University Institute of Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Bachelor of Engineering

Subject Name: System Programming

Subject Code: CST-315

Compilers

Chapter-1.2
Assembler

- Multi-Pass Assemblers
- Advanced Assembly Process

Forward Reference

- All symbol-defining directives do ***not*** allow forward reference for 2-pass assembler
  - e.g., EQU, ORG…
  - All symbols used on the *right-hand side* of the statement must have been defined previously

E.g. (Cannot be assembled in 2-pass assm.)

```
ALPHA     EQU      BETA
BETA      EQU      DELTA
DELTA     RESW     1
```

2.4 Assembler Design Options

- One-pass assemblers

- Multi-pass assemblers

2.4.1 One-Pass Assemblers

- Goal: avoid a second pass over the source program
- Main problem
  - Forward references to *data items* or *labels on instructions*
- Solution
  - Data items: require all such areas be defined before they are referenced
  - Label on instructions: cannot be eliminated
    - E.g. the logic of the program often requires a forward jump
    - It is too inconvenient if forward jumps are not permitted

Two Types of One-Pass Assemblers:

- Load-and-go assembler
  - Produces object code directly in memory for immediate execution
- The other assembler
  - Produces usual kind of object code for later execution

Load-and-Go Assembler

- No object program is written out, no loader is needed
- Useful for program development and testing
  - Avoids the overhead of writing the object program out and reading it back in
- Both one-pass and two-pass assemblers can be designed as load-and-go
  - However, one-pass also avoids the overhead of an additional pass over the source program
- For a load-and-go assembler, the actual address must be known at assembly time.

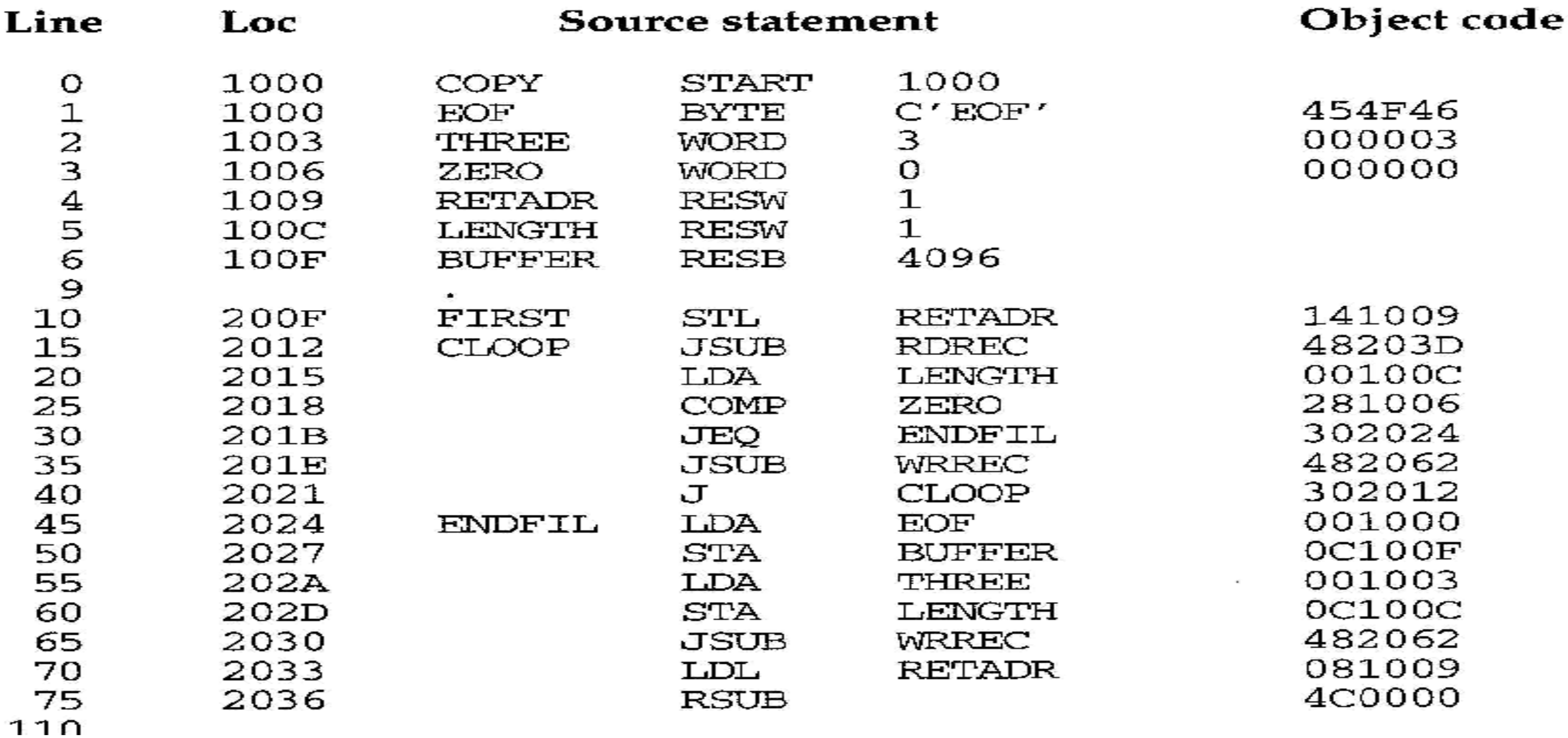
Forward Reference Handling in One-pass Assembler

- **When the assembler encounter an instruction operand that has not yet been defined:**
  1. **The assembler omits the translation of operand address**
  2. **Insert the symbol into SYMTAB, if not yet exist, and mark this symbol *undefined***
  3. **The address that refers to the undefined symbol is added to *a list of forward references* associated with the symbol table entry**
  4. When the definition for a symbol is encountered
     1. The forward reference list for that symbol is scanned
     2. The proper address for the symbol is inserted into any instructions previous generated.

Handling Forward Reference in One-pass Assembler (Cont.)

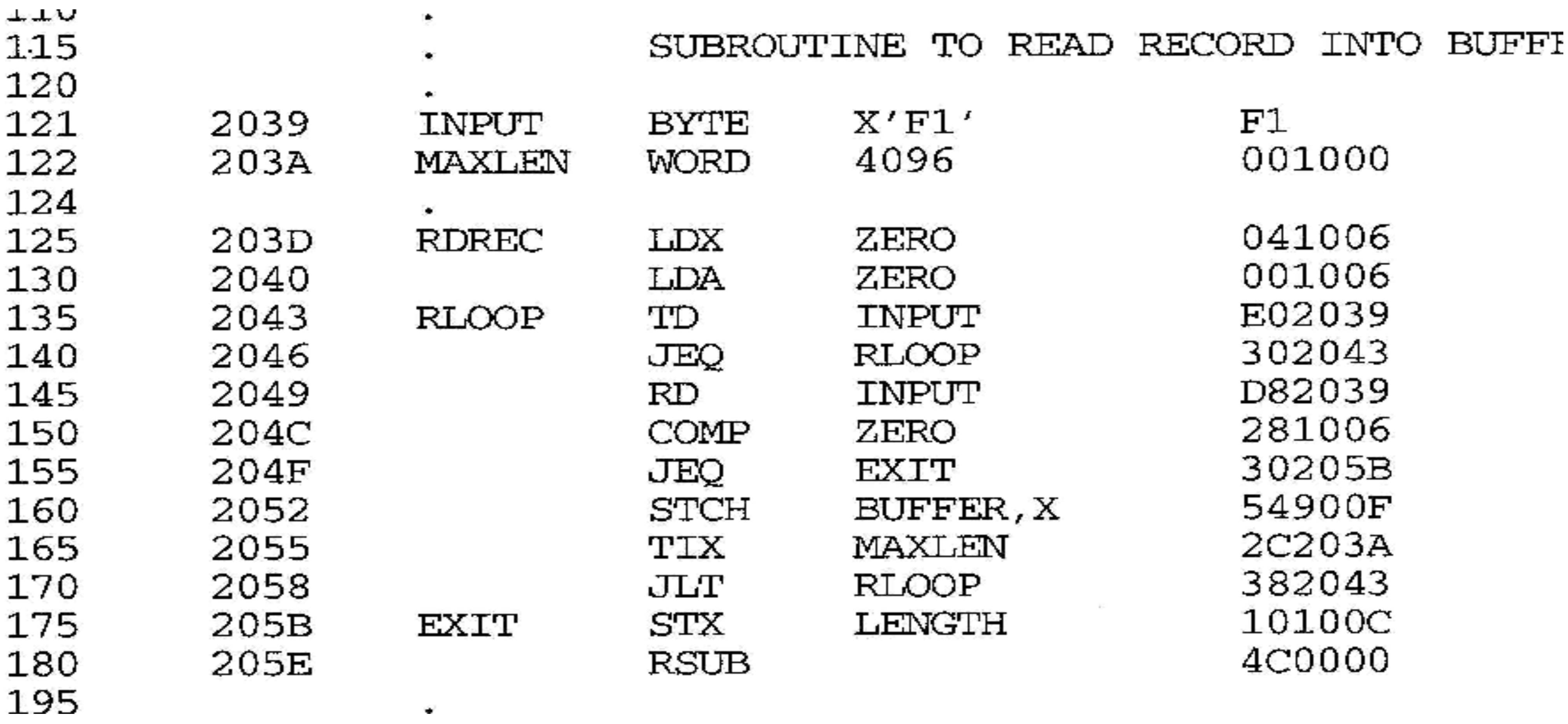
- At the end of the program
  - Any SYMTAB entries that are still marked with * indicate *undefined symbols*
    - Be flagged by the assembler as errors
  - Search SYMTAB for the symbol named in the END statement and jump to this location to begin execution of the assembled program.

