CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

145

Internal Assesment Test - II

Sub:	SYSTEM SOFTWARE							Code:	10CS52
Date:	02/ 11 / 2016	Duration:	90 mins	Max Marks:	50	Sem:	V	Branch:	CSE(B &C)

Answer Any FIVE FULL Questions

		Marks	OBE CO	OBE RBT
1(a)	Discuss different design options of an assembler with suitable examples.	[10]	CO2	L2
2(a)	Write a Yacc program to function as a calculator which performs addition, subtraction, multiplication, division and unary operations.	[04]	CO6	L1
(b)	Explain the following machine Independent features of an assembler. a)Symbol Defining Statements b)Control Sections	[06]	CO2	L4
3(a)	Generate the complete object program for the following SIC/XE program .	[10]	CO2	L3

```
COPY          START    1000
CLOOP        +JSUB     RDREC
              LDA      LENGTH
              COMP     ZERO
              JEQ      EXIT
              J        CLOOP
EXIT          STA      BUFFER
              LDA      THREE
              STA      TOTAL_LENGTH
              RSUB
BUFFER        RESW     100
EOF           BYTE     C 'EOF'
ZERO          WORD     0
THREE         WORD     3
LENGTH        RESW     1
TOTAL_LENGTH  RESW     1
RDREC         LDX      ZERO
```

MNEMONICS:

JSUB=A0, LDA=80, LDX=60, STA=50, COMP=90, RSUB=4C, JEQ=B0, J=B8

		Marks	CO	RBT
4(a)	Explain the ambiguity in arithmetic expressions. What is the ambiguity in parsing 2+3*4 ? Explain the solution for it.	[10]	CO6	L4
5(a)	What are Program Blocks? With a suitable example, explain how program blocks are handled by an assembler.	[10]	CO2	L1
6(a)	With a neat diagram, explain the working of a typical Editor structure.	[10]	CO4	L4
7(a)	Distinguish between literals and Immediate operands. How does the assembler handle the Literal operands .	[06]	CO1	L2
(b)	List the basic tasks of a text editor.	[04]	CO4	L1

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Explain the basic concepts, conditions and mechanisms to create system software.	3	3	0	0	1	0	0	0	0	1	0	0
CO2:	Describe functions of single and multi-pass assemblers through assembly language concepts.	1	1	0	0	0	0	0	0	0	0	0	0
CO3:	Explain the pre-processing, linking and loading of programs.	3	1	0	0	0	0	0	0	0	0	0	0
CO4:	Describe the design of text editors and debuggers.	2	2	0	2	2	2	0	0	0	0	1	0
CO5:	Explain various macro processor features.	0	0	0	0	0	1	0	0	0	0	0	0
CO6:	Implement a lex and yacc program for a simple language.	0	0	0	0	0	0	0	0	0	0	0	0

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*

CMR
INSTITUTE OF

TECHNOLOGY
Internal Assessment Test 1 – September. 2016

Sub:	**SYSTEM SOFTWARE**						**Code:**	**10CS52**
Date:	**6/09 / 2016**	**Duration:**	**90 mins**	**Max Marks:**	**50**	**Sem:** **5**	**Branch:**	**CSE(B & C sec)**

