Machine Code and Assemblers

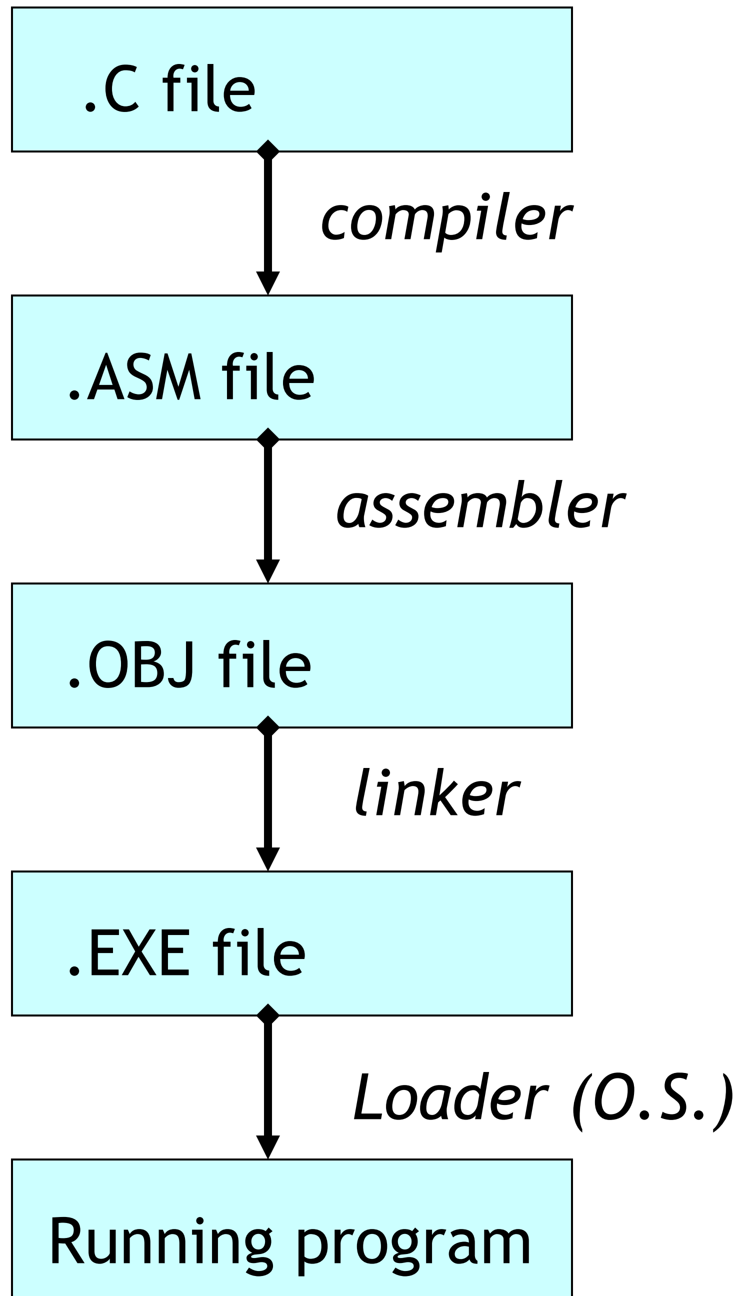
November 6

CSC201 Section 002

Fall, 2000

Definitions

- Assembly time vs. link time vs. load time vs. run time



Copyright 2000, Douglas Reeves

The Assembler

- Converts assembly language program into a machine-executable binary program
- The assembler does as much processing of the program as possible
  - so program running time will be speeded up

Copyright 2000, Douglas Reeves

Tasks of the Assembler

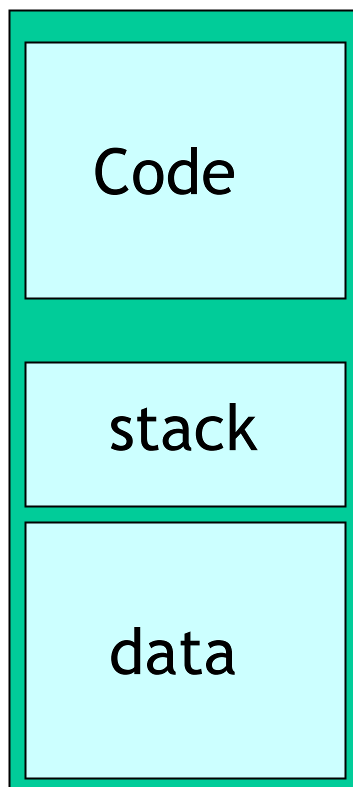
- Macro expansion (preprocessing)
- Organize memory, reserve and initialize memory for data
- Convert instructions and constants into binary
- Convert symbolic addresses into absolute addresses
- Mark external references for later resolution

