Lecture 7

Assembling, Linking and Executing a Program

Text: Chapter 5

There are three main steps in running any program:

Translation

- Converting the program from source form (usually a character file) into binary object code. (Or in the case of Java, "byte code")
- For high-level languages an appropriate compiler is used (Pascal, C, C++, Ada,…)
- For assembly language, an assembler is used. For IBM PC's, TASM, MASM,…

Command:

```
C>TASM  <filename>
```

Linking/Loading

- Combining (linking) the programmer's code with other subprograms that are necessary to make the program complete.

- Placing the program into memory by putting each segment at an appropriate address.

Command:

```
C>TLINK /Tde  <filename>
```

Execution

- Running the program either stand-alone, or under the control of a debugger or other systems program.

See figure 5-1 from the text.

How to Invoke the Assembler

As previously seen, the Turbo Assembler is started by the following command at the DOS prompt:

```
C:>TASM <options> <file name>
```

However, since you may be working on a computer attached to the Queens College network, the assembler may be on a different disk, perhaps “N:”

```
N:DOSAPP\TASM\BIN>
```

The assembler will attempt to write your .OBJ file to the same disk from which it was invoked (N:), and you cannot write to the N: drive, there will be a problem. In addition, your program is most likely on a floppy drive (A:)

The following may work presuming you have a file on your A: disk called “project1.asm”.

Make the default drive your A: disk so the assembler can write to it:

```
N:DOSAPP\TASM\BIN>A:
```

Invoke the assembler batch file on the N: drive:

```
A:>N:TASM /l project1
```

Note that it is slash "ell", not slash one!

(the only option you’ll likely need is “/l” for the listing)

Command Line Options

Turbo Assembler

```
Syntax:  TASM [options] source [,object] [,listing] [,xref]
/a,/s         Alphabetic or Source-code segment ordering
/c            Generate cross-reference in listing
/dSYM[=VAL]   Define symbol SYM = 0, or = value VAL
/e,/r         Emulated or Real floating-point instructions
/h,/?         Display this help screen
/iPATH        Search PATH for include files
/jCMD         Jam in an assembler directive CMD (eg. /jIDEAL)
/kh#          Hash table capacity # symbols
/l,/la        Generate listing: l=normal listing, la=expanded listing
/ml,/mx,/mu   Case sensitivity on symbols: ml=all, mx=globals, mu=none
/mv#          Set maximum valid length for symbols
/m#           Allow # multiple passes to resolve forward references
/n            Suppress symbol tables in listing
/os,/o,/op,/oiObject code: standard, standard w/overlays, Phar Lap, or IBM
/p            Check for code segment overrides in protected mode
/q            Suppress OBJ records not needed for linking
/t            Suppress messages if successful assembly
/uxxxx        Set version emulation, version xxxx
/w0,/w1,/w2   Set warning level: w0=none, w1=w2=warnings on
/w-xxx,/w+xxx Disable (-) or enable (+) warning xxx
/x            Include false conditionals in listing
/z            Display source line with error message
/zi,/zd,/zn   Debug info: zi=full, zd=line numbers only, zn=none
```

Turbo Link

```
Syntax: TLINK objfiles, exefile, mapfile, libfiles, deffile
@xxxx indicates use response file xxxx
/m  Map file with publics         /x  No map file at all
/i  Initialize all segments       /l  Include source line numbers
/L  Specify library search paths  /s  Detailed map of segments
/n  No default libraries          /d  Warn if duplicate symbols in libraries
/c  Case significant in symbols   /3  Enable 32-bit processing
/o  Overlay switch                /v  Full symbolic debug information
/P[=NNNNN]  Pack code segments    /A=NNNN  Set NewExe segment alignment
/ye Expanded memory swapping      /yx Extended memory swapping
/e  Ignore Extended Dictionary
/t  Create COM file (same as /Tdc)
/C  Case sensitive exports and imports
/Txx  Specify output file type
             /Tdx  DOS image (default)
             /Twx  Windows image
                (third letter can be c=COM, e=EXE, d=DLL)
```

Example Assembler Listing

Location Counter (Offset)- counts how far into the segment each item is.