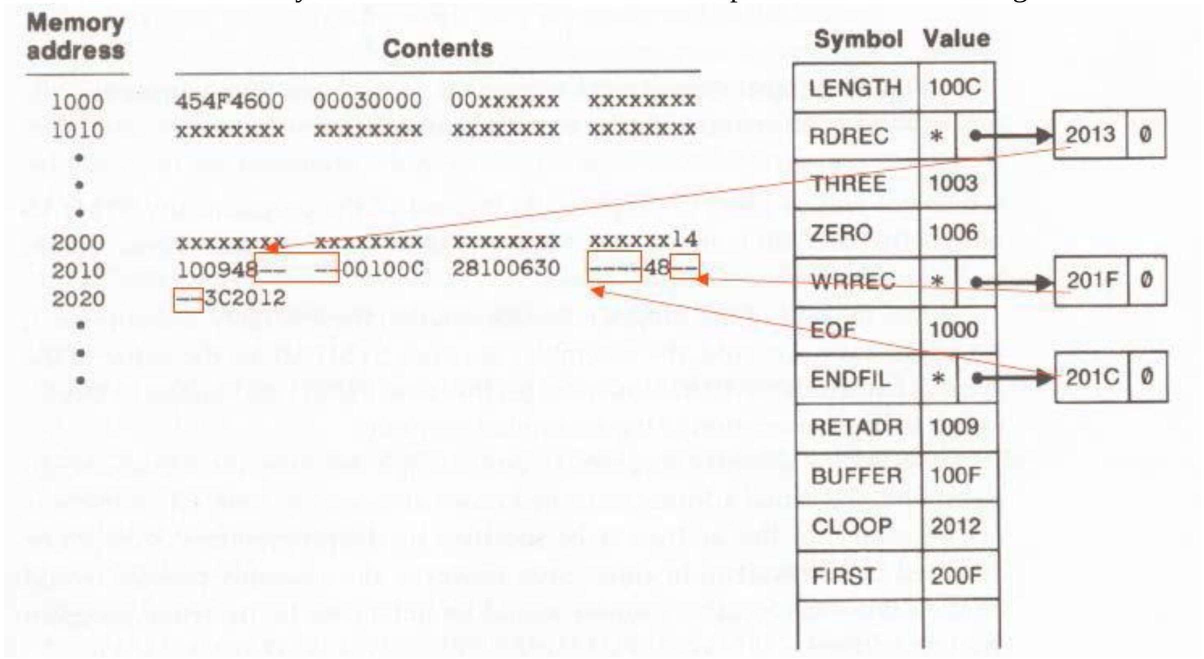
Note: Answer any five full questions. Each question carries 10 marks.

Scheme and Solution

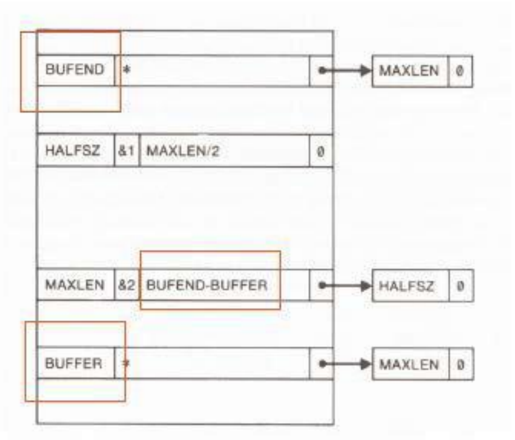
1. Discuss different design options of an assembler with suitable examples.

Mentioning 2 design options	1M
Explaining one pass assembler with example	4M
Explaining Multipass assembler with example	5M

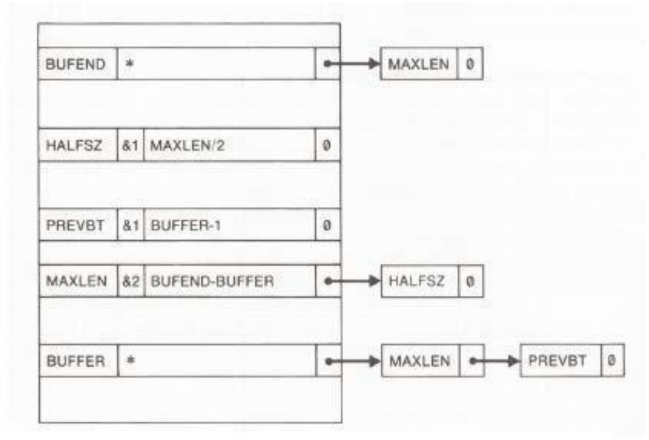
There are two types of one-pass assemblers:
• One that produces object code directly in memory for immediate execution (Load-and-go assemblers).
• The other type produces the usual kind of object code for later execution.
Load-and-Go Assembler • Load-and-go assembler generates their object code in memory 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 o 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.
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 o Symbol definition must be completed in pass 1. • Prohibiting forward references in symbol definition is not a serious inconvenience. o Forward references tend to create difficulty for a person reading the program.



