MODULE 2

SYLLABUS

Assemblers: Basic assembler functions, machine dependent and machine independent assembler features, one-pass assemblers, multi pass assemblers, MASM assembler,

SPARC assembler

MACHINEINDEPENDENTASSEMBLERFEATURES

The featureswhichareNOTcloselydependenttomachinearchitecturearecalled machine independent assembler features. The machine independent assembler features includes:

1. Literals
2. SymbolDefiningStatements
3. Expressions
4. ProgramBlocks
5. ControlSections andProgramLinking

LITERALS

- It is convenient for the programmer to be able to write the value of a constant operand as part of the instruction that uses it.
- Thisavoidshavingtodefinetheconstantelsewhereintheprogramandmakealabelforit.
- SuchanoperandiscalledaLiteralbecausethevalueis literallyintheinstruction.
- Aliteralisdefinedwithaprefix'='followedbyaspecificationoftheliteralvalue.
- Considerthefollowingexample:

```
                .
                .
                LDA         FIVE
                .
                .
FIVE            WORD5
```

Using theconceptofliteralwecanrewritethe abovecode as:

```
                .
                .
                LDA         =X'05'
```

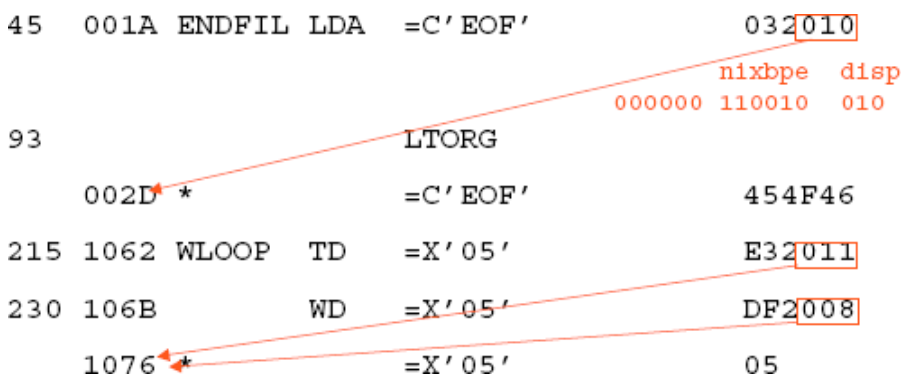
Differencebetweenliteraloperandsandimmediate operands

- Forliteralsprefix is=,and forimmediateaddressingprefixis #.
- Inimmediateaddressing,theoperand value isassembledaspartofthe machine instruction, ie there is no memory reference.

  Lineno LocationCounter

  55 0020 LDA #03 010**003**

  Intheaboveexamplethe last 12bitsofthe machinecodecorresponds to003whichisequal to the immediate value.
- With a literal, the assembler generates the specified value as a constant at some other memory location. The address of this generated constant is used as the target address (TA) for the machine instruction (using PC-relative or base-relative addressing with memory reference.)



LiteralPool

- All theliteral operandsusedin aprogram aregatheredtogetherintooneormoreliteral pools. This is usually placed at the end of the program.
- Insomecases,itisdesirabletoplaceliteralsintoapoolatsomeotherlocationinthe object program. To allow this an assembler directive **LTORG** is used.
- Whenthe assembler encounters a LTORG statement, it generates a *literal pool* containing all literal operands used since previous LTORG or the beginning of the program
- LiteralsplacedinapoolbyLTORGwillnotberepeatedinapoolattheendofthe program.
- Reason for using LTORG is tokeepthe literaloperand close to the instruction(otherwise PC-relative addressing may not be allowed)

LiteralTable (LITTAB)

- Aliteraltable(**LITTAB**) iscreatedforstoringtheliteralswhichareusedinthe program.
- The literaltablecontainstheliteralname, operandvalueandlength.
- Theliteraltableisusuallycreatedasahashtableontheliteralname.

Duplicateliterals

- Thesame literalused morethanonce intheprogram, thenit canbeconsider asaduplicate literal.
- Insuchcases,onlyone copyofthe specifiedvalueneedstobestored
- Torecognizetheduplicateliterals,twomethodsarethere

1. Comparethecharacterstringsdefiningthem
   Easiertoimplemente.g.=X'05'.Butnotpossible tohandlethe literalslike
   =C'EOF'and=X'454F46'.
   Herebothliteralsare sameintheformoftheirdatavalue.
2. Comparethegenerateddatavalue
   Possible to handle the literals like**=C'EOF'** and**=X'454F46'.** Herebothliteralsare same in the formof their generated data value. So comparison based on generated data value is needed to identify duplicate literals or not. But this is difficult to implement compared to the first method.

