

UNIVERSIDADE DO MINHO

Escola de Engenharia  
Departamento de Informática



**Versão 1.2**

**Manual do Utilizador**

José Carlos Rufino Amaro  
Jorge Alexandre Santos

Orientação:  
Engº Fernando Mário Martins

Braga  
1994

1 Introdução .....	2
1.1 Estrutura do Manual .....	2
1.2 Requisitos mínimos de Hardware e Software .....	2
2 Edição de programas MSP .....	3
2.1 Modo Assistido.....	3
2.1.1 Princípios Básicos .....	3
2.1.2 Front-end do Modo Assistido.....	3
2.1.3 Lista de Instruções .....	4
Configurações da Lista de Instruções.....	4
Sequências de Teclas e Toques de Rato .....	5
Ajuda Contextual.....	9
2.1.4 Zona de Edição .....	10
Relação com a Lista de Instruções .....	10
Sequências de Teclas e Toques de Rato .....	10
2.1.5 Comutador de Modo.....	14
2.1.6 Barra de Opções .....	14
2.1.7 Assemblagem .....	17
2.2 Modo Normal .....	19
2.2.1 Princípios Básicos .....	19
2.2.2 Front-end do Modo Normal.....	19
3 Execução de programas MSP .....	20
3.1 Acesso à Janela de Execução .....	20
3.2 Descrição da Janela de Execução .....	21
3.2.1 A Barra de Opções .....	22
3.2.2 A Zona Informativa .....	23
3.3 Input durante a execução - IN e INC.....	25
3.4 Erros de Execução .....	26

# 1 Introdução

Este manual descreve a aplicação<sup>1</sup> WinMSP 1.2, um ambiente integrado de programação em MSP, correndo em Windows 3.1.

O WinMSP 1.2, para além de ser compatível com o MSP da versão 3.1 para DOS, estende ainda a linguagem com a possibilidade de usar números negativos e operações *bit-a-bit*. A validação dos acessos às Memórias e *Stack* foi também reforçada a fim de garantir acessos apenas dentro do espaço de endereçamento normal ([0, 31999]) e evitar também colisões no espaço reservado para as diferentes variáveis.

Mas, para o utilizador, talvez a principal diferença entre esta e anteriores versões resida no ambiente amigável, intuitivo e integrado que o WinMSP 1.2 pretende oferecer.

Para além de tirar partido do ambiente Windows 3.1, o WinMSP 1.2 disponibiliza ainda um modo de programação caracterizado por uma assistência sintáctica constante, a fim de ajudar os programadores menos experientes a iniciar-se na linguagem. Para os que já dominarem o MSP, em vez do **Modo Assistido**, a alternativa é o **Modo Normal**, semelhante a um editor de texto normal.

O WinMSP 1.2 torna ainda possível acompanhar pormenorizadamente (e à velocidade que se desejar) todos os passos da execução de um programa MSP. O comportamento dinâmico da *Stack* é simulado e os próprios acessos à Memória de Dados são também evidenciados.

Refira-se ainda que as componente de edição e de execução de programas MSP encontram-se agora completamente integradas numa única aplicação, o que torna ainda mais recomendável o uso do WinMSP.

Por todas estas razões, esperamos que o utilizador final ache o ambiente WinMSP produtivo e agradável de utilizar.

## 1.1 Estrutura do Manual

A descrição do WinMSP 1.2 divide-se em duas partes fundamentais:

- a descrição da parte de edição, que por sua vez se subdivide no Modo Normal e no Modo Assistido;
- a descrição da parte de execução, que compreende também uma referência dos erros de execução.

## 1.2 Requisitos mínimos de Hardware e Software

Recomenda-se executar o WinMSP 1.2 numa máquina com pelo menos um processador 386SX a 16Mhz, 2MB RAM e Windows 3.1 ou superior.

---

<sup>1</sup>Para uma descrição completa da Linguagem MSP em que se baseia o WinMSP 1.2, consultar o Manual da Linguagem.

## 2 Edição de programas MSP

### 2.1 Modo Assistido

#### 2.1.1 Princípios Básicos

Quando se executa o WinMSP, entra-se por defeito no Modo Assistido. Este modo de edição, tenta assegurar, em qualquer instante, que o programa está correcto sob o ponto de vista sintáctico. Sendo assim, se carregarmos um ficheiro para o editor assistido, ele será implicitamente assemblado, e só se não tiver erros de assemblagem é que lhe é "permitido" permanecer nesse modo. Se pelo contrário há erros sintácticos, eles são assinalados e comuta-se imediatamente para o Modo Normal. O Utilizador terá então que corrigir os erros do programa se quiser regressar ao Modo Assistido.

Uma vez no Modo Assistido, a edição de um programa MSP é feita praticamente à base de toques de rato e janelas que emergem pedindo pela introdução dos campos de uma declaração de variável, dos argumentos de uma instrução, etc.

#### 2.1.2 Front-end do Modo Assistido

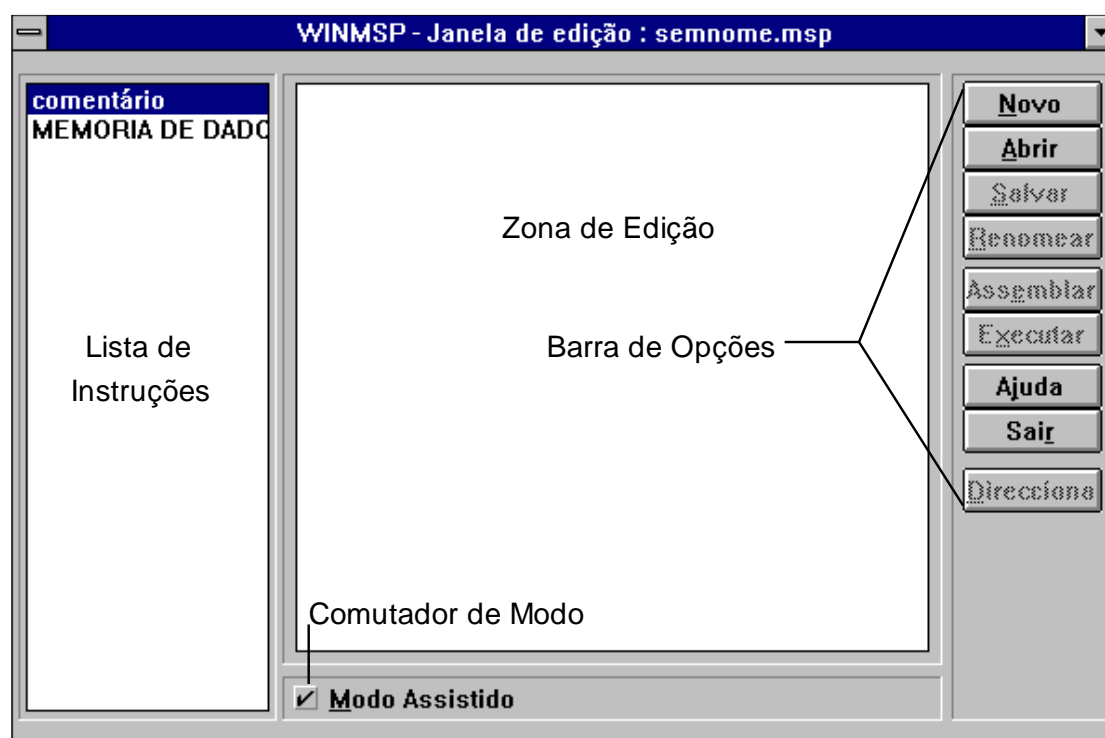


Figura 1 - *Front-end* do Modo Assistido

A Figura 1 apresenta o *front-end* de entrada do WinMSP 1.2 em Modo Assistido. Basicamente, a área de trabalho divide-se em quatro elementos principais:

- uma **Lista de Instruções**, sensível ao contexto, de tal forma que apenas disponibiliza os elementos da linguagem que podem ser introduzidos no programa actual sem invalidá-lo sintacticamente;
- uma **Zona de Edição**, onde se encontra o programa MSP em construção;
- um **Comutador de Modo**, que permite transitar entre o Modo Assistido e o Normal (e vice-versa);
- uma **Barra de Opções**, com funcionalidades várias, eventualmente disponíveis;

### 2.1.3 Lista de Instruções

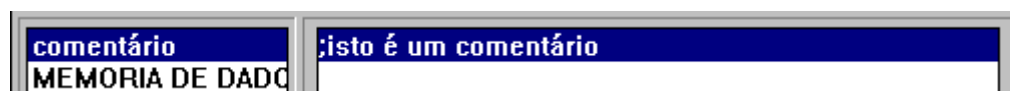
A Lista de Instruções não compreende apenas elementos do *Instruction-Set*. Declarações completas de uma variável, *labels*, comentários e os separadores `MEMORIA DE DADOS` e `CODIGO` são possíveis de se introduzirem directamente no programa a ser construído na Zona de Edição. A Lista de Instruções, disponibiliza assim todos os constructores eventualmente necessários para escrever programas MSP.

