Total No. of Questions : 6] InSem - MAR-18 SEAT No. :

P539 [Total No. of Pages : 2

APR - 18/TE/Insem. - 141

T.E. (Computer Engineering)

SYSTEMS PROGRAMMING AND OPERATING SYSTEM

(2015 Course) (Semester - II) (310251)

Time : 1 Hour] *[Max. Marks : 30*

Instructions to the candidates:

1) *Answer Q1 or Q2, Q3 or Q4, Q5 or Q6.*
2) *Neat diagrams must be drawn wherever necessary.*
3) *Figures to the right indicate full marks.*
4) *Assume suitable data, if necessary.*

Q1) a) Differentiate between literal and immediate operand. How assembler handles them? Give examples. [6]

b) Define Assembler Directive. Explain ORGIN, EQU & LTORG with example. [4]

OR

Q2) a) Consider following Assembly code and show output of pass-1 of two pass assembler with entries in MOT, POT, ST, LT and BT. [5]

```
PROG START 50

USING PROG+2, 15

L 1, FIVE

A 1, = F '2'

LTORG

ST 1, RES

FIVE DC F '4'

RES DS IF

END
```

b) Discuss with example what is forward reference? How is it handled in single pass assembler? [5]

P.T.O.

Q3) a) Define Macro. What are the advantages of macro facility? How are macros different from functions? **[6]**

b) What is the need of DLL? Differentiate between How Dynamic and static linking? **[4]**

OR

Q4) a) What are the types of loaders? Explain compile and go loader scheme with advantages and disadvantages using suitable diagram. **[6]**

b) Comment on the statement "Programs with macros require more space and less time at runtime than programs with functions". **[2]**

c) Discuss four different functions of loader. **[2]**

Q5) a) What is interpreter? Explain various components of interpreter? **[4]**

b) Consider input "d = a + b * 2;" and show the output of each phase of compiler with suitable diagram? **[6]**

OR

Q6) a) Write regular expressions to recognize following : **[4]**

i) Signed and unsigned integer numbers.

ii) Identifiers.

iii) Few Keyword in "C" program.

iv) Relational Operators.

b) What is LEX? Explain working of LEX with suitable diagram? **[4]**

c) What is Syntax Error? Give suitable example? **[2]**

++++

Name: Prof. Gharu A. N.
College: PVGCOE, Nashik
Dept: COMPUTER
Subject: SPOS
Date: 08/03/2018.

SYSTEM PROGRAMMING & OPERATING SYSTEM

INSEM MODEL ANSWER - 2018

08/03/2018.

Q.1 a) Differentiate between literal & Immediate operand. How assembler handle them ? Give e.g. -[6 Marks]

→ [4-marks].

Sr No	Literal	Sr No	Immediate Operand.
1.	Assembler generates specified value as a constant at Some other mem. location.	1.	The operand value is assembled as part of the m/c instruction.
2.	Example: MOVER B, '=5'	2.	Example: MOVE A, #5
3.	Literal use '=' symbol	3.	Immediate Operand use '#' sym.
4	There is mem. reference	4.	There is no mem reference
5.	LTORG (AD) instr is needed	5.	LTORG Assembler directive is not needed

- Assembler recognise the immediate operand in the instruction by scanning # symbol with constant value.
  e.g. MOVER AREG, #5 , Y = 5

[2-marks]

- Assembler determines the literal in the instr by scanning = symbol with constant value.
  e.g. MOVER AREG, '=5'

and

- LTORG instr is used to recognise the literal from literal table & it makes entry in pool table as total nos of literal.
- Literal value cannt be changed during exe.

b) Define Assembler Directive. Explain ORIGIN, EQU & LTORG with example? [04-Marks]

→ Assembler Directives are the instruⁿ that gives the direction to the assembeer, which operation is to be. performed. —[1 Marks]

1] ORIGIN ÷

- This directive gives the direction to assemble to set. Origin location for particular prog/sub-prog. [1 marks]
- Syntax ÷

  ORIGIN <addr. specification>

  where addr. specⁿ is an operand, constant.
- This directive sets the address g LC to address given in instruⁿ.
- e.g. ORIGIN 200

2] LTORG ÷ —[1 marks]

- LTORG instruⁿ/Directives allows programmer to specify where literal should be placed.
- If LTORG instruⁿ is not used/present, literal are placed after END directive.

```
e.g.  LC
      205   MOVER A, '=2'
      206   MOVER B, '=4'
            LTORG
```

LT

Symb	Addr
=1	207
=2	208

PT

0	2
1	-
2	-

pool table.

