KCG College of Technology,Chennai-96
Computer Science and Engineering

CS 2304 SYSTEM SOFTWARE III Sem CSE

QUESTION BANK - UNIT-II ASSEMBLERS

---

1) Why an Assembly Language is needed?

Programming in machine code, by supplying the computer with the numbers of the operations it must perform, can be quite a burden, because for every operation the corresponding number must be looked up or remembered. Looking up all numbers takes a lot of time, and mis-remembering a number may introduce computer bugs.

So Assembly Languages are evolved which contains mnemonic instructions corresponding to the Machine codes using which the program can be written easily.

Therefore a set of mnemonics was devised. Each number was represented by an alphabetic code. So instead of entering the number corresponding to addition to add two numbers one can enter "add".

Although mnemonics differ between different CPU designs some are common, for instance: "sub" (subtract), "div" (divide), "add" (add) and "mul" (multiply).

2) What is an Assembler?

An assembler is a program that accepts an *assembly language program* as input and produces its *machine language equivalent* along with *information for the loader*
(An Assembler translates a program written in an assembly language to it machine language equivalent)

3) Explain the terms a)Label,b)Opcode,c)Operand,and d)Comment
(What is the format in which the assembly language program is written?).

- **Label field**.
  - The label is a symbolic name that represents the memory address of an executable statement or a variable.
- **Opcode/directive fields**.
  - The opcode (e.g. operation code) specifies the symbolic name for a machine instruction.
  - The directive specifies commands to the assembler about the way to assemble the program.
- **Operand field.**
  - The operand specifies the data that is needed by a statement.
- **Comment field.**
  - The comment provides clear explanation for a statement.

4) **What are the basic functions of an assembler?**

Functions of a Basic Assembler

- Convert *mnemonic operation codes* to their *machine language equivalents*
  E.g. STL -> 14 (line 10)
- Convert *symbolic operands* to their equivalent *machine addresses*
  E.g. RETADR -> 1033 (line 10)
- Build the machine instructions in the proper format
- Convert the *data constants* to *internal machine representations*
  E.g. EOF -> 454F46 (line 80)
- Write the *object program* and the *assembly listing*

5) **What are assembler Directives?**

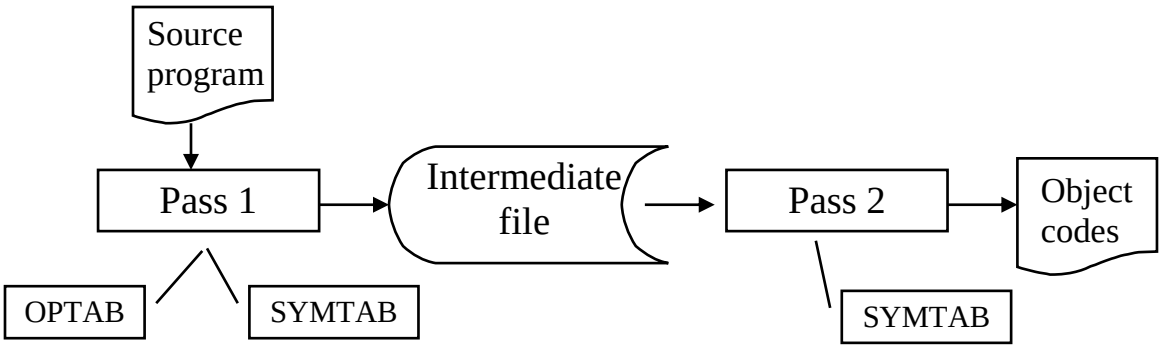
Assembler directives are **Pseudo-instructions that are n**ot translated into machine instructions and they provide instructions to the assembler itself.

- **The SIC assembler directives.**
  - START
    - Specification of the name and start address of the program.
  - END
    - Indication of the end of the program and optionally the address of the first executable instruction.
  - BYTE
    - Declaration of character or string constants.
  - WORD
    - Declaration of integer constants.
  - RESB
    - Declaration of character variables or arrays.
  - RESW
    - Declaration of integer variables or arrays.

6) **What are the functions of two pass assembler?**

Functions of Two Pass Assembler

- **Pass 1 - define symbols (assign addresses)**
  - Assign addresses to all statements in the program
  - Save the values assigned to all labels for use in Pass 2
  - Process some assembler directives
- **Pass 2 - assemble instructions and generate object program**
  Assemble instructions
  - Generate data values defined by BYTE, WORD, etc.
  - Process the assembler directives not done in Pass 1
  - Write the object program and the assembly listing



7) What is the format of the Object Program generated by the Assembler?

Contains 3 types of records:

Header record:

Col. 1 H
Col. 2-7 Program name
Col. 8-13 Starting address (hex)
Col. 14-19 Length of object program in bytes (hex)

Text record

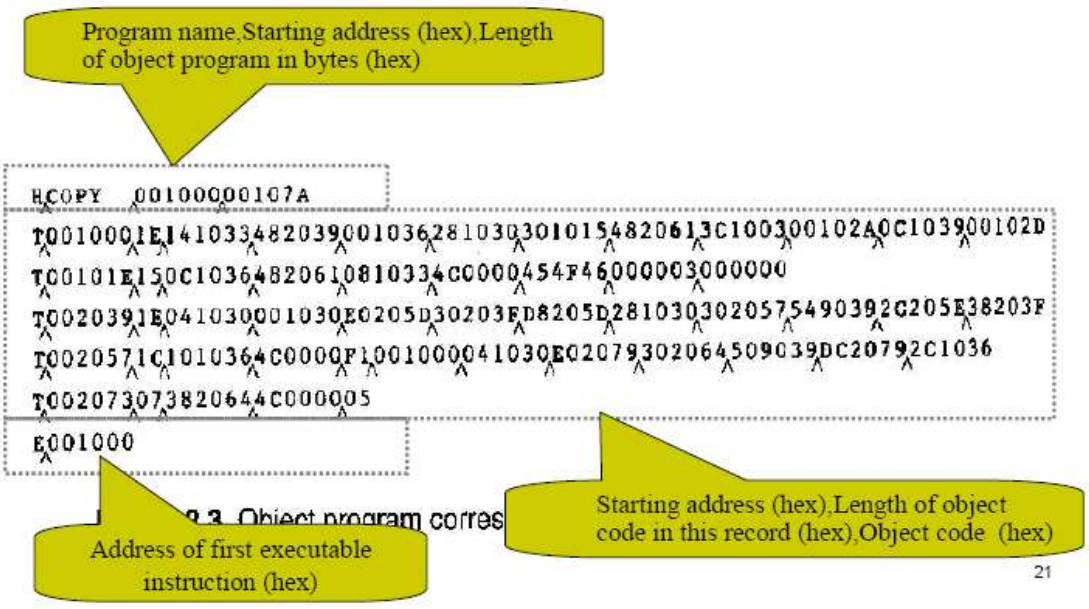
Col.1 T
Col.2-7 Starting address in this record (hex)
Col. 8-9 Length of object code in this record in bytes (hex)
Col. 10-69 Object code (hex) (2 columns per byte)

