Linguagem Assembly

Prof. Daniel Caetano

Objetivo: Apresentar uma breve visão sobre uma linguagem assembly e como programas são construídos nesta linguagem.

Bibliografia:

- ROSSINI, F; LUZ, H.F. **Linguagem de Máquina**: Assembly Z80 - MSX. 1ed. São Paulo: Ed. Aleph, 1987.
- CARVALHO, J.M. **Assembler para o MSX**. 1ed. São Paulo: McGraw Hill, 1987.
- ASCII Corporation. **O Livro Vermelho do MSX**. 1ed. São Paulo: Ed. McGraw Hill, 1988.

INTRODUÇÃO

Nas aulas anteriores foram apresentados os elementos internos dos computadores, tendo sido o funcionamento interno dos principais deles descritos previamente. Todos estes conceitos são importantes para assimilar as limitações e potencialidades de cada equipamento que um programador tome contato; e é importante ressaltar que, diferentemente do que se pode pensar a princípio, hoje é dia é bastante comum a necessidade de programação para diferentes plataformas como videogames, celulares, PDAs, mídia players, computadores etc.

Como mais uma ferramenta para conhecer as limitações e potencialidades de uma máquina e também como uma forma de compreender melhor os conceitos previamente apresentados, nesta aula veremos uma breve introdução à linguagem de máquina.

1. POR QUE USAR ASSEMBLY Z80?

O primeiro aspecto importante é que esta aula trata do assembly de um processador bastante antigo, do fim da década de 70. Mas por que estudar isso? Por algumas razões, dentre elas:

- O Assembly do Z80 é, reconhecidamente, um dos mais amigáveis.
- O Z80 é o processador de maior sucesso da história, embora poucos falem nele hoje.
- O Z80 é usado ainda hoje, com a função de microcontrolador.
- O Z80 tem um número interessante de registradores de propósito geral.

Estas características possibilitam que até mesmo alguém que nunca programou aprenda a linguagem Assembly Z80, embora possa ser uma tarefa mais árdua para aqueles que jamais programaram, mas isso não tem a ver com a linguagem em si: assembly é, em essência, a linguagem mais fácil (e simples) que existe. Ao mesmo tempo, as inerentes limitações da linguagem assembly permitem, como nenhuma outra linguagem, o desenvolvimento da habilidade dos programadores em "dar soluções geniais" para problemas gerais.

2. PRIMEIROS PASSOS NO ASSEMBLY Z80

Primeiramente, vamos esquecer que o Assembly é uma linguagem de programação: pensemos que é uma seqüência de ordens que devem ser transmitidas às peças do computador. Pensando desta forma, tudo começa a fazer muito mais sentido, pois é isso que de fato ocorre quando programamos Assembly: damos ordens aos microchips.

Assembly também não é a linguagem de máquina (zeros e uns): Assembly é uma linguagem compreensível pelos seres humanos, ainda que ela represente exatamente as instruções da máquina. Por esta razão, é necessário o uso de um programa "Assembler", que converta o texto que escrevemos em código de máquina (em linguagem de máquina) para que o computador execute o programa. Isso será visto passo a passo.

2.1. Primeiro Software em Assembly Z80

Antes de mais nada, é preciso definir o objetivo. O objetivo desta introdução será usar o Assembly Z80 para escrever o famoso programa "Alo, Mundo". Então, claramente, a idéia é fazer um programa que escreva:

"Alo, Mundo"

Na tela do computador... mas isso é muito vago. Z80 é só um processador e ele precisa estar inserido em um equipamento inteiro para que essa tarefa possa ser realizada. Neste caso, será então selecionado um computador genérico que use Z80 (como um MSX), executando um sistema operacional compatível com CP/M (como o MSX-DOS). É claro que não usaremos a máquina real; usaremos um emulador, no caso o Z80Mu. Adicionalmente, precisaremos de um "assembler", que é um programa que converterá o que escrevemos em Assembly para a Linguagem de Máquina. Neste curso usaremos o M80/L80 da Microsoft.

Voltando ao problema, que já é bem mais definido: A idéia será escrever "Alo, Mundo" usando o CP/M, em um (emulador de) computador Z80. Isso é importante, porque, como já vimos, a linguagem Assembly muda de máquina para máquina.

Mas vamos por partes... se queremos escrever algo na tela do computador, lembrando que o computador é uma máquina de Von Neuman, a primeira coisa a se fazer é colocar a informação a ser impressa na **memória**. A memória dos computadors com Z80 tem 64KB, ou seja, 65536 *posições de memória*. Isso significa que você tem essas 65536 posições para colocar todos os dados e instruções do seu programa. Pode parecer pouco (de fato, a maioria dos documentos de Word tem pelo menos o dobro disso de tamanho), mas como vocês verão, em Assembly é muito difícil gastar tudo isso de memória.

