Montador MASM611 & CodeView

Prof. Sérgio L. Cechin

.inf INSTITUTO DE INFORMÁTICA UFRGS

Introdução

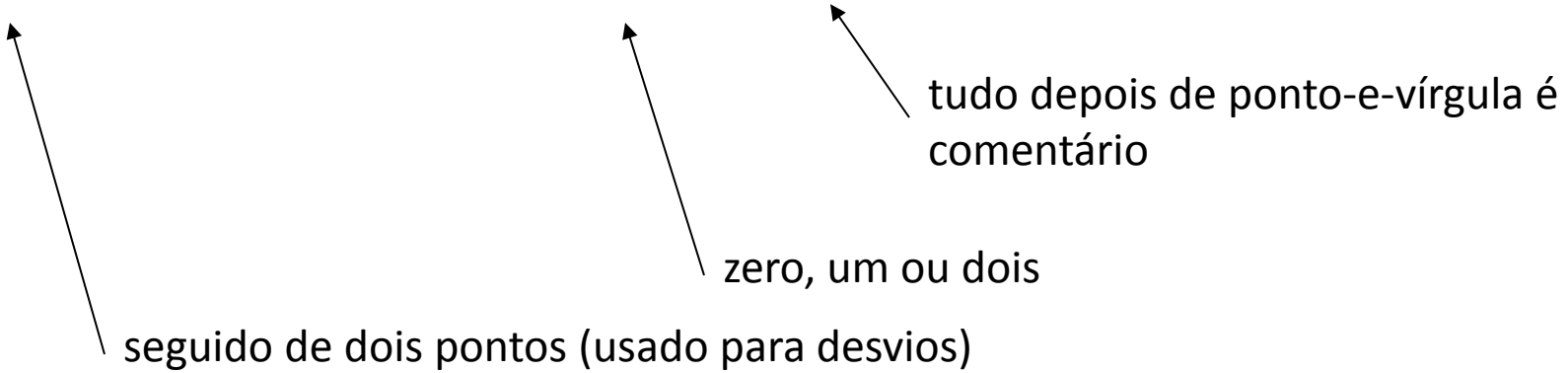
Passos Principais

- Escrever o programa fonte (xx.ASM)
  - Usar o seu editor de texto preferido
- Utilizar o montador (MASM)
  - Geração de código objeto (xx.OBJ)
- Utilizar o carregador (LINK)
  - Geração de código executável (xx.EXE)
- Utilizar o depurador (CodeView, Debug)

Formatos de linha

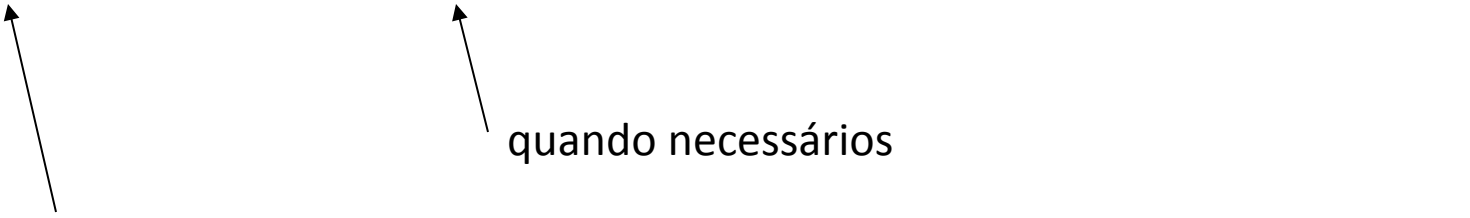
- Linha de instrução

  rótulo: prefixo instrução operandos ; comentários



- Linha de diretiva

  nome diretiva operandos ; comentários



Nomes de variáveis

- Letras, dígitos e os símbolos “**@**”, “**_**” e “**$**”
- Nome deve iniciar por letra (não pode ser dígito)
  - Recomenda-se não usar “**@**” e “**_**”
- Palavras reservadas não devem ser usadas
  - Mnemônicos, Nomes de diretivas, Nomes de registradores
- Sem limite de comprimento
  - Mas o montador considera somente os primeiros 31 caracteres

Números

- Decimal
  - Sistema default
  - Dígitos de 0 a 9
- Binário
  - Dígitos 0 e 1
  - Terminado por B
- Hexadecimal
  - Dígitos 0 a 9, letras A a F
  - Deve iniciar por dígito (usar 0 se iniciar por letra)
  - Terminado por H