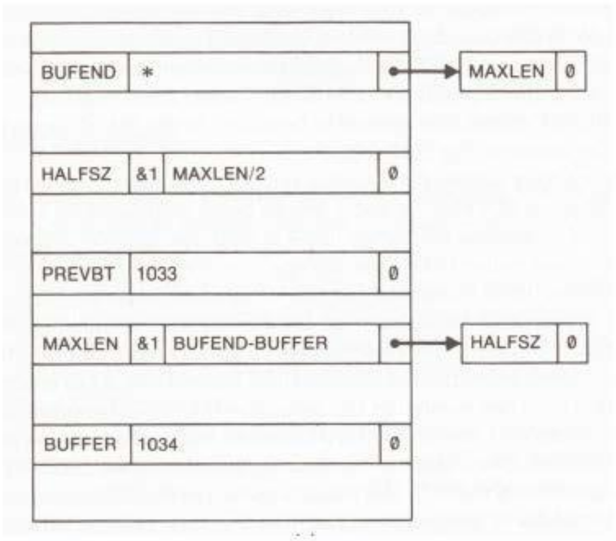
Multi_Pass Assembler: • For a two pass assembler, forward references in symbol definition are not allowed: ALPHA EQU BETA BETA EQU DELTA DELTA RESW 1 o Symbol definition must be completed in pass 1. • Prohibiting forward references in symbol definition is not a serious inconvenience. o Forward references tend to create difficulty for a person reading the program.