ImplementationofLiterals

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 or END statement for literal definition.

When Pass 1 encounters a LTORG statement or the end of the program, theassembler makes a scan ofthe literaltable. 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.

DuringPass-2:

The assembler searches the LITTAB for each literal encountered in the instructionand replaces it with itsequivalent value as ifthese valuesare generated byBYTEorWORD. 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

SYMBOLDEFININGSTATEMENTSANDEXPRESSIONS

EQU Statement:

- Mostassemblersprovideanassemblerdirectivethatallowstheprogrammertodefine symbols and specify their values.
- ThedirectiveusedforthisEQU (Equate).
- The generalformofthe statementis

  Symbol EQUvalue
- This statement defines the given symbol (i.e., entering in the SYMTAB) and assigns the value specified to that symbol.
- The value can be a constant or an expression involving constants and any other symbolwhich is already defined.
- Onecommonusage isto definesymbolic namesthat canbeusedto improvereadability in place of numeric values. For example

  LDA#100

  Thisloads theregisterA withimmediatevalue100, this does notclearlymention what exactly this value indicates. If a statement is included as:

  MAXLENEQU 100

  and then LDA#MAXLEN then it clearlyindicatesthatthe value ofMAXLEN issome maximum length value and it is to be loaded in A register.

- When the assembler encounters EQU statement, it enters the symbol MAXLEN alongwith its value in the symbol table. During LDA 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 100 to 500, 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 100 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.

ORGStatement:

- This directive can be used to indirectly assign values to the symbols. The directive is usually called ORG (means origin).
- Itsgeneralformatis:

  ORGvalue

  wherevalueisaconstantoranexpressioninvolvingconstantsandpreviouslydefined symbols.
- Whenthis statement isencountered during assemblyofa 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 ORGstatement will affect the values of all labels defined until the next ORG is encountered.
- Eg:ORGAlPHA

  Whenthis statement is encountered during assemblyofa program, the assembler resets its location counter (LOCCTR) to the value of ALPHA.

EXPRESSIONS

- Theassemblersallow **theuseofexpressionsasoperand**
- Theassembler evaluatestheexpressionsandproducesasingleoperandaddressorvalue.
- Assemblers generally allow arithmetic expressions as operands formed according to the normalrules using arithmetic operators +, - *, /.(Division is usuallydefined to produce an integer result.)
- Individualterms maybeconstants,user-definedsymbols,orspecialterms.
- The onlyspecialtermused is * (the current value of locationcounter) which indicates the value of the next unassigned memory location.

Thus the statement

BUFFENDEQU*

Assigns the value of LOCCTR 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.
- **AbsoluteExpressions:**
  - 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:

    MAXLENEQUBUFEND-BUFFER

    Intheabove instructionthedifference intheexpressionBUFEND-BUFFERgivesa value that does not depend on the location of the program and hence gives an absolute value
- **RelativeExpressions:**
  - Theexpressionthatusesthevaluesrelativetotheprogramarecalledrelative expression.
  - Absolute expression may contain relative term provided the relative terms occur in pairs with opposite signs for each pair.
  - Example:

    MAXLENEQUALPHA+BUFEND-BUFFER

    In the above instruction the difference in the expression BUFEND-BUFFERgives a value that does not depend on the location of the program but it is added to the value of ALPHA which is program relative. Hence this expression is relative.

PROGRAMBLOCKS

- Program blocks allow the generated machine instructions and data to appear in theobject program in a different order by **separating blocks for storing code, data, stack, and larger data block.**
- ToimplementtheprogramblocktheAssemblerDirective usedis**USE**

- Syntaxis

  USE[blockname]

- Atthebeginning,statementsareassumed to bepartoftheunnamed (ordefault)block.
- Whenever a USE CDATAstatement is encountered, statements upto next USE belongs to the program block named CDATA.
- IfnoUSEstatementsareincluded, theentireprogrambelongstothissingleblock.
- 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.
- Considerthefollowingexample:

```
COPY        START0
            LDA         LENGTH
            ……
            ……
            USE         CDATA
MAX         RESW        1
LENGTH      RESW        1
            USE         CBLOCKS
BUFFER      RESB        00
            ……
//Subroutinetoreadrecord intobuffer
            USE
RDREC       CLEAR       XLDA
                        INPUT
            ……..
            ……
            USE         CDATA
INPUT       BYTE        X'F1'
            ……
//Subroutinetowriterecordfrombuffer
            USE
WRREC       STA         MAX
            ……
            USE         CDATA
MIN         RESW        1
BUFEND      RESW        1
```

- Intheexamplegiveabovethreeprogramblocksareused :

  DEFAULT: executable instructions.