- When LTORG instruⁿ occured then total nos literal g literal table counts & stored entry in pool table. it also assign LC to literal.

3] EQU ÷ [1-marks]

- EQU directive allow to set equale address to variable/label.
- Syntax is ÷

  <symbol> EQU <address specification>

  where addr. specⁿ can be operand or constant.
- The EQU simply associates symbol with the addr. specⁿ

e.g.

BACK EQU LOOP

The symbol back is set to address g loop.

Q. 2 a) Consider the following Assembly code and show o/p. g Pass - I g two pass assembler with entries in MOT, POT, ST, LT & BT. [5-Marks]

```
PROG START 50
USING PROG+2, 15
L1, FIVE
 A1 , =F'2'
LTORG
ST 1, RES
FIVE DC F'4'
RES DS 1F
END.
```

→

```
LC
50   USING PROG+2, 15
51    L1 , FIVE
52    A1 , =F'2'
      LTORG
54    ST 1, RES
55    FIVE DC F'4'
56    RES DS 1F.
```

[02-marks]

ST

Symbol	Addres
FIVE	55
RES	56.

LT

Literal	addres
=F'2'	53.

PT

0	1

[02-marks]

MOT

Mnemonics opcode	class	Opcode	length
PROG START 50	AD	01	-
USING PROG+2, 15	IS	-	-
L1, FIVE	RG	01	-
A1, =F'2'	RG	02	-
LTORG	AD	05	
ST 1, RES.	IS	-	
FIVE DC F'4'	DL	02	
RES DS 1F	DL	01	
END .	AD	02	-

[01-marks]

Q 2 Discuss with example what is forward References?
How is it handled in Single Pass Assembler? [05-marks]

→ forward references mean that variable define in the program before the declaratian then we can say it is forward reference.

e.g.
```
START 100
MOVER AREG, X
______
______
______
X DC '1'
```
A forward reference to variable X.

[02-marks]

- As assembler can't generate m/c code for assembly instn with forward references.
- It can be generated after address of variable used in instn is known
- Symbol table is used to store addresses of variables. These addresses can be used during genn of m/c code.

• How to Handle forward References.

e.g.
```
     START 100
100  MOVER AREG, X
101  ADD   BREG, ABC
102  COMP  BREG, XY2
103  STOP
104  X   DC '2'
105  ABC DC '1'
106  XY2 DC '5'
107  END.
```
Forward References

Symbol	address
X	104
ABC	105
XY2	106.

[03-marks]

- In above e.g. you can see X, ABC & XY2 are forward references variable, which are stored in Symbol table &
- address are assigned to it as variable is declared
- If variable is used in instn & value is not available, then such condn is called as Backpatching.

Q.3. a) Define macro. what are the advantages of macro facility. How are macro different from function. [6-Marks]

→ "macro allows sequence of code to be defined once and then refered by name each time it is toe be refered."

- Each time this name occurs in prog, the seq. of code is substituted at this point.
- It consist of Name of macro, set of parameter & Body of macro.

Syntax: macro name
macro body
End of def^n

example: myprog
ADD AREG, A
ADD BREG, B
MEND.

[02-marks]

* Advantages of MACRO:-

- macro facility allow user to defined code once & used it in prog. multiple by it's name.
- easy to understand [02-marks]
- it contain small code.
- requires less space.
- it can be called by it's name.
- parameter can be passed.
- same procedure can be used multiple time.
- macro can be expanded.

[02-marks]

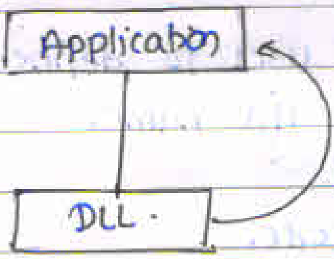
Sr No	Macro	Sr No	Function
1.	macro can be small code defined once & used in Prog. repeatedly.	1.	function can be program, it can be used as we need by calling it's name
2.	execution speed is more bcz no calling overhead	2	Speed is less due to Calling overhead.
3.	Macro can't handle label	3	Function can handle label.
4	Macro can be expaned	4.	Fu^n can't be expanded.
5.	Macro is defined in same code	5.	Fu^n can be defined outside code.

