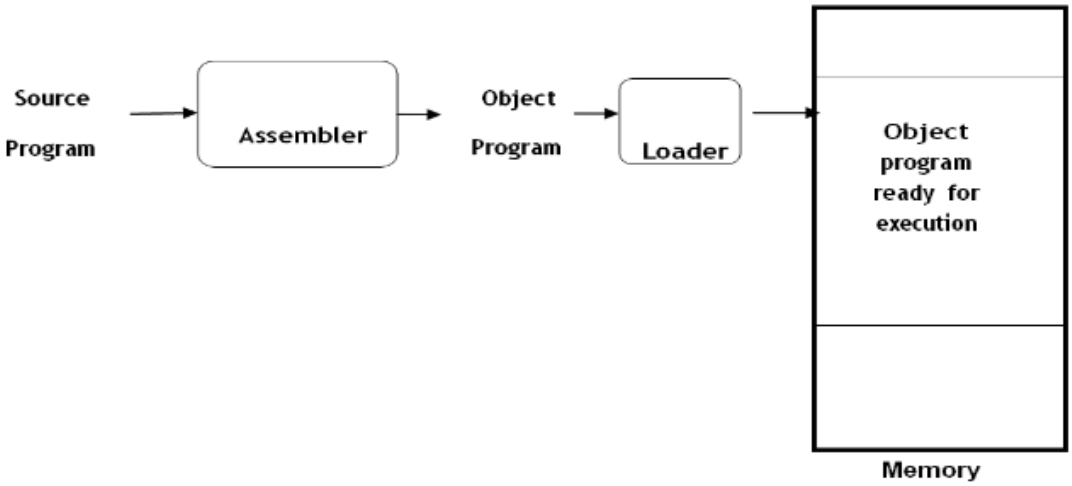
SS Module IV Important Questions

1. What are the basic loader function

- Loader is a system software
- It performs loading function
- It brings object program into memory and starts its execution



-
- Source program is converted to object program using assembler
- This object program is loaded into memory

Types of Loaders

- Absolute
- Bootstrap
- Relocating Loader
- Linking Loader

2. Write absolute loader algorithm

- Object code is loaded to specified locations in the memory
- At the end, Loader jumps to the specified address to begin execution of loaded program.

Algorithm for Absolute loader

- Read header record
- Verify the program name and its length

    - To verify the correct program is presented for loading
- Then we read the first Text record
- Loop until the record type ! = E
    - if object code is in character form
        - Convert it to internal representation
    - Move the object code to specified location in memory
    - Read next Program record
- Jump to address specified in record

```
Begin
    read Header record
    verify program name and length
    read first Text record
    while record type is != '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
```

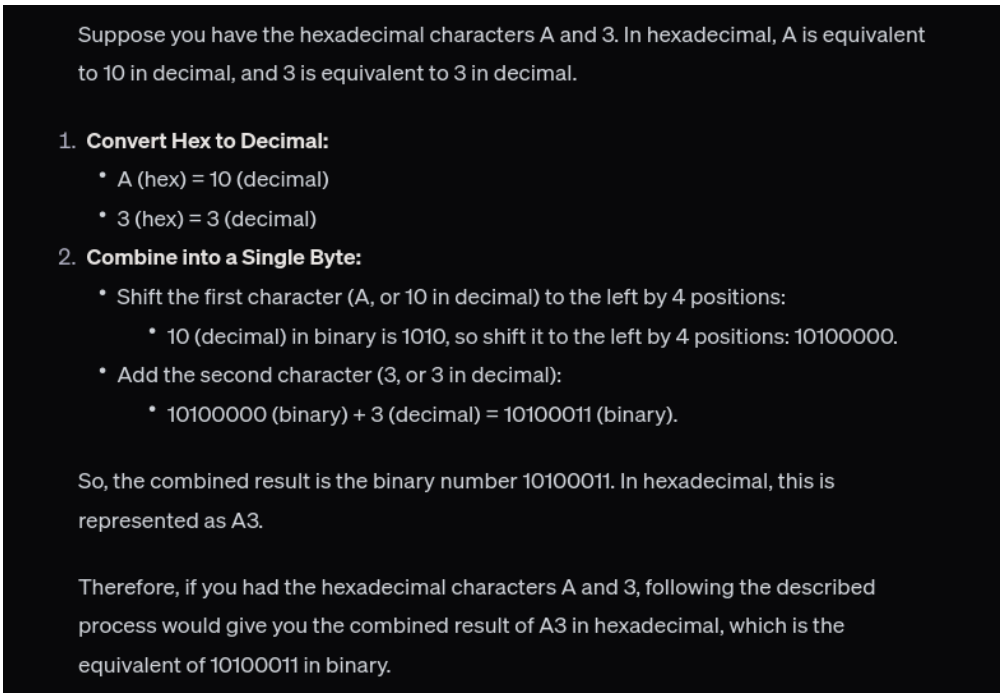
3. Write the algorithm for bootstrap loader

