Microprocessor and Assembly Language Lab

Lab Material 2 for CSE 312 (M&AL Lab)

Dr. Shah Murtaza Rashid Al Masud
Associate Prof.
Dept. of CSE, UAP

Machine Language

The operations of the computer's hardware are controlled by its software. When the computer is on, it is always in the process of executing instructions. To fully understand the computer's operations, we must also study its instructions.

Machine Language

A CPU can only execute machine language instructions. As we've seen, they are bit strings. The following is a short machine language program for the IBM PC:

Machine instruction	*Operation*
10100001 00000000 00000000	Fetch the contents of memory word 0 and put it in register AX.
00000101 00000100 00000000	Add 4 to AX.
10100011 00000000 00000000	Store the contents of AX in memory word 0.

As you can well imagine, writing programs in machine language is tedious and subject to error!

Assembly Language

Assembly Language

A more convenient language to use is **assembly language**. In assembly language, we use symbolic names to represent operations, registers, and memory locations. If location 0 is symbolized by A, the preceding program expressed in IBM PC assembly language would look like this:

Assembly language instruction	*Comment*
`MOV AX,A`	`;fetch the contents of ;location A and ;put it in register AX`
`ADD AX,4`	`;add 4 to AX`
`MOV A,AX`	`;move the contents of AX ;into location A`

A program written in assembly language must be converted to machine language before the CPU can execute it. A program called the **assembler** translates each assembly language statement into a single machine language instruction.

Data/Character Representation (ASCII American Standard Code for Information Interchange)

Table 2.5 ASCII Code

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	<CC>	32	20	SP	64	40	@	96	60	`
1	01	<CC>	33	21	!	65	41	A	97	61	a
2	02	<CC>	34	22	"	66	42	B	98	62	b
3	03	<CC>	35	23	#	67	43	C	99	63	c
4	04	<CC>	36	24	$	68	44	D	100	64	d
5	05	<CC>	37	25	%	69	45	E	101	65	e
6	06	<CC>	38	26	&	70	46	F	102	66	f
7	07	<CC>	39	27	'	71	47	G	103	67	g
8	08	<CC>	40	28	(	72	48	H	104	68	h
9	09	<CC>	41	29	)	73	49	I	105	69	i
10	0A	<CC>	42	2A	*	74	4A	J	106	6A	j
11	0B	<CC>	43	2B	+	75	4B	K	107	6B	k
12	0C	<CC>	44	2C	,	76	4C	L	108	6C	l
13	0D	<CC>	45	2D	-	77	4D	M	109	6D	m
14	0E	<CC>	46	2E	.	78	4E	N	110	6E	n
15	0F	<CC>	47	2F	/	79	4F	O	111	6F	o
16	10	<CC>	48	30	0	80	50	P	112	70	p
17	11	<CC>	49	31	1	81	51	Q	113	71	q
18	12	<CC>	50	32	2	82	52	R	114	72	r
19	13	<CC>	51	33	3	83	53	S	115	73	s
20	14	<CC>	52	34	4	84	54	T	116	74	t
21	15	<CC>	53	35	5	85	55	U	117	75	u
22	16	<CC>	54	36	6	86	56	V	118	76	v
23	17	<CC>	55	37	7	87	57	W	119	77	w
24	18	<CC>	56	38	8	88	58	X	120	78	x
25	19	<CC>	57	39	9	89	59	Y	121	79	y
26	1A	<CC>	58	3A	:	90	5A	Z	122	7A	z
27	1B	<CC>	59	3B	;	91	5B	[	123	7B	{
28	1C	<CC>	60	3C	<	92	5C	\	124	7C	|
29	1D	<CC>	61	3D	=	93	5D	]	125	7D	}
30	1E	<CC>	62	3E	>	94	5E	^	126	7E	~
31	1F	<CC>	63	3F	?	95	5F	_	127	7F	<CC>

<CC> denotes a control character
SP = blank space

Data/Character Representation (contd.)