Q-3 b) what is need of DLL. Differena betⁿ static & Dynamic Linking.

[04-Marks]

⟶ - it stands for Dynamic Link library

- Dynamic Link Library is microsoft implementation of shared library in window. [02-marks]
- The file format for DLL are same as windows EXE. file.
- A DLL can contain code, data & Resources.
- Need of DLL :-
  - To share code in program, DLL is required.
  - shared code is placed in single seperate file.
  - DLL code may be shared But data is private.
  - DLL allow Interprocess commⁿ thro. shared memory.



DLL calling a fuⁿ in an application.

- DLL may control several clients, when event occures from DLL.

* Compare static & Dynamic Linking :- [02-marks]

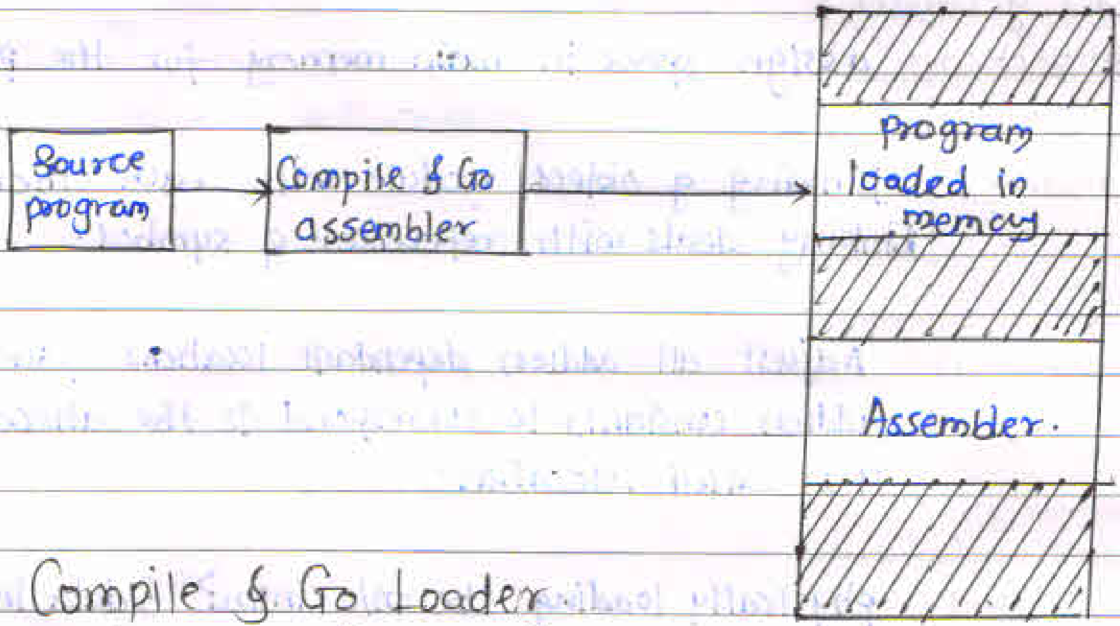
Sr NO	Static Linking	Sr NO	Dynamic Linking.
1.	Static linker takes object files produced by compiler including lib. fuⁿ & produces an executable file.	1.	Dynamic linking defers much of linking process until a prog. starts running.
2.	static linking can be fixed	2.	Dynamic linking can be variable.
3.	No relocation during run time.	3.	Perform relocⁿ during run time
4	Linking is done at compile time	4	Linking is done at run time.
5.	there is wastage of memory	5.	There is no wastage of memory
6.	It does not support sharing of libraries/memory	6.	Dynamic linking provides automatic sharing of code.

Q.4 a) What are the types of Loaders? Explain Compile & Go loader schemes with advantages & Disadvantages. [6-Marks]

→ Types of loader:-

1) Compile & Go loader 4) Subroutine linkage [02-marks]
2) General Loader 5) Relocating Loader
3) Absolute Loader 6) Direct linking Loader.

- Compile & Go Loader:- [02-marks]
- In this, Assembler is loaded in one part of memory and it place the assembled prog. (m/c instrⁿ) directly into their assigned memory location.
- After loading process completes, the assembler transfers the control to starting instrⁿ of the loaded prog.