- When a computer is first turned on or restarted, a special type of absolut loader, called bootstrap loader is executed
- This bootstrap loads the first program to be run by the computer

    - usually an operating system
- The bootstrap itself begins at address 0.
- It loads the OS starting address 80

Working

- Consider the bootstrap loader for SIC/XE. The bootstrap loader begins at address 0 in the memory. It loads the OS starting at address 80
- Each byte of object code to be loaded is represented on device F1 as two hexa decimal digits(Text record)
- Object code is loaded to consecutive memory locations starting at address 80.
- After all the object code from device F1 has been loaded the bootstrap jumps to the address 80.
- GETC subroutine
    - This subroutine reads one character from device F1 and converts from ASCII to hex
    - Conversion from ASCII to hex
        - This is done by subtracting 48 if the character is from 0 to 9.
        - For characters A to F subtract 55.
    - Subroutine jumps to address 80 when end of line is reached
- Main loop of the bootstrap loader
    - This is like the brain of bootstrapping
    - There is something called Register X
        - Which is a temporary storage space
        - This register keeps track of the memory location where the next set of instructions or data needs to be loaded
    - Now, there's a function called "GETC." Think of it like a command that says, "Hey, go to this specific place (device F1) and grab a pair of characters.
        - These characters are in hexadecimal
    - So, you grab these two hexadecimal characters
        - These hexadecimal characters are of 4 bits each
        - To combine them into a single byte (which is a unit of digital information),
            - 4+4 = 8 = 1 byte
        - you shift the first character to the left by 4 positions
            - Then add the second character to it.
            - Example

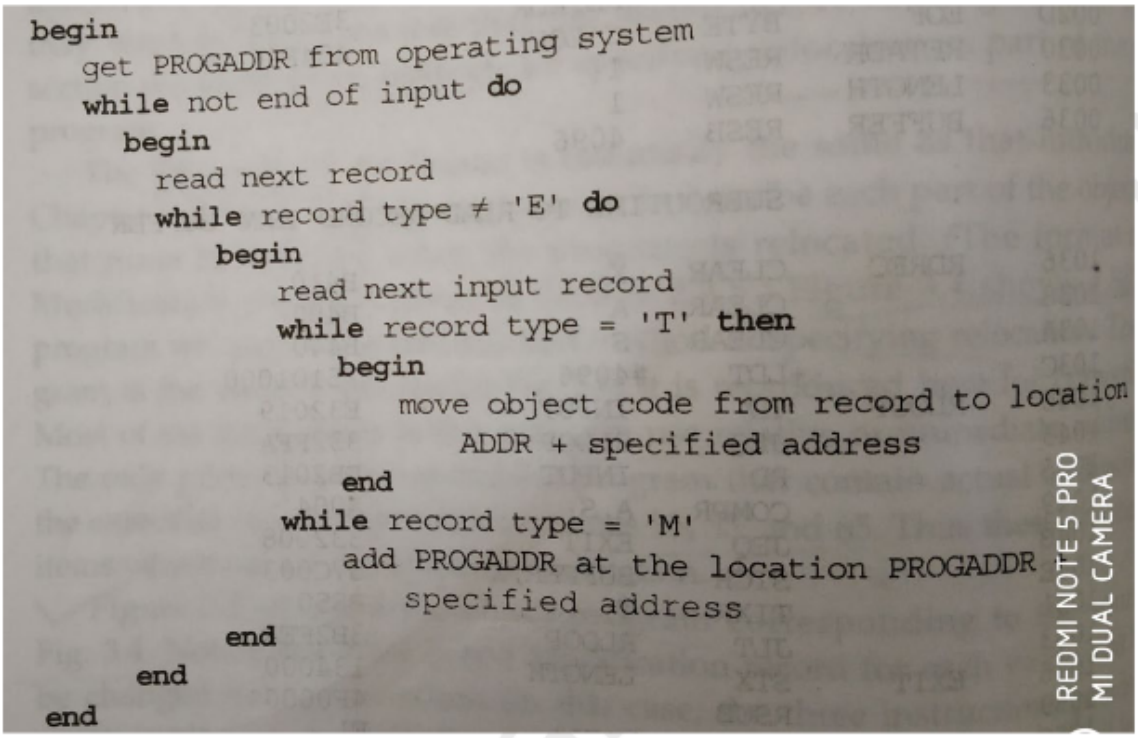
- 

    - This combination gives you a complete byte of information.
    - this resulting byte is stored in the memory of the computer, and specifically, it goes to the memory location that is currently stored in register X

