- **O que entende por linguagem *assembly*?**

  i. **Apresenta um nível de abstração intermédia entre os dois extremos: linguagem máquina e linguagem alto-nível**

  ii. **Facilita a programação pela substituição do código binário da linguagem máquina com símbolos**

  Escrita usando *labels*(etiquetas), mnemónicas, e.t.c.

  - **Um programa em *assembly* não é executável**

  Todos os símbolos tais como mnemónicas, etiquetas devem ser traduzidos para código binário

Estrutura do programa e programação

Assembly com 8051

- **Como é que se processa a tradução de um programa *assembly* para programa máquina ?**

  Dependendo da complexidade do ambiente de programação pode envolver várias etapas até a produção do código executável

  i. Assembler

  a. **Traduz um programa em *assembly* para programa em linguagem máquina (código objeto)**

  b. **O código objecto pode estar na forma absoluta ou forma *relocatable***

  ii. Linker

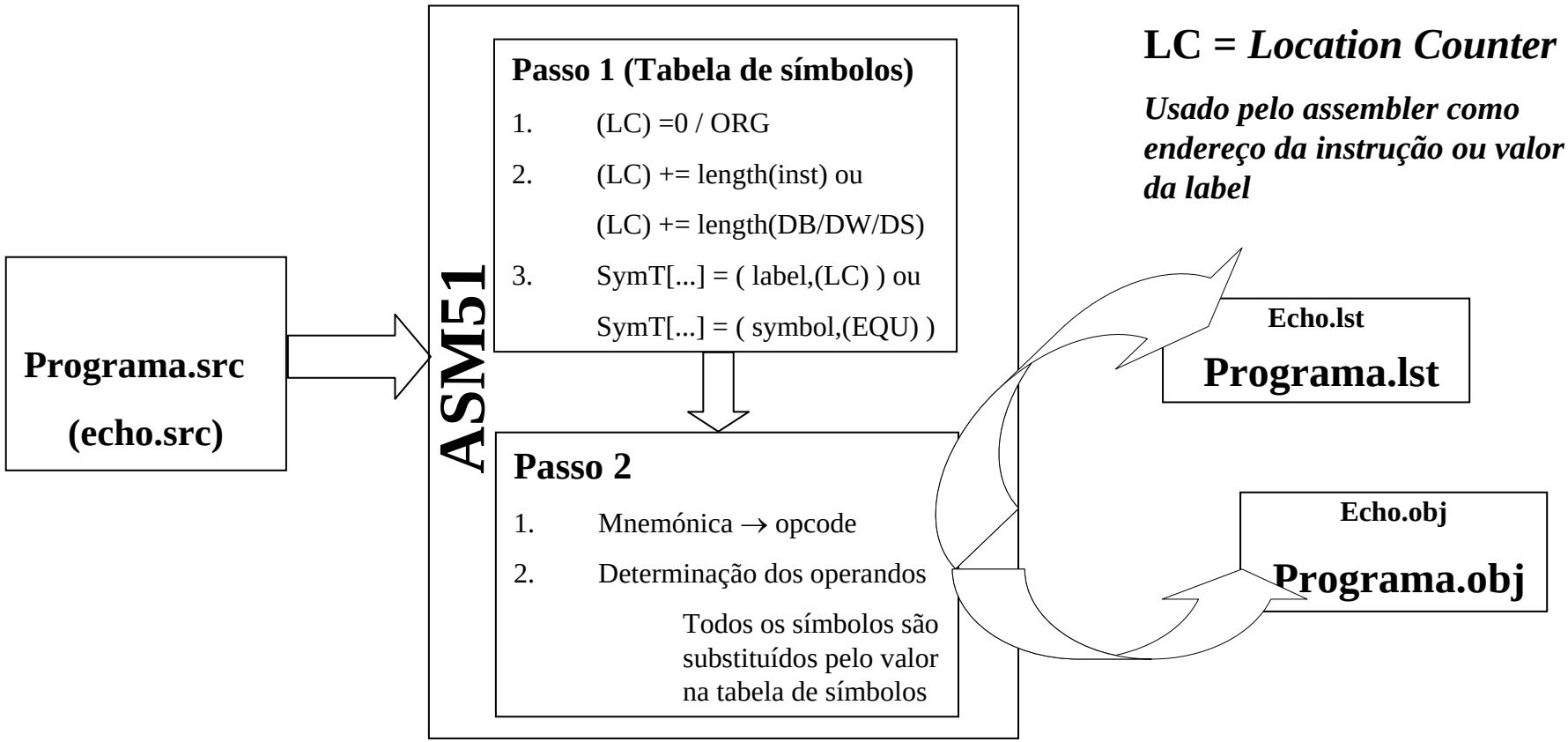
  a. **Combina vários programas objectos na forma *relocatable*, produzindo um programa executável através da atribuição de endereços absolutos**

  b. **Produz também um ficheiro contendo o mapa de memória e tabela de símbolos**

- **O que acontece ao invocar o *assembler* a partir da linha de comando ?**

Sintaxe: ASM51 ficheiro fonte [controles do *assembler*]

Exemplo: ASM51 echo.src



• Contador de localização (*Location Counter*)

- **O *assembler* possui um contador de localização para cada um dos cincos segmentos**

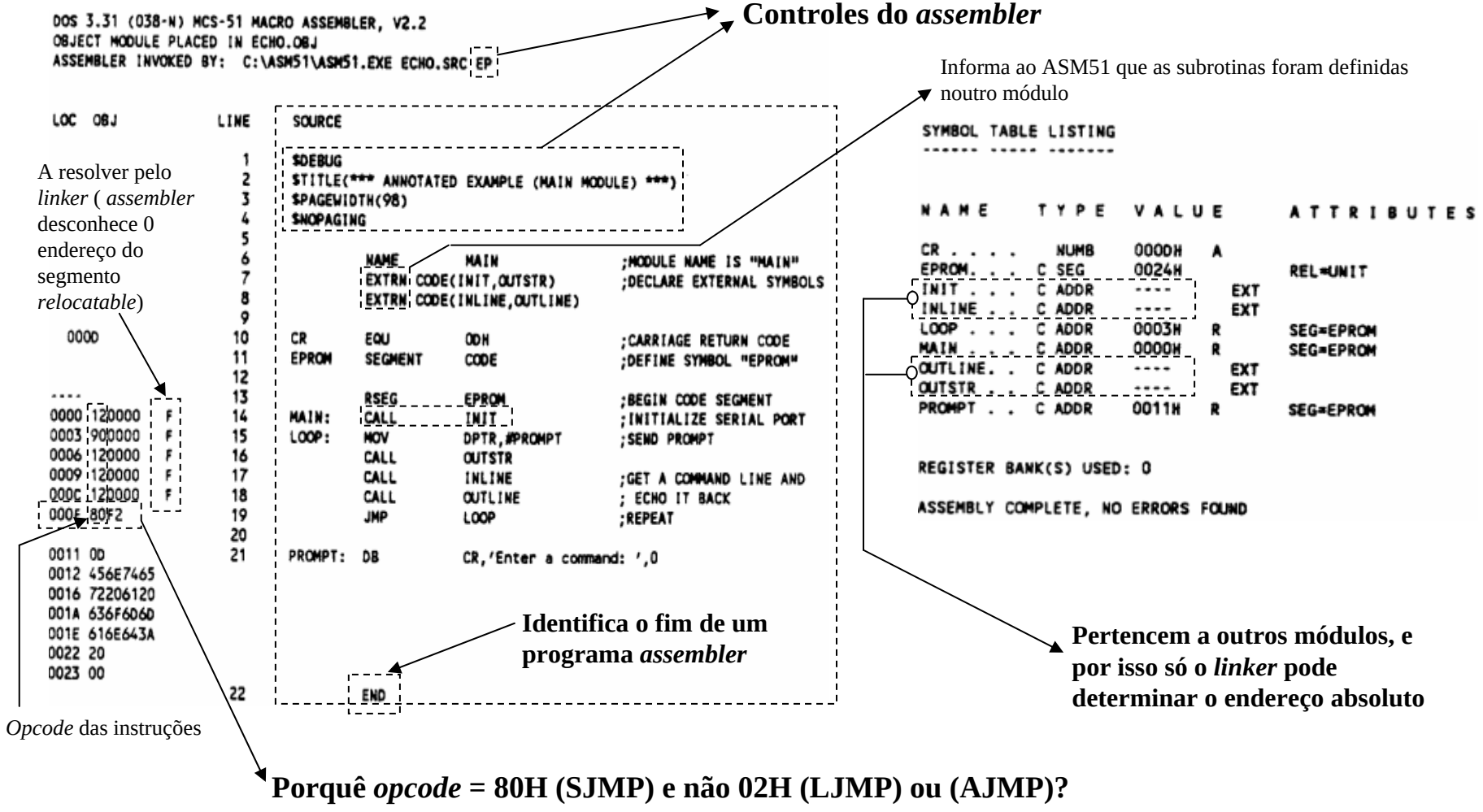
1.	**CODE**	**(0000h - FFFFH)**	**(0000h)**
2.	**DATA**	**(00H – FFH)/(00H – 7FH)**	**(30h)**
3.	**IDATA**	**(00H – FFH)/(00H – 7FH)**	**(80h)**
4.	**BIT**	**(00H –FFH)/(20H – 2FH)**	**(00h)**
5.	**XDATA**	**(0000H – FFFFH)**	**(0000h)**