Antes de vermos como solicitar que o computador imprima a frase na tela, vejamos como colocar **o que** queremos que ele escreva na memória. Assim, o primeiro passo será abrir o "NotePad" e digitar, na primeira linha, o seguinte:

```
Alo, Mundo
```

Ótimo, só que o computador não vai entender isso e irá simplesmente travar se tiver de executar esse "código". Para entender o porque disso, imagine um japonês tentando falar em russo com você, e você não sabe nem japonês nem russo. É mais ou menos assim que o computador vai encarar se isso for colocado na memória dele, deste jeito. Precisamos então indicar para o computador o que deve ser feito com esses dados, em uma linguagem que ele entenda.

Para compreender esta necessidade, não é preciso se colocar na posição do computador: isso que foi escrito no NotePad não tem significado sequer para uma pessoa: se alguém recebesse um papel com isso escrito, a pessoa simplesmente não ia entender o que significa ou para que serve. Com o computador, não seria diferente. Para isso fazer algum sentido para uma pessoa, o texto deveria ser algo do tipo:

```
"Pegue a frase abaixo:
Alo, Mundo
e cole na tela do monitor"
```

É claro que uma pessoa seria capaz de seguir estas ordens (embora ainda possa achá-las esquisitas). É importante lembrar que o computador sempre agirá exatamente desta forma: ele fará o que o programador mandar, desde que o programador mande direito. (pode-se dizer que ele faz exatamente o que o programador manda, o que não é necessariamente o que o programador quer...)

De qualquer forma, existe uma tarefa aí nesta seqüência de instruções que é complexa: "cole na tela do monitor"... como se faz isso? Com CTRL+V, com cola branca ou com post-it? Bem, dissemos anteriormente que iríamos trabalhar com o CP/M e, por sorte, o CP/M sabe exatamente como "colar um texto na tela do monitor". A instrução do CP/M que faz isso é a função número **9**. Embora pareça estranho no início, com o tempo o programador assembly se acostuma com "nomes" numéricos para tudo.

A função número 9 faz exatamente isso: "Pegue *a frase X* e a cole na tela do monitor". Ocorre que "a frase X" é algo que estará na memória do computador, como vimos anteriormente. E se "a frase X" está na memória do computador, ela deve ter um **endereço de memória**, que no caso das strings é (quase) sempre o endereço do primeiro caractere da string. Assim, se dizemos que a frase "Alo, Mundo" está no endereço 10000h, o que teremos na memória do computador é:

Posição de Mem.	Conteúdo
10000	A
10001	l
10002	o
10003	,
10004	
10005	M
10006	u
10007	n
10008	d
10009	o

A função 9 do CP/M exige que este valor inicial da frase esteja indicada por um registrador de uso geral chamado **DE**. Assim, se soubéssemos que a frase estava realmente no endereço 10000h, bastaria indicar este valor no registrador DE e em seguida chamar a função 9 do CP/M que a frase seria impressa na tela. Então, a primeira coisa que precisa ser feita após o texto "Alo, Munto", é indicar no registrador DE a posição de memória em que a frase estará... mas aí vem o primeiro problema: qual é esta posição?

Poderíamos definir um valor fixo; entretanto, não iremos fazê-lo. No início desta aula foi citado que usaríamos um programa "Assembler" para converter o código Assembly para Linguagem de Máquina. Existe uma maneira de deixar que o programa Assembler se preocupe com os endereços de memória por nós: usando apelidos.

Quando um apelido é definido para um dado, não importa onde o Assembler colocará aquele dado: podemos nos referir àquela posição de memória através do apelido. A indicação do apelido se faz usando a estrutura:

APELIDO: Dados

Por se tratar de uma string, é necessário indicar ao Assembler onde a string começa e onde ela termina, fazendo isso com as aspas simples: ' e ':

APELIDO: 'Uma String'

Por outro lado, também é necessário informar ao Assembler qual é o tipo de dado que está recebendo o apelido, que no caso é uma seqüência de bytes, que é indicado desta forma:

APELIDO: DB 'Uma String'

Onde DB significa "Data Bytes" ou "Bytes de Dados".

No nosso exemplo, vamos usar o apelido "FRASE" para a frase a ser impressa, e a indicação no programa fica assim:

```
FRASE:   DB   'Alo, Mundo'
```

Isso nos fornece o apelido "FRASE" para trabalhar. Se precisamos indicar o endereço da frase no registrador DE, basta carregar (ou ler) este valor no registrador DE. Isso é feito com a instrução LD (de LoaD) do Z80, cuja sintaxe é:

LD DE, *dado*

Onde *dado* é um número qualquer. No caso, deve ser indicada a posição de memória (ou seu apelido) da frase, para que ela possa ser impressa. Esta instrução precisa ser acrescentada antes de solicitarmos ao CP/M que imprima a frase. Isso pode ser feito como apresentado abaixo:

```
FRASE:   DB   'Alo, Mundo'
         LD   DE, FRASE
```