A principal característica da Lista de Instruções é a sensibilidade ao contexto: a Lista de Instruções adapta-se automaticamente por forma a disponibilizar os constructores que são válidos de introduzir imediatamente a seguir a uma linha do programa seleccionada na Zona de Edição.

### Configurações da Lista de Instruções

As principais configurações da Lista de Instruções são:

- Se a Zona de Edição estiver vazia ou apenas com comentários, então apenas mais comentários ou o separador `MEMORIA DE DADOS` são permitidos;



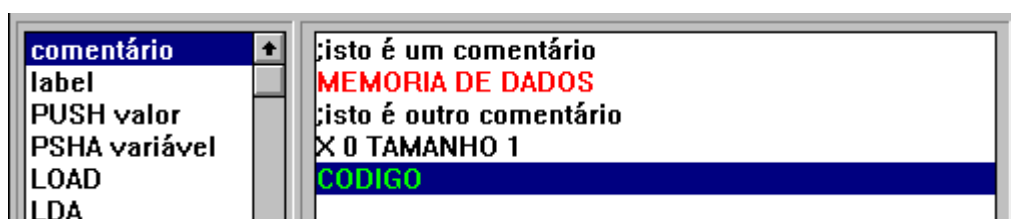
- Tendo introduzido um comentário e o separador `MEMORIA DE DADOS`, então, depois de `MEMORIA DE DADOS` pode vir um comentário, uma declaração de variável ou o separador `CODIGO`;



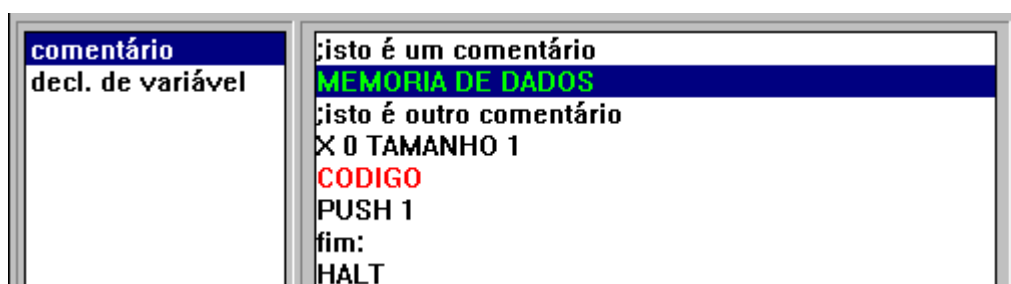
- Antes de `MEMORIA DE DADOS`, apenas comentários (ou linhas vazias) são permitidos;



- Depois de `CODIGO`, só serão permitidos comentários, *labels* e instruções;



- Estando `CODIGO` definido, a Zona de Dados só admite comentários e mais declarações de variáveis;



## Sequências de Teclas e Toques de Rato

Descreve-se aqui a forma de usar o teclado e o rato para seleccionar elementos da Lista de Instruções destinados ao programa MSP em construção.

Os **cursores** (cima e baixo) podem ser usados para nos movimentarmos na Lista de Instruções. Um toque de **rato** é também suficiente para seleccionar um item da lista. Outra maneira de a percorrer é usando as iniciais dos constructores, ou seja, para aceder a todas as instruções começadas por **S**, por exemplo, basta pressionar repetidamente a tecla **s** (ou **S**); desta forma acederíamos, a `STORE`, `STRA` e `SUB`. Este pormenor revela-se prático na medida em que os items da lista não estão ordenados por ordem alfabética, mas sim agrupados por funcionalidade.

Depois de seleccionar o constructor, fazendo *double-click* ou pressionando **ENTER** aparece uma janela destinada a introduzir e validar os diversos componentes do item. Se o item é atómico (caso das instruções sem argumentos), passa imediatamente para a Zona de Edição.

Os elementos não atómicos da Lista de Instruções (e que exigem portanto validação antes da sua introdução no programa) e respectivas janelas de descrição são:

- comentário: qualquer<sup>2</sup> sequência de caracteres;

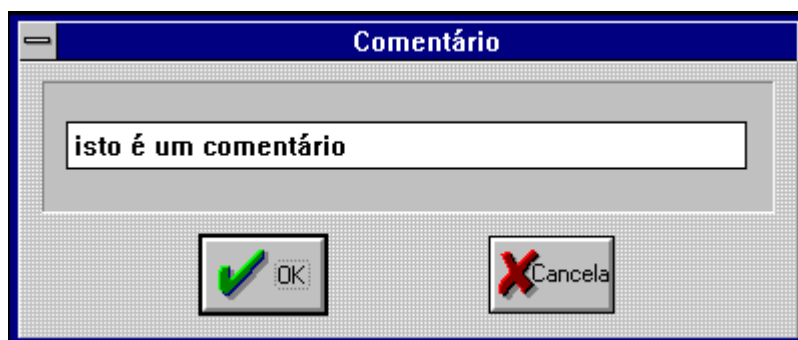


Figura 2

- decl. de variável: permite introduzir e validar todos os campos relativos à declaração de uma variável. A navegação por entre os campos faz-se usando a tecla **TAB** ou acedendo directamente com o ponteiro do **rato**. Pressionando **OK** ou carregando em **ENTER** (mesmo dentro de um campo) dá-se por terminado o preenchimento dos campos.

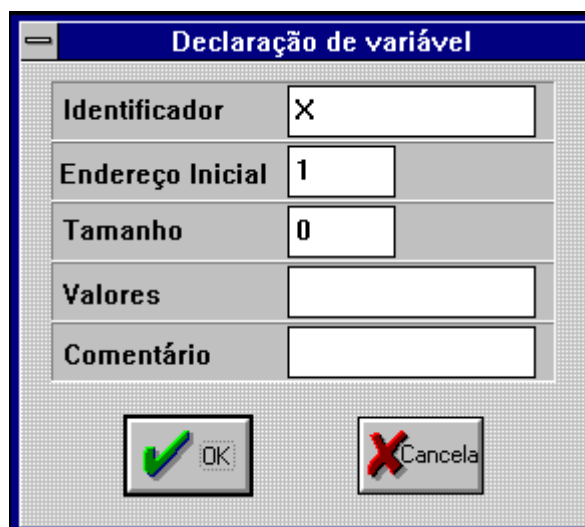


Figura 3

Por exemplo, se se tentar pressionar **OK** sem definir valores para nenhum parâmetro, então a assemblagem (implícita) da Zona de Dados produz:

---

<sup>2</sup>excepto *newline*.

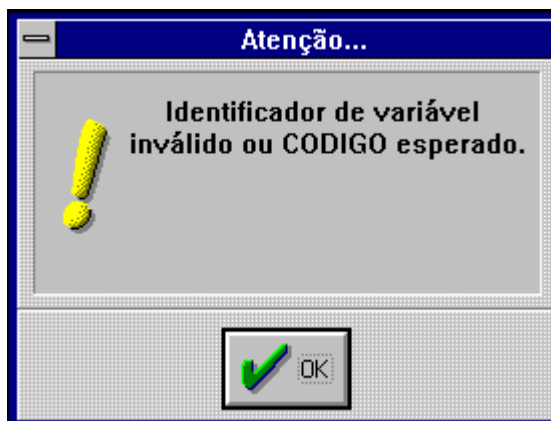


Figura 4

Neste e noutros casos onde a declaração de uma variável possa estar sintacticamente errada ou incompleta, a assistência sintactica permanente encarrega-se de evitar que essa declaração aceda ao programa em construção. Para além destas situações, são ainda detectadas tentativas de usar identificadores coincidentes com os de variáveis previamente declaradas e de tentar alocar zonas da Memória de Dados já reservadas (parcial ou totalmente) também por outras variáveis.

- `label`: permite introduzir e validar uma *label* (de 8 caracteres no máximo). Neste caso, só se verifica se os caracteres da *label* formam uma sequência permitida pela gramática do MSP. Situações como o uso de *labels* repetidas não são detectadas neste contexto (pois não há garantias de que a construção da Zona de Código do programa já tenha acabado).

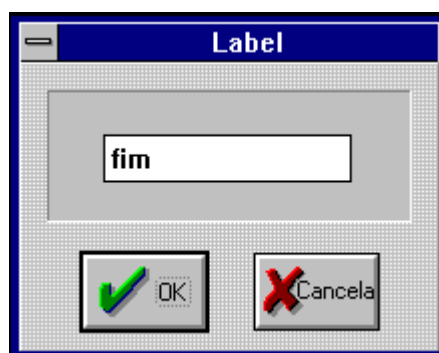


Figura 5

- `PUSH valor`: efectua a introdução e validação do parâmetro `valor` na gama [-128, 255], com sinal positivo opcional.



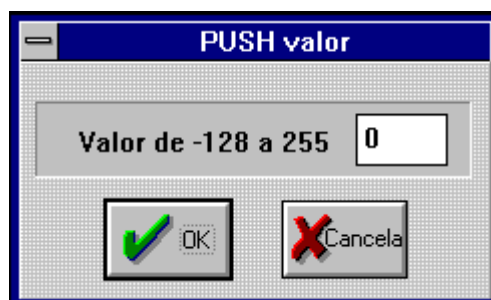


Figura 6

- **PSHA variável:** o parâmetro variável, que pode ser um identificador de variável ou o seu endereço, é introduzido e validado; no caso de um endereço, verifica-se se pertence à gama [0, 31999] mas não se verifica se corresponde ao endereço de uma variável efectivamente declarada na Zona de Dados; no caso de um identificador (e à semelhança da *label*) verifica-se se os caracteres formam uma sequência permitida pela gramática do MSP, mas não se verifica se há realmente uma variável previamente declarada usando esse identificador.

