1100CST305122101 Pages 11

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

Scheme for Valuation/Answer Key

Scheme of evaluation (marks in brackets) and answers of problems/key

FIFTH SEMESTER B.TECH DEGREE EXAMINATION, DECEMBER 2021

Course Code: CST 305

Course Name: SYSTEM SOFTWARE

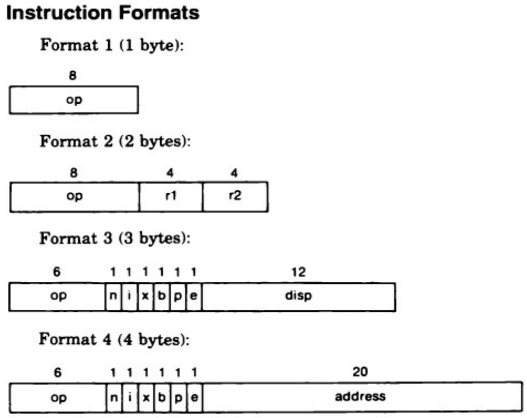
Max. Marks: 100 Duration: 3 Hours

PART A

(Answer all questions; each question carries 3 marks) Marks

1 I/O instructions- TD, RD, WD explain- 3 marks 3

2 3



Maximum 3 marks

3 The scenario in which a label is referenced in an instruction before it is defined is known as forward reference. – 1 mark 3

In one pass assembler it is handled by including a link field in the SYMTAB, which stores the links to the instructions which uses the label so that once the label address is obtained the corresponding instructions object code can be modified. – 1 mark

Suitable example – 1 mark

4 Any correct SIC/XE program can be given marks. Also if the logic is correct without floating point operation, partial marks can be awarded. 3

(Sample Program:

```
PGM START 1000
     LDF #4
     MULF BETA
     SUBF #9
     STF ALPHA
BETA BYTE 09 11 0A 23 24 56 ; 6 byte floating point number
```

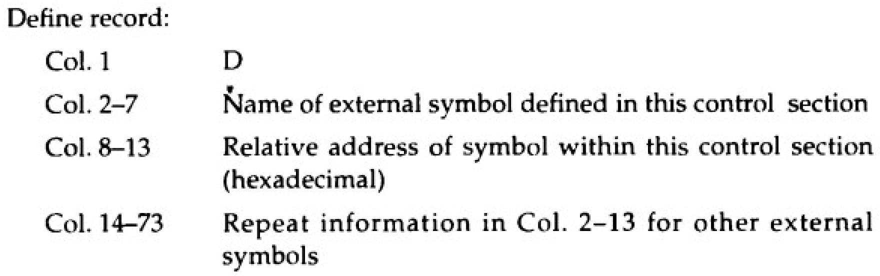
ALPHA RESB 6

END 1000)

Maximum 3 marks

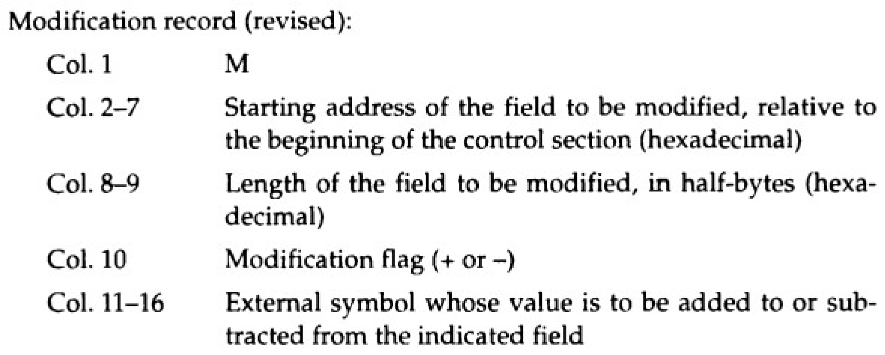
5 Modification record is used to list the object code fields which need to be modified as part of the loading and linking process. 3

Define record is used to list the labels of the control sections which would be referenced by some external control section.



