

## Lab # 03

## Assembly Language Program Structure

An assembly language (or assembler language) is a low-level programming language for a computer, or other programmable device, in which there is a very strong correspondence between the language and the architecture's machine code instructions. Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

## How Does Assembly Language Relate to Machine Language?

Machine language is a numeric language specifically understood by a computer's processor (the CPU). All x86 processors understand a common machine language. Assembly language consists of statements written with short mnemonics, such as ADD, MOV, SUB, and CALL. Assembly language has a one-to-one relationship with machine language; Each assembly language instruction corresponds to a single machine-language instruction.

## How Does C++ and Java Relate to Assembly Language?

High-level languages such as C++ and Java have a one-to-many relationship with assembly language and machine language. A single statement in C++ expands into multiple assembly language or machine instructions. We can show how C++ statements expand into machine code. Most people cannot read raw machine code, so we will use its closest relative, assembly language.

## Is Assembly Language Portable?

A language whose source programs can be compiled and run on a wide variety of computer systems is said to be portable. A C++ program, for example, should compile and run on just about any computer, unless it makes specific references to library functions that exist under a single operating system. A major feature of the Java language is that compiled programs run on nearly any computer system.

Assembly language is not portable because it is designed for a specific processor family. There are a number of different assembly languages widely used today, each based on a processor family. Some well-known processor families are Motorola 68x00, x86, SUN Sparc, Vax, and IBM-370. The instructions in assembly language may directly match the computer's architecture or they may be translated during execution by a program inside the processor known as a microcode interpreter.

### Assembly Language Syntax:

Name: operation      operand (s) ;comment

#### Name field

Assembler translate name into memory addresses. It can be 31 characters long. The NAME field allows the program to ref of code by name.

Examples of legal names

- COUNTER1
- @character
- SUM\_OF\_DIGITS • .TEST

Examples of illegal names

- TWO WORDS
- 2abc
- A45.28

#### Operation field

It contains symbolic operation code (opcode) called "mnemonics"(MOV, ADD, e.t.c.). The mnemonic (instruction) and operands together accomplish the tasks for which program was written. The assembler translates mnemonics into machine language opcode.

#### Operand field

It specifies the data that are to be acted on by the operation. An instruction may have a zero, one or two operands.

Examples

- NOP
- INC AX
- ADD AX, 2

#### Comment field

A semicolon marks the beginning of a comment. Good programming practice dictates comment on every line Examples

- MOV CX, 0 ; move 0 to CX
- MOV CX, 0 ; CX counts terms, initially 0

### ✓ Program Structure:

The machine language programs consist of code, data and stack. Each part occupies a memory segment. The same organization is reflected in an assembly language program. This time, the code, data and stack structured as program segments. Each program segment is translated into a memory segment by the assembler.

### ✓ Memory Models:

The size of code and data in a program can have determined by specifying a memory model using the **.MODEL** directive. The syntax is **.MODEL memory\_model**. The most frequently used memory models are **SMALL**, **MEDIUM**, **COMPACT** and **LARGE**. They are described in table below. Unless there is a lot of code or data, the appropriate model is **SMALL**. The **.MODEL** directive should come before any segment definition.

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	Code in more than one segment Data in more than one segment No array larger than 64KB
HUGE	Code in more than one segment Data in more than one segment Arrays may be larger than 64KB

#### Stack Segment:

The purpose of the stack segment declaration is to set aside a block of memory (the stack area) to store the stack. The declaration syntax is **.STACK size**. For example, **.STACK 100H** sets aside 100h bytes for the stack area (a reasonable size for most applications). If size is omitted, 1KB is set aside for the stack- area.

