Assembler/Linker/Loader

Mooly Sagiv

html://www.cs.tau.ac.il/~msagiv/courses/wcc20.html

Chapter 4.3
J. Levine: Linkers & Loaders
http://linker.iecc.com/

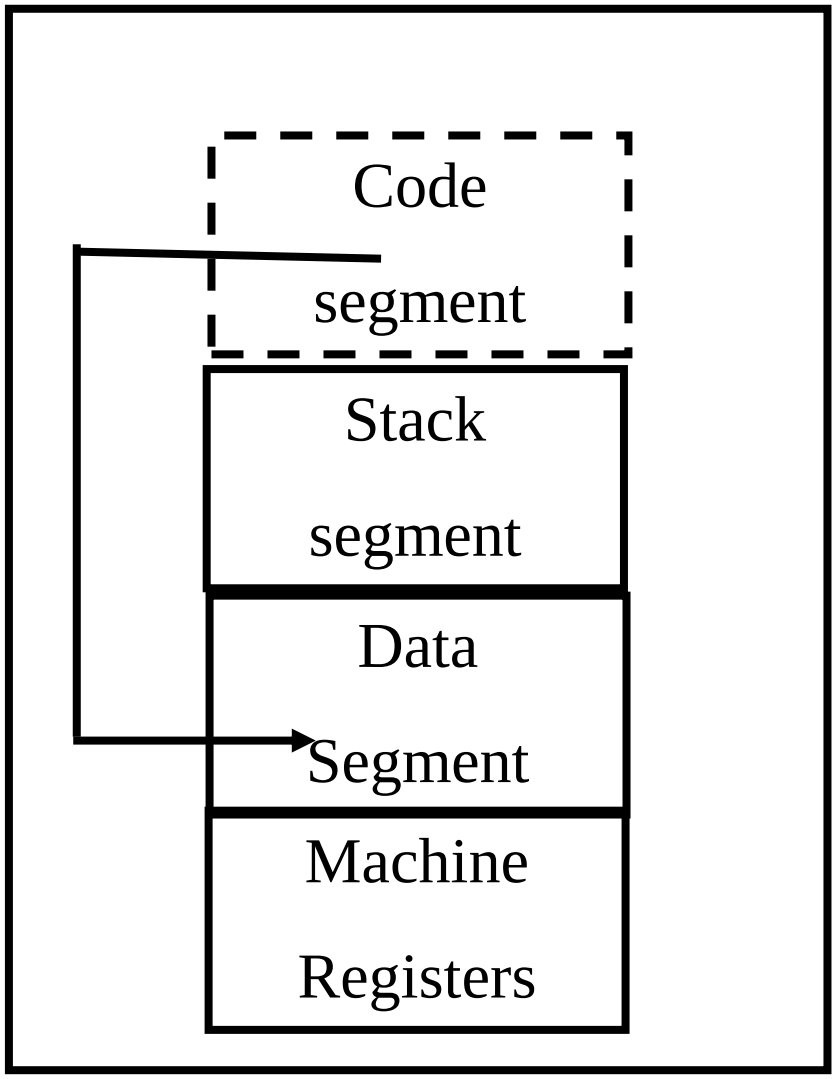
Outline

- Where does it fit into the compiler
- Functionality
- “Backward” description
- Assembler design issues
- Linker design issues