Define record:

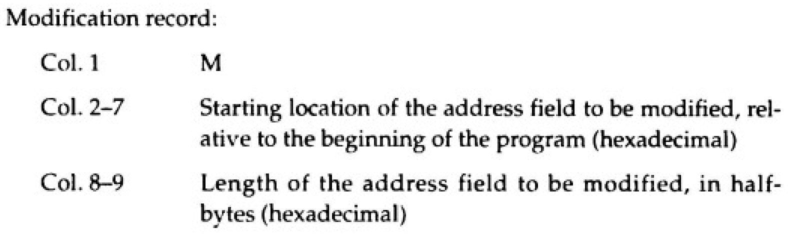
Col. 1	D
Col. 2–7	Name of external symbol defined in this control section
Col. 8–13	Relative address of symbol within this control section (hexadecimal)
Col. 14–73	Repeat information in Col. 2–13 for other external symbols



Modification record (revised):

Col. 1	M
Col. 2–7	Starting address of the field to be modified, relative to the beginning of the control section (hexadecimal)
Col. 8–9	Length of the field to be modified, in half-bytes (hexadecimal)
Col. 10	Modification flag (+ or –)
Col. 11–16	External symbol whose value is to be added to or subtracted from the indicated field

Or



Modification record:

Col. 1	M
Col. 2–7	Starting location of the address field to be modified, relative to the beginning of the program (hexadecimal)
Col. 8–9	Length of the address field to be modified, in half-bytes (hexadecimal)

Modification Record – 1.5 marks

Define Record – 1.5 marks

6 Machine dependent features depend on the underlying hardware while machine independent features is realized by the assembler and do not depend on the underlying hardware. 3

Examples:

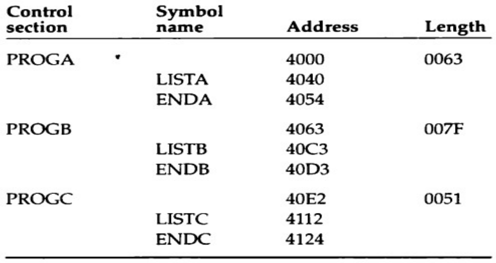
Machine dependent – Instruction format and addressing mode

Machine Independent – Literals, Symbol defining statements.

Basic explanation – 2 marks

Examples – 1 mark.

7 ESTAB stands for External Symbol Table. It is used to store the external labels defined within each control section and published through the Define record of each control section. It has four fields as shown below: 3



Control section	Symbol name	Address	Length
PROGA		4000	0063
	LISTA	4040	
	ENDA	4054	
PROGB		4063	007F
	LISTB	40C3	
	ENDB	40D3	
PROGC		40E2	0051
	LISTC	4112	
	ENDC	4124	

8 Bootstrap loader is used to load the first program (alternate loader) into the memory during the system start-up. 3

The main functions include:

i. Creating an environment which is conducive for the first program to execute.

ii. Locating and loading the first program to be executed.

iii. Transferring the control to the first program.

Maximum 3 marks.

Marks can be given if the process/algorithm is explained.

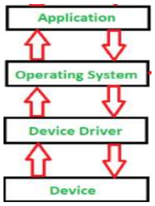
9 NAMTAB – Used to maintain the name and link to DEFTAB for each macro definition. 3

DEFTAB – Used to maintain the definition code for each macro

ARGTAB – Used during the expansion of the macro for storing the arguments.

1 mark for each. Maximum 3 marks.