#### Data Segment:

A program's data segment contains all the variable definitions. Constant definitions are often made here as well, but they may be placed elsewhere in the program since no memory allocation is involved. To declare a data segment, we use the directive **.DATA**, followed by variable and constant declaration. For example,

```
.DATA
Word1 Dw 2
Msg Db "This Is A Message"
```



**Code Segment:**

The code segment contains a program's instructions. The declaration syntax is `.CODE`

Here name is the optional name of the segment (there is no need for a name in a `SMALL` program, because the assembler will generate an error). Inside a code segment, instructions are organized as procedures. The simplest procedure definition is:

Name PROC ;body of the procedure

Name ENDP ;where name is the name of the procedure;

PROC and ENDP are pseudo-ops that delineate the procedure. Here is an example of a code segment definition:

```
.CODE
MAIN PROC
; main procedure instructions
MAIN ENDP
; other procedures go here
End Main
```

```
.MODEL SMALL
.STACK 100H
.DATA
; data definitions go here
.CODE
MAIN PROC
; instructions go here
MAIN ENDP
;other procedures go here
END MAIN
```

;The last line in the program should be the `END` directive, followed by name of the main procedure

**ASCII Character Chart** ASCII, American Standard Code for Information Interchange, is a scheme used for assigning numeric values to punctuation marks, spaces, numbers and other characters. ASCII uses 7 bits to represent characters. The values 000 0000 through 111 1111 (00 through 7F) are used giving ASCII the ability to represent 128 different characters. An extended version of ASCII assigns characters from 80 through FF.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00		16	10	P	32	20	R	48	30	S	64	40	T	80	50	U
1	01	Space	17	11	Q	33	21	S	49	31	T	65	41	V	81	51	W
2	02	!"#\$%	18	12	R	34	22	T	50	32	U	66	42	X	82	52	X
3	03	&'()*+,-	19	13	S	35	23	U	51	33	V	67	43	Y	83	53	Z
4	04	./:;<=>?	20	14	T	36	24	V	52	34	W	68	44	Z	84	54	[
5	05	at\b	21	15	U	37	25	W	53	35	X	69	45	{	85	55	^
6	06	cn	22	16	V	38	26	X	54	36	Y	70	46		86	56	_
7	07	de	23	17	W	39	27	Y	55	37	Z	71	47	~	87	57	`
8	08	ef	24	18	X	40	28	Z	56	38	[	72	48		88	58	
9	09	gh	25	19	Y	41	29	[	57	39	]	73	49		89	59	
10	0A	ij	26	1A	Z	42	2A	]	58	40	^	74	4A		90	5A	
11	0B	kl	27	1B		43	2B	^	59	41	_	75	4B		91	5B	
12	0C	mn	28	1C		44	2C	_	60	42		76	4C		92	5C	
13	0D	op	29	1D		45	2D		61	43		77	4D		93	5D	
14	0E	qr	30	1E		46	2E		62	44		78	4E		94	5E	
15	0F	st	31	1F		47	2F		63	45		79	4F		95	5F	
16	10	uv	32	20		48	30		64	46		80	50		96	60	
17	11	wx	33	21		49	31		65	47		81	51		97	61	
18	12	yz	34	22		50	32		66	48		82	52		98	62	
19	13		35	23		51	33		67	49		83	53		99	63	
20	14		36	24		52	34		68	50		84	54				
21	15		37	25		53	35		69	51		85	55				
22	16		38	26		54	36		70	52		86	56				
23	17		39	27		55	37		71	53		87	57				
24	18		40	28		56	38		72	54		88	58				
25	19		41	29		57	39		73	55		89	59				
26	1A		42	2A		58	40		74	56		90	5A				
27	1B		43	2B		59	41		75	57		91	5B				
28	1C		44	2C		60	42		76	58		92	5C				
29	1D		45	2D		61	43		77	59		93	5D				
30	1E		46	2E		62	44		78	5A		94	5E				
31	1F		47	2F		63	45		79	5B		95	5F				

### Disk operating system (DOS) routines

INT 21H is used to invoke a large number of DOS function. The type of called function is specified by pulling a number in AH register. For example

AH=1     input with echo

AH=2     single-character output

AH=9     character string output

AH=8     single-key input without echo

AH=0Ah    character string input

**Single-Key Input with Echo:** Output: AL= ASCII code if character key is pressed, otherwise 0.

MOV AH,1

INT 21 ; read character will store in AL register **Single –**

**Key Input without Echo:**

MOV AH,8

INT 21 ; read character will store in AL register **Single-Character**

**Output:** DL= ASCII code of character to be displayed.

MOV AH,2

MOV DL,"?"

