Assembly: Introduction.

Yipeng Huang

Rutgers University

February 25, 2021

Table of contents

Announcements

Big picture view of computer architecture

- The memory hierarchy

Assembly

- Human readable machine code
- Instructions for the microarchitecture

Looking ahead

Class plan

1. Today, Thursday, 2/25: Assembly, machine code.
2. Reading assignment for next four weeks: CS:APP Chapter 3.
3. Thursday, 2/25: Programming Assignment 3 on bits, bytes, integers, floats out.
4. Monday, 3/1: Programming Assignment 2 due. Be sure to test on ilab, "make clean". Quiz 6 on floating point trickiness out.

Programming Assignment 2: FAQs

1. In recursive code, the return type contains important information. isTreeDFS() returns a Boolean. When you call isTreeDFS(), you need to capture the return and use it.
2. What the parents array in solveMaze BFS represents.

Table of contents

Announcements

Big picture view of computer architecture

- The memory hierarchy

Assembly

- Human readable machine code
- Instructions for the microarchitecture

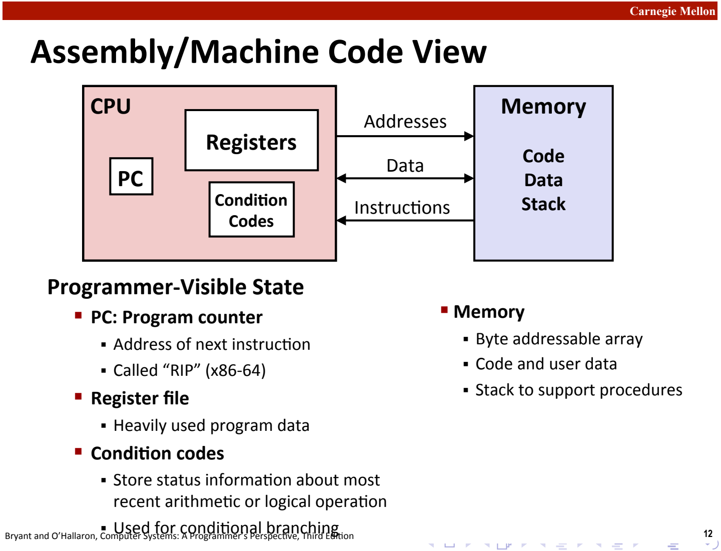
Stored program computer

Stored program:

Instructions reside in memory, loaded as needed.

von Neumann architecture:

Data and instructions share same connection to memory.



Assembly/Machine Code View

Programmer-Visible State

- **PC: Program counter**
  - Address of next instruction
  - Called "RIP" (x86-64)
- **Register file**
  - Heavily used program data
- **Condition codes**
  - Store status information about most recent arithmetic or logical operation
  - Used for conditional branching

- **Memory**
  - Byte addressable array
  - Code and user data
  - Stack to support procedures

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memory hierarchy

	Capacity	Access speed
Internet		
Tape	250Pb	
Hard drives	16TB	2Mb/s
Solid state drives	4TB	2Gb/s
DRAM	8Gb - 1Tb+	8Gb/s
Last-level cache	64Mb	
Level-1 cache	1Mb	
Registers	1Kb	

- Registers (.25ns; 4GHz => .25e-9s)

Table of contents

Announcements

Big picture view of computer architecture

- The memory hierarchy

Assembly

- Human readable machine code
- Instructions for the microarchitecture

Assembly

Human readable machine code

- Very limited
- Not much control flow
- Any more complex functionality is built up
- for loops, while loops, turn into assembly sequence

Choice of what assembly to experiment with

- MIPS
- ARM
- x86 / x86-64 (not ideal for teaching, but it allows us to experiment on ilab)

Why are instruction set architectures important

Interface between computer science and electrical and computer engineering

- Software is varied, changes
- Hardware is standardized, static

Computer architect Fred Brooks and the IBM 360

- IBM was selling computers with different capacities,
- Compile once, and can run software on all IBM machines.
- Backward compatibility.
- An influential idea.