Assim, a função 9 já saberá onde encontrar a frase a ser impressa. A necessidade, agora, é informar ao CP/M que ele deve executar a função 9. Para isso, o CP/M solicita que o número da função a ser executada seja indicada no registrador de uso geral **C**. O formato é o mesmo já visto anteriormente:

LD C, *dado*

Desta forma, para indicar a função 9 para o CP/M, o seguinte deve ser escrito:

```
FRASE:   DB   'Alo, Mundo'
         LD   DE, FRASE
         LD   C, 9
```

Agora todo o cenário está preparado para que o CP/M execute a função 9 e o texto apareça escrito na tela, mas precisamos "solicitar" que ele execute a função selecionada no registrador C e isso pode ser feito com uma função que está na posição 5 da memória (ela foi carregada lá quando o CP/M foi iniciado). Como queremos que a execução do programa se desloque para lá temporariamente, faça o que tem de fazer e depois volte, usamos a instrução CALL do Assembly:

CALL [endereço]

Com essa chamada, o programa irá até a posição de memória indicada, executará o código que lá encontrar... até encontrar a instrução RET (RETurn), quando então volta para continuar a execução logo após o CALL.

O código, já implementando a instrução CALL para o endereço 5 está a seguir:

```
FRASE:   DB   'Alo, Mundo'
         LD   DE, FRASE
         LD   C, 9
         CALL 5
```

Ok, parece que isso vai funcionar: primeiro definimos os dados, depois indicamos os parâmetros nos registradores e finalmente chamamos a execução da função de impressão que o CP/M fornece... mas este programa está muito pouco legível. É possível melhorá-lo.

O Assembler que iremos usar, assim como a maioria dos Assemblers, permite que o programador dê *apelidos* a valores numéricos, usando uma *pseudo-instrução* chamada EQU (de EQUivalente). A forma desta instrução é:

APELIDO EQU VALOR

Assim, se quisermos dar um apelido interessante para a função 9 do CP/M, como por exemplo STROUT (de STRing OUT - "saída de texto", em português) podemos fazer da seguinte forma:

STROUT EQU 9

É possível fazer o mesmo com o endereço 5 de memória, dando um apelido BCPM (Bios do CP/M) a ele, como pode ser visto no código abaixo:

```
STROUT   EQU  9
BCPM     EQU  5

FRASE:   DB   'Alo, Mundo'

         LD   DE, FRASE
         LD   C, STROUT
         CALL BCPM
```

É possível tornar o código ainda mais legível com alguns comentários, que são indicados pelo caractere ";":

```
STROUT   EQU  9
BCPM     EQU  5

FRASE:   DB   'Alo, Mundo'

         LD   DE, FRASE      ; Indica endereço do texto
         LD   C, STROUT      ; Indica função do CPM
         CALL BCPM           ; Solicita execução pelo CPM
```

Este código já está com uma cara de programa, mas ainda não vai funcionar... e por quê? Porque começamos o nosso programa com uma seqüência de dados que nada têm a ver com um programa... e o Z80 (como qualquer outro processador) vai pensar que estes dados são, na verdade, instruções.

Isso nos remente ao modelo de Von Neuman: para o processador, não existe diferença física entre dados e instruções: ambos são números na memória. O que diferencia entre um e outro é a permissão que o programador dá para que a CPU execute um trecho da memória ou não. Por exemplo: se comandarmos no software a instrução:

```
CALL FRASE
```

Teremos dito para o Z80 "execute o que está na posição de memória indicada pelo apelido FRASE". Mas isso será um desastre, porque "FRASE" não aponta para um programa! "FRASE" aponta para um texto! Este tipo de coisa (CPU processando dados como instruções) normalmente faz com que o computador trave, simplesmente.

Mas a correção disso é simples: como não faz a menor diferença onde a definição da frase foi colocada (uma vez que o apelido será corrigido em qualquer posição que você a coloque), é possível colocá-la no fim do programa, e aí o problema acaba... ou quase:

```
STROUT   EQU  9
BCPM     EQU  5

         LD   DE, FRASE      ; Indica endereço do texto
         LD   C, STROUT      ; Indica função do CPM
         CALL BCPM           ; Solicita execução pelo CPM

FRASE:   DB   'Alo, Mundo'
```

Mas... por que quase? Observe a execução do programa: o que vai acontecer quando a execução voltar do "CALL BCPM"? Isso mesmo! O Z80 vai voltar a executar a frase 'Alo, Mundo', só que desta vez após a impressão da frase na tela. Para evitar isso, é preciso indicar para o Z80 onde o programa acaba... na verdade, iremos chamar um outro comando do CP/M que indica para que o programa seja finalizado e voltemos ao prompt. Isso pode ser feito com a instrução JP (de JumP, salto) que é o "pulo para nunca mais voltar", indicando para que o programa vá para o endereço 0:

```
JP     0
```

Inserindo esta instrução ao fim do programa, tudo estará pronto...:

```
STROUT   EQU  9
BCPM     EQU  5

         LD   DE, FRASE      ; Indica endereço do texto
         LD   C, STROUT      ; Indica função do CP/M
         CALL BCPM           ; Solicita execução pelo CP/M
         JP   0              ; Volta ao prompt do CP/M

FRASE:   DB   'Alo, Mundo'
```

Salve este arquivo na sua pasta de aluno, dentro de um diretório chamado \ASM, com o nome HELLO.MAC . Agora falta usar o Assembler para executar nosso programa.