Definição de Constantes

- Diretiva EQU
  - Associa um nome a um número
  - Não utiliza espaço de memória
- Exemplos
  - MAXIMO EQU 32
  - MENOS_UM EQU 0FFH
  - ZERO EQU 0
  - SETE EQU 0111B

Definição de Espaço para Variáveis

- Diretivas DB, DW, DD, DQ
  - Reserva um *Byte*, *Word*, *DoubleWord* ou *QuadWord*
  - Reserva espaço em memória
- Exemplos

```
VAR1    DB 5     ; reserva um byte com o nome de VAR1 e inicializa com 5
VAR2    DW 0FH   ; reserva palavra com o nome de VAR2 e inicializa com 15
VAR3    DW ?     ; reserva uma palavra com o nome de VAR3 e não inicializa
AB      DB 'AB'  ; string armazenado como 41H seguido de 42H
BA      DW 'AB'  ; string armazenado como 42H seguido de 41H
END$AB  DW AB    ; inicializa com offset da variável AB
```

Definição de Espaço para Variáveis

- Exemplos

```
TBL1    DW 6 DUP(0)       ; reserva seis palavras
                          ; a 1ª apontada por TBL1 e
                          ; inicializa todas palavras com 0
TBL2    DB 12 DUP(?)      ; reserva doze bytes
                          ; o 1º apontado por TBL2 e
                          ; não inicializa
NUM     DW 1234H          ; armazena 34H em NUM e 12H em NUM+1
        DB 0              ; reserva um byte sem nome com valor 0
PILHA   DW 1024 DUP (?) ; reserva espaço para 1024 palavras
```

Definição de Espaço para Variáveis

- Exemplos

```
DIGIT           DB '0123456789'          ; dez bytes alocados
SINGLE$QUOTE    DB ''''                  ; um byte alocado (com aspas)
PRIMES          DW 2,3,5,7,11,13,17      ; sete palavras alocadas
MSG             DB 'Meu primeiro programa Assembler', 0DH, 0AH
```

- Strings podem ter até 255 caracteres
- Listas (DB seguido de valores) podem ter até 16 elementos

Definição de Nomes (Instruções)

- Diretiva **LABEL**
  - Usada para identificar (os endereços) as instruções
- Tipos

  LABEL NEAR (acesso de dentro do segmento)

  LABEL FAR (acesso a partir de outros segmentos)
- Exemplo 1 (label NEAR)
  - Forma explícita

```
SOMA_VETOR      LABEL    NEAR
                ADD      AX,VETOR[BX]
```

  - Forma abreviada

```
SOMA_VETOR:     ADD      AX,VETOR[BX]
```

- Exemplo 2 (mais de um rótulo por instrução)

```
SOMA_VETOR_EXT                LABEL    FAR
SOMA_VETOR:                   ADD      AX,VETOR[BX]
```

Definição de Nomes (Operandos)

- Diretiva **LABEL**
  - Usada para identificar o endereço do operando
  - Define o tipo do operando (BYTE, WORD, DWORD, etc)
- Exemplo 1

```
ARRAYW      LABEL   WORD
            DW      1000 DUP(0)
```

  - Forma abreviada

```
ARRAYW  DW      1000 DUP(0)
```

- Exemplo 2 (mais de um rótulo por operando)

```
ARRAYB              LABEL   BYTE
ARRAYW      DW      1000 DUP(0)
```

  - Isso é necessário para possibilitar acessar o *array* byte-a-byte ou word-a-word

Acessando características de símbolos

- **OFFSET** – fornece o deslocamento do símbolo dentro do segmento

  MOV BX, OFFSET VAR

- **SEG** – fornece o segmento do símbolo

  MOV BX, SEG VAR

- **PTR** – altera o tipo de um símbolo.
  - Exemplo: se VAR foi definida como WORD, a instrução

    INC BYTE PTR VAR

  - acessa VAR como um byte (usa apenas o byte menos significativo)

Estrutura dos Programas

- Os programas são formados por módulos (arquivos) que, por sua vez, são formados de segmentos
- Todo o programa possui um módulo principal, onde a execução do programa inicia
  - Esse módulo pode conter segmentos de **código, dados e pilha**
  - Os outros módulos só podem conter segmentos de **código e dados**
- Existem duas formas de declarar os segmentos
  - Modelo Completo
  - Modelo Simplificado