Sample Program for a One-Pass Assembler (Fig. 2.18)



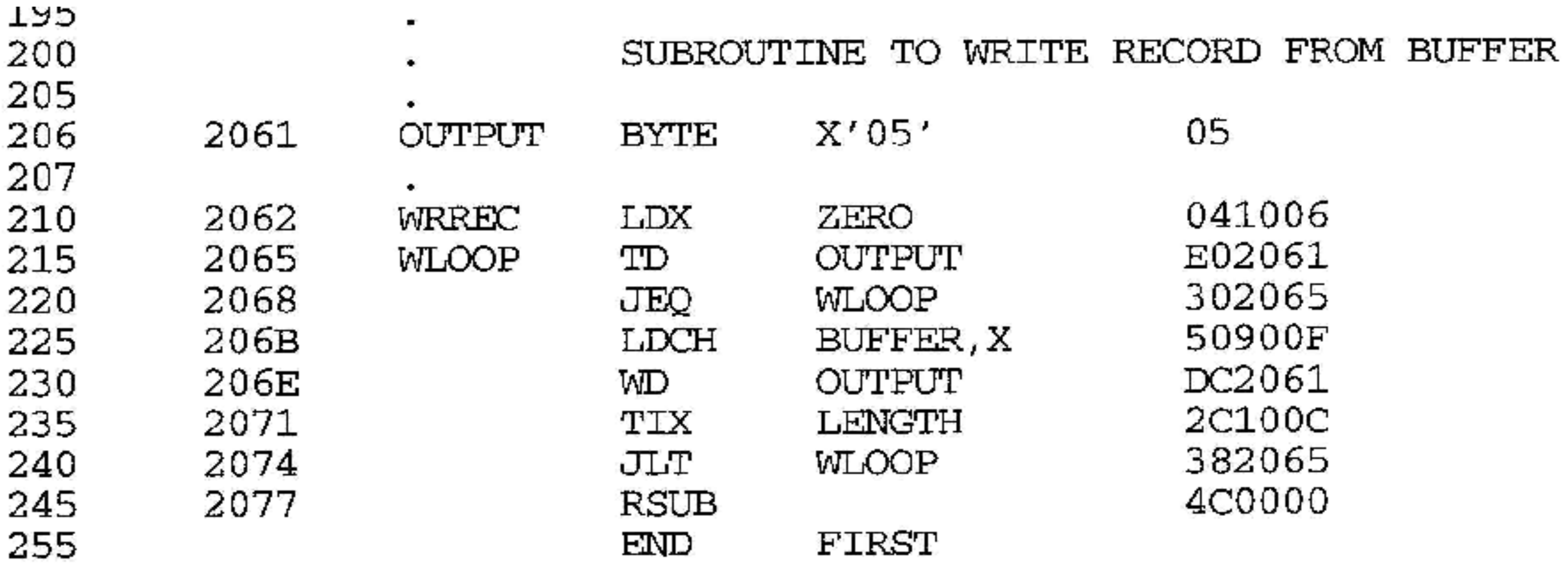
```
Line    Loc       Source statement                    Object code

  0     1000    COPY      START    1000
  1     1000    EOF       BYTE     C'EOF'             454F46
  2     1003    THREE     WORD     3                  000003
  3     1006    ZERO      WORD     0                  000000
  4     1009    RETADR    RESW     1
  5     100C    LENGTH    RESW     1
  6     100F    BUFFER    RESB     4096
  9             .
 10     200F    FIRST     STL      RETADR             141009
 15     2012    CLOOP     JSUB     RDREC              48203D
 20     2015              LDA      LENGTH             00100C
 25     2018              COMP     ZERO               281006
 30     201B              JEQ      ENDFIL             302024
 35     201E              JSUB     WRREC              482062
 40     2021              J        CLOOP              302012
 45     2024    ENDFIL    LDA      EOF                001000
 50     2027              STA      BUFFER             0C100F
 55     202A              LDA      THREE              001003
 60     202D              STA      LENGTH             0C100C
 65     2030              JSUB     WRREC              482062
 70     2033              LDL      RETADR             081009
 75     2036              RSUB                        4C0000
110
```

Sample Program for a One-Pass Assembler (Fig. 2.18) (Cont.)



```
110               .
115               .         SUBROUTINE TO READ RECORD INTO BUFFE
120               .
121    2039       INPUT     BYTE     X'F1'              F1
122    203A       MAXLEN    WORD     4096               001000
124               .
125    203D       RDREC     LDX      ZERO               041006
130    2040                 LDA      ZERO               001006
135    2043       RLOOP     TD       INPUT              E02039
140    2046                 JEQ      RLOOP              302043
145    2049                 RD       INPUT              D82039
150    204C                 COMP     ZERO               281006
155    204F                 JEQ      EXIT               30205B
160    2052                 STCH     BUFFER,X           54900F
165    2055                 TIX      MAXLEN             2C203A
170    2058                 JLT      RLOOP              382043
175    205B       EXIT      STX      LENGTH             10100C
180    205E                 RSUB                        4C0000
195               .
```

Sample Program for a One-Pass Assembler (Fig. 2.18) (Cont.)



```
195                      .
200                      .          SUBROUTINE TO WRITE RECORD FROM BUFFER
205                      .
206      2061      OUTPUT     BYTE       X'05'              05
207                      .
210      2062      WRREC      LDX        ZERO               041006
215      2065      WLOOP      TD         OUTPUT             E02061
220      2068                 JEQ        WLOOP              302065
225      206B                 LDCH       BUFFER,X           50900F
230      206E                 WD         OUTPUT             DC2061
235      2071                 TIX        LENGTH             2C100C
240      2074                 JLT        WLOOP              382065
245      2077                 RSUB                          4C0000
255                           END        FIRST
```

