Q.2 a. There are different ways in which resources can be shared by a set of programs. Discuss them briefly. (8)

Answer: Refer pages 274-275 of Text Book-I

b. Discuss briefly about four fundamental states for a process. (8)

Answer: Refer page 322 of Text Book-I

Q.3 a. Many tasks are involved in process scheduling. State them briefly. (8)

Answer: Refer page 353 of Text Book-I

b. Write down the conditions that are necessary for deadlocks to arise. Also discuss about one fundamental approach which is generally used for handling deadlocks. (4+4)

Answer: Refer pages 376-377 of Text Book-I

Q.4 a. Write down the properties of a CS Implementation. How will you illustrate a CS implementation using a binary semaphore with the help of figure? (4+4)

Answer: Refer pages 399, 413 of Text Book-I

b. Discuss briefly about the UNIX file system. (8)

Answer: Refer page 584 of Text Book-I

Q.5 a. During the execution of a program, the memory allocated to it contains various components. Discuss them briefly. (8)

Answer: Refer page 469 of Text Book-I

b. Discuss briefly the kernel actions to implement memory protection with the help of suitable figure. (8)

Answer: Refer pages 472, 473 of Text Book-I

Q.6 a. What do you mean by language processing? Describe language processing activities. (8)

Answer: Refer pages 5, 2 of Text Book-I

b. How the data structures used for language processors are classified? Explain. (8)

Answer: Refer pages 36, 37 of Text Book-I

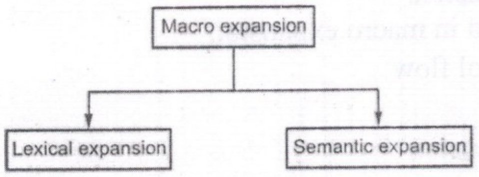
Q.7 a. Describe parsing and what are two fundamental approaches to parsing? Draw parse tree and abstract syntax tree for the source string a+b*c. (8)

b. What is macro-expansion? List the key notions concerning macro expansion. Write an algorithm to outline the macro-expansion using macro-expansion counter. (8)

Answer:

Macro Definition and Expansion

- Macro is a unit of specification for program generation through expansion.
- A macro consists of a name, a set of formal parameters and body of code.



4

- **Macro expansion** is a macro name with a set of formal parameters is replaced by some code.
- **Lexical expansion** is replacement of a character string by another character string during program generation.
- **Semantic expansion** is generation of instructions tailored to the requirements of specific usage.
- Macro definition consists of
  i) macro prototype ii) one of more model iii) macro preprocessor
- Macro definition is in between a macro header statement and a macro e statement.
- Syntax of macro prototype statement
  < macro name> [< formal parameter spec> [; ...]] where
  < macro name> = It is in the mnemonic field of an assembly statement < formal parameter spec > is in the following form
  < parameter name > [< parameter kind >]
- Syntax of the macro call is
  < macro name> [< actual parameter spec> [; ...]]

4

Macro Expansion

- Macro call leads to macro expansion.
- Macro call statement is replaced by a sequence of assembly statement in macro expansion.
- Two key notions are used in macro expansion
  a) Expansion time control flow
  b) Lexical substitution
- Algorithm for macro expansion

```
1. Initialise the macro expansion counter (MEC)
2. Check the statement which is pointed by MEC is not a MEND
statement a) if the statement is model statement
        then
              expand the statement
              and increment the MEC
              by 1
        else
        MEC := new value specified in the
   statement;
3. Exit from the macro expansion
```

Q.8 a. What is assembler and also write about task performed by the passes of a two pass assembler? (6)

Answer:

An assembler is a type of computer program that interprets software programs written in assembly language into machine language, code and instructions that can be executed by a computer. An assembler enables software and application developers to access, operate and manage a computer's hardware architecture and components.

An assembler is sometimes referred to as the compiler of assembly language. It also provides the services of an interpreter.

2mks

Passes of a two pass assembler are

Pass I

1. Separate the symbol, mnemonics opcode and operands fields.
2. Build the symbol table.
3. Perform LC processing
4. Construct intermediate representation.

