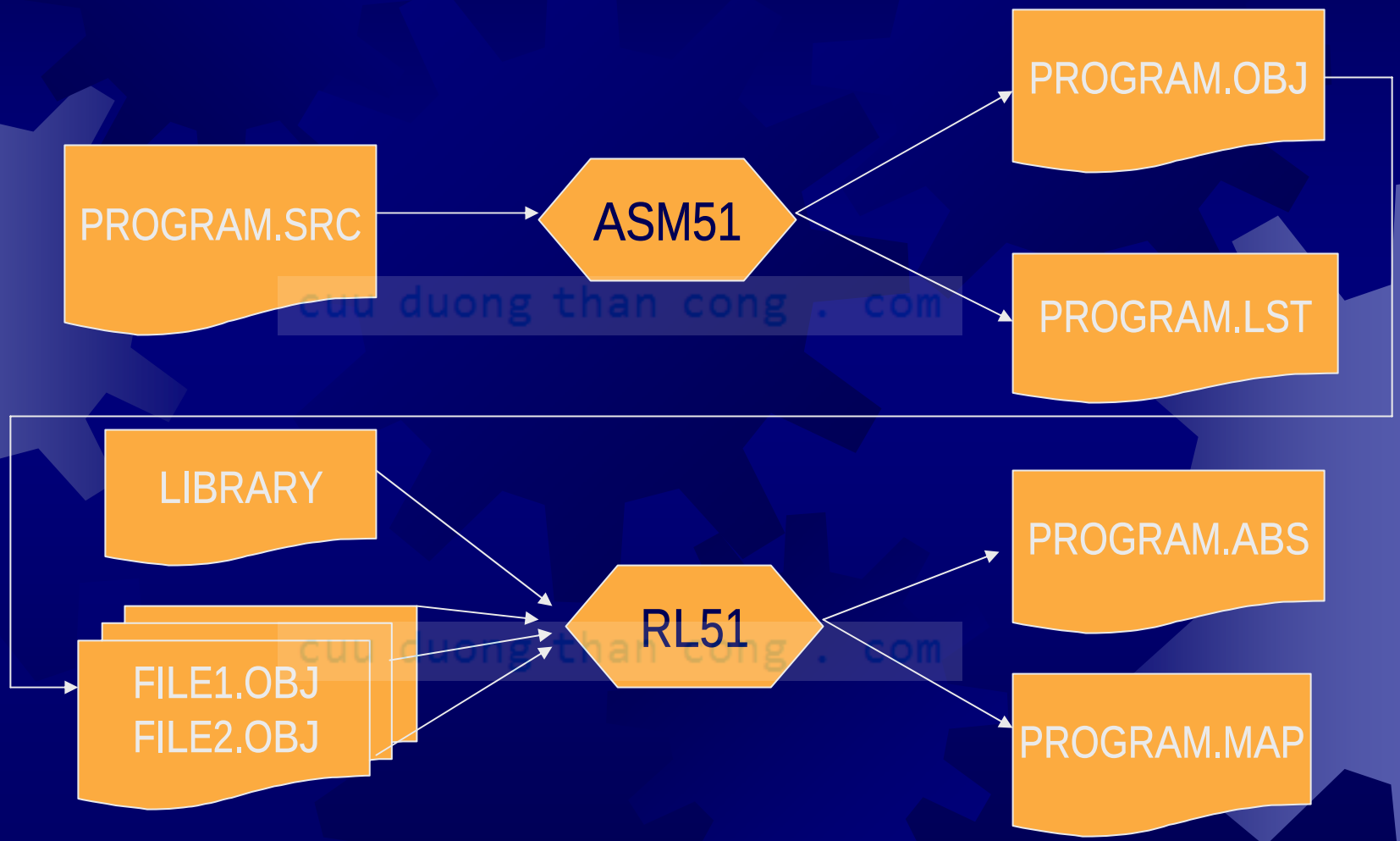
System & Program Developments of 8051

- Assembly Language Programming
  - Assembler Operation
  - Assembly Language Program Format
  - Assemble-Time Expression Evaluation
  - Assembler Directives
  - Assembler Controls
  - Linker Operation
  - Linking Relocatable Segments and Modules
  - Macros

System & Program Developments of 8051

- Program Structure and Design
  - Introduction
  - Advantages and Disadvantages of Structured Programming
  - The Three Structures: statements, loops, choice
  - Pseudo Code Syntax
  - Assembly Language Programming
- Tools & Techniques for Program Development
  - The Development Cycle
  - Integration and Verification
  - Command and Environments