- **Cada contador de localização é inicializado com o valor zero e pode posteriormente ser alterado usando as diretivas para o *assembler***

- **O símbolo $ pode ser usado para especificar o valor do contador de localização do segmento ativo:**

```
TABLE:  DB   1,2,3,4,5,6,7,8,9
LEN     EQU  $-TABLE
```

• Ficheiro .lst e tabela de <u>símbolos</u>




```
DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN ECHO.OBJ
ASSEMBLER INVOKED BY:  C:\ASM51\ASM51.EXE ECHO.SRC EP

LOC  OBJ            LINE     SOURCE

                       1     $DEBUG
                       2     $TITLE(*** ANNOTATED EXAMPLE (MAIN MODULE) ***)
                       3     $PAGEWIDTH(98)
                       4     $NOPAGING
                       5
                       6             NAME    MAIN            ;MODULE NAME IS "MAIN"
                       7             EXTRN CODE(INIT,OUTSTR)   ;DECLARE EXTERNAL SYMBOLS
                       8             EXTRN CODE(INLINE,OUTLINE)
                       9
  0000                10     CR      EQU     0DH             ;CARRIAGE RETURN CODE
                      11     EPROM   SEGMENT CODE            ;DEFINE SYMBOL "EPROM"
                      12
                      13             RSEG    EPROM           ;BEGIN CODE SEGMENT
                      14     MAIN:   CALL    INIT            ;INITIALIZE SERIAL PORT
                      15     LOOP:   MOV     DPTR,#PROMPT    ;SEND PROMPT
                      16             CALL    OUTSTR
                      17             CALL    INLINE          ;GET A COMMAND LINE AND
                      18             CALL    OUTLINE         ; ECHO IT BACK
                      19             JMP     LOOP            ;REPEAT
                      20
                      21     PROMPT: DB      CR,'Enter a command: ',0

                      22             END

SYMBOL TABLE LISTING
------ ----- -------

N A M E     T Y P E  V A L U E      A T T R I B U T E S

CR . . . .    NUMB    000DH   A
EPROM. . .  C SEG     0024H           REL=UNIT
INIT . . .  C ADDR    ----      EXT
INLINE . .  C ADDR    ----      EXT
LOOP . . .  C ADDR    0003H   R       SEG=EPROM
MAIN . . .  C ADDR    0000H   R       SEG=EPROM
OUTLINE. .  C ADDR    ----      EXT
OUTSTR . .  C ADDR    ----      EXT
PROMPT . .  C ADDR    0011H   R       SEG=EPROM

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

```
0000 120000   F
0003 900000   F
0006 120000   F
0009 120000   F
000C 120000   F
000F 80F2
0011 0D
0012 456E7465
0016 72206120
001A 636F6D60
001E 616E643A
0022 20
0023 00
```

• Saltos e chamadas genéricas

- **ASM51 permite o uso de mnemónicas genéricas JMP ou CALL em vez de SJMP, AJMP, LJMP ou ACALL, LCALL**

- **A conversão da mnemónica genérica para instrução real segue a seguinte regra:**

1. **JMP é substituído por SJMP se não for usada *forward reference* e o destino do salto estiver na gama de –128 localizações**

   ***forward reference*:** uso de um símbolo antes da respectiva definição

2. **JMP/CALL é substituído por AJMP/ACALL se não for usada *forward reference* e a instrução que segue JMP/CALL pertencer à mesma página de 2K que a instrução destino**

3. **Caso contrário, será usado a conversão para LJMP/LCALL**

r) Resposta à pergunta do slide anterior: sendo o salto para trás com deslocamento (*offset*) inferior a 128 *bytes*, aplica-se a condição 1

Estrutura do programa e programação

Assembly com 8051

- **Ficheiro .lst e tabela de símbolos**

```
MCS-51 MACRO ASSEMBLER    *** ANNOTATED EXAMPLE (SUBROUTINES MODULE) ***

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN IO.OBJ
ASSEMBLER INVOKED BY:  C:\ASM51\ASM51.EXE IO.SRC EP

LOC  OBJ              LINE     SOURCE

                         1     $DEBUG
                         2     $TITLE(*** ANNOTATED EXAMPLE (SUBROUTINES MODULE) ***)
                         3     $PAGEWIDTH(98)
                         4     $NOPAGING
                         5
                         6             NAME    SUBROUTINES         ;MODULE NAME
                         7             PUBLIC  INIT,OUTCHR,INCHAR  ;DECLARE PUBLIC SYMBOLS
                         8             PUBLIC  INLINE,OUTLINE,OUTSTR
                         9
                        10     ;*******************************************************************
                        11     ; DEFINE SYMBOLS                                                  *
                        12     ;*******************************************************************
  000D                  13     CR            EQU     0DH           ;CARRIAGE RETURN
  0028                  14     LENGTH        EQU     40            ;40-CHARACTER BUFFER
                        15     EPROM         SEGMENT CODE          ;"EPROM" IS A CODE SEGMENT
                        16     ONCHIP        SEGMENT DATA          ;"ONCHIP" IS A DATA SEGMENT
                        17
----                    18                   RSEG    EPROM         ;BEGIN RELOCATABLE CODE SEGMENT
                        19
                        20     ;*******************************************************************
                        21     ; INITIALIZE THE SERIAL PORT                                      *
                        22     ;*******************************************************************
0000 759852             23     INIT:         MOV     SCON,#52H     ;8-BIT UART MODE
0003 758920             24                   MOV     TMOD,#20H     ;TIMER 1 SUPPLIES BAUD RATE CLOCK
0006 758DF3             25                   MOV     TH1,#-13      ;2400 BAUD
0009 D28E               26                   SETB    TR1           ;START TIMER
000B 22                 27                   RET
                        28
                        29     ;*******************************************************************
                        30     ; OUTPUT CHARACTER IN ACC (NOTE: VDT MUST CONVERT CR INTO CR/LF)  *
                        31     ;*******************************************************************
000C 3099FD             32     OUTCHR:       JNB     TI,$          ;WAIT FOR TRANSMIT BUFFER EMPTY
000F C299               33                   CLR     TI            ;WHEN EMPTY, CLEAR FLAG AND
0011 F599               34                   MOV     SBUF,A        ; SEND CHARACTER
0013 22                 35                   RET
                        36
                        37     ;*******************************************************************
                        38     ; INPUT CHARACTER TO ACC                                          *
                        39     ;*******************************************************************
0014 3098FD             40     INCHAR:       JNB     RI,$          ;WAIT FOR RECEIVE BUFFER FULL
0017 C298               41                   CLR     RI            ;WHEN CHAR ARRIVES, CLEAR FLAG &
0019 E599               42                   MOV     A,SBUF        ; INPUT CHAR TO ACC
001B 22                 43                   RET
                        44
                        45     ;*******************************************************************
                        46     ; OUTPUT NULL-TERMINATED STRING                                   *
                        47     ;*******************************************************************
001C E4                 48     OUTSTR:       CLR     A             ;DPTR POINTS TO STRING OF CHAR
001D 93                 49                   MOVC    A,@A+DPTR     ;GET CHARACTER
001E 6006               50                   JZ      EXIT          ;IF NULL BYTE, DONE
0020 120000     F       51                   CALL    OUTCHR        ;OTHERWISE, SEND IT
0023 A3                 52                   INC     DPTR          ;POINT TO NEXT CHARACTER
0024 80F6               53                   JMP     OUTSTR        ; AND SEND IT TOO
0026 22                 54     EXIT:         RET
```

Torna as subrotinas visíveis a outros módulos

- **Ficheiro .lst e tabela de símbolos (cont.)**