2.2. Assemblando o Software HELLO.MAC

No computador já deve estar instalado o software M80/L80 da Microsoft. Para executá-lo, abra o prompt, mude para o diretório onde criou o arquivo HELLO.MAC e comande:

CL80 HELLO

A saída deverá ser algo do tipo:

```
M80/L80 Z80 Compiler - IBM PC
Ported by A&L Software

MSX.M-80  1.00  01-Apr-85  (c) 1981,1985 Microsoft
%No END statement
%No END statement

No Fatal error(s)

MSX.L-80  1.00  01-Apr-85  (c) 1981,1985 Microsoft

Data    0103    0118    <   21>

49189 Bytes Free
[ 0000   0118        1]
```

Isso terá gerado o arquivo HELLO.COM, que já é o executável pronto!

2.3. Executando o Software HELLO.COM

Agora que já temos o executável pronto, já no diretório em que reside o arquivo HELLO.COM, digite:

Z80MU

Isso fará com que entremos no prompt do CP/M. Neste prompt (roxo) basta digitar:

HELLO

E o programa criado será executado... só que o resultado não é exatamente o esperado! Após aparecer "Alo, Mundo" muito rapidamente, um monte de lixo é impresso e só um vidente pode dizer o que acontece em seguida.

Isso não devia estar acontecendo, afinal, tomamos todas as precauções para que o programa finalizasse corretamente, certo? Correto, mas faltou uma informação... e a falta dela está causando todo este transtorno.

A culpa não é sua, aluno. Faltou dizer que a função STROUT imprime uma string terminada pelo símbolo "$". Sem isso, a função STROUT vai imprimindo tudo que encontrar na memória, até achar um caractere desse (acredite, ela **não para** até achar!). Assim, saia do Z80Mu com o comando "QUIT" e experimente alterar o seu programa da seguinte forma:

```
STROUT   EQU  9
BCPM     EQU  5

         LD   DE, FRASE      ; Indica endereço do texto
         LD   C, STROUT      ; Indica função do CP/M
         CALL BCPM           ; Solicita execução pelo CP/M
         JP   0              ; Volta ao prompt do CP/M

FRASE:   DB   'Alo, Mundo$'
```

"Reassemble" o programa como indicado na seção 2.2 e re-execute o mesmo pelo emulador, como indicado na seção 2.3. Finalmente, o programa irá funcionar corretamente, imprimindo:

Alo, Mundo

E finalizando, voltando ao prompt do CP/M. Voilà! Seu programa funcionou perfeitamente! Esse é o seu primeiro programa em Assembly. Antes de finalizar, porém, é interessante comentar o seguinte. Você deve ter observado que, quando usou o CL80, os seguintes comentários apareceram:

```
MSX.M-80  1.00  01-Apr-85  (c) 1981,1985 Microsoft
%No END statement
%No END statement
```

Isso ocorre sempre que não indicamos ao M80 onde o programa inicia e onde ele termina. Para corrigir isso, basta acrescentar as *pseudo-instruções* START e END no programa, da seguinte forma:

```
STROUT   EQU  9
BCPM     EQU  5

START:
         LD   DE, FRASE       ; Indica endereço do texto
         LD   C, STROUT       ; Indica função do CP/M
         CALL BCPM            ; Solicita execução pelo CP/M
         JP   0               ; Volta ao prompt do CP/M

FRASE:   DB   'Alo, Mundo$'

         END
```

Se assemblá-lo novamente, verá que o M80 não reclama de mais nada em seu programa.

3. AVANÇANDO UM POUCO MAIS

Na seção anterior foi apresentada a idéia de se escrever um programa, ou seja, uma seqüência de ordens para o computador, que escrevesse na tela a frase "Alo, Mundo". Bem, para fazer isso, foi usado o NotePad (Bloco de Notas) como editor e o Microsoft Macro Assembler (M80/L80).

Foram apresentados alguns conceitos sobre o ambiente de programação: há apenas 64K posições de memória, a linguagem assembly sendo usada é a do processador (Z80) e que para carregar dados em seus registradores usamos a instrução LoaD (LD). Foi visto que para o Z80 para executar uma função do sistema operacional CP/M (STROUT) é necessário indicar o numero da função no registrador C (LD C, STROUT) e mandar ele executar essa função no endereço BDOS (endereço 0005h), pedindo para que ele volte e continue com o programa após a execução da tarefa - usando para isso a instrução CALL (CALL BDOS). Finalmente foi apresentado como voltar ao prompt do CP/M, usando o comando JumP para o endereço 0 da memória (JP 0).