10 The device driver has communication pathway as shown below. 3



Proper explanation. Maximum 3 marks

PART B

(Answer one full question from each module, each question carries 14 marks)

Module -1

11 a) Registers of SIC and SIC/XE – 2 marks 6

Instruction Format of SIC and SIC/XE – 2 marks

Data Formats of SIC and SIC/XE – 2 marks

b) Operating System: Controls and monitors the overall resource management of system – 2 marks 8

Assembler: Provides a means to the programmer to use mnemonic codes instead of machine instruction. – 2 marks

Compiler: Provides an environment to the programmer to apply modern programming concepts in code development without bothering the underlying hardware – 2 marks

Linker: Helps in linking the object code generated by different sources but meant for the same machine – 2 marks

12 a) SIC has no I/O channels while SIC/XE has IO channels. Both supports reading and writing of 8bits from and to a device identified by a 8bit address. 6

The instructions supported by SIC are

TD (Test Device), RD (Read Device) and WD (Write device)

Apart from the above instructions SIC/XE also supports:

TIO (Test IO), SIO (Start IO) and HIO (Halt IO)

Proper comparison – 2 marks

Instructions – 4 marks

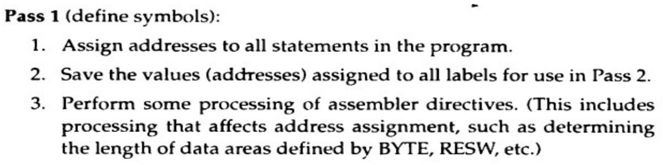
b) 8

		n	i	x	b	p	e	Addressing Mode
i	(PC) + disp	1	1	0	0	1	0	Program Counter Relative
		0	0	0	0	1	0	
ii	(B) + disp	1	1	0	1	0	0	Base Relative
		0	0	0	1	0	0	
iii	(PC) + disp + (X)	1	1	1	0	1	0	Program Counter Relative plus index
		0	0	1	0	1	0	
iv	(B) + disp + (X)	1	1	1	1	0	0	Base Relative plus index
		0	0	1	1	0	0	

2 marks for each binary representation (of n,I,x,b,p and e bits) -only SIC/XE needed- and the addressing modes. Maximum 8 marks

Module -2

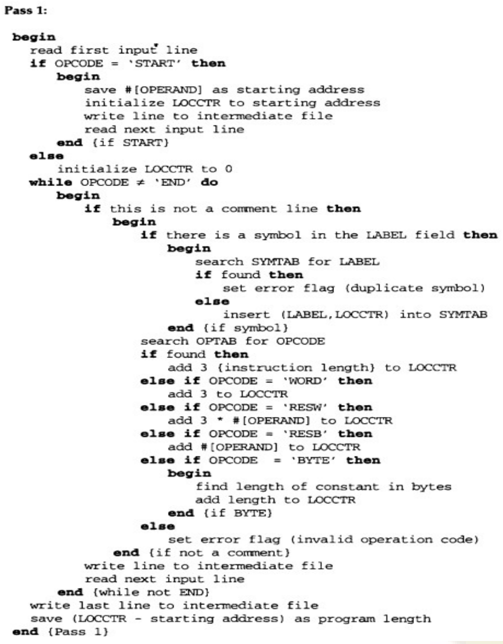
13 a) Functions of Pass 1 8



Pass 1 (define symbols):

1. Assign addresses to all statements in the program.
2. Save the values (addresses) assigned to all labels for use in Pass 2.
3. Perform some processing of assembler directives. (This includes processing that affects address assignment, such as determining the length of data areas defined by BYTE, RESW, etc.)

Algorithm:



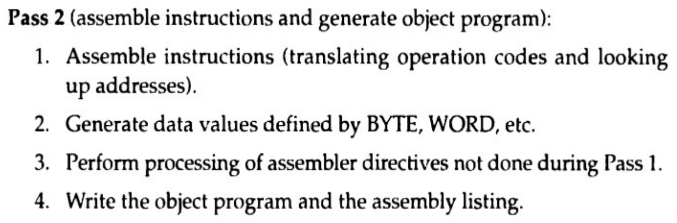
Pass 1:

```
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}
```

