# Compilers, Linkers, Loaders

Computer Science 104

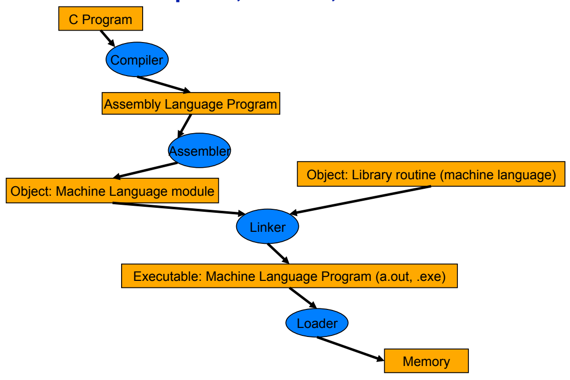
## Administrivia

- Homework #2 Due Today
  - ➢ Extra credit is on Blackboard
- Midterm: Monday Feb 16 in class
  - ➢ Covers up through today's lecture
  - ➢ Open book/notes
  - ➢ Review session Friday
- Friday: finish ASM programming on DE2 boards
- Reading
  - ➢ Compilers, linking & loading 2.12

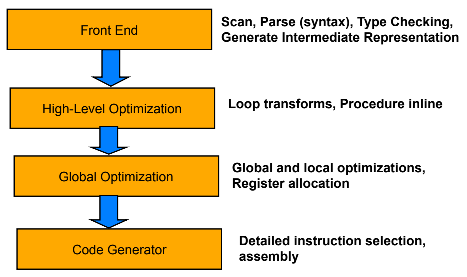
<u>Next week</u>

- Logic Design
- Reading Appendix C.1-C.3, C.5

## Compilers, Linkers, Loaders

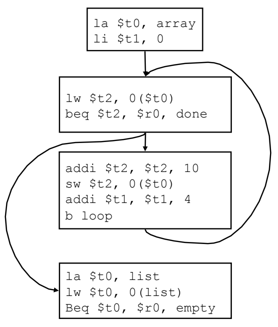

C Program
Compiler
Assembly Language Program
Assembler
Object: Machine Language module
Object: Library routine (machine language)
Linker
Executable: Machine Language Program (a.out, .exe)
Loader
Memory

## Compiler


Front End
Scan, Parse (syntax), Type Checking, Generate Intermediate Representation
High-Level Optimization
Loop transforms, Procedure inline
Global Optimization
Global and local optimizations, Register allocation
Code Generator
Detailed instruction selection, assembly

## Compilers

- Basic block = sequence of instructions with a single entry point and a single exit point
  - ➢ First instruction is target of branch or jump or first instruction after branch or jump
  - ➢ Last instruction is branch or jump
- Connect basic blocks to form control flow graph
- Local optimizations are within a basic block
- Global optimizations are across basic block



```
la $t0, array
li $t1, 0
```

```
lw $t2, 0($t0)
beq $t2, $r0, done
```

```
addi $t2, $t2, 10
sw $t2, 0($t0)
addi $t1, $t1, 4
b loop
```

```
la $t0, list
lw $t0, 0(list)
Beq $t0, $r0, empty
```

## Compiler Optimizations

- Common subexpression elimination
  - ➢ Array index address computation
- Strength reduction
  - ➢ Replace complex operations with simpler ops
- Constant Propagation
  - ➢ int x = 200; … y = x + 40; z = y – 10;
- Copy Propagation
- Dead Store (code) elimination
  - ➢ Stores whose values are never used again
  - ➢ Code that is never executed

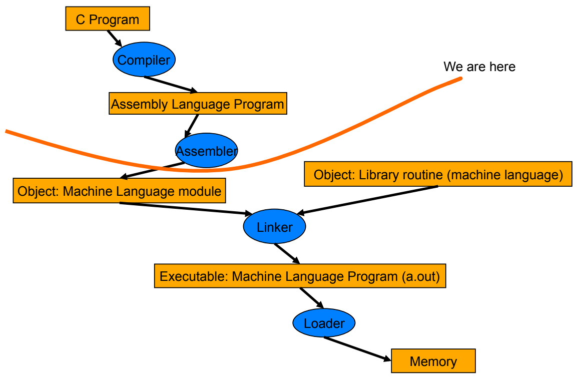
## Global Optimizations

- Occur across basic blocks
- In addition to the previous set
- Code Motion
  - Find code that is loop invariant and move it before loop
  - Computes the same value every iteration
- Induction variable elimination
  - Iterate over array using index k
  - Could compute address using array_start plus offset (k * 4)
  - Use pointer based approach where you can just increment address by 4

## Register Allocation

- Compiler first generates an intermediate representation (IR)
  - Virtual registers (unlimited number of them…)
- Must map from virtual registers to real set of registers
- Goal is to reduce the number of loads & stores
- If not enough registers, must "spill" = save to stack and restore from stack
- Sophisticated algorithms, reduces to graph coloring
  - Given graph, color each node such that no two adjacent nodes have the same color
  - Color = register number

## Linker


C Program
Compiler
Assembly Language Program
We are here
Assembler
Object: Machine Language module
Object: Library routine (machine language)
Linker
Executable: Machine Language Program (a.out)
Loader
Memory

## Linker

- Ability to resolve labels across multiple files
- Compiler creates one object file per source file
- Includes symbol table that identifies labels within a file and any instructions that need to be "fixed"
- Linker fills in values when they become known
- Static linking, all objects are linked to create executable file
- Dynamic linking, (DLL), occurs during execution
  - Jump table

## Loader: starting execution

- Part of Operating System that reads executable file off disk and starts execution
- Executable file has header information that identifies size of text and data in (ELF, COFF)
- Works with OS to establish address space
  - That ideal view of memory being $2^{32}$ bytes large
- Copies arguments into registers and stack
- Points PC to first instruction of startup code (which calls main)
  - On return from main, this "startup" code executes an exit system call

## Virtual Functions

- A little NiosII IDE and gnutools demo….

## Summary

- Procedure calls
- Compilers, linkers, loaders

<u>Next Time</u>

- Midterm Monday

<u>Reading</u>

- Start Appendix C -- logic design