Adicionalmente, foram apresentados alguns conceitos sobre o *assembler* em uso (o M80/L80): como colocar dados na memória (APELIDO: DB 'dado') e como dar apelidos a endereços de memória (APELIDO EQU endereço). Foi apresentada a idéia de que dados e programas convivem na memória e que, se for comandado ao Z80 que ele execute um conjunto de dados (JP FRASE ou CALL FRASE), ele realmente vai tentar (e provavelmente o resultado não será bom); assim, é necessário evitar que o Z80 chegue a "executar dados".

Por fim, foi visto como "assemblar" o programa com o M80/L80, ou seja, como tornar o código assembly escrito algo legível para o computador, além de apresentar o uso do Z80MU.

Nas próximas seções os conceitos vistos anteriormente serão usados e extendidos, usando várias novas "funções" do CP/M, indicando como ler uma tecla e apresentar o valor lido.

4. TORNANDO O PROGRAMA MAIS COMPLETO

Inicialmente, é interessante relembrar o programa no estágio como ficou na seção anterior:

```
STROUT     EQU   9
BCPM EQU   5

START:
           LD    DE, FRASE          ; Indica endereço do texto
           LD    C, STROUT          ; Indica função do CP/M
           CALL  BCPM               ; Solicita execução pelo CP/M
           JP    0                  ; Volta ao prompt do CP/M

FRASE:     DB    'Alo, Mundo$'

           END
```

Antes de mais nada, é interessante criar um arquivo com esse conteúdo (PROG2.MAC, no diretório \ASM) para que as mudanças possam ser executadas. Em seguida, modifiquemos o apelido "FRASE" para um mais adequado, como **NOMEDOPRG** (Nome do Programa), alterando também a frase para **Programa 2 - Conversando com o usuário**.

```
STROUT     EQU   9
BCPM EQU   5

START:
           LD    DE, NOMEDOPROG     ; Indica endereço do nome
           LD    C, STROUT          ; Indica função do CP/M
           CALL  BCPM               ; Solicita execução pelo CP/M
           JP    0                  ; Volta ao prompt do CP/M

NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario$'

           END
```

Assim, o "nome do programa" sempre será apresentado quando o programa for iniciado. Se o programa for assemblado (CL80 PROG2) e depois executado dentro do Z80MU, ele deve imprimir a frase. Quando se programa em assembly é bastante útil "assemblar" e testar muito bem cada pequena mudança, para evitar dificuldades em resolver eventuais bugs.

Uma segunda alteração será acrescentar o nome do autor ao programa, o que pode ser feito da mesma maneira com que foi impresso o nome do programa, definindo-se um novo texto com um novo apelido:

```
STROUT     EQU   9
BCPM EQU   5

START:
           LD    DE, NOMEDOPROG    ; Indica endereço do nome
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M
           JP    0                 ; Volta ao prompt do CP/M

NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario$'
AUTOR:     DB    ' Por Seu_Nome_Aqui$'

           END
```

Mas ainda falta a impressão do autor na tela; é preciso escrever o código para que o Z80 faça isso, sendo ele exatamente o mesmo que foi usado para imprimir o nome do programa:

```
STROUT     EQU   9
BCPM EQU   5

START:
           LD    DE, NOMEDOPROG    ; Indica endereço do nome
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           LD    DE, AUTOR         ; Indica endereço do autor
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           JP    0                 ; Volta ao prompt do CP/M

NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario$'
AUTOR:     DB    ' Por Seu_Nome_Aqui$'

           END
```

Bem, o código começou a ficar meio sujo; uma forma de "sujar para limpar" é acrescentar alguns comentários de blocos.

Comentários de blocos são comentários que ficam alinhados com as instruções e explicam em linhas gerais o que as próximas linhas de código fazem. Programadores experientes, em situações normais, fazem apenas comentários de blocos. Alguns comentários deste tipo são apresentados na listagem a seguir.

```
STROUT     EQU   9
BCPM EQU   5

START:
           ; Mostra nome do programa
           LD    DE, NOMEDOPROG    ; Indica endereço do nome
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           ; Mostra nome do autor
           LD    DE, AUTOR         ; Indica endereço do autor
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           JP    0                 ; Volta ao prompt do CP/M

; Dados do Programa
NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario$'
AUTOR:     DB    ' Por Seu_Nome_Aqui$'

           END
```

Ao assemblar este programa, algo parece ter saído errado: o nome do autor e o nome do programa estão saindo "grudados". Bem, para que uma linha seja pulada, é preciso inserir um código no texto impresso, para solicitar essa "quebra" de linha.