End record

Col.1 E
Col.2~7 Address of first executable instruction (hex)
(END program_name)

8) Give an example of object program generated by an Assembler.

Object Program for Fig 2.2 (Fig 2.3)



9) What is forward reference?

Forward reference *is a* reference to a label that is defined later in the program.

Example

10 STL RETADR

o ***RETADR*** is not yet defined when we encounter *STL* instruction

o So it is called *forward reference*

10) Give an example of Assembly language along with the objectcode generated.

Line	Loc	Source statement			Object code
5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
50	1018		STA	BUFFER	0C1039
55	101B		LDA	THREE	00102D
60	101E		STA	LENGTH	0C1036
65	1021		JSUB	WRREC	482061
70	1024		LDL	RETADR	081033

```
75    1027                RSUB                    4C0000
80    102A  EOF     BYTE        C'EOF'    454F46
85    102D  THREE   WORD          3        000003
90    1030  ZERO    WORD          0        000000
95    1033  RETADR  RESW          1
100   1036  LENGTH  RESW          1
105   1039  BUFFER  RESB        4096
110         .
115         .       SUBROUTINE TO READ RECORD INTO BUFFER
120         .
125   2039  RDREC   LDX         ZERO      041030
130   203C          LDA         ZERO      001030
135   203F  RLOOP   TD          INPUT     E0205D
140   2042          JEQ         RLOOP     30203D
145   2045          RD          INPUT     D8205D
150   2048          COMP        ZERO      281030
155   204B          JEQ         EXIT      302057
160   204E          STCH        BUFFER,X  549039
165   2051          TIX         MAXLEN    2C205E
170   2054          JLT         RLOOP     38203F
175   2057  EXIT    STX         LENGTH    101036
180   205A          RSUB                  4C0000
185   205D  INPUT   BYTE X'F1'  F1
190   205E  MAXLEN  WORD        4096   001000
195         .
200         .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205         .
210   2061  WRREC   LDX   ZERO  041030
215   2064  WLOOP   TD    OUTPUT    E02079
220   2067          JEQ   WLOOP     302064
225   206A          LDCH  BUFFER,X  509039
230   206D          WD    OUTPUT    DC2079
235   2070          TIX   LENGTH    2C1036
240   2073          JLT   WLOOP     382064
245   2076          RSUB            4C0000
250   2079  OUTPUT  BYTE X'05'  05
255                 END   FIRST
```

10) Write an Algorithm for pass 1 of SIC Assembler.

```
Pass 1:

begin
   read first input line
   if OPCODE = 'START' then
       begin
           save #[OPERAND] as starting address
           initialize LOCCTR to starting address
           write line to intermediate file
           read next input line
       end {if START}
   else
       initialize LOCCTR to 0
   while OPCODE ≠ 'END' do
       begin
           if this is not a comment line then
               begin
                   if there is a symbol in the LABEL field then
                       begin
                           search SYMTAB for LABEL
                           if found then
                               set error flag (duplicate symbol)
                           else
                               insert (LABEL,LOCCTR) into SYMTAB
                       end {if symbol}
                   search OPTAB for OPCODE
                   if found then
                       add 3 {instruction length} to LOCCTR
                   else if OPCODE = 'WORD' then
                       add 3 to LOCCTR
                   else if OPCODE = 'RESW' then
                       add 3 * #[OPERAND] to LOCCTR
                   else if OPCODE = 'RESB' then
                       add #[OPERAND] to LOCCTR
                   else if OPCODE = 'BYTE' then
                       begin
                           find length of constant in bytes
                           add length to LOCCTR
                       end {if BYTE}
                   else
                       set error flag (invalid operation code)
               end {if not a comment}
           write line to intermediate file
           read next input line
       end {while not END}
   write last line to intermediate file
   save (LOCCTR - starting address) as program length
end {Pass 1}
```

Figure 2.4(a) Algorithm for Pass 1 of assembler.

11) Write an algorithm for pass 2 of SIC assembler.

Pass 2:

```
begin
   read first input line (from intermediate file)
   if OPCODE = 'START' then
      begin
         write listing line
         read next input line
      end (if START)
   write Header record to object program
   initialize first Text record
   while OPCODE ≠ 'END' do
      begin
         if this is not a comment line then
            begin
               search OPTAB for OPCODE
               if found then
                  begin
                     if there is a symbol in OPERAND field then
                        begin
                           search SYMTAB for OPERAND
                           if found then
                              store symbol value as operand address
                           else
                              begin
                                 store 0 as operand address
                                 set error flag (undefined symbol)
                              end
                        end (if symbol)
                     else
                        store 0 as operand address
                     assemble the object code instruction
                  end (if opcode found)
               else if OPCODE = 'BYTE' or 'WORD' then
                  convert constant to object code
               if object code will not fit into the current Text record then
                  begin
                     write Text record to object program
                     initialize new Text record
                  end
               add object code to Text record
            end (if not comment)
         write listing line
         read next input line
      end (while not END)
   write last Text record to object program
   write End record to object program
   write last listing line
end (Pass 2)
```

Figure 2.4(b) Algorithm for Pass 2 of assembler.

