UNIT 1 ASSEMBLER, MACRO, LOADER & LINKER

Assembler: Overview of the assembly process - Design of two pass assembler-Macros: Macro definition and usage- schematics for macro expansion - Design of a Macro pre- processor-Design of a Macro assembler; Introduction to Loaders and Linkers.

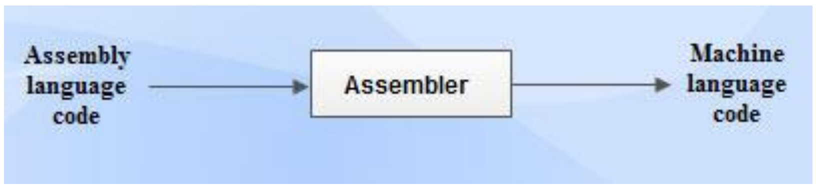
- ✓ Normally the C's program building process involves four stages and utilizes different 'tools' such as a preprocessor, compiler, assembler, and linker.

- ➢ **Preprocessing** is the first pass of any C compilation. It processes include-files, conditional compilation instructions and macros.
- ➢ **Compilation** is the second pass. It takes the output of the preprocessor, and the source code, and generates assembler source code.
- ➢ **Assembly** is the third stage of compilation. It takes the assembly source code and produces an assembly listing with offsets. The assembler output is stored in an object file.
- ➢ **Linking** is the final stage of compilation. It takes one or more object files or libraries as input and combines them to produce a single (usually executable) file. In doing so, it resolves references to external symbols, assigns final addresses to procedures/functions and variables, and revises code and data to reflect new addresses (a process called relocation).

Assembler:

- ✓ An assembler is a program that translates each instruction to its binary machine code equivalent.
- ✓ It is a relatively simple program.
- ✓ There is a one-to-one or near one-to-one correspondence between assembly language instructions and machine language instructions.

The Assembly Process:



• The design of assembler can be to perform the following: –

- ➢ Scanning (tokenizing)
- ➢ Parsing (validating the instructions)
- ➢ Creating the symbol table

- ➢ Resolving the forward references
- ➢ Converting into the machine language.

Assembler Design can be done in:

- ➢ One Pass Assembler
- ➢ Two Pass Assembler

Single Pass Assembler:

- ➢ Does everything in single pass
- ➢ Cannot resolve the forward referencing

- ✓ A single pass assembler scans the program only once and creates the equivalent binary program.
- ✓ The assembler substitute all of the symbolic instruction with machine code in one pass.
- ✓ Advantages every source statement needs to be processed once.
- ✓ Disadvantages we cannot use any forward reference in our program. Forward Reference Forward reference means; reference to an instruction which has not yet been encountered by the assembler.
- ✓ In order to handle forward reference, the program needs to be scanned twice. In other words a two pass assembler is needed.

Two Pass Assembler:

- ✓ An assembler is a translator, that translates an assembler program into a conventional machine language program.
- ✓ Basically, the assembler goes through the program one line at a time, and generates machine code for that instruction. Then the assembler proceeds to the next instruction.
- ✓ In this way, the entire machine code program is created. For most instructions this process works fine, for example for instructions that only reference registers, the assembler can compute the machine code easily, since the assembler knows where the registers are.

Consider an assembler instruction like the following

```
      JMP  LATER
      ...
      ...
LATER:
```

- ✓ This is known as a forward reference. If the assembler is processing the file one line at a time, then it doesn't know where LATER is when it first encounters the jump instruction.
- ✓ So, it doesn't know if the jump is a short jump, a near jump or a far jump. There is a large difference amongst these instructions. They are 2, 3, and 5 bytes long respectively. The assembler would have to guess how far away the instruction is in order to generate the correct instruction.
- ✓ If the assembler guesses wrong, then the addresses for all other labels later in the program would be wrong, and the code would have to be regenerated. Or, the assembler could always choose the worst case.
- ✓ But this would mean generating inefficiency in the program, since all jumps would be considered far jumps and would be 5 bytes long, where actually most jumps are short jumps, which are only 2 bytes long.