Functions – 2 marks. Algorithm – 6 marks

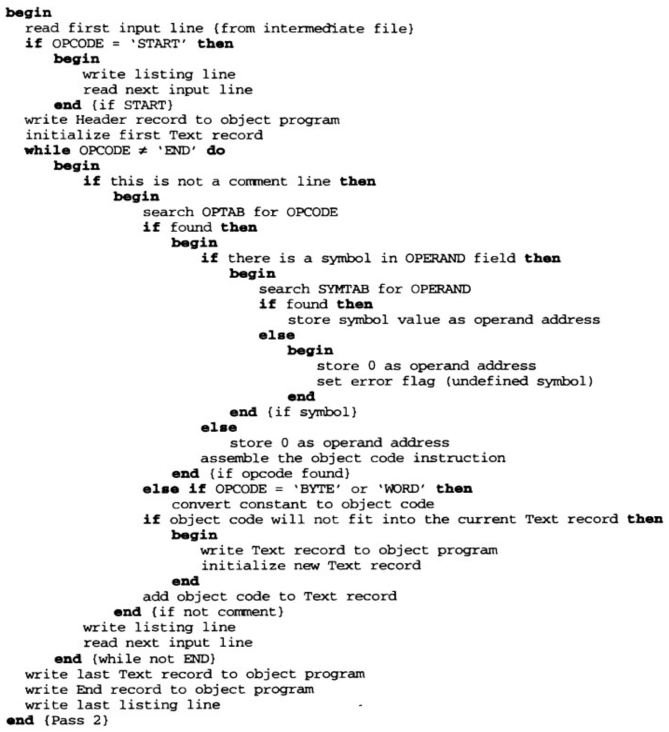
b) Any correct SIC program can be given full marks. Also if the program is partially correct, partial marks can be awarded. 6

14 a) 8



Pass 2 (assemble instructions and generate object program):

1. Assemble instructions (translating operation codes and looking up addresses).
2. Generate data values defined by BYTE, WORD, etc.
3. Perform processing of assembler directives not done during Pass 1.
4. Write the object program and the assembly listing.



```
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}
```

Functions – 2 marks. Algorithm – 6 marks

b) 6

Loc	Label	Opcode	Operand	ObjectCode
	SUM	START	4000	
4000	FIRST	LDX	ZERO	045788
4003		LDA	ZERO	005788
4006	LOOP	ADD	TABLE,X	18C015
4009		TIX	COUNT	2C5785
400C		JLT	LOOP	384006
400F		STA	TOTAL	0C578B
4012		RSUB		4C0000
4015	TABLE	RESW	2000	
5785	COUNT	RESW	1	
5788	ZERO	WORD	0	000000
578B	TOTAL	RESW	1	
578E		END	FIRST	

Object Program (optional)

H^SUM^4000^78E

T^4000^15^045788^005788^18C015^2C5785^384006^0C578B^4C0000

T^5788^3^000000

E^4000

Maximum – 7 marks (Full marks can be given if the object program in H, T and E is not written)

Module -3

15 a) Definition of control section – 1 marks 7

Control sections are defined using assembler directive such as CSECT. – 1 mark

Suitable example with proper use of CSECT – 2 marks

Object code program for each control sections have the records such as header, Define, Refer, text, Modification and End. Proper description for each records – 3 marks.

b) Relocation is required so that the object code program for given assembly program can be loaded at any given location and executed. It is controlled by the modification record within the object code program. - 2marks 7

Proper explanation with suitable examples – Maximum 5 marks

16 a) Segments are handled using the assembler directives such as CODE, STACK, CONST, DATA and USE. Proper explanation with appropriate examples – 3 marks 6