12) What are the Data Structures used in an Assembler?

Data Structures:

Operation Code Table (OPTAB)
Symbol Table (SYMTAB)
Location Counter(LOCCTR)

13) Explain the features of a Symbol Table.

- **SYMTAB (symbol table)**
- **Content**
  Label name and its value (address)
  May also include flag (type, length) etc.
- **Usage**
  Pass 1: labels are entered into SYMTAB with their address (from LOCCTR) as they are encountered in the source program
  Pass 2: symbols used as operands are looked up in SYMTAB to obtain the address to be inserted in the assembled instruction
- **Characteristic**
  Dynamic table (insert, delete, search)
- **Implementation**
  Hash table for efficiency of *insertion* and *retrieval*

COPY	**1000**
FIRST	**1000**
CLOOP	**1003**
ENDFIL	**1015**
EOF	**1024**
THREE	**102D**
ZERO	**1030**
RETADR	**1033**
LENGTH	**1036**
BUFFER	**1039**
RDREC	**2039**

SYMBOL TABLE(SYMTAB)

14) What is Location Counter?

Location Counter

- A variable used to help in *assignment of addresses*
- Initialized to the beginning address specified in the START statement
- Counted in bytes

15) What are the machine dependant fetures of a SIC/XE Assembler?

Machine-dependent features of assemblers

Features of the SIC/XE machine

Programming features.

a. # symbol.
    i. Indication of the immediate addressing mode.
    ii. Immediate addressing provides a faster access to an operand reference.
b. @ symbol.
    i. Indication of the indirect addressing mode.
    ii. Indirect addressing reduces the number of instructions.
c. + symbol.
    i. Explicit selection of the format 4 instruction with a direct addressing mode.
    ii. Format 4 is selected when the 12-bit displacement of format 3 is too small.
d. BASE directive.
    i. Indication that the base register B holds a base address used in a base addressing.
    ii. NOBASE directive disables the base register.
    iii. LDB instruction loads the base register with a base address.
e. Register-to-register addressing.
    i. Register addressing reduces the size of a machine instruction and speeds up a computation

Assembling features.

f. Multiprogramming.
    i. Larger memory allows us to load many programs.
    ii. The object code is relative to zero because the load address is variable.
    iii. Program must be relocated when it is loaded in memory.
g. Register set mapping.
    i. A separate register table can store the numeric values of the registers.
    ii. The numeric values of the registers can be preloaded with the symbol table.
h. Relative (PC and base) addressing mode.
    i. Operand value is subtracted from PC or base register value.
    ii. PC relative addressing provides a displacement from –2048 to +2047.
    iii. Base relative addressing provides a displacement from 0 to 4095.

16) What is Program Relocation?

Program relocation

- **Principles.**
  - The load address of an object program is unknown at assembly time if the system implements the multiprogramming feature.
  - The assembler generates addresses relative to zero in the object program.
  - At load time, relocation is performed by adding the load address to the relative addresses.
  - Operands of instructions that use direct addressing must be relocated, and the assembler provides the relocation information in the object program.
  - Operands of instructions that use relative addressing do not need to be relocated.
  - Relocation can be processed by the loader or by the CPU using relocation registers.

17) What are the advantages of program relocation?

Program Relocation

- The larger main memory of SIC/XE
  - Several programs can be loaded and run at the same time.
  - This kind of sharing of the machine between programs
  - is called ***multiprogramming***
- To take full advantage
  - Load programs into memory wherever there is room
  - Not specifying a fixed address at assembly time
  - Called ***program relocation***

18) What are program blocks?

Program Blocks

- Refer to segments of code that are rearranged within a single object program unit
- USE [blockname]
- At the beginning, statements are assumed to be part of the unnamed (default) block
- If no USE statements are included, the entire program belongs to this single block
- Each program block may actually contain several separate segments of the source program

19) How the program blocks are assembled?