  CDATA:alldataareasthat arelessin length.

  CBLOCKS:alldata areasthatconsistsoflargerblocksof memory.

DEFAULT
CDATA
CBLOCKS

Arrangingcodeintoprogramblocks:

DuringPass1assemblerperformsthefollowingoperations:

- Aseparatelocationcounterforeachprogramblockismaintained.
  - Atthebeginningofablock,LOCCTRissetto 0.
  - SaveandrestoreLOCCTRwhenswitchingbetweenblocks.
- Assigneachlabelanaddress relative tothestartoftheblock.
- StoretheblocknameornumberintheSYMTABalongwiththeassignedrelative address of the label
- Indicate theblocklength asthelatestvalueof LOCCTRforeach blockat theendof Pass1
- Assigntoeachblockastartingaddressintheobjectprogrambyconcatenatingthe program blocks in a particular order
- Attheend ofpass1ablocktableis generated.

BlockTable

BlockName	Block Number	Starting Address	Ending Address	Length of Block
Default	**0**	**0000**	**0065**	**0066**
CDATA	**1**	**0066**	**0070**	**000B**
CBLKS	**2**	**0071**	**1070**	**1000**

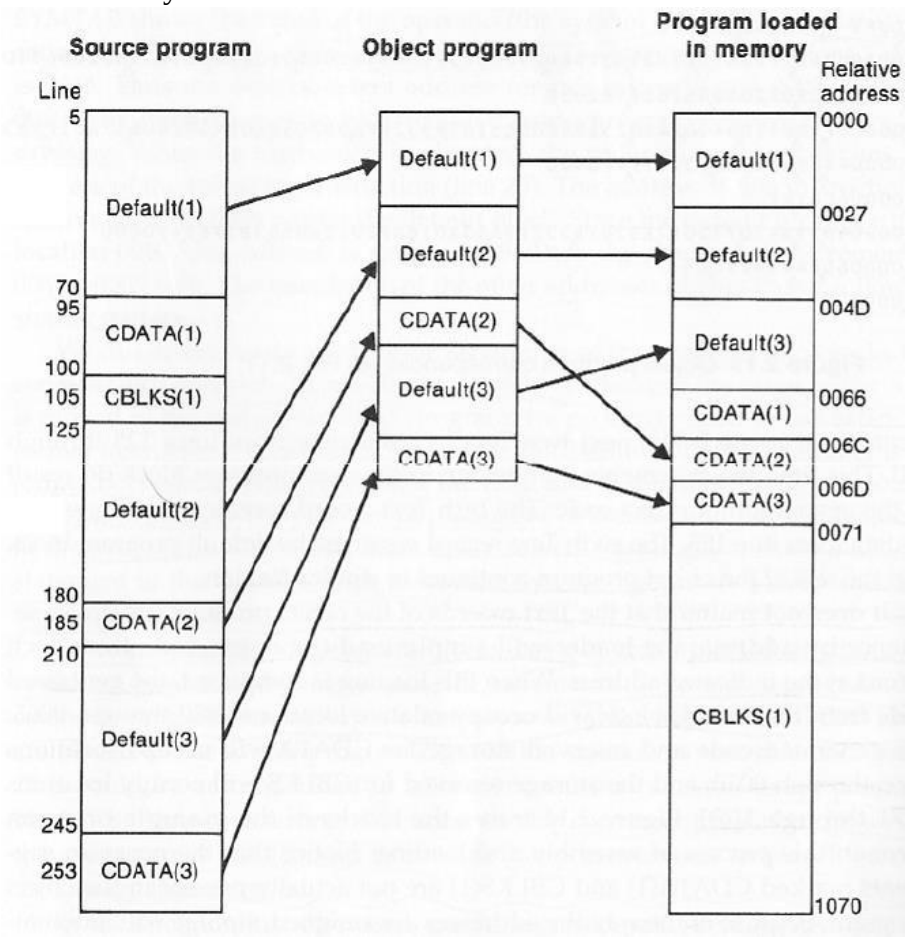
DuringPass2assemblerperformsthefollowingoperations:

- Calculate theaddressforeachsymbol relativetothestart of theobjectprogram by adding
  - Thelocationofthe symbolrelativetothe startofitsblock
  - Thestarting addressofthisblock

ProgramBlocksLoadedinMemory

Separationofprogramintoblocksresultsinthemovementofthelargebuffer (CBLKS)totheendoftheobjectprogram.Asaresultextendedformat,baseregister

addressingetcareno longerneeded.Modificationrecordsarealso not needed.Thisimproves program readability.



