Microprocessor and Assembly Language

Assembly Language Basics

Prof. Dr. Md. Rabiul Islam
Dept. of Computer Science & Engineering (CSE)
Rajshahi University of Engineering & Technology (RUET)
Bangladesh

Outline

- ❑ Assembly Language – Basic Elements [1]
  - ✓ Statement Syntax: Name Field, Operation Field, Operand Field, Comments
  - ✓ Program Data
  - ✓ Variables
  - ✓ Named Constants
- ❑ Some Basic Instructions [1]
- ❑ Input and Output Instructions [1]
- ❑ An Assembly Program [1]
- ❑ Creating and Running An Assembly Program [1]
- ❑ String Display [1]
- ❑ An Assembly Program to Display String [1]

References

[1] ***Chapter 4*** Yutha Yu and Charles Marut, "Assembly Language Programming and Organization of the IBM PC", McGraw-Hill International Edition, 1992.

Assembly Language – Basic Elements

Statements:

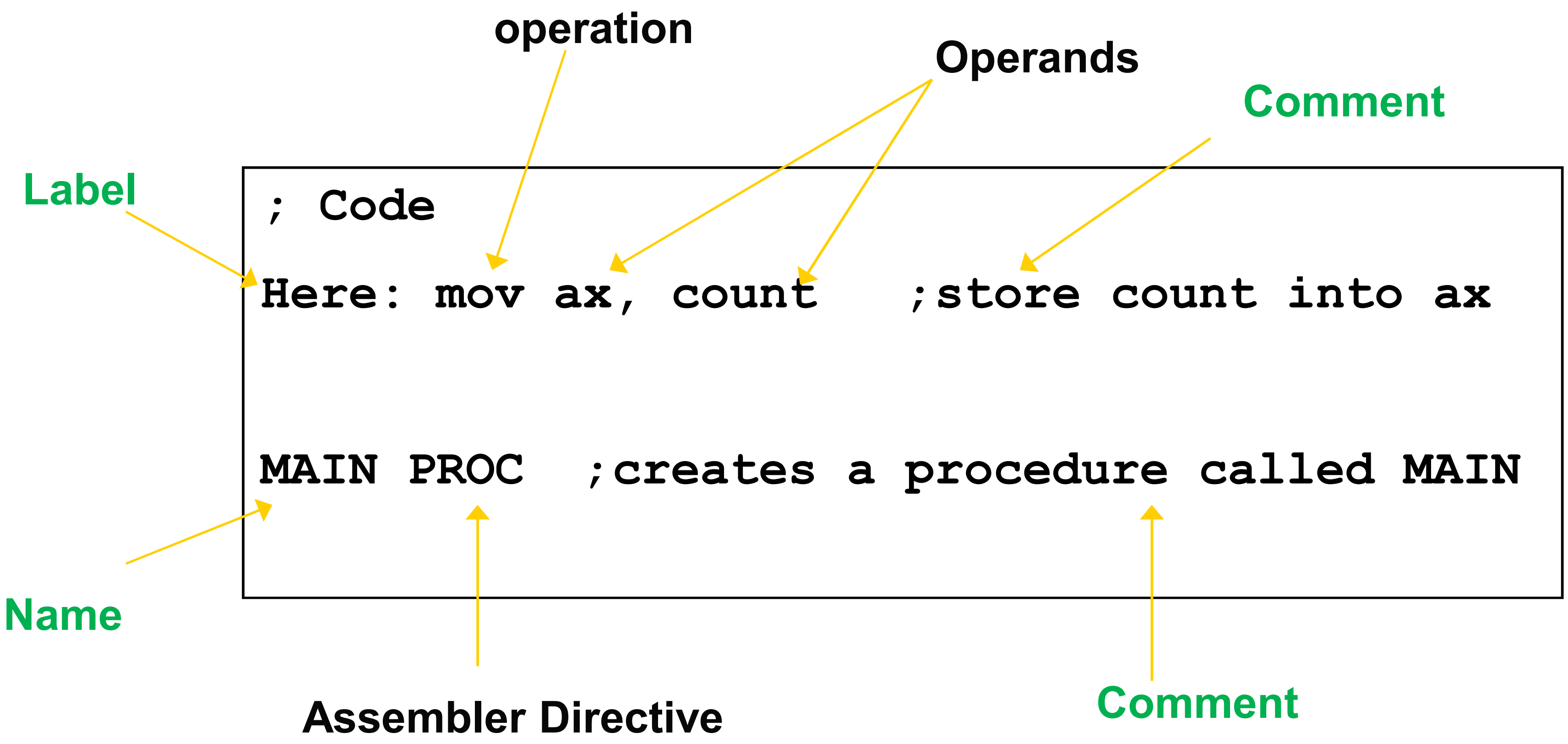
- ❑ Syntax:

