Alves
Faculdade de Computação e Engenharia Elétrica - FACEEL
Universidade Federal do Sul e Sudeste do Pará - UNIFESSPA
Marabá, Brasil
arthurj167@unifesspa.edu.br

Luana Batista Araújo
Faculdade de Computação e Engenharia Elétrica - FACEEL
Universidade Federal do Sul e Sudeste do Pará - UNIFESSPA
Marabá, Brasil
lluanabatist@unifesspa.edu.br

Manoel Malon Costa de Moura
Faculdade de Computação e Engenharia Elétrica - FACEEL
Universidade Federal do Sul e Sudeste do Pará - UNIFESSPA
Marabá, Brasil
malloncosta@unifesspa.edu.br

***Resumo* — O Emu8086 é uma ferramenta utilizada para emular programas que foram desenvolvidos para processador de 16bits. Através de suas funções e instruções é possível testar e emular os mais variados softwares que rodam na arquitetura x86 de microprocessadores. Utilizando a linguagem Assembly desenvolveu-se um script capaz de realizar conversão de sistemas numéricos.**

Palavras-chaves— microprocessador, programas, Assembly, Emu8086.

I. INTRODUÇÃO

O 8086 foi o primeiro microprocessador de 16bits da Intel criado na década de 70, com o seu lançamento surgiu a arquitetura de x86 que eventualmente virou a linha de maior sucesso da Intel. O Emu8086 é um emulador que permite executar em máquinas com uma tecnologia mais atual programas desenvolvidos para processadores 8086 antigos, que foram usados em computadores Macintosh e Windows entre a década de 80 e década de 90. Através do editor Emu8086 é possível programar na linguagem Assembly, que é uma linguagem utilizada para programar códigos que são entendidos por dispositivos computacionais, como por exemplo: microprocessadores e microcontroladores. Em Assembly temos os símbolos chamados “mnemónicos”, que são substituídos pelos valores brutos de linguagem de máquina. A seguir, será abordado a utilização dessas 2 tecnologias para realizar as mais variadas conversões entre os seguintes sistemas numéricos: decimal, binário e hexadecimal.

II. METODOLOGIA

O objetivo do desenvolvimento deste projeto consistiu em aplicar os conhecimentos adquiridos em sala de aula e nos experimentos realizados utilizando a programação Assembly baseada na arquitetura do microprocessador 8086/8088. Logo, com o intuito de avaliar o aprendizado destes conhecimentos, foi solicitado aos alunos o desenvolvimento de uma aplicação prática e a partir disso, a elaboração de um trabalho escrito, bem como apresentação do projeto, assim, os alunos referentes a este trabalho optaram pela elaboração de um conversor de bases numéricas como primeiro projeto.

O projeto teve como elementos centrais de apreciação a utilização de operações de saltos, instruções de loops e funções de bibliotecas do Emu8086. Para melhor compreensão do programa desenvolvido, o trabalho foi dividido em cinco partes principais detalhadas nos itens A, B, C, D e E abaixo.

A. Menu

Com a finalidade de tornar o programa mais amigável para o usuário um menu para o projeto foi desenvolvido. Para isto, foram utilizados, as funções da biblioteca Emu86: CLEAR_SCREEN (procedimento para limpar a tela), PUTC (macro utilizada para mover o cursor) e PRINTN (macro utilizada para imprimir uma *string* e em seguida quebrar uma linha).

Após a exibição do menu é solicitado ao usuário que insira o número correspondente a operação de conversão que se deseja realizar, este número é lido pelo programa através de outra função da biblioteca Emu86, a função SCAN_NUM (procedimento que lê um numero de um ou mais dígitos e o armazena no registrador CX). A partir disso, o número armazenado em CX é comparada através da instrução CMP, com os dígitos 1, 2, 3 e 4, passando por uma série de validações de saltos condicionais que levam a execução da operação correspondente ou à impressão da mensagem de opção inválida. Na Figura 1 é possível observar o código para o menu desenvolvido.