CONTROLSECTIONS

- 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 instructionsof 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 willallowtheloadertoperformtherequiredlinking.Whenaprogramiswritten

usingmultiplecontrolsections,thebeginningofeachofthecontrolsectionis indicated by an assembler directive: **CSECT**

- Thesyntax

  secnameCSECT

- TheassemblermaintainseparateLOCCTRbeginningat0foreachcontrolsections.
- Controlsections differ fromprogramblocks inthattheyare handled separatelybythe assembler.

HandlingofExternalReferences

Instructions in one controlsection may need to refer to instructions or data located in another section. This is called as **external references**. The external references are indicated by two assembler directives: **EXTDEF and EXTREF**

EXTDEF(ExternalDefinition)

- ItdefinesthesymbolsthataredefinedinthiscontrolsectionandmaybeusedBy other sections
- Syntax-EXTDEFname [,name]
- Ex: EXTDEF BUFFER, BUFEND, LENGTH which means the symbols BUFFER, BUFFEND and LENGTH are defined in this control section and may be used bysome other control sections.

EXTREF(ExternalReference)

- Itnames symbols thatare usedin this section butare definedin some other control section.
- Syntax- EXTREFname [,name]
- Ex: EXTREF A,B which means the symbols A and B are used in this control section but are defined in some other control section.

The assembler must include information in the object program that will cause the loader to handle external references properly.For this three types of records are used in object program: **Define, Refer and Modification Record.**

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

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

Modification Record (revised)	
Col. 1	M
Col. 2-7	Starting location of the target address to be modified, relative to the beginning of the program (not relative to the first text record)
Col. 8-9	Length of this record in half-byte
Col. 10	Modification flag (+ or -)
Col. 11-16	External symbol whose value is to be added to or subtracted from the indicated field

The formatofmodificationrecordwhichwestudiedinModule2 isrevisedtosupport the handling of external references.

Considerthefollowingcode segments:

```
COPY        START0
            EXTDEFBUFFER,BUFFEND,LENGTH
            EXTREFA,B
            LDA         ALPHA
            ……
            ……
            ……
BUFFER      WORD        3
BUFFEND     EQU         *
LENGTH      EQU         BUFFEND-BUFFER

            RDREC       CSECT
            EXTREF      BUFFER,BUFFEND,LENGTH
            ……..
            ……..
            ……
            LDA         BUFFER
            ……
            ……..
            ……

            END
```

Theobjectprogramgeneratedfortheabovecodesegmentis:

H^ COPY^ 000000^001033
D^BUFFER^000033^BUFEND^001033^LENGTH^00002D
R^A ^B
T^……
T^……
……..
……
M^000004^05^+RDREC
……
E^000000

ASSEMBLERDESIGNOPTIONS

Inthissection,twoalternativestothestandardtwo-passassemblerlogicisdiscussed. Theyare:

SinglePassAssembler

Multipass Assembler

SINGLEPASSASSEMBLER

These assemblers are used when it is necessary or desirable to avoid a second pass overthe sourceprogram. The mainproblemindesigning the assembler using single passwas to resolve forward references.

One-pass assemblers could produce object codes either in memory or to external storage. One-pass assemblers usually need to modify object code already generated, so whether object code is stored in memoryor external storage imposes different considerations on assembler design. Based onthis one-pass assemblers can be classified into two types:

1. One thatproduces objectcode directly in memory forimmediate execution (**Load-and-go assemblers**).
2. Onepassassemblergeneratingobjectcodeforlaterexecution.

1. Load-and-GoAssembler

Load-and-go assembler generates their object code in memory for immediate execution. Sinceno object programiswrittenout,no loader is needed. Itisusefulinasystem with frequent program development and testing. Since the object program is produced in memory, the handling of forward references becomes less difficult.

WorkingofOne passassembler(LoadandGoAssembler)

Inload-and-Goassemblerswhenaforwardreferenceisencountered:

- Omits the operand address if the symbol has not yet been defined(placess 000 at the operand addressesposition)
- EntersthisundefinedsymbolintoSYMTABandindicatesthatitisundefined
- Adds thelocation atwhich the operand is referenced to a listof forward referencesassociated with the SYMTAB entry
- Whenthe definition forthe symbolisencountered, scansthe reference list and inserts the address.
- AttheendoftheprogramreportstheerroriftherearestillSYMTABentriesindicated undefined symbols(* indicates undefined).
- When the END statement is encountered, search SYMTAB for the symbol named in the END statement and jumps to this location to begin execution if there is no error.

In short, whenever any undefined symbol is encountered it will insert into SYMTAB as a new entry and indicate that it is undefined and also adds the location at which the operand is referenced as a linked list associated with that SYMTAB entry. When the definition forthe symbolis encountered, scans the reference list and inserts the address in proper location.

