UNIT – 3

ASSEMBLERS – 2

3.1. Machine-Independent features:

These are the features which do not depend on the architecture of the machine. These are:

- Literals
- Expressions
- Program blocks
- Control sections

Literals:

A literal is defined with a prefix = followed by a specification of the literal value. Example:

```
45      001A   ENDFIL      LDA   =C'EOF'      032010
-
-
93                         LTORG
        002D   *                 =C'EOF'      454F46
```

The example above shows a 3-byte operand whose value is a character string EOF. The object code for the instruction is also mentioned. It shows the relative displacement value of the location where this value is stored. In the example the value is at location (002D) and hence the displacement value is (010). As another example the given statement below shows a 1-byte literal with the hexadecimal value '05'.

```
215     1062   WLOOP       TD    =X'05'       E32011
```

It is important to understand the difference between a constant defined as a literal and a constant defined as an immediate operand. In case of literals the assembler generates the specified value as a constant at some other memory location In immediate mode the operand value is assembled as part of the instruction itself. Example

```
55      0020               LDA   #03          010003
```

All the literal operands used in a program are gathered together into one or more *literal pools*. This is usually placed at the end of the program. The assembly listing of a program containing literals usually includes a listing of this literal pool, which shows the assigned addresses and the generated data values. In some cases it is placed at some other location in the object program. An assembler directive LTORG is used. Whenever the LTORG is encountered, it creates a literal pool that contains all the literal operands used since the beginning of the program. The literal pool definition is done after LTORG is encountered. It is better to place the literals close to the instructions.

A literal table is created for the literals which are used in the program. The literal table contains the *literal name, operand value and length.* The literal table is usually created as a hash table on the literal name.

Implementation of Literals:

During Pass-1:

The literal encountered is searched in the literal table. If the literal already exists, no action is taken; if it is not present, the literal is added to the LITTAB and for the address value it waits till it encounters LTORG for literal definition. When Pass 1 encounters a LTORG statement or the end of the program, the assembler makes a scan of the literal table. At this time each literal currently in the table is assigned an address. As addresses are assigned, the location counter is updated to reflect the number of bytes occupied by each literal.

During Pass-2:

The assembler searches the LITTAB for each literal encountered in the instruction and replaces it with its equivalent value as if these values are generated by BYTE or WORD. If a literal represents an address in the program, the assembler must generate a modification relocation for, if it all it gets affected due to relocation. The following figure shows the difference between the SYMTAB and LITTAB

SYMTAB

Name	Value
COPY	0
FIRST	0
CLOOP	6
ENDFIL	1A
RETADR	30
LENGTH	33
BUFFER	36
BUFEND	1036
MAXLEN	1000
RDREC	1036
RLOOP	1040
EXIT	1056
INPUT	105C
WREC	105D
WLOOP	1062

LITTAB

Literal	Hex Value	Length	Address
C'EOF'	454F46	3	002D
X'05'	05	1	1076

3.2. Symbol-Defining Statements:

EQU Statement:

Most assemblers provide an assembler directive that allows the programmer to define symbols and specify their values. The directive used for this **EQU** (Equate). The general form of the statement is

```
Symbol          EQU         value
```

This statement defines the given symbol (i.e., entering in the SYMTAB) and assigning to it the value specified. The value can be a constant or an expression involving constants

and any other symbol which is already defined. One common usage is to define symbolic names that can be used to improve readability in place of numeric values. For example

```
+LDT        #4096
```

This loads the register T with immediate value 4096, this does not clearly what exactly this value indicates. If a statement is included as:

```
MAXLEN      EQU         4096 and then
            +LDT        #MAXLEN
```

Then it clearly indicates that the value of MAXLEN is some maximum length value. When the assembler encounters EQU statement, it enters the symbol MAXLEN along with its value in the symbol table. During LDT the assembler searches the SYMTAB for its entry and its equivalent value as the operand in the instruction. The object code generated is the same for both the options discussed, but is easier to understand. If the maximum length is changed from 4096 to 1024, it is difficult to change if it is mentioned as an immediate value wherever required in the instructions. We have to scan the whole program and make changes wherever 4096 is used. If we mention this value in the instruction through the symbol defined by EQU, we may not have to search the whole program but change only the value of MAXLENGTH in the EQU statement (only once).

Another common usage of EQU statement is for defining values for the general-purpose registers. The assembler can use the mnemonics for register usage like a-register A , X – index register and so on. But there are some instructions which requires numbers in place of names in the instructions. For example in the instruction RMO 0,1 instead of RMO A,X. The programmer can assign the numerical values to these registers using EQU directive.

```
A           EQU         0
X           EQU         1 and so on
```

These statements will cause the symbols A, X, L… to be entered into the symbol table with their respective values. An instruction RMO A, X would then be allowed. As another usage if in a machine that has many general purpose registers named as R1, R2,…, some may be used as base register, some may be used as accumulator. Their usage may change from one program to another. In this case we can define these requirement using EQU statements.

```
BASE        EQU         R1
INDEX       EQU         R2
```

```
COUNT    EQU    R3
```

One restriction with the usage of EQU is whatever symbol occurs in the right hand side of the EQU should be predefined. For example, the following statement is not valid:

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

As the symbol ALPHA is assigned to BETA before it is defined. The value of ALPHA is not known.