Characters

- Must be enclosed in single or double quotes:
  - “Hello”, ‘Hello’, “A”, ‘B’

- Translated into ASCII code by the assembler:

  ‘A’ has ASCII code 41H

  ‘a’ has ASCII code 61H

  ‘0’ has ASCII code 30H

  Line feed has ASCII code 0AH

  Carriage Return has ASCII code 0DH

  Back Space has ASCII code 08H

  Horizontal tab has ASCII code 09H

Variable Declaration

Data-defining pseudo-ops

- DB define byte
- DW define word
- DD define double word (two consecutive words)

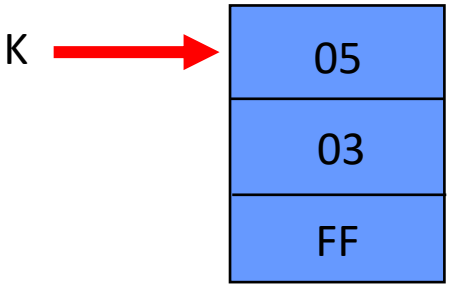
Byte Variables

name DB initial value

- I DB 4 ; define variable I (memory location) with initial value 4
- J DB ? ; define variable J with uninitialized value

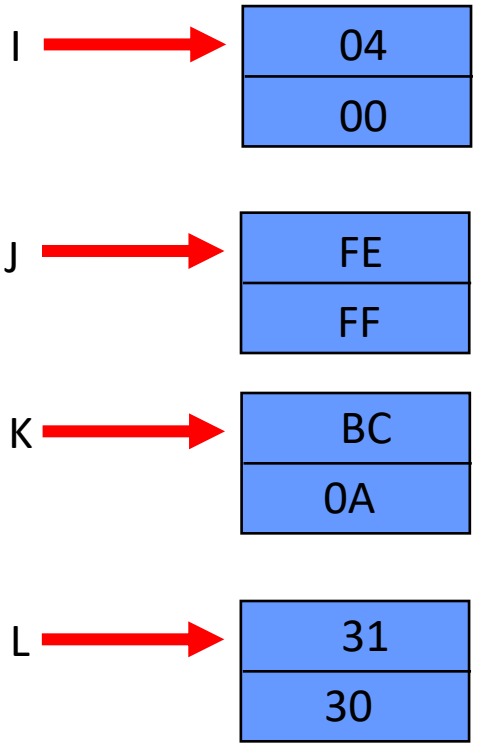
Byte Array

- K DB 5, 3, -1 ; allocates 3 bytes



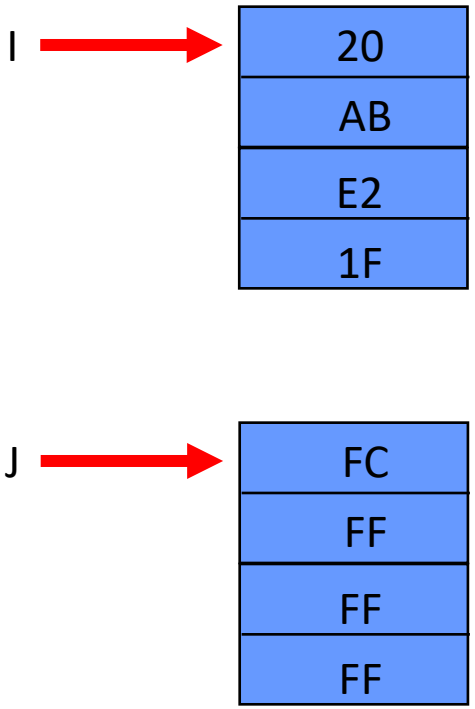
Word Variables

name	DW	initial value
I	DW	4
J	DW	-2
K	DW	0ABCH
L	DW	“01”



Double Word Variables

name	DD	initial value
I	DD	1FE2AB20H
J	DD	-4



Few Basic Instruction

- Over 100 instructions for 8086

- Today we will discuss few most useful ones:
    - MOV
    - ADD
    - SUB
    - INC
    - DEC
    - NEG

MOV Instruction

- The MOV instruction is used to transfer data between registers, between a register and a memory location, or to move a number directly into a register or memory location.
- **Syntax:** MOV destination, source
- **Transfer data between**
  - Two registers
  - A register and a memory location
  - A constant to a register or memory location

```
MOV     AX, WORD1
MOV     AX, BX
MOV     AH, 'A'
```

ADD & SUB Instructions

Syntax:

ADD destination, source ; destination = destination+ source
SUB destination, source ; destination = destination-source

ADD and SUB instructions affect all the flags.