Macros:

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

CLASSIFICATION OF MACROS:

Lexical expansion:

- ✓ Lexical expansion implies replacement of a character string by another character string during program generation.
- ✓ Lexical expansion is to replace occurrences of formal parameters by corresponding actual parameters.

Semantic expansion:

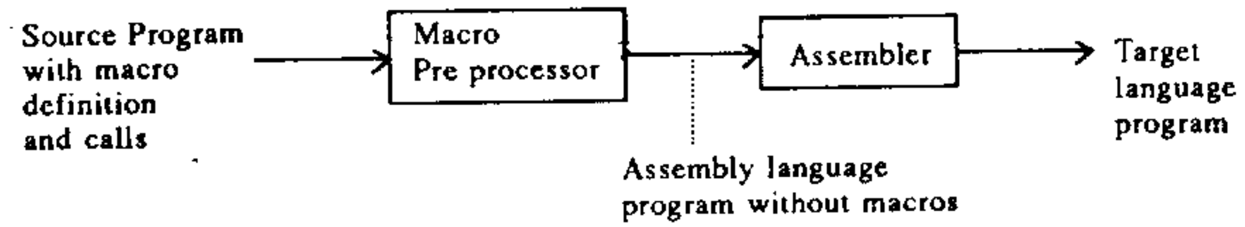
- ✓ Semantic expansion implies generation of instructions tailored to the requirements of a specific usage.
- ✓ Semantic expansion is characterized by the fact that different uses of a macro can lead to codes which differ in the number, sequence and opcodes of instructions.

Schematics for macro expansion:

- ✓ The use of a macro name with a set of actual parameters is replaced by some code generated from its body is called macro expansion.
- ✓

Design of Macro Pre-processor:+

- ✓ The source program containing macro definitions and calls is translated into an assembly language, program without any macro definitions or calls.
- ✓ This program form can now be handed over to a conventional assembler as to obtain the target languages form of the program.


Source Program with macro definition and calls
Macro Pre processor
Assembler
Target language program
Assembly language program without macros

  - ✓ The process of macro expansion is completely segregated from the process of assembly program.
  - ✓ The translator which performs macro expansion in this manner is called a macro pre-processor.
  - ✓ The advantage of this scheme is that any existing conventional assembler can be enhanced in this manner to incorporate macro processing.
  - ✓ It would reduce the programming cost involved in making a macro facility available to programmer using a computer system.
  - ✓ The disadvantage is that this scheme is probably not very efficient because of the time spent in generating assembly language statements and processing them again for the purpose of translation to the target language.

Design of a Macro assembler:

- ✓ A program that translates assembly language instructions into machine code and which the programmer can use to define macro instructions.
  - ➢ **Comments**
  - ➢ **Labels**
  - ➢ **Addressing modes**
  - ➢ **Arithmetic Expressions**

Comments:

- ✓ Any texts after all operands for a given mnemonic have been processed.
- ✓ A line beginning with * (in the first column) up to the end of the line.
- ✓ An empty line.

Labels:

- ✓ The Assembler has the facility to generate symbolic labels during assembly process.

Addressing Modes:

- ✓ The Assembler will iden3fy what addressing mode each instruc3on is in, and assigns the appropriate opcode.