  name operation operand(s) comments

  - ✓ name and comment are optional
  - ✓ Number of operands depend on the instruction
  - ✓ One statement per line
  - ✓ At least one blank or tab character must separate the field.
  - ✓ Each statement is either:
  - ✓ Instruction (translated into machine code)
  - ✓ Assembler Directive (instructs the assembler to perform some specific task such as allocating memory space for a variable or creating a procedure)

Assembly Language – Basic Elements

Statement Example:



operation

Operands

Comment

Label

```
; Code
Here: mov ax, count    ;store count into ax


MAIN PROC  ;creates a procedure called MAIN
```

Name

Assembler Directive

Comment

Name/Label Field:

- ✓ The assembler translates names into memory addresses.
- ✓ Names can be 1 to 31 character long and may consist of letter, digit or special characters. If period is used, it must be first character.
- ✓ Embedded blanks are not allowed.
- ✓ May not begin with a digit.
- ✓ Not case sensitive

Examples of legal names	Examples of illegal names
COUNTER_1	TWO WORDS
@character	2abc
.TEST	A45.28
DONE?	YOU&ME

Operation Field: Symbolic operation (Op code):

- ✓ Symbolic op code translated into Machine Language op code
- ✓ ***Examples***: ADD, MOV, SUB
- ✓ In an assembler directive, the operation field represents Pseudo-op code
- ✓ Pseudo-op is not translated into Machine Language op code, it only tells assembler to do something.
- ✓ ***Example***: **PROC** psuedo-op is used to create a procedure

Operand Field:

- ✓ An instruction may have zero, one or more operands.
- ✓ In two-operand instruction, first operand is destination, second operand is source.
- ✓ For an assembler directive, operand field represents more information about the directive
- ✓ ***Examples***
- ✓ NOP ;no operand, does nothing
- ✓ INC AX ;one operand, adds 1 to the contents of AX
- ✓ ADD AX, 2 ;two operands, adds value 2 to the contents of AX

Assembly Language – Basic Elements

Comments:

- Optional
- Marked by semicolon in the beginning
- Ignored by assembler
- Good practice

Assembly Language – Basic Elements

Program Data:

- ✓ Processor operates only on binary data.
- ✓ In assembly language, you can express data in:
    - ✓ Binary
    - ✓ Decimal
    - ✓ Hexadecimal
    - ✓ Characters
- ✓ Numbers
    - ✓ For Hexadecimal, the number must begin with a decimal digit. E.g.: write 0ABCh not only ABCH.
    - ✓ Cannot contain any non-digit character. E.g.: 1,234 not allowed
- ✓ Characters enclosed in single or double quotes.
    - ✓ ASCII codes can be used
    - ✓ No difference in “A” and 41h

Assembly Language – Basic Elements

Program Data:

- Use a radix symbol (suffix) to select binary, octal, decimal, or hexadecimal

```
6A15h          ; hexadecimal
0BAF1h         ; leading zero required
32q            ; octal
1011b          ; binary
35d            ; decimal (default)
```

Variables:

- ✓ Each variable has a data type and is assigned a memory address by the program.
- ✓ Possible Values:
  - ✓ Numeric, String Constant, Constant Expression, ?
  - ✓ **8 Bit Number Range**: Signed (-128 to 127), Unsigned (0-255)
  - ✓ **16 Bit Number Range:** Signed (-32,678 to 32767), Unsigned (0-65,535)
  - ✓ **?** To leave variable uninitialized

Assembly Language – Basic Elements

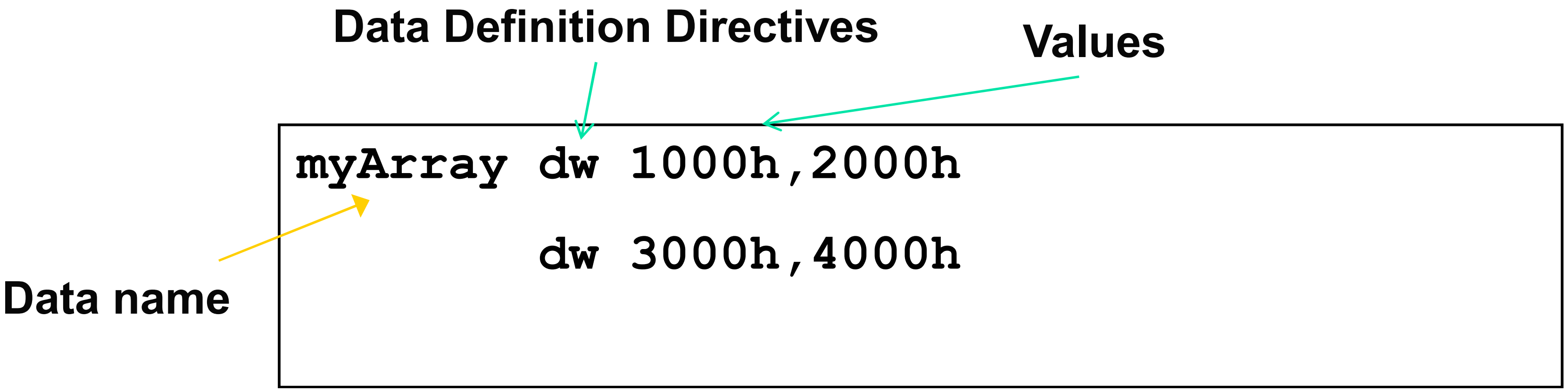
Variables:

- Syntax

  variable_name type initial_value

  variable_name type value1, value2, value3

- Data Definition Directives Or Data Defining Pseudo-ops
  - DB, DW, DD, DQ, DT



```
myArray dw 1000h,2000h
        dw 3000h,4000h
```

Remember: you can skip variable name!

Variables:

Pseudo-ops	Description	Bytes	Examples
DB	Define Byte	1	var1 DB 'A' Var2 DB ? array1 DB 10, 20,30,40
DW	Define Word	2	var2 DW 'AB' array2 DW 1000, 2000
DD	Define Double Word	4	Var3 DD -214743648

Note:

Consider

var2 DW 10h

Still in memory the value saved will be 0010h

Assembly Language – Basic Elements

Arrays:

- Sequence of memory bytes or words
- **Example 1:**

  B_ARRAY DB 10h, 20h, 30h

Symbol	Address	Contents
B_ARRAY	0200h	10h
B_ARRAY+1	0201h	20h
B_ARRAY+2	0202h	30h

***If B_ARRAY is assigned offset address 0200h by assembler**

Example:

- W_ARRAY DW 1000, 40, 29887, 329

***If W_ARRAY is assigned offset address 0300h by assembler**

Symbol	Address	Contents
W_ARRAY	0300h	1000d
W_ARRAY+ 2	0302h	40d
W_ARRAY+ 4	0304h	29887d
W_ARRAY+ 6	0306h	329d

- **High & Low Bytes of a Word**
- WORD1 DW 1234h
- Low Byte = 34h, symbolic address is WORD1
- High Byte = 12h, symbolic address is WORD1+1

Assembly Language – Basic Elements

Character String:

LETTERS DB ‘ABC’

Is equivalent to

LETTERS DB 41h, 42h, 43h

- Assembler differentiates between upper case and lower case.
- Possible to combine characters and numbers.