ORG Statement:

This directive can be used to indirectly assign values to the symbols. The directive is usually called ORG (for origin). Its general format is:

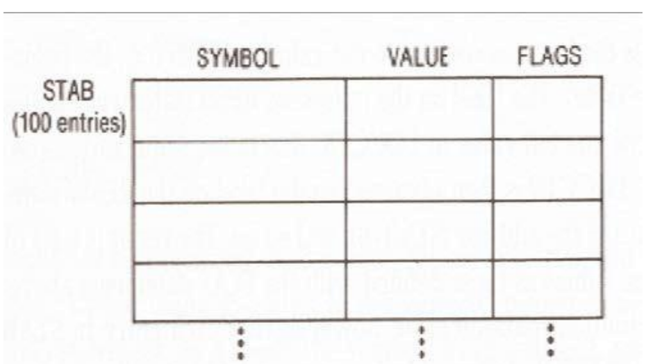
```
ORG      value
```

Where value is a constant or an expression involving constants and previously defined symbols. When this statement is encountered during assembly of a program, the assembler resets its location counter (LOCCTR) to the specified value. Since the values of symbols used as labels are taken from LOCCTR, the ORG statement will affect the values of all labels defined until the next ORG is encountered. ORG is used to control assignment storage in the object program. Sometimes altering the values may result in incorrect assembly.

ORG can be useful in label definition. Suppose we need to define a symbol table with the following structure:

SYMBOL 6 Bytes
VALUE 3 Bytes
FLAG 2 Bytes

The table looks like the one given below.



The symbol field contains a 6-byte user-defined symbol; VALUE is a one-word representation of the value assigned to the symbol; FLAG is a 2-byte field specifies symbol type and other information. The space for the ttable can be reserved by the statement:

```
STAB      RESB      1100
```

If we want to refer to the entries of the table using indexed addressing, place the offset value of the desired entry from the beginning of the table in the index register. To

refer to the fields SYMBOL, VALUE, and FLAGS individually, we need to assign the values first as shown below:

```
SYMBOL    EQU       STAB
VALUE     EQU       STAB+6
FLAGS     EQU       STAB+9
```

To retrieve the VALUE field from the table indicated by register X, we can write a statement:

```
LDA       VALUE, X
```

The same thing can also be done using ORG statement in the following way:

```
STAB      RESB      1100
          ORG       STAB
SYMBOL    RESB      6
VALUE     RESW      1
FLAG      RESB      2
          ORG       STAB+1100
```

The first statement allocates 1100 bytes of memory assigned to label STAB. In the second statement the ORG statement initializes the location counter to the value of STAB. Now the LOCCTR points to STAB. The next three lines assign appropriate memory storage to each of SYMBOL, VALUE and FLAG symbols. The last ORG statement reinitializes the LOCCTR to a new value after skipping the required number of memory for the table STAB (i.e., STAB+1100).

While using ORG, the symbol occurring in the statement should be predefined as is required in EQU statement. For example for the sequence of statements below:

```
          ORG       ALPHA
BYTE1     RESB      1
BYTE2     RESB      1
BYTE3     RESB      1
          ORG
ALPHA     RESB      1
```

The sequence could not be processed as the symbol used to assign the new location counter value is not defined. In first pass, as the assembler would not know what value to assign to ALPHA, the other symbol in the next lines also could not be defined in the symbol table. This is a kind of problem of the forward reference.

3.3 .Expressions:

Assemblers also allow use of expressions in place of operands in the instruction. Each such expression must be evaluated to generate a single operand value or address. Assemblers generally arithmetic expressions formed according to the normal rules using arithmetic operators +, - *, /. Division is usually defined to produce an integer result. Individual terms may be constants, user-defined symbols, or special terms. The only special term used is * (the current value of location counter) which indicates the value of the next unassigned memory location. Thus the statement

```
BUFFEND    EQU         *
```

Assigns a value to BUFFEND, which is the address of the next byte following the buffer area. Some values in the object program are relative to the beginning of the program and some are absolute (independent of the program location, like constants). Hence, expressions are classified as either absolute expression or relative expressions depending on the type of value they produce.

Absolute Expressions: The expression that uses only absolute terms is absolute expression. Absolute expression may contain relative term provided the relative terms occur in pairs with opposite signs for each pair. Example:

```
MAXLEN     EQU         BUFEND-BUFFER
```

In the above instruction the difference in the expression gives a value that does not depend on the location of the program and hence gives an absolute immaterial o the relocation of the program. The expression can have only absolute terms. Example:

```
MAXLEN     EQU         1000
```

Relative Expressions: All the relative terms except one can be paired as described in “absolute”. The remaining unpaired relative term must have a positive sign. Example:

```
STAB       EQU         OPTAB + (BUFEND – BUFFER)
```

Handling the type of expressions: to find the type of expression, we must keep track the type of symbols used. This can be achieved by defining the type in the symbol table against each of the symbol as shown in the table below:

Symbol	Type	Value
RETADR	R	0030
BUFFER	R	0036
BUFEND	R	1036
MAXLEN	A	1000

3.4 Program Blocks:

Program blocks allow the generated machine instructions and data to appear in the object program in a different order by Separating blocks for storing code, data, stack, and larger data block.