Program Blocks - Implementation

- Pass 1
  - Each program block has a separate location counter
  - Each label is assigned an address that is relative to the start of the block that contains it
  - At the end of Pass 1, the latest value of the location counter for each block indicates the length of that block
  - The assembler can then assign to each block a starting address in the object program
- Pass 2
  - The address of each symbol can be computed by adding the assigned block starting address and the relative address of the symbol to that block

```
Line        Source statement
  5     COPY        START   0               COPY FILE FROM INPUT TO OUTPUT
 10     FIRST       STL     RETADR          SAVE RETURN ADDRESS
 15     CLOOP       JSUB    RDREC           READ INPUT RECORD
 20                 LDA     LENGTH          TEST FOR EOF (LENGTH = 0)
 25                 COMP    #0
 30                 JEQ     ENDFIL          EXIT IF EOF FOUND
 35                 JSUB    WRREC           WRITE OUTPUT RECORD
 40                 J       CLOOP           LOOP
 45     ENDFIL      LDA     =C'EOF'         INSERT END OF FILE MARKER
 50                 STA     BUFFER
 55                 LDA     #3              SET LENGTH = 3
 60                 STA     LENGTH
 65                 JSUB    WRREC           WRITE EOF
 70                 J       @RETADR         RETURN TO CALLER
 92                 USE     CDATA
 95     RETADR      RESW    1
100     LENGTH      RESW    1               LENGTH OF RECORD
103                 USE     CBLKS
105     BUFFER      RESB    4096            4096-BYTE BUFFER AREA
106     BUFEND      EQU     *               FIRST LOCATION AFTER BUFFER
107     MAXLEN      EQU     BUFEND-BUFFER   MAXIMUM RECORD LENGTH
110     .                                                   .
115     .           SUBROUTINE TO READ RECORD INTO BUFFER
120     .
123                 USE
125     RDREC       CLEAR   X               CLEAR LOOP COUNTER
130                 CLEAR   A               CLEAR A TO ZERO
132                 CLEAR   S               CLEAR S TO ZERO
133                 +LDT    #MAXLEN
135     RLOOP       TD      INPUT           TEST INPUT DEVICE
140                 JEQ     RLOOP           LOOP UNTIL READY
145                 RD      INPUT           READ CHARACTER INTO REGISTER A
150                 COMPR   A,S             TEST FOR END OF RECORD (X'00')
155                 JEQ     EXIT            EXIT LOOP IF EOR
160                 STCH    BUFFER,X        STORE CHARACTER IN BUFFER
165                 TIXR    T               LOOP UNLESS MAX LENGTH
170                 JLT     RLOOP             HAS BEEN REACHED
175     EXIT        STX     LENGTH          SAVE RECORD LENGTH
180                 RSUB                    RETURN TO CALLER
183                 USE     CDATA
185     INPUT       BYTE    X'F1'           CODE FOR INPUT DEVICE
195     .                                                   .
200     .           SUBROUTINE TO WRITE RECORD FROM BUFFER
205     .
208                 USE
210     WRREC       CLEAR   X               CLEAR LOOP COUNTER
212                 LDT     LENGTH
215     WLOOP       TD      =X'05'          TEST OUTPUT DEVICE
220                 JEQ     WLOOP           LOOP UNTIL READY
225                 LDCH    BUFFER,X        GET CHARACTER FROM BUFFER
230                 WD      =X'05'          WRITE CHARACTER
235                 TIXR    T               LOOP UNTIL ALL CHARACTERS
240                 JLT     WLOOP             HAVE BEEN WRITTEN
245                 RSUB                    RETURN TO CALLER
252                 USE     CDATA
253                 LTORG
255                 END     FIRST
```

Figure 2.11 Example of a program with multiple program blocks.

- Each source line is given a relative address assigned and a block number

Block name	Block number	Address	Length
(default)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

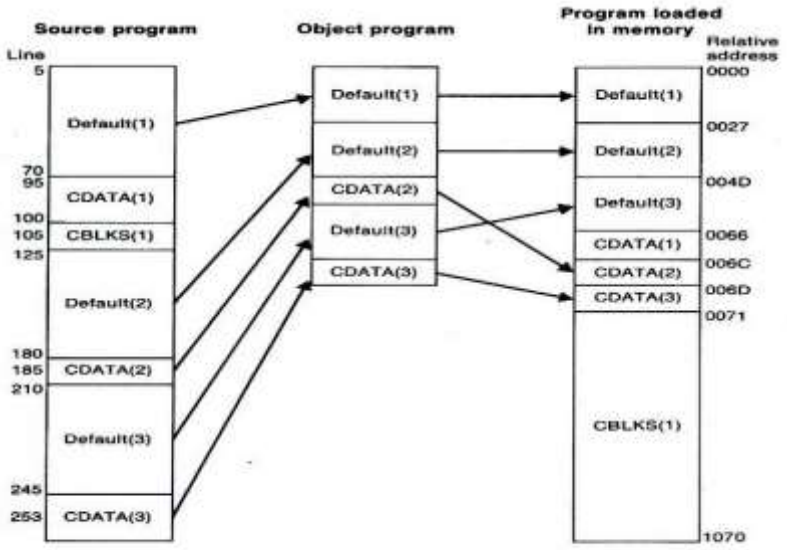


Figure 2.14 Program blocks from Fig. 2.11 traced through the assembly and loading processes.

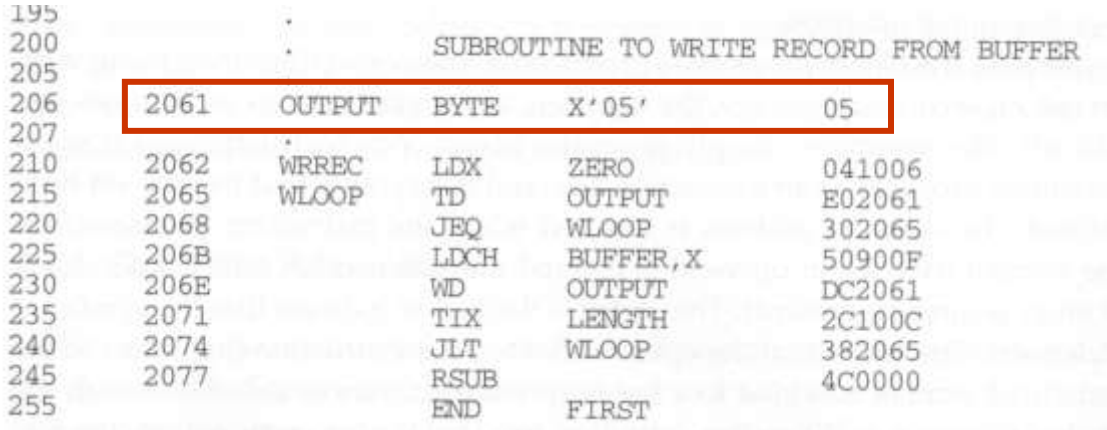
20) What is one pass assembler? Explain the functioning of one-pass assembler.

- One-pass assemblers are used when
  - it is necessary or desirable to avoid a second pass over the source program
  - the external storage for the intermediate file between two passes is slow or is inconvenient to use
- Main problem: forward references to both data and instructions
- One simple way to eliminate this problem: require that all areas be defined before they are referenced.
  - It is possible, although inconvenient, to do so for data items.
  - Forward jump to instruction items cannot be easily eliminated.