Manipulação de Segmentos
Modelo Simplificado

Introdução

- No Modelo Simplificado são usadas as diretivas “.” (ponto)
- As diretivas podem:
  - Declarar parâmetros (ex: tamanho da pilha)
  - Iniciar uma sessão de declarações (ex: segmento de código)
- Nesse modelo, o módulo principal deve apresentar a seguinte estrutura:
  - .MODEL
  - .STACK
  - .DATA
  - .CODE
  - .STARTUP
  - .EXIT
  - END
- Notar o “END” (sem ponto) no final do programa.
  - É necessário para o MASM encontrar o final do módulo

.MODEL

- Deve ser a primeira diretiva no módulo, antes de qualquer outra
- Define os atributos que afetam todo o módulo
  - Modelo de memória
  - Convenções de chamada e de nomes
  - Sistema operacional
  - Tipo de pilha
- Essa diretiva possibilita o uso simplificado
  - Dos segmentos
  - Dos nomes
  - Da forma de chamada das subrotinas
- Formato: .MODEL *memorymodel* [[, *modeloptions*]]

Parâmetros de ".MODEL"

- Modelos de Memória
  - TINY, SMALL, COMPACT, MEDIUM, LARGE, HUGE, FLAT
  - Determinam o tamanho dos ponteiros de código e dados (NEAR ou FAR)
- Opções
  - Especificação da linguagem. Usado para ligar módulos assembler com módulos em linguagem de alto nível.
    - C, BASIC, FORTRAN, PASCAL, SYSCALL, ou STDCALL
  - Localização do segmento de pilha
    - NEARSTACK, indica que o segmento de dados e o da pilha são o mesmo segmento físico (chamado de DGROUP)
    - FARSTACK, indica que dados e pilha estão em segmentos diferentes

Escolha do Modelo de Memória

Memory Model	Default Code	Default Data	Operating System	Data and Code Combined
Tiny	Near	Near	MS-DOS	Yes
Small	Near	Near	MS-DOS, Windows	No
Medium	Far	Near	MS-DOS, Windows	No
Compact	Near	Far	MS-DOS, Windows	No
Large	Far	Far	MS-DOS, Windows	No
Huge	Far	Far	MS-DOS, Windows	No
Flat	Near	Near	Windows NT	Yes

Small: 1 segmento de código (NEAR) e 1 segmento de dados (NEAR)
Large: "n" segmentos de código (FAR) e "m" segmentos de dados (FAR)
Medium: "n" segmentos de código (FAR) e 1 segmento de dados (NEAR)
Compact: 1 segmento de código (NEAR) e "m" segmentos de dados (FAR)

Small, Medium, Compact, Large, Huge

- Small
  - 1 segmento de código (NEAR) e 1 segmento de dados (NEAR)
- Medium
  - “n” segmentos de código (FAR) e 1 segmento de dados (NEAR)
- Compact
  - 1 segmento de código (NEAR) e “m” segmentos de dados (FAR)
- Large
  - “n” segmentos de código (FAR) e “m” segmentos de dados (FAR)
- Huge
  - Semelhante ao Large, porém suporta itens de dados maiores do que 1 segmento.
- Os ponteiros podem ter tamanhos diferentes dos “default”, desde que especificados pelas diretivas adequadas (NEAR e FAR)

Tiny e Flat

- Tiny
  - Roda apenas no MS-DOS (portanto, também no DosBox)
    - São os programas do tipo “.com” (e não “.exe”)
  - Código, dados e pilha são colocados em um único segmento físico
    - Portanto, os programas não podem ter mais de 64K bytes
  - Ponteiros para código e dados estáticos são NEAR
  - Pode-se alocar memória dinamicamente. Nesse caso, usa-se ponteiros FAR
- Flat
  - Semelhante ao Tiny, porém com um segmento de 32 bits (4G bytes)
  - Só existe a partir do 80386

.STACK

- Só pode existir no módulo principal (aquele onde está o início do programa)
- Cria (e encerra) um segmento para a pilha do programa
  - Se não tiver parâmetros, cria uma pilha com 1K byte
  - Senão, o parâmetro especifica o tamanho da pilha a ser criada
    - Exemplo: criar uma pilha com 2K bytes