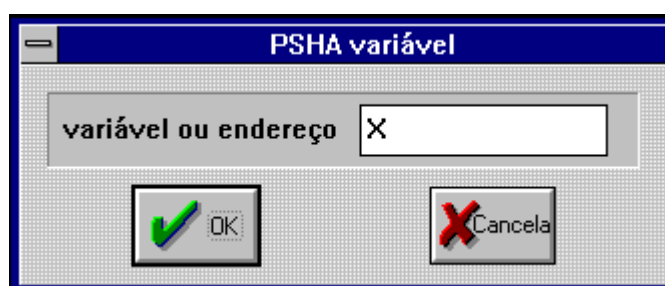


Figura 7

- **JMP subrotina, JMPF subrotina, CALL subrotina:** à semelhança do caso anterior, se o parâmetro for um endereço, não se verifica se existe uma *label* previamente declarada e associada a esse endereço (essa *label* ainda pode vir a ser declarada); se o parâmetro for um identificador, valida-se apenas em termos da gramática do MSP e não se verifica se essa *label* já foi definida.

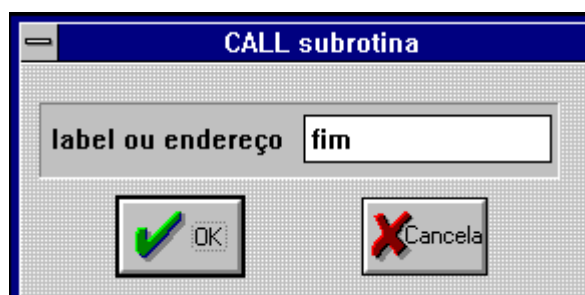


Figura 8

### Nota importante:

As situações previamente referidas em que as validações sintacticas não eram completas devem-se à incerteza sobre o estado actual da síntese do programa, *i.e.*, já terminou ou ainda está em construção? Por isso, instruções que façam referência a uma variável não podem ser validadas testando se essa variável já foi declarada, pois se o não foi, nada impede que tal ainda venha acontecer. Para as *labels*, o constrangimento é semelhante.

Contudo, quando se procede a uma montagem completa do programa, tais situações são imediatamente detectadas sob a forma de erros de montagem, e o utilizador é convidado a corrigi-las.

## Ajuda Contextual

Para obter ajuda sobre um determinado tópico da linguagem MSP basta seleccionar esse tópico na Lista de Instruções e de seguida pressionar o botão de Ajuda na Barra de Opções. Assim, se por exemplo desejássemos informação sobre a sintaxe dos comentários, obteríamos:

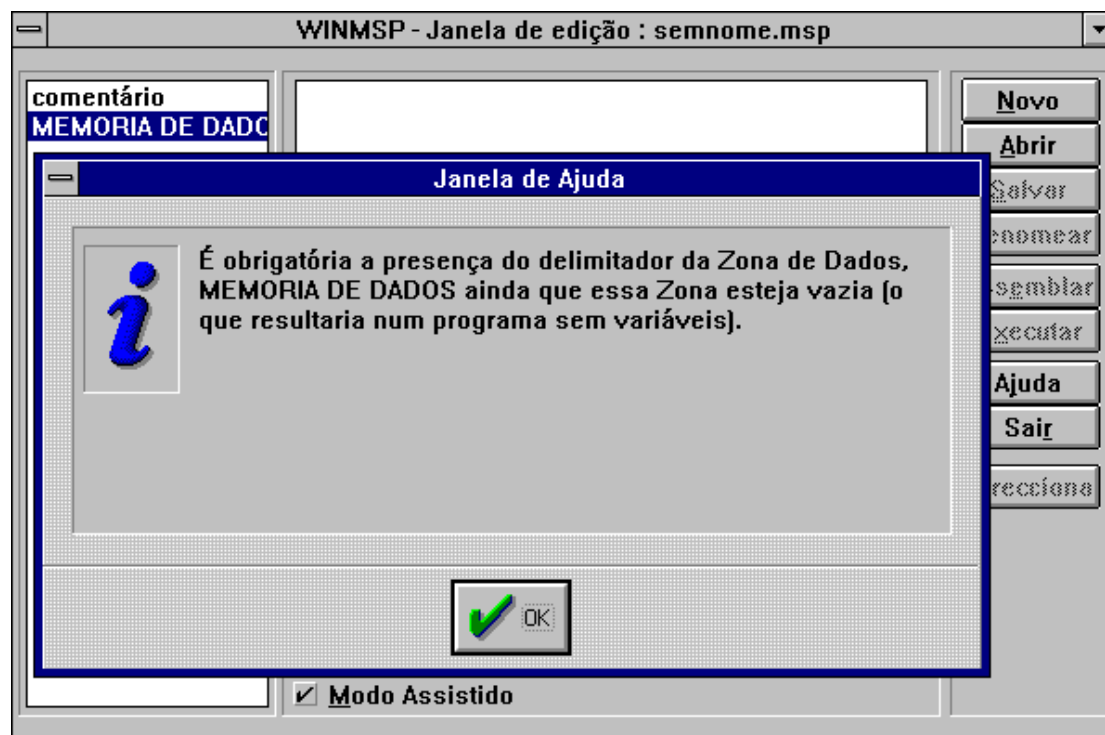


Figura 9

Esta funcionalidade só existe no Modo Assistido, uma vez que o *interface* do Modo Normal não disponibiliza a Lista de Instruções.

### 2.1.4 Zona de Edição

A Zona de Edição é a zona da área de trabalho que comporta o programa MSP que está a ser construído com base na Lista de Instruções. Esse programa pode também ser originário do Modo Normal, e então podemos estar a servir do Modo Assistido para o editar interactivamente.

#### Relação com a Lista de Instruções

Para além de aceitar linhas de programa sintetizadas com base na Lista de Instruções, a Zona de Edição também influencia o estado dessa lista<sup>3</sup>. Assim, sempre que se selecciona um linha da Zona de Edição, seja usando os **cursores** (cima e baixo), seja com um toque de **rato**, a configuração da Lista de Instruções muda por forma a disponibilizar os constructores válidos de introduzir na linha da Zona de Edição imediatamente a seguir à seleccionada.

#### Sequências de Teclas e Toques de Rato

Quer seja proveniente da Lista de Instruções, quer do Modo Normal, é possível editar qualquer linha do programa MSP actualmente na Zona de Edição. Para esse fim é possível usar as seguintes combinações de teclas e toques de rato:

- selecção ainda sem edição: usar **cursores** (cima e baixo) para seleccionar a linha pretendida ou um toque de **rato**;
- edição de uma linha: após selecção: pressionar **ENTER** ou então editar directamente com um duplo toque (*doubleclick*) de **rato** em cima da linha.  
As linhas da Zona de Dados são editadas usando as mesmas janelas que serviram à sua introdução<sup>3</sup>. Assim, uma declaração de variável é editada com base na janela da Figura 3 e um comentário anterior ao separador CODIGO é editado com base na janela da Figura 2. Os separadores MEMORIA DE DADOS e CODIGO não são editáveis.  
Uma linha da Zona de Código é editada com base numa única janela polivalente, que permite qualquer combinação válida na Zona de Código (linhas só de comentários, ou só de *labels*, ou só com instruções, ou com qualquer combinação destas três variantes). Contudo, linhas completamente vazias só são conseguidas comutando para o Modo Normal, criando a(s) linha(s) vazia(s) e regressando ao Modo Assistido.  
As Figuras 10 a 13 mostram algumas possibilidades de edição para a seguinte linha da Zona de Código:

```
Cil:PSHA 0; Controla Ciclo
```

---

<sup>3</sup>recordar 2.1.3 Lista de Instruções, nomeadamente Configurações da Lista de Instruções.

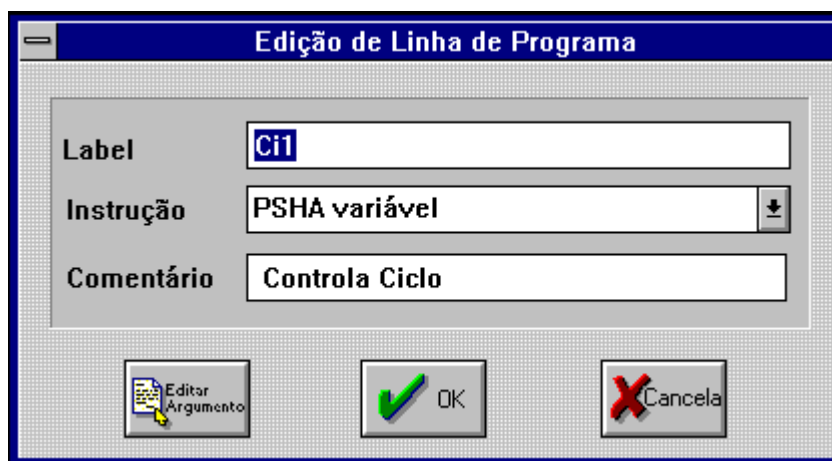


Figura 10

Assim, altera-se a *Label* e o *Comentário* editando os respectivos campos. A *Label* será validada de maneira idêntica à que ocorre no caso da Figura 5. É ainda possível alterar o valor dos eventuais argumentos de uma instrução (nesse caso o botão **Editar Argumentos** está activo), bem como a própria natureza da Instrução.

