Computer Architecture 1

Computer Organization and Design

T H E H A R D W A R E / S O F T W A R E I N T E R F A C E

[Adapted from Computer Organization and Design, RISC-V Edition, Patterson & Hennessy, © 2018, MK]
[Adapted from Great ideas in Computer Architecture (CS 61C) lecture slides, Garcia and Nikolíc, © 2020, UC Berkeley]

What you need to know about this class

- **Course information**
- Course Website:

  https://sites.google.com/site/kimhuetadtvtbk/cac-mon-giang-day/kien-truc-may-tinh

  https://cs61c.org/sp20/#lectures
- Textbooks: Average 15 pages of reading/week
  - Patterson & Hennessy, Computer Organization and Design, RISC-V edition
  - Kernighan & Ritchie, The C Programming Language 2nd Edition
  - Barroso & Holzle, The Datacenter as a Computer 3rd

What you need to know about this class

- **Course Grading**
- Homework, Assignments, mini-tests(15%)
- Project, midterm exam (15%)
- Final exam (70%)

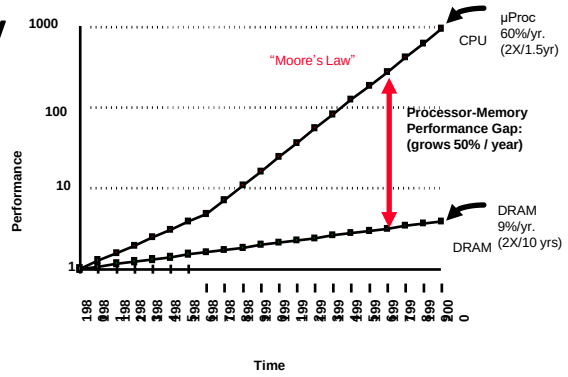
- **Extra Score: EPA!**
- Effort
  - Completing all assignments and homework in time
- Participation
  - Asking/Answering Qs in TEAMS/ offline class & making it interactive
- Altruism
  - Helping others in class
  - Writing software, tutorials that help others learn
- EPA! can bump students up to the next grade level

What is expected to this course ?

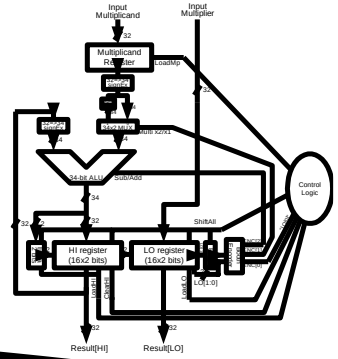
- To explain what's inside the revolutionary machine, unraveling the software below your program and the hardware under the covers of your computer
- To understand the aspects of both the hardware and software that affect program performance
- By the time you complete this course, you will be able to answer the following questions:
  - ✓How are programs written in a high-level language, such as C, Java translated into the language of hardware
  - ✓How does the hardware execute the resulting program?
  - ✓What determines the performance of a program?
  - ✓What techniques can be used by a programmer/hardware designers to improve the performance