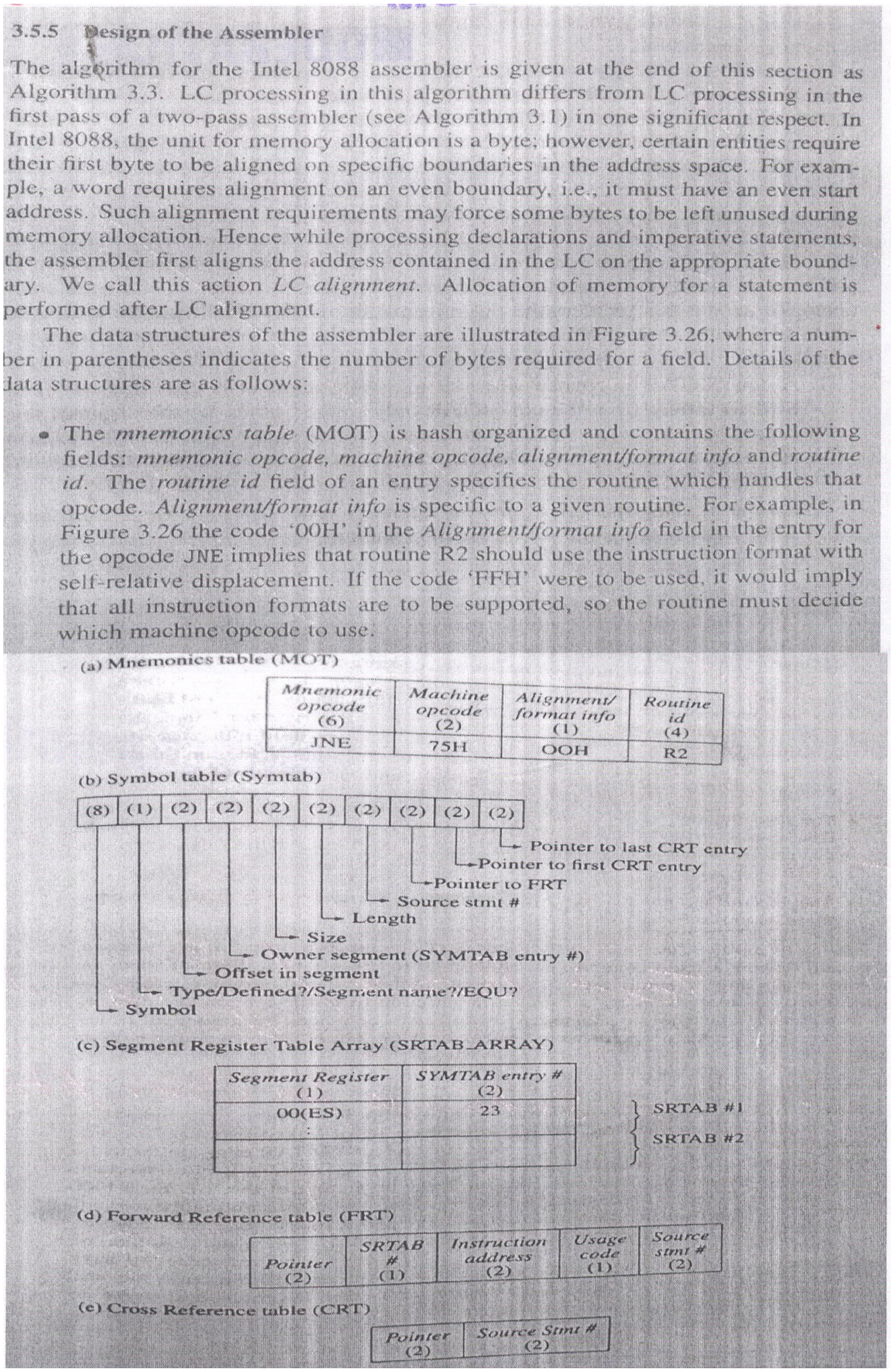
3mks

Pass II Synthesize the target program.

3mks

b. Describe data structure of the assembler **(10)**

Answer:



3.5.5 Design of the Assembler

The algorithm for the Intel 8088 assembler is given at the end of this section as Algorithm 3.3. LC processing in this algorithm differs from LC processing in the first pass of a two-pass assembler (see Algorithm 3.1) in one significant respect. In Intel 8088, the unit for memory allocation is a byte; however, certain entities require their first byte to be aligned on specific boundaries in the address space. For example, a word requires alignment on an even boundary, i.e., it must have an even start address. Such alignment requirements may force some bytes to be left unused during memory allocation. Hence while processing declarations and imperative statements, the assembler first aligns the address contained in the LC on the appropriate boundary. We call this action *LC alignment*. Allocation of memory for a statement is performed after LC alignment.

