Experiment 3

Introduction:
In this experiment the students are exposed to the structure of an assembly language program and the definition of data variables and constants.

Objectives:

- Assembly language program structure
- Instructions and Directives
- Data representation
- Variable & constant declaration,
- ADD & SUB instructions

Assembly Language Program Structure

- An assembly language program is a sequence of instructions and directives.
- A p**rogram consists of one statement per line.**
- The general structure of an assembly language program follows the guidelines shown in the following table:

TITLE "Optional: Write here the Title of your program"
.MODEL SMALL
This directive defines the memory model used in the program.

.STACK
This directive specifies the memory space reserved for the stack

.DATA
Assembler directive that reserves a memory space for constants and variables

.CODE
Assembler directive that defines the program instructions

END
Assembler directive that finishes the assembler program

Table 3.1: Assembly Language Program Structure

Instructions and Directives:

Instruction:

- The format of an assembly instruction closely mirrors the structure of a machine instruction
- An instruction is meant for the processor.
- The assembler translates this instruction into machine code

Statement syntax:

Name operation operand(s) ;comment

Examples:

```
MOV AX, BX        ; Load AX to prepare for multiplication
ADD AX, MEM16     ; AX = AX + MEM16
```

Directive:

Pseudo-instructions or **assembler directives** are instructions that are directed to the assembler. They will affect the machine code generated by and will not be translated directly into machine code. Directives are used to declare variables, constants, segments, macros, and procedures as well as supporting conditional assembly

Model Directive:

The model determines the size of the code stack and data segments of the program. Each, of the segments is called a logical segment. Depending on the model used, the code and data segments may be in the same or in different physical segments as shown in table 3.2.

In most of our programs, the model small is sufficient. The tiny model is usually used to generate **command** files (files with extension **.com**). This type of files is smaller in size than the executable files with extension **.exe**.

Memory Model	Size of Code and Data		
	Code	Data	Note
TINY	≤ 64KB	≤ 64KB	Code + Data ≤ 64KB
SMALL	≤ 64KB	≤ 64KB	
MEDIUM	may be ≥ 64KB	≤ 64KB	
COMPACT	≤ 64KB	may be ≥ 64KB	
LARGE	may be ≥ 64KB	may be ≥ 64KB	no array ≥ 64KB
HUGE	may be ≥ 64KB	may be ≥ 64KB	arrays can be ≥ 64KB

Table 3.2: Memory Models

Stack Directive:

- Directive is .stack for stack segment
- Should be declared even if program itself does not use stack needed for subroutine calling (return address) and possibly passing parameters
- May be needed to temporarily save registers or variable content

Memory Segment:

- Directive = **.Data**
- All variables must be declared at this level
- All constants must be defined at this level
- A variable is declared by : DB, DW ,…..
- A constant is defined using: the directive **equ**.

Code Segment:

- The directive **.code** is used for code segment
- The program code resides here

End of Program:

- The Directive **End** is used to tell the assembler that this is the end of the program source file.

Note:

The following sequence of instructions is always used at the beginning of a program to assign the data segment:

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

This sequence may be replaced by the following directive:

.STARTUP

which assigns both DATA and CODE segments, and hence the assembler will issue no warning. However, it should be noted that the program would start at address CS:0017H. The Startup directive occupies the bytes CS:0000 to CS:0017H.

.EXIT
ically, the sequence used to terminate and exit to DOS

```
MOV AH, 4CH
INT 21H
```

can be replaced by the **.EXIT** directive, which has exactly the same effect.

Data Representation:

Numbers:

- 11011 decimal
- 11011B binary
- 64223 decimal
- -21843D decimal
- 1,234 illegal, contains a non-digit character
- 1B4DH hexadecimal number
- 1B4D illegal hex number, does not end with "H"
- FFFFH illegal hex number, does not begin with a digit
- 0FFFFH hexadecimal number

- Signed numbers are represented using 2's complement notation

Characters

- A character must be enclosed in single or double quotes: e.g. "Hello", 'Hello', "A", 'B'
- The ASCII code is used to encode characters
- Examples:
  - '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

Note:

- **The value of a variable, the content of registers or memory is based on the programmer interpretation:**
- **AL = FFH**
  - represents the unsigned number 255
  - represents the signed number -1 (in 2's complement)
- **AH = 30H**
  - represents the decimal number 48
  - represents the character '0'
- **BL = 80H**
  - represents the unsigned number +128
  - represents the signed number -128

Variable Declaration

- Each variable has a type
- Based on its definition, a variable is assigned a memory location
- The location is defined by its address and number of bytes.
- Different data definition directives for different size types of memory

    - DB define byte
    - DW define word
    - DD define double word (two consecutive words)
    - DQ define quad word (four consecutive words)
    - DT define ten bytes (five consecutive words)

- Each pseudo-op can be used to define one or more data items of given type.