4. Write the algorithm for relocating loader

- Program that allow program relocation are called relocating loaders
- There are two methods for relocation
    - Modification record
    - Bit masking
- A modification record is used to describe each part of the object code that must be changed when the program is relocated
- Consider SIC/XE programs, Most of the instructions in this program uses relative or immediate addressing. So modification not required. Only format 4 instructions require modification
- Each modification record specifies the starting address and length of the field to be modified and what modification to be performed.(adding the start address).

Algorithm for SIC/XE relocation loader



```
begin
   get PROGADDR from operating system
   while not end of input do
     begin
       read next record
       while record type ≠ 'E' do
           begin
               read next input record
               while record type = 'T' then
                   begin
                       move object code from record to location
                          ADDR + specified address
                   end
               while record type = 'M'
                   add PROGADDR at the location PROGADDR
                       specified address
           end
     end
end
```


5. Explain bitmasking

- In SIC program relative addressing is not used. So every instruction needs modification. We can not write modification records for all instructions
- So relocation bits are used. Each instruction object code is associated with relocation bit.

- If the relocation bit is 1 program starting address is to be added to this word.

  FFC= 111111111100

SIC relocation loader algorithm

```
begin
   get PROGADDR from operating system
   while not end of input do
     begin
       read next record
       while record type ≠ 'E' do
       while record type = 'T'
          begin
             get length = second data
             mask bits(M) as third data
                 For(i = 0, i < length, i++)
                       if Mi = 1 then
                          add PROGADDR at the location PROGADDR + specified
                              address
                       else
                          move object code from record to location PROGADDR +
                          specified address
             read next record
          end
     end
end
```

6. Write the algorithm for pass1 of a linking loader

- During the first pass the loader is concerned only with Header and Define record types in the control sections.
- The beginning load address for the linked program(PROGADDR) is obtained from OS. This becomes the starting address for the first control section(CSADDR)
- The control section name is entered into ESTAB with value given by CSADDR.
- All external symbols appearing in the define record for the control section are also entered into ESTAB.
- Their addresses are obtained by adding the value specified in the Define record to CSADDR
- When the END record is read the control section length CSLTH which was saved from the Header record is added to CSADDR. This gives the starting address for the next control section
- At the end of pass1 , ESTAB contains all external symbols defined in the control sections together with addresses assigned to each

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}
```

Figure 3.11(a) Algorithm for Pass 1 of a linking loader.

7. Write the algorithm for pass2 of a linking loader

- Performs the actual loading, relocation and linking of the program.
- CSADDR holds the starting address of the control section currently being loaded.
- As each Text record is read , the object code is moved to the specified address (plus the current value of the CSADDR).
- When a modification record is encountered , the symbol whose value is to be used for modification is looked up in ESTAB. This value is then added to or subtracted from the indicated location in memory.
- The last step performed by the loader is transferring of control to the loaded program to begin execution.

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}
```

Figure 3.11(b) Algorithm for Pass 2 of a linking loader.

8. Explain automatic library search

- This feature allows a programmer to use standard subroutines without explicitly including them in the program to be loaded. The routines are automatically retrieved from library as they are needed during linking
- The programmer has to only give the subroutine name in the external reference. The routine will be automatically fetched from the library and linked with the main program
- Working:
  - Enter symbols from Refer record into the symbol table(ESTAB) . When the definition is encountered the address is assigned to the symbol.
  - At the end of pass the symbols in ESTAB remain undefined represent unresolved external references . The loader searches the library for the routines and process the subroutines as if they are part of the input stream
- A special file structure is used for libraries. This is known as directory.

    - This contains the name of the subroutine and a pointer to its address within the file.