AlgorithmforSinglePassAssembler(LoadandGoAssembler)

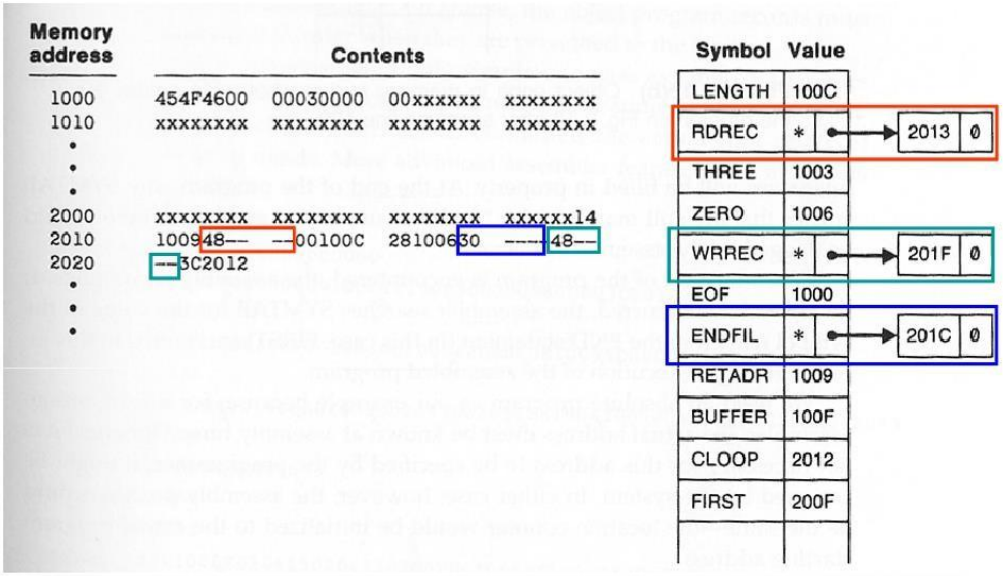
```
begin
    readfirstinput line
    ifOPCODE='START'then
    {           save#[OPERAND]asstartingaddress
                initialize LOCCTRas starting address
    }//endofifOPCODE='START' else
                initialize LOCCTR to 0
    writeHeaderrecordtoobjectprogram read
    next input line
    whileOPCODE≠'END'
    {   ifthisisnotacommentline
        {       ifthereisa symbol intheLABEL field
                {       searchSYMTABfor LABEL
```

```
        if found
        {       ifsymbol valueasnull
                {       setsymbolvalueasLOCCTR
                        search the attached forward reference list(if exist) and the address
                        ofthesymbol isinserted intoanyinstructionspreviouslygenerated
                        deletetheforwardreferencelistattachedtothatsymbol
                }
        }
        else
                insert(LABEL,LOCCTR)intoSYMTAB
}//endofifthereisa symbol intheLABELfield search
OPTAB for OPCODE
if found
        searchSYMTABforOPERANDADDRESS if
        found
        {       ifsymbol valuenot equaltonull
                        storesymbolvalueasoperandaddress
                else
                        insertanodewith address LOCCTRattheendofthe
                        forward reference list of that symbol
        }
        else
        {       insert(symbolname,null)
                insertanodewith addressLOCCTRattheendoftheforward reference
                list of that symbol
        }
        add 3 to LOCCTR
elseifOPCODE ='WORD'
        add3toLOCCTR
else if OPCODE ='RESW'
        add3#[OPERAND]toLOCCTR
elseifOPCODE ='RESB'
        add#[OPERAND]toLOCCTR
elseifOPCODE ='BYTE'
```

```
        {       findlengthofconstantinbytes
                add length to LOCCTR
                convertconstanttoobjectcode
        }
        ifobject codewillnotfitintocurrenttextrecord
        {       writeTextrecordtoobjectprogram
                initialize new text record
        }
        addobjectcodetoTextrecord read
        next input line
    }
  }//end of while OPCODE ≠ 'END'
  writelastTextrecordtoobjectprogram
  write End record to object program
end
```