```
                          55
                          56      ;*******************************************************************
                          57      ; INPUT CHARACTERS TO BUFFER                                       *
                          58      ;*******************************************************************
0027 7800      F          59      INLINE:     MOV     R0,#BUFFER  ;USE R0 AS POINTER TO BUFFER
0029 120000    F          60      AGAIN:      CALL    INCHAR      ;GET A CHARACTER
002C 120000    F          61                  CALL    OUTCHR      ; ECHO IT BACK
002F F6                   62                  MOV     @R0,A       ;PUT IT IN BUFFER
0030 08                   63                  INC     R0          ;INCREMENT POINTER TO BUFFER
0031 B40DF5               64                  CJNE    A,#CR,AGAIN ;IF NOT CR, GET ANOTHER CHAR
0034 7600                 65                  MOV     @R0,#0      ;PUT NULL BYTE AT END
0036 22                   66                  RET
                          67
                          68      ;*******************************************************************
                          69      ; OUTPUT CONTENTS OF BUFFER                                        *
                          70      ;*******************************************************************
0037 7800      F          71      OUTLINE:    MOV     R0,#BUFFER  ;USE R0 AS POINTER TO BUFFER
0039 E6                   72      AGAIN2:     MOV     A,@R0       ;GET CHARACTER FROM BUFFER
003A 6006                 73                  JZ      EXIT2       ;IF NULL BYTE, DONE
003C 120000    F          74                  CALL    OUTCHR      ;OTHERWISE, SEND IT
003F 08                   75                  INC     R0          ;POINT TO NEXT CHAR IN BUFFER
0040 80F7                 76                  JMP     AGAIN2      ; AND SEND IT TOO
0042 22                   77      EXIT2:      RET
                          78
                          79      ;*******************************************************************
                          80      ; CREATE A BUFFER IN ONCHIP RAM                                    *
                          81      ;*******************************************************************
----                      82                  RSEG    ONCHIP      ;BEGIN RELOCATABLE DATA SEGMENT
0000                      83      BUFFER:     DS      LENGTH      ;ALLOCATE INTERNAL RAM AS BUFFER
                          84                  END
```

Reserva espaço para 40 caracteres

```
SYMBOL TABLE LISTING
------ ----- -------

N A M E          T Y P E   V A L U E        A T T R I B U T E S

AGAIN . . .      C ADDR    0029H    R       SEG=EPROM
AGAIN2. . .      C ADDR    0039H    R       SEG=EPROM
BUFFER. . .      D ADDR    0000H    R       SEG=ONCHIP
CR. . . . .        NUMB    000DH    A
EPROM . . .      C SEG     0043H            REL=UNIT
EXIT. . . .      C ADDR    0026H    R       SEG=EPROM
EXIT2 . . .      C ADDR    0042H    R       SEG=EPROM
INCHAR. . .      C ADDR    0014H    R PUB   SEG=EPROM
INIT. . . .      C ADDR    0000H    R PUB   SEG=EPROM
INLINE. . .      C ADDR    0027H    R PUB   SEG=EPROM
LENGTH. . .        NUMB    0028H    A
ONCHIP. . .      D SEG     0028H            REL=UNIT
OUTCHR. . .      C ADDR    000CH    R PUB   SEG=EPROM
OUTLINE . .      C ADDR    0037H    R PUB   SEG=EPROM
OUTSTR. . .      C ADDR    001CH    R PUB   SEG=EPROM
RI. . . . .      B ADDR    0098H.0  A
SBUF. . . .      D ADDR    0099H    A
SCON. . . .      D ADDR    0098H    A
SUBROUTINES        ----    ----
TH1 . . . .      D ADDR    008DH    A
TI. . . . .      B ADDR    0098H.1  A
TMOD. . . .      D ADDR    0089H    A
TR1 . . . .      B ADDR    0088H.6  A

REGISTER BANK(S) USED: 0
```

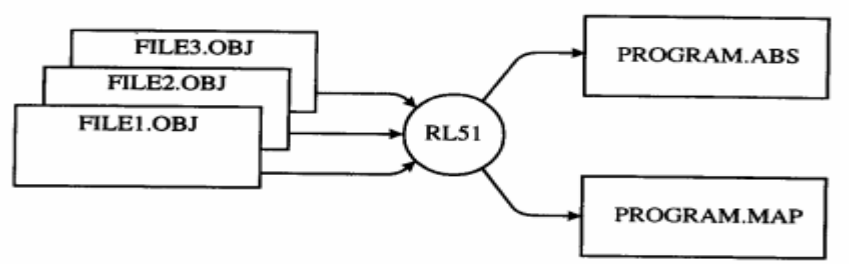
- **Qual a principal diferença entre um ficheiro objeto absoluto e um *relocatable*?**

1. **O ficheiro objeto absoluto contém apenas *bytes* binários (00H – FFh) de um programa em linguagem máquina**

2. **Um ficheiro *relocatable* contém ainda uma tabela de símbolos e outras informações necessárias para que o *linker* possa produzir o programa executável**

- **Qual a sintaxe de invocação do *linker*?**

RL51 lista_de_*relocatable* [ficheiro_destino] [controlos_de_localização]



Exemplo:

Pg. 164: errado

RL51 ECHO.obj, IO.obj TO EXAMPLE &

CODE(EPROM(8000H)) DATA(ONCHIP(30))

1. Ficheiros *relocatable*: echo.obj, io.obj
2. Programa executável: example
3. Os módulos apresentam dois segmentos *relocatable* EPROM para código localizado em 8000H e ONCHIP para dados localizado a partir de 30H

• Exemplo de ficheiro .Map criado pelo *Linker*

```
DOS 3.31 (038-N) MCS-51 RELOCATOR AND LINKER V3.0, INVOKED BY:
C:\ASM51\RL51.EXE ECHO.OBJ,IO.OBJ TO EXAMPLE CODE(EPROM(8000H))DAT
>> ))

INPUT MODULES INCLUDED
  ECHO.OBJ(MAIN)
  IO.OBJ(SUBROUTINES)

LINK MAP FOR EXAMPLE(MAIN)

            TYPE    BASE      LENGTH    RELOCATION   SEGMENT NAME
            ----    ----      ------    ----------   ------------

            REG     0000H     0008H                  "REG BANK 0"
                    0008H     0028H                  *** GAP ***
            DATA    0030H     0028H     UNIT         ONCHIP

                    0000H     8000H                  *** GAP ***
            CODE    8000H     0067H     UNIT         EPROM
```

Diga como apareceria no ficheiro executável a codificação da instrução <u>CALL INIT</u>, invocada no módulo MAIN?