An instruction jump within the current code segment is known as near jump while if the destination address of the jump is outside the current code segment it is known as far jump. It is handled in MASM by explicitly specifying that a jump is near or far by using assembler directives FAR and SHORT along with the JNP instruction. Proper explanation with examples – 3 marks

b) Program Block Definition – 1 marks 8

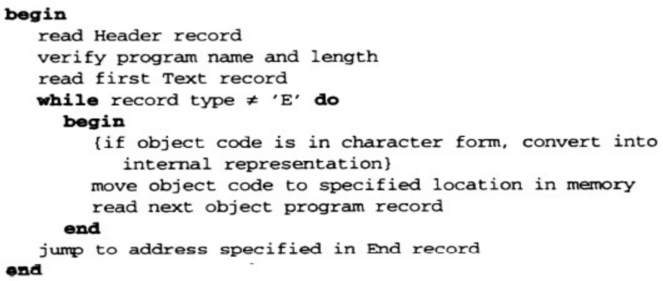
Example and Diagram – 5 marks

The records are for an assembly program containing program blocks contains the normal Header, Text, Modification and End record. If needed Define and Refer record can also be included. – 2 marks

Module -4

17 a) Definition – 1 mark 6

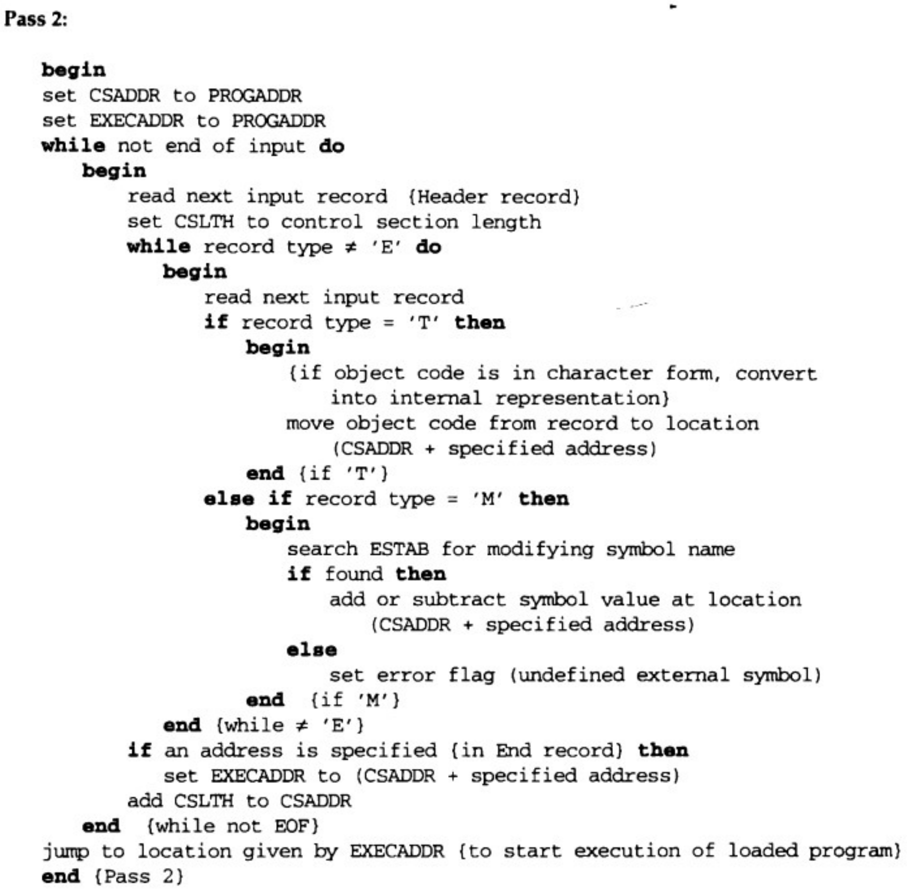
Algorithm – 5 marks



```
begin
   read Header record
   verify program name and length
   read first Text record
   while record type ≠ 'E' do
     begin
        {if object code is in character form, convert into
           internal representation}
        move object code to specified location in memory
        read next object program record
     end
   jump to address specified in End record
end
```