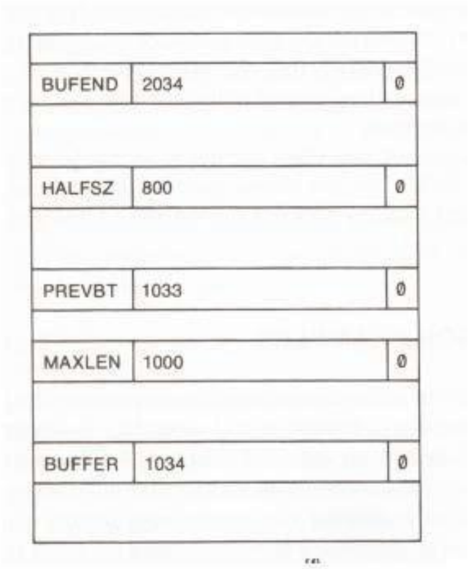
2 MAXLEN EQU BUFEND-BUFFER



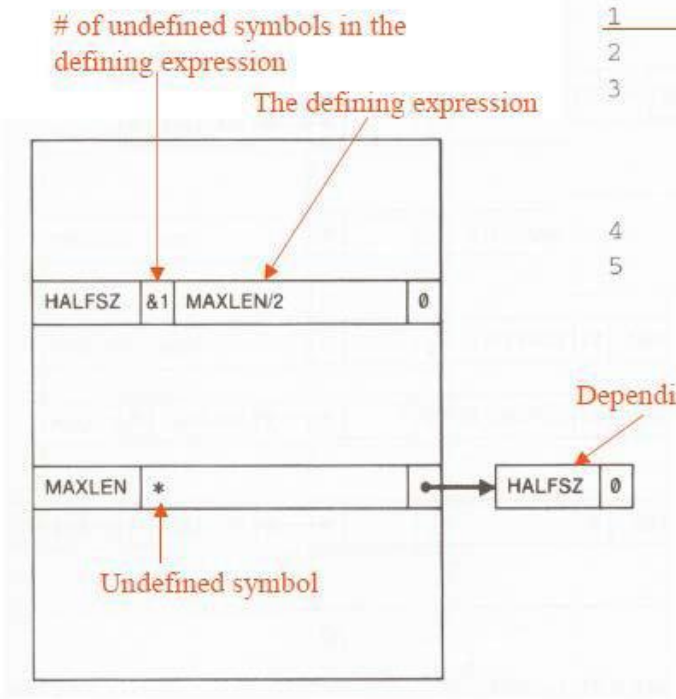
3 PREVBT EQU BUFFER-1



4 BUFFER RESB 4096



5 BUFEND EQU *



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

2. a Write a Yacc program to function as a calculator which performs addition, subtraction, multiplication, division and unary operations. 4M

```
/* Lex program to send tokens to the Yacc program */
%{
#include" y.tab.h"
 expern int yylval; %}
%% [0-9] digit char[_a-zA-Z] id {char} ({ char } | {digit })*
 %%
{digit}+ {yylval = atoi (yytext); return num; }
 {id} return name [ \t] ;
\n return 0; . return yytext [0];
%% /* Yacc Program to work as a calculator */
%{
 #include<stdio.h>
#include <string.h>
 #include <stdlib.h> %}
% token num name
 % left '+' '-'
% left '*' '/'
% left unaryminus
 %%
st : name '=' expn | expn { printf ("%d\n" $1); }
expn : num { $$ = $1 ; }
| expn '+' num { $$ = $1 + $3; }
| expn '-' num { $$ = $1 - $3; } |
expn '*' num { $$ = $1 * $3; } |
expn '/' num { if (num == 0) { printf ("div by zero \n"); exit (0); }
else { $$ = $1 / $3; } | '(' expn ')' { $$ = $2; } ;
%%
main() { yyparse(); }
 yyerror (char *s) { printf("%s", s); }
```

2b. Explain the following machine Independent features of an assembler.
a)Symbol Defining Statements b)Control Sections
Defining and explanation of symbol defining -3M
Defining and explanation of Control Sections-3M

a .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
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
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

b.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. 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

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

3(A). Generate the complete object program for the following SIC/XE program .
Generation of address 3M
Calculation of object codes 4M
Complete object program 3M

```
LOC                                                  object code
        COPY          START    1000
1000    CLOOP         +JSUB    RDREC                 A3101157
1004                  LDA      LENGTH                83214A
1007                  COMP     ZERO                  932141
100A                  JEQ      EXIT                  B32003
100D                  J        CLOOP                 BB2FFC
1010    EXIT          STA      BUFFER                532009
1013                  LDA      THREE                 832138
1016                  STA      TOTAL_LENGTH          53213B
1019                  RSUB
101C    BUFFER        RESW     100                   4C0000
1148    EOF           BYTE     C ’EOF’               454F46
114B    ZERO          WORD     0
114E    THREE         WORD     3
1151    LENGTH        RESW     1
1154    TOTAL_LENGTH  RESW     1
1157    RDREC         LDX      ZERO                  632001
```

MNEMONICS:

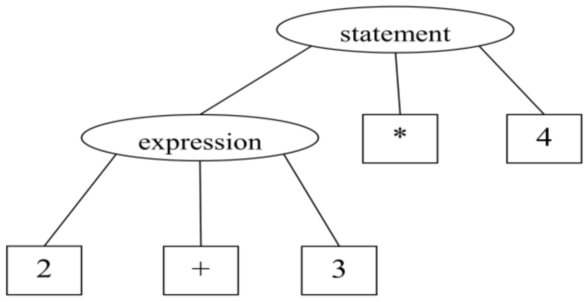
JSUB=A0, LDA=80, LDX=60, STA=50, COMP=90, RSUB=4C, JEQ=B0, J=B8

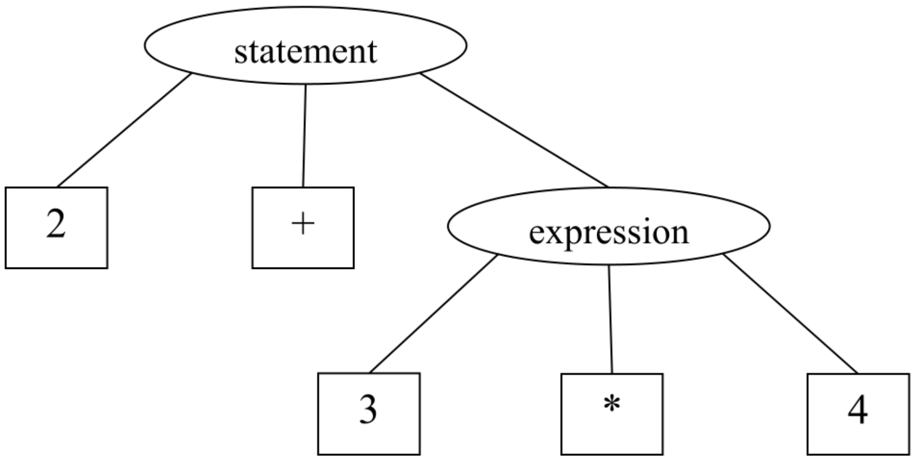
4 a. Explain the ambiguity in arithmetic expressions. What is the ambiguity in parsing 2+3*4 ?
Explain the solution for it.