Assembly Language Programming



```
ASM(input_file)    /*assemble source program in input_file)*/
BEGIN
        /*pass 1: build the symbol table */
        lc = 0;                 /*lc=location counter; default to 0*/
        mnemonic = null;
        open input_file;
        WHILE   (mnemonic != end) DO BEGIN
          get_line();           /*get line from input_file */
          scan_line();          /*scan line & get label/symbol & mnemonic */
          IF (label) THEN enter_into_symbol_table(label, lc);
          CASE  mnemonic OF BEGIN
             null, comment, END: ;   /*do nothing */
             ORG:               lc = operand_value;
             EQU:               enter_in_symbol_table( symbol, operand_value);
             DB:                WHILE (got_operand) DO increment_lc;
                                        /* increment number of bytes defined */
             DS:                lc = lc + operand_value;
             1_byte_instruction:    lc = lc + 1;
             2_byte_instruction:    lc = lc + 2;
             3_byte_instruction:    lc = lc + 3;
          END
        END
```

```
/*pass 2: create the object program*/
rewind_input_file_pointer;
lc = 0;
mnemonic = null;
open_output_file;
WHILE    (mnemonic != end) DO BEGIN
  get_line();
  scan_line(); /* determine mnemonic op code and value(s) of operand(s)*/
   /*Note: if symbols are used in operand field, their values are looked-up*/
   /* in the symbol table created during pass 1                          */
   CASE  mnemonic OF BEGIN
     null, comment, EQU, END: ;        /*do nothing */
     ORG:              lc = operand_value;
     DB:  WHILE (operand) DO BEGIN
               put_in_objectfile(operand);
               lc = lc + 1;
          END                        /* increment number of bytes defined */
```

```
        DS:   lc = lc + operand;
        1_byte_instruction:     put_in_objectfile(inst_code);
        2_byte_instruction:     put_in_objectfile(inst_code);
                                put_in_objectfile(operand);
        3_byte_instruction:     put_in_objectfile(inst_code);
                                put_in_objectfile(operand high-byte);
                                put_in_objectfile(operand low-byte);
        lc = lc + size_of_instruction;

      END
    END
    close_input_file;
    close_output_file;
END
```

Assembly Language Program Format

- Assembly language program contains:
  - Machine instructions (ANL, MOV)
  - Assembler directives (ORG,..)
  - Assembler controls ($TITLE)
  - Comments
- Lines of machine instructions or assembler directives:

  [label:] Mnemonic [operand] [,operand] […] [;comment]

Assembly Language Program Format

- **Label & symbol**

```
PAR     EQU    500          ;PAR is a symbol
                            ; which represents the
                            ; value 500
START:  MOV    A,#0FFH      ;START is a label
                            ;which represents the
                            ;address of the MOV
                            ;instruction
```

- **Special Assembler Symbols**
  - A, R0-R7, DPTR, PC, C, AB, $

Assemble-Time Expression Evaluation

- MOV A,#10 + 10H (MOV A,#1AH)
- MOV A,#25 MOD 7 (MOV A,#4)
- MOV A,#'9' AND 0FH (MOV A,#9)
- THREE EQU 3

  Minus_three EQU -3

  MOV A,#(NOT THREE)+1

  MOV A,#Minus_three

  MOV A,#11111101B
- MOV A,#8 SHL 1 (MOV A,#10H)
- MOV A,#HIGH 1234H (MOV A,12H)

Assemble-Time Expression Evaluation

- Relational operators: result is always false (0000H) or true (FFFFH)

EQ	=	equal
NE	<>	not equal
LT	<	less than
LE	<=	less than or equal to
GT	>	greater than
GE	>=	greater than or equal to

```
MOV     A,#5 = 5
MOV     A,#'X' LT 'Z'
MOV     A,#5 NE 4
MOV     A,#'X' >= 'X'
MOV     A,#$ > 0
```

Assemble-Time Expression Evaluation

- **Expression Examples:**

‘B’ - ‘A’	0001H	‘A’ SHL 8	4100H
8/3	0002H	LOW 65535	00FFH
155 MOD 2	0001H	(8 + 1)*2	0012H
4*4	0010H	5 EQ 4	0000H
8 AND 7	0000H	‘A’ LT ‘B’	FFFFH
NOT 1	FFFEH	3 <= 3	FFFFH