Destination Operand

Source Operand	General register	Memory location
General register	Yes	Yes
Memory location	Yes	**No**
Constant	Yes	Yes

```
ADD  WORD1, AX
SUB  AX, DX
ADD  BL, 5
```

INC & DEC Instructions

- **Syntax:**
  - INC operand ; operand = operand + 1
  - DEC operand ; operand = operand - 1

Operand can be a general register or memory.

INC and DEC instructions affect all the flags.

- **INC destination (memory location)**
- **DEC destination**
- INC WORD1
- DEC BYTE1

NEG instruction

- **Syntax:** NEG operand

- Finds the two's complement of operand.
- Operand can be a general register or memory location.
- NEG instruction affects all flags.

```
NEG   destination (location of memory)
NEG   BX
```

Translation of high-Level Language to Assembly Language

Statement	**Translation**
B = A	`MOV  AX, A; move A into AX` `MOV  B, AX; and then into B`

Translation of high-Level Language to Assembly Language

Statement **Translation**

A = 5 – A

```
MOV  AX, 5; put 5 in AX
SUB  AX, A; AX contains 5 – A
MOV  A, AX; put it in A
```

A = 5 – A

```
NEG  A    ; A = –A
ADD  A, 5 ; A = 5 – A
```

Translation of high-Level Language to Assembly Language

Statement **Translation**

A = B – 2 x A

```
MOV  AX, B; AX has B
SUB  AX, A; AX has B - A
SUB  AX, A; AX has B - 2 x A
MOV  A, AX; move result to A
```

Program Segments

- Machine Programs consists of
  - Code Segment
  - Data Segment
  - Stack Segment

- 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.

PROGRAM STRUCTURE

- Assembly language program occupies code, data and stack segment in memory
- Same organization reflected in assembly language programs as well
- Code data and stack are structured as program segments
- Program segments are translated to memory segments by 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

Data Segment

- All variable definitions
- Use **.DATA** directive

- For Example:

```
.DATA
  WORD1 DW 2
  BYTE1 DB 10h
```

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

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

Putting it all together!

```
.MODEL SMALL
.STACK 100h

.DATA
    ;data definition go here
.CODE
    MAIN PROC  ; Main Procedure starts here
            ;instructions go here
    MAIN ENDP   ; Main Procedure ends here

END MAIN
```

Starting to code: *Template.asm*

```
.MODEL SMALL
.STACK 100H

.DATA ; Variable declarations here

.CODE ; Code starts from here

 MAIN PROC ; Main procedure starts

 MAIN ENDP ; Main procedure ends

END MAIN
```

INPUT AND OUTPUT INSTRUCTIONS

- CPU communicates with the peripherals through IO ports
  - IN and OUT instructions to access the ports directly
    - Used when fast IO is essential
    - Seldom used as
      - Port address varies among compluter models
      - Easier to program IO with service routine

INT

- I/O service routines
  - The **B**asic **I**nput/**O**utput **S**ystem (BIOS) routines
  - The DOS routines
- The **INT (interrupt)** instruction is used to invoke a DOS or BIOS routine.
- `INT 16h`
  - invokes a BIOS routine that performs keyboard input.

INT 21H

- INT 21h may be used to invoke a large number of DOS functions.
- A particular function is requested by placing a function number in the AH register and invoking INT 21h.

I/O Instructions

- The instruction **INT 21H** transfers control to the operating system, to a subprogram that handles I/O operations.

- **AH = 1** is used for single key input
  - The **AL** register stores the input character.
  - AL = ASCII code if character key is pressed
    = 0 if non-character key (arrow key, F1-F10 etc.) is pressed
- **AH = 2** is used for single character output
  - Console shows the value stored in DL
  - DL = ASCII code of the display character.
- **AH = 9** for character string output

  - The string must end with a '$' character.