Set of grammar rules is ambiguous if there is some input string that can be structured in two or more different ways. For example, the grammar rule **expr : expr '-' expr** is a natural way of expressing the fact that one way of forming an arithmetic expression is to put two other expressions together with a minus sign between them.

Unfortunately, this grammar rule does not completely specify the way that all complex inputs should be structured. For example, if the input is **expr - expr - expr** the rule allows this input to be structured as either **(expr - expr) - expr** or as **expr - (expr - expr)** (The first is called **left association**, the second **right association**).

Yacc detects such ambiguities when it is attempting to build the parser. It is instructive to consider the problem that confronts the parser when it is given an input such as **expr - expr - expr** When the parser has read the second expr, the input that it has seen: **expr - expr** matches the right side of the grammar rule above. The parser could reduce the input by applying this rule; after applying the rule; the input is reduced to expr (the left side of the rule). The parser would then read the final part of the input: **- expr** and again reduce.





- Parsing “2 + 3 * 4”
- It sees a “*”
  - Reduce “2 + 3” to an expression
  - Shift the “*” expecting to reduce

- Precedence
  - Controls which operator is to be executed first in an expression
  - Ex : a+b*c ⇔ a + (b * c)
    d/e-f ⇔ (d / e) – f
- Associativity
  - Controls grouping of operators at same precedence level
  - Left Associativity : a-b-c ⇔ (a-b)-c
  - Right Associativity : a=b=c ⇔ a=(b=c)

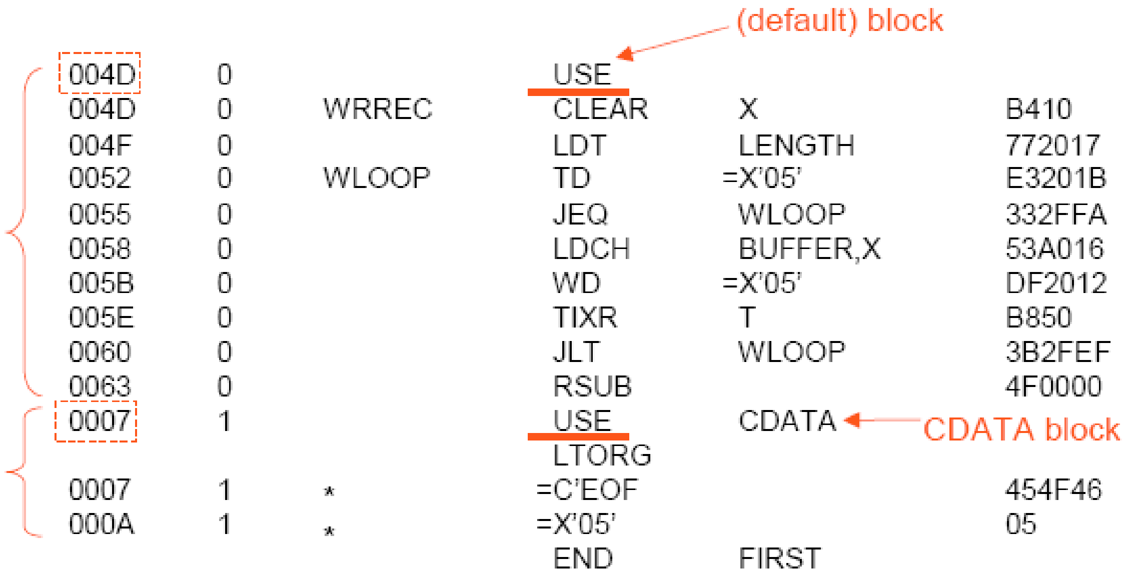
5a. What are Program Blocks? With a suitable example, explain how program blocks are handled by an assembler.