```
SYMBOL TABLE FOR EXAMPLE(MAIN)

VALUE           TYPE            NAME
-----           ----            ----

-------         MODULE          MAIN
N:0000H         SYMBOL          CR
C:8000H         SEGMENT         EPROM
C:8003H         SYMBOL          LOOP
C:8000H         SYMBOL          MAIN
C:8011H         SYMBOL          PROMPT
-------         ENDMOD          MAIN

-------         MODULE          SUBROUTINES
C:804DH         SYMBOL          AGAIN
C:805DH         SYMBOL          AGAIN2
D:0030H         SYMBOL          BUFFER
N:0000H         SYMBOL          CR
C:8000H         SEGMENT         EPROM
C:804AH         SYMBOL          EXIT
C:8066H         SYMBOL          EXIT2
C:8038H         PUBLIC          INCHAR
C:8024H         PUBLIC          INIT
C:8048H         PUBLIC          INLINE
N:0028H         SYMBOL          LENGTH
D:0030H         SEGMENT         ONCHIP
C:8030H         PUBLIC          OUTCHR
C:8058H         PUBLIC          OUTLINE
C:8040H         PUBLIC          OUTSTR
B:0098H         SYMBOL          RI
D:0099H         SYMBOL          SBUF
D:0098H         SYMBOL          SCON
D:008DH         SYMBOL          TH1
B:0098H.1       SYMBOL          TI
D:0089H         SYMBOL          TMOD
B:0088H.6       SYMBOL          TR1
-------         ENDMOD          SUBROUTINES
```

Esta tabela de símbolos aparece no ficheiro .M51 porque os ficheiros .src começam com a diretiva $DEBUG

1. Do ficheiro echo.lst, linha 14 obtém-se o *opcode* da instrução como sendo 12H
2. Da tabela de símbolos do ficheiro example.M51 obtém-se o endereço absoluto de INIT como sendo 8024H
3. Sendo CALL de 3 bytes (**porquê?**) a respectiva codificação seria **12 8024h**

- **Qual o formato da linguagem *assembly*?**

1. **Um programa *assembly* contém os seguintes <u>elementos</u>:**
    1. **Instruções máquina**
    2. **Diretivas para o *assembler***
    3. **Controles do *assembler***
    4. **Comentários**
2. **O formato genérico para cada linha é o seguinte:**
    1. **[símbolo] mnemónica [operando] [,operando] [...] [;comentário]**
    2. A mnemónica pode ser uma instrução ou uma diretiva para o *assembler*
    3. O operando contém o endereço ou dado usado pela instrução
    4. O primeiro caractere de um símbolo deve ser sempre uma letra, ´**?**´, ou ‘**_**’ que será seguida por letras, dígitos, ‘**?**’, ‘**_**’. Pode conter no máximo **31/256?** caracteres
    5. Uma etiqueta (*label*) é um símbolo que termina obrigatoriamente com o caractere **’:’**.

- **Exemplo de símbolos**

PAR	**EQU**	**500**	**Símbolo**	**legal**
START:	**MOV**	**A, #0FFH**	**Etiqueta**	**legal**
Mensagem:	**DB**	**‘Help’**	**Etiqueta**	**legal**
VAR:	**DS**	**10**	**Etiqueta**	**legal**
BitVar:	**DBIT**	**16**	**Etiqueta**	**legal**
Start:	**JMP**	**THERE**	**etiqueta**	**ilegal /duplicado**
DATA	**EQU**	**100**		**ilegal/directiva *assembly***
LOW	**CODE**	**80H**		**ilegal / operador do *assembly***
ALPHA#	**DW**	**00FFh**		**contém caractere ilegal**
?_?_?	**DBIT**	**12**		**legal**
1st_var	**DS**	**3**		**1º caractere é ilegal**
MOD	**DW**	**19**		**ilegal / operador do *assembler***
TITLE	**SEGMENT**			**ilegal / controlo do *assembler***
MOV	**IDATA**			**ilegal /instrução *assembly***

- **Características dos dados imediatos**
  - Todos os dados imediatos com excepção do MOV DPTR, #dado, requerem constantes de 8-bit
  - No entanto, os dados imediatos são tratados como constantes de 16-bit e posteriormente usado apenas o LSB

Neste caso todos os bits do MSB devem ser iguais

CONSTANTE	EQU	100	
	MOV	A, #0FEH	legal
	ORL	40H, #CONSTANTE	legal
	MOV	A, #0FF00H	legal
	MOV	A, #00D8H	legal
	MOV	A, 0FE00H	ilegal
	MOV	A, #01FFH	ilegal

- **Como especificar o endereço de um bit numa instrução?**

	ON	**EQU**	**7**
Explicitamente pelo endereço	SETB	0E7H	
	JNB	99H, $	
Usando o operador .	SETB	ACC.7	
	SETB	224.ON	
	SETB	0E0H.7	
Usando um símbolo pré-definido	JNB	TI, $	
	CLR	C	

- **Bases númericas**

Decimal	MOV	A, #15	
	MOV	A, #15D	
Binário	MOV	#1111B	
Octal	MOV	A, #17Q	
	MOV	A, #17O	
Hexadecimal	MOV	A, #0FH	
	MOV	A, #0A5H	
	~~MOV~~	~~A, #A5H~~	**?**

Para diferenciar um dado hexadecimal imediato de um símbolo

- **Cadeias de caractere**

	MOV	A, #’m’	**carrega A com 06Dh, valor do ASCII m**
ASPAS	EQU	‘’’	**Define ASPAS como sendo 22h**
	DB	‘8051’	**Armazenar na memória de código ’8’, ‘0’, ‘5’,’1’**
AGUARDE:	CJNE	A, #’Q’, AGUARDE	**Fica a espera que (A) seja ≠ do caractere ‘Q’**
	SUBB	A, #’0’	Converte dígito ASCII para dígito binário
	MOV MOV	DPTR, #’AB’ DPTR, #4142H	Ambas as instruções carregam o DPTR com a cadeia de caractere ‘AB’

- **Operadores de expressão**

Símbolo do Operador	Operação
+, -,	Adição, subtração
/, *	Divisão, multiplicação
MOD	Resto da divisão
OR	Ou-lógico
AND	E-lógico
XOR	Ou-exclusivo
NOT	Complemento
SHR	Rodar à direita
SHL	Rodar à esquerda
HIGH	Obter o MSB
LOW	Obter o LSB
EQ, =	Igual a
NE, <>	Diferente
LT, <	Menor que
LE, <=	Menor ou igual
GT, >	Maior que
GE, >=	Maior ou igual

Precedência do Operadores
( )
HIGH, LOW
* , /, MOD, SHL, SHR
+, -
EQ, NE, LT, LE, GT, GE, =, <>, <, <=, >, >=
NOT
AND
OR, XOR
Operadores com mesma precedência são avaliados da esquerda para direita

Estrutura do programa e programação

Assembly com 8051

Expressão	Resultado
'A' – 'B'	0001H
HIGH(0AADDH)	0AAH
LOW(0AADDH)	0DDH
7 MOD 4	3
1000B SHR 2	0010B
NOT 1	0FFFEH
'A' SHL 8	4100H
5 EQ 8	0000H
'A' LT 'B'	0FFFFH
3 <= 3	0FFFFH
-1	0FFFFH
7 NE 4 ou 7<>4	0FFFFH
1101B XOR 0101B	1000B
HIGH('A' SHL 8)	0041H
HIGH 'A' SHL 8	0000H
'A' OR 'A' SHL 8	4141H
NOT 'A' – 1	0FFBFH

