Montador para Intel

TASM - Turbo Assembler

Passos Principais

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

Formatos de linha

- Linha de instrução

  rótulo: prefixo instrução operandos ; comentários

  - rótulo: seguido de dois pontos (usado para desvios)
  - prefixo: zero, um ou dois
  - ; comentários: tudo depois de ponto-e-vírgula é comentário
- Linha de diretiva

  nome diretiva operandos ; comentários

  - nome: sem dois pontos (usado para definir operandos, segmentos e *procedures*)
  - operandos: quando necessários

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

  exemplos: 10, 1010
- Binário
  - Dígitos 0 e 1
  - Terminado por B

  exemplos: 10B, 1010B
- Hexadecimal
  - Dígitos 0 a 9, letras A a F
  - Deve iniciar por dígito (usar 0 se iniciar por letra)
  - Terminado por H

  exemplos: 10H, 1010H, 0ABCH

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 4142H
BA      DW 'AB'     ; string armazenado como 4241H
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 com o nome de TBL1 e
                        ; inicializa todas com 0
TBL2    DB 12 DUP(?)    ; reserva doze bytes
                        ; o 1º com o nome de 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
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 podem ter até 16 elementos

Definição de Nomes (Rótulos)

- Diretiva LABEL
- Para identificar instruções

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

- Forma abreviada:

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

- São possíveis dois ou mais rótulos:

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

Definição de Nomes (Rótulos)

- Diretiva LABEL
- Para identificar operandos

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

- Forma abreviada:

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

- São possíveis dois ou mais rótulos:

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

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. Por exemplo, se VAR foi definida como palavra, a instrução

    INC BYTE PTR VAR
  - acessa VAR como um byte

Declaração, associação e inicialização de segmentos

- para fazer uso adequado dos segmentos de 64Kbytes na arquitetura x86 no modo real ou virtual 86, :
  - os segmentos precisam ser declarados
  - os segmentos devem ser associados a registradores de segmento
  - os registradores de segmento precisam ser inicializados com valores adequados
- a forma de fazer declaração, associação e inicialização depende do montador

Declaração de segmentos

- Diretiva: SEGMENT
- Formato:

  [seg_name] SEGMENT [param. opcionais]
  ……
  ……..
  [seg_name] ENDS

```
MEUSDADOS   SEGMENT
            ...
            ...
MEUSDADOS   ENDS
```

Associação de segmentos

- Função:
  - associa segmentos lógicos (criados através do montador) com segmentos físicos (endereçados por CS, DS, SS, ES)
- Diretiva:

  ASSUME
- Formato:

  ASSUME seg_reg:seg_name [, ...]

```
MEUSDADOS   SEGMENT
            ...
MEUSDADOS   ENDS
            ...
ASSUME      DS:MEUSDADOS
```

Inicialização de segmentos

- Inicialização
  - por instruções assembler (ou seja por código executável)
- Função:
  - carrega os valores adequados nos registradores de segmento
  - CS inicializado pelo sistema operacional
  - DS, SS, ES devem ser inicializados pelo programa

Declaração, associação e inicialização de segmentos

- Exemplo

```
ARRAYS      SEGMENT
AR1     DW 100 DUP(0)  ; reserva 100 palavras com valor zero
AR2     DW 500 DUP(0)  ; reserva 500 palavras com valor zero
ARRAYS      ENDS

SUM     SEGMENT
        ASSUME    CS:SUM, DS:ARRAYS
START:  MOV     AX, ARRAYS
        MOV     DS, AX
        ……..
        ……..
SUM     ENDS
```

associação

inicialização

lembrar que reg segmento não pode receber dado imediato

Declaração, associação e inicialização de segmentos

- Exemplo

```
STK1        SEGMENT
            DW      1000 DUP(0) ; define 1000 palavras (2000 bytes)
STK1        ENDS

STK_INIT    SEGMENT
            ASSUME          CS:STK_INIT, SS:STK1
            MOV     AX,STK1
            MOV     SS,AX
            MOV     SP,2000  ; primeiro PUSH coloca SP em 1998
            ……
            ……
STK_INIT    ENDS
```

```
DATA    SEGMENT
VAR1    DB      0
DATA    ENDS

DATA2   SEGMENT
VAR2    DB      0
DATA2   ENDS

CODE    SEGMENT
   ASSUME       CS:CODE, DS:DATA, ES:DATA2
   MOV  AX,DATA
   MOV  DS,AX
   MOV  AX,DATA2
   MOV  ES,AX
   MOV  VAR1,99                 ; VAR1 endereçado por DS
   MOV  VAR2,99                 ; VAR2 endereçado por ES
CODE    ENDS
```

Declaração de subrotinas (procedures)

- Diretivas:
  - PROC (início de subrotina)
  - ENDP (fim de subrotina)
- Formato:
  - name PROC [NEAR / FAR]
  - name ENDP
- Função:
  - PROC e ENDP delimitam um bloco que contém uma subrotina

Subrotina: exemplo

Programa chama uma subrotina (FRASE) para escrever mensagem na tela

```
PILHA   SEGMENT         STACK
        DB 32 DUP ('STACK---')
PILHA   ENDS

DADOS           SEGMENT
MENSAGEM        DB 'Hello World !',0DH,0AH
TAMANHO         EQU $-MENSAGEM
CONTADOR        DB ?
DADOS           ENDS
```

parâmetro opcional

```
CODIGO  SEGMENT
        ASSUME CS:CODIGO,SS:PILHA,DS:DADOS
START:  MOV AX,DADOS            ; Inicializa segmento de dados
        MOV DS,AX
        MOV CONTADOR,10
DE_NOVO: CALL FRASE
        DEC CONTADOR
        JNZ DE_NOVO
        MOV AH, 4CH             ; Retorna ao DOS
        INT 21H
FRASE   PROC NEAR
        MOV BX,0001H
        LEA DX, MENSAGEM
        MOV CX, TAMANHO
        MOV AH,40H
        INT 21H                 ; Escreve mensagem
        RET
FRASE   ENDP
CODIGO  ENDS
        END START
```

Execução do exemplo

- Escrever o programa fonte (teste.ASM)
- Chavear para modo comando DOS (cmd)
  - Iniciar -> Executar -> cmd
  - Iniciar -> Programas -> Acessórios -> Prompt de comando
- Achar o diretório correto (cd)
- Chamar o montador (TASM)
  - tasm teste
    - Geração de código objeto (teste.OBJ)
- Utilizar o carregador (TLINK)
  - tlink teste
    - Geração de código executável (teste.EXE)
- Executar programa

Execução do exemplo

- Em caso de erro do montador
  - Montador somente indica número da linha
  - Correção deve ser feita em um editor
  - Para facilitar localização dos erros:
    tasm /l <nome>
  - gera <nome>.lst (listagem completa da montagem)
  - não corrigir <nome>.lst !!
- Em caso de erro do linker
  - Verifique se montou sem erros
  - Verifique se declarou segmento de pilha

Execução do exemplo

- Em caso de erro de execução
- Escrever mensagens de depuração é complicado....
- Revisar a lógica no papel é ineficiente...
- Utilizar o depurador !!
  - CodeView, Debug, TD
  - td teste
  - É bom ter a mão a listagem da montagem