      .STACK 2048

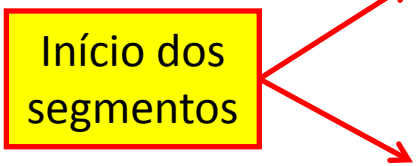
.DATA & .FARDATA

- .DATA
  - Usado para declarar o segmento “default” de dados
  - Esse segmento pode ser acessado através de ponteiros NEAR
- .FARDATA
  - Usado para declarar vários segmentos de dados
  - Esses segmentos devem ser acessados através de ponteiros FAR
    - É necessário setar o registrador de segmento adequadamente
    - Exemplo:

```
mov     ax, SEG farvar2
mov     ds, ax
mov     ax, farvar2
```

.CODE

- Usado para declarar segmentos de código
  - O segmento encerra na próxima declaração de segmento
  - Podem ser NEAR ou FAR
- Os segmentos podem ter “nomes”
  - Util para declarar mais de um segmento no mesmo módulo




```
.CODE   FIRST
.
.       ; First set of instructions here
.
.CODE   SECOND
.
.       ; Second set of instructions here
.
```

.CODE

- Segmento NEAR (endereços têm 16 bits)
  - Usado nos modelos *small* e *compact*
  - Se houverem vários módulos, os segmentos .CODE de todos são agrupados em um único segmento físico
- Segmento FAR (endereços têm 32 bits)
  - Usado nos modelos *medium*, *large* e *huge*
  - Usado quando for necessário mais de 64K de código
  - Se houverem vários módulos, cada segmento .CODE originará um segmento físico

.STARTUP e .EXIT

- Usado dentro do segmento de código do módulo principal
- .STARTUP
  - Indica o ponto de entrada do programa
  - É o endereço chamado pelo S.O. para iniciar o programa
- .EXIT
  - Indica o final do programa
  - É o ponto em que a execução retorna para o S.O.
  - O código gerado por essa diretiva é específico para o MS-DOS
    - Não pode ser usado para outros S.O.

Exemplo

- O local do .STARTUP indica o início do programa
- O local do .EXIT indica onde o programa encerra
  - Essa diretiva gera o código de retorno para o S.O.
- O END apenas indica que o fonte terminou
  - Essa diretiva é uma indicação para o montador
  - Ela não gera código

```
.CODE
.STARTUP
.
.                    ; Place executable code here
.
.EXIT
END
```

Programa Exemplo (Início, Dados e Pilha)

```
.model   small
.stack   1024
.data

MENSAGEM    DB ‘Hello World !’,0DH,0AH
TAMANHO     EQU $-MENSAGEM
CONTADOR    DB ?
```

Programa Exemplo (Programa Principal)