Definition of program block-1M
explanation of Program block-5M
Example program indicating different sections-4M

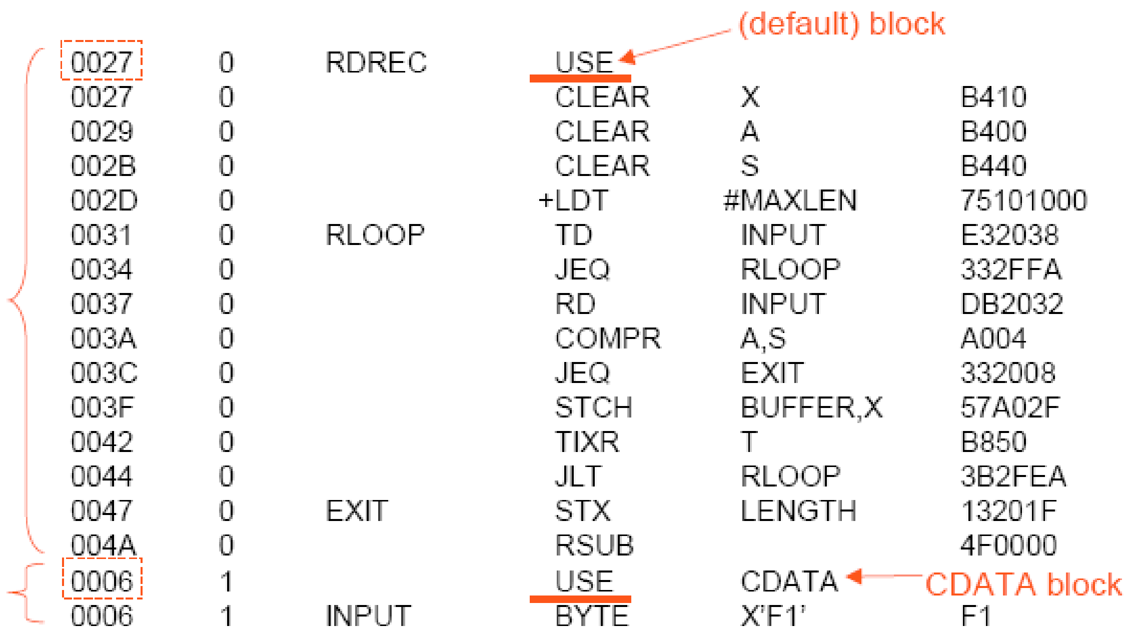
Program blocks allow the generated machine instructions and data to appear in the object program in a different order by Separating blocks for storing code, data, stack, and larger data block.

Assembler Directive USE: USE [blockname]

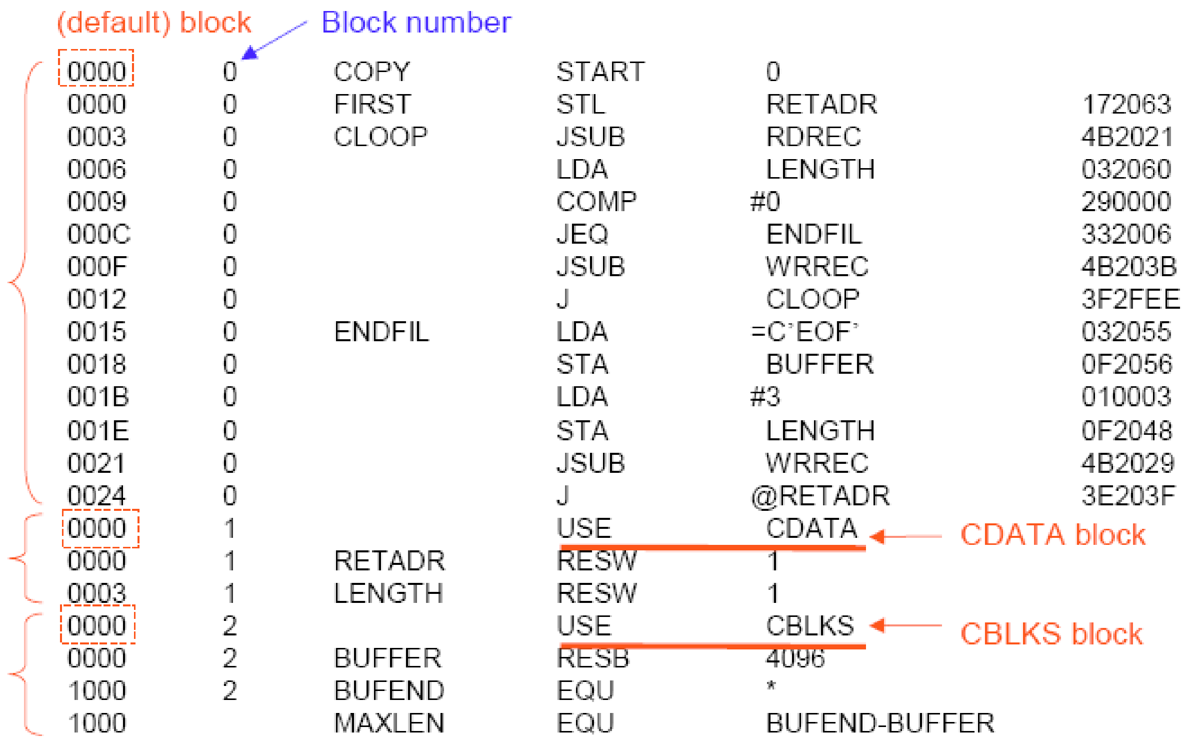
At the beginning, statements are assumed to be part of the *unnamed* (default) block. If no USE statements are included, the entire program belongs to this single block. Each program block may actually contain several separate segments of the source program. Assemblers rearrange these segments to gather 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.
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
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