MSG DB ‘HELLO’, 0Ah, 0Dh, ‘$’

Is equivalent to

MSG DB 48h, 45h, 4Ch, 4Ch, 4Fh, 0Ah, 0Dh, 24h

Example:

- Show how character string "RG 2z" is stored in memory starting at address 0.
- Solution:

Address	Character	ASCII Code (HEX)	ASCII Code (Binary) [Memory Contents]
0	R	52	0101 0010
1	G	47	0100 0111
2	Space	20	0010 0000
3	2	32	0011 0010
4	z	7A	0111 1010

Named Constants:

- Use symbolic name for a constant quantity
- **Syntax**:

  name **EQU** constant

- **Example**:

  LF **EQU** 0Ah

- No memory allocated

Some Basic Instructions

MOV:

- Transfer data
  - Between registers
  - Between register and a memory location
  - Move a no. directly to a register or a memory location
- Syntax

  MOV *destination, source*
- Example

  MOV *AX, WORD1*
- **Difference?**
  - MOV AH, ‘A’
  - MOV AX, ‘A’

Legal Combinations of Operands for MOV :

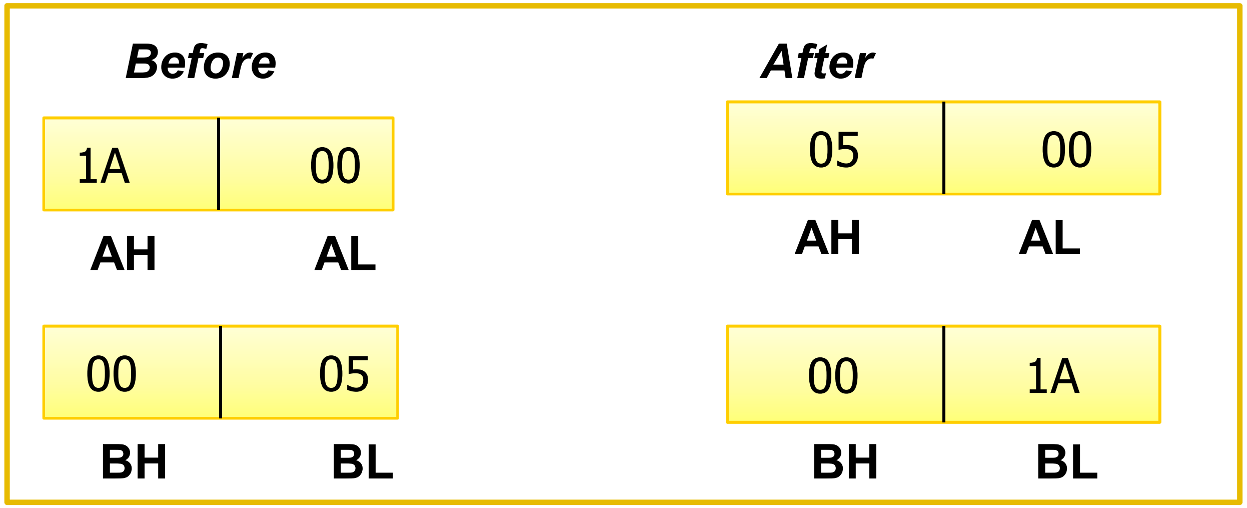
Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
General Register	Segment Register	YES
General Register	Constant	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO
Memory Location	Segment Register	YES
Memory Location	Constant	YES

XCHG:

- Exchange the contents of
  - Two registers
  - Register and a memory location
- Syntax

  XCHG *destination, source*
- Example

  XCHG *AH, BL*



Before

1A 00

AH **AL**

00 05

BH **BL**

After

05 00

AH **AL**

00 1A

BH **BL**

Some Basic Instructions

Legal Combinations of Operands for MOV :

Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO

Some Basic Instructions

ADD Instruction:

- To add contents of:
  - Two registers
  - A register and a memory location
  - A number to a register
  - A number to a memory location
- Example

ADD WORD1, AX

	Before	*After*
AX	01BC	01BC
WORD1	0523	06DF

SUB Instruction:

- To subtract the contents of:
  - Two registers
  - A register and a memory location
  - A number from a register
  - A number from a memory location
- Example

SUB AX, DX

	Before	***After***
AX	0000	FFFF
DX	0001	0001

Some Basic Instructions

Legal Combinations of Operands for ADD & SUB Instructions:

Destination Operand	Source Operand	Legal
General Register	General Register	YES
General Register	Memory Location	YES
General Register	Constant	YES
Memory Location	General Register	YES
Memory Location	Memory Location	NO
Memory Location	Constant	YES

Legal Combinations of Operands for ADD & SUB Instructions:

✓**ADD** BYTE1, BYTE2   ILLEGAL instruction

Solution?

```
        MOV AL, BYTE2
        ADD BYTE1, AL
```

✓**What can be other possible solutions?**

✓**How can you add two word variables?**

INC and DEC:

- **INC** (increment) instruction is used to add 1 to the contents of a register or memory location.
    - Syntax: INC *destination*
    - Example: INC WORD1

- **DEC** (decrement) instruction is used to subtract 1 from the contents of a register or memory location.
    - Syntax: DEC *destination*
    - Example: DEC BYTE1

- Destination can be 8-bit or 16-bits wide.
- Destination can be a register or a memory location.

Some Basic Instructions

INC and DEC:

INC WORD1

	Before	***After***
WORD1	0002	0003

DEC BYTE1

	Before	***After***
BYTE1	FFFE	FFFD

Some Basic Instructions

NEG:

- Used to negate the contents of destination.
- Replace the contents by its 2’s complement.
- Syntax

  NEG *destination*
- Example

  NEG BX

	Before	***After***
BX	0002	FFFE

How?

Some Basic Instructions

Examples:

- Consider instructions: MOV, ADD, SUB, INC, DEC, NEG
- **A** and **B** are two word variables
- Translate statements into assembly language:

Statement	Translation
B = A	MOV AX, A MOV B, AX
A = 5 - A	MOV AX, 5 SUB AX, A MOV AX, A **OR** NEG A ADD A, 5

Some Basic Instructions

Examples:

Statement	Translation
A = B – 2 x A	MOV AX, B SUB AX, A SUB AX, A MOV AX, A

- ❑ **Remember:** Solution not unique!
- ❑ **Be careful!** Word variable or byte variable?

Program Structure

Program Segments:

- Machine Programs consists of
  - Code
  - Data
  - Stack
- Each part occupies a memory segment.
- Same organization is reflected in an assembly language program as **Program Segments**.
- Each program segment is translated into a memory segment by the assembler.

Memory Models:

- Determines the size of data and code a program can have.
- Syntax:

.MODEL memory_model

Model	Description
SMALL	code in one segment, data in one segment
MEDIUM	code in more than one segment, data in one segment
COMPACT	code in one segment, data in more than one segment
LARGE	Both code and data in more than one segments No array larger than 64KB
HUGE	Both code and data in more than one segments array may be larger than 64KB

Stack Segment:

- A block of memory to store stack
- Syntax

  .STACK size

  - Where size is optional and specifies the stack area size in bytes
  - If size is omitted, 1 KB set aside for stack area

- For example:

  .STACK 100h

Program Structure

Code Segment:

- Contains a program’s instructions
- Syntax

.CODE name

  - Where name is optional
  - Do not write name when using SMALL as a memory model

Program Structure

Putting it together:

```
.MODEL SMALL
.STACK 100h
.DATA
    ;data definition go here
.CODE
MAIN PROC
    ;instructions go here
MAIN ENDP
    ;Other procedures go here
END MAIN
```

Input and Output Instructions

Invoke BIOS and DOS Routine:

- The *INT* instruction
  - To invoke a DOS or BIOS routine, the INT (interrupt) instruction is used.
  - It has the format *INT interrupt_number*
  - BIOS routine and DOS routine

INT 21h:

- *INT 21h* may be used to Invoke a large number of DOS functions.