Assemble-Time Expression Evaluation

- **Operator Precedence**
  - ( )
  - HIGH LOW
  - * / MOD SHL SHR
  - + -
  - EQ NE LT LE GT GE = <> < <= > >=
  - NOT
  - AND
  - OR XOR
- **When same precedence, operators are evaluated left-to-right**

HIGH (‘A’ SHL 8)	0041H
HIGH ‘A’ SHL 8	0000H
NOT ‘A’ - 1	FFBFH
‘A’ OR ‘A’ SHL 8	4141H

Assembler Directives

- Assembler state control (ORG, END, USING)
- Symbol definition (SEGMENT, EQU, SET, DATA, IDATA, XDATA, BIT, CODE)
- Storage initialization/reservation (DS, DB, DW, DBIT)
- Program linkage (PUBLIC, EXTRN, NAME)
- Segment selection (RSEG, CSEG, DSEG, ISEG, BSEG, XSEG)

Assembler State Controls

- **ORG, END, USING**

```
ORG     100H
ORG     ($ + 1000H) AND 0F000H      ;set to next
                                    ;4k boundary
END     ;should be the last statement in the source file
USING   expression   ;inform ASM51 of the currently
                     ;active register bank
MOV     PSW,#00011000B      ;select RB 3, the only way
                            ;to switch register banks
USING   3       ;use register bank 3
PUSH    AR7     ;1FH (R7 in bank 3)
MOV     PSW,#00001000B
USING   1
PUSH    AR7     ;0FH
```

Symbol Definition

- The symbol definition directive creates symbols representing segments, registers, numbers, & addresses.

cuu duong than cong . com

Symbol Definition

- Segment (CODE, XDATA, DATA,IDATA,BIT)

  symbol SEGMENT segment_type

  CODE (code segment)

  DATA (the internal data space accessible by direct addressing 00-7FH)

  IDATA (the entire internal data space accessible by indirect addressing, 00-7FH on 8051, 00-FFH on 8052)

  XDATA (the external data space)

  BIT (the bit space; overlapping byte locations 20H-2FH of the internal data space)

  EPROM SEGMENT CODE (declares that EPROM is a code segment. To actually begin using this segment, the RSEG directive is used)

Symbol Definition

- **EQU and SET -- numbers**

  symbol EQU expression (assigns a numeric value to a specific symbol name)

```
N27         EQU   27
HERE        EQU   $
MESSAGE:    DB    'This is a message'
LENGTH      EQU   $-MESSAGE
```

  symbol SET expression (similar to EQU except that symbol can be redefined later using another SET directive)

Symbol Definition

- **DATA, IDATA, XDATA, BIT, & CODE: assign addresses of the corresponding segment type to a symbol.**
- **e.g.,**

```
FLAG1   EQU   05H
FLAG2   BIT   05H
        SETB  FLAG1
        SETB  FLAG2
        MOV   FLAG1,#0
        MOV   FLAG2,#0
```

(error message: data segment address expected)

Storage Initialization/Reservation

- **DS (define storage)**

  [label:] DS expression (reserves spaces in **byte units**, can be used in any segment type except BIT)

```
        DSEG  AT 30H          ;put in data segment
                              ;(absolute , internal)
LENGTH  EQU   40
BUFFER: DS    LENGTH          ;reserve 40 bytes

        MOV   R7,#LENGTH
        MOV   R0,#BUFFER      ;R0=30H
LOOP:   MOV   @R0,#0
        INC   R0
        DJNZ  R7,LOOP
```

Storage Initialization/Reservation

- Create 1000 bytes in external RAM at 4000H

```
XSTART    EQU    4000H
XLENGTH   EQU    1000
          XSEG   AT XSTART     ;put I data segment
                               ;(absolute , internal)
XBUFFER:  DS     XLENGTH      ;reserve 40 bytes
          MOV    DPTR,#XBUFFER
LOOP:     CLR    A
          MOV    @DPTR,A
          INC    DPTR
          MOV    A,DPL
          CJNE   A,#LOW(XBUFFER + XLENGTH + 1),LOOP
          MOV    A,DPH
          CJNE   A,#HIGH(XBUFFER + XLENGTH + 1),LOOP
          (continue)
```

Storage Initialization/Reservation

- **DBIT: reserve space in bit units**

```
[label:]    DBIT    expression
            BSEG                ;bit segment, absolute (00H-7FH)
KBFLAG:     DBIT    1           ;keyboard status
PRFLAG:     DBIT    1           ;printer status
DKFLAG:     DBIT    1           ;disk status
```