```
TRES    EQU     3
MTRES   EQU     -3
        MOV     A, #10 + 10H
        MOV     A, #1AH
        MOV     A, #25 MOD 7
        MOV     A, #4
        MOV     A, #'9' AND  0FH
        MOV     A, #9
        MOV     A, # (NOT  TRES)  +  1
        MOV     A, #MTRES
        MOV     A, #11111101B
        MOV     A, #8 SHL 1
        MOV     A, #10H
        MOV     A, #12H
        MOV     A, #HIGH 1234H
        MOV     A, #5 = 5
        MOV     A, #$ > 0
        MOV     A, #100 GE 50
```

Estrutura do programa e programação

Assembly com 8051

Categoria	Diretiva para ASM51		Síntaxe		Função
Controle do estado	**ORG**		**ORG**	expressão	Especifica um valor para contador de localização do segmento ativo
	END		**END**		Indica ao *assembler* o fim do programa fonte
	USING		**USING**	expressão	Indica ao *assembler* o banco de registo usado no código que vem a seguir à diretiva. Repare que a comutação do banco de registro deve ser efetuada usando apenas instruções do 8051
Definição de símbolos	**SEGMENT**	Símbolo	**SEGMENT**	tipo_de_segmento	Declara um símbolo como sendo um segmento *relocatable* de um dado tipo. Para começar a usar o segmento, deve-se usar a diretiva RSEG
	EQU	Símbolo	**EQU**	expressão	Atribuí um valor a um símbolo
	SET	Símbolo	**SET**	expressão	Igual ao EQU, exceptuando o fato de permitir a redefinição o símbolo
	DATA	Símbolo	**DATA**	expressão	Atribui ao símbolo um endereço direto da RAM interna
	IDATA	Símbolo	**IDATA**	expressão	Atribui um endereço indiretamente endereçável da RAM interna ao símbolo
	XDATA	Símbolo	**XDATA**	expressão	Atribui ao símbolo um endereço da memória externa
	BIT	Símbolo	**BIT**	expressão	Atribuí um endereço direto da área de memória endereçável ao bit a um símbolo
	CODE	Símbolo	**CODE**	expressão	Atribuí um endereço da memória de código ao símbolo

Estrutura do programa e programação

Assembly com 8051

```
DEZ             EQU     10              ; O símbolo DEZ  passa a ser uma constante de valor 10
CONTADOR        EQU     R7              ; INC R7 pode agora ser substituído por INC CONTADOR
NOVO_DEZ        EQU     DEZ             ; símbolo atribuído o valor de outro símbolo já definido
CINCO           EQU     DEZ / 2         ; símbolo atribuído o valor de uma expressão aritmética
A_REG           EQU     A               ; A_REG pode ser  usado legalmente onde A é normalmente usado
ASCII_D         EQU     'D'             ; símbolo atribuído o valor do caractere ASCII D
HERE            EQU     $               ; valor do contador de localização atribuído ao símbolo
EPROM           SEGMENT CODE            ; declaração do símbolo como sendo um segmento de código
APONTADOR       SET     R0              ; endereço de R0 atribuído ao símbolo
 APONTADOR       SET     R1             ; redefinição de APONTADOR para registro R1
VALOR           SET     1               ;Inicialização de um símbolo variável com valor 1
VALOR           SET     VALOR + 1       ; incremento do conteúdo do símbolo VALOR
 CF              BIT     0D7H           ; símbolo apontado para a flag carry  no registro PSW
OFF_FLAG        BIT     6               ; endereço de memória atribuído a uma flag
ON_FLAG         BIT     OFF_FLAG + 1    ; o próximo bit é uma outra flag
RESET           CODE    0               ; atribuição de um endereço de código a um símbolo
EXIT0           CODE    RESET + (1024/16) ; atribuição de um endereço de código através de uma expressão
```

Jump

Estrutura do programa e programação

Assembly com 8051

Categoria	Diretiva para ASM51		Síntaxe		Função
Incialização e reserva de armazenamento	**DS**	[LABEL:]	**DS**	expressão	Reserva espaços em múltiplos de *bytes*. Não pode ser utilizado com segmento do tipo BIT. O valor da expressão deve ser conhecida pelo *assembler*
	DBIT	[LABEL:]	**DBIT**	expressão	Reserva espaços em múltiplos de bits. O valor da expressão deve ser conhecida pelo *assembler*
	DB/DW	[LABEL:]	**DB/DW**	expressão	Inicializa a memória de código com valores do tipo byte/word
Program linkage	**PUBLIC**		**PUBLIC**	Símbolo [, símbolo] [...]	Define uma lista de símbolos que tornam visíveis e utilizáveis a partir de outros módulos
	EXTRN		**EXTRN**	Tipo_segmento(símbolo [,símbolo] [...], ...)	Informa o *assembler* da lista de símbolos definidos noutros módulos e que vão ser utilizados neste. O tipo de segmento pode ser CODE, DATA, XDATA, IDATA, BIT e um especial designado por NUMBER que especifica um símbolo definido por EQU
	NAME		**NAME**	Nome_do_módulo	
Seleção de Segmentos	**RSEG**		**RSEG**	Nome_do_segmento	Ao encontrar uma diretiva de seleção de segmento, o *assembler* direciona o código
	CSEG		**CSEG**	[**AT** endereço]	ou dado que lhe segue para o segmento
	...		**DSEG**	[**AT** endereço]	selecionado até que seja selecionado um
	XSEG		**XSEG**	[**AT** endereço]	outro segmento

Estrutura do programa e programação

Assembly com 8051

```
PSW             DATA        0D0H                ;define o endereço do registro de estado do programa
BUFFER          DATA        32                  ; definição de um endereço na RAM interna
FREE_SPACE      DATA        BUFFER + 6          ; definição de endereço usando expressão aritmética
USER_BASE       XDATA       2048                ; definição de um endereço na memória externa
HOST_BASE       XDATA       USER_BASE+100h      ; definição de endereço usando expressão aritmética
                ORG         1000H               ; atualização do contador de localizações do segmento
                                                ;ativa com 4096
                ORG         RESET               ; atualização do contador de localização com o valor do
                                                ;símbolo já definido
                ORG         BASE + MODULE_NO    ; atualização do LC  usando expressão aritmética
                USING       0                   ; seleção dos endereços do banco de registro 0
                USING       1+1+1               ; seleção dos endereços do banco de registro 3 através de
                                                ;expressão aritmética
                ORG         ($ + 1000H) AND 0F000H ; atualização do LC  para a próxima fronteira de 4K
                DSEG                            ; seleciona o segmento absoluto de dado diretamente
                                                ;endereçável (DATA)  usando o valor do LC atual
                BSEG        AT  32              ; Seleciona o segmento de dado endereçável a bit
                                                ;(BITDATA) e atualiza o LC com 32
                XSEG        AT (USER_BASE *5) MOD 16 ; Seleciona o segmento absoluto  de dado externo
                                                ;(XDATA) e usa uma expressão aritmética para atualizar
                                                ;o LC
```

Estrutura do programa e programação

Assembly com 8051

```
                    DSEG                                ; seleciona o segmento absoluto de dado (DATA)
                    DS      32                          ; repare que a label é opcional
SP_BUFFER:          DS      16                          ; reserva um buffer de 16 bytes  para a comunicação série
IO_BUFFER:          DS      8                           ; reserva um buffer de 8 bytes para a entrada/saída
                    BSEG                                ; seleciona o segmento absoluto endereçável a bit
                    DBIT    16                          ; a label é opcional
IO_MAP:             DBIT    32                          ; reserva um buffer de 32 bit para operações de entrada/saída
MENSAGEM:           DB      '(c) Copyright, 1984'       ; cadeia de caractere
RUNTIME_CONST:
                    DB      127, 13, 64, 0, 99          ; tabela de constantes
                    DB      17, 32, 239, 163, 49        ; a label é opcional
MIXED:              DB      2*8, 'MPG', 2+6, 'ABC'      ; pode-se misturar números com caracteres ASCII
JUMP_TABLE          DW      RESET, START, END           ; tabela de endereços
                    DW      TRUE, TEST, FALSE           ; label é opcional
RADIX:              DW      'H', 1000H                  ; 1ºbyte=0, 2ºbyte=48h, 3ºbyte=10h, 4ºbyte=0
```

- **Uso das diretivas *PUBLIC* e *EXTERN***