Sample Program for a One-Pass Assembler

Sample Program for a One-Pass Assembler

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		.			
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
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
196					



```
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
```

Load-and-Go Assembler

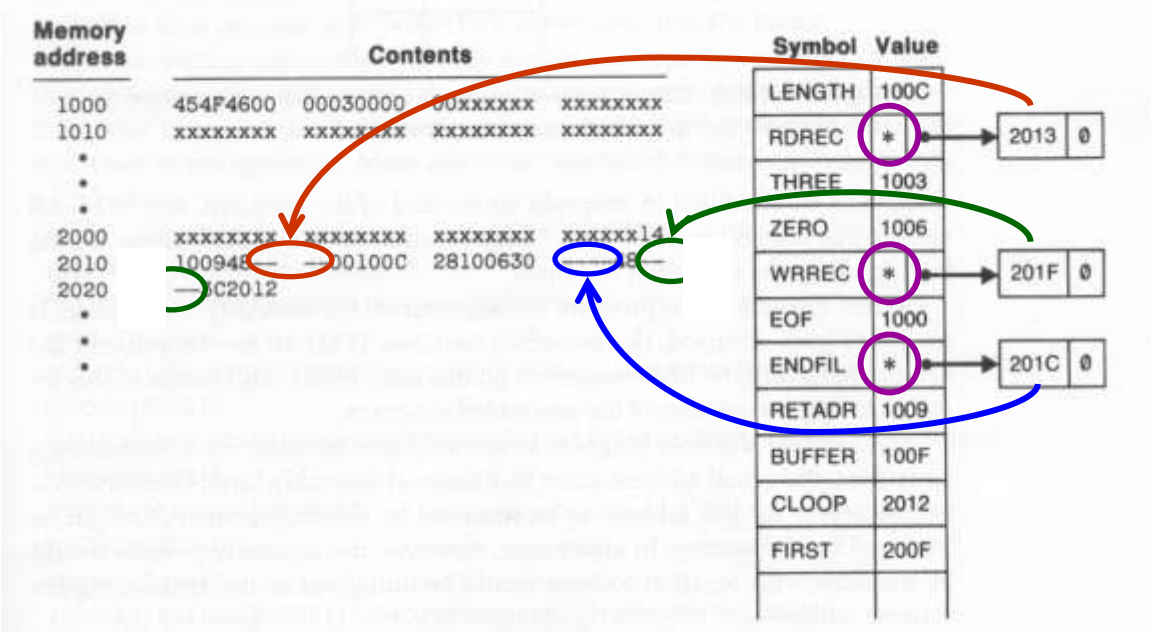
- Load-and-go assembler generates their object code in memory for immediate execution.
- No object program is written out, no loader is needed.
- It is useful in a system oriented toward program development and testing such that the efficiency of the assembly process is an important consideration.

How to Handle Forward References

- Load-and-go assembler
    - Omits the operand address if the symbol has not yet been defined
    - Enters this undefined symbol into SYMTAB and indicates that it is undefined
    - Adds the address of this operand address to a list of forward references associated with the SYMTAB entry
    - Scans the reference list and inserts the address when the definition for the symbol is encountered.
    - Reports the error if there are still SYMTAB entries indicated undefined symbols at the end of the program
    - Search SYMTAB for the symbol named in the END statement and jumps to this location to begin execution if there is no error