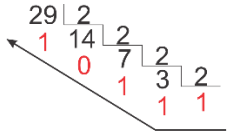
```
PRINTN '************************************************'
PRINTN '***** CONVERSOR BINARIO-DECIMAL-HEXADECIMAL *****'
PRINTN '************************************************'
PUTC 13d
PUTC 10d
PRINTN '(1) Conversao Binario->Decimal'
PRINTN '(2) Conversao Binario->Hexadecimal'
PRINTN '(3) Conversao Decimal->Binario'
PRINTN '(4) Conversao Decimal->Hexadecimal'
PUTC 13d
PUTC 10d
PUTC 13d
PUTC 10d
PRINT 'Selecione a opcao desejada: '
CALL SCAN_NUM
PUTC 13d
PUTC 10d
PUTC 13d
PUTC 10d


CMP CX, 1d
JE bindec
JL inv
CMP CX, 2d
JE binhex
CMP CX, 3d
JE decbin
CMP CX, 4d
JE dechex
JG inv
```

Figura 1 – Código do menu inicial

B. Conversão decimal para binário

Para realizar esta conversão é necessário utilizar alguns métodos matemáticos.



29 Decimal = 11101 Binário

Figura 2 - Exemplo de conversão Decimal-Binário.

Para tal, a divisão sucessiva é suficiente – divide o número por 2, guarda seu resto e divide novamente o quociente por 2, até que o resto seja menor ou igual a 1.

```
09 contador DW 0d
10
11 CALL SCAN_NUM
12 MOV AX, CX
13 MOV BL, 2
14
15
16 divisaoSuc:
17     INC contador
18     DIV BL
19     MOV DH, AH
20     PUSH DX
21     CMP AL, 1d
22     MOV AH, 0d
23     JGE divisaoSuc
24     JL cont
25 cont:
26     MOV CX, contador
27
28 PUTC 13d
29 PUTC 10d
30
31 impressao:
32     POP DX
33     MOV DL, 0d
34     MOV BL, DH
35     MOV BH, 0d
36     MOV AX, BX
37     CALL PRINT_NUM_UNS
38 LOOP impressao
39
40
41 final:
42 INT 20h
43
```

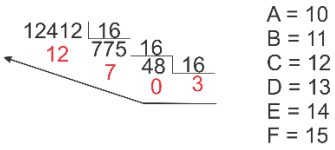
Figura 3 – Código de conversão Decimal-Binário

É declarado um contador inicializado com 0 para guardar a quantidade de iterações. Logo após, a função SCAN_NUM é chamada para ler um número do teclado em decimal. Por sua vez, o valor inserido armazenado em CX é movido para AX, a fim de se realizar operações. Também é movido para BL o valor 2, que é nosso divisor. Mais baixo tem uma condicional, onde o contador é incrementado, o valor de AL é dividido por BL, resultando no quociente armazenado no próprio AL e o resto, no AH. Agora, o resto é armazenado na pilha com o uso do registrador DH e a instrução PUSH.

Na linha 21, há uma comparação entre os valores de AL e 1 em decimal, e o valor de AH é zerado para receber o valor do próximo resto. Caso a comparação seja maior ou igual a 1, é repetido a condicional, caso contrário, é continuado o programa com a movimentação do valor do contador para CX. Portanto, os valores em binário já foram encontrados, basta agora imprimir cada caractere e para isso o laço de repetição LOOP é usado – retira-se da pilha o primeiro número da esquerda para a direita, coloca-o no registrador AX e imprime o número com a função PRINT_NUM_UNS e este processo se repete até que CX seja zero, pois o laço indiretamente já faz tal decrementado.

C. Conversão decimal para hexadecimal

Utiliza-se a mesma lógica do código anterior, porém com um divisor de 16, pois é a quantidade de números em hexadecimal.



12412 Decimal = 307C Hexadecimal

Figura 4 - Exemplo de conversão Decimal-Binário.

Muda também a impressão do número, pois, o resultado pode conter caracteres de A a F.

```
008 .DATA
009 contador DW 0d
010 msg DB 'O valor em hexadecimal e: ', 0
011 caractA DB 41h, 0
012 caractB DB 42h, 0
013 caractC DB 43h, 0
014 caractD DB 44h, 0
015 caractE DB 45h, 0
016 caractF DB 46h, 0
017
018
019 .CODE
020 CALL SCAN_NUM
021 MOV AX, CX
022 MOV BL, 16d
023
024 divisao:
025     INC contador
026     DIV BL
027     MOV DH, AH
028     PUSH DX
029     CMP AL, 15d
030     MOV AH, 0d
031     JGE divisao
032     JL cont
033
034 cont:
035     MOV CX, contador
036
037 PUTC 13d
038 PUTC 10d
039
040 LEA SI, msg
041 CALL PRINT_STRING
042
043 CALL PRINT_NUM_UNS
044 impressao:
045     POP DX
```

Figura 5 - Código da conversão Decimal-Hexadecimal, parte I.