Copyright 2000, Douglas Reeves

Memory Organization

- Separate memory "regions" for the stack, the code, and the data
- Memory is allocated sequentially as each part of the program is processed
  - The assembler keeps track of how much has been allocated so far

Example memory organization



Copyright 2000, Douglas Reeves

The Symbol Table

- Symbolic labels help readability, relocatability, and ease of modification
- A list of the symbolic names encountered so far
  - Data definitions
  - Code labels

Copyright 2000, Douglas Reeves

The Symbol Table

- For code and data, keep track of...
  - Memory address corresponding to that symbolic name, determined during memory allocation
- For data, keep track of...
  - Data type (byte? word? doubleword? floating-point?)
  - Initial value
  - Size (amount of memory)
- Table has no use after assembly

Copyright 2000, Douglas Reeves

Directives

- Provide information to the assembler, used to generate correct output
- "title" -- no processing
- ".486" -- check for 486-compatible instruction syntax, and generate 486 machine code
- ".model flat" -- use the flat (not segmented) memory model

Copyright 2000, Douglas Reeves

Directives

- "stdcall" -- generate standard names for symbols, and use a standardized calling convention for procedure calls
- ".data" -- following are data definitions
- data definitions ("db", "dw", ..) -- allocate the appropriate amount of memory, properly initialized
- ".code" -- following are executable instructions
- "end" -- ignore the remainder of this file

Copyright 2000, Douglas Reeves

Compiling C to Assembler

- The user's program will be called by the operating system like a procedure
  - On entry, save and initialize EBP, save registers
  - For this example, assumes local storage has been allocated on the stack *before* the program is called
  - On exit, restore registers and execute "ret" statement

Copyright 2000, Douglas Reeves

x86 Machine Code

- Although assembly languages on machines are similar, machine languages are completely different
- Pentium machine code "peculiarities"
  - Variable length instruction encodings
  - Many addressing modes
  - Extra byte needed to specify addressing mode in some cases
  - "Repeat" prefixes
- None of these commonly used today

Copyright 2000, Douglas Reeves

Address Resolution

- As symbols are referenced, replace by the starting address of memory allocated to that symbol
- What if symbol is defined further down in the program (ex. program label)?
- Two-pass process
  - One pass to generate machine code and allocate memory
  - second pass to resolve symbolic address references

Copyright 2000, Douglas Reeves

Instruction Formats

- Motivation: understand slightly how CPU "reads" an executable program
- Goals for instruction format design (conflicting): use minimum space, vs. make decoding as easy as possible
- x86 encoding: complex, irregular, baroque even
- All instruction encodings have an "opcode" specifier
- For the most part, we are looking at 32-bit operand versions only

Copyright 2000, Douglas Reeves

Register Specification

- We're looking at doubleword-length registers only
- Note: not in order!

Binary Symbol	Register
000	EAX
001	ECX
010	EDX
011	EBX
100	ESP
101	EBP
110	ESI
111	EDI

Copyright 2000, Douglas Reeves

Instruction Encoding

- Instructions = 1 to 9 bytes long
- First byte
  - Opcode,D,W
  - -or- Opcode,V,W
  - -or- Opcode,W
  - -or- Opcode,Reg
- Second byte (if needed) = the Mod,Reg,R/M byte
  - Mod = 2 bits, Reg = 3 bits, R/M = 3 bits)
- Third byte needed by some addressing modes: SIB byte
  - Scaling = 2 bits, Index register = 3 bits, Base register = 3 bits

Copyright 2000, Douglas Reeves

D, V, W

- D: 0= "reg" field is source, 1= "reg" field is destination
- V: 0= shift by 1 bit, 1= shift by amount in cl register
- W: 0= data is byte length, 1= data is word length

Copyright 2000, Douglas Reeves

Encoding Instructions Using Appendix C

Copyright 2000, Douglas Reeves

Decoding Instructions Using Appendix C

Copyright 2000, Douglas Reeves

Conditional Jump

- Conditional jump instruction encodes displacement from next instruction
  - 32-bit offset for our purposes

Copyright 2000, Douglas Reeves