Road map

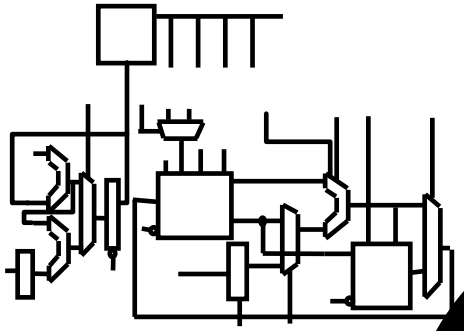
Chapter 1: Introduction



Chapter 2: ISA MIPS – RISC V

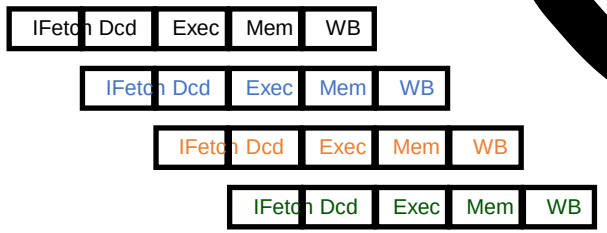


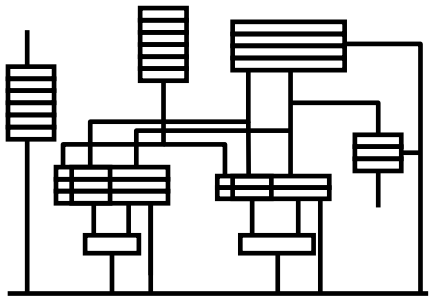
Chapter 3: Single/multicycle Datapath

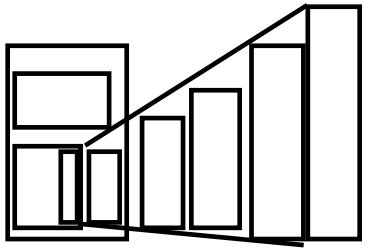


Course information is described in more detail in the syllabus

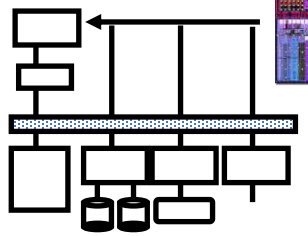
Chapter 4: Pipelining

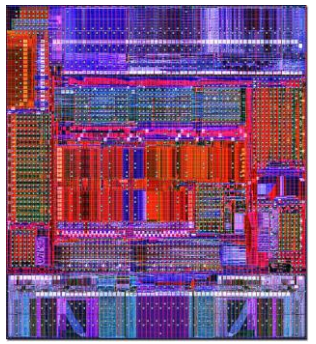






Chapter 5: Memory Systems



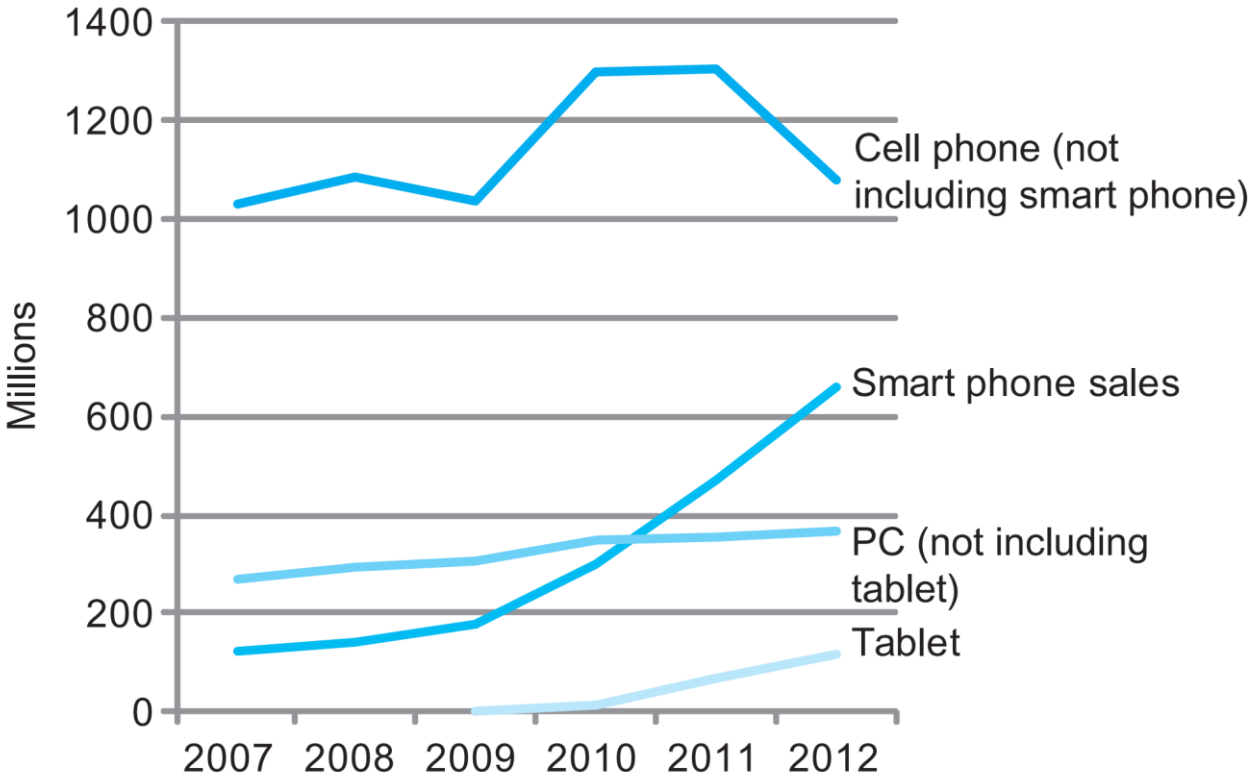


Chapter 6: Parallel processor

Introduction to Computer Abstractions and Technology

- Introduction
- Great Ideas in Computer Architecture

