CSSE 232
Computer Architecture I

Running a Program

Class Status

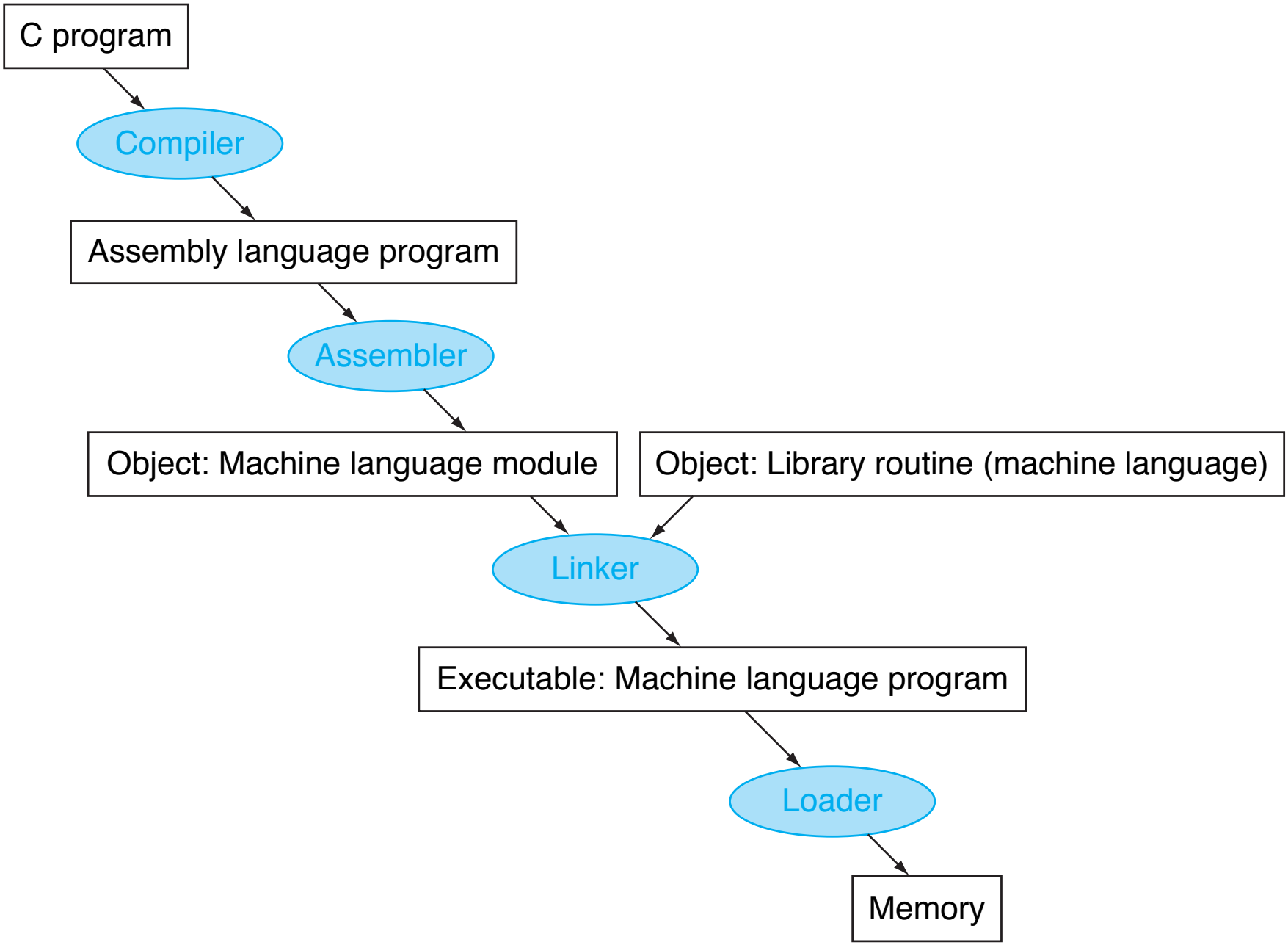
Reading for today

- 2.12, 2.13, 2.14, B.1-5

Outline

- Compilers
- Assemblers
- Linkers
- Loaders

Translation and Startup in C



Compiliers

- Early software was written primarily in assembly language
  - Limited memory
- Definition of compiler:
  A program (set of programs) that transforms high level source code written within a programming language (such as C) to assembly

Compiliers

- First compiler written by Grace Hopper for the A-0 programming language (1952)
  - The compiler itself was written using assembly language
- First self-hosting compiler developed in a high level language was for the Lisp (1962)
- Usually written in the language that they compile
  - C compiler written in C
  - First compiler for a language would have to be compiled in another compiler (bootstrapping problem)

Compiler Structure

- Input is **high level code** (C, etc.)
- Checks syntax and semantics, performs type checks
  - Generates errors
- Optimizes code
- Translates the optimized code into **assembly code**

- You can make a compiler in CSSE 404: Compiler Construction!

Assembler

- Translates the **assembly language** into the appropriate binary equivalents (**object file**)
- Most assembler instructions represent machine instructions one-to-one
- Pseudo-instructions: figments of the assembler's imagination
  - `$at` (register 1): assembler temporary

```
move $t0, $t1     →  add $t0, $zero, $t1

blt $t0, $t1, L   →  slt $at, $t0, $t1
                     bne $at, $zero, L
```

Object Files

- Determine the addresses corresponding to the different labels
- Object file contains
  - Object File Header: described contents of object module
  - Text segment: translated instructions
  - Static data segment: data allocated for the life of the program
  - Relocation info: for contents that depend on absolute location of loaded program
  - Symbol table: global definitions and external refs
  - Debug info: for associating with source code

Linker

- Links **object files** together to produce an **executable image**
  - Merges segments
  - Resolve labels (determine their addresses) - example in branches and jumps
  - Patch internal and external references
  - Determine memory locations each module will occupy
- Executable file has same format as object file but with no unresolved references

Dynamic Linking

- Only link/load library procedure when it is called
  - Windows: Dynamic Link Library (dll)
  - Unix: Shared Object (so)
- Different from static linking
  - Requires procedure code to be relocatable
  - Avoids image bloat caused by static linking of all (transitively) referenced libraries
  - Can automatically use new library versions

Loading a Program

- Load from image file on disk into memory
  1. Read header to determine segment sizes
  2. Create virtual address space
  3. Copy text and initialized data into memory
     - Or set page table entries so they can be faulted in
  4. Set up arguments on stack
  5. Initialize registers (including `$sp`, `$fp`, `$gp`)
  6. Jump to startup routine
     - Copies arguments to `$a0`, ...and calls main
     - When main returns, do exit syscall

Review and Questions

- Compilers
- Assemblers
- Linkers
- Loaders

Program demo

Demo of compiling, assembling, and linking

Project

Project details on website

- Write assembly code for the `relprime()` function.