Arithmetic Expressions:

- ✓ The Motorola assembler supports several arithme3c opera3ons which can be used to form values of labels or instruction arguments.
    - ➢ Addition +
    - ➢ Subtraction –
    - ➢ Multiplication *
    - ➢ Division /
    - ➢ Remainder after division %
    - ➢ Bitwise AND &
    - ➢ Bitwise OR |
    - ➢ Bitwise XOR ^

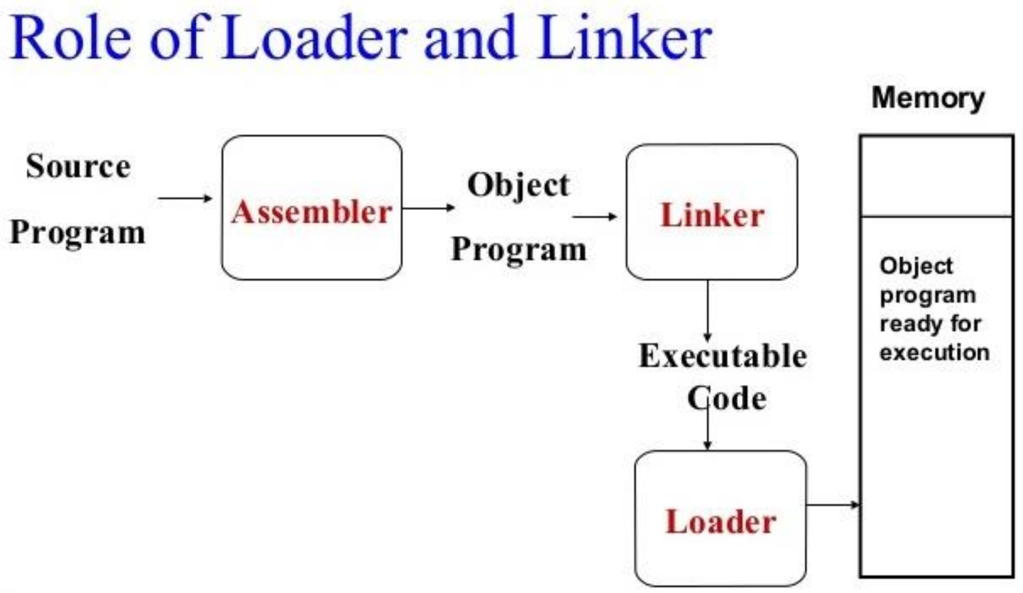
Introduction to Loaders and Linkers:

Loader:

- ✓ It is a SYSTEM PROGRAM that brings an executable file residing on disk into memory and starts it running.

STEPS:

- ➢ Read executable file's header to determine the size of text and data segments.
- ➢ Create a new address space for the program.
- ➢ Copies instructions and data into address space.
- ➢ Copies arguments passed to the program on the stack.
- ➢ Initializes the machine registers including the stack pointer.
- ➢ Jumps to a startup routine that copies the program's arguments from the stack to registers and calls the program's main routine.



Types of Loaders:

- ➢ Absolute Loader
- ➢ Bootstrap Loader
- ➢ Relocating Loader
- ➢ Direct Linking loader

Absolute Loader:

- ✓ The operation of absolute loader is very simple.
- ✓ The object code is loaded to specified locations in the memory.
- ✓ At the end the loader jumps to the specified address to begin execution of the loaded program.

Advantage of absolute loader

- ➢ is simple and efficient.

Disadvantages

- ➢ need for programmer to specify the actual address, and, difficult to use subroutine libraries.

Bootstrap Loader:

- ✓ When a computer is first tuned 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.

Relocating Loader:

- ✓ 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.

Direct Linking Loader:

- ✓ The scheme that postpones the linking functions until execution.
- ✓ A subroutine is loaded and linked to the rest of the program when it is first called – usually called dynamic linking, dynamic loading or load on call.
- ✓ The advantages of dynamic linking are, it allow several executing programs to share one copy of a subroutine or library.
- ✓ In an object oriented system, dynamic linking makes it possible for one object to be shared by several programs.
- ✓ Dynamic linking provides the ability to load the routines only when (and if) they are needed. The actual loading and linking can be accomplished using operating system service request.
- ✓ The loader cannot have the direct access to the source code.
- ✓ To place the object code 2 types of addresses can be used:-

  ABSOLUTE:

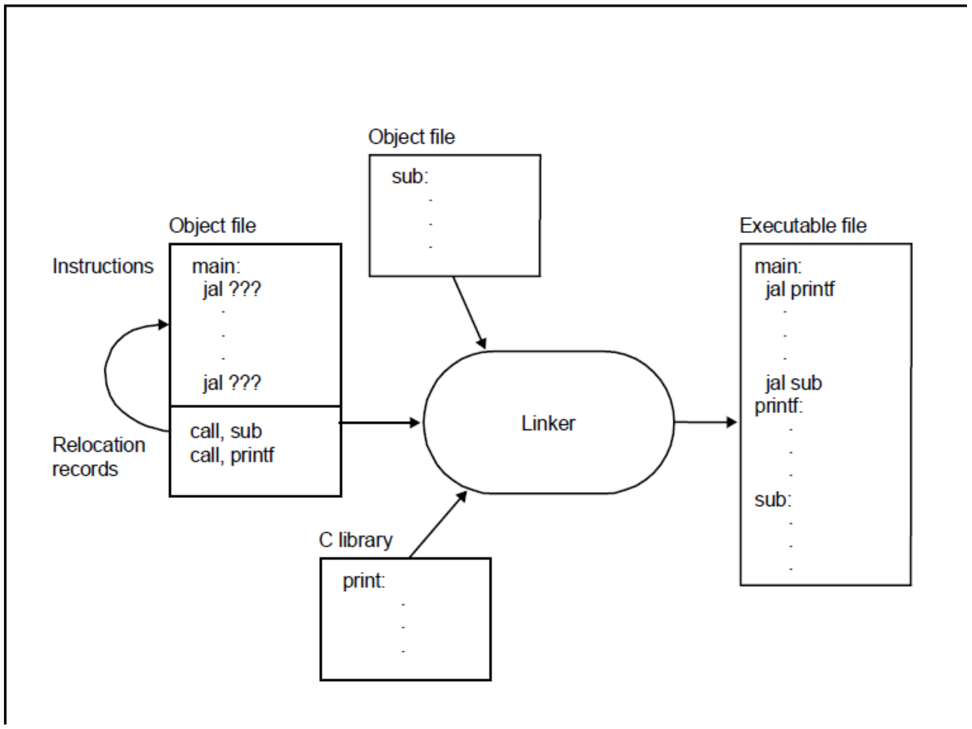
  - ➢ In this the absolute path of object code is known and the code is directly loaded in memory.

  RELATIVE:

  - ➢ In this the relative path is known and this relative path is given by assembler.

Linker:

- ✓ Tool that merges the object files produced by separate compilation or assembly and creates an executable file.



- ✓ Computer programs typically comprise several parts or modules; all these parts/modules need not be contained within a single object file, and in such case refer to each other by means of symbols.
- ✓ **Types of Linker**
    - ➢ **Static Linking**
    - ➢ **Dynamic Linking**

Static Linking:

- ✓ Static linking occurs when a calling program is linked to a called program in a single executable module.
- ✓ When the program is loaded, the operating system places into memory a single file that contains the executable code and data.

Advantage:

- ✓ Static linking is that you can create self-contained, independent programs.
- ✓ In other words, the executable program consists of one part (the .EXE file) that you need to keep track of.

Disadvantages:

- ✓ You cannot change the behavior of executable files without relinking them.

Dynamic Linking:

- ✓ Many operating system environments allow dynamic linking, that is the postponing of the resolving of some undefined symbols until a program is run.
- ✓ That means that the executable code still contains undefined symbols, plus a list of objects or libraries that will provide definitions for these.
- ✓ Loading the program will load these objects/libraries as well, and perform a final linking.

Advantages:

- ✓ Often-used libraries (for example the standard system libraries) need to be stored in only one location, not duplicated in every single binary.

Disadvantages:

- ✓ Known on the Windows platform as "DLL Hell", an incompatible updated DLL will break executables that depended on the behavior of the previous DLL.