(default) block
CDATA 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
                         LTORG
0007    1    *          =C'EOF                 454F46
000A    1    *          =X'05'                 05
                         END      FIRST
```


(default) block
CDATA 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
0006    1   INPUT    BYTE     X'F1'      F1
```


(default) block
Block number
CDATA block
CBLKS block

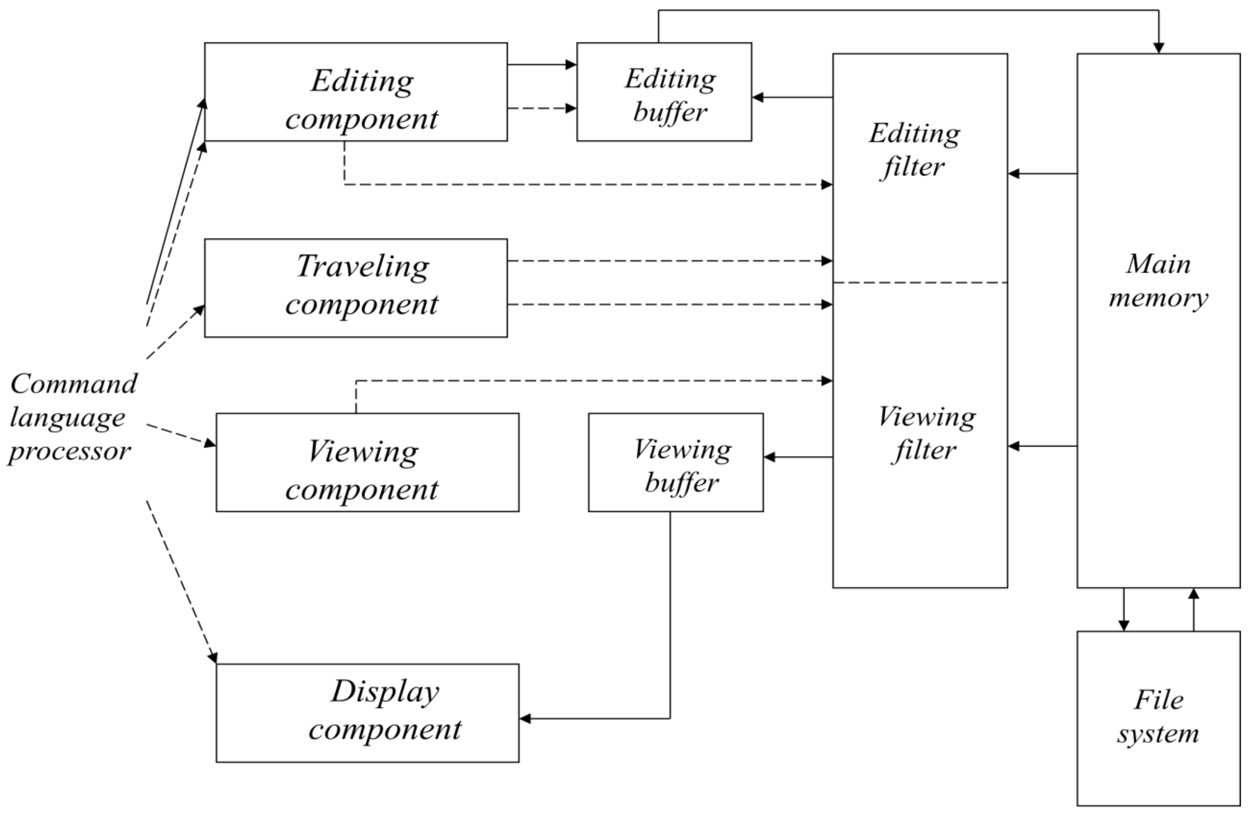
```
0000    0   COPY     START     0
0000    0   FIRST    STL       RETADR         172063
0003    0   CLOOP    JSUB      RDREC          4B2021
0006    0            LDA       LENGTH         032060
0009    0            COMP     #0              290000
000C    0            JEQ       ENDFIL         332006
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
0000    1   RETADR   RESW      1
0003    1   LENGTH   RESW      1
0000    2            USE       CBLKS
0000    2   BUFFER   RESB      4096
1000    2   BUFEND   EQU       *
1000        MAXLEN   EQU       BUFEND-BUFFER
```