Define-se uma variável de contagem, denominada **contador**, com a mesma função do código anterior e alguns caracteres também são definidos com valores de acordo com a tabela ASCII para representar os números de 10 a 15 em hexadecimal, com as letras de A a F. A leitura do número em decimal é feita e movido para AX, assim como o valor de BL é 16 em decimal. Na linha 24 tem uma label que determina o começo do laço de repetição, em que primeiro incrementa o **contador**, divide-se o número por 16 e o seu resto é armazenado na pilha com o uso do registrador DX. Posteriormente, na linha 29, compara o valor de AL com 15 em decimal (porque este número é o último da sequência em hexadecimal), caso seja maior ou igual, realiza-se novamente a divisão com o quociente resultante da divisão anterior. Contudo, se for menor que 15, pula-se a condicional e o valor do contador é movido para CX.

Pula uma linha para a apresentação do número e o curso é movido para o início. Diferentemente de decimal para binário, esta conversão necessita do valor do último quociente da divisão e por isso é chamada a função de impressão na linha 43, ou seja, apresenta o valor que está em AL.

```
LEA SI, msg
CALL PRINT_STRING

impressao:
    POP DX
    MOV DL, 0d
    num10:
        CMP DH, 0Ah
        JE apresentaA
        JMP num11

        apresentaA:
        LEA SI, caractA
        CALL PRINT_STRING
        JMP verifica

    num11:
        CMP DH, 0Bh
        JE apresentaB
        JMP num12

        apresentaB:
        LEA SI, caractB
        CALL PRINT_STRING
        JMP verifica

    num12:
        CMP DH, 0Ch
        JE apresentaC
        JMP num13

        apresentaC:
        LEA SI, caractC
        CALL PRINT_STRING
        JMP verifica

    num13:
        CMP DH, 0Dh
```

Figura 6 - Código da conversão Decimal-Hexadecimal, parte II.

O começo do LOOP é inicializado na label **impressão**, na qual é retirado o valor armazenado na pilha e movido para DX – para não se ter interferência, o valor de DL é zerado. Caso o valor de DX seja maior que 10, faz-se verificações com as labels **num10**, **num11** e demais, até o **num15**. Portanto, como o valor nos registradores são em hexadecimal, compara-se com suas representações acima de 10. E caso seja, a comparação seja verdadeira, imprime o número de acordo com seu caractere em hexadecimal e vai para a label **verifica**, na qual decrementa o contador e volta para a label do laço (impressão). Caso seja falso, a condicional anterior, aponta para a próxima condicional, até chega na última, onde faz a impressão de números abaixo de 10.

```
        JMP num15

        apresentaE:
        LEA SI, caractE
        CALL PRINT_STRING
        JMP verifica

    num15:
        CMP DH, 0Fh
        JE apresentaF
        JMP apresentaNum

        apresentaF:
        LEA SI, caractF
        CALL PRINT_STRING
        JMP verifica

    apresentaNum:
    MOV BL, DH
    MOV BH, 0d
    MOV AX, BX
    CALL PRINT_NUM_UNS

    verifica:
    DEC CX
    CMP CL, 0d
    JNE impressao


final:
INT 20h

DEFINE_SCAN_NUM
DEFINE_PRINT_NUM_UNS
DEFINE_PRINT_STRING
```

Figura 7 - Código da conversão Decimal-Hexadecimal, parte III.

D. Conversão Binário para Decimal

Utiliza divisão sucessiva também, mas desta vez para decompor o número, pois cada algoritmo é multiplicado por 2 elevado a posição do mesmo, da direita para a esquerda, começando em zero.