b) 8

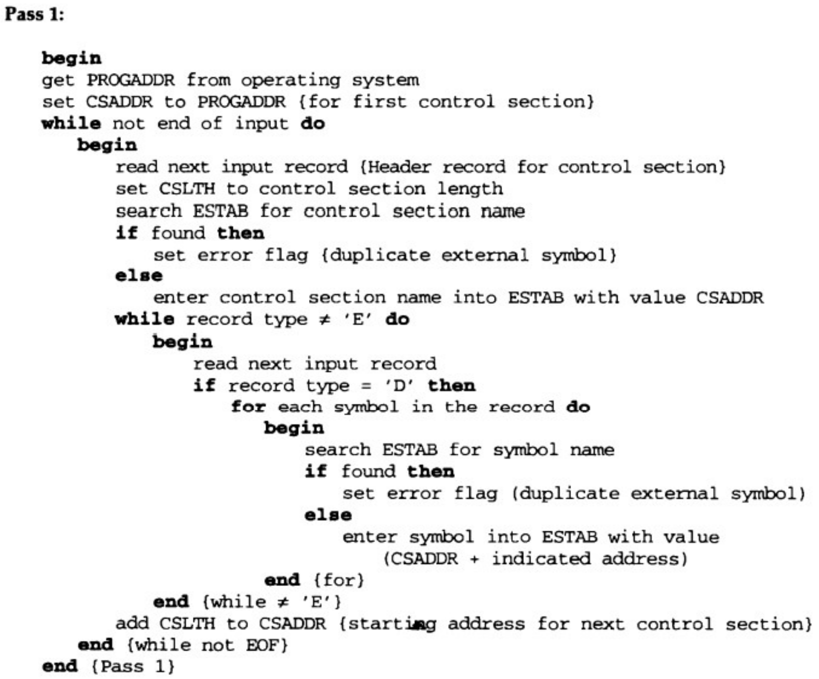
Algorithm – 8 marks



Pass 2:

```
begin
set CSADDR to PROGADDR
set EXECADDR to PROGADDR
while not end of input do
    begin
        read next input record {Header record}
        set CSLTH to control section length
        while record type ≠ 'E' do
           begin
               read next input record
               if record type = 'T' then
                   begin
                       {if object code is in character form, convert
                           into internal representation}
                       move object code from record to location
                           (CSADDR + specified address)
                   end {if 'T'}
               else if record type = 'M' then
                   begin
                       search ESTAB for modifying symbol name
                       if found then
                           add or subtract symbol value at location
                               (CSADDR + specified address)
                       else
                           set error flag (undefined external symbol)
                   end  {if 'M'}
           end {while ≠ 'E'}
        if an address is specified {in End record} then
           set EXECADDR to (CSADDR + specified address)
        add CSLTH to CSADDR
    end  {while not EOF}
jump to location given by EXECADDR {to start execution of loaded program}
end {Pass 2}
```

18 a) Algorithm – 7 marks 7



Pass 1:

```
begin
get PROGADDR from operating system
set CSADDR to PROGADDR {for first control section}
while not end of input do
   begin
      read next input record {Header record for control section}
      set CSLTH to control section length
      search ESTAB for control section name
      if found then
         set error flag {duplicate external symbol}
      else
         enter control section name into ESTAB with value CSADDR
      while record type ≠ 'E' do
         begin
            read next input record
            if record type = 'D' then
               for each symbol in the record do
                  begin
                     search ESTAB for symbol name
                     if found then
                        set error flag (duplicate external symbol)
                     else
                        enter symbol into ESTAB with value
                           (CSADDR + indicated address)
                  end {for}
         end {while ≠ 'E'}
      add CSLTH to CSADDR {starting address for next control section}
   end {while not EOF}
end {Pass 1}
```

b) Machine dependent features and machine independent features- list only 1 marks- (Relocation , Program Linking , Automatic Library Search , Loader Options) 7