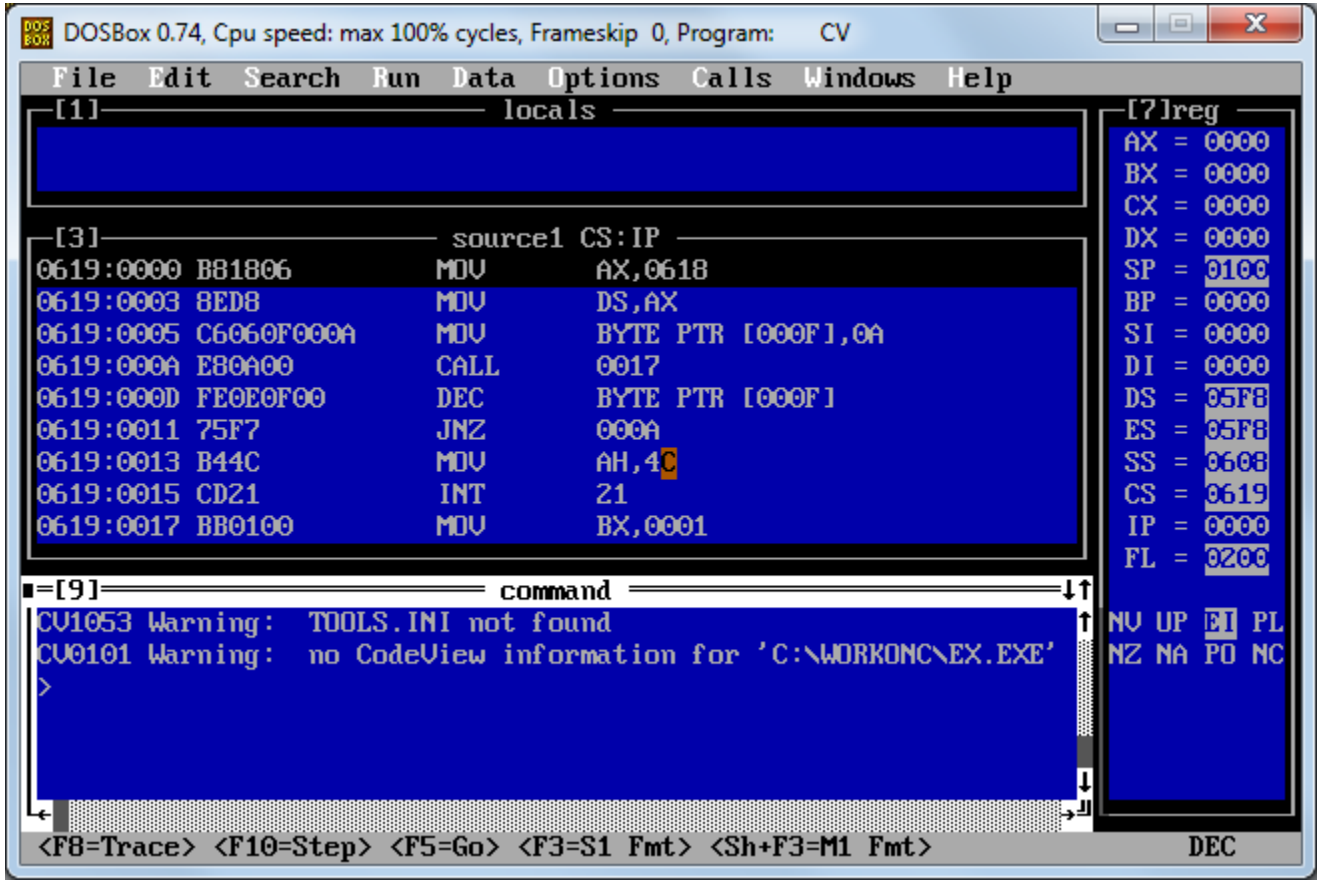
```
                .code
                .startup

                MOV     CONTADOR, 10
DE_NOVO:
                CALL    FRASE
                DEC     CONTADOR
                JNZ     DE_NOVO

                .exit   0                   ; Retorna ao DOS
```

Programa Exemplo (Subrotina)

```
FRASE       PROC    NEAR

            MOV     BX, 0001H       ; standard output
            LEA     DX, MENSAGEM
            MOV     CX, TAMANHO
            MOV     AH, 40H
            INT     21H             ; Escreve mensagem
            RET

FRASE       ENDP

            END
```