Generated code or data for each line.

```
Turbo Assembler    Version 3.2  08/06/96 22:16:31   Page 1
a:p05asm1.ASM
P05ASM1     (EXE)  Move and add operations


1                       ; ---------------------------------------------
2     0000              STACKSG SEGMENT PARA STACK  'Stack'
3     0000  20*(0000)           DW     32 DUP(0)
4     0040              STACKSG ENDS
5                       ; ---------------------------------------------
6     0000              DATASG  SEGMENT PARA 'Data'
7     0000  00FA        FLDA    DW     250
8     0002  007D        FLDB    DW     125
9     0004  ????        FLDC    DW     ?
10    0006              DATASG ENDS
11                      ; ---------------------------------------------
12    0000              CODESG  SEGMENT PARA 'Code'
13    0000              BEGIN   PROC    FAR
14                              ASSUME SS:STACKSG,DS:DATASG,CS:CODESG
15    0000  B8 0000s            MOV    AX,DATASG  ;Set address of DATASG
16    0003  8E D8               MOV    DS,AX      ;in DS register
17                      ;
18    0005  A1 0000r            MOV    AX,FLDA      ;Move 0250    to AX
19    0008  03 06 0002r         ADD    AX,FLDB      ;Add  0125    to AX
20    000C  A3 0004r            MOV    FLDC,AX      ;Store sum    in FLDC
21    000F  B8 4C00             MOV    AX,4C00H     ;Exit to DOS
22    0012  CD 21               INT    21H
23    0014              BEGIN   ENDP                ;End of procedure
24    0014              CODESG  ENDS   ;End of segment
25                              END    BEGIN        ;End of program
```

The generated code is A30004
Note the location counter value for FLDC

```
Turbo Assembler      Version 3.2         08/06/96 22:16:31       Page 2
Symbol Table
P05ASM1    (EXE)   Move and add operations


Symbol Name          Type       Value

??DATE               Text       "08/06/96"
??FILENAME           Text       "p04asm1 "
??TIME               Text       "22:16:30"
??VERSION            Number     0314
@CPU                 Text       0101H
@CURSEG              Text       CODESG
@FILENAME            Text       P04ASM1
@WORDSIZE            Text       2
BEGIN                Far        CODESG:0000
FLDA                 Word       DATASG:0000
FLDB                 Word       DATASG:0002
FLDC                 Word       DATASG:0004

Groups & Segments         Bit        Size    Align  Combine   Class

CODESG                    16         0014    Para   none      CODE
DATASG                    16         0006    Para   none      DATA
STACKSG                   16         0040    Para   Stack     STACK
```

The symbol table is list of all identifiers used in the program along with details such as offset, type, value

Initialization of the Data Segment

```
6     0000                DATASG   SEGMENT PARA 'Data'
```

```
13    0000                BEGIN    PROC    FAR
14                                 ASSUME SS:STACKSG,DS:DATASG,CS:CODESG
15    0000  B8 0000s               MOV    AX,DATASG  ;Set address of DATASG
16    0003  8E D8                  MOV    DS,AX      ;in DS register
```

Recall that the DOS Loader initializes the DS and ES registers with the address of the PSP.

It is the programmer's responsibility to make the DS register point to the Data Segment (and similarly the ES if there is one).

The instruction on line 15 looks like it wants to move the contents of the variable "DATASG" into the AX register, However, there is no variable called "DATASG" – it is a segment definition.

The assembler recognizes this fact, and generates an instruction with a zero offset, and marks it as a reference to a segment address (note the "s" next to the offset).

The LOADER will change this 0000 to the actual address of the segment, and when the instruction is executed the AX register will be loaded with that address.

The instruction on line 16 puts this address in the DS register (there is no instruction to move an address directly into the DS register, thus two instructions are needed to do this operation).

The Two Pass Assembler

Generating complete code by going through the program once from beginning to end is not possible[1].