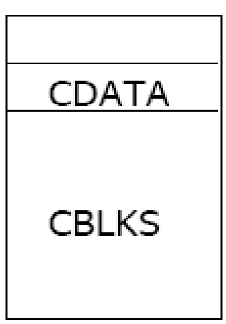
Assembler Directive USE:

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. Assemblers rearrange these segments to gather together the pieces of each block and assign address. Separate the program into blocks in a particular order. Large buffer area is moved to the end of the object program. *Program readability is better* if data areas are placed in the source program close to the statements that reference them.

In the example below three blocks are used :

Default: executable instructions
CDATA: all data areas that are less in length
CBLKS: all data areas that consists of larger blocks of memory



Example Code

```
(default) block      Block number
0000   0   COPY     START   0
0000   0   FIRST    STL     RETADR        172063
0003   0   CLOOP    JSUB    RDREC         4B2021
0006   0            LDA     LENGTH        032060
0009   0            COMP    #0            290000
000C   0            JEQ     ENDFIL        332006
000F   0            JSUB    WRREC         4B203B
0012   0            J       CLOOP         3F2FEE
0015   0   ENDFIL   LDA     =C'EOF'       032055
0018   0            STA     BUFFER        0F2056
001B   0            LDA     #3            010003
001E   0            STA     LENGTH        0F2048
0021   0            JSUB    WRREC         4B2029
0024   0            J       @RETADR       3E203F
0000   1            USE     CDATA         CDATA block
0000   1   RETADR   RESW    1
0003   1   LENGTH   RESW    1
0000   2            USE     CBLKS         CBLKS block
0000   2   BUFFER   RESB    4096
1000   2   BUFEND   EQU     *
1000       MAXLEN   EQU     BUFEND-BUFFER
```

```
                              (default) block
0027   0   RDREC    USE
0027   0            CLEAR   X             B410
0029   0            CLEAR   A             B400
002B   0            CLEAR   S             B440
002D   0            +LDT    #MAXLEN       75101000
0031   0   RLOOP    TD      INPUT         E32038
0034   0            JEQ     RLOOP         332FFA
0037   0            RD      INPUT         DB2032
003A   0            COMPR   A,S           A004
003C   0            JEQ     EXIT          332008
003F   0            STCH    BUFFER,X      57A02F
0042   0            TIXR    T             B850
0044   0            JLT     RLOOP         3B2FEA
0047   0   EXIT     STX     LENGTH        13201F
004A   0            RSUB                  4F0000
0006   1            USE     CDATA         CDATA block
0006   1   INPUT    BYTE    X'F1'         F1
```

```
                                            (default) block
004D   0            USE
004D   0   WRREC    CLEAR    X           B410
004F   0            LDT      LENGTH      772017
0052   0   WLOOP    TD       =X'05'      E3201B
0055   0            JEQ      WLOOP       332FFA
0058   0            LDCH     BUFFER,X    53A016
005B   0            WD       =X'05'      DF2012
005E   0            TIXR     T           B850
0060   0            JLT      WLOOP       3B2FEF
0063   0            RSUB                 4F0000
0007   1            USE      CDATA       CDATA block
                    LTORG
0007   1   *        =C'EOF'              454F46
000A   1   *        =X'05'               05
                    END      FIRST
```

Arranging code into program blocks:

Pass 1

- A separate location counter for each program block is maintained.
- Save and restore LOCCTR when switching between blocks.
- At the beginning of a block, LOCCTR is set to 0.
- Assign each label an address relative to the start of the block.
- Store the block name or number in the SYMTAB along with the assigned relative address of the label
- Indicate the block length as the latest value of LOCCTR for each block at the end of Pass1
- Assign to each block a starting address in the object program by concatenating the program blocks in a particular order

Pass 2

- Calculate the address for each symbol relative to the start of the object program by adding
  - ➢ The location of the symbol relative to the start of its block
  - ➢ The starting address of this block

3.5 Control Sections:

A *control section* is a part of the program that maintains its identity after assembly; each control section can be loaded and relocated independently of the others. Different control sections are most often used for subroutines or other logical subdivisions. The programmer can assemble, load, and manipulate each of these control sections separately.

Because of this, there should be some means for linking control sections together. For example, instructions in one control section may refer to the data or instructions of other control sections. Since control sections are independently loaded and relocated, the assembler is unable to process these references in the usual way. Such references between different control sections are called *external references.*

The assembler generates the information about each of the external references that will allow the loader to perform the required linking. When a program is written using multiple control sections, the beginning of each of the control section is indicated by an assembler directive

- assembler directive: **CSECT**

The syntax

secname CSECT

- separate location counter for each control section

Control sections differ from program blocks in that they are handled separately by the assembler. Symbols that are defined in one control section may not be used directly another control section; they must be identified as external reference for the loader to handle. The external references are indicated by two assembler directives:

EXTDEF (external Definition):

It is the statement in a control section, names symbols that are defined in this section but may be used by other control sections. Control section names do not need to be named in the EXTREF as they are automatically considered as external symbols.