CodeView

Janela de entrada

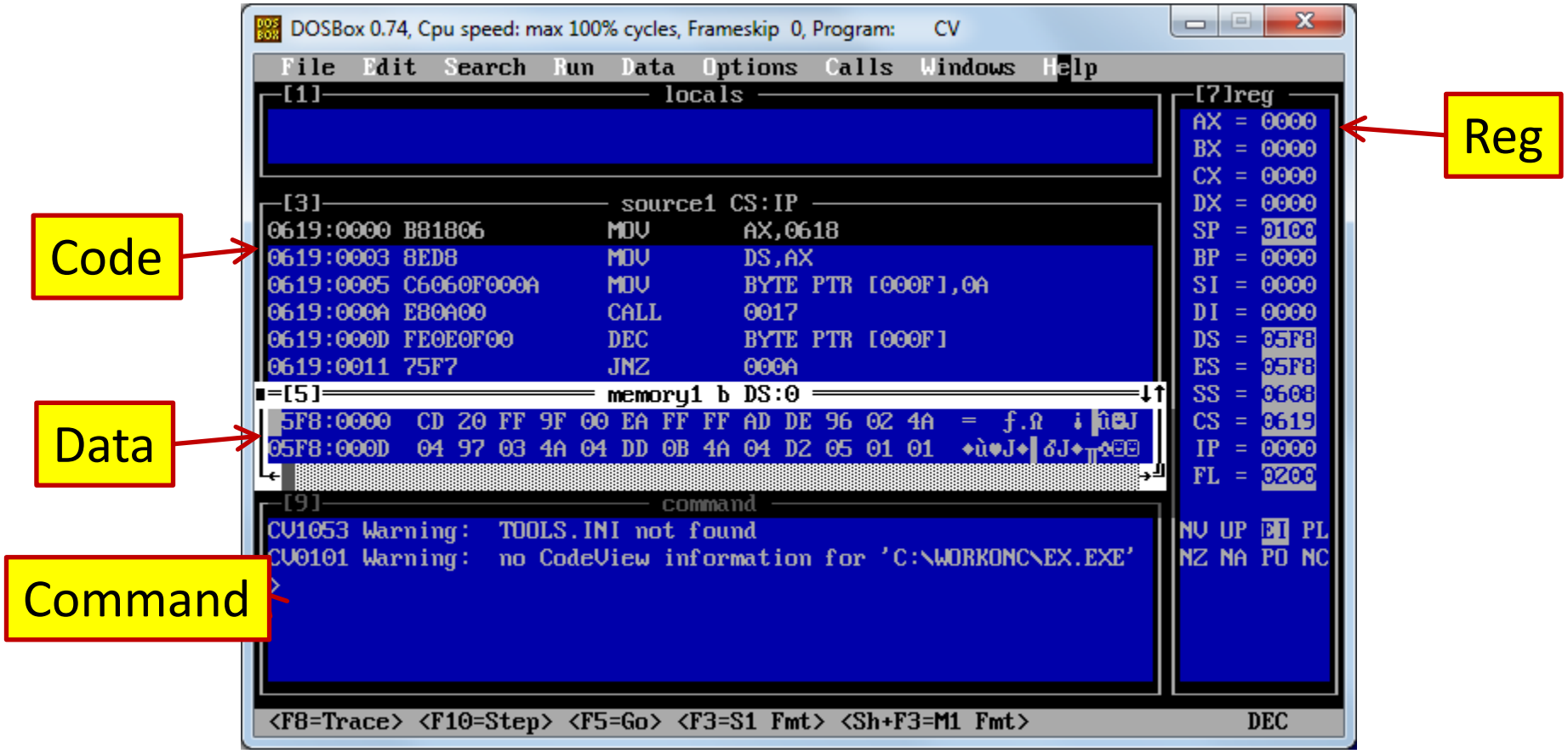


- Ambiente com múltiplas janelas
- Roda apenas do DOS (no nosso caso, no DosBox)