CISC vs. RISC

Complex instruction set computer

- Intel and AMD
- Have an extensive and complex set of instructions
- For example: x86's extensions: x87, IA-32, x86-64, MMX, 3DNow!, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.2, SSE5, AES-NI, CLMUL, RDRAND, SHA, MPX, SGX, XOP, F16C, ADX, BMI, FMA, AVX, AVX2, AVX512, VT-x, VT-d, AMD-V, AMD-Vi, TSX, ASF
- Can license Intel's compilers to extract performance
- Secret: inside the processor, they break it down to more elementary instructions

CISC vs. RISC

Reduced instruction set computer

- MIPS, ARM, RISC-V (can find Patterson and Hennessy Computer Organization and Design textbook in each of these versions), an PowerPC
- Have a relatively simple set of instructions
- For example: ARM's extensions: SVE;SVE2;TME; All mandatory: Thumb-2, Neon, VFPv4-D16, VFPv4 Obsolete: Jazelle
- ARM: smartphones, Apple ARM M1 Mac

Assembly instructions

Instructions for the microarchitecture

- Binary streams that tell an electronic circuit what to do
- Fetch, decode, execute, memory, writeback

A preview of microarchitecture

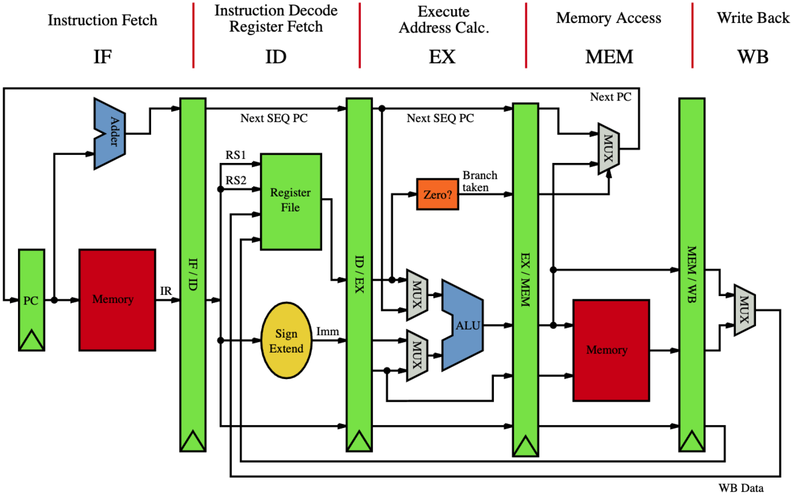
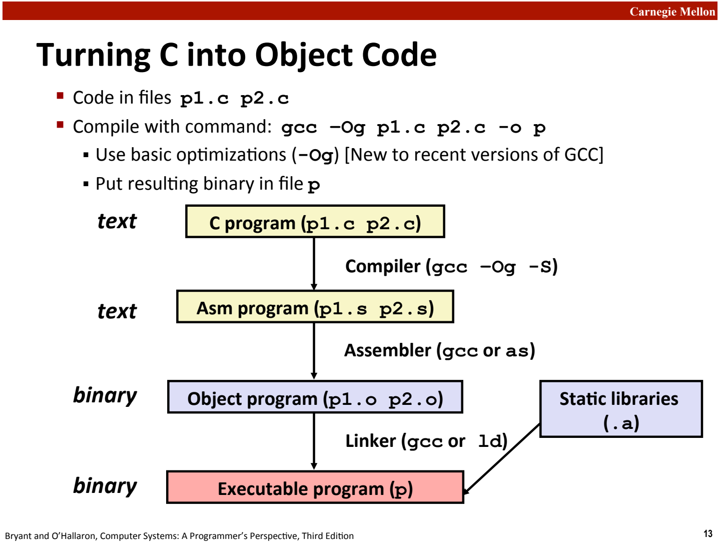


Figure: Stages of compilation. Image credit Wikimedia

Unraveling the compilation chain



Turning C into Object Code

- Code in files `p1.c p2.c`
- Compile with command: `gcc –Og p1.c p2.c -o p`
  - Use basic optimizations (`–Og`) [New to recent versions of GCC]
  - Put resulting binary in file `p`

text C program (`p1.c p2.c`)

Compiler (`gcc –Og -S`)

text Asm program (`p1.s p2.s`)

Assembler (`gcc` or `as`)

binary Object program (`p1.o p2.o`)

Static libraries (`.a`)

Linker (`gcc` or `ld`)

binary Executable program (`p`)

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition