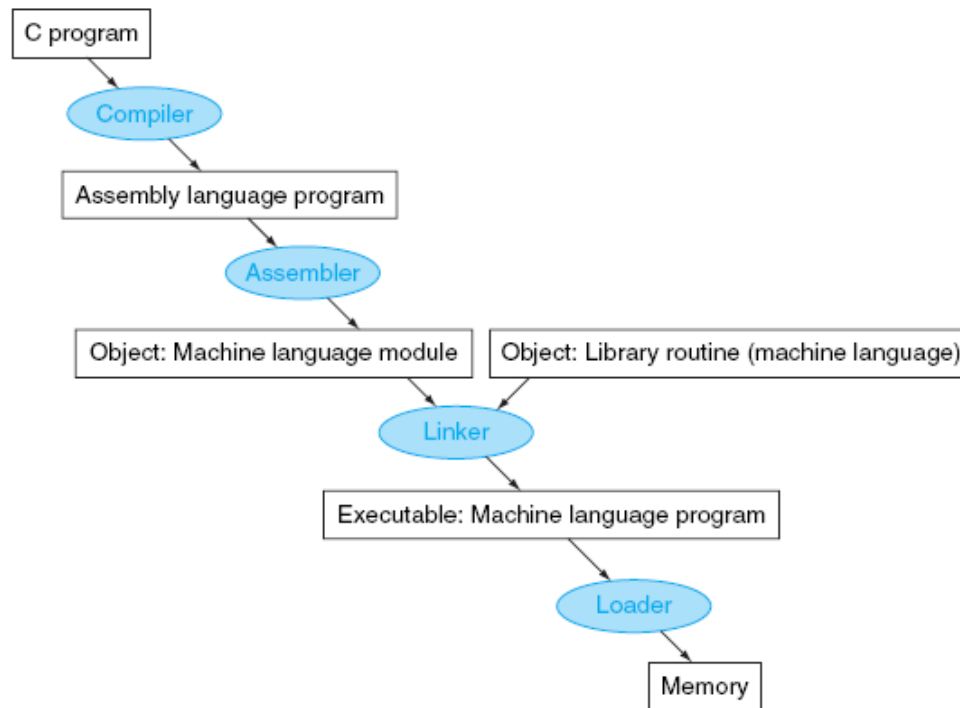


# Lecture 15

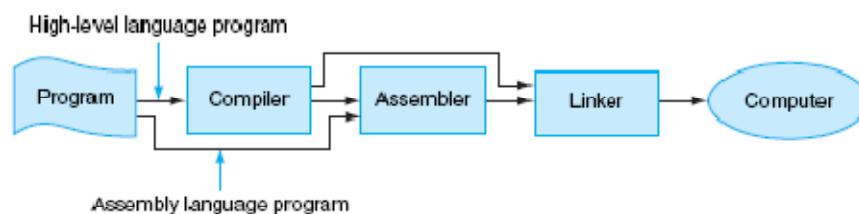
## Linkers, Loaders & Virtual Memory

- Steps to compiling and running a program:



**FIGURE 2.21 A translation hierarchy for C.** A high-level language program is first compiled into an assembly language program and then assembled into an object module in machine language. The linker combines multiple modules with library routines to resolve all references. The loader then places the machine code into the proper memory locations for execution by the processor. To speed up the translation process, some steps are skipped or combined. Some compilers produce object modules directly, and some systems use linking loaders that perform the last two steps. To identify the type of file, UNIX follows a suffix convention for files: C source files are named `x.c`, assembly files are `x.s`, object files are named `x.o`, statically linked library routines are `x.a`, dynamically linked library routines are `x.so`, and executable files by default are called `a.out`. MS-DOS uses the suffixes `.C`, `.ASM`, `.OBJ`, `.LIB`, `.DLL`, and `.EXE` to the same effect. Copyright © 2009 Elsevier, Inc. All rights reserved.

- Assembly is sometimes replaced by an intermediate form.



**FIGURE B.1.6 Assembly language either is written by a programmer or is the output of a compiler.** Copyright © 2009 Elsevier, Inc. All rights reserved.

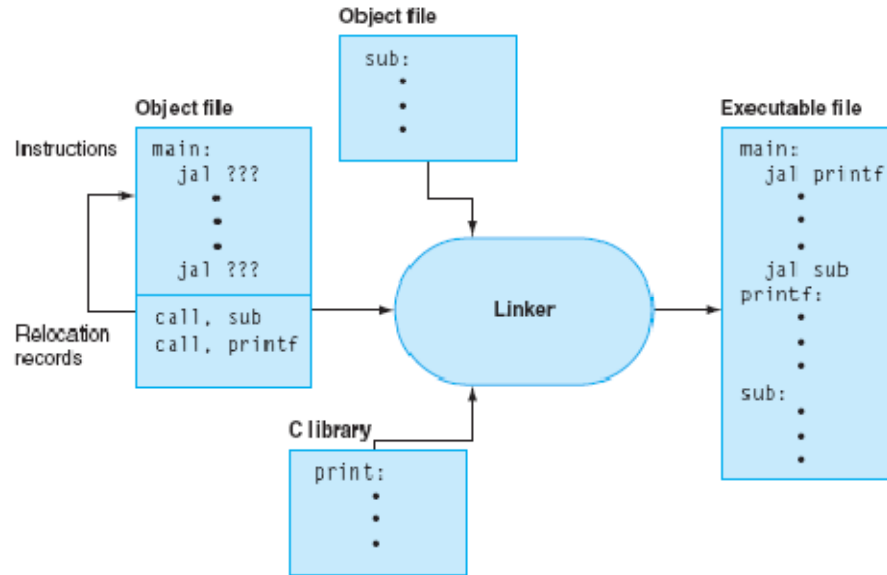
- Object file format:

Object file header	Text segment	Data segment	Relocation information	Symbol table	Debugging information
--------------------	--------------	--------------	------------------------	--------------	-----------------------

**FIGURE B.2.1 Object file.** A UNIX assembler produces an object file with six distinct sections. Copyright © 2009 Elsevier, Inc. All rights reserved.

- **Header:** Describes the size and position of the other pieces of the file.
- **Text segment:** machine code; may contain unresolved references.
- **Data segment:** data; may contain references to labels in other files.
- **Relocation information:** identifies instructions and data words that depend on absolute addresses & may change if program is moved.
- **Symbol Table:** associates addresses with external labels in source files. It lists any unresolved references.
- **Debugging Information:** Concise description of the way which the program was compiled, so the debugger can find which instruction address corresponds to link in the source file and print data structures in readable form.

- Linker (aka Link Editor):



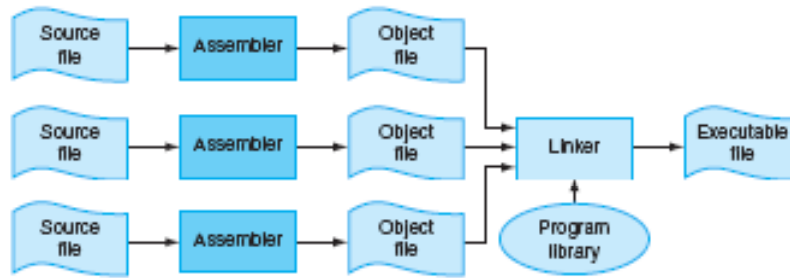
**FIGURE B.3.1** The linker searches a collection of object files and program libraries to find nonlocal routines used in a program, combines them into a single executable file, and resolves references between routines in different files. Copyright © 2009 Elsevier, Inc. All rights reserved.

- **Definition:** combines independently-assembled machine language programs and resolves all undefined labels into an executable file.
- Used to copy parts of the libraries and add them to compiled programs (in this case C).
- Allows for programs to be compiled and assembled in parts, eliminating compile time of all code. Instead, only modified parts of code are compiled and assembled. Standard library routines do not need to be compiled/assembled with each change.
  - **Example:** parts of code in C and assembly.
  - **Example:** large program is written as several, separate modules (groups working on separate parts, classes).
  - Object files are written to disk (possibly at different dates and times) when compiled separately.
- Useful in development stage, when modules and code change frequently.
- Compiling all modules every time wastes time, therefore, compile only those modules that were changed.
- Linking takes very little time compared to compilation.
- 3 steps:
  1. Place code and data modules symbolically in memory.
  2. Determine the addresses of data and instruction labels (branch, jump, data addresses).
  3. Patch both the internal and external references.

- Problem: Individual assembly means that positions of modules in memory relative to each other is unknown when assembled. Therefore, the linker must place the modules in memory, and change all absolute references, to their new relocated position.
- Determining the start address of each module:
  - Modules are placed in memory by the loader one after the other.
  - The start address of each module is dependent on the modules prior to it in memory and the size of the module.
  - Compiler is unaware of where the individual pieces will be relative to each other. Therefore, all modules are compiled with text segment starting at location 0. Same with data segment.
  - Linker then takes care of these addresses when it links the modules together, by placing all modules together in a single stream (all absolute addresses must be handled by the linker).
- Separate compilation of modules leads to another problem:
  1. What if module-1 is using a function declared & defined in module-2?
    - When module-1 is compiled, compiler does not know the address of this function.
    - Only at link time does the correct address become available.
    - Process is known as resolving external references or references among modules (separate files).
- Linking error occurs when linker has performed all matching of undefined labels and some labels still remain undefined.
- Format of file is similar to object file, but no relocation information.
- Example of linking object files (textbook, *pp.* 143 – 145). Go through it.
- Actually, Section 2.12 in the textbook (*pp.* 139 – 148) deals with this whole subject of compiling, linking and loading, including topics not discussed here, such as Dynamically Linked Libraries (*.dll*), and how a Java program is started. You should take a look at it.

- Loader:

- Makes ('loads') copy of executable code in memory and starts the program.



**FIGURE B.1.1** The process that produces an executable file. An assembler translates a file of assembly language into an object file, which is linked with other files and libraries into an executable file. Copyright © 2009 Elsevier, Inc. All rights reserved.

- Steps for loader:

1. Read executable file header to determine size of data and text segments.
2. Create an address space large enough for text and data.
3. Copy instructions and data from executable to memory.
4. Copy the parameters (if any) to the main program onto the stack.
5. Initialize the machine registers (usually to '0') and set the stack pointer to the first free location on stack.
6. Jump to a start-up routine that copies the parameters into the argument registers, and calls the main routine. When the main routine returns, the start-up routine terminates the program with an exit system call.

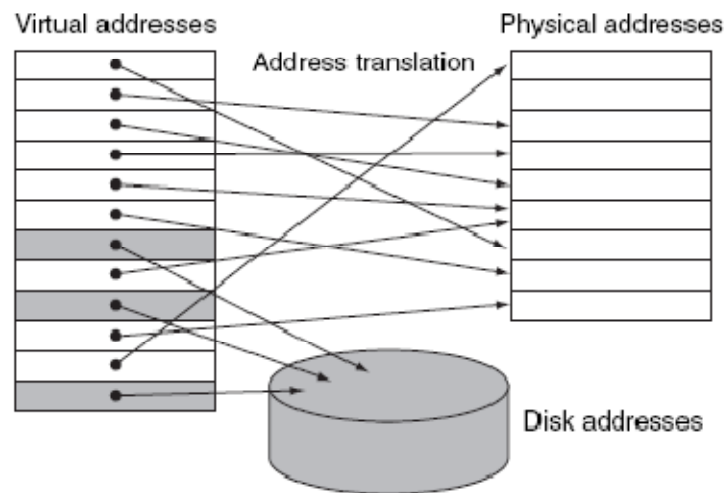
- New address space is typically virtual space for every program.

- In old machines, before virtual memory: to load a program, machine had to find space in memory where the program could fit in. This address could be different from what the linker assumed.
- Ex: Linker may have assumed a start address of 1000. But there is insufficient room there. Loader finds empty space at 3000. Again all addresses must be changed by adding 2000. This was done at load time.

- Virtual Memory:

- O/S responsibility.
- A program can require more memory than the actual physical memory space available.
- Entire program need not be in memory all the time (load parts from disk).
- Use program locality (as we do for caching).
- Only small space is allocated to the program initially (20 – 40kB).
- Required pieces of code are brought in on demand. Pieces not used for a while are discarded.
- Programs are compiled, assembled and linked assuming that they will run in virtual space.
- Each program has its own virtual space, starting at address '0'.
- During execution, all addresses that are generated are virtual addresses.
- Using an address translation table, virtual addresses are changed to the actual physical addresses in memory.

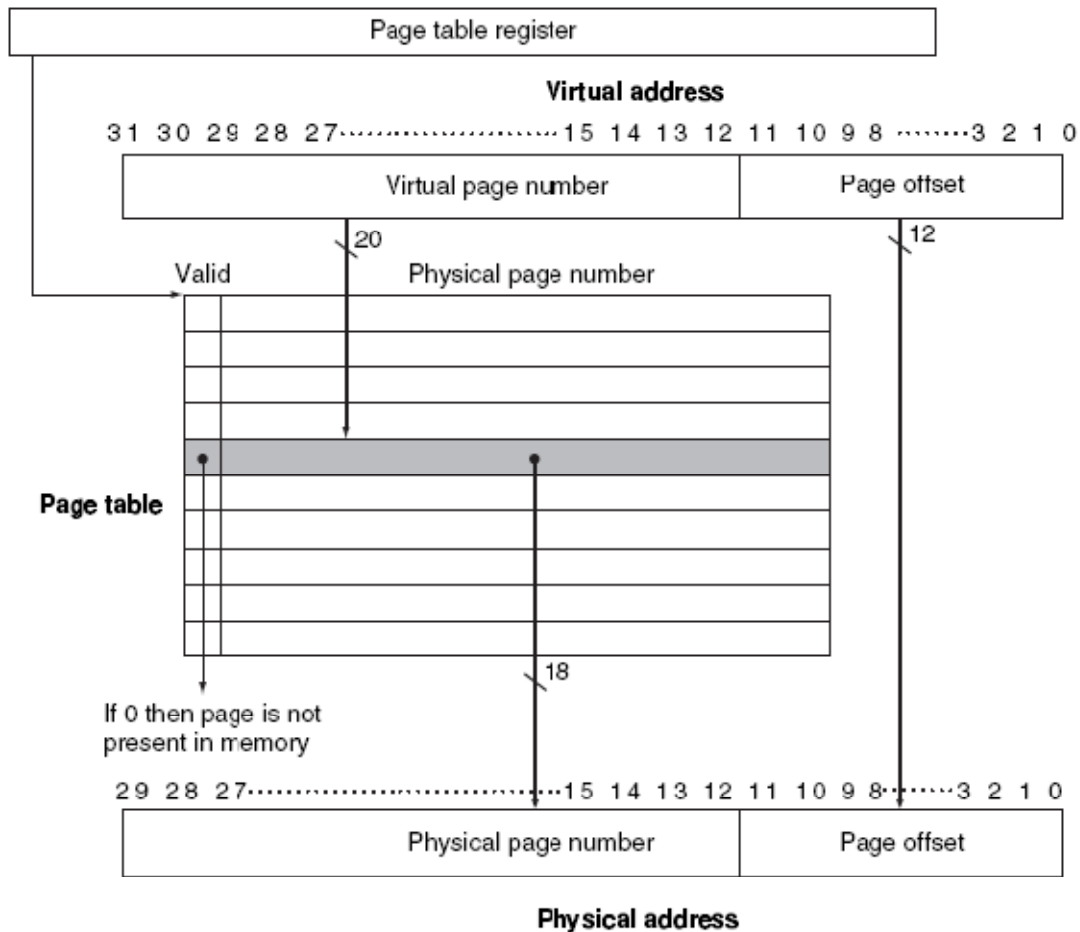
- Ex: `lw $t0, 40 ($t1) # assume 40($t1) is a virtual address`



**FIGURE 5.19** In virtual memory, blocks of memory (called *pages*) are mapped from one set of addresses (called *virtual addresses*) to another set (called *physical addresses*). The processor generates virtual addresses while the memory is accessed using physical addresses. Both the virtual memory and the physical memory are broken into pages, so that a virtual page is mapped to a physical page. Of course, it is also possible for a virtual page to be absent from main memory and not be mapped to a physical address; in that case, the page resides on disk. Physical pages can be shared by having two virtual addresses point to the same physical address. This capability is used to allow two different programs to share data or code. Copyright © 2009 Elsevier, Inc. All rights reserved.

- Contents of the actual (physical) location are then loaded into \$t0 .

- Since address translation is done using a table, it is easy to have a large address space.



**FIGURE 5.21 The page table is indexed with the virtual page number to obtain the corresponding portion of the physical address.** We assume a 32-bit address. The starting address of the page table is given by the page table pointer. In this figure, the page size is  $2^{12}$  bytes, or 4 KB. The virtual address space is  $2^{32}$  bytes, or 4 GB, and the physical address space is  $2^{30}$  bytes, which allows main memory of up to 1 GB. The number of entries in the page table is  $2^{20}$ , or 1 million entries. The valid bit for each entry indicates whether the mapping is legal. If it is off, then the page is not present in memory. Although the page table entry shown here need only be 19 bits wide, it would typically be rounded up to 32 bits for ease of indexing. The extra bits would be used to store additional information that needs to be kept on a per-page basis, such as protection. Copyright © 2009 Elsevier, Inc. All rights reserved.

- Here, virtual address space is  $2^{32}$  bytes; but actual memory is only  $2^{30}$  bytes.
- This table lookup adds some delay to the execution of each instruction.
  - Special low level hardware is provided to make this step very fast.