Por exemplo, a fim de alterar o argumento do exemplo da Figura 10, pressiona-se **Editar Argumentos** e obtém-se:

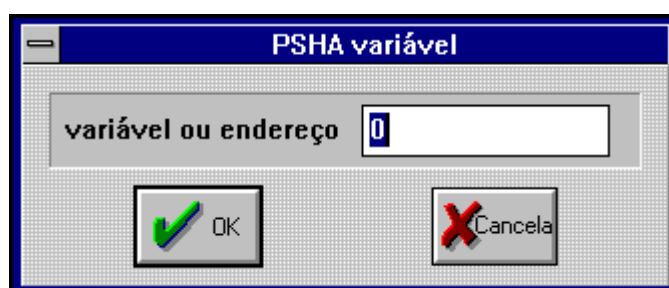


Figura 11

ou seja, uma janela idêntica à da Figura 7 que poderia ter sido usada na introdução de PSHA 0 a partir da Lista de Instruções.

A Figura 12 mostra como alterar a própria instrução: selecciona-se o campo *Instrução* e arrasta-se o **rato** na vertical através de uma lista que contém o *Instruction-Set*, até à instrução desejada. Caso a nova instrução tenha argumento, emerge uma janela a fim de o receber e validar. Essa janela é idêntica à que pode ser posteriormente acedida pelo botão **Editar Argumentos**. Aliás, todas as janelas acessíveis via **Editar Argumentos** pertencem ao conjunto das definidas pelas Figuras 6,7 e 8, *i.e.*, são as mesmas usadas na introdução das instruções na Zona de Edição a partir da Lista de Instruções.

A Figura 13 apresenta o resultado da operação da Figura 12. Repare-se que como LDA não tem argumentos, o botão **Editar Argumentos** fica desactivado.



Figura 12



Figura 13

Como a Lista de Instruções não permite combinar vários constructores da linguagem MSP, então, no Modo Assistido, a única maneira de ter *labels*, instruções e comentários na mesma linha da Zona de Código é através do mecanismo de edição exposto.

- tal como no caso da Lista de Instruções em que era possível aceder aos tópicos pela letra inicial, também aqui é possível um procedimento semelhante: na Zona de Edição, qualquer linha não indentada à esquerda, goza da propriedade de poder ser acedida de qualquer ponto do texto do programa pressionando a tecla correspondente ao 1º carácter. No entanto, como não é possível adivinhar as intenções do utilizador, uma linha introduzida via Lista de Instruções não é por defeito indentada. Adicionalmente, na versão actual (1.2) do WinMSP, a edição de uma linha no Modo Assistido também ainda não tem o cuidado de conservar a indentação original, não indentando por defeito. Assim, por enquanto, a fim de tirar partido desta potencialidade será porventura necessário indentar manualmente no Modo Normal e regressar ao Modo Assistido.

- são possíveis as seguintes operações básicas de *clipboard* sobre as linhas da Zona de Edição: *copy* (**Ctrl+Insert**), *cut* (**Shift+Del**), *paste* (**Shift+Insert**) e *del* (**Delete** ou **Del**). Para seleccionar as linhas-objecto usar **Shift+cursosores** (cima e baixo) ou arrastar o **rato** para marcar as linhas desejadas.

As operações de *clipboard* têm algumas restrições. Assim, só linhas contíguas podem ser objecto de operações de *clipboard*. Contudo, as áreas seleccionadas não poderão integrar nenhum dos delimitadores MEMORIA DE DADOS e/ou CODIGO.

Informação proveniente da Zona Neutra<sup>4</sup> (que só pode ter linhas vazias ou de comentário) pode ser transferida para qualquer zona do programa. A Zona Neutra, porém, só pode receber informação dela própria.

A Zona de Dados só pode importar informação da Zona Neutra e dela própria; por consequência, também só pode exportar para ela própria.

A Zona de Código só pode exportar para si mesma. Pode também importar da Zona Neutra.

As restrições anteriores destinam-se a evitar que as operações de *clipboard* violem a consistência sintáctica que o Modo Assistido procura em qualquer instante.

A Figura 14 mostra o que aconteceria se tentássemos uma operação de *copy* sobre uma área contendo os separadores de zonas:

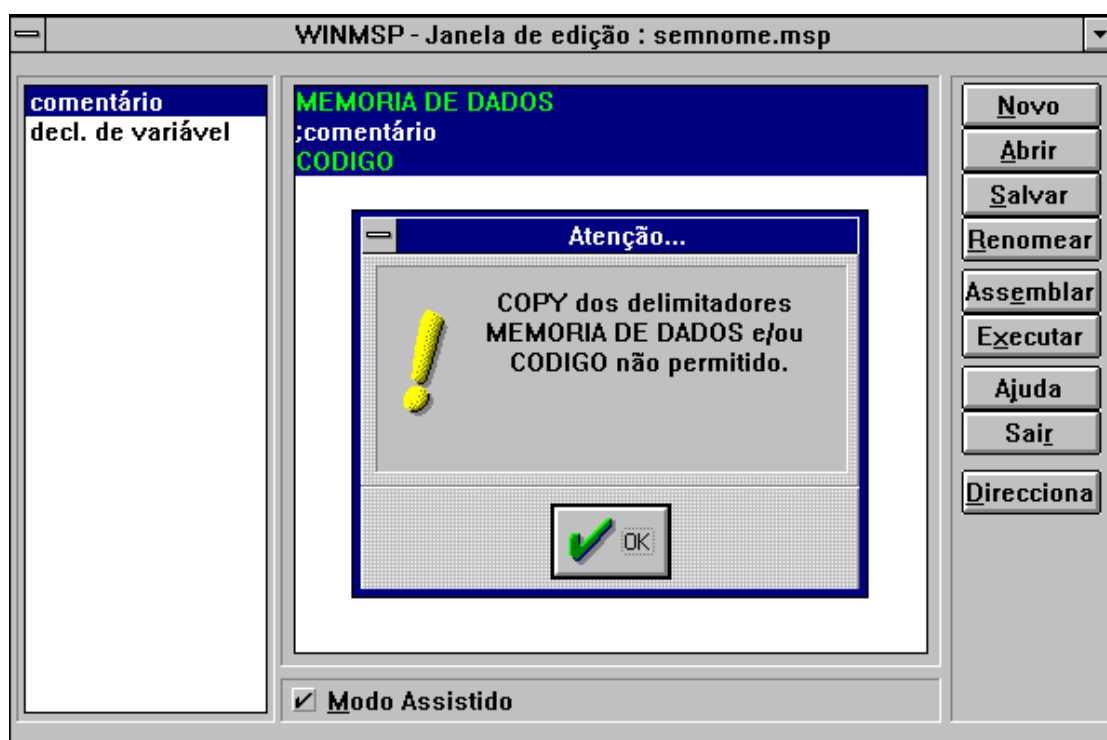


Figura 13

<sup>4</sup>convenciona-se chamar Zona Neutra à zona anterior ao separador MEMORIA DE DADOS; esta terminologia será assumida doravante.

### 2.1.5 Comutador de Modo

O Comutador de Modo é uma zona partilhada pelo *front-end* do Modo Assistido e do Modo Normal e que indica qual o modo de edição em curso. Em Modo Assistido apresenta o aspecto da Figura 14:



Figura 14

Um toque de **rato** na área interior da Figura 14 provoca uma troca de modo (Assistido para Normal e vice-versa).

### 2.1.6 Barra de Opções

A Barra de Opções permite aceder às seguintes funcionalidades:

- **Novo:** verifica se é necessário gravar o programa actual (ver Figura 15) e recomeça a edição com um programa vazio, mantendo o último modo de edição (Assistido ou Normal).

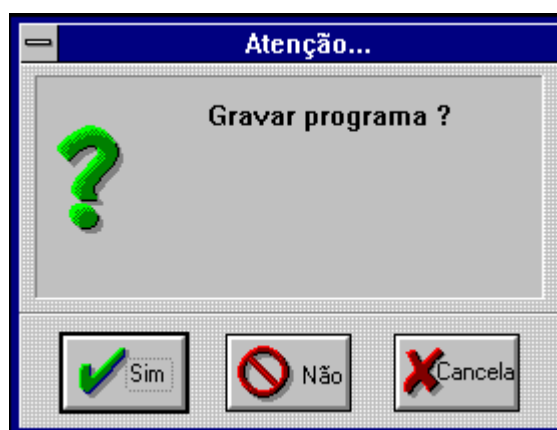


Figura 15

- **Abrir:** verifica se é necessário gravar o programa actual (ver Figura 15), acede a um novo ficheiro (ver Figura 16) e carrega-o para o modo actual de edição. Se o carregamento for feito em Modo Assistido e o programa falhar a assemblagem implícita que é feita, comuta-se automaticamente para o Modo Normal.
- **Salvar:** grava incondicionalmente o ficheiro actualmente em edição (independentemente do modo).
- **Renomear:** permite gravar com um novo nome o ficheiro actualmente em edição (independentemente do modo); ver Figura 17.



