AACS3064
Computer Systems Architecture

Chapter 6: Assembling, Linking and Executing Program

Chapter Overview

1) Preparing a program for assembling and execution.
2) Assembling a source program.
3) Linking an object program.
4) Executing a program.
5) Error diagnostics.
6) The assembler program counter.

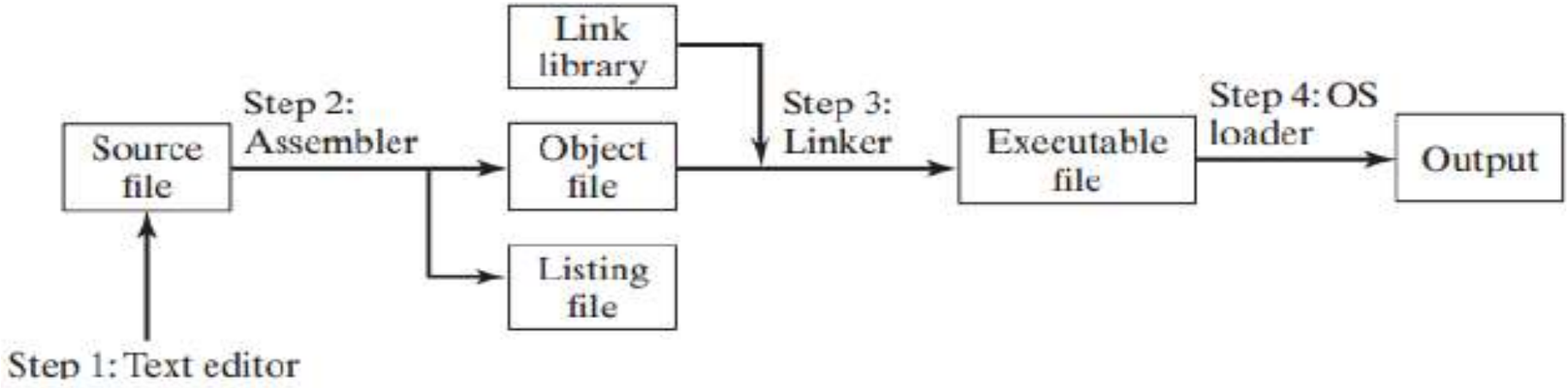
1. Preparing a program for assembling & execution

1. Preparing a program for assembling & execution

- **Preparing a program**
  - A source program written in assembly language cannot be executed directly on its target computer.
  - It must be translated, or **assembled** into **executable** code.
  - An assembler is similar to a compiler

1. Preparing a program for assembling & execution (Continued)

- Assemble-Link-Execute Cycle



1. Preparing a program for assembling & execution (Continued)

- Assemble-Link-Execute Cycle

STEP1

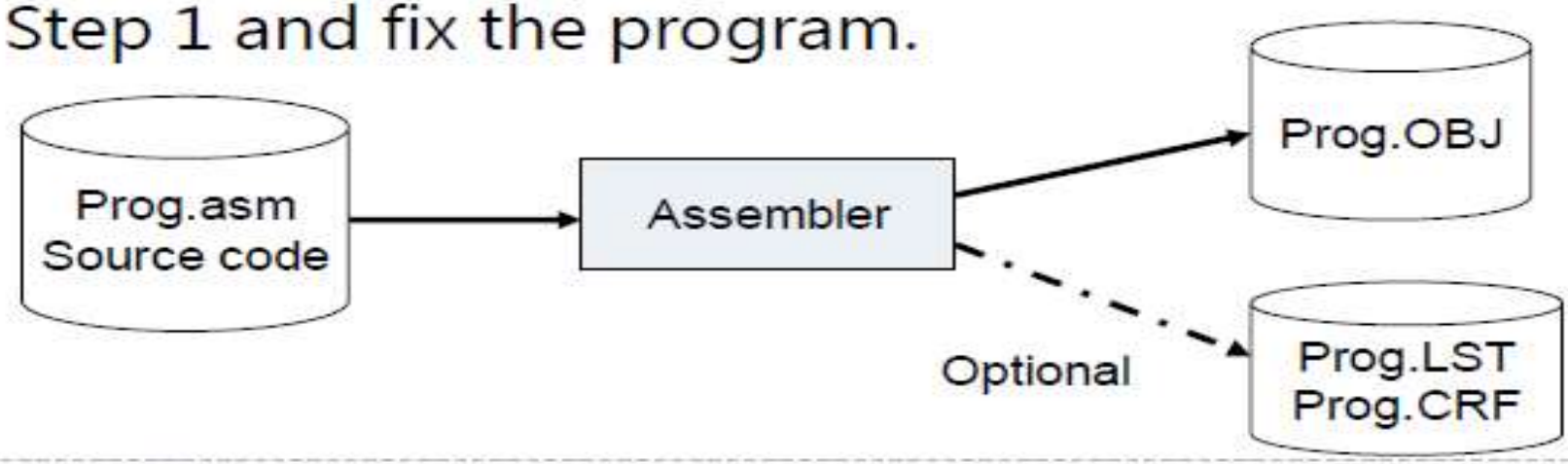
- A programmer uses a text editor to create an ASCII text file named the **source file**.
- File extension is .ASM