  - DX must hold the offset address if string.

Details: input

- For character input from keyboard, the number **1** must be stored in the **AH** register. (MOV AH, 1H)
- Then we call the **INT** 21H. (INT 21H)
- The DOS subprogram stores the input in **AL** register.

```
MOV AH, 1     ; calling the input subroutine
INT 21H       ; input goes to AL
MOV BL, AL    ; Saving the input to BL
```

Details: output

- For character input from keyboard, the number **2** must be stored in the **AH** register. (MOV AH, 2H)
- Then we call the **INT** 21H. (INT 21H)
- The DOS subprogram shows the value of **DL** register to the console.

```
MOV DL, BL
MOV AH, 2     ; calling the output subroutine
INT 21H       ; console shows the value stored in DL
```

Details: character string output

- For character string output, DX register must hold the offset address of the character string. For this, use **LEA (Load Effective Address)** instruction.
- LEA destination, source
- Puts a copy of the source offset address into the destination.

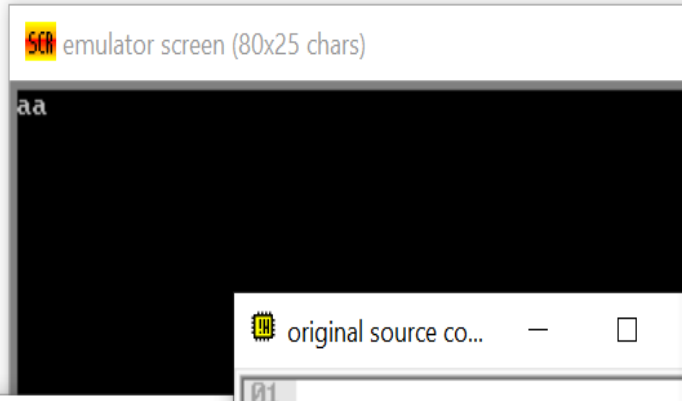
```
LEA DX, MSG ; get message
MOV AH, 9       ; display string function
INT 21H         ; console shows the string
```