Generating code requires knowledge of offsets for the symbols that are used:

```
MOV    AX,SALARY
```

If the assembler knows that the offset for SALARY is 0040, then the code generated should be

```
A1 0040r
```

But what if SALARY is declared LATER in the program than the instruction that uses it?

```
15.          MOV    AX,SALARY
…
34. SALARY   DW     ?
```

At line 15 the assembler has not yet seen the symbol SALARY, and hasn't counted what the offset should be.

This is called a **FORWARD REFERENCE**.

Forward references are resolved by using TWO PASSES.

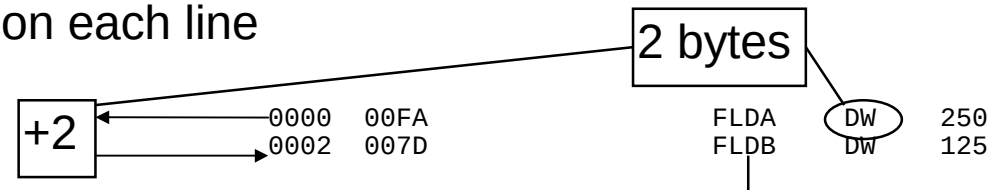
[1] OK, so it is possible with some tricks, but let's say it's not to keep things simple.

PASS 1:

- The assembler goes through the entire program looking for labels.
- Labels are put on the SYMBOL TABLE, along with their offset value.
- The assembler counts bytes from the beginning using a variable called the LOCATION COUNTER

```
2     0000                STACKSG SEGMENT PARA STACK  'Stack'
3     0000  20*(0000)             DW     32 DUP(0)
4     0040                STACKSG ENDS
5                         ; --------------------------------------------
6     0000                DATASG  SEGMENT PARA 'Data'
7     0000  00FA          FLDA    DW     250
8     0002  007D          FLDB    DW     125
9     0004  ????          FLDC    DW     ?
10    0006                DATASG ENDS
11                        ; --------------------------------------------
12    0000                CODESG  SEGMENT PARA 'Code'
13    0000                BEGIN   PROC    FAR
14                                ASSUME SS:STACKSG,DS:DATASG,CS:CODESG
15    0000  B8 0000s              MOV    AX,DATASG  ;Set address of DATASG
16    0003  8E D8                 MOV    DS,AX      ;in DS register
17                        ;
18    0005  A1 0000r              MOV    AX,FLDA      ;Move 0250 to AX
19    0008  03 06 0002r           ADD    AX,FLDB      ;Add  0125 to AX
20    000C  A3 0004r              MOV    FLDC,AX      ;Store sum in FLDC
21
```

The Location Counter increases by the amount of storage used on each line



Symbol Table

```
BEGIN              Far        CODESG:0000
FLDA               Word       DATASG:0000
FLDB               Word       DATASG:0002
FLDC               Word       DATASG:0004
```

LINKING an Object (.OBJ) Program

The Linker will produce one .EXE (or .COM) file from one or more object code files (.OBJ).

It will resolve any addresses that were unknown to the assembler (such as the address of the DATA segment).

The command is TLINK:

```
C>TLINK P04ASM1
```

or

```
A:>N:TLINK P04ASM1
```

The file "P04ASM1.OBJ" is presumed to be on the default drive.
and
The file "P04ASM1.EXE" will be written to the default drive.

The linker also produces a MAP file indicating the start address and length of each segment. This may be useful for debugging.

```
 Start  Stop   Length Name     Class

 00000H 0003FH 00040H STACKSG  STACK
 00040H 00045H 00006H DATASG   DATA
 00050H 00063H 00014H CODESG   CODE

 Program entry point at 0005:0000
```

P04ASM1.MAP

Execution of the Program

- Since it is an .EXE file, it can be run directly from DOS.

```
C>P04ASM1↵
```

- It can also be run under control of Turbo Debug by opening the EXE file

Compare the generated code from the assembly listing with the code that is displayed by Turbo Debug in the Code Segment:

```
5     0000  B8 0000s          MOV AX,DATASG


CS:0000  B8 A7 0F 8E D8 A1 00 00-03 06 02
CS:0010  00 4C CD 21 C3 8E 06 84-77 E8 F2
CS:0020  00 00 3C 02 72 18 3C 04-73 14 8E
(lines omitted)
```

The address of the Data Segment is 0FA7h. The linker knew this and changed the 0000h to 0FA7 (reversed) in the instruction.