- **DB (define byte): initialize code memory w bye values**

```
[label:]    DB      expression [,expression] […]
            CSEG    AT      0100H
SQUARES:    DB      0,1,4,9,16,25       ;squares of number 0-5
MESSAGE:    DB      ‘Login:’,0          ;null-terminated char string
```

- **DW (define word)**

```
[label:]    DW      expression [,expression] […]
            CSEG    AT      0200H
            DW      $,’A’,1234H,2,’BC’      ;02 00 00 41 12 34 00
                                            ;02 42 43
```

Program Linkage

- Allows separately assembled modules (files) to communicate by permitting intermodule referencing and naming of the modules
- **PUBLIC** (declare public for other modules to reference)

  PUBLIC symbol [,symbol] […]

  allows the list of specified symbols to be known and used outside the currently assembled module.
- **EXTRN** (declare symbols defined outside current module)

  EXTRN symbol [,symbol] […]

MAIN.SRC

```
EXTRN     CODE(HELLO,GOOD_BYE)
…
CALL      HELLO
…
CALL      GOOD_BYE
…
END
```

MESSAGE.SRC

```
          PUBLIC  HELLO,GOOD_BYE
                  …
HELLO:    (begin sub)
                  …
                  RET
GOOD_BYE: (begin sub)
                  …
                  RET
```

Segment Selection

- **RSEG (selecting relocatable segment)**

  RSEG segment_name

  segment_name is previously defined by SEGMENT directive.

- **Selecting Absolute Segments**

  CSEG [AT address]
  DSEG [AT address]
  ISEG [AT address]
  BSEG [AT address]
  XSEG [AT address]

- **Each segment has its own location counter which is set to 0 initially**

Segment Selection

```
LOC      OBJ      LINE          SOURCE
                  1      ONCHIP SEGMENT        DATA
                  2      EPROM  SEGMENT        CODE
                  3
----              4             BSEG    AT 70H ;begin abs bit seg
0070              5      FLAG1: DBIT    1
0071              6      FLAG2  DBIT    1
                  7
----              8             RSEG    ONCHIP ;begin relocatable data seg
0000              9      TOTAL: DS      1
0001              10     COUNT: DS      1
0002              11     SUM16: DS      2
                  12
----              13            RSEG    EPROM  ;begin relocatable code seg
0000   750000   F      14     BEGIN: MOV     TOTAL,#0
                  15            (continue program)
                  16            END
```

Assembler Controls

- Establish the format of the listing and object files. Controls the look of listing file, without having any effect on the program.
- Can be entered on the invocation line or placed in the source program (preceded with $)
- Primary controls and general controls
- e.g., DATE, INCLUDE, LIST, MACRO(50), XREF

Assembler Controls

NAME	Primary/ General	DEFAULT	Abbrev.	Meaning
DATE(date)	P	DATA()	DA	Places string in header (9 Char. max.)
DEBUG	P	NODEBUG	DB	Outputs debug symbol information to object file
NODEBUG	P	NODEBUG	NODB	Symbol information not placed in object file
EJECT	G	Not applicable	EJ	Continue listing on next page
ERRORPRINT	P	NOERRORPRINT	EP	Designates a file to receive error Messages in addition to the listing file (defaults to console)
NOERRORPRINT	P	NOERRORPRINT	NOEP	Designates that error messages will be printed in listing file only
GEN	P	GENONLY	GO	List only the fully expanded source as if all lines generated by a macro call were already in the source file

Assembler Controls

GENONLY	G	GENONLY	NOGE	List only the original source text in the listing file
INCLUDE(file)	G	Not applicable	IC	Designates a file to be included as part of the program
LIST	G	NOLIST	LI	Print subsequent lines of source code in listing file
NOLIST	G	NOLIST	NOLI	Do not print subsequent lines of source code in listing file
MACRO(mem_percent)	P	MACRO(50)	MR	Evaluate and expand all macro calls. Allocate percentage of free memory for macro processing
NOMACRO	P	MACRO(50)	NOMR	Do not evaluate macro calls
MOD51	P	MOD51	MO	Recognize the 8051-specific predefined special function registers
NOMOD51	P	MOD51	NOMO	Do not recognize 8051-specific predefined special function registers