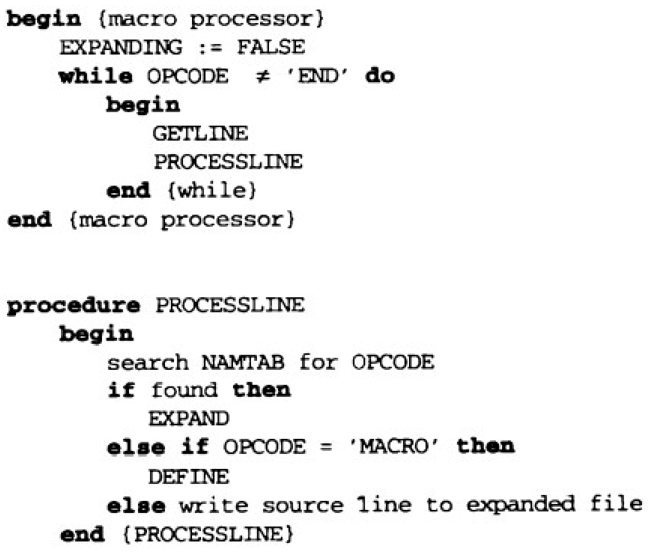
Explain Machine independent features – 6

Automatic Library Search – Proper explanation 3 marks

Loader Options – Proper explanation 3 marks

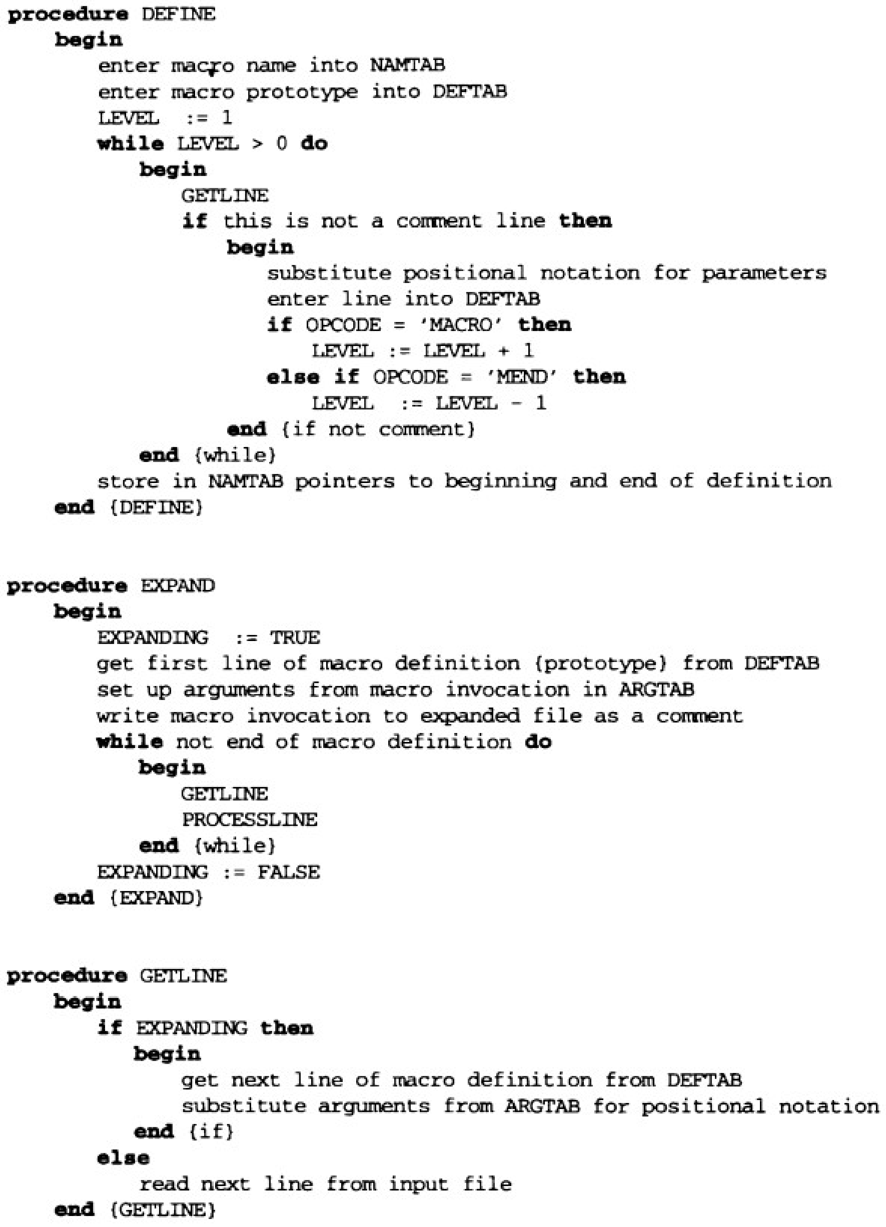
Module -5

19 a) 10



```
begin {macro processor}
   EXPANDING := FALSE
   while OPCODE ≠ 'END' do
      begin
         GETLINE
         PROCESSLINE
      end {while}
end {macro processor}


procedure PROCESSLINE
   begin
      search NAMTAB for OPCODE
      if found then
         EXPAND
      else if OPCODE = 'MACRO' then
         DEFINE
      else write source line to expanded file
   end {PROCESSLINE}
```



```
procedure DEFINE
    begin
       enter macro name into NAMTAB
       enter macro prototype into DEFTAB
       LEVEL  := 1
       while LEVEL > 0 do
          begin
             GETLINE
             if this is not a comment line then
                begin
                   substitute positional notation for parameters
                   enter line into DEFTAB
                   if OPCODE = 'MACRO' then
                      LEVEL := LEVEL + 1
                   else if OPCODE = 'MEND' then