1. Preparing a program for assembling & execution (Continued)

- Assemble-Link-Execute Cycle

STEP2 (Assembly Step)

- The assembler reads the **source file** and produces an **object file**, a machine-language translation of the program.
- Optionally, it produces a **listing file**. If any errors occur, the programmer must return to Step 1 and fix the program.



1. Preparing a program for assembling & execution (Continued)

- Assemble-Link-Execute Cycle

STEP 3 (Link Step)

- The linker reads the object file and checks to see if the program contains any calls to procedures in a link library.
- The linker copies any required procedures from the link library, combines them with the object file, and produces the executable file.

1. Preparing a program for assembling & execution (Continued)

- Assemble-Link-Execute Cycle

STEP4 (Loading Step)

- The operating system loader utility reads the executable file into memory and branches the CPU to the program' s starting address, and the program begins to execute.
- The loader creates a program segment prefix (PSP) immediately before the program loaded in memory.

2. Assembling a source program

2. Assembling a source program

- A05ASM1.asm
- Move and Add operation

```
1          page    60, 132
2
3  TITLE   A05ASM1 (EXE)       Move and add operations
4
5  STACK   SEGMENT PARA STACK 'Stack'
6          DW      32 DUP(0)
7
8
9  STACK   ENDS
10
11 DATASEG SEGMENT PARA 'Data'
12 FLDD    DW      215
13 FLDE    DW      125
14 FLDF    DW      ?
15 DATASEG ENDS
16
17 CODESEG SEGMENT PARA 'Code'
18 MAIN    PROC    FAR
19         ASSUME  SS:STACK, DS:DATASEG, CS:CODESEG
20         MOV     AX, DATASEG     ; set address of data
21         MOV     DS, AX          ; segment in DS
22
23         MOV     AX, FLDD        ; Move 0215 to AX
24         ADD     AX, FLDE        ; Add 0125 to AX
25         MOV     FLDF, AX        ; Store sum in FLDF
26         MOV     AX, 4C00H       ; End processing
27         INT     21H
28 MAIN    ENDP                    ; End of procedure
29 CODESEG ENDS                    ; End of segment
30         END     MAIN            ; End of program
```

2. Assembling a source program(Continued)

Assembling a source program

- The assembler converts your source statements into machine code and displays any **error messages** on the screen.
- Typical errors include :
  - a name that violates naming conventions,
  - an operation that is spelt incorrectly (such as MOVE instead of MOV),
  - an operand containing a name that is not defined.
- Since there are many possible errors (100 or more) and many different assembler versions, you may refer to your assembler manual for a list.

2. Assembling a source program(Continued)

Assembling a source program

- Optional output files from the assembly step are **listing (.LST)** and **cross reference (.CRF).**
- You will probably request a .LST file , especially when it contains error diagnostics or you want to examine the generated machine code.
- A .CRF file is useful for a large program where you want to see which instructions reference which data item.
  - In addition to that, requesting a .CRF file causes the assembler to generate statement numbers for items in the .LST file to which the .CRF file refers.

2. Assembling a source program (Continued)

The Cross-reference listing

- The assembler generates an optional file that you can use to produce a cross-reference listing of a program' s identifiers or symbols.
- The file extension is
  - .SBR for MASM 6.1
  - .CRF for MASM5.1
  - .XRF for TASM
- However, you still have to convert the file to a properly sorted cross-reference file. (Refer to Appendix E, IBM PC Assembly Language and Programming)

2. Assembling a source program(Continued)

Cross-Reference Table

Symbol	Cross-Reference (# definition, + modification)			
MAIN	18#	28	30	
CODE	17			
CODESEG	17#	19	29	
DATA	11			
DATASEG	11#	15	19	20
FLDD	12#	23		
FLDE	13#	24		
FLDF	14#	25+		
STACK	4			
STACK	4#	9	19	

2. Assembling a source program(Continued)

Cross-Reference Table

- The symbols in the first column are in alphabetic order.
- The numbers in the second column, shown as *n#*, indicate the line each symbol is defined.
- Numbers to the right of this column are line numbers showing where the symbol is referenced by other statements.

2. Assembling a source program(Continued)

Cross-Reference Table

- For example :
  - **CODESEG** is defined in line 17 and is referenced in lines 19 and 29.
  - **FLDF** is defined in line 14 and referenced in line 25+,
    - where the "+" means its value is modified during program execution (by MOV FLDF,AX).

