Assembly Language Program - Part I

1. Assembly, Linking and Executing Assembly Language Program

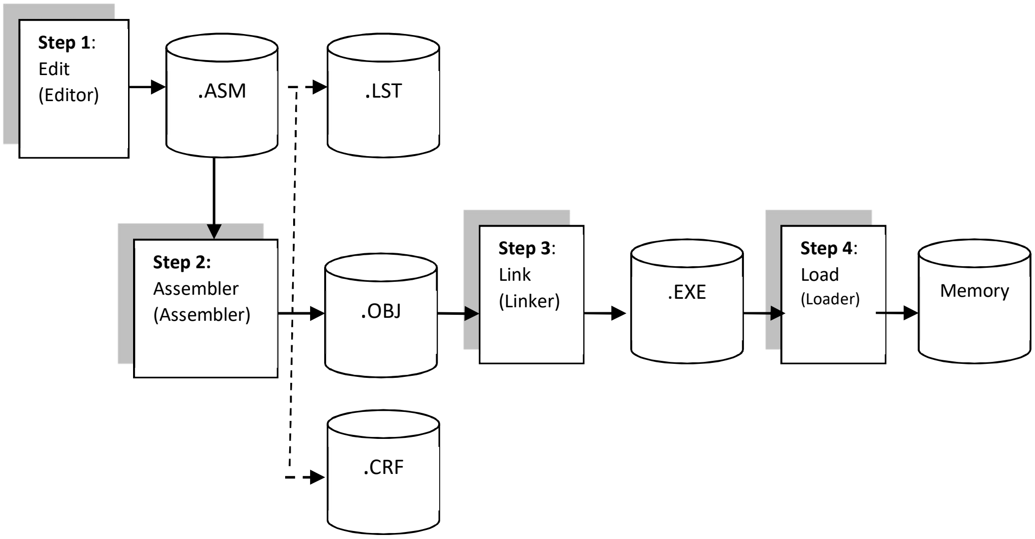

Figure: Assembly, Linking and executing Assembly Language Program

1) Step 1: Edit
   - Tools used: Editor (E.g.: Microsoft Visual C++ Express Edition 2005, notepad, DosBox)
   - Objective: Convert text into source file (`.ASM`)
   - Output: `.ASM` file (Source file. Contains source code)

2) Step 2: Assembling
   - Tools used: Assembler
   - Act as a compiler
   - Convert source file (`.ASM`) to object file (`.OBJ`) and optionally error listing file (`.LST`) and cross reference file (`.CRF`)
   - Assembler used = Two pass assembler
     - ✓ Objective: To resolve forward reference to addresses in program
     - ✓ Pass 1: Read `.ASM` & construct `SYMBOL TABLE` (with names & labels) to determine the number of codes generated for each instruction.
     - ✓ Pass 2: Complete the object code for each instruction into symbol table.

- ✓ Output: **`.OBJ`** file (Object file. Contain machine language), **`.LST`** file (Error listing file. Contains error diagnostic) & **`.CRF`** file (Cross reference file contains references for large program).

3) Step 3: Linking
   - Tools used: Linker.
   - Read object file (**`.OBJ`**)
   - Check references
   - Combine procedure from link library with object file (**`.OBJ`**)
   - Objective: Convert object file (**`.OBJ`**) to executable file (**`.EXE`**)
   - Functions:
     - ✓ Combine separated module
     - ✓ Generate **`.EXE`** file
   - Outputs: **`.EXE`** file (Executable file. Contains executable code), **`.MAP`** file and **`.LIB`** file

4) Step 4: Executing
   - Tools used: Loader.
   - Creates PSP (Program segment prefix).
   - Objective: Read executable file (**`.EXE`**) into memory for execution.