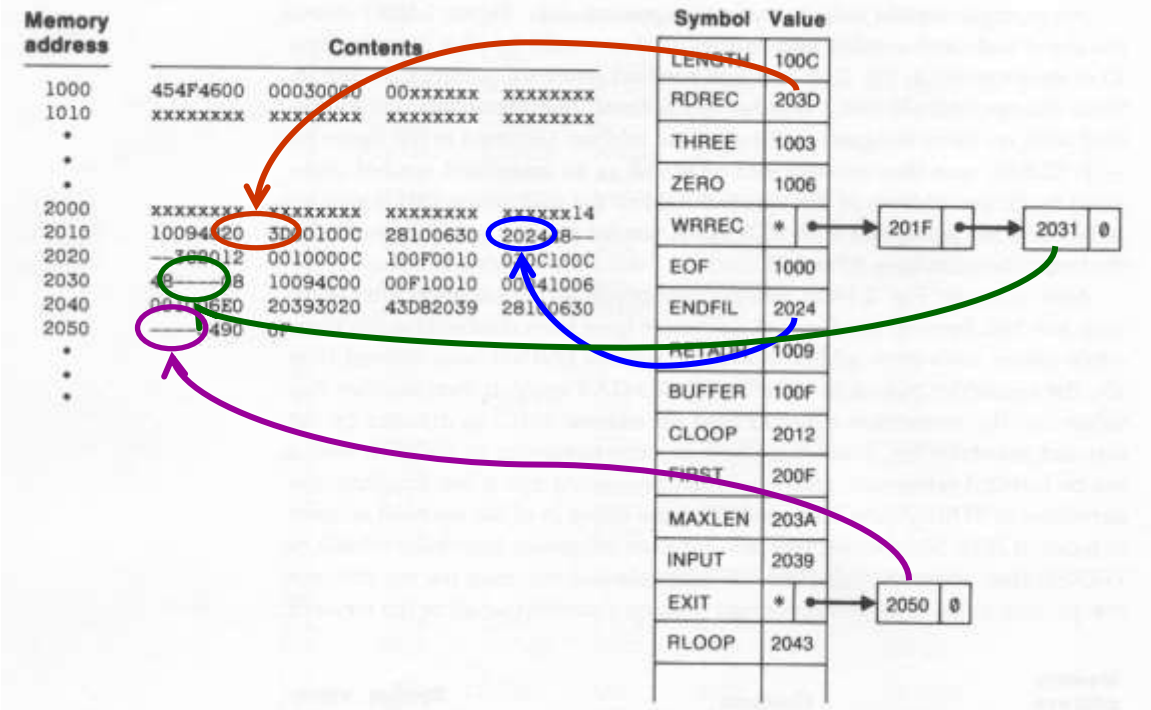
Object Code in Memory and SYMTAB

After scanning line 40

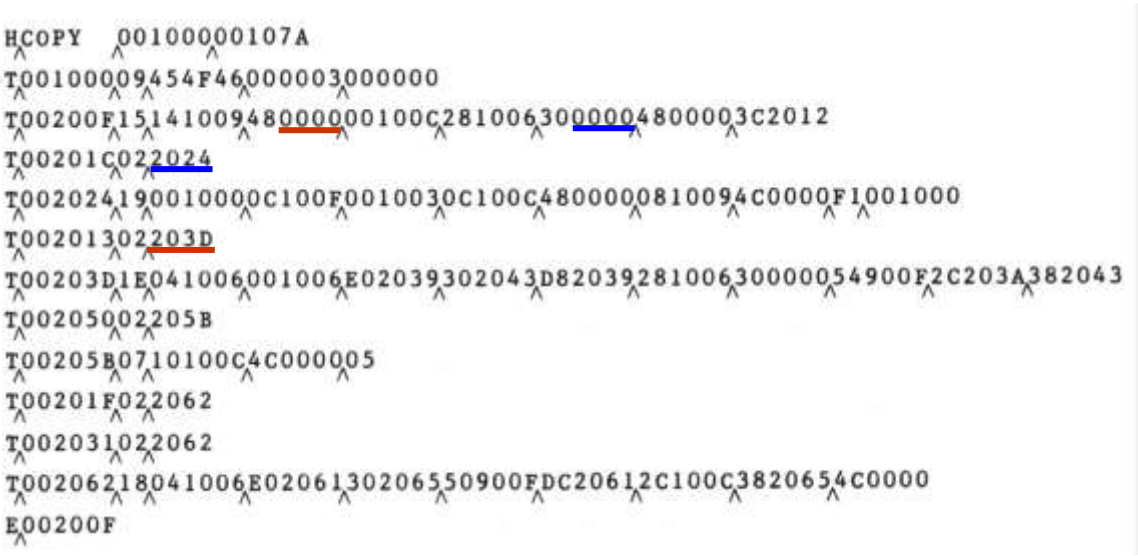


Object Code in Memory and SYMTAB

After scanning line 160



Object Program from One-Pass Assembler



21) What is a multi-pass assembler? Explain with an example,the functioning of a multi-pass assembler.

Multi-Pass Assemblers

- Prohibiting forward references in symbol definition:
  - This restriction is not a serious inconvenience.
  - Forward references tend to create difficulty for a person reading the program.
- Allowing forward references
  - To provide more flexibility
  - Solution:
    - A multi-pass assembler that can make as many passes as are needed to process the definitions of symbols.
    - Only the portions of the program that involve forward references in symbol definition are saved for multi-pass reading.
- For a two pass assembler, forward references in symbol definition are not allowed:

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

- Reason: symbol definition must be completed in pass 1.
- Motivation for using a multi-pass assembler
  - DELTA can be defined in pass 1
  - BETA can be defined in pass 2
  - ALPHA can be defined in pass 3

Implementation

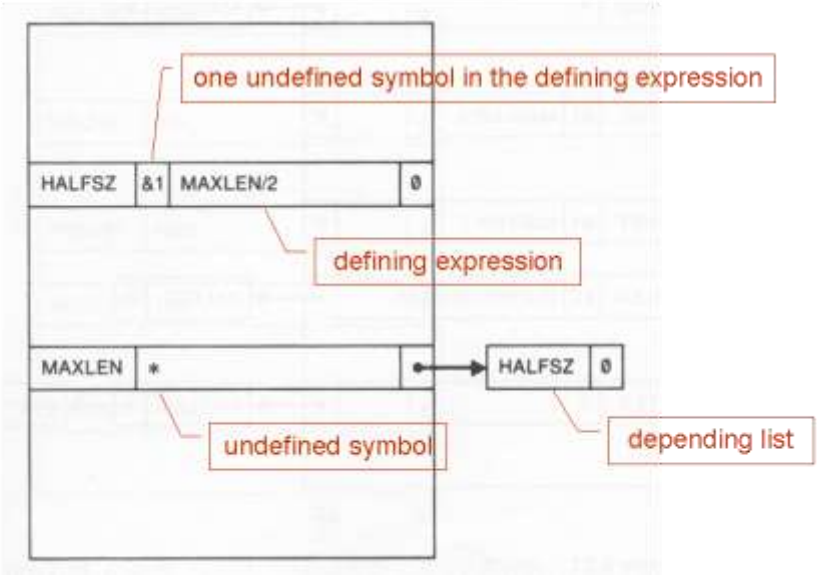
- A symbol table is used
  - to store symbol definitions that involve forward references
  - to indicate which symbols are dependant on the values of others
  - to facilitate symbol evaluation
- For a forward reference in symbol definition, we store in the SYMTAB:
  - the symbol name
  - the defining expression
  - the number of undefined symbols in the defining expression
  - the undefined symbol (marked with a flag *) associated with a list of symbols depend on this undefined symbol.
- When a symbol is defined, we can recursively evaluate the symbol expressions depending on the newly defined symbol.