Na verdade, é preciso adicionar **dois** códigos: um que pula uma linha (Line Feed, ou LF) e outro que volta o cursor para a posiçao 0 (Carriage Return, ou CR). O Line Feed é representado pelo número 10 e o Carriage Return pelo código 13. Desta forma, basta adicionar estes números nos dados para que uma linha seja pulada, como por exemplo:

```
NOMEDOPRG: DB  'Programa 2 - Conversando com o usuario$',13,10
```

MAS, se isso for feito, nada mudará... porque o caractere que indica o "fim do texto" ($) está *antes* dos códigos 13 e 10 (e, portanto, estes códigos não estão sendo interpretados na impressão do texto). Assim, estes códigos precisam estar entre o fim do texto e o caractere $, como pode ser visto na listagem a seguir:

```
STROUT     EQU   9
BCPM EQU   5

START:
           ; Mostra nome do programa
           LD    DE, NOMEDOPROG    ; Indica endereço do nome
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           ; Mostra nome do autor
           LD    DE, AUTOR         ; Indica endereço do autor
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           JP    0                 ; Volta ao prompt do CP/M
```

```
; Dados do Programa
NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario',10,13,'$'
AUTOR:      DB     ' Por Seu_Nome_Aqui$'

            END
```

O próximo passo será solicitar ao usuário que pressione uma tecla, para que seja possível executar alguma atividade com base nesta resposta. Entretanto, para solicitar que o usuário pressione uma tecla, é preciso ainda imprimir mais uma informação na tela, o que pode ser feito imprimindo o texto PERGU1, como no código a seguir, lembrando de adicionar os códigos para pular linha na frase anterior.

```
STROUT      EQU   9
BCPM EQU    5

START:
            ; Mostra nome do programa
            LD    DE, NOMEDOPROG     ; Indica endereço do nome
            LD    C, STROUT          ; Indica função do CP/M
            CALL  BCPM               ; Solicita execução pelo CP/M

            ; Mostra nome do autor
            LD    DE, AUTOR          ; Indica endereço do autor
            LD    C, STROUT          ; Indica função do CP/M
            CALL  BCPM               ; Solicita execução pelo CP/M

            ; Mostra pergunta
            LD    DE, PERGU1         ; Indica texto da pergunta
            LD    C, STROUT          ; Indica função do CP/M
            CALL  BCPM               ; Solicita execução pelo CP/M

            JP    0                  ; Volta ao prompt do CP/M

; Dados do Programa
NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario',10,13,'$'
AUTOR:      DB     ' Por Seu_Nome_Aqui',10,10,13,'$'
PERGU1:     DB     ' Pressione alguma tecla: $'

            END
```

Observe que foram adicionados dois códigos 10 na frase da etiqueta AUTOR. Isso faz com que sejam puladas duas linhas após o nome do autor. O resultado da execução deste programa deve ser algo assim:

```
A>PROG2
Programa 2 - Conversando com o usuário
  Por Daniel Caetano

  Pressione alguma tecla:
A>
```

Assim, o programa mostra tudo que é necessário, mas ainda falta receber a tecla do usuário. Bem, no CP/M isso é feito com a função 1 (apelidada, comumente, de CONIN, de

CONsole IN). Para chamá-la, basta indicar seu número no registrador C e chamar o endereço 5 (CP/M) da mesma forma com que era feito para imprimir um texto:

```
LD    C,1   ; Indica a função que pega uma tecla
CALL  5     ; Chama CP/M
```

Ou, usando os apelidos,

```
CONIN EQU   1
BCPM  EQU   5
LD    C,CONIN     ; Indica a função que pega uma tecla
CALL  BCPM        ; Chama CP/M
```

Adicionando estes ao programa, o resultado será:

```
CONIN       EQU   1
STROUT      EQU   9
BCPM        EQU   5


START:
            ; Mostra nome do programa
            LD    DE, NOMEDOPROG    ; Indica endereço do nome
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Mostra nome do autor
            LD    DE, AUTOR         ; Indica endereço do autor
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Mostra pergunta
            LD    DE, PERGU1        ; Indica texto da pergunta
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Recebe uma tecla
            LD    C, CONIN          ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            JP    0                 ; Volta ao prompt do CP/M

; Dados do Programa
NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario',10,13,'$'
AUTOR:      DB    ' Por Seu_Nome_Aqui',10,10,13,'$'
PERGU1:     DB    ' Pressione alguma tecla: $'

            END
```

Ao assemblar e executar este código, é possível ver que ele será executado normalmente, pedirá que uma tecla seja digitada e então o programa é finalizado. Mas como conseguir o valor digitado pelo usuário?

Como visto nas aulas teóricas, o lugar usual em que a ULA coloca os resultados das operações é o registrador A (Acumulador). Para não fundir a cabeça dos programadores,

quase sempre se prepara funções que retornam valor para que também coloquem seus resultados neste registrador. Assim, como CONIN não é uma função estranha, o valor numérico da tecla pressionada está no registrador A (seu valor ASCII).