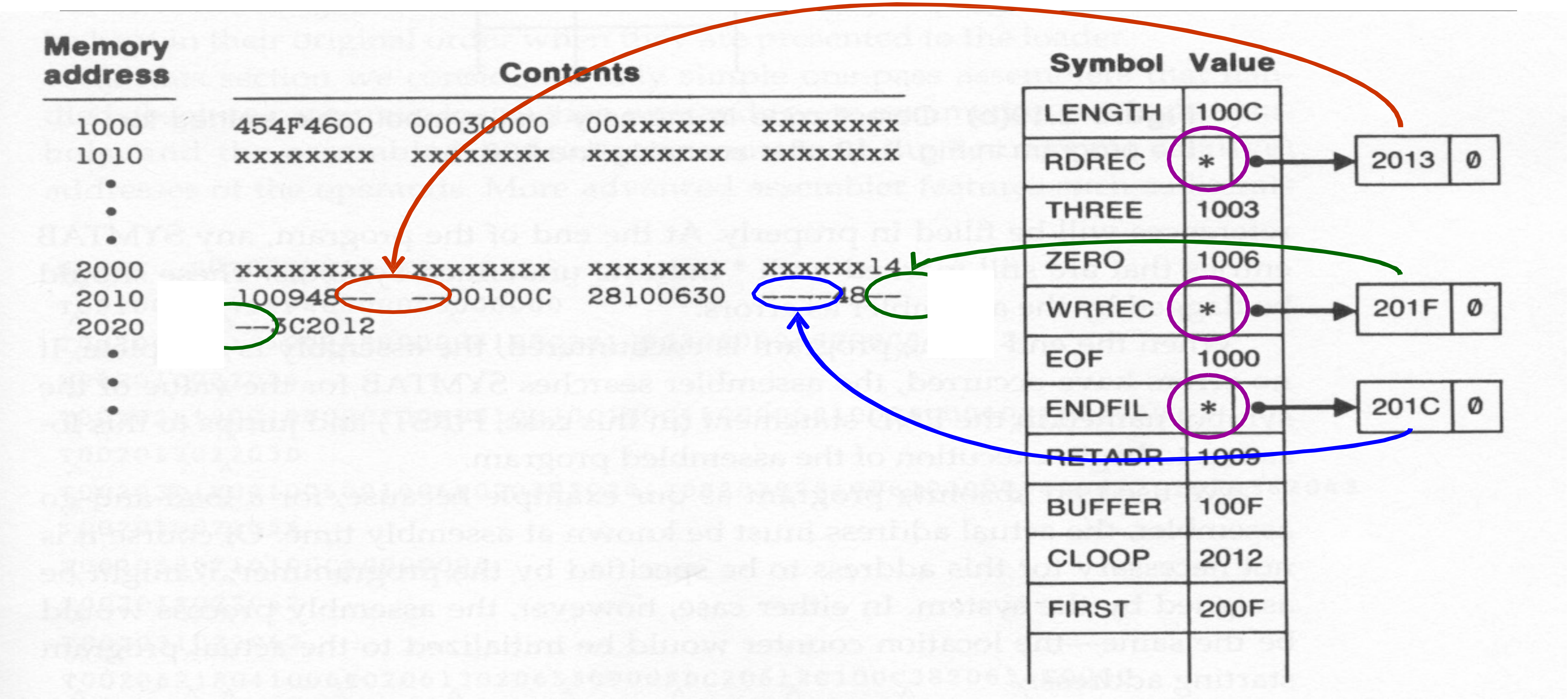
Figure 2.18 Sample program for a one-pass assembler.

Example

- Fig. 2.19 (a)
  - Show the object code in memory and symbol table entries after scanning line 40
  - Line 15: forward reference (RDREC)
    - Object code is marked ----
    - Value in symbol table is marked as * (undefined)
    - Insert *the address of operand* (2013) in a list associated with RDREC
  - Line 30 and Line 35: follow the same procedure

Object Code in Memory and SYMTAB

After scanning line 40



```
Memory
address                      Contents

 1000    454F4600  00030000  00xxxxxx  xxxxxxxx
 1010    xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx
   •
   •
   •
 2000    xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxx14
 2010    100948--  --00100C  28100630  ----48--
 2020    --3C2012
   •
   •
   •
```

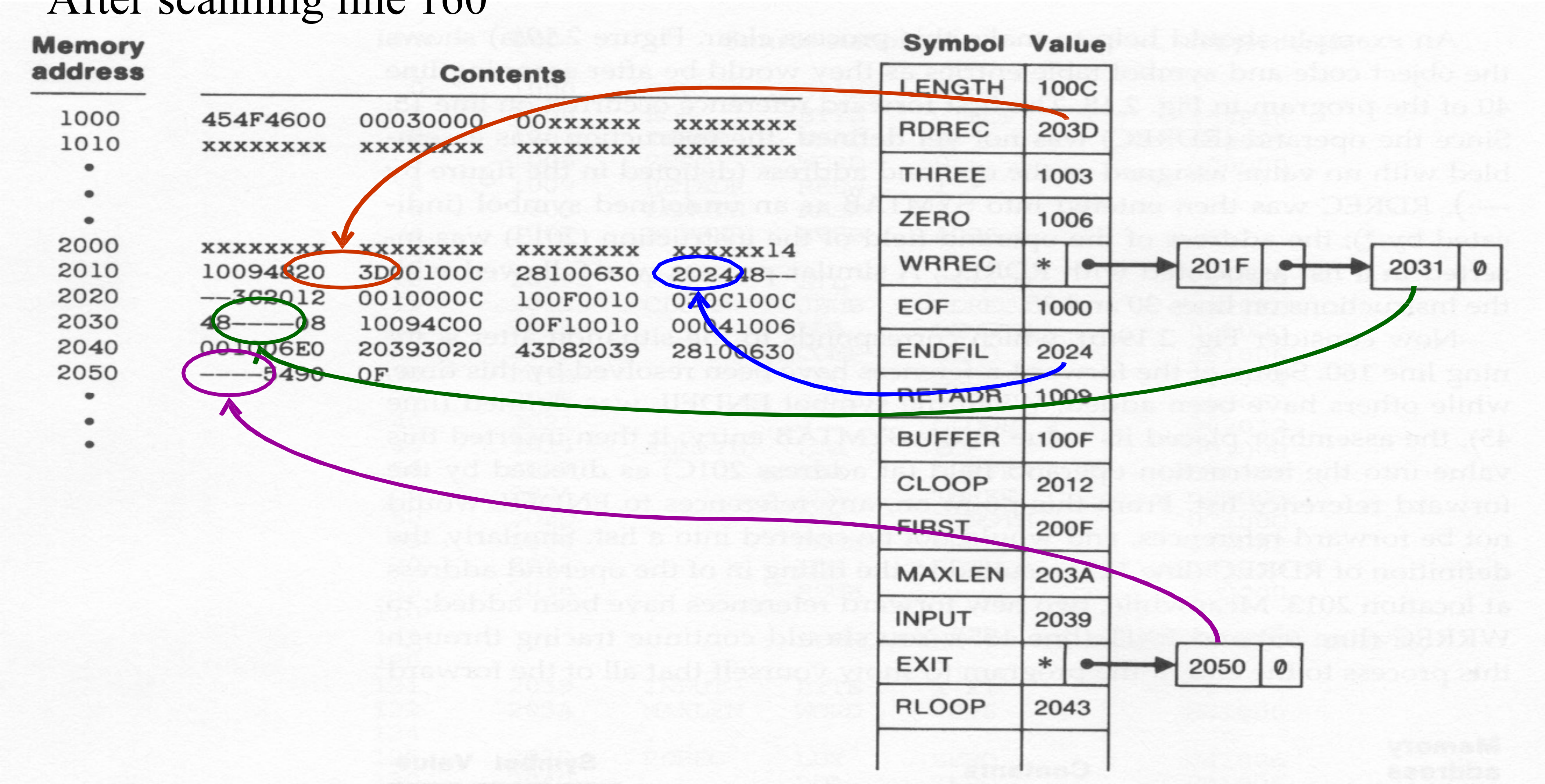
Symbol	Value	
LENGTH	100C	
RDREC	*	2013 Ø
THREE	1003	
ZERO	1006	
WRREC	*	201F Ø
EOF	1000	
ENDFIL	*	201C Ø
RETADR	1009	
BUFFER	100F	
CLOOP	2012	
FIRST	200F	

Example (Cont.)