EXTREF (external Reference):

It names symbols that are used in this section but are defined in some other control section.

The order in which these symbols are listed is not significant. The assembler must include proper information about the external references in the object program that will cause the loader to insert the proper value where they are required.

Implicitly defined as an external symbol
first control section

```
COPY      START     0                    COPY FILE FROM INPUT TO OUTPUT
          EXTDEF    BUFFER,BUFEND,LENGTH
          EXTREF    RDREC,WRREC
FIRST     STL       RETADR               SAVE RETURN ADDRESS
CLOOP     +JSUB     RDREC                READ INPUT RECORD
          LDA       LENGTH               TEST FOR EOF (LENGTH=0)
          COMP      #0
          JEQ       ENDFIL               EXIT IF EOF FOUND
          +JSUB     WRREC                WRITE OUTPUT RECORD
          J         CLOOP                LOOP
ENDFIL    LDA       =C'EOF'              INSERT END OF FILE MARKER
          STA       BUFFER
          LDA       #3                   SET LENGTH = 3
          STA       LENGTH
          +JSUB     WRREC                WRITE EOF
          J         @RETADR              RETURN TO CALLER
RETADR    RESW      1
LENGTH    RESW      1                    LENGTH OF RECORD
          LTORG
BUFFER    RESB      4096                 4096-BYTE BUFFER AREA
BUFEND    EQU       *
MAXLEN    EQU       BUFFEND-BUFFER
```

Implicitly defined as an external symbol
second control section

```
RDREC     CSECT
.
.         SUBROUTINE TO READ RECORD INTO BUFFER
.
          EXTREF    BUFFER,LENGTH,BUFFEND
          CLEAR     X                    CLEAR LOOP COUNTER
          CLEAR     A                    CLEAR A TO ZERO
          CLEAR     S                    CLEAR S TO ZERO
          LDT       MAXLEN
RLOOP     TD        INPUT                TEST INPUT DEVICE
          JEQ       RLOOP                LOOP UNTIL READY
          RD        INPUT                READ CHARACTER INTO REGISTER A
          COMPR     A,S                  TEST FOR END OF RECORD (X'00')
          JEQ       EXIT                 EXIT LOOP IF EOR
          +STCH     BUFFER,X             STORE CHARACTER IN BUFFER
          TIXR      T                    LOOP UNLESS MAX LENGTH HAS
          JLT       RLOOP                   BEEN REACHED
EXIT      +STX      LENGTH               SAVE RECORD LENGTH
          RSUB                           RETURN TO CALLER
INPUT     BYTE      X'F1'                CODE FOR INPUT DEVICE
MAXLEN    WORD      BUFFEND-BUFFER
```

Implicitly defined as an external symbol
third control section

```
WRREC      CSECT
.
.          SUBROUTINE TO WRITE RECORD FROM BUFFER

            EXTREF    LENGTH,BUFFER
            CLEAR     X                 CLEAR LOOP COUNTER
           +LDT       LENGTH
WLOOP       TD       =X'05'             TEST OUTPUT DEVICE
            JEQ       WLOOP             LOOP UNTIL READY
           +LDCH      BUFFER,X          GET CHARACTER FROM BUFFER
            WD       =X'05'             WRITE CHARACTER
            TIXR      T                 LOOP UNTIL ALL CHARACTERS HAVE
            JLT       WLOOP                BEEN WRITTEN
            RSUB                        RETURN TO CALLER
            END       FIRST
```

Handling External Reference

Case 1

15 0003 CLOOP +JSUB RDREC 4B100000

- The operand RDREC is an external reference.
  - The assembler has no idea where RDREC is
  - inserts an address of zero
  - can only use extended format to provide enough room (that is, relative addressing for external reference is invalid)
- The assembler generates information for each external reference that will allow the loader to perform the required linking.

Case 2

190 0028 MAXLEN WORD BUFEND-BUFFER 000000

- There are two external references in the expression, BUFEND and BUFFER.
- The assembler inserts a value of zero
- passes information to the loader
- Add to this data area the address of BUFEND
- Subtract from this data area the address of BUFFER

Case 3

On line 107, BUFEND and BUFFER are defined in the same control section and the expression can be calculated immediately.

```
107     1000  MAXLEN     EQU           BUFEND-BUFFER
```