```
        EXTRN   CODE (HELLO, GOODBYE)   ; declaração de subrotinas externas
        EXTRN   DATA (BUF)              ; declaração de um dado externo
        PUBLIC  FLAG                    ; torna o símbolo FLAG visível e utilizável noutros módulos
FLAG    EQU     10H                     ; definição de uma constante com valor 10h
        ...     ...
        CALL    HELLO                   ; invocar a subrotina HELLO a partir do módulo 2
        ...     ...
        CALL    GOODBYE                 ; invocar a subrotina GOODBYE a partir do módulo 2
        ...     ...
        END                             ; seleciona o segmento absoluto endereçável a bit
```

- **Uso das diretivas *PUBLIC* e *EXTERN***

```
                EXTRN     NUMBER (FLAG)          ; declaração de uma constante externa
                PUBLIC    HELLO, GOODBYE, BUF    ; torna as subrotinas e o buffer  visíveis e utilizáveis
                                                 ;noutros módulos
                ...       ...
HELLO:          MOV       A, #FLAG               ; início da subrotina HELLO
                ...       ...
                RET                              ; conclusão da subrotina HELLO

GOODBYE:        CLR       C                      ; início da subrotina GOODBYE
                ...       ...
                RET                              ; conclusão da subrotina GOODBYE
BUF:            DS        10H                    ;
                END                              ; fim do módulo
```

Estrutura do programa e programação

Assembly com 8051

- **Definição e inicialização de segmentos**

```
ONCHIP   SEGMENT   DATA       ; declaração de um segmento de dado relocatable
EPROM    SEGMENT   CODE       ; declaração de um segmento de código  relocatable
         BSEG      AT   70H   ; inicialização de um segmento absoluto  endereçável ao bit a partir do endereço de bit
                              ;70h que é precisamente o bit 0 do endereço 2EH (consultar a área endereçável ao bit
                              ;visto nas primeiras aulas)
FLAG1:   DBIT      1          ; FLAG1 é um bit endereçável  pelo endereço 70h
FLAG2:   DBIT      2          ; FLAG2 é  endereçável  pelo endereço 71h
         RSEG      ONCHIP     ;inicialização do segmento relocatable ONCHIP
TOTAL:   DS        1          ; label que aponta para localização endereçável ao byte. Qual o endereço?
COUNT:   DS        1          ; label que aponta para localização endereçável ao byte. Qual o endereço?
SUM:     DS        2          ; label para uma localização endereçável a Word. Qual o endereço?
         RSEG      EPROM      ; inicialização de do segmento relocatable EPROM
BEGIN:   MOV       TOTAL, #0  ; Qual o endereço da label BEGIN?
         ...       ...
         END                  ; fim do módulo
```

Sendo este símbolos definidos em segmentos *relocatable*, só serão estabelecidos pelo *linker*

Estrutura do programa e programação

Assembly com 8051

- **Criar um *buffer* de 40 bytes(a partir do endereço 40H) na RAM interna e inicializá-la com o ASCII a:**

```
        DSEG    AT  40H     ; inicialização do segmento de dado interno (DATA) a partir do endereço 40H
LEN     EQU     40          ; símbolo constante com a dimensão do buffer
BUFFER: DS      LEN         ; 40 bytes são reservadas a partir do endereço 40h  do segmento DATA
        CSEG  AT    0
        MOV     R7, #LEN    ; usa R7 como o contador de posições a inicializar
        MOV     R0, #BUFFER ; aponta R0 para a primeira posição do buffer
LOOP:   MOV     @R0, #'a'   ;escrever no endereço apontado por R0 actualmente, o caractere 'a'
        DJNZ    R7, LOOP    ; já atingiu a última posição? Se não, então continua a preencher
        ...                 ; caso contrário, acabou
        END                 ; fim do módulo
```

Apresente uma alternativa para a colocação do *buffer* a partir do endereço 40H?

Basta substituir a primeira linha por

DSEG

ORG 40H

Estrutura do programa e programação

Assembly com 8051

- **Criar um um *buffer* de 500 words(a partir do endereço 4000H) na RAM externa e inicializá-la com zeros**

```
ADDR     EQU    4000H                          ;inicialização de uma constante com o endereço do buffer
LEN      EQU    1000                           ; inicialização de uma constante com a dimensão do buffer
                                               ;(500 words)
         XSEG   AT  ADDR                       ; inicialização do segmento absoluto XDATA
BUFFER:  DS     LEN                            ;" alocação " de espaço para as 500 words
         CSEG   AT 0000H
         MOV    DPTR, # BUFFER                 ; apontar para o início do buffer a "zerar"
         CLR    A                              ; preparar o caractere de inicialização no acumulador
LOOP:    MOV    @DPTR, A                       ; "zerar" a posição do buffer actualmente apontada por
                                               DPTR
         INC    DPTR                           ; aponta para a próxima posição
         MOV    A, DPL                         ;após  preencher a última posição do buffer dptr apontará
         CJNE   A, #LOW(BUFFER + LEN +1), LOOP ; uma posição abaixo. Como não existe uma instrução para
         MOV    A, DPH                         ;comparar dados de 16-bits, torna-se necessário efectuar a
         CJNE   A, #HIGH(BUFFER + LEN +1), LOOP ;comparação do MSB e LSB do DPTR com os do endereço
                                               ; que está uma posição após a última posição do buffer
         ...                                   ; se forem iguais então acabou o"zerar" do buffer
         END                                   ; fim do módulo
```

Estrutura do programa e programação

Assembly com 8051

- **Sabendo que o ASCII A= 41H, ASCII a = 61H e ASCII : =3AH, descreva o estado da memória de código após o *assembler* interpretar as seguintes diretivas:**

```
            CSEG    AT      0100H
SQUARES:    DB      0, 1, 4, 9, 16, 25      ;quadrados dos números 0-5
MESSAGE:    DB      'Login:' , 0            ;string terminado com caractere nulo
```

'L' = 'A' + 0Ch – 01h = 4Ch

'o' = 'a' + 0Fh – 01h = 6Fh

....

Endereço	Conteúdo
0100h	00h
0101h	01h
0102h	04h
0103h	09h
0104h	10h
0105h	19h
0106h	4Ch
0107h	6Fh
0108h	67h
0109h	69h
010Ah	6Eh
010Bh	3Ah
010Ch	00h

Estrutura do programa e programação

Assembly com 8051

- **Descreva o estado da memória de código após o *assembler* interpretar as seguintes diretivas:**