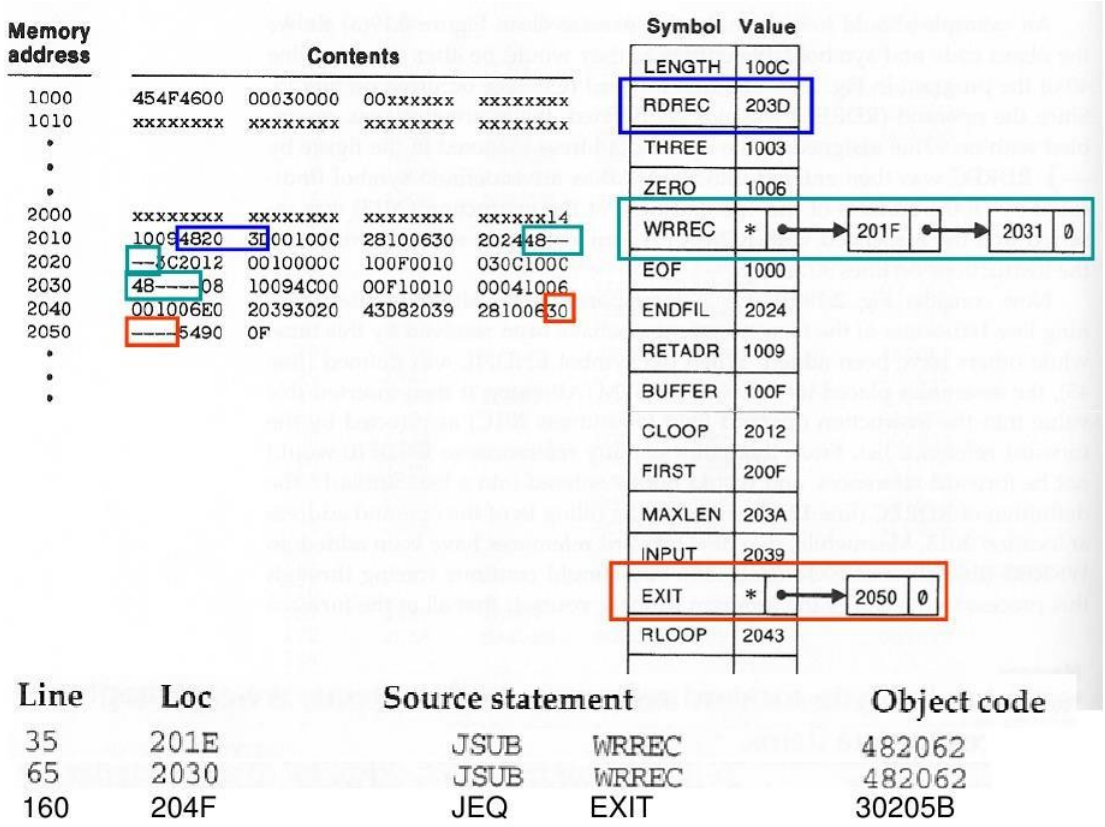
Example:

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



Line	Loc	Source statement			Object code
15	2012	CLOOP	JSUB	RDREC	48203D
30	201B		JEQ	ENDFIL	302024
35	201E		JSUB	WRREC	482062

When the definition for the symbols RDREC and ENDFILL are encountered, the reference list associated with the symbols is scanned and the address is inserted at proper location. It is gioven in following figure:



2. Onepassassemblergeneratingobjectcodeforlaterexecution.

Inthis type ofone pass assembler, the generatedobject program is storedinexternal storage (e.g.,files on disks). So random updates to operands target addresses(as in load-and-go load-and- assemblers do) are not permitted.

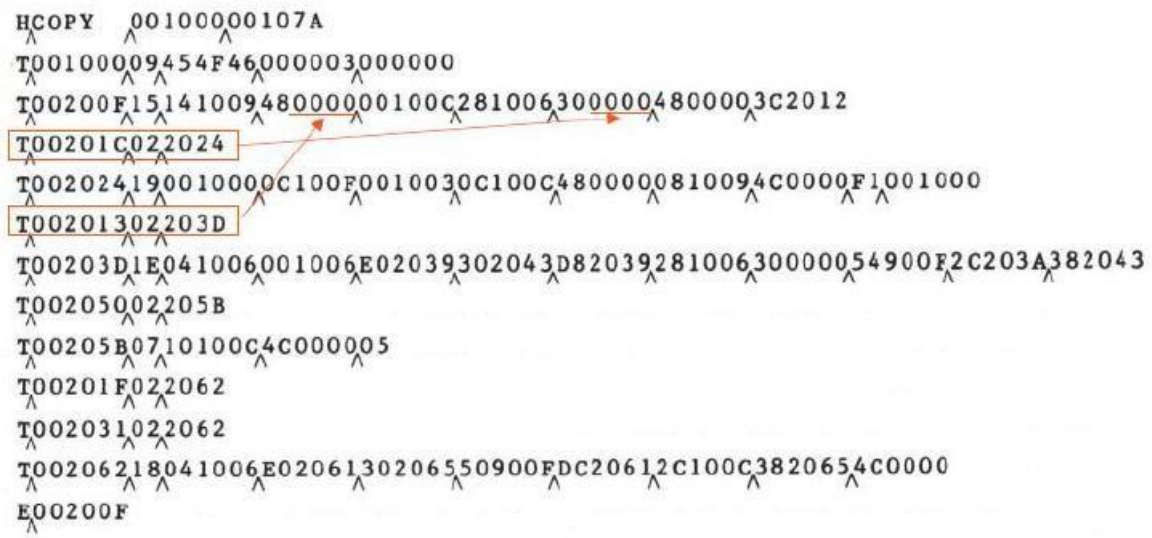
For anysymbol involved in forward references, once the target address ofthe symbol is identified, additional text records must be generated to overwrite those previously omitted target addresses. Records must be loaded in the same order as they appear in the object program. Actually, the handling of forward references are jointly done by the assembler and the linking loader.