                      LEVEL  := LEVEL - 1
                end {if not comment}
          end {while}
       store in NAMTAB pointers to beginning and end of definition
    end {DEFINE}


procedure EXPAND
    begin
       EXPANDING  := TRUE
       get first line of macro definition {prototype} from DEFTAB
       set up arguments from macro invocation in ARGTAB
       write macro invocation to expanded file as a comment
       while not end of macro definition do
          begin
             GETLINE
             PROCESSLINE
          end {while}
       EXPANDING := FALSE
    end {EXPAND}


procedure GETLINE
    begin
       if EXPANDING then
          begin
             get next line of macro definition from DEFTAB
             substitute arguments from ARGTAB for positional notation
          end {if}
       else
          read next line from input file
    end {GETLINE}
```

Algorithm – 10 marks

b) 4 comparison points. Maximum 4 marks. 4

20 a) 10

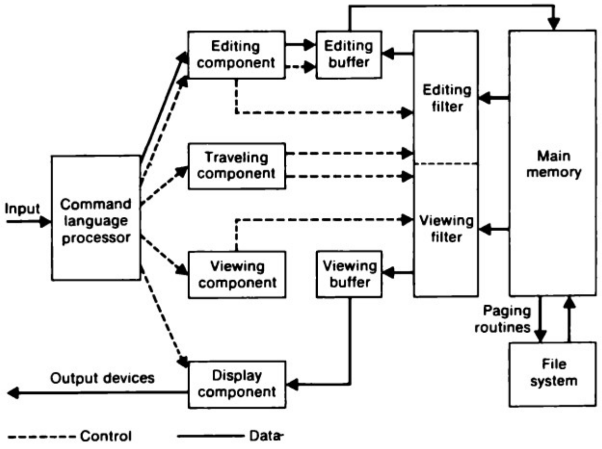


Diagram – 4 marks

Proper description of each component – 6 marks

b) Any 2 points of comparison. Maximum 4 marks. 4

****