Function	Routine
1	Single-key input
2	Single-character output
9	Character string output

Input and Output Instructions

Function 1: Single-key Input

Input:	AH = 1
Output:	AL = ASCII code if character key b pressed = 0 if non-character key is pressed

✓To invoke the routine, the following instructions is executed:

```
MOV AH, 1  ; input  key  function
INT 21h     ;ASCII  code in  AL
```

Input and Output Instructions

Function 2: Display a Character or Execute a Control Function

Input:	AH = 2 DL = ASCII code of the display character or control character
Output:	AL = ASCII code of the display character or control character

✓To display a character with this function, we put its ASCII code in DL.

✓The following instructions cause a question mark to appear on the screen:

```
MOV AH,2      ;display character function
MOV DL, '?'   ;character is ‘?’
INT  21h       ;display character
```

Function 2: Display a Character or Execute a Control Function

✓Function 2 may also be used to perform· control functions.

✓If DL contains the ASCII code of a control character, *INT 21h* causes the control function to be performed.

ASCII Code	Symbol	Function
7	BEL	Beep (Sounds a tone)
8	BS	Backspace
9	HT	Tab
A	LF	Line feed (New line)
D	CR	Carriage return (Start of current line)

Function 9: Character String Output

Input:	DX = Offset address of the string

✓The string mist end with a '$' character.

✓If the string contains the ASCII code of a control character, the control function is performed.

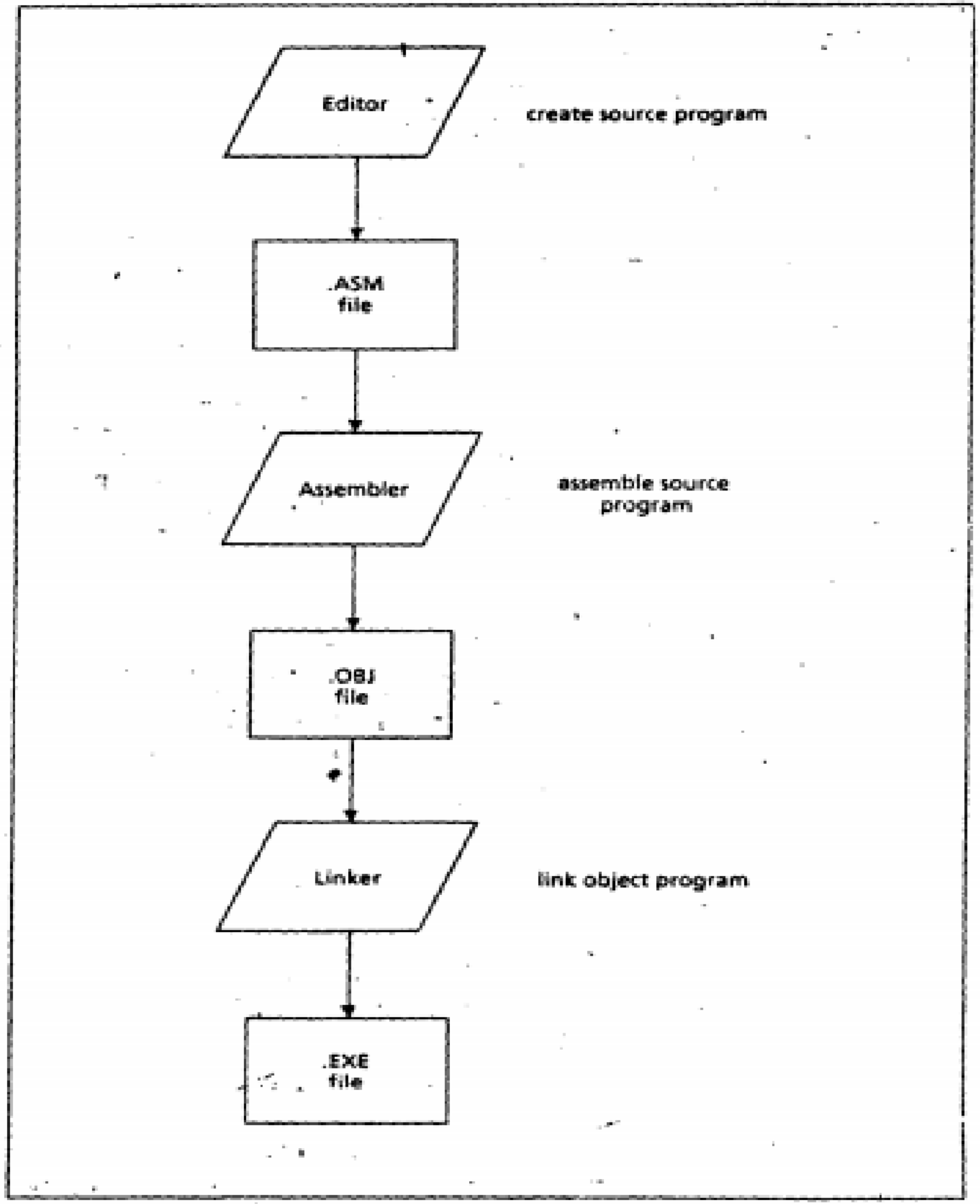
An Assembly Program

```
.MODEL SMALL
.STACK 100h
.DATA
.CODE
MAIN PROC
;display prompt
MOV AH, 2
MOV DL, ‘?’
INT 21h
;input a character
MOV AH,1
INT 21h
MOV BL, AL ;save it in BL
;go to a new line
MOV AH,2 ;display character function
MOV DL, 0DH ; carriage retune
INT 21h ;execute carriage return
MOV DL, 0AH ;line feed
INT 21h ;execute line feed
;display character
MOV DL, BL ;retrieve character
INT 21h ;display character
;return to dos
MOV AH,4CH ;DOS exit function
INT 21h ;exit to DOS
MAIN ENDP
END MAIN
```