- Fig. 2.19 (b)
  - Show the object code in memory and symbol table entries after scanning line 160
  - Line 45: ENDFIL was defined
    - Assembler place its value in the SYMTAB entry
    - Insert this value into the address (at 201C) as directed by the forward reference list
  - Line 125: RDREC was defined
    - Follow the same procedure
  - Line 65 and 155
    - Two new forward reference (WRREC and EXIT)

Object Code in Memory and SYMTAB

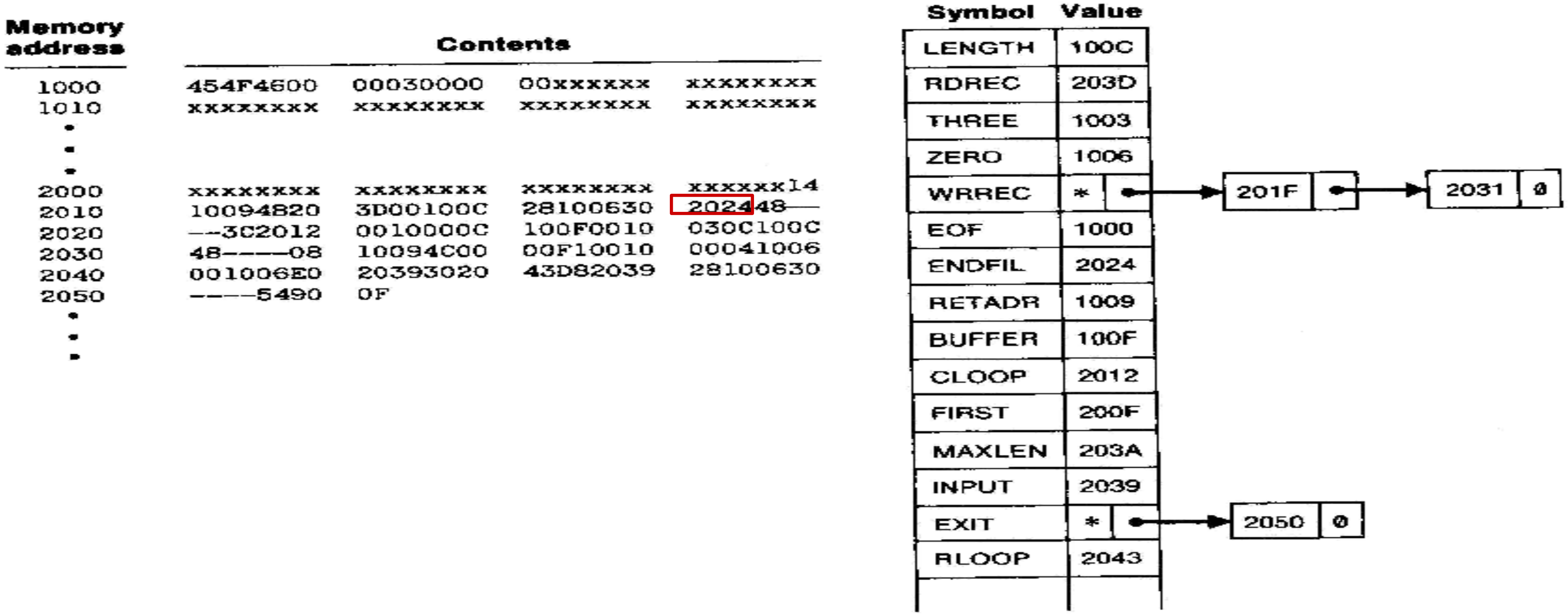
After scanning line 160



```
Memory
address          Contents

1000     454F4600  00030000  00xxxxxx  xxxxxxxx
1010     xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx
  •
  •
  •
2000     xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxx14
2010     10094820  3D00100C  28100630  202448--
2020     --3C2012  0010000C  100F0010  020C100C
2030     48----08  10094C00  00F10010  00041006
2040     001006E0  20393020  43D82039  28100630
2050     ----5490  0F
  •
  •
  •
```

Symbol	Value
LENGTH	100C
RDREC	203D
THREE	1003
ZERO	1006
WRREC	*
EOF	1000
ENDFIL	2024
RETADR	1009
BUFFER	100F
CLOOP	2012
FIRST	200F
MAXLEN	203A
INPUT	2039
EXIT	*
RLOOP	2043

Object Code in Memory and SYMTAB Entries for Fig 2.18 (Fig. 2.19b)



```
Memory
address                         Contents

 1000      454F4600   00030000   00xxxxxx   xxxxxxxx
 1010      xxxxxxxx   xxxxxxxx   xxxxxxxx   xxxxxxxx
   •
   •
   •
 2000      xxxxxxxx   xxxxxxxx   xxxxxxxx   xxxxxx14
 2010      10094820   3D00100C   28100630   202448--
 2020      --3C2012   0010000C   100F0010   030C100C
 2030      48----08   10094C00   00F10010   00041006
 2040      001006E0   20393020   43D82039   28100630
 2050      ----5490   0F
   •
   •
   •
```

Symbol	Value
LENGTH	100C
RDREC	203D
THREE	1003
ZERO	1006
WRREC	*
EOF	1000
ENDFIL	2024
RETADR	1009
BUFFER	100F
CLOOP	2012
FIRST	200F
MAXLEN	203A
INPUT	2039
EXIT	*
RLOOP	2043

Figure 2.19(b) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 160.

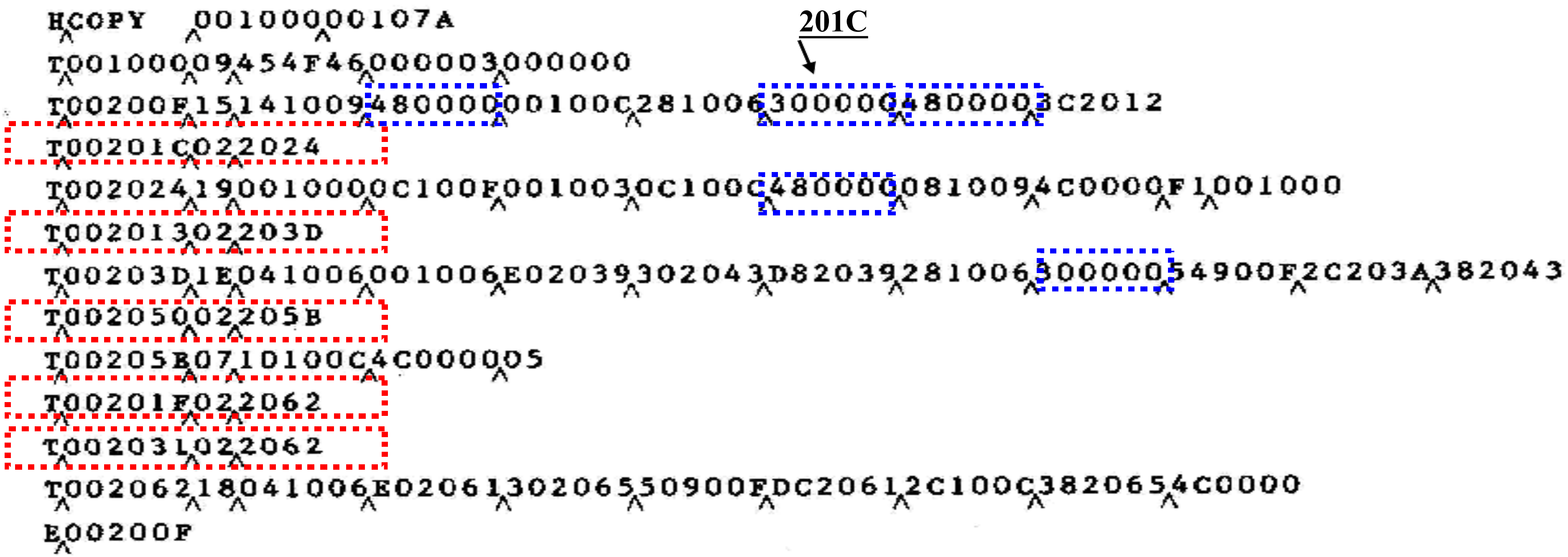
One-Pass Assembler Producing Object Code