2. Assembling a source program (Continued)

Listing File

```
Segments and Groups:
                Name            Length  Align   Combine         Class
DGROUP          .....           GROUP
_DATA           .....           0006    WORD    PUBLIC    'DATA'
STACK           .....           0040    PARA    STACK     'STACK'
_TEXT           .....           0014    WORD    PUBLIC    'CODE'

Symbols:
                Name            Type     Value  Attr
MAIN            .....           F  PRO   0000   _TEXT   Length = 0014

FLDD            .....           L  WORD  0000   _DATA
FLDE            .....           L  WORD  0002   _DATA
FLDF            .....           L  WORD  0004   _DATA

@CODE           .....           TEXT     _TEXT
@FILENAME       .....           TEXT     a05asm2

        0     Warning Errors
        0     Severe  Errors
```

2. Assembling a source program(Continued)

Listing File

- The first part of the symbol table under "Segments and Groups" shows the three segments renamed by the assembler and listed alphabetically:
  - _DATA, with a length of 6 bytes
  - STACK, with a length of 40H (64 bytes)
  - _TEXT, for the code segment, with a length of 14H (20 bytes)

2. Assembling a source program(Continued)

Listing File

- Listed under the heading "Symbols" are names defined in the program or default names.
  - Names of data fields and program labels and their relative location (offsets) within the segment.
  - Default names, such as

@CODE Equated to the name of the code segment, _TEXT
@FILENAME Name of the program

- You may use @code and @data in ASSUME and executable statements,
  - such as MOV AX,@data

2. Assembling a source program (Continued)

Symbol

- With symbol name

```
    int x = 0;
013F13CE  mov         dword ptr [x], 0
    printf("%d", x);
013F13D5  mov         esi, esp
013F13D7  mov         eax, dword ptr [x]
```

- Without symbol name

```
    int x = 0;
013F13CE  mov         dword ptr [ebp-8], 0
    printf("%d", x);
013F13D5  mov         esi, esp
013F13D7  mov         eax, dword ptr [ebp-8]
```

2. Assembling a source program (Continued)

A05ASM2.asm

```
1                page    60, 132
2
3   TITLE        A05ASM2 (EXE)       Move and add operations
4                .MODEL  SMALL
5                .STACK  64
6                .DATA
7
8   FLDD         DW      215
9   FLDE         DW      125
10  FLDF         DW      ?
11
12               .CODE
13  MAIN         PROC    FAR
14               MOV     AX, @data           ; set address of data
15               MOV     DS, AX              ; segment in DS
16
17               MOV     AX, FLDD            ; Move 0215 to AX
18               ADD     AX, FLDE            ; Add 0125 to AX
19               MOV     FLDF, AX            ; Store sum in FLDF
20
21               MOV     AX, 4C00H           ; End processing
22               INT     21H
23  MAIN         ENDP                        ; End of procedure
24
25               END     MAIN                ; End of program
```

2. Assembling a source program(Continued)

Using simplified segment definitions

- The simplified segment directives provide a number of predefined equates,
  - which begin with an @ symbol and which you are free to reference in a program.
- For the simplified segment directives, initialize DS like this:

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

- Please compare A05ASM1.asm and A05ASM2.asm

2. Assembling a source program(Continued)

TWO-PASS ASSEMBLER

- Assemblers typically make two or more passes through a source program

- **to resolve forward reference** to addresses not yet encountered in the program.

2. Assembling a source program (Continued)

TWO-PASS ASSEMBLER – Pass 1

- The assembler reads the entire source program and construct a symbol table of names and labels used in the program.

Symbols:

Name	Type	Value	Attr
@code	Text	_TEXT	
@data	Text	DGROUP	
FLDD	Word	0000	_DATA
FLDE	Word	0002	_DATA
FLDF	Word	0004	_DATA

- Determines the amount of codes to be generated for each instruction.

2. Assembling a source program(Continued)

TWO-PASS ASSEMBLER – Pass 2

- The assembler uses the symbol table that it constructed in Pass 1.
- Now, it knows the length and relative position of each data field and instruction, it can **complete the object code for each instruction.**
- It then produces, on request, the various object (**.OBJ**), list (**.LST**) and cross reference (**.CRF**) files.

2. Assembling a source program (Continued)

TWO-PASS ASSEMBLER

- A potential problem in Pass 1 :
  - **forward reference**

    Certain types of instructions in the code segment may reference the label of an instruction, but the assembler has not yet encountered its definition.

```
          JMP Label1
            :
            :
Label1 :  MOV AL, 01H
```

2. Assembling a source program(Continued)

TWO-PASS ASSEMBLER

- MASM constructs object code based on what it supposes is the length of each generated machine language instruction.
  - If there are any differences between Pass 1 and Pass 2 concerning instruction lengths, MASM issues an error message "Phase error between passes". (Rare)

3. Linking an object program

3. Linking an object program

Linking an object program

- **Link the object module** (.obj) that was produced by the assembler.
- The linker performs the following functions :
  - **Combines** more than one separately assembled module into one executable program.
  - **Generate an .EXE module** and initialize it with special instructions to facilitate its subsequent loading for execution.
- The output files can be :
  - **executable** (.EXE), **map** (.MAP), and **library** (.LIB).