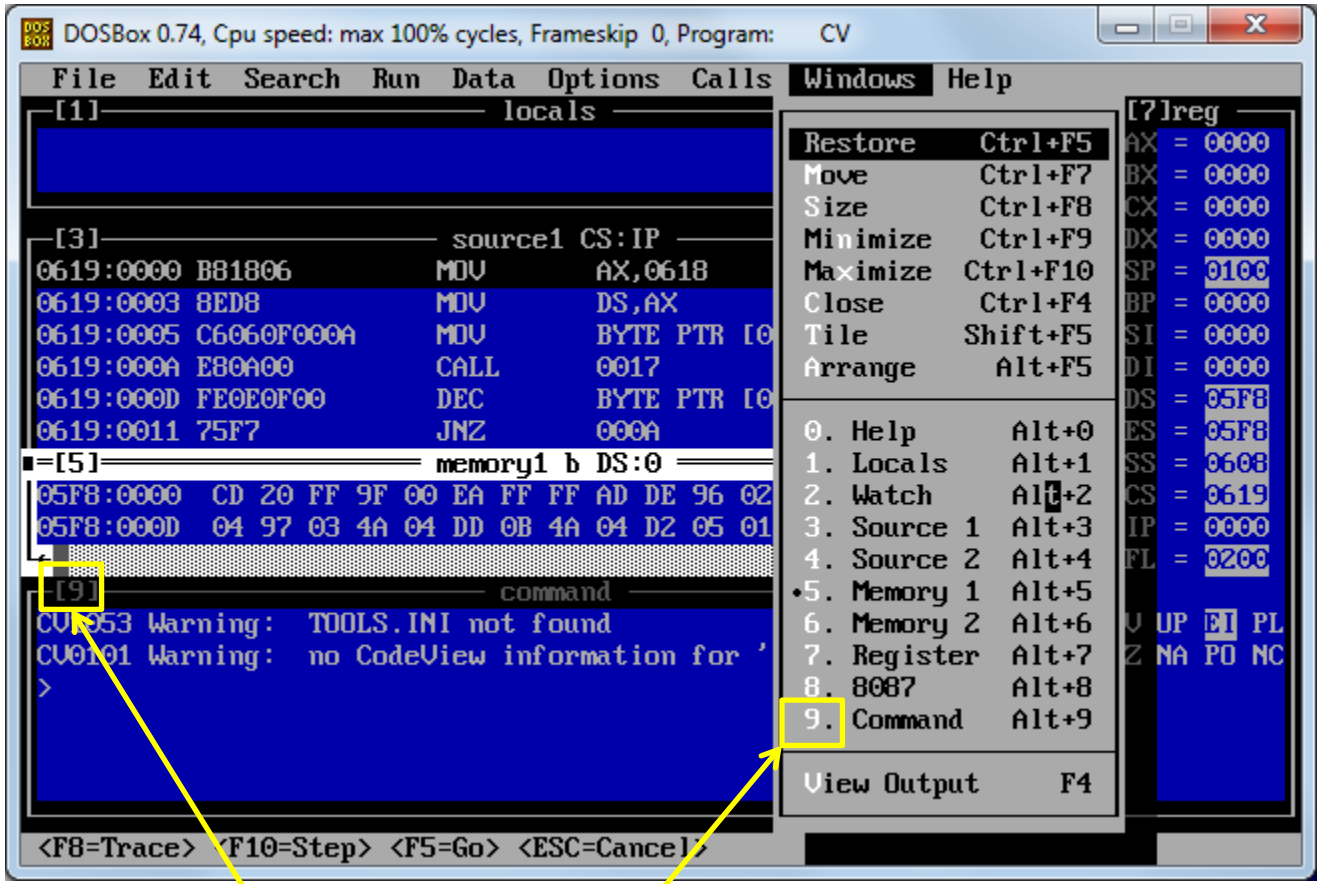
Principais janelas

- **Code** window
  - Existem duas janelas: Source1 e Source2
  - Mostra, em assembly, o conteúdo da memória
  - Código armazenado na memória para depuração
  - Se o programa foi corretamente montado e ligado, apresenta o código fonte
- **Data** window
  - Existem duas janelas: Memory1 e Memory2
  - Mostra, em formato “dump”, o conteúdo da memória
  - Para alterar o endereço, basta digitar sobre o endereço apresentado
- Watch window
  - Apresenta o conteúdo das variáveis desejadas
- **Locals** window
  - Apresenta o conteúdo das variáveis locais da função em que se está executando
- **Command** window
  - Janela para o usuário digitar comandos para o CodeView
- **Register** window
  - Permite visualizar o conteúdo dos registradores do 80x86
  - Inclui os registradores de uso geral, segmento, flags e FPU (Floating Point Unit)

Localização de algumas janelas



Ativação das janelas



Comandos de exame/alteração

- Pode-se informar endereços de várias formas
  - Decimal: **100**
  - Hexadecimal: **0x100**
  - Com segmento: **0x619:**0x100, **DS:**0x100, **CS:**100, etc
- Alguns comandos
  - Dump
    - d *end*
    - Lista o conteúdo da memória a partir do endereço "*end*"
    - Pode ser seguido por qualificadores: B, W, I, IU, A
  - Assembly
    - a *end*
    - Monta as instruções digitadas a partir do endereço "*end*"
  - Unassembly
    - u *end*
    - Desmonta os bytes a partir do endereço "*end*" em mnemônicos
  - Enter
    - e *end*
    - Entra valores na memória, no endereço "*end*"
    - Pode ser seguido por qualificadores: B, W, I, IU, A
    - ENTER encerra a entrada; SPACE passa para o próximo

Comandos de execução

- São os comandos usados para simular os programas
- ***Trace*** – "T" (ou ***<F8>***): executa uma única instrução
- ***Procedure trace*** – "P" (ou ***<F10>***): se for uma instrução CALL, executa toda a subrotina chamada
  - Se for outra instrução, comporta-se como um trace
- ***Go*** – "G" (ou ***<F5>***): executa o programa a partir do endereço fornecido
  - Se não tiver endereço, executa a partir do IP
- ***Execute*** – "E": executa o programa passo a passo até encerrar
  - Atualiza a tela entre a execução de uma instrução e outra
- ***Restart*** – "L": *reset* o programa para o início

Montador MASM611 & CodeView

Prof. Sérgio L. Cechin

.inf INSTITUTO DE INFORMÁTICA UFRGS