The data structures of the assembler are illustrated in Figure 3.26, where a number in parentheses indicates the number of bytes required for a field. Details of the data structures are as follows:

- The *mnemonics table* (MOT) is hash organized and contains the following fields: *mnemonic opcode, machine opcode, alignment/format info* and *routine id*. The *routine id* field of an entry specifies the routine which handles that opcode. *Alignment/format info* is specific to a given routine. For example, in Figure 3.26 the code '00H' in the *Alignment/format info* field in the entry for the opcode JNE implies that routine R2 should use the instruction format with self-relative displacement. If the code 'FFH' were to be used, it would imply that all instruction formats are to be supported, so the routine must decide which machine opcode to use.

(a) Mnemonics table (MOT)

Mnemonic opcode (6)	*Machine opcode* (2)	*Alignment/ format info* (1)	*Routine id* (4)
JNE	75H	OOH	R2

(b) Symbol table (Symtab)

(8)	Symbol
(1)	Type/Defined?/Segment name?/EQU?
(2)	Offset in segment
(2)	Owner segment (SYMTAB entry #)
(2)	Size
(2)	Length
(2)	Source stmt #
(2)	Pointer to FRT
(2)	Pointer to first CRT entry
(2)	Pointer to last CRT entry

(c) Segment Register Table Array (SRTAB_ARRAY)

Segment Register (1)	*SYMTAB entry #* (2)	
00(ES)	23	SRTAB #1
:		
		SRTAB #2

(d) Forward Reference table (FRT)

Pointer (2)	*SRTAB #* (1)	*Instruction address* (2)	*Usage code* (1)	*Source stmt #* (2)

(e) Cross Reference table (CRT)

Pointer (2)	*Source Stmt #* (2)

4

4m

Q.9 a. What is the difference between Compiler and Interpreter? Define static as well as dynamic memory allocation. (4+4)

Answer:

S.No	Compiler	Interpreter
1	Compiler Takes **Entire** program as input	Interpreter Takes **Single** instruction as input
2	Intermediate Object Code is **Gen erated**	**No** Intermediate Object Code is **Generated**
3	Conditional Control Statements are Executes **faster**	Conditional Control Statements are Executes **slower**
4	**Memory Requirement** : **More** (Since Object Code is Generated)	**Memory Requirement** is **Less**
5	Program need not be **compiled** every time	Every time higher level program is converted into lower level program
6	**Errors** are displayed after **entire program** is checked	**Errors** are displayed for **every instruction** interpreted (if any)
7.	**Example** : C Compiler	**Example** : BASIC

Static memory allocation refers to the process of allocating memory at compile-time before the associated program is executed, unlike dynamic memory allocation or automatic memory allocation where memory is allocated as required at run-time.

An application of this technique involves a program module (e.g. function or subroutine) declaring static data locally, such that these data are inaccessible in other modules unless references to it are passed as parameters or returned. A single copy of static data is retained and accessible through many calls to the function in which it is declared. Static memory allocation therefore has the advantage of modularizing data within a program design in the situation where these data must be retained through the runtime of the program.

Dynamic memory allocation is when an executing program requests that the operating system give it a block of main memory. The program then uses this memory for some purpose. Usually the purpose is to add a node to a data structure. In object oriented languages, dynamic memory allocation is used to get the memory for a new object.

The memory comes from above the static part of the data segment. Programs may request memory and may also return previously dynamically allocated memory. Memory may be returned whenever it is no longer needed. Memory can be returned in any order without any relation to the order in which it was allocated. The heap may develop "holes" where previously allocated memory has been returned between blocks of memory still in use. A new dynamic request for memory might return a range of addresses out of one of the holes. But it might not use up all the hole, so further dynamic requests might be satisfied out of the original hole.

Examples of functions in C for dynamic memory allocation are calloc, malloc etc.

b. Explain various parameter passing techniques. **(8)**

Answer: Refer page 123 of Text Book-I

<u>TEXT BOOK</u>

I. Systems Programming and Operating Systems, D. M. Dhamdhere, Tata McGraw-Hill, Second Revised Edition, 2005