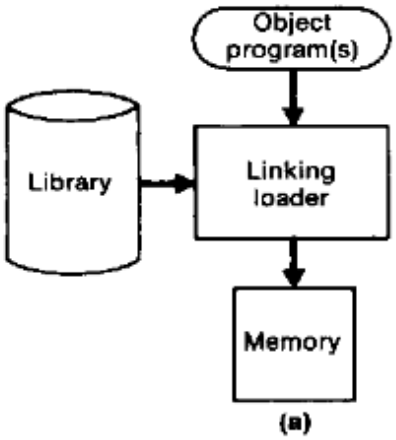
9. Explain loader options

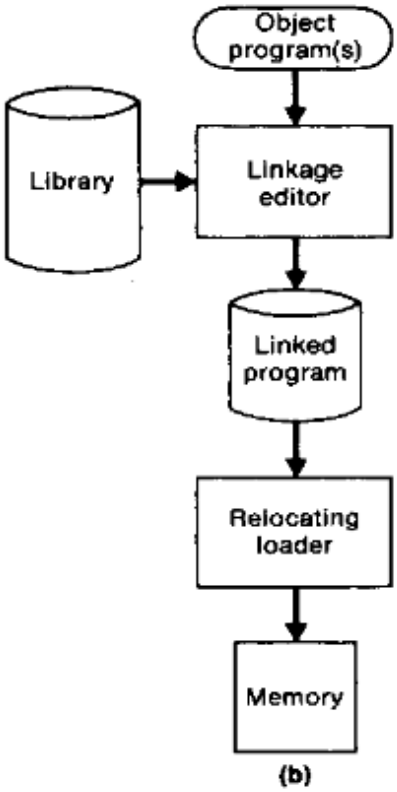
- Some of the loader options are
    - Selection of alternative sources of input
        - Direct the loader to read designated object program and treat it as primary loader input
    - Command to delete external symbols or entire control section
    - CHANGE name1, name2
        - This command causes the external symbol name1 to be changed to name2 wherever it appears in the object program
    - Option to avoid some reference

10. Differentiate between linking loader and linkage editor.

- Linking Loader
    - Performs all linking and relocation operations
        - Loads the linked program directly into the program for execution
    - Resolution of external reference every time program is executed
    - More than one pass required
- Linkage editor
    - Produces a linked version of the program called load module
    - Resolution of external reference done only once
    - Can be accomplished in one pass

Linking loader	linkage editor
1. Performs all linking and relocation operations and loads the linked program directly into memory for execution	1. Produces a linked version of the program called load module which is written to a file for later execution
2. A linking loader searches the library and resolves external references every time the program is executed.	2. Resolution of external references and library searching are only performed once.
3. More than one pass required.	3. The loading can be accomplished in one pass and no external symbol table required, much less overhead than a linking loader.