Example of Assembly Listing with Errors

```
            page 60,132
Turbo Assembler      Version 3.2       08/09/96 16:00:19
Page 1
a:p05asm3.ASM
P05ASM3   (EXE)  Illustrate assembly errors


 1                      ; ---------------------------------
 2   0000               STACKSG SEGMENT PARA STACK 'Stack'
 3   0000  20*(0000)             DW             32 DUP(0)
 4   0040               TACKSG  NDS
 5                      ; ---------------------------------
 6   0000               DATASG  SEGMENT PARA 'Data'
 7   0000  00FA         FLDA    DW             250
 8   0002  007D         FLDB    DW             125
 9   0004  ????         FLDC    DW
*Warning* a:p05asm3.ASM(11) Missing operand - trailing ? assumed
10   0006               DATASG  ENDS
11                      ; ---------------------------------
12   0000               CODESG  EGMENT PARA 'Code'
13   0000               BEGIN   PROC    FAR
14                      ASSUME  S:CODESG,DS:DATASG
15   0000  A1 0000              MOV        AX,DATSEG
**Error** a:p05asm3.ASM(17) Undefined symbol: DATSEG
16   0003  8B D0                MOV        DX,AX
17
18   0005  8C A6 0000           MOV        AS,FLDA
**Error** a:p05asm3.ASM(20) Undefined symbol: AS
19   0009  03 06 0002r          ADD        AX,FLDB
20   000D  A3 0000              MOV        FLDD,AX
**Error** a:p05asm3.ASM(22) Undefined symbol: FLDD
21   0010  B8 4C00              MOV        AX,4C00H ;Exit to DOS
22   0013  CD 21                INT        21H
23   0015               BEGIN   ENDP
24   0015               CODESG  ENDS
25                              END        BEGIN
```

```
Turbo Assembler      Version 3.2       08/09/96 16:00:19
Page 3
Error Summary
P05ASM3   (EXE)  Illustrate assembly errors


*Warning* a:p05asm3.ASM(11) Missing operand - trailing ? assumed
**Error** a:p05asm3.ASM(17) Undefined symbol: DATSEG
**Error** a:p05asm3.ASM(20) Undefined symbol: AS
**Error** a:p05asm3.ASM(22) Undefined symbol: FLDD
```

Exercises - Lecture 7

Fill in the location counter values for each line in the assembly program below. Recall that the location counter starts at zero for each segment, and increases by the number of bytes used on each line.

```
 1  0000                STACKSG  SEGMENT  PARA  STACK
 2  ____                         DW       32 DUP(0)
 3  ____                STACKSG  ENDS
 4  ____                DATASG   SEGMENT  PARA 'Data'
 5  ____  0004          FOUR     DW      4
 6  ____                DATASG   ENDS
 7  ____                CODESG   SEGMENT  PARA 'Code'
 8  ____                BEGIN    PROC     FAR
 9                      ASSUME   SS:STACKSG,DS:DATASG,CS:CODESG
10  ____  B8 0000s              MOV      AX,DATASG
11  ____  8E D8                 MOV      DS,AX
12                      ;
13  ____  B8 0123               MOV   AX,0123H
14  ____  8B D8                 MOV   BX,AX
15  ____  03 1E 0000r           ADD   BX,FOUR
16                      ;
17  ____  B8 4C00               MOV      AX,4C00H
18  ____  CD 21                 INT      21H
19  ____                BEGIN    ENDP
20  ____                CODESG   ENDS
21                              END      BEGIN
```