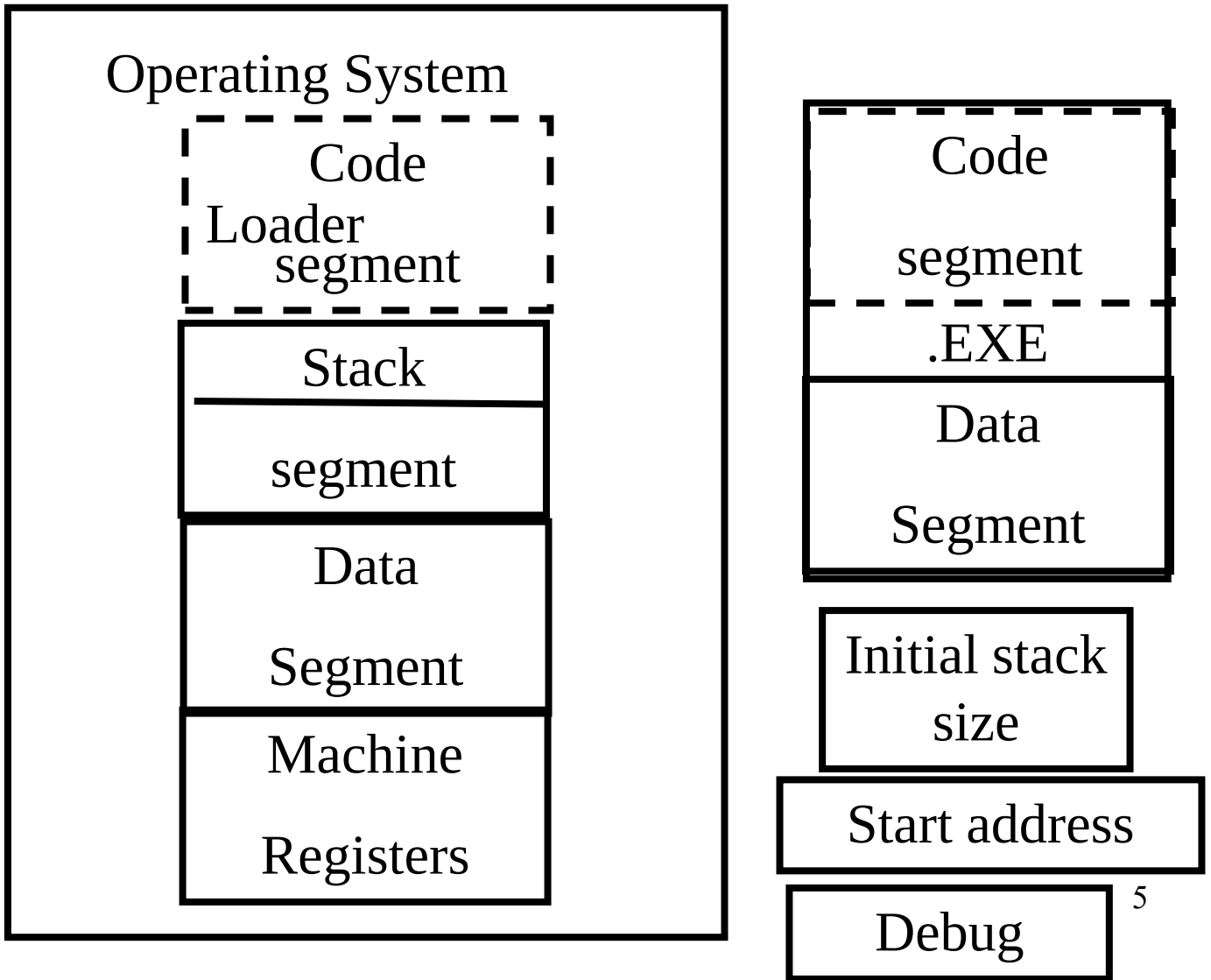
Assembler

- Generate executable code from assembly
- Yet another compiler
- One-to one translation
- Resolve external references
- Relocate code
- How does it fit together?
- Is it really part of the compiler?

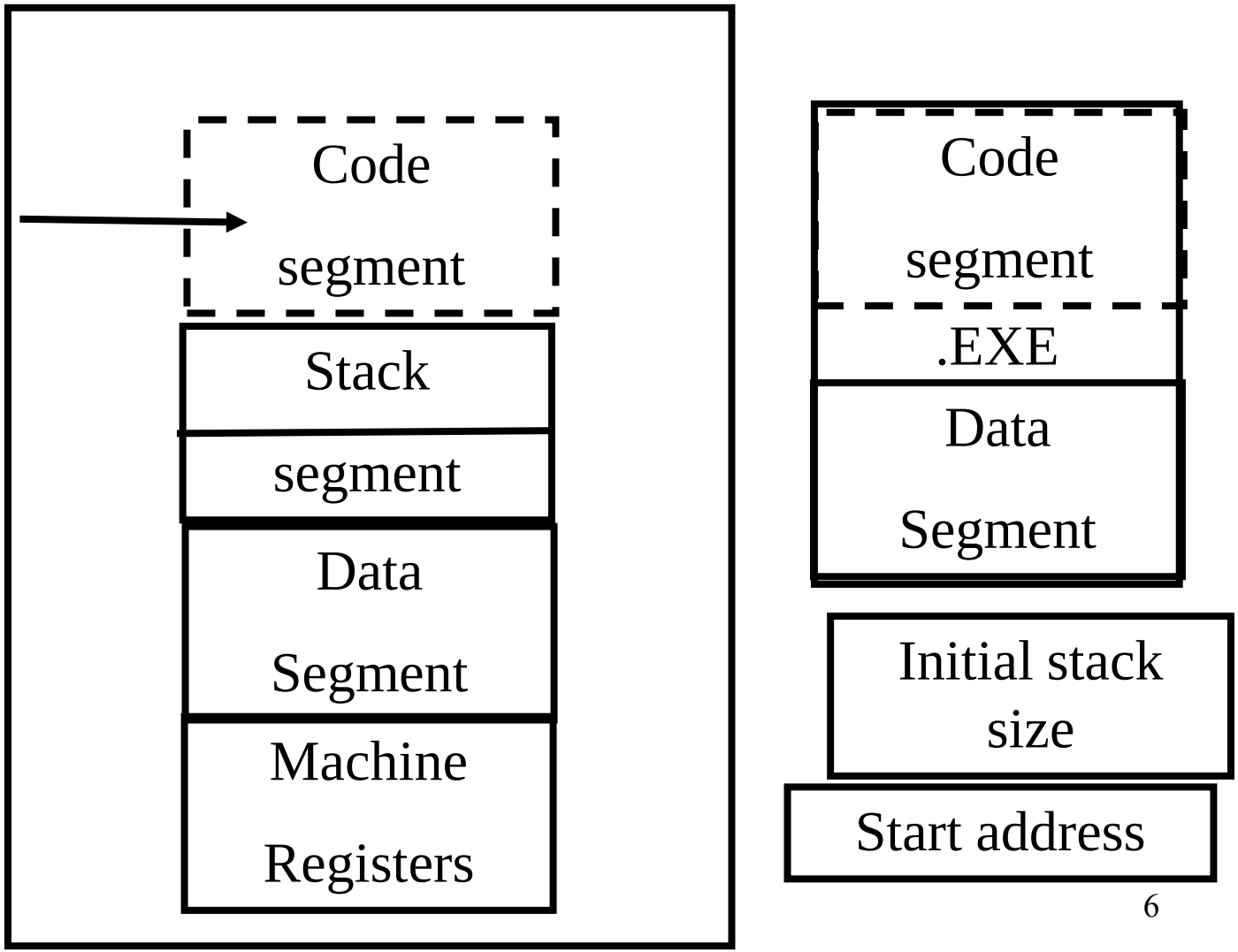
Program Runtime State



Program Run


Operating System
Code
Loader
segment
Stack
segment
Data
Segment
Machine
Registers
Code
segment
.EXE
Data
Segment
Initial stack
size
Start address
Debug

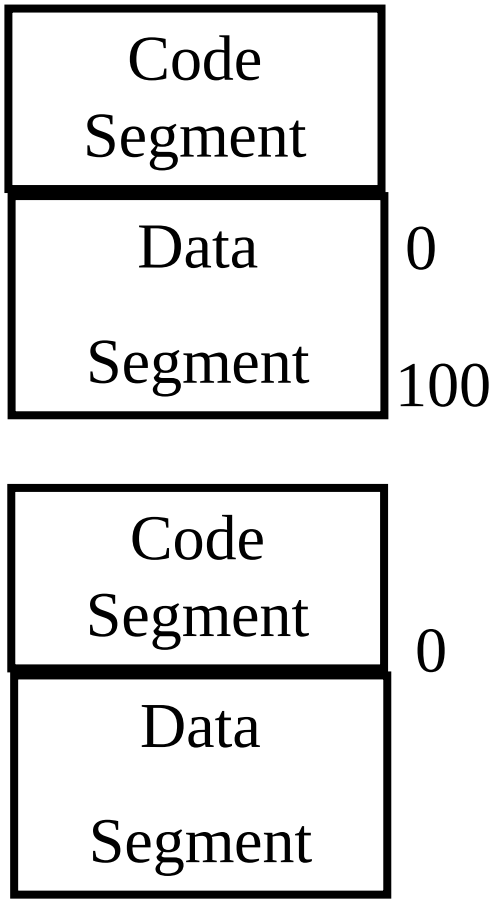
Program Run


Code
segment
Stack
segment
Data
Segment
Machine
Registers
Code
segment
.EXE
Data
Segment
Initial stack
size
Start address

Loader (Summary)

- Part of the operating system
- Does not depend on the programming language
- Privileged mode
- Initializes the runtime state
- Invisible activation record

Linker



External Symbol Table

Relocation Bits

Linker

- Merge several executables
- Resolve external references
- Relocate addresses
- User mode
- Provided by the operating system
- But can be specific for the compiler
  - More secure code
  - Better error diagnosis

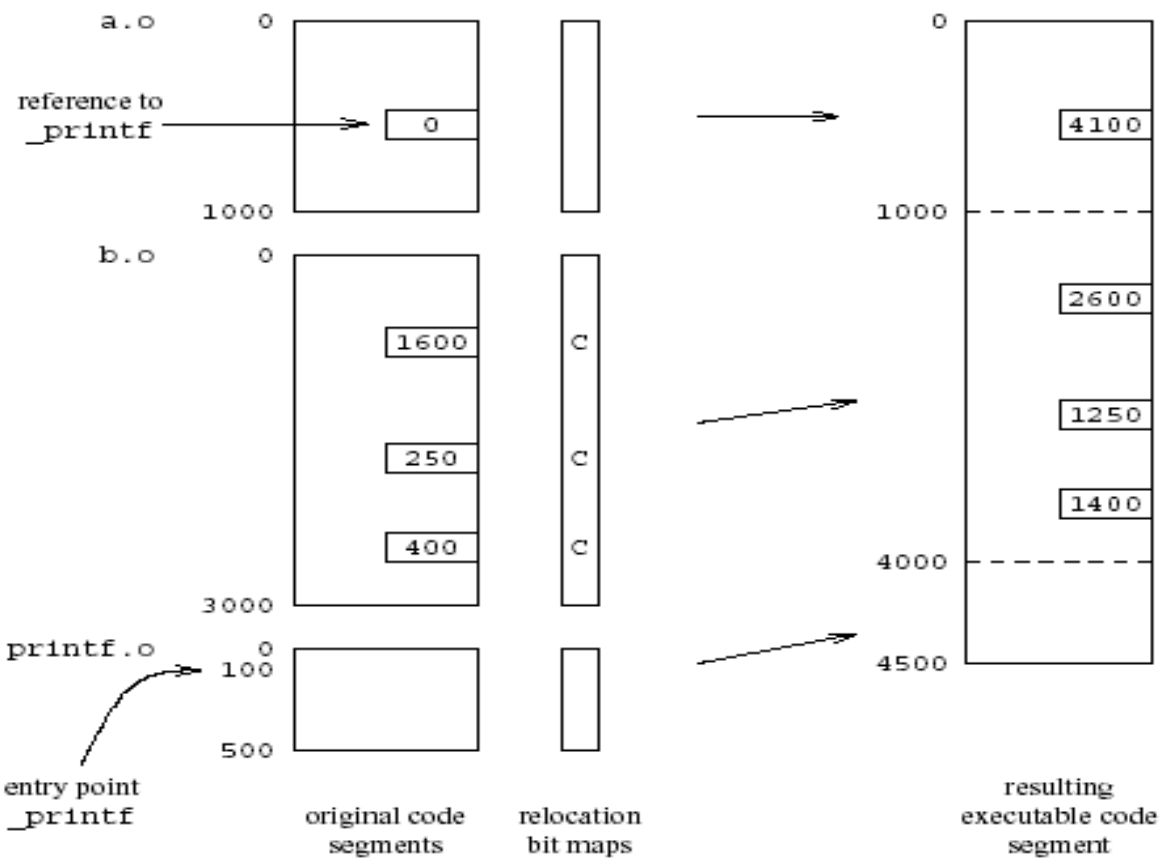
Relocation information

- How to change internal addresses
- Positions in the code which contains addresses (data/code)
- Two implementations
  - Bitmap
  - Linked-lists

External References

- The code may include references to external names (identifiers)
  - Library calls
  - External data
- Stored in external symbol table

Example


a.o
0
reference to
_printf
0
1000
b.o
0
1600
250
400
3000
printf.o
0
100
500
entry point
_printf
C
C
C
original code
segments
relocation
bit maps
0
4100
1000
2600
1250
1400
4000
4500
resulting
executable code
segment