2. Basic Elements in Assembly Language Program

1) Constants

   a) Numeric constant
   - Can be written in any base
     - ✓ Debug program: HEX (default base)
     - ✓ Assembly Language editor: DEC (default base)
   - Specified by radix suffices
     - ✓ B = Binary (Or Y = Binary
     - ✓ D = Decimal (Or T = Ten)
     - ✓ H = Hexadecimal
     - ✓ R = real value for HEX or DEC constant
   - E.g.:
     ```
     VAR1 DB 2
     VAR2 DB 2AH
     VAR3 DB 10101010B
     ```

   b) Character constant
   - Enclosed using single quote (' ') or double quote (‶ ″)
   - E.g.:
     ```
     VAR4 DB "A"
     VAR5 DB 'B'
     ```

   c) String constant
   - Enclosed using single quote (' ') or double quote (‶ ″)
   - E.g.:
     ```
     VAR6 DB "HRLLO WORLD$"
     VAR7 DB 'HELLO WORLD',"$"
     ```

2) Comments
   - Used to improve the program clarity.

a) Single line comment

- ✓ Begin with a semicolon (;)
- ✓ E.g.: `MOV AX,BX                  ; comment`

b) Multi-lines comment

- ✓ Begin and end with delimiter
- ✓ E.g.:

```
COMMENT + xxxxxxxxxxxxxxxxxxx
        + xxxxxxxxxxxxxxxxxxxxx
```

3) Reserved words

- Has special meaning for special purpose
  - a) Instruction (e.g.: `MOV`, `ADD`, `MUL`)
  - b) Directives (e.g.: `END`, `SEGMENT`)
  - c) Operator (e.g.: `FAR`, `SIZE`)
  - d) Predefined symbols (e.g.: `@DATA`, `@MODEL`)

4) Identifiers

- To represent an item / act as a marker
- Rules:
  - ✓ 1st char MUST be a letter / underscore / special character
  - ✓ 1st char cannot be a dot
  - ✓ 1st char cannot be a digit
  - ✓ No casa-sensitive

a) Data label

- ✓ Refer to data item
- ✓ E.g.: `NUM  DB  0               ; NUM = data identifier`

b) Code label

- ✓ Refer to instruction / procedure / segment
- ✓ E.g.: `SUM: ADD AL,BL        ;SUM = instruction identifier`

5) Statements

- General format:

```
[Identifier]    Operation   [operand(s)]  [;comment]
```

a) Instructions

- ✓ Definition: Executable code
- ✓ E.g.: `L10: MOV AL,BL`

b) Directive

- ✓ Definition: Acted upon by the assembler
- ✓ E.g.: `NUM DB 1`

6) Directives

a) Memory model

- ✓ Objective: Provide space for object code & optimize execution
- ✓ Format: `.MODEL mem-model`

b) Stack segment

- ✓ Default size = 1024 bytes

- ✓ Format: `.STACK`
- ✓ Characteristic: Can be override

c) Data segment

- ✓ Format: `.DATA`
- ✓ Declaration:

```
MOV AX,@data
MOV DS,AX
```

d) Code segment

- ✓ Format: `.CODE`

e) PROC directive

- ✓ Initiation: `PROC FAR`
- ✓ Ending : `ENDP`

f) **END** directive

- ✓ Objective: End of entire program
- ✓ E.g.: `END MAIN`
- ✓ Procedure:

```
MOV AX,4C00H
INT 21H
```

- ✓ Same as:

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

3. Assembly Language Program Structure

Program structure	Description

.MODEL SMALL
.STACK 64
.DATA
    ;data definition
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ;codes
 	
    MOV AX,4C00H
    INT 21H
MAIN ENDP
  END MAIN
	✓ .MODEL = define program size
	✓ SMALL = uses data segment & 1 code segment
	✓ .STACK = define stack segment
	✓ 64 = defined size. The default size is 1024.Since it is reusable, therefore 64 or 100 bytes shall be sufficient.
	✓ .DATA = define data segment (for variables & constants)
	✓ .CODE = define code segment (for all instructions
	✓ MAIN PROC = MAIN start up procedure
	✓ MOV AX@DATA = @DATA retrieves value from current machine to AX register
	✓ MOV DS,AX = The value of AX is moved to / used to define the data segment address

	✓ `MOV AX,4CH` = When it is followed by `INT 21H`, it is used to terminate program execution ✓ `MAIN ENDP` = end the PROC ✓ `END MAIN` = end the MAIN