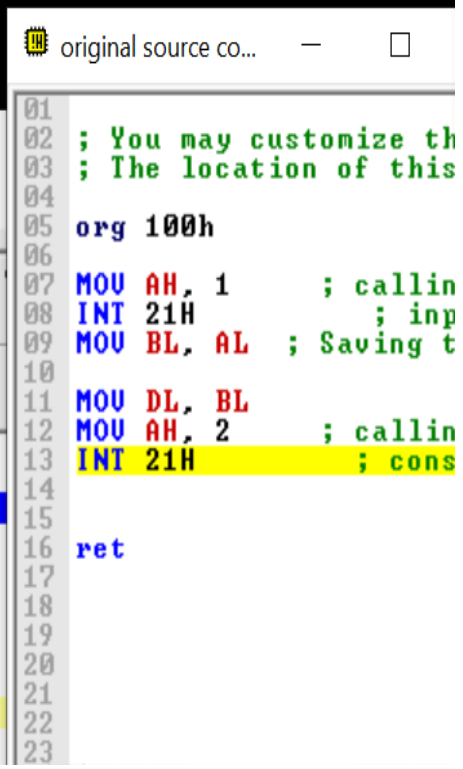
An Example for basic I/O

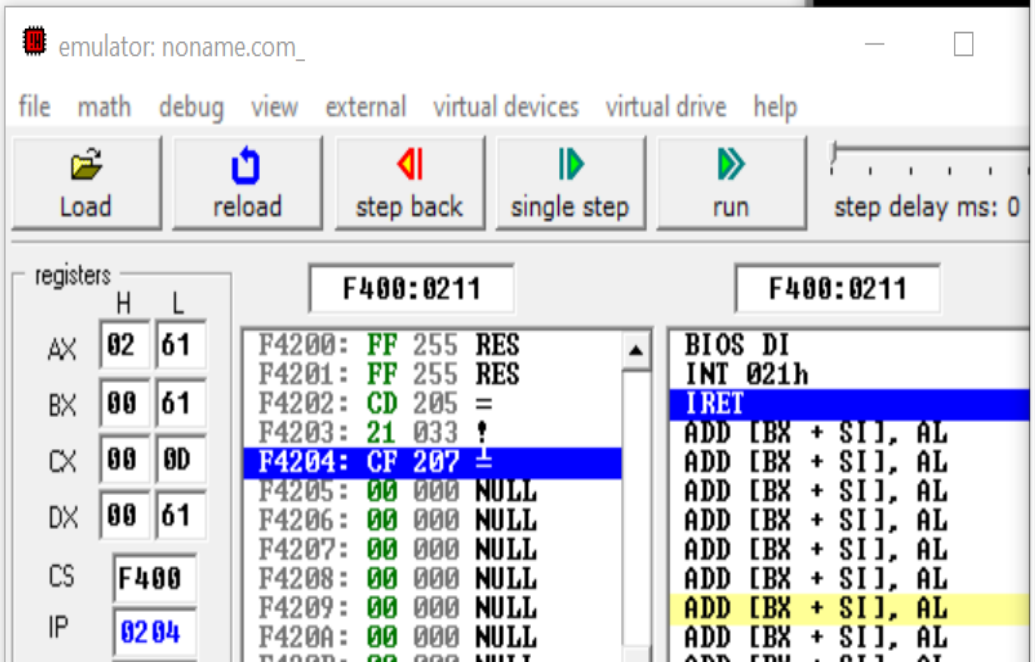
```
MOV AH, 1    ; calling the input subroutine
INT 21H      ; input goes to AL
MOV BL, AL   ; Saving the input to BL

MOV DL, BL
MOV AH, 2    ; calling the output subroutine
INT 21H      ; console shows the value stored in DL
```

An Example for basic I/O

```
; You may customize this and other start-up templates;
; The location of this template is c:\emu8086\inc\0_com_template.txt

org 100h

        MOV AH, 1    ; calling the input subroutine
        INT 21H        ; input goes to AL
        MOV BL, AL   ; Saving the input to BL

        MOV DL, BL
        MOV AH, 2    ; calling the output subroutine
        INT 21H        ; console shows the value stored in DL


ret
```







An Example for basic I/O

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Program Segment Prefix

If a program contains a Data Segment, DS register needs to contain the segment number of the data segment. For this, the program begins with the following 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.

As a number (constant) cannot be moved directly into a segment register (DS), AX register is used to move the number.

Print String Program

```
.MODEL SMALL
.STACK 100H

.DATA
MSG1 DB 'HELLO WORLD!$'
NL DB 0DH, 0AH, '$'

.CODE

 MAIN PROC
   MOV AX, @DATA ; Program segment prefix
   MOV DS, AX      ; DS = data segment register

   LEA DX, MSG1   ; LEA = Load Effective Address
   MOV AH, 9
   INT 21H

   LEA DX, NL   ; LEA = Load Effective Address
   MOV AH, 9
   INT 21H

MAIN ENDP
END MAIN
```

Print String Program

```
.MODEL SMALL
.STACK 100H
.DATA
message db 'EID MUBARAK to Shormi',0DH,0AH,
        db 'EID MUBARAK to Jubaer','$'
.CODE
MAIN   PROC
       mov     ax,@data
       mov     ds,ax

       mov     ah,9h              ; function to di
       lea     dx, message           ; offset of
                                        ; messag
       int     21h                ; Dos Interrupt




       mov     ax,4C00h              ; function t
       int     21h                ; Dos Interrupt
MAIN   ENDP
END    MAIN
```

Print String Program

```
.MODEL SMALL
.STACK 100H
.DATA
message db 'EID MUBARAK to Shormi',0DH,0AH,          ; Message to be displayed
        db 'EID MUBARAK to Jubaer','$'       ; Message to be displayed
.CODE
MAIN  PROC
      mov     ax,@data
      mov     ds,ax

      mov     ah,9h           ; function to display a string
      mov     dx,offset message       ; offset of Message string. This instructio
                                      ; message variable to DX register.
      int     21h             ; Dos Interrupt function (initiate the process)




      mov     ax,4C00h            ; function to terminate
      int     21h             ; Dos Interrupt
MAIN  ENDP
END   MAIN
```

Print String Program

There is **LEA** (Load Effective Address) instruction and alternative **OFFSET** operator.