- Forward reference are entered into the symbol table's list as before
  - If the operand contains an undefined symbol, use 0 as the address and write the Text record to the object program.
- However, when definition of a symbol is encountered, the assembler must generate *another Text record* with the *correct operand address*.
- When the program is loaded, this address will be inserted into the instruction by *loader*.
- The object program records must be kept in their original order when they are presented to the loader

Example

- In Fig. 2.20
  - Second Text record contains the object code generated from lines 10 through 40
    - The operand addressed for the instruction on line 15, 30, 35 have been generated as 0000
  - When the definition of ENDFIL is encountered
    - Generate the third Text record
      - Specify the value 2024 (the address of ENDFIL) is to be loaded at location 201C (the operand field of JEQ in line 30)
      - Thus, the value 2024 will replace the 0000 previously loaded

Object Program from one-pass assembler for Fig 2.18 (Fig 2.20)



```
H^COPY  ^001000^00107A
T^001000^09^454F46^000003^000000
T^00200F^15^141009^480000^00100C^281006^300000^480000^3C2012
T^00201C^02^2024
T^002024^19^001000^0C100F^001003^0C100C^480000^081009^4C0000^F1^001000
T^002013^02^203D
T^00203D^1E^041006^001006^E02039^302043^D82039^281006^300000^54900F^2C203A^382043
T^002050^02^205B
T^00205B^07^10100C^4C0000^05
T^00201F^02^2062
T^002031^02^2062
T^002062^18^041006^E02061^302065^50900F^DC2061^2C100C^382065^4C0000
E^00200F
```

Figure 2.20 Object program from one-pass assembler for program in Fig. 2.18.

2.4.2 Multi-Pass Assemblers

- Motivation: for a 2-pass assembler, any symbol used on the right-hand side should be defined previously.
  - No forward references since symbols' value can't be defined during the first pass
    - E.g.

```
APLHA       EQU   BETA
BETA        EQU   DELTA
DELTA       RESW 1
```

Not allowed !

Multi-Pass Assemblers (Cont.)

- Multi-pass assemblers
  - Eliminate the restriction on EQU and ORG
  - Make as many passes as are needed to process the definitions of symbols.
- Implementation
  - To facilitate symbol evaluation, in SYMTAB, each entry must indicates *which symbols are dependent on the values of it*
  - Each entry keeps a *linking list* to keep track of whose symbols' value depend on an this entry

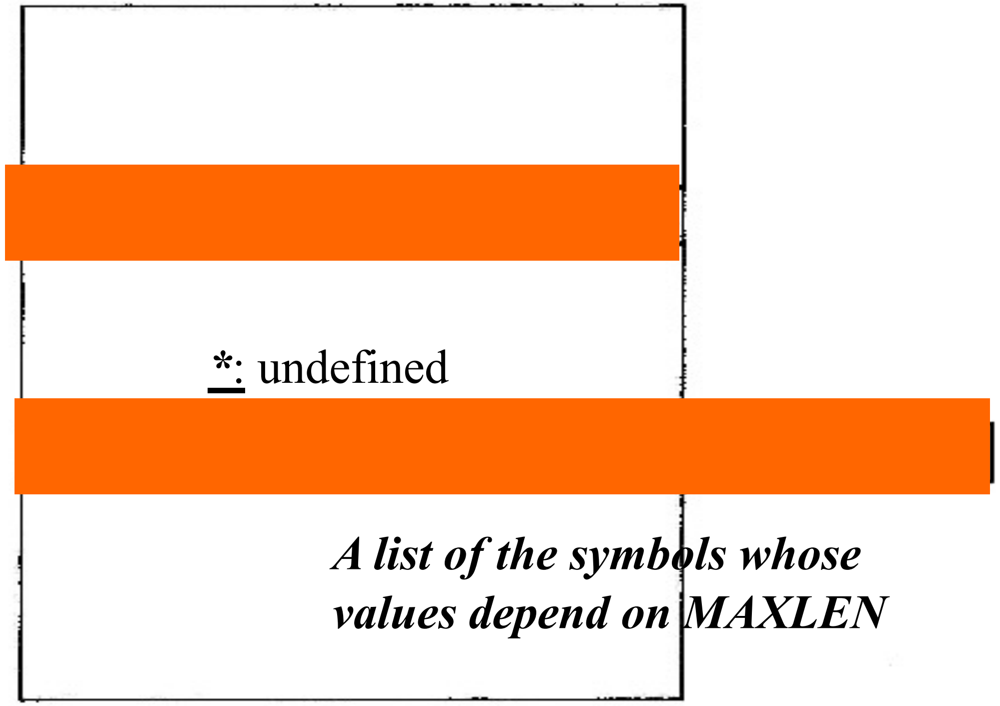
Example of Multi-pass Assembler Operation (fig 2.21a)

```
HALFSZ      EQU     MAXLEN/2  BUFEND-
MAXLEN      EQU     BUFFER  BUFFER-1
PREVBT      EQU

       .

       .

       .
  BUFFER      RESB     4096
  BUFEND      EQU       *
```

Example of Multi-Pass Assembler Operation (Fig 2.21b)

&1: one system in the defining expression is undefined

```
HALFSZ      EQU       MAXLEN/2  BUFEND-
MAXLEN      EQU       BUFFER  BUFFER-1
PREVBT      EQU
      .
      .
      .
  BUFFER    RESB      4096
BUFEND      EQU        *
```



*: undefined

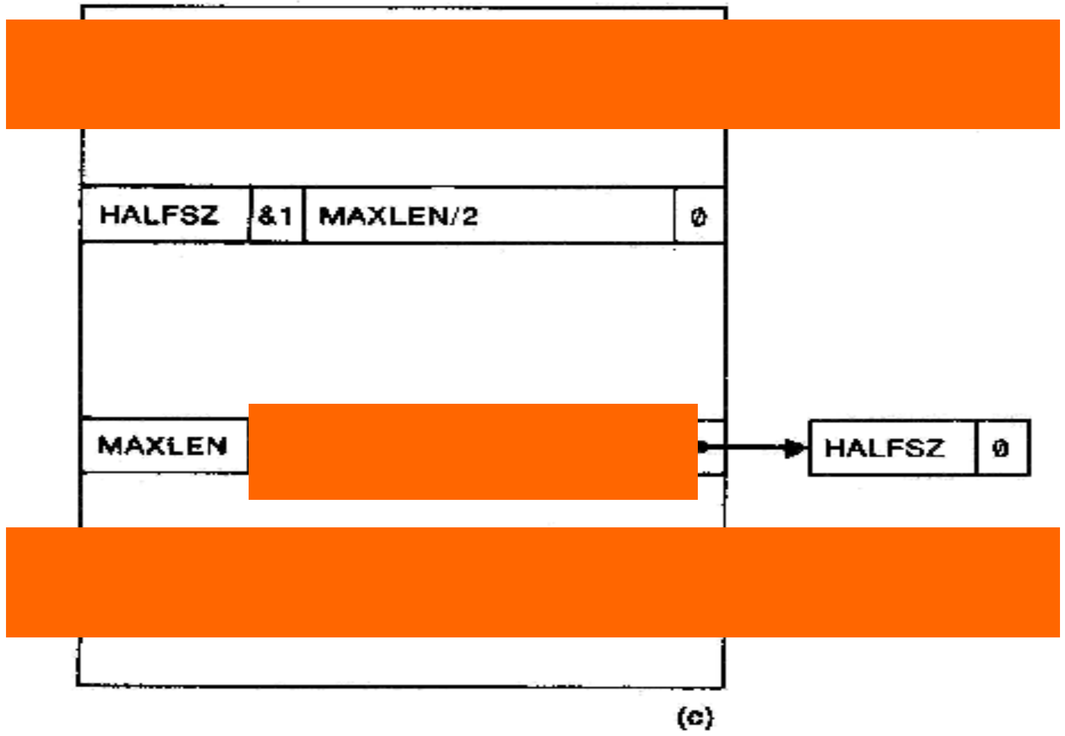
A list of the symbols whose values depend on MAXLEN

(b)

Figure 2.21 Example of multi-pass assembler operation.