Para apresentar esta informação para o usuário, é preciso primeiro construir uma frase de resposta que será impressa, como por exemplo:

```
RESP1:      DB    13,10,10,'  A tecla pressionada foi: $'
```

Observe que o texto se inicia com a indicação para que duas linhas sejam puladas e nenhuma linha é pulada ao fim do texto. Isso ocorre para que as linhas sejam puladas após o usuário digitar o valor mas não entre o texto "a tecla foi pressionada: " e o valor ser apresentado. A apresentação deste texto deve ser feita depois que o Z80 recebeu a tecla que foi pressionada, como pode ser visto no código a seguir:

```
CONIN       EQU   1
STROUT      EQU   9
BCPM        EQU   5

START:
            ; Mostra nome do programa
            LD    DE, NOMEDOPROG    ; Indica endereço do nome
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Mostra nome do autor
            LD    DE, AUTOR         ; Indica endereço do autor
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Mostra pergunta
            LD    DE, PERGU1        ; Indica texto da pergunta
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Recebe uma tecla
            LD    C, CONIN          ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Apresenta resposta
            LD    DE, RESP1         ; Indica texto da resposta
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            JP    0                 ; Volta ao prompt do CP/M

; Dados do Programa
NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario',10,13,'$'
AUTOR:      DB    ' Por Seu_Nome_Aqui',10,10,13,'$'
PERGU1:     DB    ' Pressione alguma tecla: $'
RESP1:      DB    13,10,10,'  A tecla pressionada foi: $'

            END
```

Quase tudo pronto, mas falta ainda mostrar a tecla pressionada na tela. Isso pode ser feito com a função CONOUT (número 2, CONsole OUT) do CP/M, que faz exatamente o inverso de CONIN: pega um código ASCII de uma tecla e apresenta o texto dela na tela. Uma diferença importante, entretanto, é que o CONIN recebe o valor do teclado no registrador A, enquanto o CONOUT envia o valor da tecla presente no registrador E para a tela; por isso, antes de enviar o valor para a tela, é preciso copiá-lo do registrador A para o E, usando a instrução LoaD:

```
LD    E,A
```

Observe que a instrução LD E, A executa algo similar a E := A, ou seja, o valor que existia em E será **sobrescrito** e o número que estava em A continua em A. Inserindo a nova função no código, é obtido:

```
CONIN       EQU   1
CONOUT      EQU   2
STROUT      EQU   9
BCPM        EQU   5

START:      ; Mostra nome do programa
            LD    DE, NOMEDOPROG    ; Indica endereço do nome
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Mostra nome do autor
            LD    DE, AUTOR         ; Indica endereço do autor
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Mostra pergunta
            LD    DE, PERGU1        ; Indica texto da pergunta
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Recebe uma tecla
            LD    C, CONIN          ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Apresenta resposta
            LD    DE, RESP1         ; Indica texto da resposta
            LD    C, STROUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            ; Apresenta caractere digitado pelo usuário
            LD    E,A               ; Coloca em E o caractere
            LD    C, CONOUT         ; Indica função do CP/M
            CALL  BCPM              ; Solicita execução pelo CP/M

            JP    0                 ; Volta ao prompt do CP/M

; Dados do Programa
NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario',10,13,'$'
AUTOR:      DB    ' Por Seu_Nome_Aqui',10,10,13,'$'
PERGU1:     DB    ' Pressione alguma tecla: $'
RESP1:      DB    13,10,10,'  A tecla pressionada foi: $'

            END
```

Aparentemente está tudo ok, mas se alguém assemblar este programa e tentar executar este programa, ele sempre responderá que a tecla pressionada foi a tecla ' '. Onde está o problema? Este é um problema comum em assembly e linguagem de máquina e os programadores precisam ficar atentos.

Pelo código anterior, é possível observar que entre ser lido o caractere e a apresentação do caractere existe um trecho de código que imprime a primeira parte da resposta ("Apresenta Resposta"). Se esta parte for comentada (com um ";" no início de cada linha) e o código assemblado, será possível ver que o problema some (a tecla pressionada será repetida logo ao seu lado)! O que aconteceu?

O que aconteceu foi que quando foi chamada a função STROUT para imprimir o primeiro trecho da mensagem, ela **modificou** o valor do registrador A, onde havia sido armazenado o caractere digitado. Mas como evitar este problema? Existem duas formas: a primeira delas é descobrindo um registrador que a função STROUT não use, e armazenando o valor neste registrador. Infelizmente a função STROUT modifica quase todos os registradores, o que nos leva a uma outra alternativa: guardar este valor na **memória!**

Mas como fazer isso? Bem, primeiro é necessário criar um local para armazenamento, com uma etiqueta, dentro de nosso programa. Isso pode ser feito com a seguinte linha:

```
VAR1:       DB     0
```