Both **OFFSET** and **LEA** can be used to get the offset address of the variable.

LEA is more powerful because it also allows you to get the address of an indexed variables. Getting the address of the variable can be very useful in some situations, for example when you need to pass parameters to a procedure.

Code: *add.asm*

To give an idea of what an assembly language program looks like, here is a simple example. The following program adds the contents of two memory locations, symbolized by A and B. The sum is stored in location SUM.

Program Listing PGM1_1.ASM

```
TITLE  PGM1_1: SAMPLE  PROGRAM
.MODEL   SMALL
.STACK   100H
.DATA
A    DW   2
B    DW   5
SUM  DW   ?
.CODE
MAIN   PROC
;initialize DS
       MOV AX,@DATA
       MOV DS,AX
;add  the  numbers
       MOV AX,A              ;AX has A
       ADD AX,B              ;AX has A+B
       MOV SUM,AX            ;SUM = A+B
;exit  to  DOS
       MOV AX,4C00H
       INT 21H
MAIN   ENDP
       END MAIN
```

Code: *add.asm*

Variables are declared in the data segment. Each variable is assigned space in memory and may be initialized. For example, A DW 2 sets aside a memory word for a variable called A and initializes it to 2 (DW stands for "Define Word"). Similarly, B DW 5 sets aside a word for variable B and initializes it to 5 (these initial values were chosen arbitrarily). SUM DW ? sets aside an uninitialized word for SUM.

A program's instructions are placed in the code segment. Instructions are usually organized into units called *procedures*. The preceding program has only one procedure, called MAIN, which begins with the line MAIN PROC and ends with line MAIN ENDP.

The main procedure begins and ends with instructions that are needed to initialize the DS register and to return to the DOS operating system. Their purpose is explained in Chapter 4. The instructions for adding A and B and putting the answer in SUM are as follows:

```
MOV AX,A                ;AX has A
ADD AX,B                ;AX has A+B
MOV SUM,AX              ;SUM = A+B
```

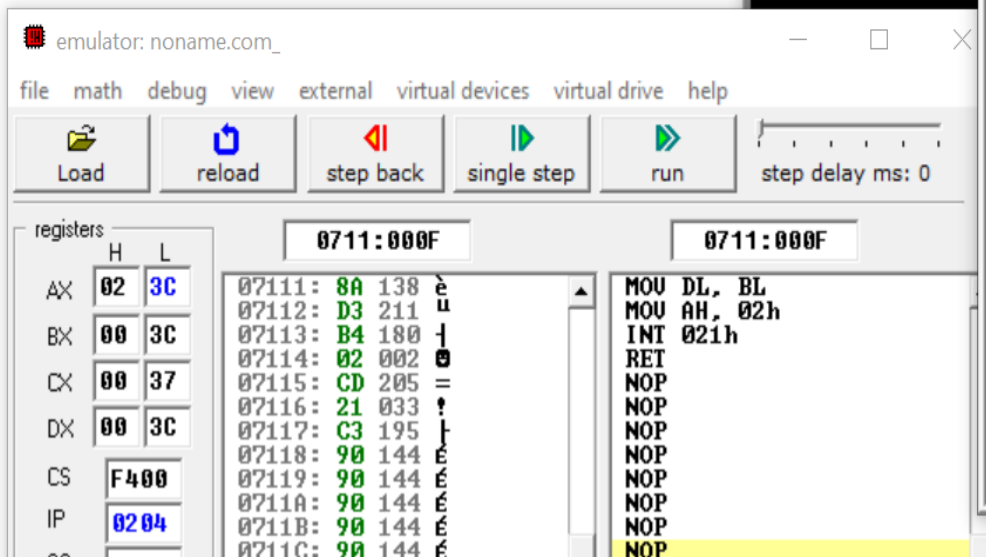
MOV AX,A copies the contents of word A into register AX. ADD AX,B adds the contents of B to it, so that AX now holds the total, Z. MOV SUM,AX stores the answer in variable SUM.