Example of Multi-Pass Assembler Operation (Fig 2.21c)

```
HALFSZ        EQU         MAXLEN/2  BUFEND-
MAXLEN        EQU         BUFFER  BUFFER-1
PREVBT        EQU
          .
          .
          .
  BUFFER     RESB        4096
  BUFEND     EQU          *
```



Example of Multi-pass Assembler Operation (fig 2.21d)

```
HALFSZ      EQU      MAXLEN/2  BUFEND-
MAXLEN      EQU      BUFFER  BUFFER-1
PREVBT      EQU
        .
        .
        .
  BUFFER   RESB      4096
BUFEND     EQU        *
```

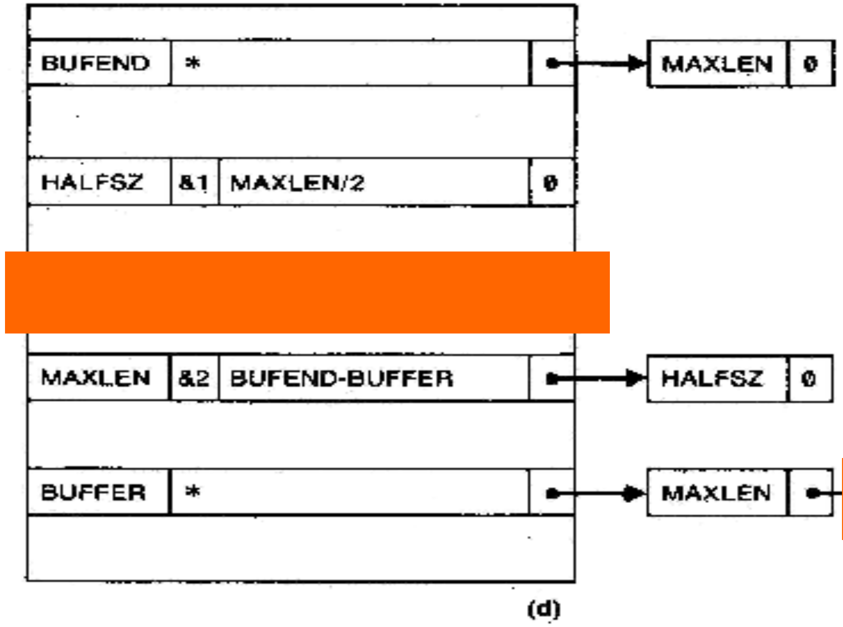
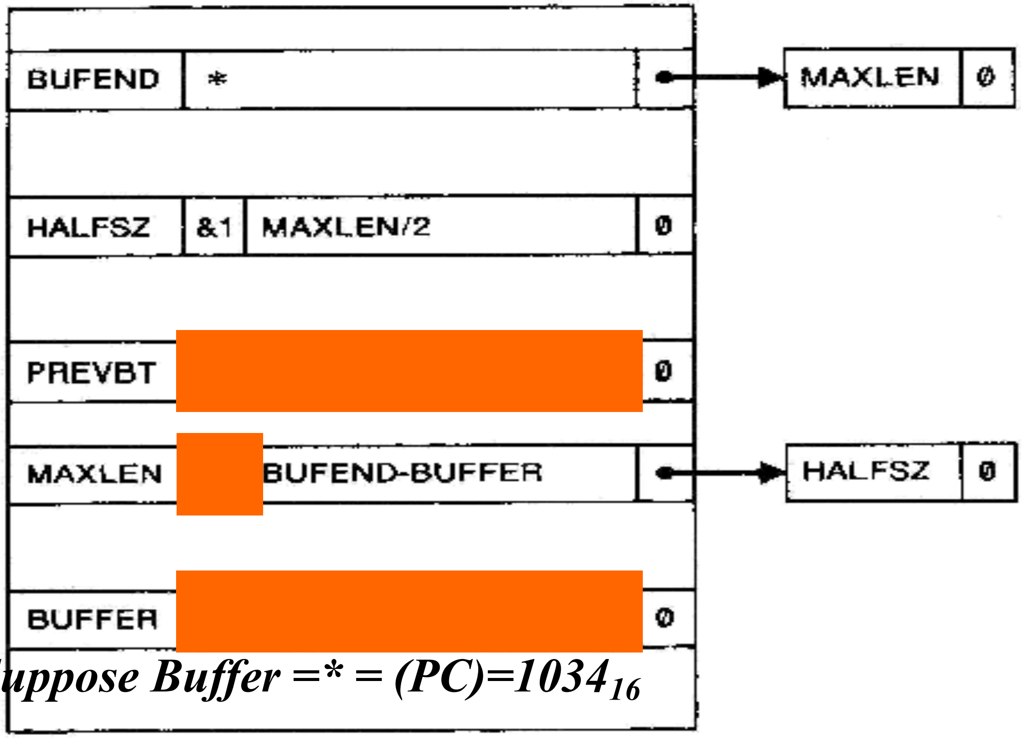


Figure 2.21 (cont'd)

Example of Multi-pass Assembler Operation (fig 2.21e)

```
HALFSZ       EQU        MAXLEN/2  BUFEND-
MAXLEN       EQU        BUFFER  BUFFER-1
PREVBT       EQU
        .
        .
        .
  BUFFER     RESB        4096
BUFEND       EQU         *
```



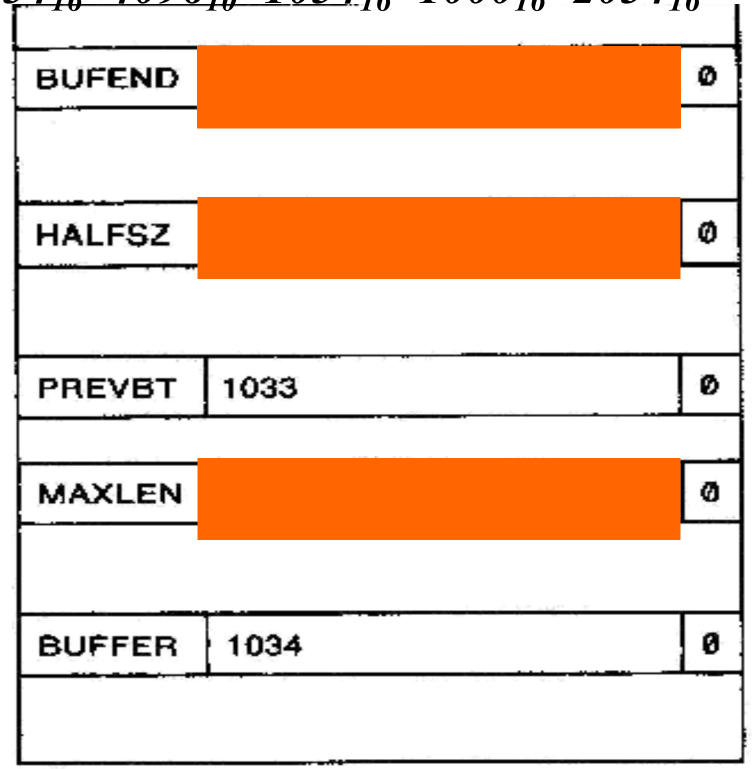
Suppose Buffer = = (PC)=1034_{16}*

(e)

Example of Multi-pass Assembler Operation (Fig 2.21f)

$BUFEND=*(PC)=1034_{16}+4096_{10}=1034_{16}+1000_{16}=2034_{16}$

```
HALFSZ      EQU      MAXLEN/2  BUFEND-
MAXLEN      EQU      BUFFER  BUFFER-1
PREVBT      EQU
       .
       .
       .
  BUFFER    RESB     4096
BUFEND      EQU      *
```



BUFEND		0
HALFSZ		0
PREVBT	1033	0
MAXLEN		0
BUFFER	1034	0

(f)

Figure 2.21 *(con'd)*

2.5 Implementation Examples

- Microsoft MASM Assembler
- Sun Sparc Assembler
- IBM AIX Assembler

2.5.1 Microsoft MASM Assembler

- Microsoft MASM assembler for Pentium and other x86 systems

- Programmer of an x86 system views memory as a collection of segments

Multi-Pass Assemblers

If we use a two-pass assembler, the following symbol definition cannot be allowed.