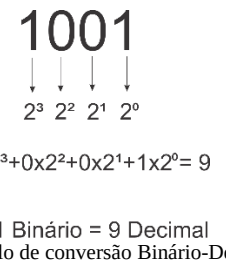


Figura 8 - Exemplo de conversão Binário-Decimal.

Após isso, soma-se os resultados das multiplicações e assim o resultado fica em decimal.

```
.CODE
CALL SCAN_NUM
MOV AX, CX
MOV BL, 10d
MOV CX, 0d
MOV DX, 0d

divisaoSuc:
    DIV BL
    MOV DL, AH
    PUSH DX
    MOV AH, 0d
    INC CX
    CMP AL, 0d
    JNE divisaoSuc
    JE cont

cont:
MOV DL, 0d
MOV BH, 0d
MOV BL, 0d
MOV DX, CX
MOV AL, 1d
MOV BL, 2d

CMP CX, 1d
JE cont2

DEC CX
mult:
MUL BL

LOOP mult
cont2:

MOV CX, DX
MOV BL, AL
MOV AL, 0d
```

Figura 9 - Código da conversão Binário-Decimal, parte I.

O primeiro comando é para receber o número, que é passado de CX para AX. Move o valor de 10 para BL e os valores de CX e DX são zerados porque serão utilizados depois. Na divisão sucessiva, o valor em AX é dividido por BL, e armazenado o resto da divisão na pilha com o uso do registrador DX. Por fim, CX é incrementado, pois é o contador e o valor de AL (quociente) é comparado com zero, se for diferente, a condicional é feita novamente. Contudo, se for igual a 0, o programa é continuado depois da condicional, na qual, os valores dos registradores são zerados e o valor de CX é guardado. Caso o valor de CX é 1, o laço é saltado, pois este laço é responsável por encontrar o maior valor da potência de 2, porque o valor na pilha é começado pelo o número da esquerda.

```
38 mult:
39 MUL BL
40
41 LOOP mult
42 cont2:
43
44 MOV CX, DX
45 MOV BL, AL
46 MOV AL, 0d
47
48 MOV AL, BL
49 MOV BL, 2d
50
51 retiraPilha:
52     MOV SI, AX
53     POP DX
54     MUL DL
55     ADD BH,AL
56     MOV AX, SI
57     MOV AH, 0d
58     DIV BL
59
60 LOOP retiraPilha
61
62 fim:
63 INT 21h
64
65
66 DEFINE_SCAN_NUM
67
```

Figura 10 - Código da conversão Binário-Decimal, parte II.

Depois, alguns valores são retornados e zerados, pois são utilizados no **LOOP retiraPilha**, onde o valor de AX é movido para SI, é retirado o valor da pilha; há a multiplicação e depois o valor é adicionado em BH. Assim, o valor de AX é recuperado na linha 56 e depois dividido por BL (2 em decimal) e continua até o contador CX for 0. Portanto, o valor da soma é armazenado em BH.

E. Conversão Binário Hexadecimal

Para uma melhor utilização dos procedimentos utilizados, esta conversão em especial, utilizado os códigos passados. Primeiro é chamada a Conversão Binário-Decimal e depois o valor resultante armazenado em BH é passado para o código de Conversão Decimal-Hexadecimal, apresentando um valor de Binário-Hexadecimal indiretamente.

III. Resultados Obtidos

As figuras, a seguir, exibem os resultados da execução do programa. Na Figura 11, é possível verificar a impressão do menu inicial e a solicitação do número correspondente a operação que se deseja realizar.

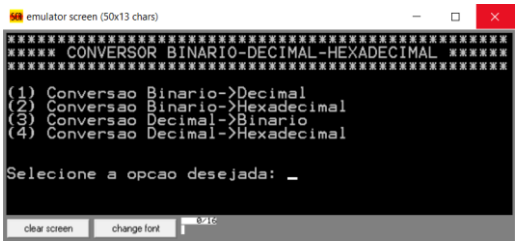


Figura 11 – Exibição do menu inicial

A Figura 12, corresponde ao resultado da operação executada após a seleção da opção 1 do menu (conversão binário → decimal) e inserção de um número binário de até 4 dígitos, neste caso, o número 1111, para ser convertido.