Creating and Running an Assembly Program

The following four steps are required to run the assembly program:

1. Use a text editor or word processor to create a source program file.
2. Use an assembler to create a machine language object file.
3. Use the LINK program (see description later) to link one or more object files to create a run file.
4. Execute the run file.



String Display

The LEA Instruction:

LEA stands for Load Effective Address.
INT 2lh, function 9, expects the offset address of the character string to be In DX.
To get it there, we use a new instruction: ***LEA destination, source***
It puts a copy of the source offset address into the destination.
For example,

MSG DB 'HEI.LO!$
LEA DX, MSG **; puts the offset address of the variable MSG into UX.**

String Display

Program Segment Prefix (PSP):

When a program is loaded in memory, DOS prefaces it with a 256-byte program segment prefix (PSP). The PSP contains information about the program. So that programs may access this area, DOS places its segment number in both DS and ES before executing the program. The result is that DS does not contain the segment number of the data segment. To correct this, a program containing a data segment begins with these two instructions:

```
MOV AX,@DATA
MOV DS,AX
```

@Data is the name of the data segment defined by @DATA. The assembler translates the name @DATA into a segment number. Two instructions are needed because a number (the data segment number) may not be moved directly into a segment register.

Example:

```
  LEA  DX,MSG ; get message
  MOV  AH,9 ; display string function
  INT  2lh ;display string
```

An Assembly Program to Display String

```
.MODEL SMALL
.STACK 100h
.DATA
MSG DB ‘HELLO!$’ ;define the message
.CODE
MAIN PROC
;initialize DS
MOV AX,  @DATA
MOV DS, AX ;initialize DS
;display message
LEA DX, MSG ;get message
MOV AH, 9 ;display string function
INT 21h ;display message
;return to dos
MOV AH,4CH ;DOS exit function
INT 21h ;exit to DOS
MAIN ENDP
END MAIN
```