One pass assembler which generates object code unlike load and go assembler operates in the following fashion:

- Iftheoperandcontainsanundefinedsymbol, use0astheaddressandwritetheText record to the object program.
- Forwardreferencesareenteredintolistsas intheload-and-goassembler.

- Whenthedefinitionofasymbolisencountered,theassemblergeneratesanotherText record with the correct operand address of each entry in the reference list.
- Whenloaded,the incorrect address0willbeupdatedbythe latterText record containing the symbol definition.

Example:



MULTIPASSASSEMBLER

- Foratwopassassembler,forwardreferencesinsymboldefinitionarenot allowed:

  ALPHA EQU BETA

  BETA EQU DELTA

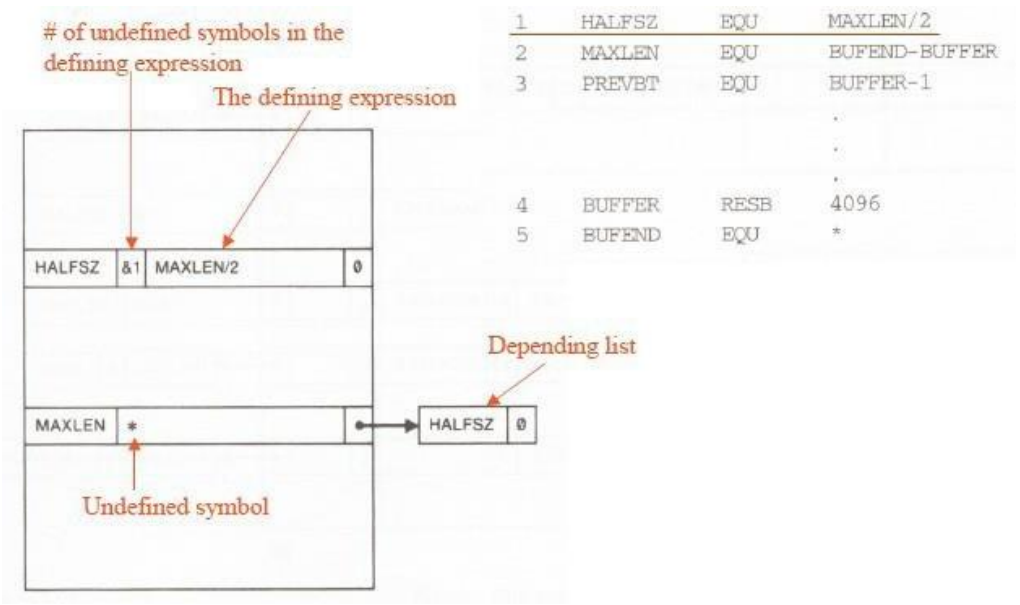
  DELTA RESW1

- Heretheproblem is, thesymbolBETAcannot beassigneda valuewhenit isencountered during Pass 1 because DELTA has not yet been defined. Hence ALPHA cannot be evaluated during Pass 2. So that the symbol definition must be completed in pass 1.
- The general solution for this type of forward references is to use a multi-pass assembler that can make as many passes as are needed to process the definitions of symbols.
- It is not necessary for such an assembler to make more than 2 passes over the entire program.
- The portions of the program that involve forward references in symbol definition are saved during Pass 1.Additional passes through these stored definitions are made as the assembly progresses. This process is followed by a normal Pass 2.

ImplementationofMultipassAssembler

- Foraforwardreferenceinsymboldefinition,westoreintheSYMTAB:
    - Thesymbol name
    - Thedefiningexpression
    - Thenumberofundefinedsymbols inthedefiningexpression
- The undefined symbol (marked as *) associated with a list of symbols depend on this undefined symbol.
- When a symbol is defined, we can recursively evaluate the symbol expressionsdepending on the newly defined symbol.
- The portions of the program that involve forward references in symbol definition are saved during Pass 1.Additional passes through these stored definitions are made as the assembly progresses. This process is followed by a normal Pass 2.