Isso definiu um byte de memória onde se pode ler (como foi feito com os textos) ou escrever, com a etiqueta VAR1 (de VARiável 1), com o valor inicial igual a 0. Mas como colocar um valor neste endereço de memória? Exatamente com a instrução LoaD. Ela também serve para copiar coisas da **memória** para um **registrador**, ou de um **registrador** para a **memória**. Se a idéia é armazenar o dado do registrador A na posição de memória VAR1, nada mais natural do que um comando:

```
LD     VAR1, A
```

Mas, infelizmente, isso não funciona, porque esta ordem não é clara. É importante lembrar que VAR1 é uma etiqueta que é convertida para um número, no momento da assemblagem. Assim, no momento da assemblagem isso poderia virar uma instrução bizarra do tipo:

```
LD     1253, A
```

O que, em uma linguagem como Pascal ou Delphi equivaleria a escrever:

1253 := X ;

Certamente isso ia causar um erro no compilador porque não podemos mudar o valor de um número. 1253 é e sempre será 1253. Esta instrução escrita tentaria **mudar o nome do**

endereço de memória, o que é impossível e nada desejável. O que se deseja é mudar o **conteúdo** deste endereço de memória. Para indicar isso, em assembly, usamos **parênteses** ao redor do número (ou etiqueta), da seguinte forma:

```
LD    (VAR1), A
```

Isso será compreendido pelo Z80 como "Escreva o valor do registrador A na posição de memória cujo nome é VAR1". Inserindo isso *logo após a leitura do caractere*, tem-se:

```
CONIN      EQU   1
CONOUT     EQU   2
STROUT     EQU   9
BCPM       EQU   5

START:     ; Mostra nome do programa
           LD    DE, NOMEDOPROG    ; Indica endereço do nome
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           ; Mostra nome do autor
           LD    DE, AUTOR         ; Indica endereço do autor
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           ; Mostra pergunta
           LD    DE, PERGU1        ; Indica texto da pergunta
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           ; Recebe uma tecla
           LD    C, CONIN          ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M
           LD    (VAR1),A          ; Armazena valor lido na memória

           ; Apresenta resposta
           LD    DE, RESP1         ; Indica texto da resposta
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           ; Apresenta caractere digitado pelo usuário
           LD    E,A               ; Coloca em E o caractere
           LD    C, CONOUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           JP    0                 ; Volta ao prompt do CP/M

; Dados do Programa
NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario',10,13,'$'
AUTOR:     DB    ' Por Seu_Nome_Aqui',10,10,13,'$'
PERGU1:    DB    ' Pressione alguma tecla: $'
RESP1:     DB    13,10,10,'  A tecla pressionada foi: $'
VAR1:      DB    0

           END
```

Agora o dado já está preservado na memória, só falta lê-lo de volta para o registrador E no momento em que for necessário usá-lo. Infelizmente, a instrução:

```
LD    E, (VAR1)
```

Não funciona. A Unidade de Controle do Z80 não sabe ler valores da memória diretamente para o registrador E, ela precisa usar o registrador A para isso... e só então será possível copiar o valor lido do registrador A para o E, com a seqüência:

```
LD    A, (VAR1)
LD    E, A
```

O resultado pode ser visto no próximo trecho de código:

```
CONIN      EQU   1
CONOUT     EQU   2
STROUT     EQU   9
BCPM       EQU   5

START:     ; Mostra nome do programa
           LD    DE, NOMEDOPROG    ; Indica endereço do nome
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           ; Mostra nome do autor
           LD    DE, AUTOR         ; Indica endereço do autor
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           ; Mostra pergunta
           LD    DE, PERGU1        ; Indica texto da pergunta
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           ; Recebe uma tecla
           LD    C, CONIN          ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M
           LD    (VAR1),A          ; Armazena valor lido na memória

           ; Apresenta resposta
           LD    DE, RESP1         ; Indica texto da resposta
           LD    C, STROUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           ; Apresenta caractere digitado pelo usuário
           LD    A,(VAR1)          ; Recupera valor do caract. em A
           LD    E,A               ; Coloca em E o caractere
           LD    C, CONOUT         ; Indica função do CP/M
           CALL  BCPM              ; Solicita execução pelo CP/M

           JP    0                 ; Volta ao prompt do CP/M

; Dados do Programa
NOMEDOPROG: DB 'Programa 2 - Conversando com o usuario',10,13,'$'
AUTOR:     DB    ' Por Seu_Nome_Aqui',10,10,13,'$'
PERGU1:    DB    ' Pressione alguma tecla: $'
RESP1:     DB    13,10,10,'  A tecla pressionada foi: $'
VAR1:      DB    0

           END
```

5. BIBLIOGRAFIA

ROSSINI, F; LUZ, H.F. **Linguagem de Máquina**: Assembly Z80 - MSX. 1ed. São Paulo: Ed. Aleph, 1987.

CARVALHO, J.M. **Assembler para o MSX**. 1ed. São Paulo: McGraw Hill, 1987.

ASCII Corporation. **O Livro Vermelho do MSX**. 1ed. São Paulo: Ed. McGraw Hill, 1988.