Object Code for the example program:

```
0000   COPY      START     0
                 EXTDEF    BUFFER,BUFFEND,LENGTH
                 EXTREF    RDREC,WRREC
0000   FIRST     STL       RETADR                     172027
0003   CLOOP    +JSUB      RDREC                      4B100000    Case 1
0007             LDA       LENGTH                     032023
000A             COMP     #0                          290000
000D             JEQ       ENDFIL                     332007
0010            +JSUB      WRREC                      4B100000
0014             J         CLOOP                      3F2FEC
0017   ENDFIL    LDA      =C'EOF'                     032016
001A             STA       BUFFER                     0F2016
001D             LDA      #3                          010003
0020             STA       LENGTH                     0F200A
0023            +JSUB      WRREC                      4B100000
0027             J        @RETADR                     3E2000
002A   RETADR    RESW      1
002D   LENGTH    RESW      1
                 LTORG
0030   *        =C'EOF'                               454F46
0033   BUFFER    RESB      4096
1033   BUFEND    EQU       *
1000   MAXLEN    EQU       BUFEND-BUFFER

0000   RDREC     CSECT
       .
       .         SUBROUTINE TO READ RECORD INTO BUFFER
       .
                 EXTREF    BUFFER,LENGTH,BUFEND
0000             CLEAR     X                          B410
0002             CLEAR     A                          B400
0004             CLEAR     S                          B440
0006             LDT       MAXLEN                     77201F
0009   RLOOP     TD        INPUT                      E3201B
000C             JEQ       RLOOP                      332FFA
000F             RD        INPUT                      DB2015
0012             COMPR     A,S                        A004
0014             JEQ       EXIT                       332009
0017            +STCH      BUFFER,X                   57900000
001B             TIXR      T                          B850
001D             JLT       RLOOP                      3B2FE9
0020   EXIT     +STX       LENGTH                     13100000
0024             RSUB                                 4F0000
0027   INPUT     BYTE      X'F1'                      F1
0028   MAXLEN    WORD      BUFFEND-BUFFER             000000      Case 2
```

```
0000    WRREC   CSECT
        .
        .       SUBROUTINE TO WRITE RECORD FROM BUFFER
        .
                EXTREF    LENGTH,BUFFER
0000            CLEAR     X                          B410
0002           +LDT       LENGTH                     77100000
0006    WLOOP   TD       =X'05'                      E32012
0009            JEQ       WLOOP                      332FFA
000C           +LDCH      BUFFER,X                   53900000
0010            WD       =X'05'                      DF2008
0013            TIXR      T                          B850
0015            JLT       WLOOP                      3B2FEE
0018            RSUB                                 4F0000
                END       FIRST
001B    *      =X'05'                                05
```

The assembler must also include information in the object program that will cause the loader to insert the proper value where they are required. The assembler maintains two new record in the object code and a changed version of modification record.

Define record (EXTDEF)

- Col. 1 D
- Col. 2-7 Name of external symbol defined in this control section
- Col. 8-13 Relative address within this control section (hexadecimal)
- Col.14-73 Repeat information in Col. 2-13 for other external symbols

Refer record (EXTREF)

- Col. 1 R
- Col. 2-7 Name of external symbol referred to in this control section
- Col. 8-73 Name of other external reference symbols

Modification record

- Col. 1 M
- Col. 2-7 Starting address of the field to be modified (hexadecimal)
- Col. 8-9 Length of the field to be modified, in half-bytes (hexadecimal)
- Col.11-16 External symbol whose value is to be added to or subtracted from the indicated field

A define record gives information about the external symbols that are defined in this control section, i.e., symbols named by EXTDEF.

A refer record lists the symbols that are used as external references by the control section, i.e., symbols named by EXTREF.

The new items in the modification record specify the modification to be performed:

adding or subtracting the value of some external symbol. The symbol used for modification my be defined either in this control section or in another section.

The object program is shown below. There is a separate object program for each of the control sections. In the *Define Record* and *refer record* the symbols named in EXTDEF and EXTREF are included.

In the case of *Define*, the record also indicates the relative address of each external symbol within the control section.

For EXTREF symbols, no address information is available. These symbols are simply named in the *Refer record.*

```
COPY
H^COPY  ^000000^001033
D^BUFFER^000033^BUFEND^001033^LENGTH^00002D
R^RDREC ^WRREC
T^000000^1D^172027^4B100000^032023^290000^332007^4B100003^3F2FEC^032016^0F2016
T^00001D^0D^010003^0F200A^4B100000^3E2000
T^000030^03^454F46
M^000004^05^+RDREC
M^000011^05^+WRREC
M^000024^05^+WRREC
E^000000

RDREC
H^RDREC ^000000^00002B
R^BUFFER^LENGTH^BUFEND
T^000000^1D^B410^B400^B440^77201F^E3201B^332FFA^DB2015^A004^332009^57900000^B850
T^00001D^0E^3B2FE9^13100000^4F0000^F1^000000
M^000018^05^+BUFFER
M^000021^05^+LENGTH
M^000028^06^+BUFEND
M^000028^06^-BUFFER      BUFEND - BUFFER
E

WRREC
H^WRREC ^000000^00001C
R^LENGTH^BUFFER
T^000000^1C^B410^77100000^E32012^332FFA^53900000^DF2008^B850^3B2FEE^4F000005
M^000003^05^+LENGTH
M^00000D^05^+BUFFER
E
```

Handling Expressions in Multiple Control Sections:

The existence of multiple control sections that can be relocated independently of one another makes the handling of expressions complicated. It is required that in an expression that all the relative terms be paired (for absolute expression), or that all except one be paired (for relative expressions).