ALPHA EQU BETA

BETA EQU DELTA

DELTA RESW 1

This is because ALPHA and BETA cannot be defined in pass 1. Actually, if we allow multi-pass processing, DELTA is defined in pass 1, BETA is defined in pass 2, and ALPHA is defined in pass 3, and the above definitions can be allowed.

This is the motivation for using a multi-pass assembler.

Multi-Pass Assemblers

- It is unnecessary for a multi-pass assembler to make more than two passes over the entire program.
- Instead, only the parts of the program involving forward references need to be processed in multiple passes.
- The method presented here can be used to process any kind of forward references.
- Use a symbol table to store symbols that are not totally defined yet.
- For a undefined symbol, in its entry, – We store the names and the number of undefined symbols which contribute to the calculation of its value. – We also keep a list of symbols whose values depend on the defined value of this symbol.
- When a symbol becomes defined, we use its value to reevaluate the values of all of the symbols that are kept in this list.
- The above step is performed recursively.

Multi-Pass Assemblers

- **Examples**

Microsoft MASM Assembler, Sun Sparc Assembler, IBM AIX Assembler

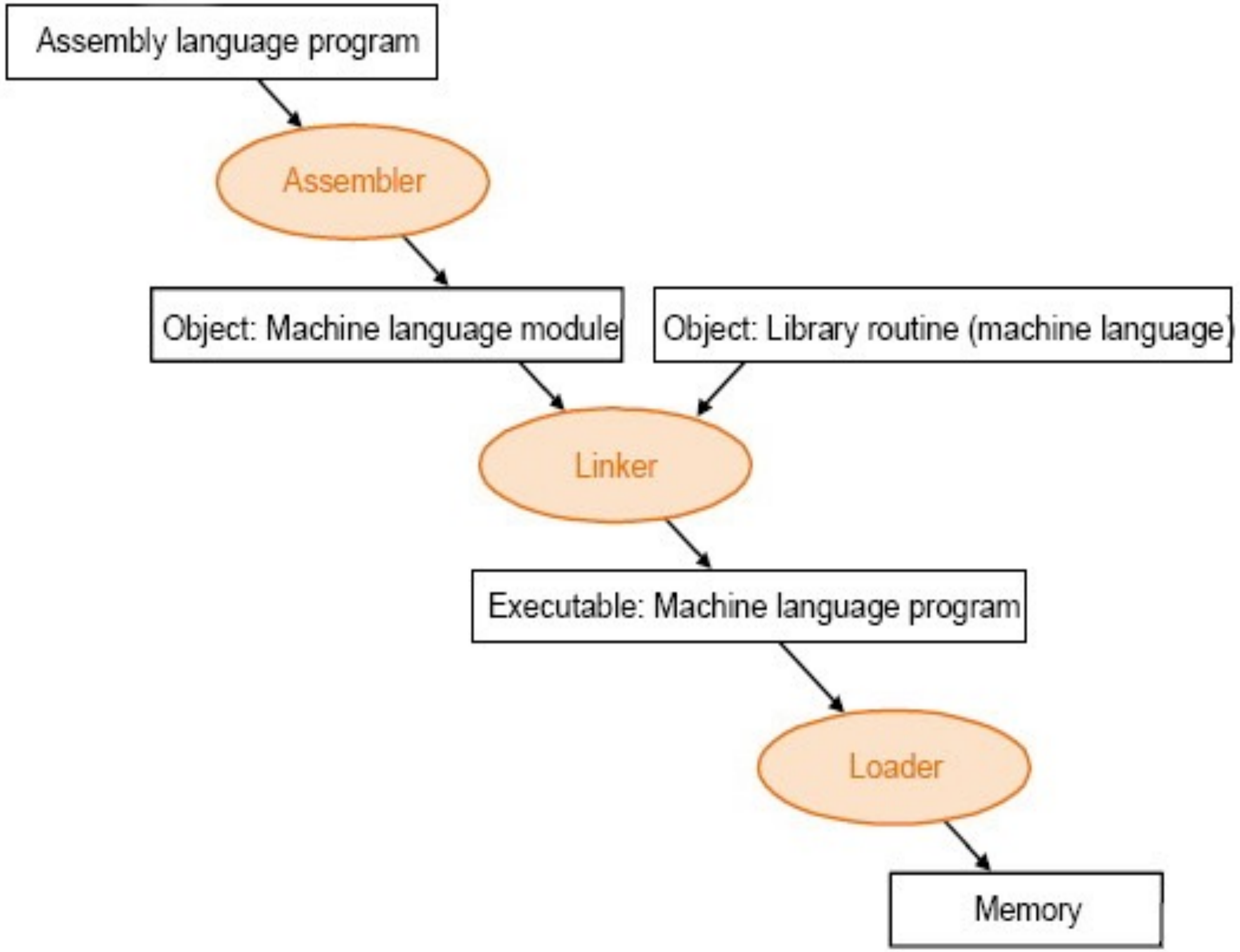
- **Microsoft MASM Assembler**
- SEGMENT - a collection segments, each segment is defined as belonging to a particular class, CODE, DATA, CONST, STACK
- registers: CS (code), SS (stack), DS (data), ES, FS, GS
- similar to program blocks in SIC l ASSUME

e. g. MOVE ES: DATASEG 2 AX, DATASEG 2 ES, AX » similar to BASE in SIC 11

Multi-Pass Assemblers

- l**Microsoft MASM Assembler (Contd.)**
- JUMP with forward reference
- near jump: 2 or 3 bytes
- far jump: 5 bytes
- e. g. JMP TARGET
- Warning: JMP FAR PTR TARGET
- Warning: JMP SHORT TARGET
- Pass 1: reserves 3 bytes for jump instruction phase error PUBLIC, EXTRN
- similar to EXTDEF, EXTREF in SIC 12

Advanced Assembly process



Assembly language program

Assembler

Object: Machine language module

Object: Library routine (machine language)