Forward Reference Example

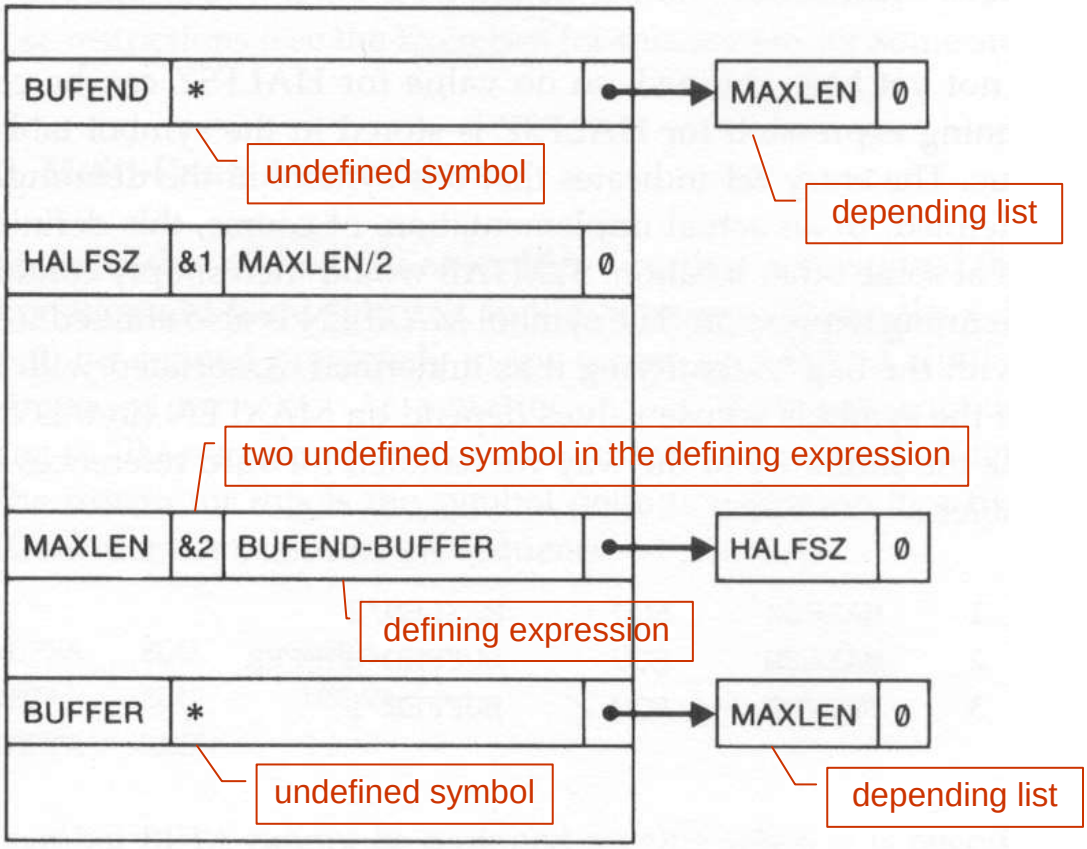
```
1    HALFSZ    EQU    MAXLEN/2
2    MAXLEN    EQU    BUFEND-BUFFER
3    PREVBT    EQU    BUFFER-1
                      .
                      .
                      .
4    BUFFER    RESB   4096
5    BUFEND    EQU    *
```

Forward Reference Example

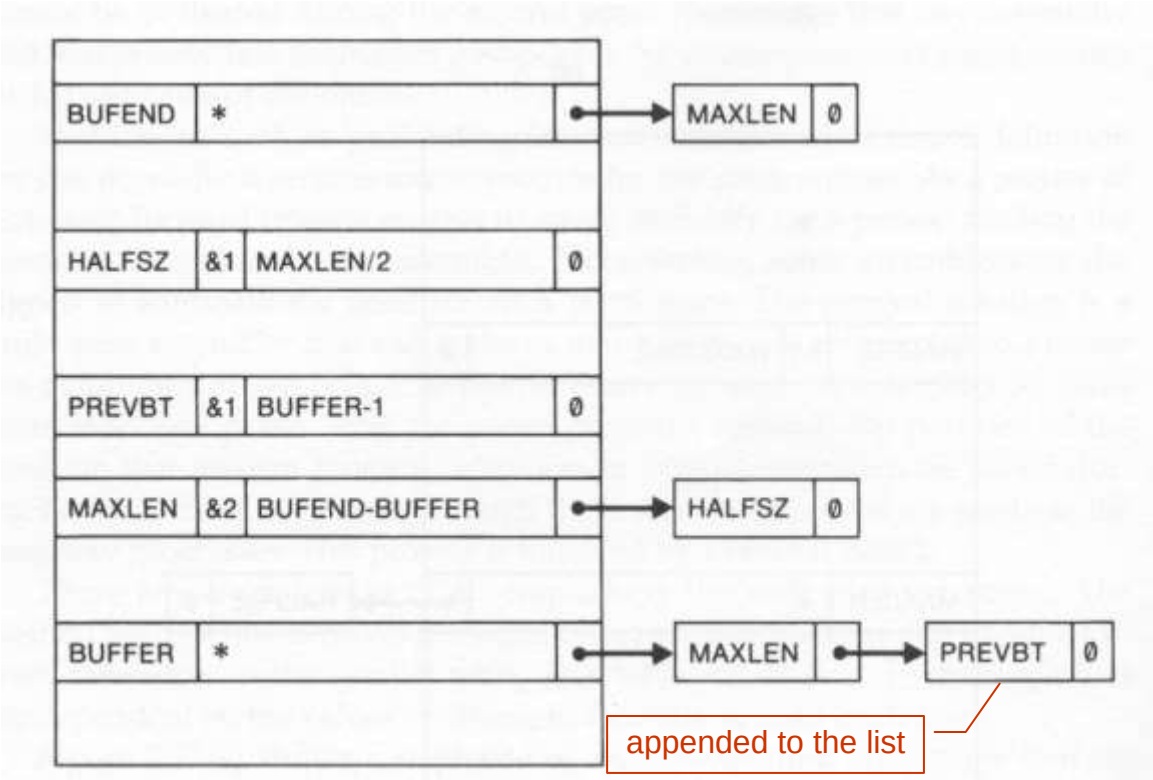
```
1       HALFSZ    EQU    MAXLEN/2
```



