Module 4, A simple bootstrap loader

• When a computer is first turned on or restarted, a special type of absolute 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 0x80.
• No header record or control information, the object code is consecutive bytes of memory.
• The object code from device F1 is loaded into consecutive bytes of memory starting at address 80.
• After the entire object code is loaded the bootstrap jumps to address 80 and begin the execution of the OS program.

Algorithm

Begin
X=0x80 (the address of the next memory location to be loaded Loop
A¬GETC (and convert it from the ASCII character code to the value of the hexadecimal digit) save the value in the high-order 4 bits of S
A¬GETC combine the value to form one byte A¬(A+S) store the value (in A) to the address in register X X¬X+1
End

Dynamic Linking:

It is also called as dynamic loading or load on call.
• It is a scheme that postpones the linking function until execution time.
• Subroutine is loaded and linked to the rest of the program when it is first called.

Applications of Dynamic Linking:

It is used to allow several executing programs to share one copy of a subroutine or library.

Bootstrap Loaders:

• When the computer is just switched on, the machine is empty and has no program in the memory, so program relocation is not necessary.
• We can simply specify the absolute address of the program, which will be first, loaded usually an operating system

Linkage Editor

• The difference between a linkage editor and a linking loader:
– A linking loader performs all linking and relocation operations, including automatic library search, and loads the linked program into memory for execution.
– A linkage editor produces a linked version of the program, which is normally written to a file for later execution.
• When the user is ready to run the linked program, a simple relocating loader can be used to load the program into memory.
• The only object code modification necessary is the addition of an actual address to relative values within the program.
• The linkage editor performs relocation of all control sections relative to the start of the linked program.
• All items that need to be modified at load time have values that are relative to the start of the linked program.
• This means that the loading can be accomplished in one pass with no external symbol table required.
• Thus, if a program is to be executed many times without being reassembled, the use of a linkage editor can substantially reduces the overhead required.

Dynamic Linking

• Linkage editors perform linking before the program is loaded for execution.
• Linking loaders perform these same operations at load time.
• Dynamic linking postpones the linking function until execution time.
– A subroutine is loaded and linked to the test of the program when it is first called

Application

• Dynamic linking is often used to allow several executing programs to share one copy of a subroutine or library.
• For example, a single copy of the standard C library can be loaded into memory.
• All C programs currently in execution can be linked to this one copy, instead of linking a separate copy into each object program.
• In an object-oriented system, dynamic linking is often used for references to software object.
• This allows the implementation of the object and its method to be determined at the time the program is run. (e.g., C++)

• The implementation can be changed at any time, without affecting the program that makes use of the object

Advantage • The subroutines that diagnose errors may never need to be called at all. • However, without using dynamic linking, these subroutines must be loaded and linked every time the program is run. • Using dynamic linking can save both space for storing the object program on disk and in memory, and time for loading the bigger object program.

Implementation

• A subroutine that is to be dynamically loaded must be called via an operating system service request.

– This method can also be thought of as a request to a part of the loader that is kept in memory during execution of the program

• Instead of executing a JSUB instruction to an external symbol, the program makes a load-andcall service request to the OS.

• The parameter of this request is the symbolic name of the routine to be called..

• The OS examines its internal tables to determines whether the subroutine is already loaded.

• If needed, the subroutine is loaded from the library.

• Then control is passed from the OS to the subroutine being called.

• When the called subroutine completes its processing, it returns to its caller (operating system).

• The OS then returns control to the program that issues the request.

• After the subroutine is completed, the memory that was allocated to it may be released.

Linking & Loading, Introduction

The Source Program written in assembly language or high level language will be converted to object program, which is in the machine language form for execution. This conversion either from assembler or from compiler, contains translated instructions and data values from the source program, or specifies addresses in primary memory where these items are to be loaded for execution. This contains the following three processes, and they are,

Loading - which allocates memory location and brings the object program into memory for execution - (Loader)

Linking- which combines two or more separate object programs and supplies the information needed to allow references between them - (Linker)

Relocation - which modifies the object program so that it can be loaded at an address different from the location originally specified - (Linking Loader)

Basic Loader Functions

A loader is a system program that performs the loading function. It brings object program into memory and starts its execution

Machine-Dependent Loader Features

Absolute loader is simple and efficient, but the scheme has potential disadvantages One of the most disadvantage is the programmer has to specify the actual starting address, from where the program to be loaded. This does not create difficulty, if one program to run, but not for several programs. Further it is difficult to use subroutine libraries efficiently. This needs the design and implementation of a more complex loader. The loader must provide program relocation and linking, as well as simple loading functions.

Relocation The concept of program relocation is, the execution of the object program using any part of the available and sufficient memory. The object program is loaded into memory wherever there is room for it. The actual starting address of the object program is not known until load time. Relocation provides the efficient sharing of the machine with larger memory and when several independent programs are to be run together. It also supports the use of subroutine libraries efficiently. Loaders that allow for program relocation are called relocating loaders or relative loaders.

Loader Design Options There are some common alternatives for organizing the loading functions, including relocation and linking. Linking Loaders – Perform all linking and relocation at load time. The Other Alternatives are Linkage editors, which perform linking prior to load time and, Dynamic linking, in which linking function is performed at execution time