Recap

- Assembler generates binary code
  - Unresolved addresses
  - Relocatable addresses
- Linker generates executable code
- Loader generates runtime states (images)

Assembler Design Issues

- Converts symbolic machine code to binary
- One to one conversion
  addl %edx, %ecx $\Rightarrow$ 000 0001 11 010 001 = 01 D1 (Hex)
- Some assemblers support overloading
  - Different opcodes based on types
- Format conversions
- Handling internal addresses

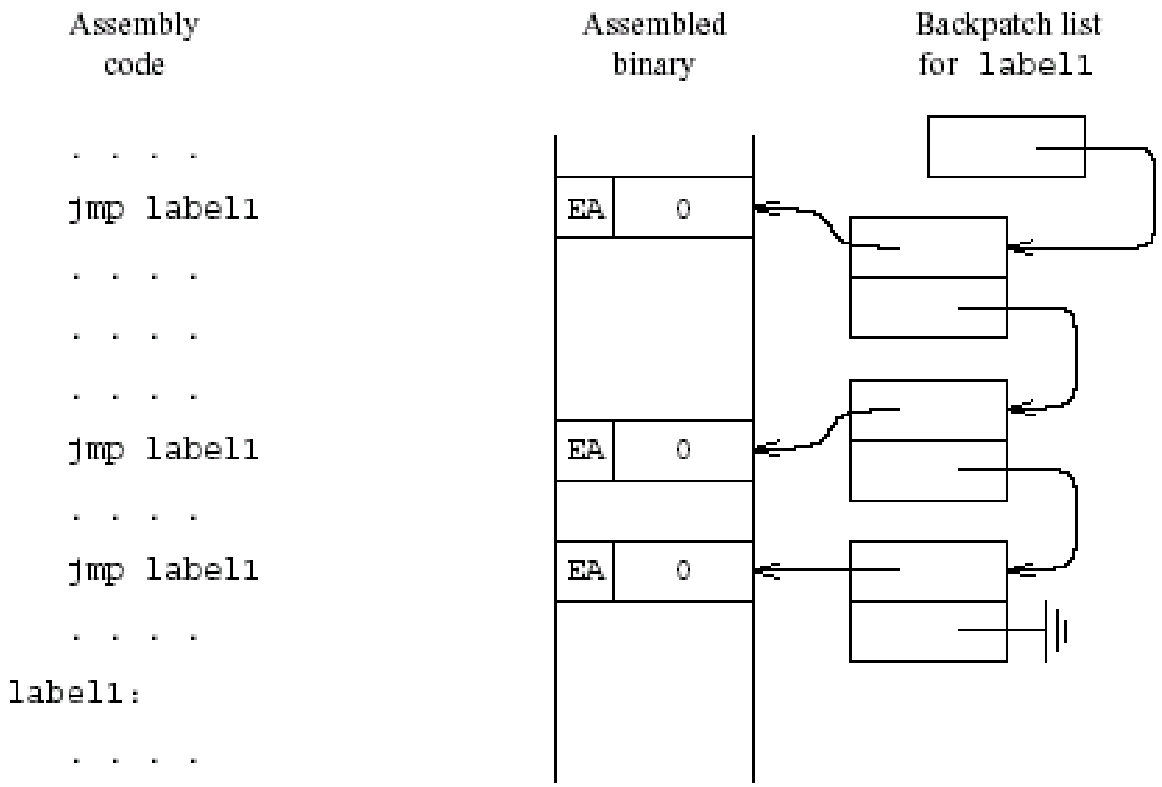
Handling Internal Addresses

```
.data
        ...
        .align 8
var1:
        .long 666
        ...
.code
        ...
        addl var1,%eax
        ...
        jmp label1
        ...
label1:
        ...
        ...
```

Resolving Internal Addresses

- Two scans of the code
  - Construct a table label $\rightarrow$ address
  - Replace labels with values
- Backpatching
  - One scan of the code
  - Simultaneously construct the table and resolve symbolic addresses
  - Maintains list of unresolved labels
  - Useful beyond assemblers