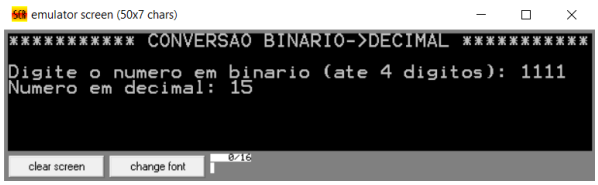


Figura 12 – Resultado da operação Binário-Decimal.

A Figura 13, corresponde ao resultado da operação executada após a seleção da opção 2 do menu (conversão binário → hexadecimal) e inserção de um número binário de até 4 dígitos, neste caso, o número 1010, para ser convertido.

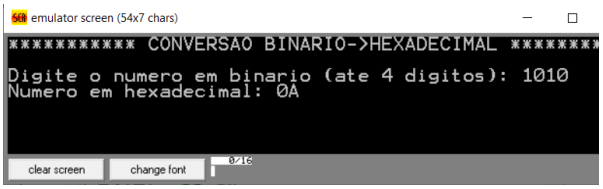


Figura 13 – Resultado da operação Binário-Hexadecimal.

A Figura 14, corresponde ao resultado da operação executada após a seleção da opção 3 do menu (conversão decimal → binário) e inserção de um número decimal, neste caso, o número 38, para ser convertido.

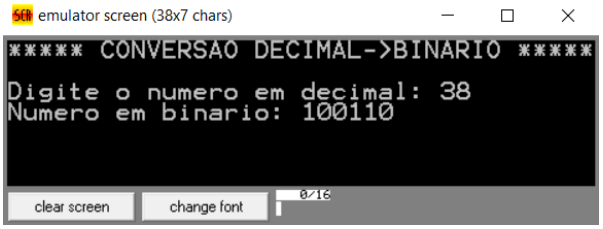


Figura 14 – Resultado da operação Decimal -Binário.

A Figura 15, corresponde ao resultado da operação executada após a seleção da opção 4 do menu (conversão decimal → hexadecimal) e inserção de um número decimal, neste caso, o número 192, para ser convertido. O valor permitido é de até 256.

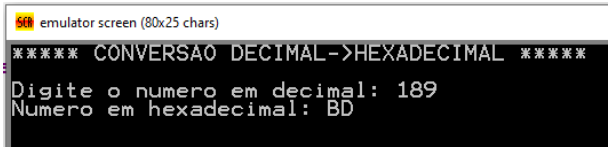


Figura 15 – Resultado da operação Decimal-Hexadecimal.

Por fim, a Figura 16, corresponde ao resultado da inserção de um valor inválido no menu inicial.

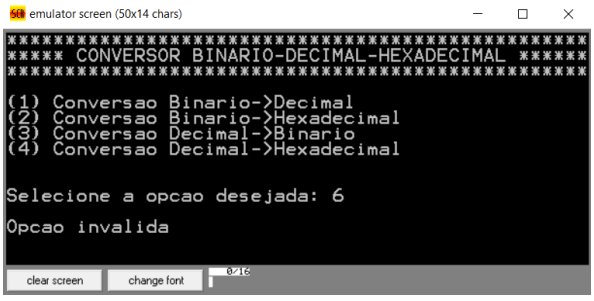


Figura 16 – Resultado Opção Inválida

IV. Conclusão

Conclui-se que o Emu8086 se mostrou uma ferramenta bastante útil e eficiente como um emulador de programas de 16bits, principalmente para fins didáticos, pois através dele pode-se colocar em prática conhecimentos teóricos. Tal

afirmação pode ser comprovada através dos objetivos deste trabalho que foram alcançados com êxito, efetuando assim, o script de conversor de bases numéricas.

REFERÊNCIAS

[1] Documentation for Emu8086 - Assembler and Microprocessor Emulator presente no próprio software.

[2] MANZANO, José. Programação Assembly: Padrão IBM-PC 8086/8088. 7ª Edição. São Paulo: Érica, 2013.

[3] CARVALHO, Murilo; CARVALHO, Carmem. Os Microprocessadores 8086/88: Hardware. Universidade Federal Fluminense. 2020.

[4] PANNAIN, Ricardo. Capítulo 3: Introdução à linguagem montadora do 8086. Universidade Estadual de Campinas. 2001.