Compile & Go Loader

- Advantages:- [02-marks]

1) easy to implement
2) It is very simple scheme.
3) Assembler available in memory.

- Disadvantages:-

1) A portion of memory is wasted, as it is occupied by assembler.
2) It is difficult to handle multiple segment
3) It is required to retranslate user prog. every time.
4) Difficult to develop modular program.

Q.4 b) Comment on the statement "Program with macro requires more space and less time at runtime than programs with functions" [02-marks]

→ As macro declared as once and used multiple time in prog. by calling with it's name.

- macro requires more space because when macro is called in program then macro is expanded so. seq. of instrⁿ can be increased (more space required)

- macro can be called using it's name in program and macro is also present in same program so that less time. required to call macro. rather than function. [02-marks]

Q.4 c) Discuss 4 different function of Loader. [02-marks]

→ Fuⁿ of Loader :-

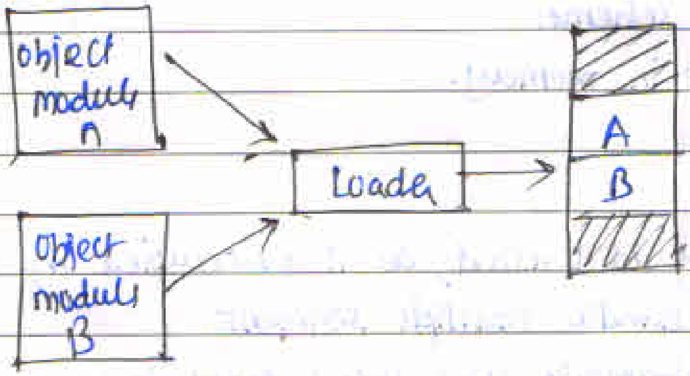
1) Allocation - assign space in main memory for the prog.

2) Linking :- Linking of object module with each other. Linking deals with references of symbol.

3) Relocation :- Adjust all addres dependant locations, such as addres constants to correspond to the allocated space, it is called relocation.

4) Loading :- physically loading the m/c instrⁿ & data into main memory.

[02-marks]



General Loader Scheme.

Q.5. a) what is Interpreter? Explain Various Component g Interpreter?

→ " Interpreter is a program/tool that which reads the source. code one instruction or line at a time, convert this line into machine code. or some intermediate form & execute it.

* Component g Interpreter :- [01-marks]

1) Assembler 3) Linker 5) Debugger

2) loader 4) preprocessor [03-marks]

1) Assembler :- is a program which converts Assembly lang. into machine level language (code)

2) Loader - it loads the prog. into main memory for the execution.

3) Linker - it links multiple module/ prog. together for exe.

4) Debugger :- is used to debug the program line by line.

Q.5 b) Consider input " d = a+b*2; " and show the o/p g each phase g compiler with suitable diagram? [06-marks]

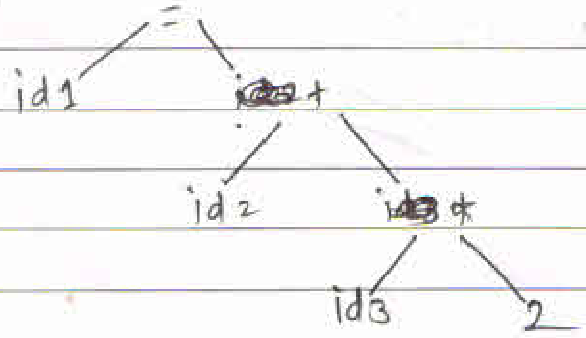
→ 1) Lexical Analysis :-

d = a+b*2; [01-mark]

id1 = id2 + id3 * 2;

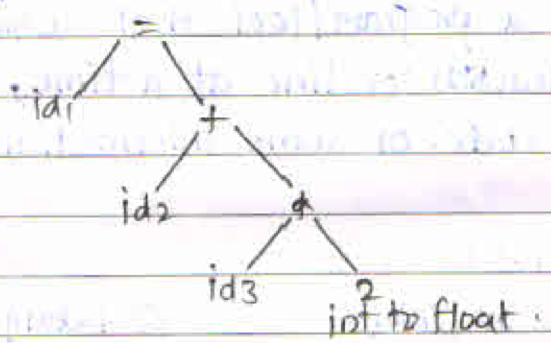
it scan the from left to right & generates token

2) Syntax Analysis [01-mark]



it checks the syntax g prog.