INT 21h ; displaying character ? **Input a**

**String:**

MOV AH,0A

INT 21

**Display a String:** DX= offset address of a string. String must end with a „\$“ character.

LEA DX,n1

MOV AH,9

INT 21

**EXERCISE:**

Q1 Write down the CODE of following scenario

- Input a one-digit number from keyboard
- Save that number to DL register
- Display the number you have entered in new line

```
.MODEL SMALL
.STACK 100h
.CODE
MAIN PROC
    MOV AH, 1
    INT 21H

    MOV DL, AL

    CALL NEWLINE

    MOV AH, 2
    INT 21H

    MOV AH, 4Ch
    INT 21H

MAIN ENDP

NEWLINE PROC
    MOV DL, 0Dh
    MOV AH, 2
    INT 21H

    MOV DL, 0Ah
    MOV AH, 2
    INT 21H
    RET
NEWLINE ENDP

END MAIN
```

OUTPUT



The screenshot shows an x86 emulator interface. At the top, a list of assembly instructions is displayed:

```

11 MOV AH, 2          ; BIOS i
12 INT 21H           ; Print
13
14 MOV AH, 4Ch        ; Exit p
15 INT 21H
16
17 MAIN ENDP
18
19 NEULINE PROC       ; Carria
20 MOV DL, 0Dh
21 MOV AH, 2
22 INT 21H
23
24 MOV DL, 0Ah        ; Line f
25 MOV AH, 2
26 INT 21H
27 RET
28 NEULINE ENDP

```

Below the code, a 'watch' window shows the value of register DX. It displays the value in hex (00 0A), bin (00000000 00001010), oct (000 012), and decimal 8 bit (0 10), signed (0 10), and decimal 16 bit (10 10).

The 'registers' window shows the state of various registers:

Register	H	L
AX	4C	0A
BX	00	00
CX	01	1E
DX	00	0A
CS	F400	
IP	0204	
SS	0710	
SP	00FA	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

The 'memory' window shows the contents of memory addresses starting from F400:0204:

Address	Value	Comment
F4200	FF 255	RES
F4201	FF 255	RES
F4202	CD 205	=
F4203	21 033	!
F4204	0F 202	!
F4205	00 000	NULL
F4206	00 000	NULL
F4207	00 000	NULL
F4208	00 000	NULL
F4209	00 000	NULL
F420A	00 000	NULL
F420B	00 000	NULL
F420C	00 000	NULL
F420D	00 000	NULL
F420E	00 000	NULL
F420F	00 000	NULL
F4210	00 000	NULL
F4211	00 000	NULL
F4212	00 000	NULL
F4213	00 000	NULL
F4214	00 000	NULL
F4215	00 000	NULL

The 'BIOS' window shows the contents of BIOS memory addresses starting from 021h:

Address	Value	Comment
021h	INT 021h	
022h	INT 021h	
023h	INT 021h	
024h	INT 021h	
025h	INT 021h	
026h	INT 021h	
027h	INT 021h	
028h	INT 021h	
029h	INT 021h	
02Ah	INT 021h	
02Bh	INT 021h	
02Ch	INT 021h	
02Dh	INT 021h	
02Eh	INT 021h	
02Fh	INT 021h	
030h	INT 021h	
031h	INT 021h	
032h	INT 021h	
033h	INT 021h	
034h	INT 021h	
035h	INT 021h	
036h	INT 021h	
037h	INT 021h	
038h	INT 021h	
039h	INT 021h	
03Ah	INT 021h	
03Bh	INT 021h	
03Ch	INT 021h	
03Dh	INT 021h	
03Eh	INT 021h	
03Fh	INT 021h	

Q2 Write down the CODE to input First Letter of your name and NOT echo.

```

.MODEL SMALL
.STACK 100h
.CODE
MAIN PROC
MOV AH, 0
INT 16H

```

MOV DL, AL

CALL NEWLINE

MOV AH, 2

INT 21H

MOV AH, 4Ch

INT 21H

MAIN ENDP

NEWLINE PROC

MOV DL, 0Dh

MOV AH, 2

INT 21H

MOV DL, 0Ah

MOV AH, 2

INT 21H

RET

NEWLINE ENDP

END MAINQ3 Write down the CODE to Display your name through ASCII Code (One character @ a time) each new character should display in new line.