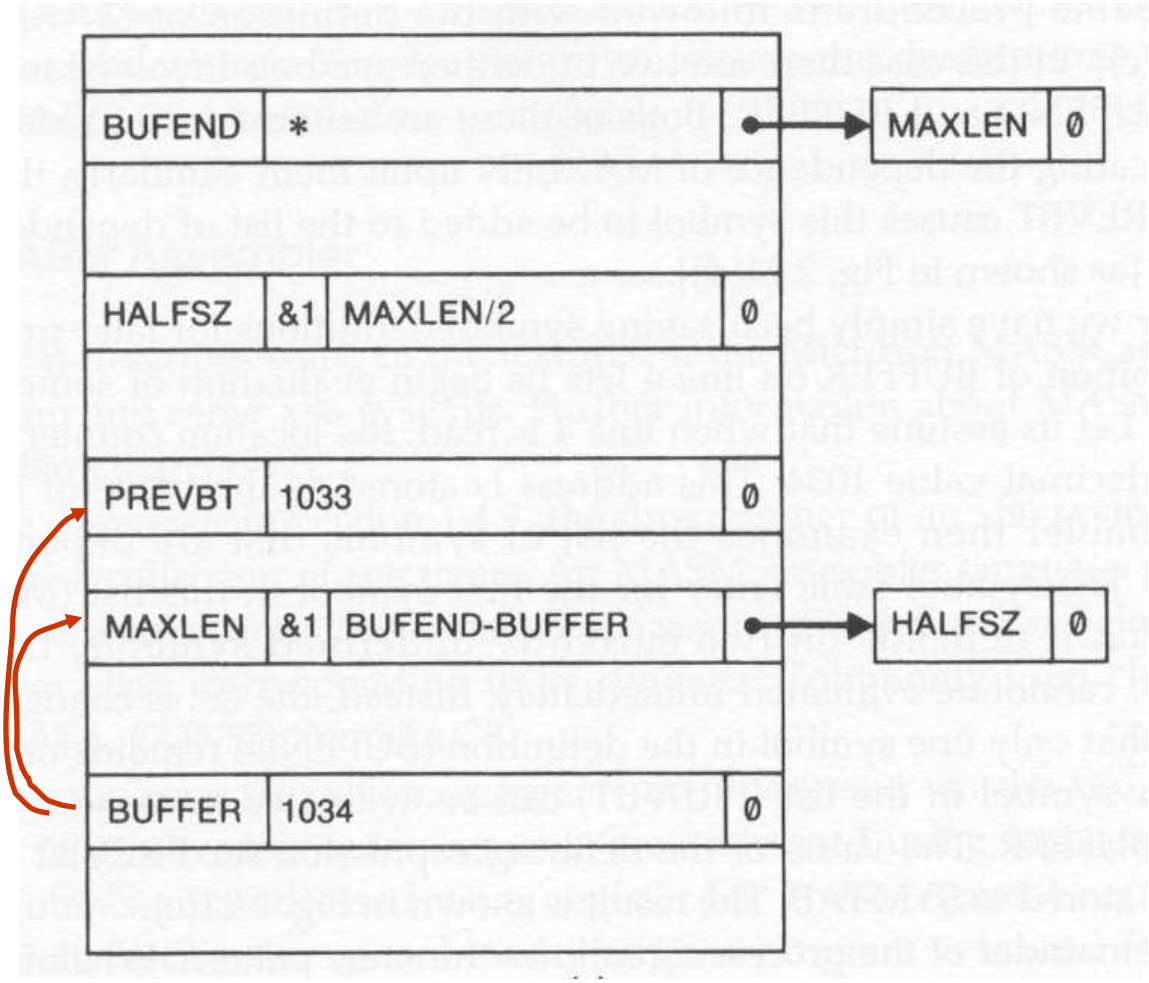

```
2       MAXLEN    EQU    BUFEND-BUFFER
```



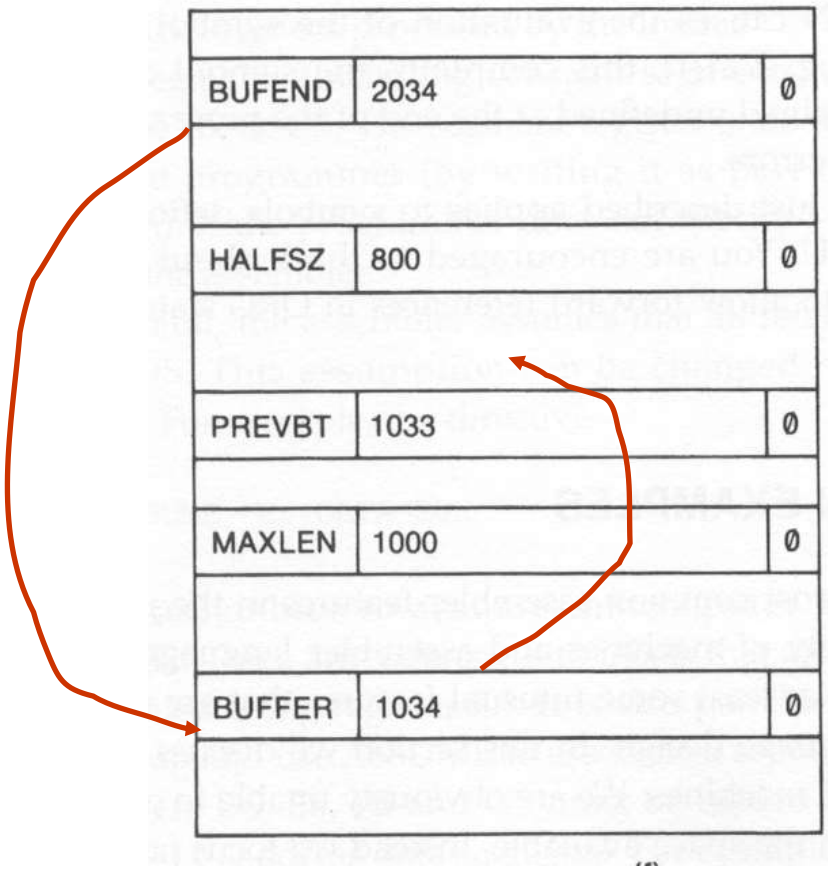
```
3       PREVBT      EQU      BUFFER-1
```



```
4       BUFFER      RESB      4096
```



```
5       BUFEND     EQU     *
```


BUFEND
2034
0
HALFSZ
800
0
PREVBT
1033
0
MAXLEN
1000
0
BUFFER
1034
0