3) Semantic Analysis :- [01-mark]



It perform type checking & conversion.

4) Intermediate Code Generation :- [01-mark]

```
t1 = Intoreal (2)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3.
```

5) Code optimizer :- [01-mark]

```
t1 = id3 * 2·0
id1 = id2 + t1
```

6) Code Generator :- [01-mark]

```
MOVF  R1, id3
MULF  R1, #2·0
MOVF  R2, id2
ADDF  R1, R2
MOVF  id1, R1
```

Q6. c) What is Syntax Error? Give Suitable Example. [02-marks]

→ " Syntax error is used checks the Syntax of program whether it is properly defined or not " [02-marks]

- This oper^n is done in 2nd phase of compile. i.e. syntax analysis
- It is also called as syntax tree.
- It gives various types of error.

e.g. lvalue, rvalue not declared,
Semicolon not defined,

e.g.

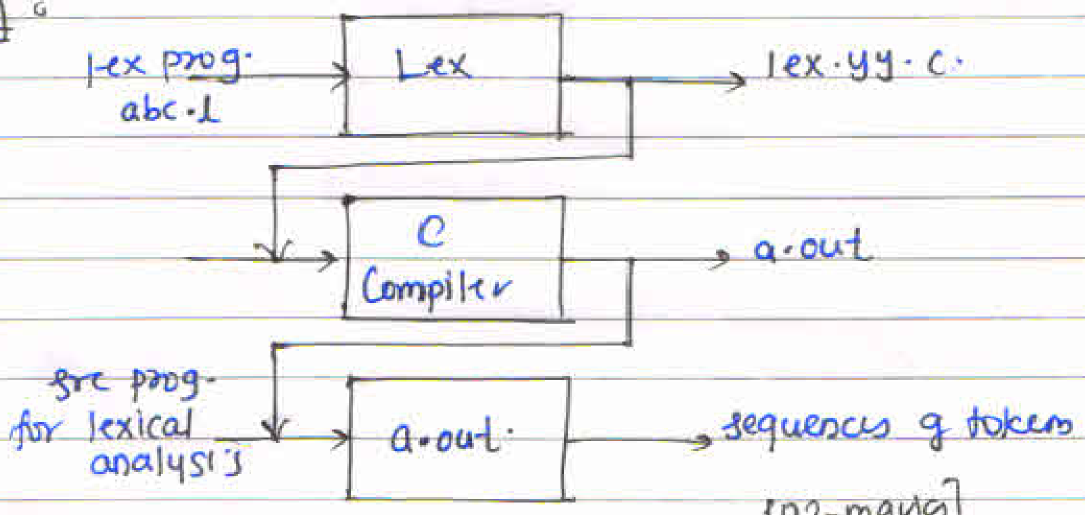
x = 4+2*30

Semi-colon missing error.

Q6. b) What is LEX? Explain working of LEX with suitable example? [04-Marks]

→ "LEX is lexical Analyzer generator tool, which is used to scan the program from left to right & seperate out the token to generate parse tree." [01-marks]

Working:



[02-marks]

- Lex is s/w tool, which takes i/p as regular expression together with action to be taken on recognising each of these pattern.
- The o/p of lex is C program for lexical analysis.
- .l prog is to be passed to Lex. it compile lex file & generate C files. then C compiler compiles C file & generate exe files.
- & finally generates token using a.out cmd [01-marks]

Q6. b) Syntex g Lex

→ declaration

%.%

translation Rule.

%.%:

User function

How to run LEX code

```
> Lex abc.l
> gcc cc.yy.c.
> a-out ↵
```

Q 6 a) write Regular exprenion to recognise following.

1) Signed & Unsigned Integer number

2) Identifier

3) Few keyword in 'C' prog

4) Relational Operator. [04-Marks]

→ 1) The set g strings accepted by finite automata is known as regular lang.

RE = 1, 2, 3, 4.   RE = -1, -2, -3

(01-marks)

2) RE = (id1+id2+id3)*

[01-marks]

3) RE = (for+while+Do)

[01-marks]

4) RE = (==+!=+<=+>=)*

[01-marks].

Note: Question No. 6 a), you can solve as per concept g Theory of Computation.

* * *