6. With a neat diagram, explain the working of a typical Editor structure 10M

Text editor Structure-4M

Explanation about the various components and working -6M

Most text editors have a structure similar to that shown in the following figure. That is most text editors have a structure similar to shown in the figure regardless of features and the computers Command language Processor accepts command, uses semantic routines –performs functions such as editing and viewing. The semantic routines involve traveling, editing, viewing and display functions.



Typical editor structure

- Editing operations are specified explicitly by the user and display operations are specified implicitly by the editor. Traveling and viewing operations may be invoked either explicitly by the user or implicitly by the editing operations. In editing a document, the start of the area to be edited is determined by the current editing pointer maintained by the editing component.
- Editing component is a collection of modules dealing with editing tasks. Current editing pointer can be set or reset due to next paragraph, next screen, cut paragraph, paste paragraph etc..,. When editing command is issued, editing component invokes the editing filter –generates a new editing buffer – contains part of the document to be edited from current editing pointer.
- Filtering and editing may be interleaved, with no explicit editor buffer being created.
- In viewing a document, the start of the area to be viewed is determined by the current viewing pointer maintained by the viewing component. Viewing component is a collection of modules responsible for determining the next view. Current viewing pointer can be set or reset as a result of previous editing operation.
- When display needs to be updated, viewing component invokes the viewing filter – generates a new viewing buffer – contains part of the document to be viewed from current viewing pointer. In case of line editors – viewing buffer may contain the current line, Screen editors - viewing buffer contains a rectangular cutout of the quarter plane of the text.
- Viewing buffer is then passed to the display component of the editor, which produces a display by mapping the buffer to a rectangular subset of the screen –called a window. Identical – user edits the text directly on the screen. Disjoint –Find and Replace (For example, there are 150 lines of text, user is in 100th line, decides to change all occurrences of „text editor‟ with „editor‟).
- The editing and viewing buffers can also be partially overlap, or one may be completely contained in the other. Windows typically cover entire screen or a rectangular portion of it. May show different portions of the same file or portions of different file. Inter-file editing operations are possible.
- The components of the editor deal with a user document on two levels: In main memory and in the disk file

system. Loading an entire document into main memory may be infeasible – only part is loaded – demand paging is used – uses editor paging routines. • Documents may not be stored sequentially as a string of characters. Uses separate editor data structure that allows addition, deletion, and modification with a minimum of I/O and character movement.

7a . Distinguish between literals and Immediate operands. How does the assembler handle the Literal operands 6M

Mentioning 3 differnces-3M

Handling of literals-3M

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

Example:

45 001A ENDFIL LDA =C‟EOF‟ 032 - -

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 pool*s. 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.

.

7b**.** List the basic tasks of a text editor. 4M

Document-editing process in an interactive user-computer dialogue has four tasks

- Select the part of the target document to be viewed and manipulated
- Determine how to format this view on-line and how to display it
- Specify and execute operations that modify the target document
- Update the view appropriately