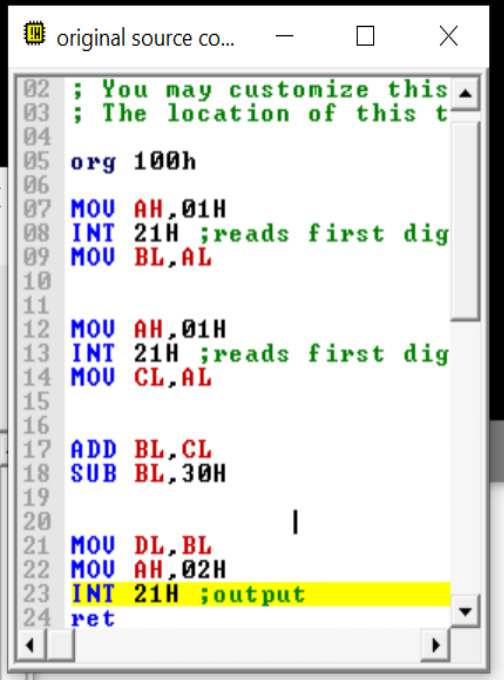
Code: *add sub.asm*

```
.MODEL SMALL
.STACK 100H

.DATA

.CODE


 MAIN PROC

    MOV AH, 1   ; AH = 1 (input)
    INT 21H     ; Call 21st interrept routine
    MOV BL, AL  ; Save input to BL

    MOV AH, 1   ; AH = 1 (input)
    INT 21H     ; Call 21st interrept routine
    MOV CL, AL  ; Save input to CL

    ADD BL, CL  ; BL = BL + CL
    SUB BL, 30H


    MOV DL, BL  ; Output from DL
    MOV AH, 2   ; AH = 2 (output)
    INT 21H


 MAIN ENDP
END MAIN
```

Code: *add sub.asm*

```
; You may customize this and other start-up templates;
; The location of this template is c:\emu8086\inc\0_com_template.txt

org 100h

MOV AH,01H
INT 21H ;reads first digit/input
MOV BL,AL


MOV AH,01H
INT 21H ;reads first digit
MOV CL,AL


ADD BL,CL
SUB BL,30H


MOV DL,BL
MOV AH,02H
INT 21H ;output
ret
```