Figura 16



Figura 17

- **Assemblar:** faz uma assemblagem completa do programa actualmente em edição (independentemente do modo); a secção **2.1.7 Assemblagem** cobre todos os aspectos do processo de assemblagem.
- **Executar:** efectua uma assemblagem completa e, caso seja bem sucedida, permite aceder à Janela de Execução (independentemente do modo); o capítulo **3 Execução de Programas MSP** cobre todos os aspectos do processo de execução.
- **Ajuda:** fornece ajuda sobre o item actualmente seleccionado na Lista de Instruções (apenas em Modo Assistido); ver também secção **2.1 Modo Assistido**, sub-secção **Ajuda Contextual**;
- **Sair:** verifica se é necessário gravar o programa actual (ver Figura 15), após o que provoca a terminação do WinMSP.



- **Direcciona:** esta opção destina-se a invocar um programa executável que tenha interesse em processar o programa MSP actualmente em edição; para isso, recorre à informação contida no ficheiro `winmsp.ini`:

[Direccionamento]  
Executavel=edit.com

É possível editar manualmente este ficheiro e modificar o destino por defeito. A Janela de Direccionamento (ver Figura 18) sugere sempre como destino do direccionamento o conteúdo da secção Executável do ficheiro `winmsp.ini`, mas também permite alterar *on-line* esse destino, que passa a ser o novo valor da secção Executável.



Figura 18

O estado dos botões da Barra de Opções nem sempre é o mesmo. Relativamente a esse estado, a única diferença que separa o Modo Assistido do Modo Normal é que no Modo Normal o botão de Ajuda está sempre desligado. À parte esse pormenor, a Barra de Opções assume dois estados fundamentais, conforme haja (Figura 19 - Modo Assistido) ou não (Figura 20- Modo Assistido) algum programa MSP em edição.



Figura 19



Figura 20

### 2.1.7 Assemblagem

Esta secção demonstra o processo de assemblagem<sup>5</sup> de um programa MSP, acessível via botão **Assemblar** da Barra de Opções. A demonstração também é válida para a assemblagem implícita que ocorre via botão **Executar**. Ambos os modos de edição (Normal e Assistido) partilham os detalhes da descrição que se segue.

Seja por exemplo o seguinte programa MSP:

```
MEMORIA DE DADOS  
X 0 TAM 1  
Y1 TAM 2
```

A Figura 21 mostra a Janela de Assemblagem produzida pelo WinMSP quando tenta assemblar o programa anterior.

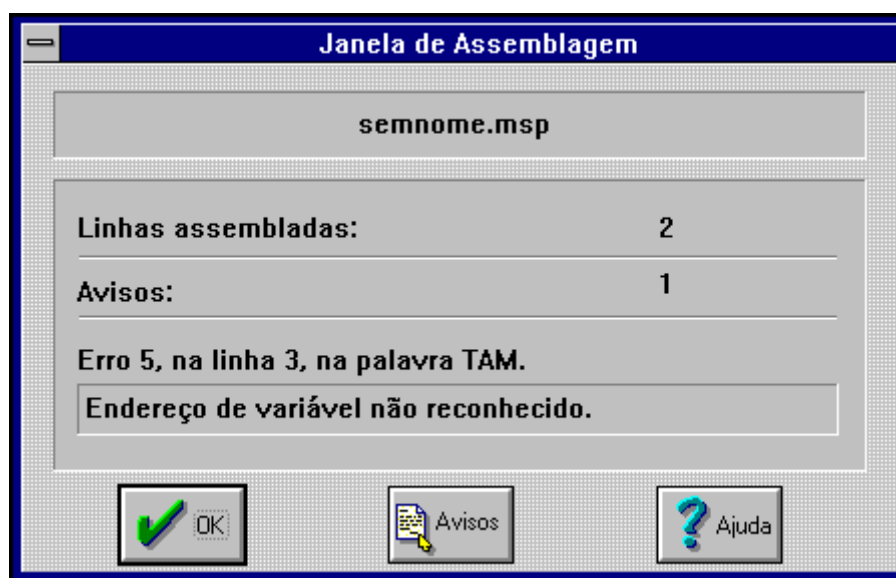


Figura 21

Relativamente à Figura 21, o campo *Linhas assembladas* dá o total de linhas do programa que foram assembladas com sucesso: neste caso, foram apenas 2 linhas já que a 3ª tem um erro sintáctico<sup>6</sup>.

Foi também detectado um aviso (ver campo *Avisos*); recorde-se que um aviso é um alerta para uma situação que apesar de não constituir erro sintáctico pode ser sintoma de um erro semântico por parte do utilizador.

<sup>5</sup>assemblagem completa, pois o Modo Assistido efectua internamente assemblagens incompletas (sobre uma zona específica do programa, por exemplo a fim de manter sintacticamente válida a Zona de Dados ...)

<sup>6</sup>em geral, a assemblagem é interrompida ao 1º erro sintáctico detectado...

De seguida é fornecida informação suficiente para localizar de forma precisa o erro<sup>7</sup>: o código do erro, a linha onde ocorreu e a palavra responsável por esse erro. Uma descrição sumariada do erro é também apresentada. O botão de **Ajuda**<sup>8</sup> permite aceder a uma descrição mais detalhada sobre o significado do erro. Para o caso presente, teríamos:

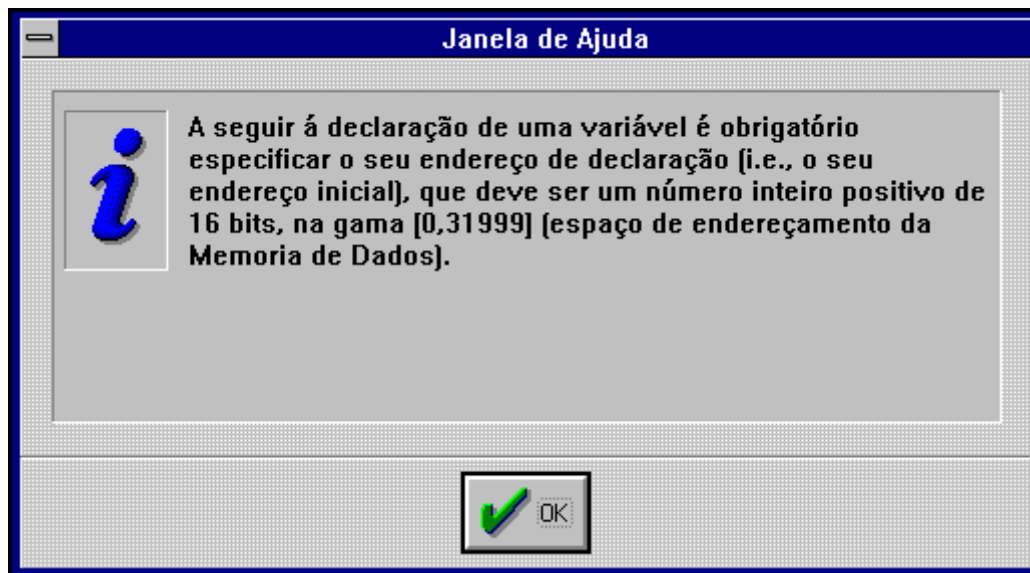


Figura 22

O botão **Avisos** faz emergir a Janela de Avisos (ver Figura 23) com a lista dos avisos detectados. Um toque de **rato** em cada aviso, evidencia a linha do programa que o originou. É também providenciada uma descrição rigorosa do contexto do aviso: o seu código, a linha e a palavra são fornecidos, além duma descrição sumária.

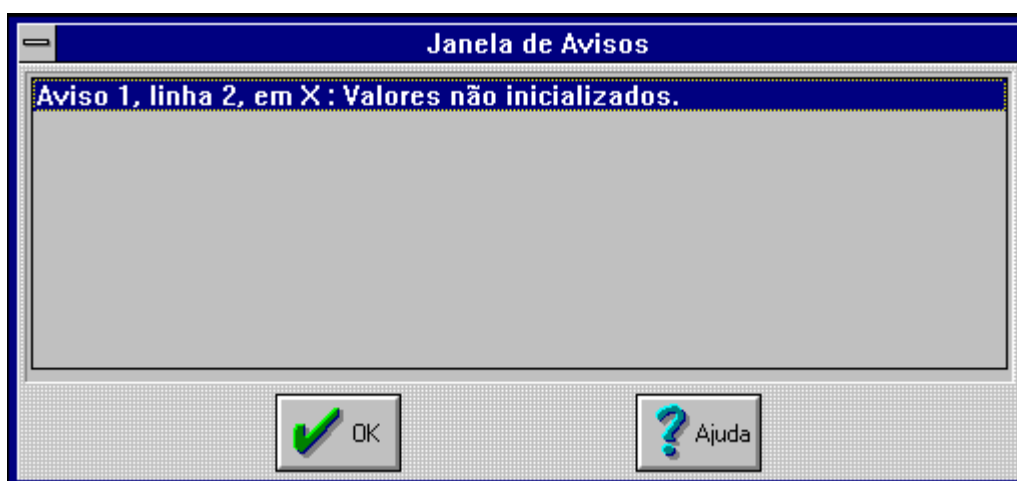


Figura 23

<sup>7</sup>adicionalmente, sempre que ocorre um erro, a linha respectiva no programa é posta em evidência.