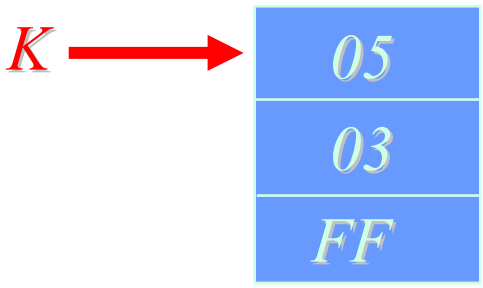
Byte Variables

- The following directive defines a variable of size byte**:**

    - Var_name DB initial value
    - a question mark (?) place in initial value leaves variable non-initialized

Examples:

- I DB 4 define variable I with initial value 4
- J DB ? Define variable J with no initial value
- Name DB "Course" allocate 6 bytes for the variable Name
- K DB 5, 3, -1 allocates 3 bytes

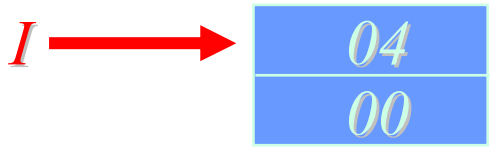
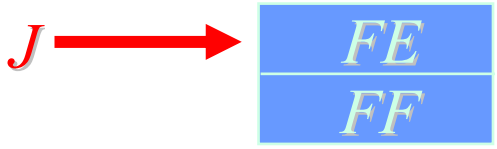
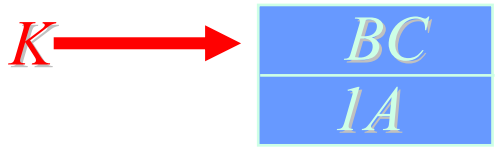
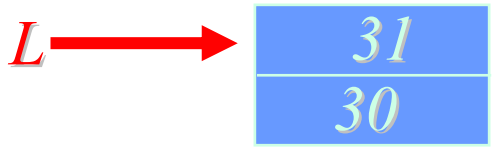


Word Variables:

- The following directive defines a variable of size word**:**

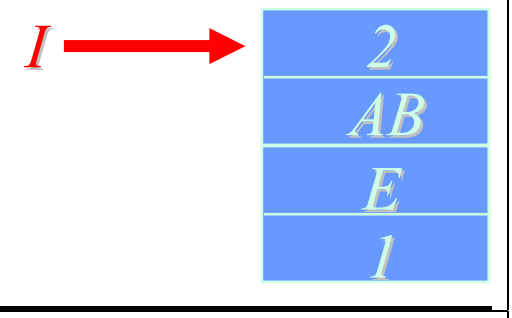
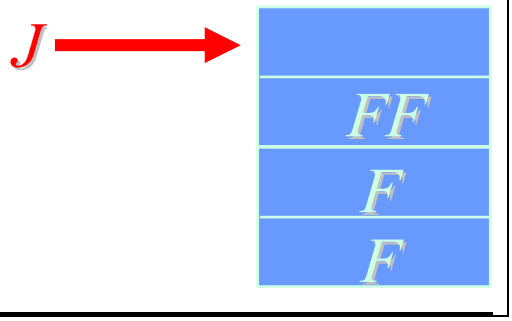
    - **Var_name DW initial value**
    - a question mark (?) place in initial value leaves variable non-initialized

Examples:

I DW 4	
J DW -2	
K DW 1ABCH	
L DW "01"	

Double Word Variables

- The following directive defines a variable of size double word**:**
  - **Var_name DD initial value**

I DD 1FE2AB20H	
J DD -4	

Constant Definition:

- The EQU pseudo-op is used to assign a name to a constant
- Syntax:

  Cst_name EQU Cst_Value

- No memory allocated for EQU names.
- Makes assembly language easier to understand

Examples:

Example 1:

```
MOV DL, 0AH
```

Can be replaced by:

```
LF          EQU         0AH
MOV DL, LF
```

Example 2:

```
MSG DB "Type your name"
```

Can be replaced by:

```
PROMPT      EQU         "Type your name"
MSG DB PROMPT
```