When it comes in a program having multiple control sections then we have an extended restriction that:

- Both terms in each pair of an expression must be within the same control section
  - o If two terms represent relative locations within the same control section , their difference is an absolute value (regardless of where the control section is located.
    - **Legal:** BUFEND-BUFFER (both are in the same control section)
  - o If the terms are located in different control sections, their difference has a value that is unpredictable.
    - **Illegal:** RDREC-COPY (both are of different control section) it is the difference in the load addresses of the two control sections. This value depends on the way run-time storage is allocated; it is unlikely to be of any use.

- **How to enforce this restriction**
  - o When an expression involves external references, the assembler cannot determine whether or not the expression is legal.
  - o The assembler evaluates all of the terms it can, combines these to form an initial expression value, and generates Modification records.
  - o The loader checks the expression for errors and finishes the evaluation.

3.6. ASSEMBLER DESIGN

Here we are discussing

- o The structure and logic of one-pass assembler. These assemblers are used when it is necessary or desirable to avoid a second pass over the source program.
- o Notion of a multi-pass assembler, an extension of two-pass assembler that allows an assembler to handle forward references during symbol definition.

One-Pass Assembler

The main problem in designing the assembler using single pass was to resolve forward references. We can avoid to some extent the forward references by:

- Eliminating forward reference to data items, by defining all the storage reservation statements at the beginning of the program rather at the end.
- Unfortunately, forward reference to labels on the instructions cannot be avoided. (forward jumping)
- To provide some provision for handling forward references by prohibiting forward references to data items.

There are two types of one-pass assemblers:

- One that produces object code directly in memory for immediate execution (Load-and-go assemblers).
- The other type produces the usual kind of object code for later execution.

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 with frequent program development and testing
    - The efficiency of the assembly process is an important consideration.
- Programs are re-assembled nearly every time they are run; efficiency of the assembly process is an important consideration.

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		.			

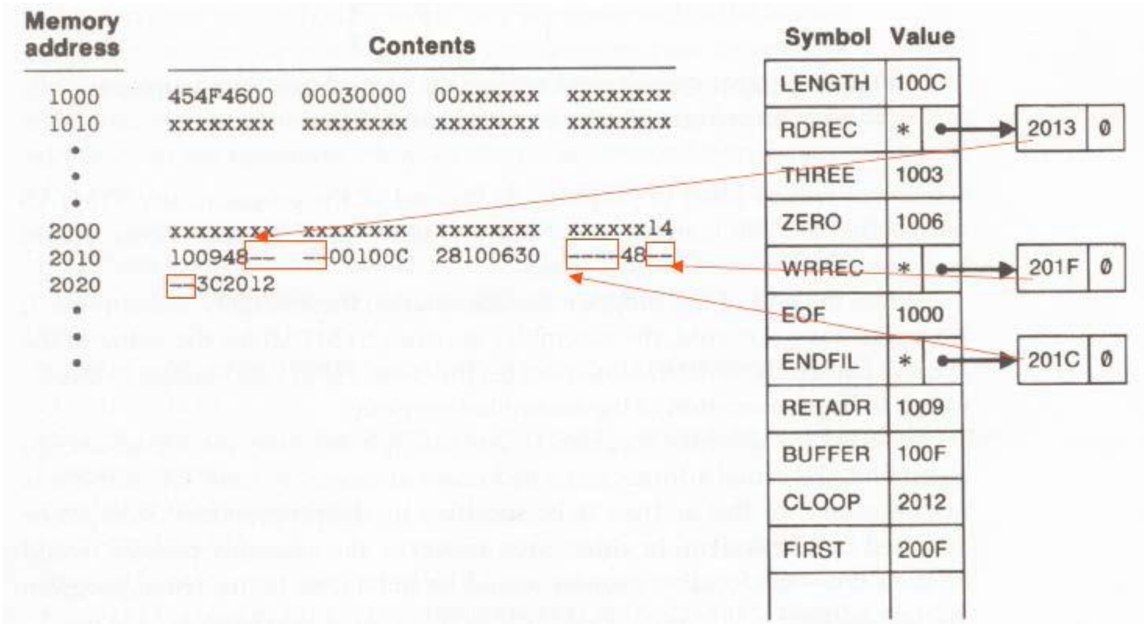
Forward Reference in One-Pass Assemblers: In load-and-Go assemblers when a forward reference is encountered :

- 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
- When the definition for the symbol is encountered, scans the reference list and inserts the address.
- At the end of the program, reports the error if there are still SYMTAB entries indicated undefined symbols.
- For Load-and-Go assembler
    - Search SYMTAB for the symbol named in the END statement and jumps to this location to begin execution if there is no error