<sup>8</sup>caso não haja erro de assemblagem, este botão estará inibido.

Uma descrição mais detalhada<sup>9</sup> sobre o aviso seleccionado é acessível via botão **Ajuda** da Janela de Avisos (ver Figura 24).

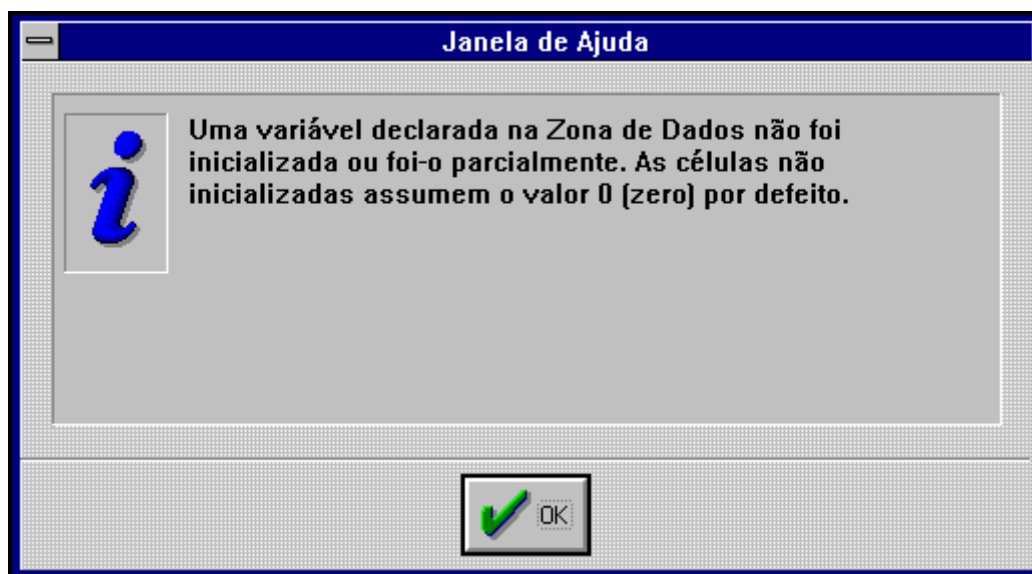


Figura 24

## 2.2 Modo Normal

### 2.2.1 Princípios Básicos

Ao contrário do Modo Assistido, o Modo Normal de edição não fornece assistência sintáctica nem ajuda contextual, tendo sido implementado a pensar naqueles utilizadores cuja prática dispensa uma ajuda constante e insistente, limitativa porventura da sua produtividade. O Modo Normal ainda dispõe da Barra de Opções e do Comutador de Modo, ambos de funcionalidades idênticas às que possuem no Modo Assistido, mas dispensa a Lista de Instruções. Em vez disso, a área de trabalho é praticamente "açambarcada" pela Zona de Edição, destinada apenas a editar texto, sem qualquer validação.

### 2.2.2 *Front-end* do Modo Normal

A Figura 25 dá conta do *front-end* do WinMSP em Modo Normal de Edição. São três as suas componentes principais:

- Zona de Edição: é basicamente um editor de texto de funcionalidades muito reduzidas; dispõe de operações de *clipboard* usando as mesmas combinações de teclas e toques de **rato** referidas para o Modo Assistido<sup>10</sup>, embora sem as restrições impostas pela assistência sintáctica permanente;

<sup>9</sup>o código de um erro ou de um aviso, o seu sumário e descrição detalhada coincidem com o exposto na secção **3.4 Erros e Avisos de Assemblagem** do Manual da Linguagem.

<sup>10</sup> rever **2.1.4 Zona de Edição, Sequências de Teclas e Toques de Rato**.

- Comutador de Modo: funções iguais às desempenhadas no Modo Assistido<sup>11</sup>;
- Barra de Opções: funções idênticas às do Modo Assistido<sup>12</sup>, com excepção do botão de **Ajuda** que está permanentemente desactivado;

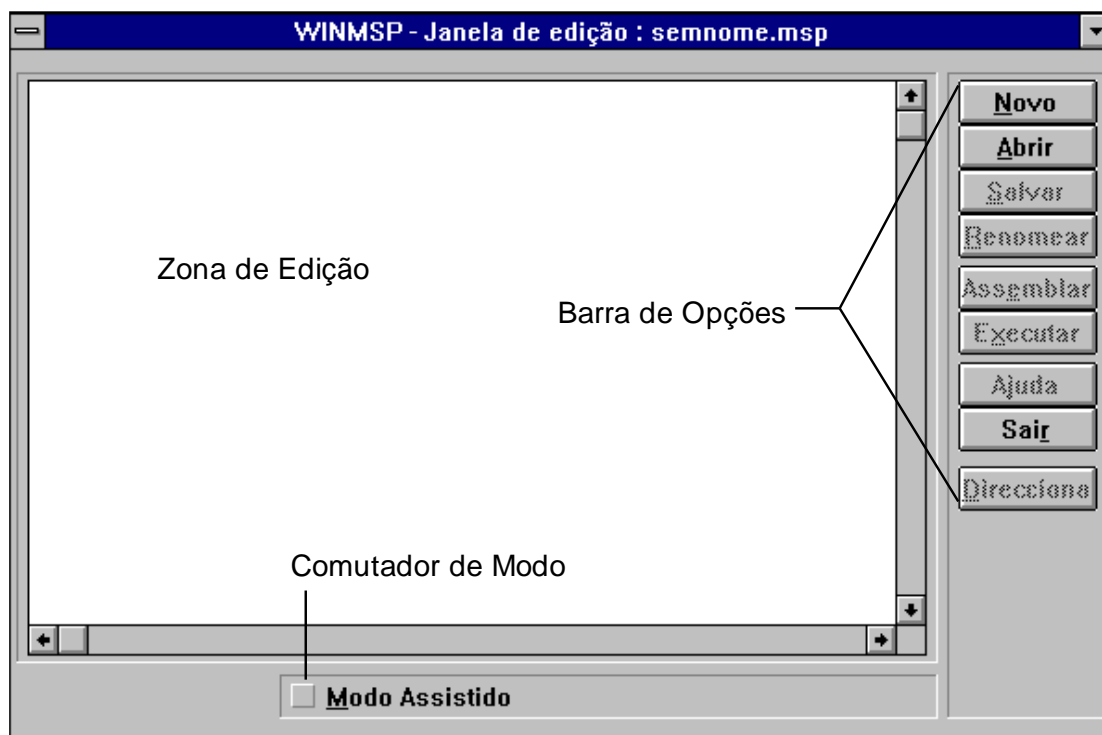


Figura 25 - *Front-end* do Modo Normal

### 3 Execução de programas MSP

#### 3.1 Acesso à Janela de Execução

O Acesso à Janela de Execução é feito da mesma maneira a partir da Janela de Edição em ambos os modos (Normal ou Assistido). Na Barra de Opções, pressiona-se o botão **Executar** o que desencadeia a assemblagem do programa a fim de gerar o respectivo código executável.<sup>13</sup> A diferença relativamente a uma assemblagem isolada reside agora no facto de que não havendo erros de assemblagem, o botão **OK** da Figura 21 dá acesso à Janela de Execução em vez de regressar à Janela de Edição.

<sup>11</sup>rever 2.1.5 Comutador de Modo

<sup>12</sup>rever 2.1.6 Barra de Opções

<sup>13</sup>na realidade, este código não é directamente executável na família *ix86*, sendo interpretado a fim de simular o funcionamento da Máquina Virtual de Stack.