```
CSEG    AT      200H
DW      $, ‘A’, 1234H, 2 , ‘BC’
```

Endereço	Conteúdo
0200h	00h
0201h	02h
0202h	41h
0203h	00h
0204h	34h
0205h	12h
0206h	02h
0207h	00h
0208h	43h
0209h	42h

Estrutura do programa e programação

Assembly com 8051

- **Controles do *Assembler***

  Usados para controlar:

  - **onde o *assembler* obtém os ficheiros de entrada**
  - **onde colocar os ficheiros objetos**
  - **como formatar os ficheiros de listagem**

Controlos do *assembler*	
$DATE(data)	Coloca a data no cabeçalho da página
$TITLE(*string*)	Coloca uma cadeia de caracteres no cabeçalho da página
$PAGING	Dividir o ficheiro de listagem em várias páginas
$PAGEWIDTH(n)	Especifica o número de colunas (caracteres por linha) numa página de listagem
$SYMBOLS	Cria uma tabela formatada de símbolos usada no programa
$NOPRINT	Impede a criação de ficheiros de listagem
$DEBUG	Escreve informação sobre símbolo de depuração nos ficheiros objetos
$EJECT	Continua a listagem na próxima página
$ERRORPRINT(file)	Especifica um ficheiro para as MSGs de erros
$MOD51	Reconhece os SFR pré-definidos para o 8051
$LIST	Escreve cada linha do ficheiro de código *assembly* no ficheiro de listagem
...	**...**

- **Macros**

  Útil quando uma parcela de código é usado repetidamente no programa

  i. **Escreve-se a parcela de código uma única vez e será usada posteriormente em qualquer parte do programa através do uso do nome da macro (posição onde o *assembler* expande o código da macro)**

  ii. **Ao contrário das macros, as subrotinas são invocadas através de instruções especiais – CALLs (o corpo da subrotina não é expandida, mas muda-se o fluxo de execução)**

  iii. **As macros são usadas no campo das mnemónicas de uma linha de código *assembler* como se fosse uma instrução**

    i. **O nome da macro é precedido de % para a distinguir de uma instrução 8051**

- **Quando usar macro ou subrotina?**

1. **Subrotinas são normalmente usadas para tarefas complexas e que envolvam grandes quantidades de código em que o *overhead* associado á chamada/retorno seja tolerável**

   Poupa na memória de código

2. **Macros são usadas para tarefas simples ou então quando se requer a velocidade do código *inline***

   Consome mais memória de código

- **O que acontece quando uma subrotina é invocada?**

  1. **O apontador de instruções (PC) é automaticamente guardado na pilha**
  2. **PC será carregado com o endereço inicial da subrotina**
  3. **A subrotina é executada**
  4. **Ao concluir a execução da subrotina, a instrução RET carrega PC com o endereço anteriormente guardado na pilha**

     A execução continua com a instrução que segue a instrução CALL

- **Sintaxe da Macro**

1. **Sem passagem de parâmetro**

%*DEFINE (nome_macro) (corpo da macro)

2. **Com Passagem de parâmetro**

%*DEFINE (nome_macro (lista_parm)) (corpo da macro)

3. **Genérica com etiquetas**

%*DEFINE (nome_macro [(lista_parm)])
[Local lista_label] (corpo da macro)

Nota: repare que as listas de argumentos e de etiquetas são opcionais

- **Exemplos de Macro**

```
%*DEFINE (PUSH_DPTR)
        ( PUSH DPH
        PUSH DPL )
```

```
%*DEFINE (CMP_NUM (VALOR) )
        (CJNE A, #%VALOR, $+3)
```

```
%*DEFINE (JGT (VALOR, SALTO) )
        (CJNE A, #%VALOR+1, $+3
        JNC     %SALTO)
```

CJNE é uma instrução de 3 *bytes*

```
%*DEFINE (DEC_DPTR)  LOCAL SKIP
        (DEC DPL
        MOV A, DPL
        CJNE A, #0FFH, %SKIP
        DEC DPH
%SKIP:)
```

```
...
START:  MOV A, #3
        %CMP_NUM (20)
        JNC MAIOR
        JMP $
MAIOR:  MOV A, #0FFH
        END
```

- **Preservação de dados**

1. **Deve-se sempre salvaguardar o conteúdo dos registros alterados pelas macros/subrotinas, caso sejam necessários (com os valores anterior à chamada) após a chamada da macro/subrotina**

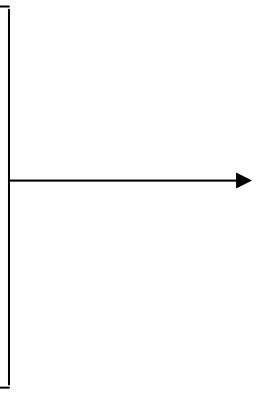
2. **Uma forma de garantir a preservação destes dados consiste em armazená-los na pilha (*push*), no início do corpo da macro/subrotina e restaurá-los no final (*pop*)**

No caso das funções antes da execução do RET

- **Sendo a pilha uma estrutura LIFO (*Last In First Out*) a ordem dos *PUSHs* e *POPs* devem ser invertidas**

- **Preservação de dados**

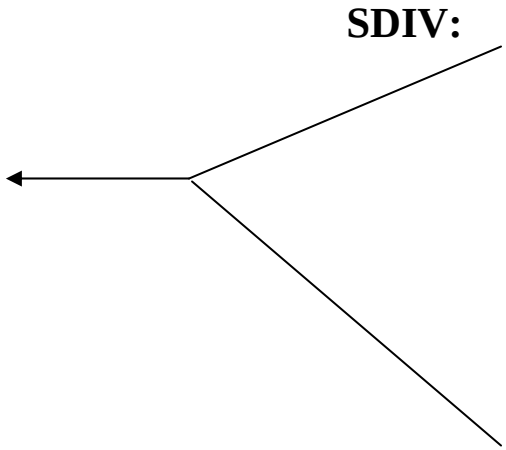
```
%*DEFINE (DEC_DPTR)  LOCAL SKIP
        (  PUSH  ACC
        DEC DPL
        MOV A, DPL
        CJNE A, #0FFH, %SKIP
        DEC DPH
        POP  ACC
%SKIP:)

START:  MOV DPTR, #10F4H
        MOVX A, @DPTR
        %DEC_DPTR
        ADD  A, #3
        END
```



Salvaguarda e restauração de contexto

Pretende-se efetuar (A) = (10F4H) + 3 e de seguida apontar DPTR para 10F3H. Será?

- **Preservação de dados**



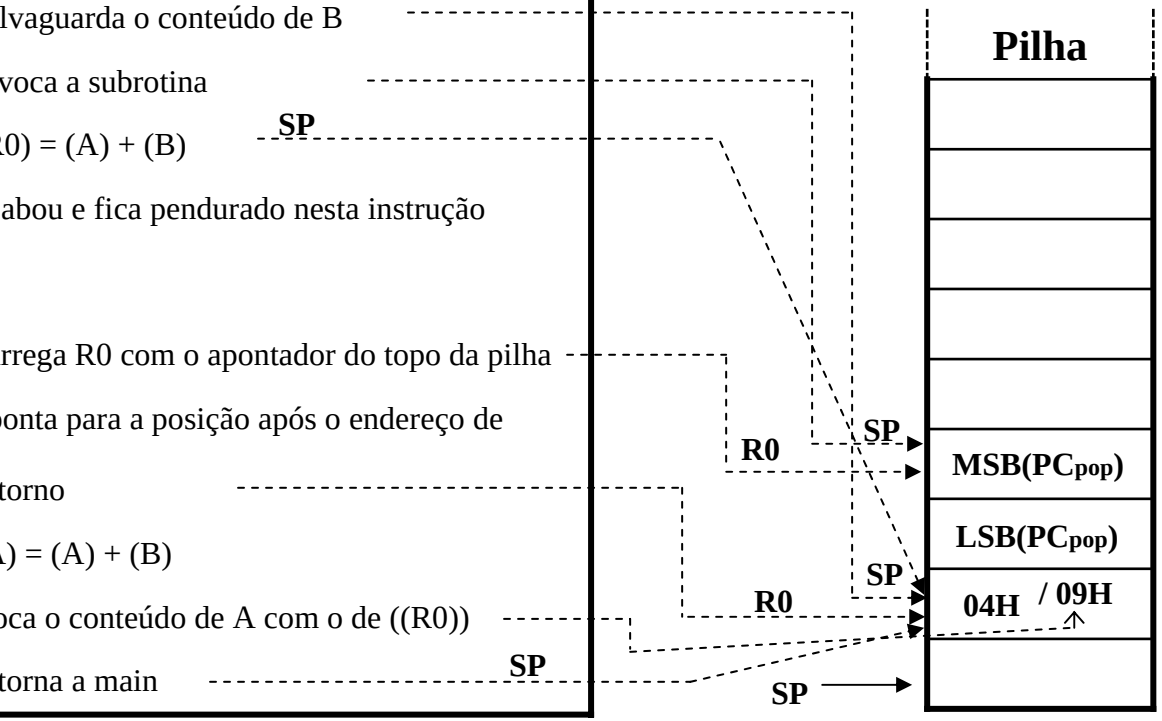
(A) = 9 (B) = 3

Pretende-se efetuar (R0) = 2 + (A) / 3 = 5
e 12 = (A) = (A) + (B). Será?