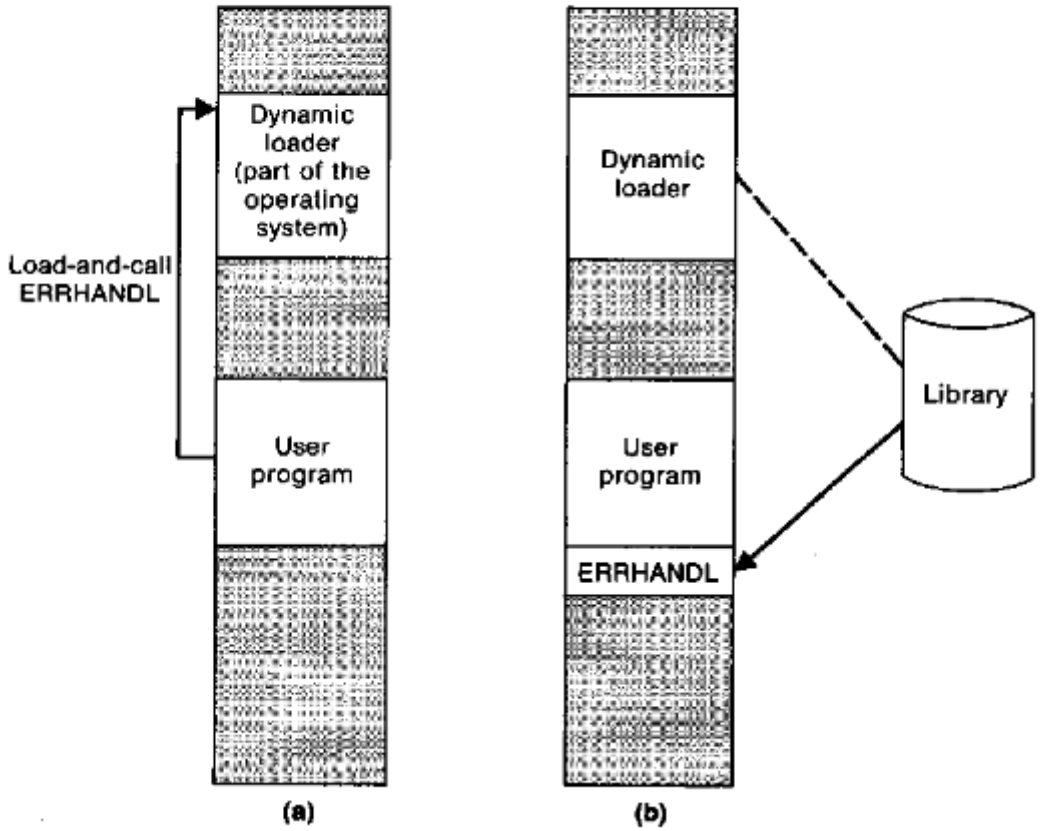
Advantages

- build packages of subroutines that are generally used togethe
- Linkage editors can also allow the user to specify that external references are not to be resolved by automatic library search.

11. Explain dynamic Linking with example

- In dynamic linking the linking function is done at execution time.
- That is a subroutine is loaded and linked to the rest of the program when it is first called
- Dynamic linking provide the ability to load the routines only when they are required
    - Example: consider the subroutine which diagnose the error in input data during execution. If such errors are rare these subroutines need not be used

Loading and calling a subroutine via dynamic linking



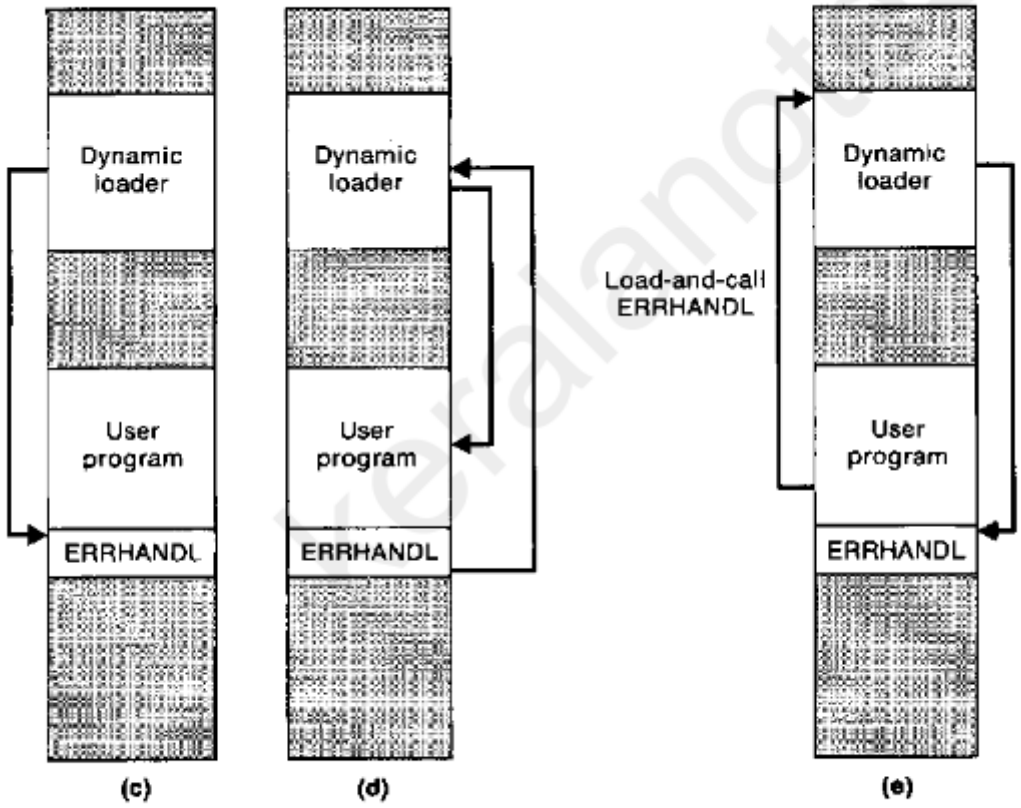


Figure 3.14 Loading and calling of a subroutine using dynamic linking.

12. Machine dependent and machine independent

Machine dependent

- Relocation
- Program Linking
-

Machine independent

- Automatic library search
- Loader Options