### 3.2 Descrição da Janela de Execução

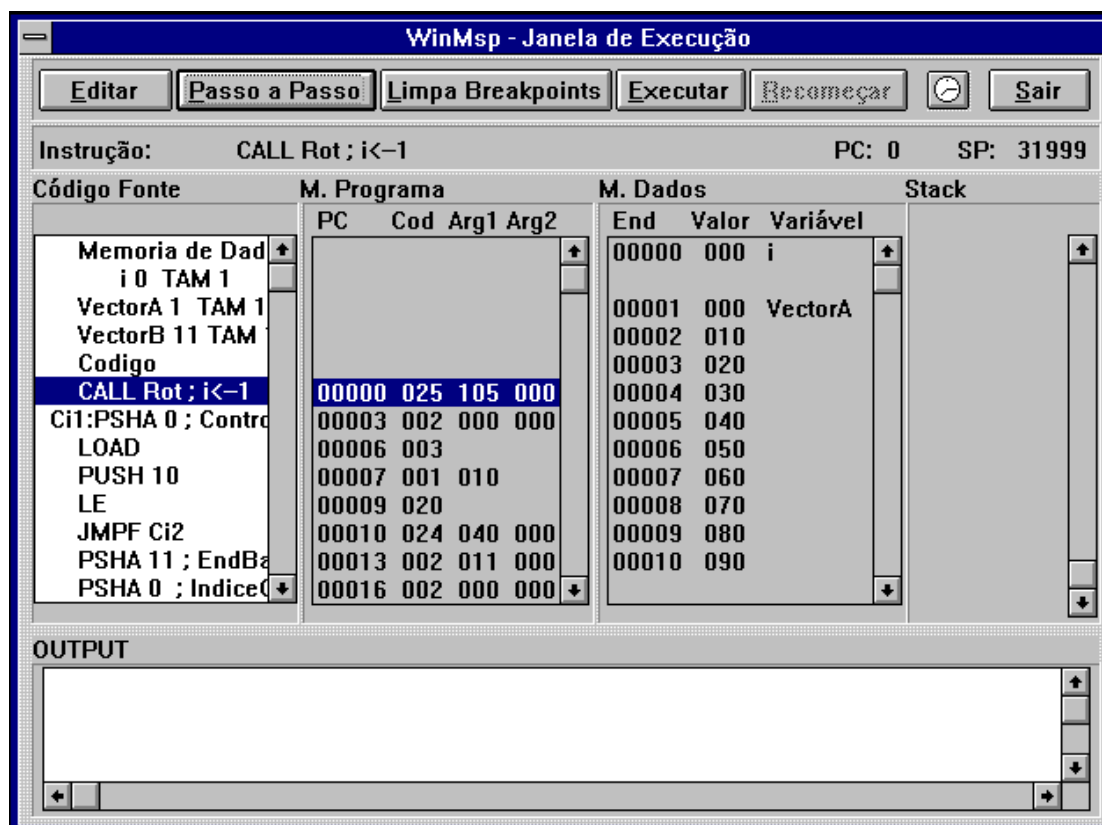


Figura 26 - A Janela de Execução

A Figura 26 apresenta a Janela de Execução, pronta a iniciar a execução de um determinado programa previamente assemblado num dos dois modos de edição (Normal ou Assistido).

Nesta janela são controlados todos os detalhes da execução de programas MSP, bem como é fornecida toda a informação relevante sobre o seu estado de execução num determinado instante.

A fim de facilitar a descrição das facilidades que ela oferece, podemos considerá-la dividida em duas áreas fundamentais:

- **Barra de Opções:** é o conjunto de botões da parte superior da janela que permite controlar a execução propriamente dita de um programa MSP, bem como regressar, eventualmente á sua edição;
- **Zona Informativa:** a restante área da janela que basicamente se ocupa de informar sobre o estado da execução do programa, reflectindo as acções desencadeadas a nível da Barra de Opções.

As próximas secções refinam a descrição de cada uma destas áreas.

### 3.2.1 A Barra de Opções

A descrição das funcionalidades que esta zona do *interface* de execução implementa é feita botão a botão.

- **Editar:** permite, em qualquer instante, regressar à edição do código fonte do programa que se estava a executar; essa edição, é retomada no modo de edição (Normal ou Assistido) a partir do qual foi despoletada a execução<sup>14</sup>
- **Passo a Passo:** possibilita acompanhar a execução do programa, instrução a instrução. Após a execução da instrução actual, o programa é "congelado", i.e., fica à espera de nova ordem para prosseguir a execução na instrução apontada pelo PC (*Program Counter*). Esta funcionalidade permite observar atentamente todos os acessos (de leitura, escrita ou execução) às memórias usadas pelo programa (Memoria de Dados, Memoria de Programa e *Stack*). A fim de progredir instrução a instrução, é suficiente um toque de **rato** no botão **Passo a Passo**. Se, a partir de determinada altura desejarmos que o programa execute normalmente, i.e., sem parar após a execução de cada instrução, basta um toque de **rato** no botão **Continuar**<sup>15</sup>. Nestas circunstâncias, se quisermos voltar a ter acesso ao modo passo-a-passo, teremos de interromper o modo normal de execução pressionando o botão **Stop**<sup>16</sup>; verifica-se então que o botão **Passo a Passo** está novamente activo.
- **Limpa Breakpoints:** este botão permite eliminar todos os pontos de paragem (*breakpoints*) colocados ao longo do programa. Estes pontos de paragem suspendem (congelam) a execução do programa imediatamente antes da instrução que lhes está associada, e independentemente de a execução estar a ser feita passo-a-passo ou normalmente. Os *breakpoints* são introduzidos com um duplo toque (*doubleclick*) de **rato** sobre a linha com a instrução pretendida, na janela do Código Fonte. A linha muda de cor, indicando a associação de um *breakpoint* à respectiva instrução<sup>17</sup>. É possível introduzir *breakpoints* durante a própria execução do programa. Para eliminar um *breakpoint* em particular, basta um novo *doubleclick* na linha correspondente. Para eliminá-los todos de uma só vez, é mais expedito pressionar o botão **Limpa Breakpoints**.
- **Executar:** este botão inicia a execução normal do programa (outra forma de iniciar a execução é através do botão **Passo a Passo**) e assume diversas "personalidades". Assim, antes de executar o programa, o botão chama-se efectivamente **Executar**. Pressionando-o, entramos no modo normal de execução e o botão passa a chamar-se **Stop**. Se **Stop** é pressionado ou a execução é interrompida por um *breakpoint*, o botão passa a designar-se **Continuar** (identificação que mantém caso a execução do programa prossiga passo a passo). Pressionando **Continuar**, muda novamente para **Stop**. Só

<sup>14</sup>pressionando o botão **Executar**, recorde-se.


<sup>15</sup>o botão **Continuar** é o botão no qual se transforma o botão **Executar** sempre que se interrompe a execução do programa, seja por execução **Passo a Passo**, seja por Breakpoint, seja por activação do botão de **Stop**.

<sup>16</sup>o botão **Stop** é o botão no qual se transforma o botão **Continuar** após ser pressionado.

<sup>17</sup>Logicamente, só são permitidos pontos de paragem na Zona de Código...

através do botão **Recomeçar** é que colocamos o botão **Continuar** no seu estado (**Executar**).

- **Recomeçar**: se se pretender terminar o programa actualmente em execução e re-executá-lo, um toque de **rato** neste botão é suficiente. Durante uma execução passo-a-passo, este botão está sempre activo. Porém, durante uma execução normal, há que pressionar **Stop** ou aproveitar um *breakpoint* para aceder ao botão **Recomeçar**.

-  (Velocidade de Execução): este botão permite alterar a velocidade de execução do programa MSP, mesmo durante a sua execução. A Figura 27 mostra a janela que surge a fim de configurar este parâmetro.

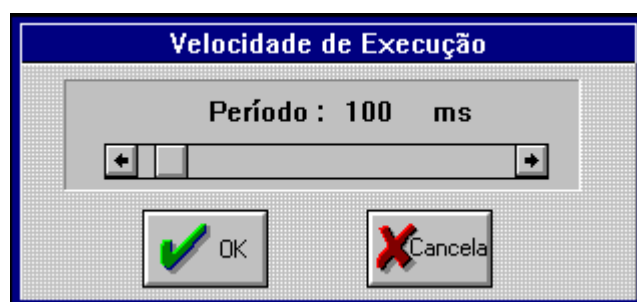


Figura 27

Por defeito, o *Período*, ou seja, o intervalo de tempo entre a execução de 2 instruções sucessivas é de 100ms (100 milissegundos). Valores mais altos tornam a execução mais lenta e vice-versa.

A possibilidade de alterar a velocidade de execução é uma forma de adaptar o WinMSP 1.2 à velocidade específica de cada máquina. Permite ainda correr zonas diferentes do programa a diferentes velocidades.

- **Sair**: permite abandonar o WinMSP 1.2, mesmo durante a execução de um programa. Se o código fonte tiver sido alterado e ainda não tiver sido gravado, a janela da Figura 15 surge a fim de o possibilitar. Caso se cancele a saída do WinMSP 1.2, ele fica no modo de edição de onde se tinha previamente partido para a execução.