ASCII Table

binary	MSN	0000		0001		0010		0011		0100		0101		0110		0111	
LSN	hex	*0*		*1*		*2*		*3*		*4*		*5*		*6*		*7*	
0000	*0*	NUL	0 00	DLE	16 10	SP	32 20	0	48 30	@	64 40	P	80 50	`	96 60	p	112 70
0001	*1*	SOH	1 01	XON (DC1)	17 11	!	33 21	1	49 31	A	65 41	Q	81 51	a	97 61	q	113 71
0010	*2*	STX	2 02	DC2	18 12	"	34 22	2	50 32	B	66 42	R	82 52	b	98 62	r	114 72
0011	*3*	ETX	3 03	XOFF (DC2)	19 13	#	35 23	3	51 33	C	67 43	S	83 53	c	99 63	s	115 73
0100	*4*	EOT	4 04	DC4	20 14	$	36 24	4	52 34	D	68 44	T	84 54	d	100 64	t	116 74
0101	*5*	ENQ	5 05	NAK	21 15	%	37 25	5	53 35	E	69 45	U	85 55	e	101 65	u	117 75
0110	*6*	ACK	6 06	SYN	22 16	&	38 26	6	54 36	F	70 46	V	86 56	f	102 66	v	118 76
0111	*7*	BEL	7 07	ETB	23 17	'	39 27	7	55 37	G	71 47	W	87 57	g	103 67	w	119 77
1000	*8*	BS	8 08	CAN	24 18	(	40 28	8	56 38	H	72 48	X	88 58	h	104 68	x	120 78
1001	*9*	HT	9 09	EM	25 19	)	41 29	9	57 39	I	73 49	Y	89 59	i	105 69	y	121 79
1010	*A*	LF	10 0A	SUB	26 1A	*	42 2A	:	58 3A	J	74 4A	Z	90 5A	j	106 6A	z	122 7A
1011	*B*	VT	11 0B	ESC	27 1B	+	43 2B	;	59 3B	K	75 4B	[	91 5B	k	107 6B	{	123 7B
1100	*C*	FF	12 0C	FS	28 1C	,	44 2C	<	60 3C	L	76 4C	\	92 5C	l	108 6C	|	124 7C
1101	*D*	CR	13 0D	GS	29 1D	-	45 2D	=	61 3D	M	77 4D	]	93 5D	m	109 6D	}	125 7D
1110	*E*	SO	14 0E	RS	30 1E	.	46 2E	>	62 3E	N	78 4E	^	94 5E	n	110 6E	~	126 7E
1111	*F*	SI	15 0F	US	31 1F	/	47 2F	?	63 3F	O	79 4F	_	95 5F	o	111 6F	DEL	127 7F

Example on the use of the ASCII table

Character	Column #	Row #	Code (H)	Code (binary)
a	6	1	**61H**	
A	4	1	**41H**	
β	E	1	**E1H**	
%	2	5	**25H**	

Table 3.3: Using the ASCII table:

ADD & SUB instructions:

Type	Inst.	Example	Meaning	Flags Affected					
				OF	SF	ZF	AF	PF	CF
Addition	**ADD**	**ADD AX, 7BH**	**AX ← AX + 7B**	*	*	*	*	*	*
	ADC	**ADC AX, 7BH**	**AX ← AX + 7B +CF**	*	*	*	*	*	*
	INC	**INC [BX]**	**[BX]←[BX]+1**	*	*	*	*	*	-
	DAA	DAA		?	*	*	*	*	*
Subtraction	**SUB**	**SUB CL,AH**	**CL ← CL – AH**	*	*	*	*	*	*
	SBB	**SBB CL,AH**	**CL ← CL – AH – CF**	*	*	*	*	*	*
	DEC	**DEC DAT**	**[DAT] ← [DAT] – 1**	*	*	*	*	*	-
	DAS	DAS		?	*	*	*	*	*
	NEG	NEG CX	CX ← 0 – CX	*	*	*	*	*	*

Table 3. 4:: Summary of add and sub instructions

Exercises

Program 1: A Case Conversion Program

Write a program that prompts the user to enter a lowercase letter, and on next line displays another message with letter in uppercase.

- Enter a lowercase letter: a
- In upper case it is: A

Title "Program Small to Upper Case Conversion"

```
.Model Small
.Stack 100
.DATA
    CR EQU 0DH
    LF EQU 0AH
    MSG1 DB 'Enter a lower case letter: $'
    MSG2 DB CR, LF, 'In upper case it is: '
    Char   DB  ?, '$'

.CODE
.STARTUP            ; initialize data segment
    LEA DX, MSG1    ; display first message
    MOV AH, 9
    INT 21H
    MOV AH, 1               ; read character
    INT 21H
    SUB AL, 20H     ; convert it to upper case
    MOV CHAR, AL    ; and store it
    LEA DX, MSG2    ; display second message and
    MOV AH, 9               ; uppercase letter
    INT 21H
.EXIT               ; return to DOS

END
```

Program 2: A Case Conversion Program 2

Write a program that prompts the user to enter an uppercase letter, and on the next line displays another message with letter in lowercase.

Program 3:

Write a program that reads small characters from the keyboard and converts them online to uppercase ones. Use the following to make your program loop. Also use function 08 to read a character without echo.

```
next:   ……          ; read character
        ……          ; convert
        ……          ; display
        Loop next
```