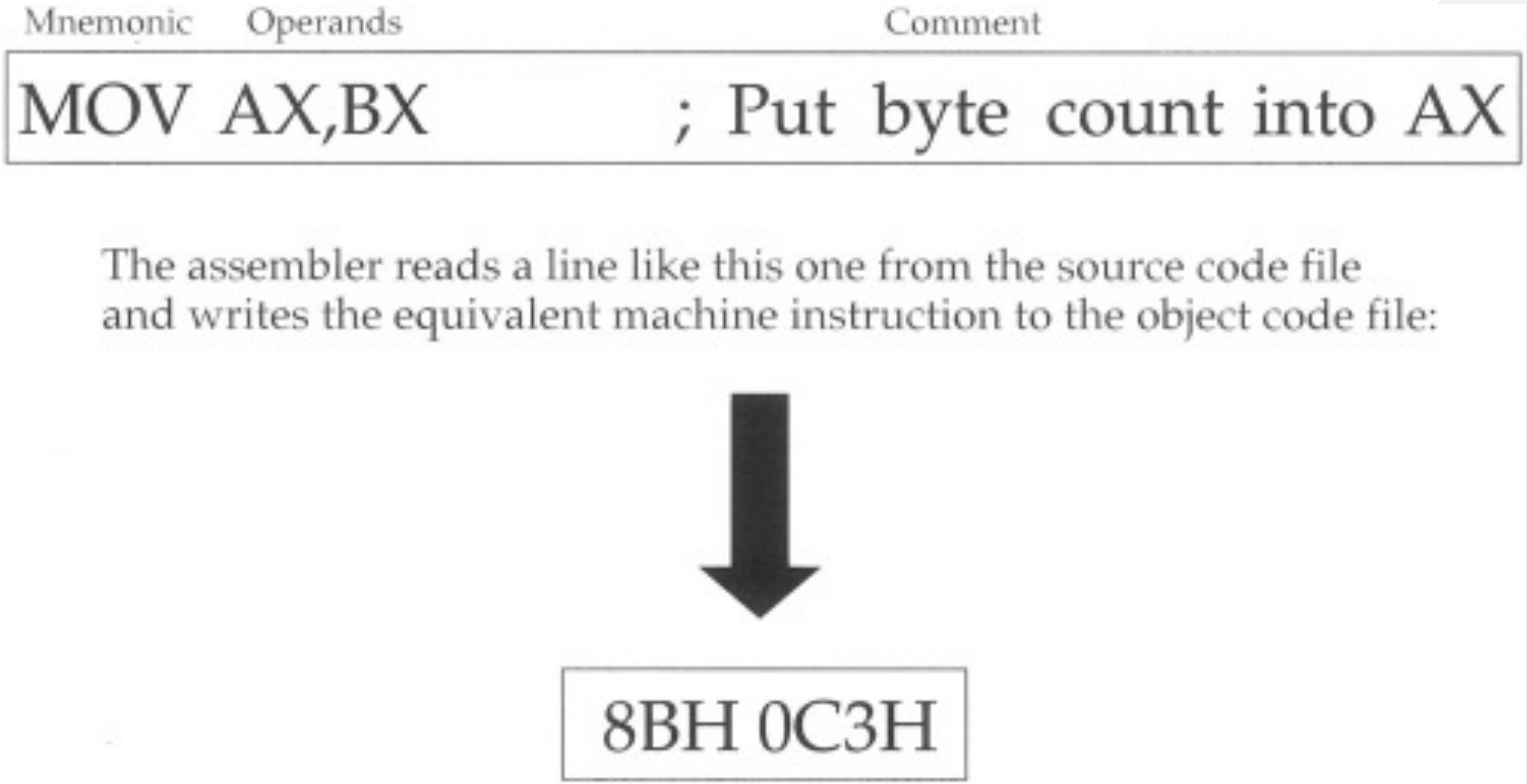
Linker

Executable: Machine language program

Loader

Memory

Advanced Assembly Process (Example)



Mnemonic Operands Comment

```
MOV AX,BX          ; Put byte count into AX
```

The assembler reads a line like this one from the source code file and writes the equivalent machine instruction to the object code file:

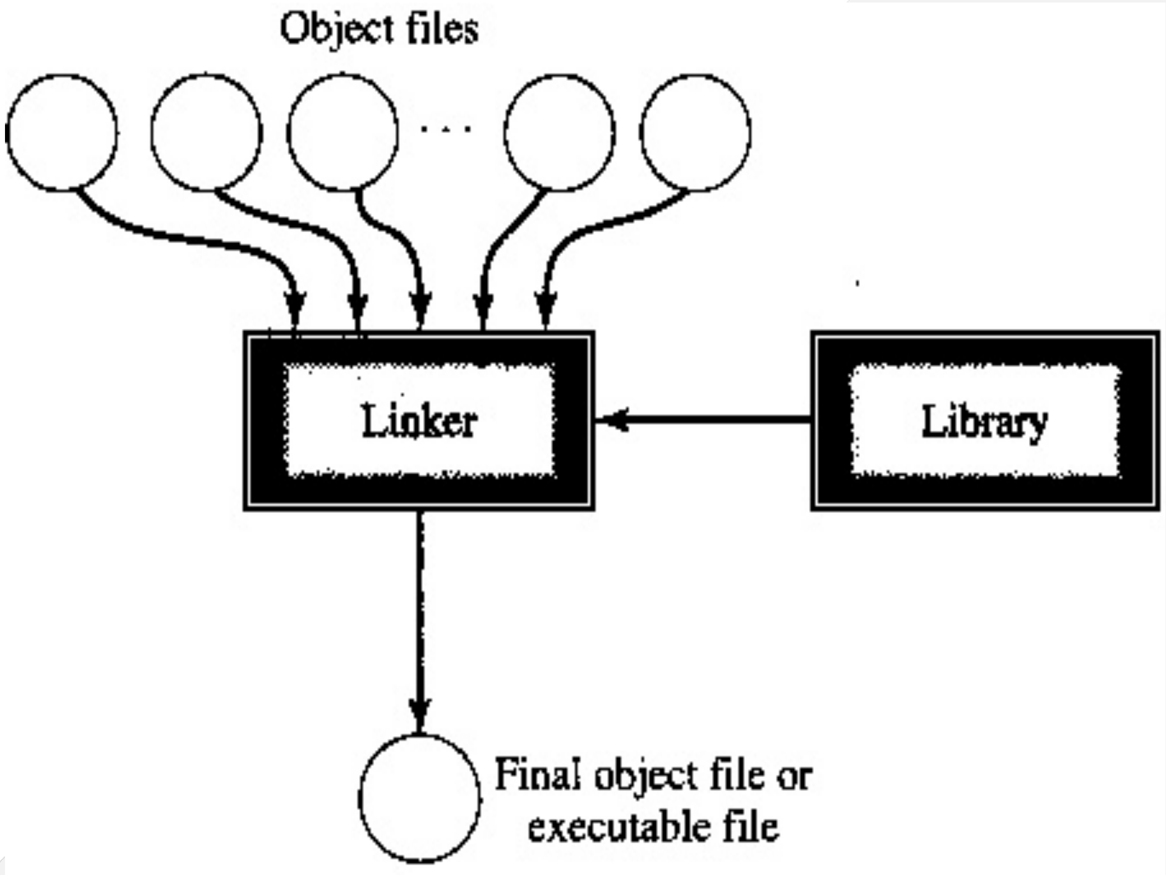
```
8BH 0C3H
```

Advanced Assembly Process

- **Assembling**
- At assembly time, the assembler:
  - Evaluates conditional-assembly directives, assembling if the conditions are true.
  - Expands macros and macro functions.
  - Evaluates constant expressions such as **MYFLAG AND 80H**, substituting the calculated value for the expression.
  - Encodes instructions and non address operands. For example, **mov cx, 13;** can be encoded at assembly time because the instruction does not access memory.
  - Saves memory offsets as offsets from their segments.
  - Places segments and segment attributes in the object file.
  - Saves placeholders for offsets and segments (relocatable addresses).
  - Outputs a listing if requested.
- Passes messages (such as INCLUDELIB) directly to the linker.

Advanced Assembly Process

- Once your source code is assembled, the resulting object file is passed to the linker. At this point, the linker may combine several object files into an executable program. The linker:
  - Combines segments according to the instructions in the object files, rearranging the positions of segments that share the same class or group.
  - Fills in placeholders for offsets (relocatable addresses).
  - Writes relocations for segments into the header of .EXE files (but not .COM files).
  - Writes the result as an executable program file.



Advanced Assembly Process

- **Loading**

  After loading the executable file into memory, the operating system:

  - Creates the program segment prefix (PSP) header in memory.
  - Allocates memory for the program, based on the values in the PSP.
  - Loads the program.
  - Calculates the correct values for absolute addresses from the relocation table.
  - Loads the segment registers SS, CS, DS, and ES with values that point to the proper areas of memory.

Advanced Assembly Process

Useful Tools and Utilities

- DUMPBIN disassembly program
- Debuggers: OllyDbg and WinDbg
- Consol I/O: iolib.

References

- [PDF] Systems Programming and Operating Systems by Dhamdhere - Free Download PDF (dlscrib.com)
- [PDF] Principles of Compiler Design By Alfred V. Aho & J.D.Ullman Free Download – Learnengineering.in

THANK YOU