### 3.2.2 A Zona Informativa

A Zona Informativa compreende:

- o campo *Instrução*: contém a linha de código equivalente à instrução em execução (actualmente apontada pelo PC);



- os campos *PC* e *SP*: mostram os valores actuais dos registos *Program Counter* e *Stack Pointer*, respectivamente;
- a janela do *Código Fonte*: compreende o texto do programa MSP cuja execução se pretende observar. Um toque de **rato** numa das suas linhas provoca a selecção do correspondente código gerado, na janela da *Memória de Programa*, permitindo confirmar a equivalência das duas representações (o texto e o correspondente executável). Um duplo toque de **rato** permite associar (ou desassociar) um *breakpoint* a uma linha de código (e por equivalência, à correspondente instrução na janela da *Memória de Programa*).  
Durante a execução do programa, a linha equivalente à instrução em execução é evidenciada.
- a janela da *Memória de Programa*: contém a imagem executável do programa descrito na janela do *Código Fonte*. A informação reparte-se por quatro colunas e segue o sistema decimal de numeração.  
A primeira coluna (*PC*) diz qual o endereço da Memória de Programa que foi atribuído a uma determinada instrução; esse endereço é o valor que o *PC* há-de assumir quando essa instrução estiver a ser executada.  
A segunda (*Cod*) tem o valor do código inteiro da instrução (o seu *opcode*), tal como definido em **3.3.2 Zona de Código - Conjunto de Instruções**, no Manual da Linguagem.  
A terceira (*Arg1*) e quarta (*Arg2*) colunas apresentam os eventuais argumentos da instrução. Um argumento de 8 *bits* (um valor na gama [0,255]) usa apenas a terceira coluna. Um argumento de 16 *bits* (em geral um endereço na gama [0,31999]) distribui-se pelas duas colunas (com a parte menos significativa associada a *Arg1* e a mais significativa a *Arg2*).  
Um toque de **rato** numa linha desta janela revela a associação com a linha respectiva da janela do *Código Fonte*. Também à semelhança do que se passa na janela do *Código Fonte*, durante a execução do programa, a linha equivalente à instrução em execução é evidenciada.
- a janela da *Memória de Dados*: apresenta as zonas da Memória de Dados que foram reservadas pelas variáveis do programa.  
A informação desta janela distribui-se por três colunas, usando o sistema decimal. A primeira coluna (*End*) contém os endereços da zona alocada pela variável. A segunda (*Valor*) mostra o(s) valor(es) actual(ais) da variável e a terceira (*Variável*) apresenta o identificador da variável.  
Uma linha vazia separa a informação associada a diferentes variáveis. Sempre que durante a execução do programa ocorre um acesso (de escrita ou leitura) a uma variável, a linha correspondente da janela da *Memória de Dados* é evidenciada<sup>18</sup>.  
É possível aceder a zonas da Memória de Dados que não estejam associadas a nenhuma variável. Basta manipular convenientemente o conteúdo da *Stack*, construindo os endereços desejados e acedendo-lhes. Porém, esses acessos não são visíveis na janela da *Memória de Dados* pois só as zonas alocadas pelas variáveis declaradas no programa é que são contempladas nessa janela.

---

<sup>18</sup>nota: o último acesso à Memória de Dados permanece em evidência.

- a janela da *Stack*: mostra o comportamento dinâmico da *Stack* sempre que se introduz ou retira um valor. A representação dos valores empilhados na *Stack* distingue entre componentes de endereço (de fundo verde) e valores que não foram assim interpretados (de fundo branco). A cada valor aparece ainda associado o seu endereço em termos da *Stack*.
- a janela de *Output*: simula um terminal de 80 colunas por 25 linhas. Para este terminal são despejados os valores que as instruções *OUT* e *OUTC* retiram da *Stack*. Por defeito, essas instruções não provocam mudança de linha. Uma forma de forçar essa mudança de linha é usando o carácter de código *ascii* 10 (*newline*). Por exemplo, o programa:

```

MEMORIA DE DADOS
CODIGO
PUSH 10 ; mete o caracter newline na Stack
PUSH 1
OUT
OUTC ; muda de linha
PUSH 2
OUT
HALT

```

escreve primeiro o número 1, muda de linha e depois escreve o número 2.

### 3.3 Input durante a execução - IN e INC

Os programas MSP recebem dados do exterior através das instruções *IN* e *INC*.

No caso do *IN*, a janela usada para introduzir uma valor de 8 *bits* é a seguinte:

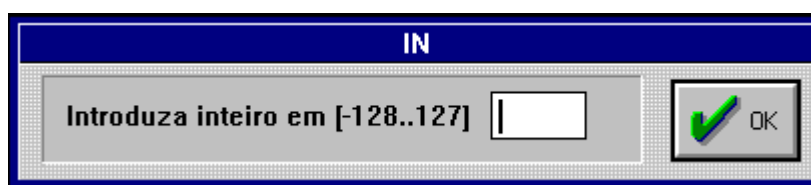


Figura 28

Qualquer valor fora da gama [-128,127] não é aceite pois o *IN* assume que esse valor será argumento de alguma operação aritmética.<sup>19</sup>

Para o *INC*, o *interface* é fornecido pela janela da Figura 29.

<sup>19</sup>poderá eventualmente ser usado como componente de endereço, mas então será o utilizador a ter presente que para introduzir valores na gama [128,255] terá que fazer as conversões necessárias para a gama [-128,-1].

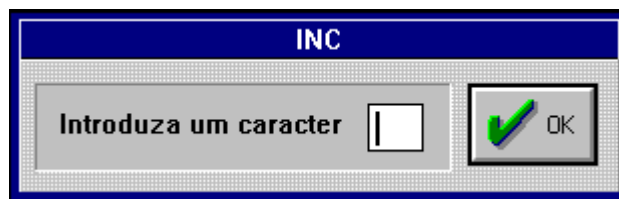


Figura 29

### 3.4 Erros de Execução

Neste capítulo são apresentadas e discutidas diversas situações que originam erros durante a execução de um programa MSP, sobre o ambiente WinMSP 1.2.

Sempre que ocorre um erro de execução, esta é automaticamente interrompida e surge a janela da Figura 30<sup>20</sup> com um sumário do erro.

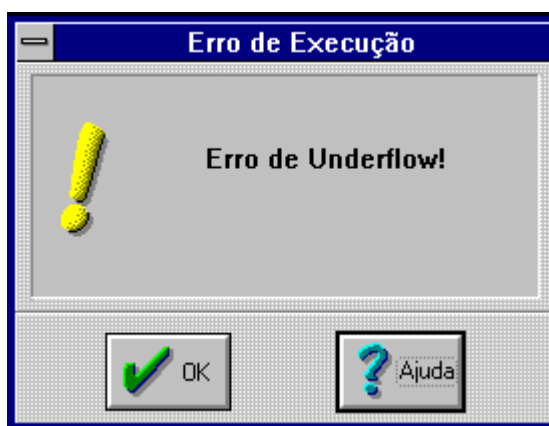


Figura 30

Pretendendo uma descrição mais detalhada, pressiona-se o botão de **Ajuda**. A fim de voltar a executar o programa é necessário pressionar **Recomeçar** na Barra de Opções, depois de sair da janela do Erro de Execução.

Para cada erro, a informação que se segue fornece um código, uma descrição sumária, uma descrição desenvolvida e, quando se ache conveniente, um exemplo de uma situação que provoca o erro em questão.

Código: **0**

Sumário: Divisão por zero!

Descrição: Tentativa de realizar uma divisão em que o divisor é zero. Verifique o código.

Exemplo: desnecessário.

---

<sup>20</sup>específica, neste caso para o Erro de Underflow

**Código: 1****Sumário:** Erro de Underflow!**Descrição:** Tentativa de realizar uma operação cujo resultado é demasiado pequeno para armazenar.**Exemplo:**

```
MEMORIA DE DADOS
CODIGO
PUSH -100
PUSH -100
ADD ; (-100)+(-100)=-200, mas para as operações aritméticas
      ; argumentos e resultados têm que estar na gama [-127,128]
```

**Código: 2****Sumário:** Erro de Overflow!**Descrição:** Tentativa de realizar uma operação cujo resultado é demasiado grande para armazenar.**Exemplo:**

```
MEMORIA DE DADOS
CODIGO
PUSH 100
PUSH 100
ADD ; (100)+(100)=200, mas para as operações aritméticas
      ; argumentos e resultados têm que estar na gama [-127,128]
```

**Código: 3****Sumário:** Stack cheia!**Descrição:** Tentativa de introduzir um valor sobre uma pilha cheia. Ou a pilha estava cheia antes de iniciar a operação, ou ficou cheia antes de a completar.**Exemplo:** Esta situação é muito rara em termos práticos e por isso não se fornece exemplo.**Código: 4****Sumário:** Stack vazia!**Descrição:** Tentativa de extrair um valor de uma pilha vazia. Ou a pilha estava vazia antes de iniciar a operação, ou então não conseguiu fornecer os argumentos necessários à instrução e ficou vazia durante a sua tentativa de extracção.**Exemplo:**

```
MEMORIA DE DADOS
CODIGO
LOAD ; a Stack ainda está vazia; por isso não consegue fornecer
      ; os componentes de endereço que LOAD precisa
```

**Código: 5****Sumário:** Violação do Segmento de Dados!**Descrição:** Tentativa de aceder à Memória de Dados através de um endereço que não pertence ao seu espaço de endereçamento [0..31999]. Qualquer instrução que constrói um endereço a partir da Stack está sujeita a este erro se os

componentes  $Msb^{21}$  e  $Lsb^{22}$  forem tal que  $Msb*256 + Lsb$  não pertence a  $[0..31999]$ .

Exemplo:

MEMORIA DE DADOS

CODIGO

PUSH 255

PUSH 255

LOAD ; tentativa de carregar o conteúdo do endereço  $255*256+255=$   
; 65535, que não pertence à gama válida  $[0,31999]$

---

<sup>21</sup>*Most significant byte* (byte mais significativo)

<sup>22</sup>*Less significant byte* (byte menos significativo)