```
START:  MOV A, #9
        MOV B, #3
        CALL SDIV
        ADD  A, 0F0H
        END
```

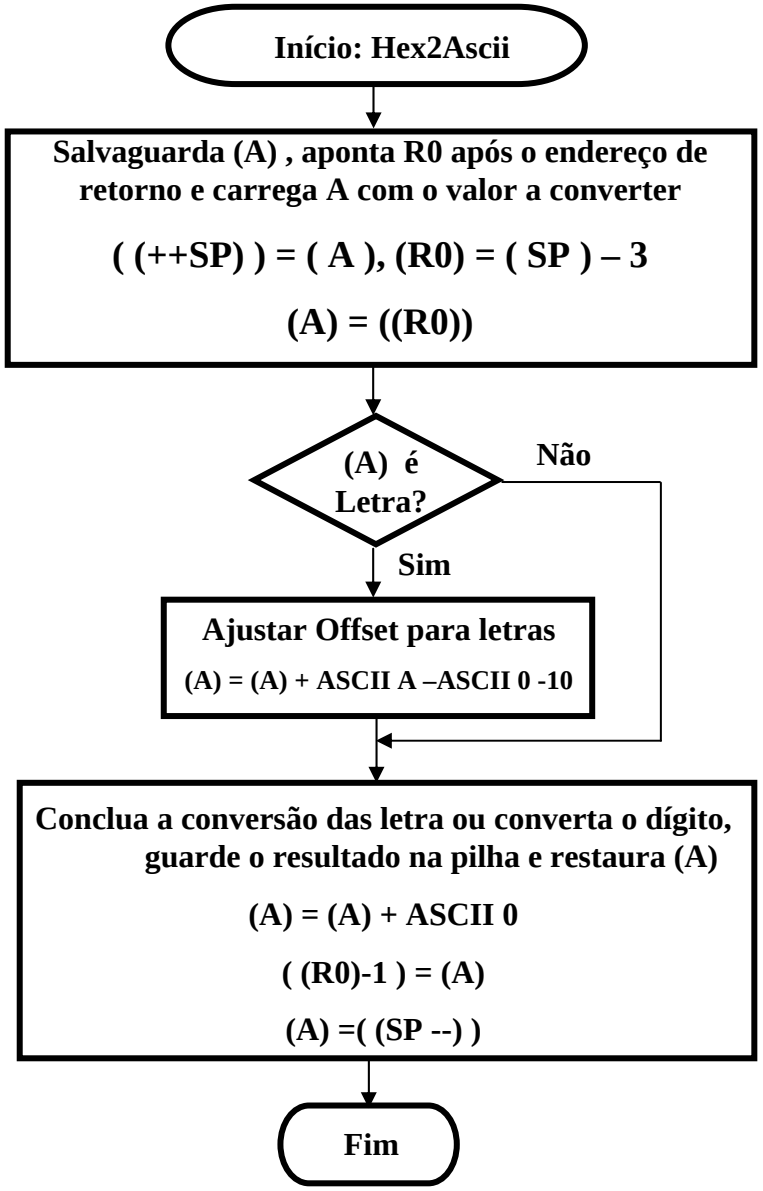
• Passagem de parâmetros através da pilha

```
MAIN:     MOV     A, #05H     ; (A) = 5
          MOV     B, #4H      ; (B) = 4
          PUSH    0F0H        ; salvaguarda o conteúdo de B
          CALL    ADD_IT      ; invoca a subrotina
          POP     00H         ; (R0) = (A) + (B)
          SJMP    $           ; acabou e fica pendurado nesta instrução

ADD_IT:   MOV     R0, SP      ; carrega R0 com o apontador do topo da pilha
          DEC     R0          ; aponta para a posição após o endereço de
          DEC     R0          ; retorno
          ADD     A, B        ; (A) = (A) + (B)
          XCH     A, @R0      ; troca o conteúdo de A com o de ((R0))
          RET                 ; retorna a main
```

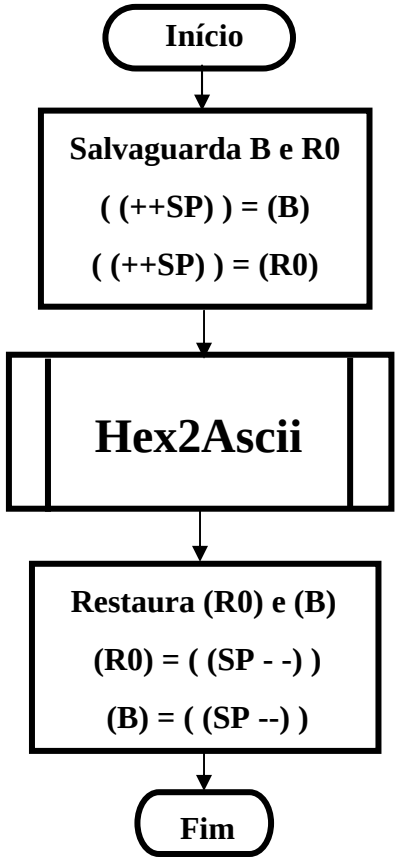


Estrutura do programa e programação

Assembly com 8051



<u>Exercício:</u>Converta um dígito armazenado em R0 para um caracter ASCII que representa o seu valor hexadecimal. O valor em R0 contém apenas um dígito hexadecimal (o *MSnibble* é 0). Guarde o resultado no registro B. Deve invocar uma subrotina que recebe o parâmetro e devolve o resultado via pilha.




```
MAIN:  PUSH   0F0H ; salvaguarda (B)
       PUSH   00H  ; salvaguarda (R0)
       CALL   H2Ascii ; converta o
                      ; dígito Hexdecimal
       POP    00H  ; restaura  (R0)
       POP    0F0H ; coloca em B o
                   ;valor convertido
       JMP  $      ; acabou
```

Estrutura do programa e programação

Assembly com 8051

; Subrotina:	**Hex2Ascii**		
; Descrição:	**Conversão de um dígito Hexadecimal para o correspondente caractere ASCII**		
; Entrada:	**Na pilha, logo após o endereço de retorno estará o dígito a converter**		
; Saída:	**Na pilha, logo após o parâmetro de entrada será colocado o resultado da conversão**		
; Registo(s) Afectado(s):	**R0**		
; Exemplo:	**entrada = 06h**		**saída = 36h ('6')**
Hex2Ascii:	PUSH	ACC	; salvaguardar o (A)
	MOV	R0, SP	; 1º apontar R0 para topo da pilha
	DEC	R0	; e depois aponta-o para
	DEC	R0	; o valor a converter localizado
	DEC	R0	; após o endereço de retorno
	MOV	A, @R0	; carregar A com o valor a converter
	CJNE	A, #10, $+3	; verificar se é letra ou dígito
	JNC	LETRA	; sendo letra, vai ajustar o offset
	JMP	GUARDA	; sendo dígito não precisa ajustar o offset, apenas vai concluir a conversão
LETRA:	ADD	A, # ('A' - '0' - 10)	; ajustar o offset no caso de letra
GUARDA:	ADD	A, #'0'	; concluir a conversão tanto para letra como dígito
	DEC	R0	; apontar R0 para a localização da pilha onde foi salvaguadado B
	MOV	@R0, A	; guardar o valor convertido nesta localização
	POP	ACC	; restaurar o (A)
	RET		; retornar ao main