An Example for basic I/O

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Example PROGRAM

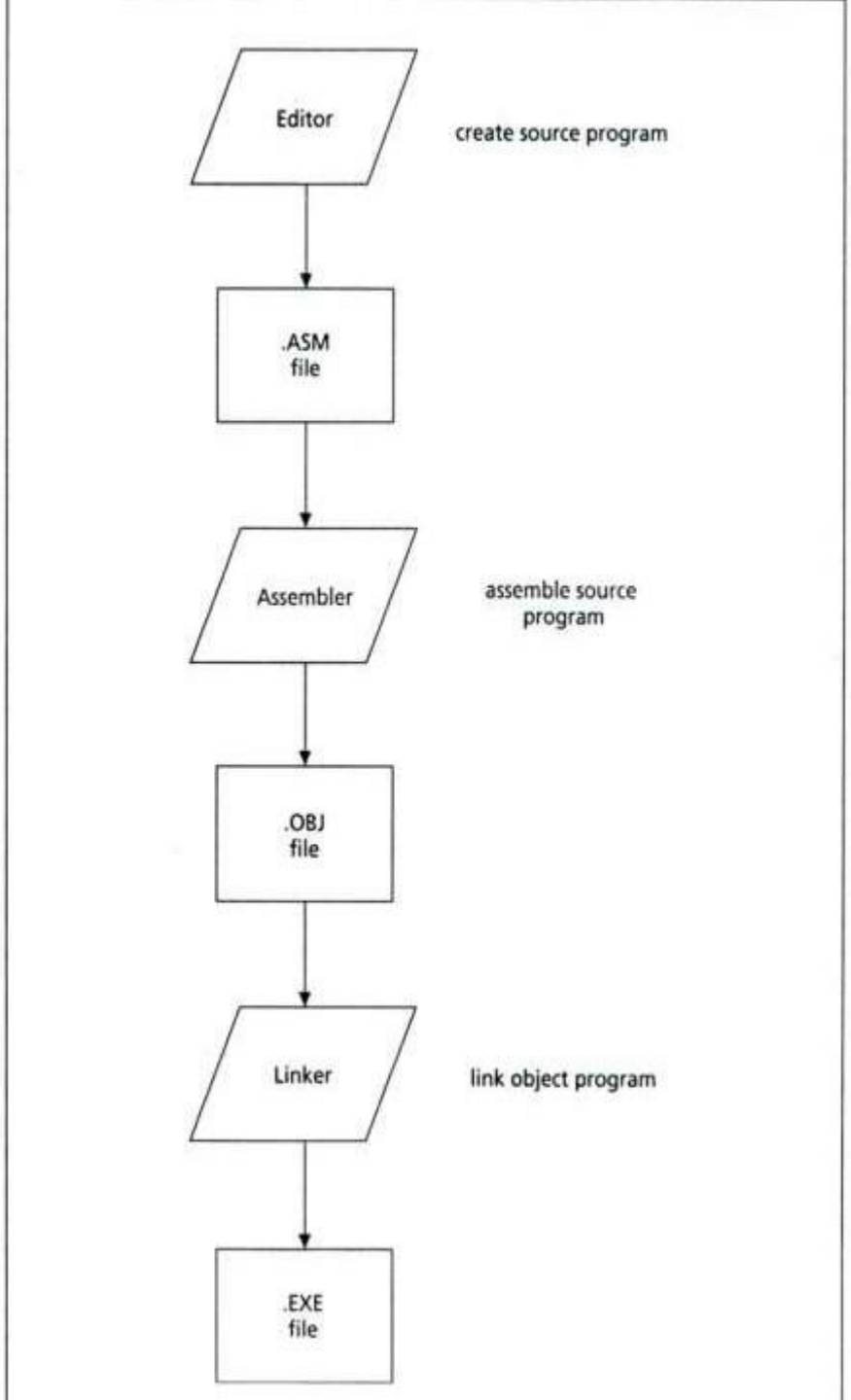
- The following program will read a character from the keyboard and display it at the beginning of the next line.
- The data segment was omitted because no variables were used.
- When a program terminates, it should return control to DOS.
- This can be accomplished by executing INT 21h, function 4Ch.

ASSEMBLY CODE

```
TITLEECHO PROGRAM
.MODEL    SMALL
.STACK    100H
.CODE
MAIN PROC
; display prompt
  MOV AH, 2      ; display character function
  MOV DL, '?'          ; character is '?'
  INT21H         ; display it
; input a character
  MOV AH, 1      ; read character function
  INT     21H          ; character in AL
  MOV BL, AL           ; save it in BL
```

```
; go to a new line
   MOV AH, 2          ; display character function
   MOV DL, 0DH              ; carriage return
   INT 21H            ; execute carriage return
   MOV DL, 0AH              ; line feed
   INT 21H            ; execute line feed
; display character
   MOV DL, BL               ; retrieve character
   INT 21H            ; and display it
; return to DOS
   MOV AH, 4CH              ; DOS exit function
   INT 21H            ; exit to DOS
MAIN  ENDP
   END MAIN
```

PROGRAMMING STEPS



INT 21H, FUNCTION 9: DISPLAY A STRING

Input:

DX = offset address of string.

The string must end with a '$' character.

LEA

- LEA is used to load effective address of a character string.
- **LEA destination, source**
- MSG DB ‘HELLO!$’

```
LEA     DX, MSG    ; get message
MOV     AH, 9      ; display string function
INT     21h        ; display string
```

PROGRAM SEGMENT PREFIX

- When a program is loaded into memory, DOS prefaces it 256 byte PSP which contains information about the program
- DOS places segment no of PSP in DS and ES before executing the program
- To correct this, a program containing a data segment must start with these instructions;

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

Print String Program

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

Any Questions?

Thank You