Assembler Controls

OBJECT(file)	P	OBJECT(source.OBJ)	OJ	Designates file to receive object code
NOOBJECT	P	OBJECT(source.OBJ)	NOOJ	Designates that no object file will be created
PAGING	P	PAGING	PI	Designates that listing file be broken into pages and each will have a header
NOPAGING	P	PAGING	NOPI	Designates that listing file will contain no page breaks
PAGELENGTH	P	PAGELENGTH(60)	PL	Sets maximum number of lines in each page of listing file (range = 10 to 65,536)
PAGEWIDTH	P	PAGEWIDTH(120)	PW	Sets maximum number of characters in each line of listing file (range = 72 to 132)
PRINT(file)	P	PRINT(source.LST)	PR	Designates file to receive source listing
NOPRINT	P	PRINT(source.LST)	NOPR	Designates that no listing file will be created

Assembler Controls

SAVE	G	Not applicable	SA	Stores current control settings from SAVE stack
RESTORE	G	Not applicable	RS	Restores control settings from SAVE stack
REGISTERBANK(rb,..)	P	REGISTERBANK(0)	RB	Indicates one or more banks used in program module
NOREGISTERBANK	P	REGISTERBANK(0)	NORB	Indicates that no register banks are used
SYMBOLS	P	SYMBOLS	SB	Creates a formatted table of all symbols used in program
NOSYMBOLS	P	SYMBOLS	NOSB	Designates that no symbol table is created
TITLE(string)	G	TITLE()	TT	Places a string in all subsequent page headers (max. 60 characters)
WORKFILES(path)	P	Same as source	WF	Designates alternate path for temporary workfiles

Assembler Controls

XREF	P	NOXREF	XR	Creates a cross reference listing of all symbols used in program
NOXREF	P	NOXREF	NOXR	Designates that no cross reference list is created

Linker Operations

- **Intel RL51: links modules into an output file:**

  RL51 input_list [TO output_file] [location_controls]

  input_list: a list of relocatable object modules (files) separated by commas.

  Output_file: the name of the output absolute object module (executable program).

  Location controls: set start addresses for the named segments.