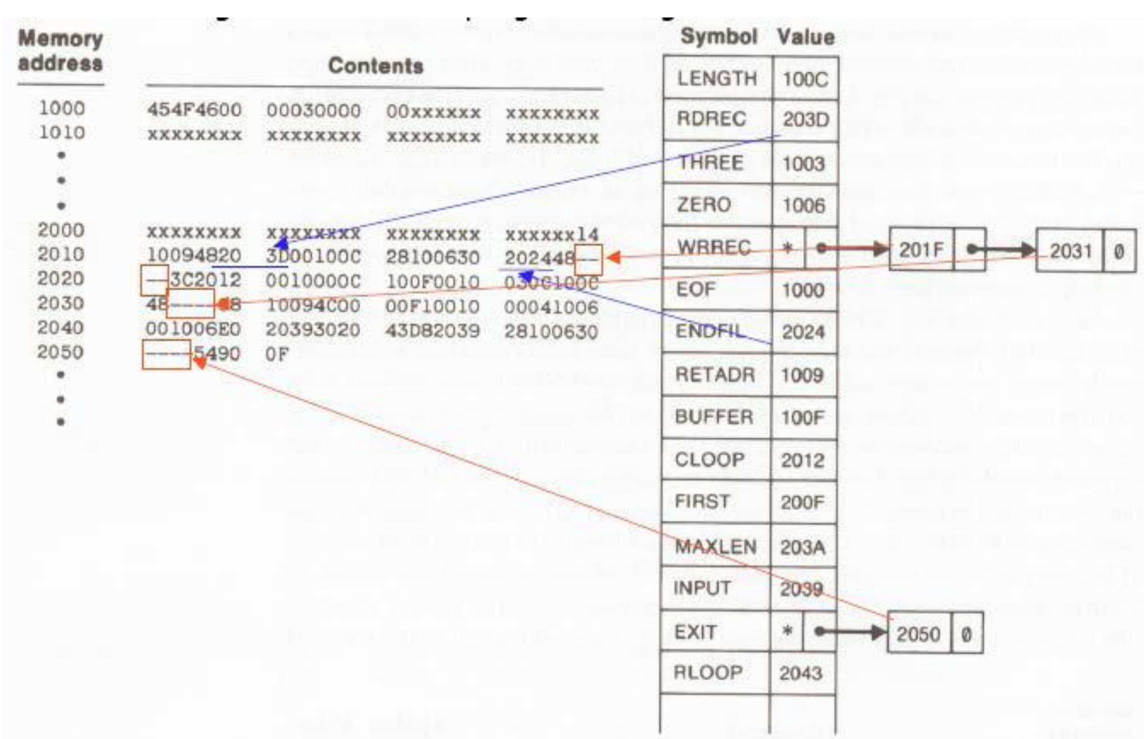
After Scanning line 40 of the program:

40 2021 J` CLOOP 302012

The status is that upto this point the symbol RREC is referred once at location 2013, ENDFIL at 201F and WRREC at location 201C. None of these symbols are defined. The figure shows that how the pending definitions along with their addresses are included in the symbol table.



The status after scanning line 160, which has encountered the definition of RDREC and ENDFIL is as given below:



If One-Pass needs to generate object code:

- If the operand contains an undefined symbol, use 0 as the address and write the Text record to the object program.
- Forward references are entered into lists as in the load-and-go assembler.
- When the definition of a symbol is encountered, the assembler generates another Text record with the correct operand address of each entry in the reference list.
- When loaded, the incorrect address 0 will be updated by the latter Text record containing the symbol definition.

Object Code Generated by One-Pass Assembler:

```
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
```

Multi_Pass Assembler:

- For a two pass assembler, forward references in symbol definition are not allowed:

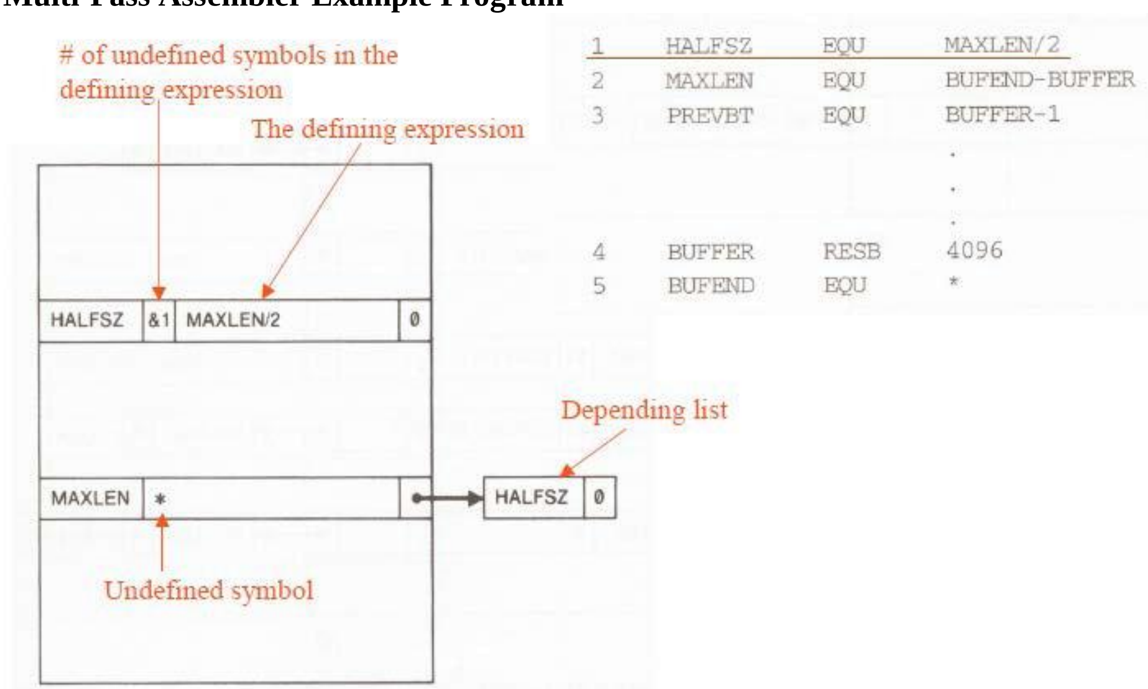
  ```
  ALPHA    EQU   BETA
  BETA     EQU   DELTA
  DELTA    RESW  1
  ```
  - Symbol definition must be completed in pass 1.
- Prohibiting forward references in symbol definition is not a serious inconvenience.
  - Forward references tend to create difficulty for a person reading the program.