Backpatching



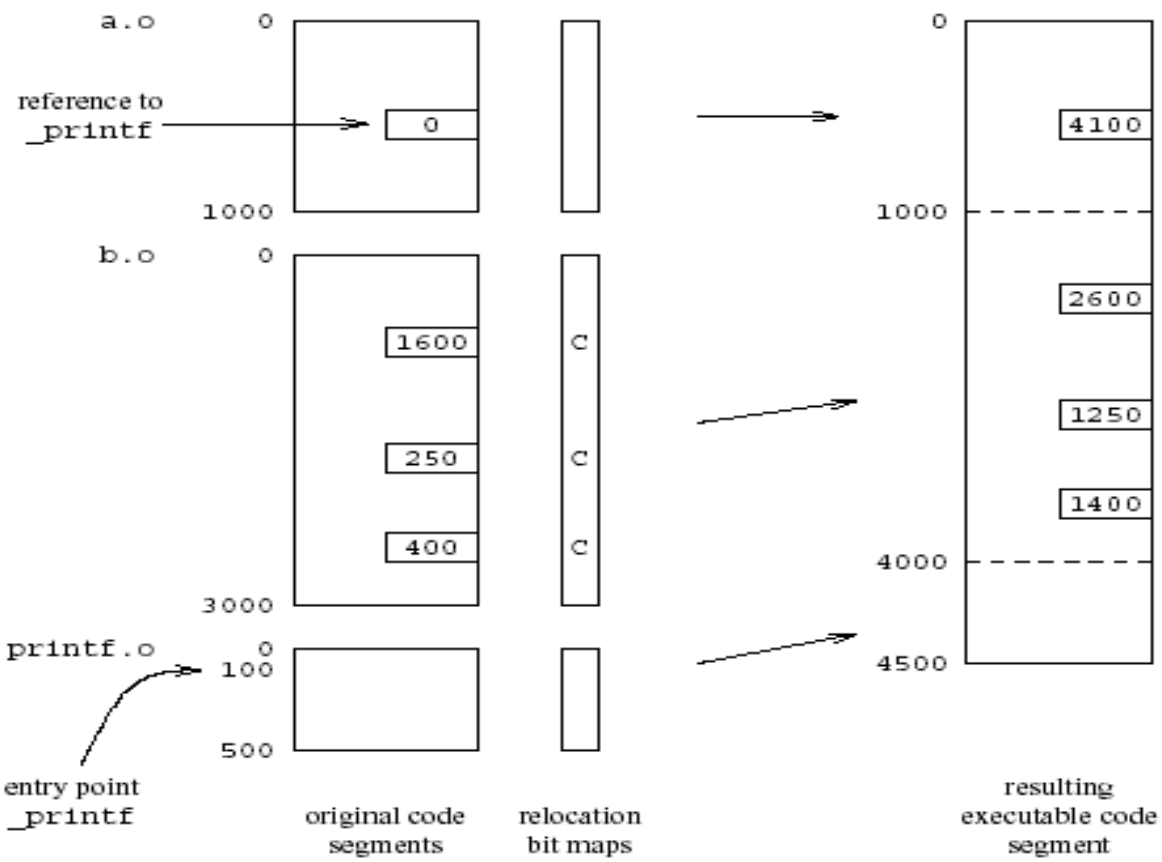
Handling External Addresses

- Record symbol table in external table
- Produce binary version together with the code and relocation bits
- Output of the assembly
  - Code segment
  - Data segment
  - Relocation bits
  - External table

Example of External Symbol Table

External symbol	Type	Address	
_options	entry point	50	data
__main	entry point	100	code
_printf	reference	500	code
_atoi	reference	600	code
_printf	reference	650	code
_exit	reference	700	code
_msg_list	entry point	300	data
_Out_Of_Memory	entry point	800	code
_fprintf	reference	900	code
_exit	reference	950	code
_file_list	reference	4	data

Example


a.o
0
reference to
_printf
0
1000
b.o
0
1600
250
400
3000
printf.o
0
100
500
entry point
_printf
C
C
C
original code
segments
relocation
bit maps
0
4100
1000
2600
1250
1400
4000
4500
resulting
executable code
segment

Summary

- Code generation yields code which is still far from executable
  - Delegate to existing assembler
- Assembler translates symbolic instructions into binary and creates relocation bits
- Linker creates executable from several files produced by the assembly
- Loader creates an image from executable
- Missing: Dynamic loading