3. Linking an object program (Continued)

Link map for the program

- The assembler physically **rearranged the segments** into alphabetical order,

- Shown by the link map below:

```
START         STOP          LENGTH        NAME          CLASS
00000H        00013H        0014H         _TEXT         CODE

00014H        00019H        0006H         _DATA         DATA

00020H        0005FH        0040H         STACK         STACK

Program entry point at 0000:0000
```

4. Executing a program

4. Executing a program

Tracing a program - DEBUG

- Key in the following, including the .EXE extension

```
DEBUG   A05ASM1.EXE
```

- DEBUG loads the .EXE program module and displays its hyphen prompt.

4. Executing a program (Continued)

Tracing a program – View SS

- To view the stack segment, key in `D SS:0`
  - The stack contains all zeros because it was initialized that way.

- To view the code segment, key in `D CS:0`
  - Compare the displayed machine code

    `B8 D2 17 8E D8 A1 00 00.....`
  - with machine code in the assembled listing

    `B8 -- -- 8E D8 A1 00 00.....`

The assembled listing does not accurately show the machine code, since the assembler did not know the **address** for the operand of the first instruction.

4. Executing a program (Continued)

Tracing a program – view Register

- To view the content of the registers,
  - press **R** followed by **<Enter>**.

- SP = 0040H
  - the size of the stack (32 words = 64 bytes = 40H).
- IP = 0000H.
- SS & CS are initialized for execution.
  - Values depend on where in memory the program is loaded.

4. Executing a program (Continued)

Tracing a program – Trace

- The first instruction **`MOV AX, xxxx`** is ready to execute.
  - To execute the first MOV
    - press **`T`** (for Trace) followed by **`<Enter>`**
    - note the effect on IP.
  - To execute the second MOV
    - again press **`T`** followed by **`<Enter>`**
    - check DS, which is now initialized with the segment address.

4. Executing a program (Continued)

Tracing a program - Trace

- The 3rd MOV
  - Loads the contents of FLDD into AX.

- Press **T** ,
  - note that AX = 00D7
- Press **T** , to execute the ADD instruction
  - note AX = 0154.
- Press **T** ,
  - cause MOV to store AX in offset 0004 of the DS.

4. Executing a program (Continued)

Tracing a program – Reload & Quit

- Check the contents of data segment : **`D DS:0`**
  - 3 data items display as : `D7 00 7D 00 54 01`

- Use **`L`** : to reload and rerun the program
- Use **`Q`** : to quit the DEBUG session.

5. Error diagnostics

5. Error diagnostics

Error Diagnostics

- A05ASM3 has a number of intentional errors inserted for illustrative purposes.
- The diagnostics will vary by assembler version.
  - Refer to A05ASM3.LST

5. Error diagnostics (Continued)

Error Diagnostics

LINE	EXPLANATION
9	The definition of FLDF requires an operand
14	DX should be coded as DS
16	AS should be coded as AX
18	FLDQ should be coded as FLDF
19	Field sizes must agree
22	Correcting the other errors will cause this diagnostics disappear
23	MIAN should be coded as MAIN

6. The assembler program counter

6. The assembler program counter

The assembler location counter

- The assembler maintains a location counter that it uses to account for each defined items in the data segment.

```
0000    .....    FLDD  DW
0002    .....    FLDE  DW
0004    .....    FLDF  DW
```

- Initially, the location counter is set at 0, where the assembler establishes the first data item, FLDD.

6. The assembler program counter (Continued)

The assembler location counter

- Since FLDD is defined as word, the assembler advances the location counter by 2, to 0002, where it establishes FLDE.
- Since FLDE is also defined as a word, the assembler again advances its location counter by 2, to 0004.
- For the next data item, FLDF, also a word.
- The location counter is again advanced by 2, to 0006, but there are no further data items.

6. The assembler program counter (Continued)

The assembler location counter

- The assembler provides a number of ways to change the current value in the location counter.

- For example :
  - Use EQU to redefine data items with different names.
  - Use the ORG directive to begin a program at a particular offset.
  - The EVEN or ALIGN directive to facilitate aligning an address on an even-numbered boundary.

Chapter Review

1) Preparing a program for assembling and execution.
2) Assembling a source program.
3) Linking an object program.
4) Executing a program.
5) Error diagnostics.
6) The assembler program counter.