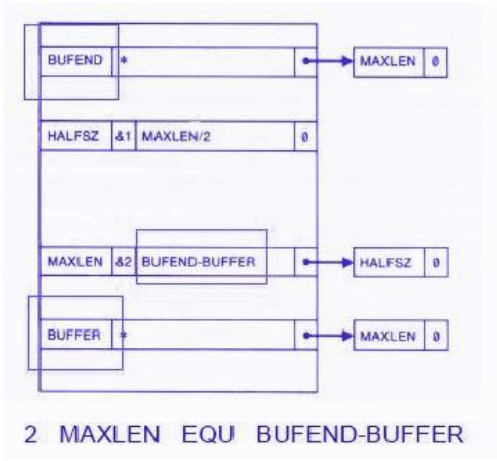
Example:



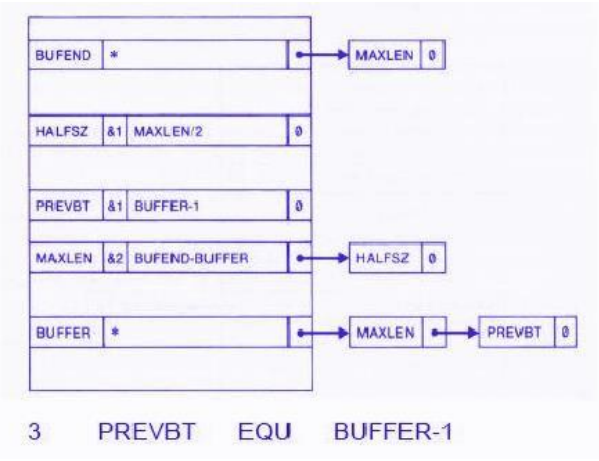

```
1    HALFSZ    EQU    MAXLEN/2
2    MAXLEN    EQU    BUFEND-BUFFER
3    PREVBT    EQU    BUFFER-1
                      .
                      .
                      .
4    BUFFER    RESB   4096
5    BUFEND    EQU    *
```

- Considerthesymboltableentries fromPass1processingofthestatement.
  HALFS2 EQU MAXLEN/2
- Since MAXLEN has not yet been defined, no value for HALFS2 can be computed. The defining expression for HALFS2 is stored in the symbol table in place of its value.
- Theentry&1indicatesthat1symbolinthedefiningexpressionundefined.
- SYMTABsimplycontainapointertothedefiningexpression.
- The symbolMAXLEN is also entered in the symboltable, withthe flag * identifying it as undefined. Associated with this entry is a list of the symbols whose values depend on MAXLEN.

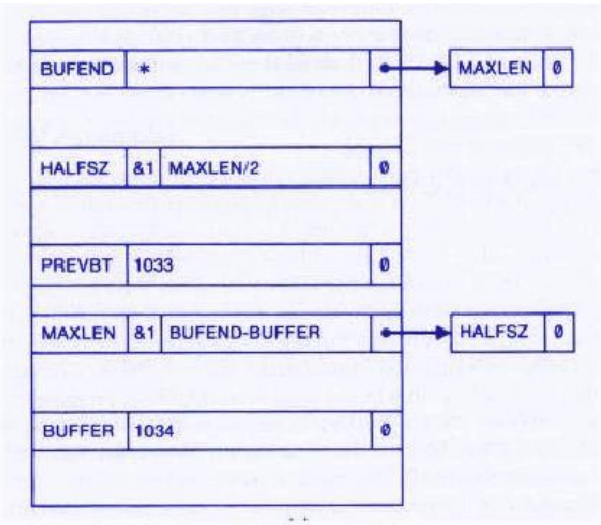
Ifpossiblestudytheportiongivenbelow



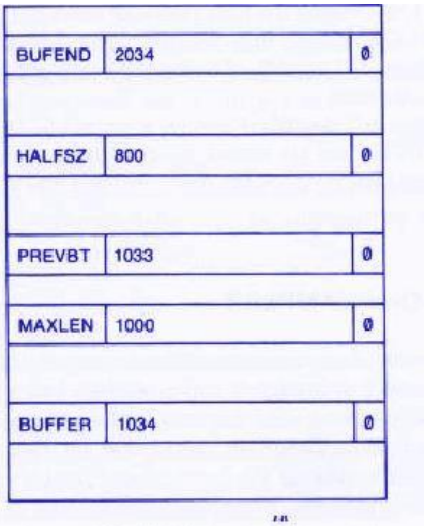
2 MAXLEN EQU BUFEND-BUFFER



3 PREVBT EQU BUFFER-1



4 BUFFER RESB 4096



5 BUFEND EQU *