Commands for program execution

`ALT + ENTER`	To change screen size
`MOUNT C C:\8086`	To establish connection
`C:`	To change directory
`EDIT FILE1.ASM`	To edit source file (to ASM file) Or notepad / notepad++ / etc.
`MASM FILE1.ASM;`	To assemble source code (to OBJ file)
`LINK FILE1.OBJ;`	To link object file (to EXE file)
`FILE1.EXE`	To execute the exe file (program)
Note: For efficiency: File name max to 7 char/num	

4. Data Definition

1) Format
   `[name]      directive        initializer`

2) Storage size / directive
   a) `DB` (Define byte) = 1 byte (8 bits)
   b) `DW` (Define word) = 2 bytes (16 bits)
   c) `DD` (Define double word) = 4 bytes (32 bits)
   d) `DQ` (Define quad word) =8 bytes (64 bits)

3) Initializer: 0, ? or value

4) Sample date stored in memory

<table>
<tr><th>Data Definition
(FFH = 255D)</th><th>Data stored in memory
(Reversed byte sequence)</th><th>Data stored in register
(Normal byte sequence)</th></tr>
<tr><td><b><u>Single element / initializer</u></b>
A   DB 10
; A =0AH

B   DW 10
; B =000AH

C   DD 10
; C =00 00 00 0AH</td><td><table><tr><th>var</th><th>Data seg (DS)</th><th>Offset address (SI)</th></tr><tr><td></td><td>00H</td><td></td></tr><tr><td></td><td>00H</td><td></td></tr><tr><td></td><td>00H</td><td></td></tr></table></td><td>AX = 2 bytes

AX = 0AH<table><tr><td>00H</td><td>0AH</td></tr><tr><td>AH
(1 byte)</td><td>AL
(1 byte)</td></tr></table></td></tr>
</table>

```
D   DQ 10
;D = 00 00 00 00 00 00
00 0AH
```

	00H	
	00H	
	00H	
	00H	
D	0AH	0007H
	00H	
	00H	
	00H	
C	0AH	0003H
	00H	0002H
B	0AH	0001H
A	0AH	0000H

AX = 310AH

31H	0AH
AH (1 byte)	AL (1 byte)

5. Assembly Language Instructions

1) MOV instruction
   - Objective: Copy data from a location to another
   - Rule : Agreed size
   - Format : MOV reg/mem, reg/mem/imm
   - E.g.: MOV AX,BX
MOV AX, VAR1
MOV VAR2,BX

```
MOV AX,BX
MOV AX, VAR1
MOV VAR2,BX
```

2) XCHG instruction
   - Objective: Swap the contents of two memory location
   - Format : XCHG reg/mem, mem/reg
   - E.g.:

```
XCHG AX,BX
XCHG AX,VAR1
XCHG VAR2,BX
```

3) ADD / SUB instruction
   - Rule : - Operands of same size
     - No memory to mememory
   - Format: ADD/SUB destination, source
   - E.g.:

```
ADD AL,BL
ADD AX,VAR1
ADD VAR1,BL
```

4) INC / DEC instruction
   - Characteristic: + / operand by 1

- Format : `INC / DEC reg/mem`
- E.g.:
```
INC VAR1
INC SI
DEC AX
```

5) `MUL / DIV` instruction

- Rule : - Perform on unsigned data
  - No mem to mem
- Format: `MUL./DIV reg/mem`
- `MUL` instruction

```
  6    (Multiplicand)
X 3    (Multiplier)
 18    (Product)
```

✓ If multiplier = 1 byte, product in AX

✓ If multiplier = 2 bytes, product in DX, AX

✓ E.g.:
```
MOV AL, 2   ;AL=02H (multiplicand)
MOV BL, 3   ;BL=03H (multiplier)
MUL BL      ; AX=AL* BL
            ; AX=0006H
```

- `DIV` instruction

```
    41     (Quotient)
  2 )83    (Divisor) ) (Dividend)
    82
            (Remainder)
```

✓ If divisor = 1 byte, product in AX (R,Q)

✓ If divisor = 2 bytes, product in DX, AX (R, Q)
```
MOV AL,83   ;AL=83 (dividend)
MOV BL,2    ;BL=2 (divisor)
DIV BL      ;RQ→ AX =0141H =(R)(Q)
```