- **e.g., RL51 MAIN.OBJ,MESSAGE.OBJ, SUBROUTINE.OBJ TO EXAMPLE & CODE (EPROM(4000H)) DATA(ONCHIP(30H))**

Linker Operations

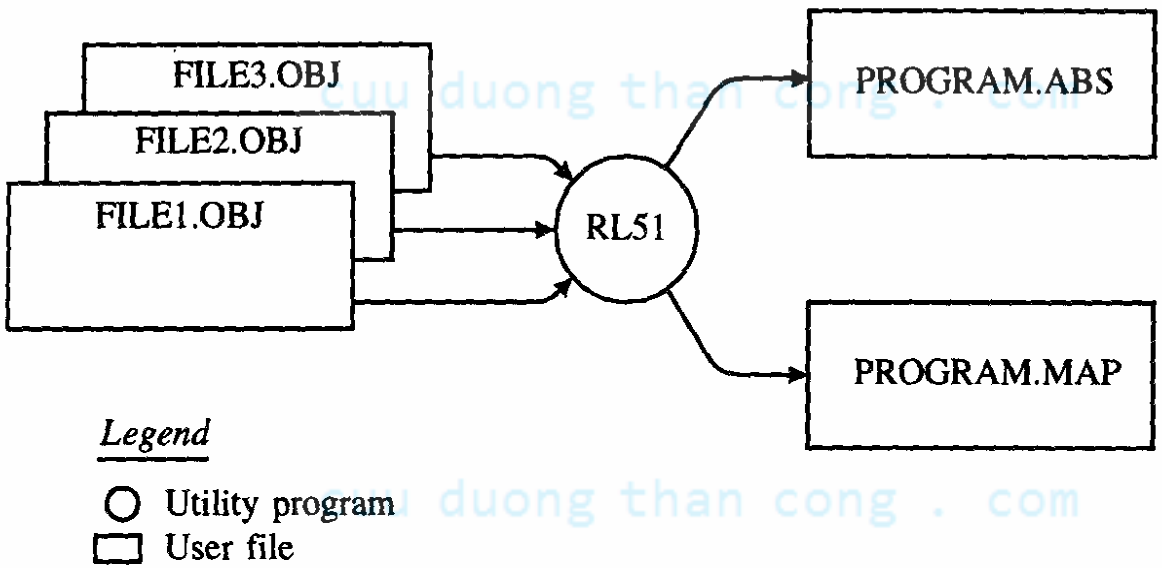


FIGURE 7–7
Linker operation

Annotated Example: ECHO.LST

```
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
  000D                      10     CR      EQU         0DH                 ;CARRIAGE RETURN CODE
                            11     EPROM   SEGMENT     CODE                ;DEFINE SYMBOL "EPROM"
                            12
----                        13             RSEG        EPROM               ;BEGIN CODE SEGMENT
0000 120000     F           14     MAIN:   CALL        INIT                ;INITIALIZE SERIAL PORT
0003 900000     F           15     LOOP:   MOV         DPTR,#PROMPT        ;SEND PROMPT
0006 120000     F           16             CALL        OUTSTR
0009 120000     F           17             CALL        INLINE              ;GET A COMMAND LINE AND
000C 120000     F           18             CALL        OUTLINE             ; ECHO IT BACK
000F 80F2                   19             JMP         LOOP                ;REPEAT
                            20
0011 0D                     21     PROMPT: DB         CR,'Enter a command: ',0
0012 456E7465
0016 72206120
001A 636F6D6D
001E 616E643A
0022 20
0023 00
                            22             END
```

Annotated Example: ECHO.LST

```
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
```

Annotated Example: IO.LST

```
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
```

Annotated Example

```
                        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
```

Annotated Example

```
                                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
```

Annotated Example

```
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

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Annotated Example: ECHO+IO

```
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))DATA(ONCHIP(30H
>> ))


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
```

SYMBOL TABLE FOR EXAMPLE(MAIN)

VALUE	TYPE	NAME
-------	MODULE	MAIN
N:000DH	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:000DH	SYMBOL	CR
C:8000H	SEGMENT	EPROM
C:804AH	SYMBOL	EXIT
C:8066H	SYMBOL	EXIT2
C:8038H	PUBLIC	INCHAR
C:8024H	PUBLIC	INIT
C:804BH	PUBLIC	INLINE
N:0028H	SYMBOL	LENGTH
D:0030H	SEGMENT	ONCHIP
C:8030H	PUBLIC	OUTCHR
C:805BH	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

MACROS

- Macro allows frequently used sections of code to be defined once using a simple mnemonic and used anywhere in the program by inserting the mnemonic.
- ASM51's MPL: string replacement.
- %DEFINE (call_pattern) (macro_body)
- %DEFINE (PUSH_DPTR)
  (PUSH DPH
  PUSH DPL)
- %PUSH_DPTR will be replaced by PUSH DPH and PUSH DPL two instructions in the .LST file

Advantages of Using Macros

- More readable
- The source program is shorter and requires less typing
- Using macros reduces bugs
- Using macros frees the programmer from dealing with low-level details

cuu duong than cong . com

Parameter Passing

- %DEFINE
  (macro_name(parameter_list))
  (macro_body)
- %DEFINE (CMPA# (VALUE))
  (CJNE A,#%VALUE, $ + 3
)
- %CMPA# (20H) = CJNE A,#20H,$+3

Parameter Passing

- JUMP if A is greater than X:
- %DEFINE (macro_name(parameter_list))
  (macro_body)
- %DEFINE (JGT(VALUE,LABEL))

```
    (CJNE    A,#%VALUE+1, $ + 3 ;
     JNC     %LABEL                ;JGT
    )
```

- %JGT('Z',GREATER_THAN)

Local Labels

- %DEFINE (macro_name [(parameter_list)])
  [LOCAL list_of_local_labels]
  (macro_body)
- %DEFINE (DEC_DPTR) LOCAL SKIP
  (DEC DPL
  MOV A,DPL
  CJNE A,#0FFH,%SKIP
  DEC DPH
  %SKIP:)

Local Labels

- %DEC_DPTR =
- ```
        DEC   DPL
        MOV   A,DPL
        CJNE  A,#0FFH, SKIP00
        DEC   DPH
  SKIP00:
  ```
- Side effect: A is used and changed
  ```

Preserve A by Push-Pop

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

Repeat Operations-- Built-in Macros

- %REPEAT (expression) (text)
- %REPEAT (100)
  (NOP
)

Control Flow Operations

- Conditional Assembly in ASM51:
- %IF (expression) THEN (balanced_text) [ELSE (balanced_text)]

```
INTERNAL    EQU   1                ;1=8051 serial I/O drivers
                  .                ;0=8251 serial I/O drivers
            %IF (INTERNAL) THEN
(INCHAR:          .                ;8051 driver
                  .
OUTCHR:           .
                  .
            ) ELSE
(INCHAR:          .                ;8251 driver
                  .
OUTCHR:           .
                  .
            )
```