.MODEL SMALL

.STACK 100h

.CODE

MAIN PROC

MOV DL, 46h ; 'F'

MOV AH, 2

INT 21H

CALL NEWLINE

MOV DL, 61h ; 'a'

MOV AH, 2

INT 21H

CALL NEWLINE

MOV DL, 74h ; 't'



```
MOV AH, 2  
INT 21H  
CALL NEWLINE
```

```
MOV DL, 69h ; 'i'  
MOV AH, 2  
INT 21H  
CALL NEWLINE
```

```
MOV DL, 6Dh ; 'm'  
MOV AH, 2  
INT 21H  
CALL NEWLINE
```

```
MOV DL, 61h ; 'a'  
MOV AH, 2  
INT 21H  
CALL NEWLINE
```

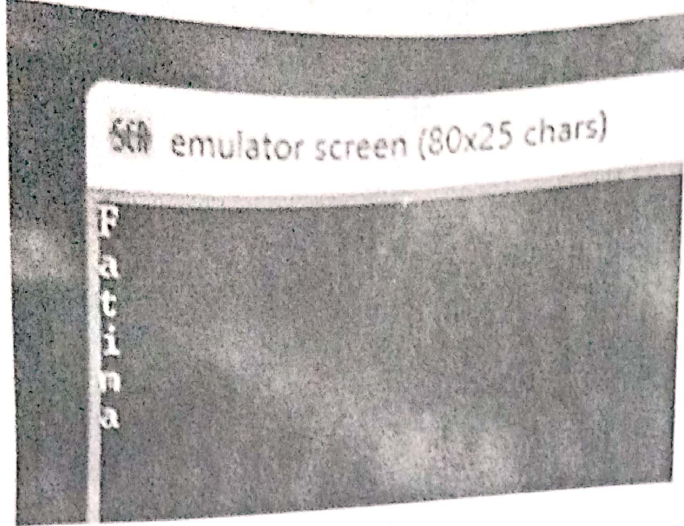
```
MOV AH, 4Ch  
INT 21H
```

```
MAIN ENDP
```

```
NEWLINE PROC  
MOV DL, 0Dh  
MOV AH, 2  
INT 21H
```

```
MOV DL, 0Ah  
MOV AH, 2  
INT 21H  
RET  
NEWLINE ENDP
```

```
END MAIN
```



Q4 Design first letter of your name through suitable ASCII codes.

```
.MODEL SMALL
.STACK 100h

.DATA
line1 db '*****', 0Ah, 0Dh, '$'
line2 db '*****', 0Ah, 0Dh, '$'
line3 db '***', 0Ah, 0Dh, '$'
line4 db '***', 0Ah, 0Dh, '$'
line5 db '*****', 0Ah, 0Dh, '$'
line6 db '*****', 0Ah, 0Dh, '$'
line7 db '***', 0Ah, 0Dh, '$'
line8 db '***', 0Ah, 0Dh, '$'
line9 db '***', 0Ah, 0Dh, '$'
line10 db '***', 0Ah, 0Dh, '$'

.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX

    ; Print each line
    MOV DX, OFFSET line1
    MOV AH, 9
    INT 21H

    MOV DX, OFFSET line2
    MOV AH, 9
    INT 21H

    MOV DX, OFFSET line3
    MOV AH, 9
```

INT 21H

MOV DX, OFFSET line4

MOV AH, 9

INT 21H

MOV DX, OFFSET line5

MOV AH, 9

INT 21H

MOV DX, OFFSET line6

MOV AH, 9

INT 21H

MOV DX, OFFSET line7

MOV AH, 9

INT 21H

MOV DX, OFFSET line8

MOV AH, 9

INT 21H

MOV DX, OFFSET line9

MOV AH, 9

INT 21H

MOV DX, OFFSET line10

MOV AH, 9

INT 21H

MOV AH, 4Ch

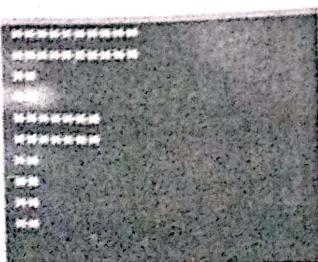
INT 21H

MAIN ENDP

END MAIN

OUTPUT

508 C:\Users\user\Documents\100205\_20191212



3/2/20