6) `SHL / SHR` instruction

- Characteristic: Shift 1 bit to LEFT / RIGHT
- Format : `SHL / SHR destination, 1 / CL`
- E.g.:
```
MOV AL, 5    ; AL = 5D=5H = 0000 0101B
SHL AL,1     ; AL = 0000 1010B = 0AH
```

6. Bitwise Logical Operations

1) Format

```
AND / OR/ XOR destination / source
NOT register / memory
```

2) Instructions

	1	1	0	0
AND	1	0	1	0
	1	0	0	0

	1	1	0	0
OR	1	0	1	0
	1	1	1	0

	1	1	0	0
XOR	1	0	1	0
	0	1	1	0

NOT	1	0
	0	1

7. ASCII Table

HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII	HEX	ASCII
00		20		40	@	60	`	80	Ç	A0	á	C0	└	E0	α
01	☺	21	!	41	A	61	a	81	ü	A1	í	C1	┴	E1	ß
02	☻	22	"	42	B	62	b	82	é	A2	ó	C2	┬	E2	Γ
03	♥	23	#	43	C	63	c	83	â	A3	ú	C3	├	E3	π
04	♦	24	$	44	D	64	d	84	ä	A4	ñ	C4	─	E4	Σ
05	♣	25	%	45	E	65	e	85	à	A5	Ñ	C5	┼	E5	σ
06	♠	26	&	46	F	66	f	86	å	A6	ª	C6	╞	E6	µ
07	•	27	'	47	G	67	g	87	ç	A7	º	C7	╟	E7	τ
08	◘	28	(	48	H	68	h	88	ê	A8	¿	C8	╚	E8	Φ
09	○	29	)	49	I	69	i	89	ë	A9	⌐	C9	╔	E9	Θ
0A	◙	2A	*	4A	J	6A	j	8A	è	AA	¬	CA	╩	EA	Ω
0B	♂	2B	+	4B	K	6B	k	8B	ï	AB	½	CB	╦	EB	Δ
0C	♀	2C	,	4C	L	6C	l	8C	î	AC	¼	CC	╠	EC	∞
0D	♪	2D	-	4D	M	6D	m	8D	ì	AD	¡	CD	═	ED	φ
0E	♫	2E	.	4E	N	6E	n	8E	Ä	AE	«	CE	╬	EE	ε
0F	☼	2F	/	4F	O	6F	o	8F	Å	AF	»	CF	╧	EF	∩
10	►	30	0	50	P	70	p	90	É	B0	░	D0	╨	F0	≡
11	◄	31	1	51	Q	71	q	91	æ	B1	▒	D1	╤	F1	±
12	↕	32	2	52	R	72	r	92	Æ	B2	▓	D2	╥	F2	≥
13	‼	33	3	53	S	73	s	93	ô	B3	│	D3	╙	F3	≤
14	¶	34	4	54	T	74	t	94	ö	B4	┤	D4	╘	F4	⊡
15	§	35	5	55	U	75	u	95	ò	B5	╡	D5	╒	F5	⊡
16	▬	36	6	56	V	76	v	96	û	B6	╢	D6	╓	F6	÷
17	|?|	37	7	57	W	77	w	97	ù	B7	╖	D7	╫	F7	≈
18	↑	38	8	58	X	78	x	98	ÿ	B8	╕	D8	╪	F8	°
19	↓	39	9	59	Y	79	y	99	Ö	B9	╣	D9	┘	F9	∙
1A	→	3A	:	5A	Z	7A	z	9A	Ü	BA	║	DA	┌	FA	·
1B	←	3B	;	5B	[	7B	{	9B	¢	BB	╗	DB	█	FB	√
1C	∟	3C	<	5C	\	7C	|	9C	£	BC	╝	DC	▄	FC	ⁿ
1D	↔	3D	=	5D	]	7D	}	9D	¥	BD	╜	DD	▌	FD	²
1E	▲	3E	>	5E	^	7E	~	9E	₧	BE	╛	DE	▐	FE	■
1F	▼	3F	?	5F	_	7F	⊡	9F	ƒ	BF	┐	DF	▀	FF	