Implementation Issues for Modified Two-Pass Assembler:

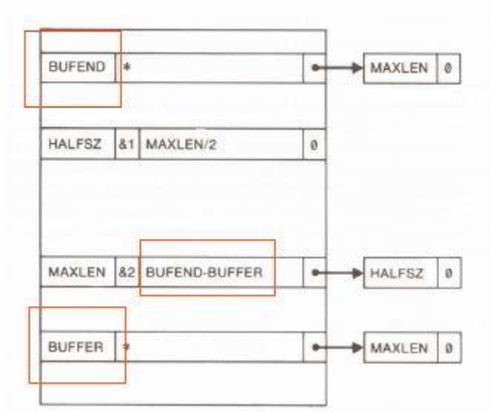
Implementation Isuues when forward referencing is encountered in *Symbol Defining statements* :

- 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.

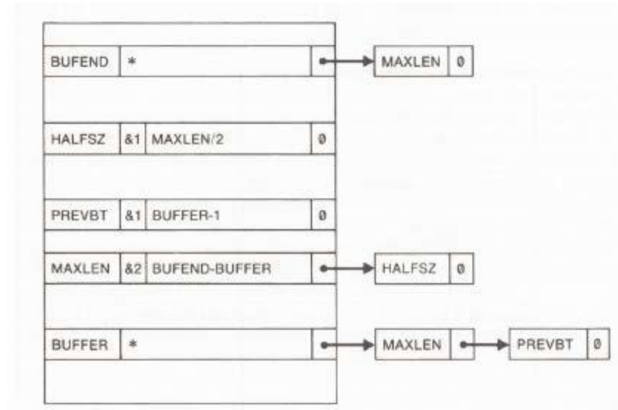
Multi-Pass Assembler Example Program



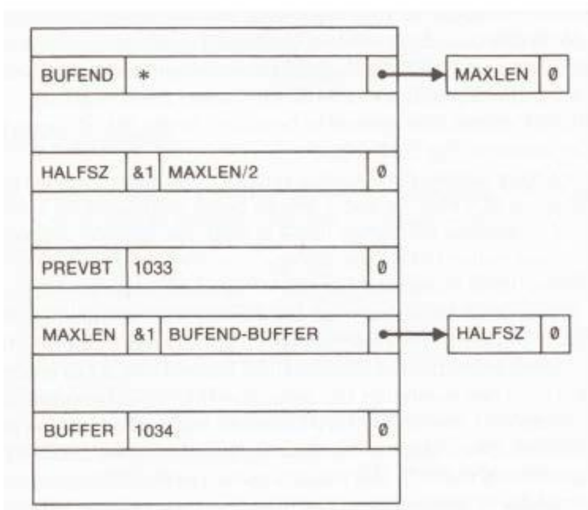
Multi-Pass Assembler (Figure 2.21 of Beck): Example for forward reference in Symbol Defining Statements:



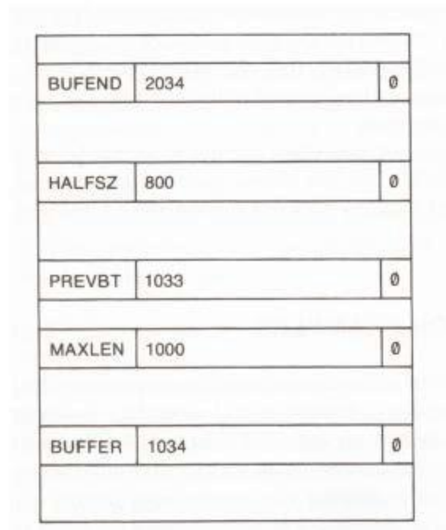
2 MAXLEN EQU BUFEND-BUFFER



3 PREVBT EQU BUFFER-1



4 BUFFER RESB 4096



5 BUFEND EQU *