Introduction

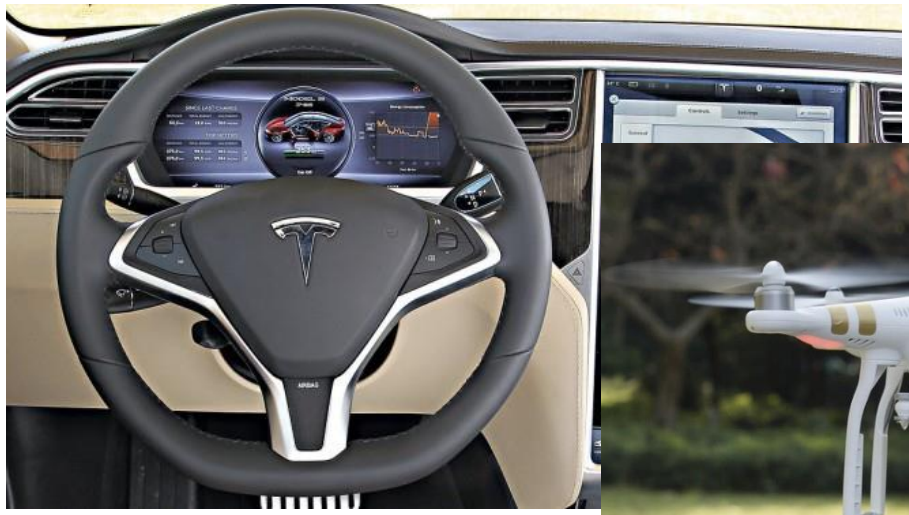


Hardware advances have allowed programmers to create wonderfully useful software, which explains why computers are omnipresent

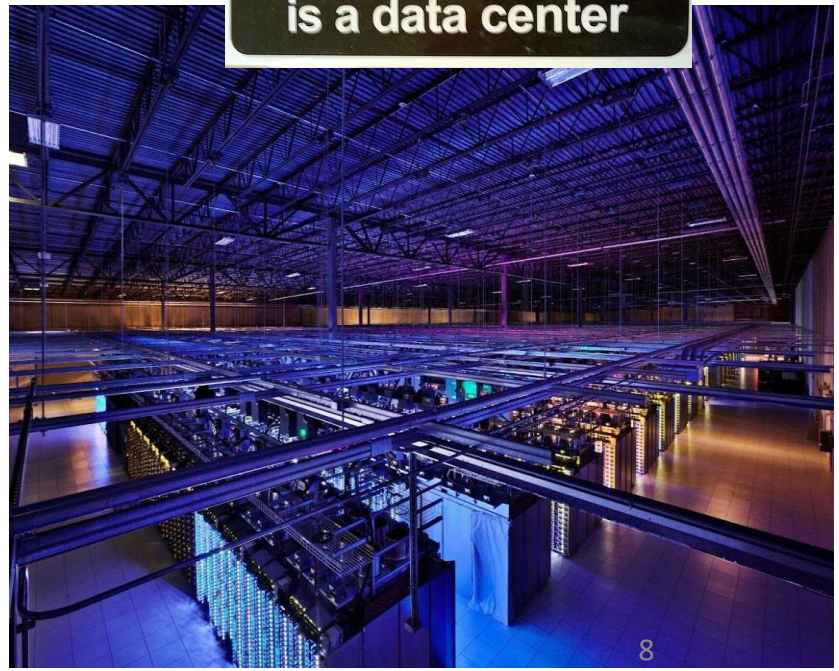
- *Computers in automobiles*
- *Cell phones*
- *Human genome project*
- *World Wide Web*
- *Search engines*

Smart phones represent the recent growth in the cell phone industry, and they passed PCs in 2011.
Tablets are the fastest growing category, nearly doubling between 2011 and 2012.
Recent PCs and traditional cell phone categories are relatively flat or declining

Welcome to the Post-PC Era



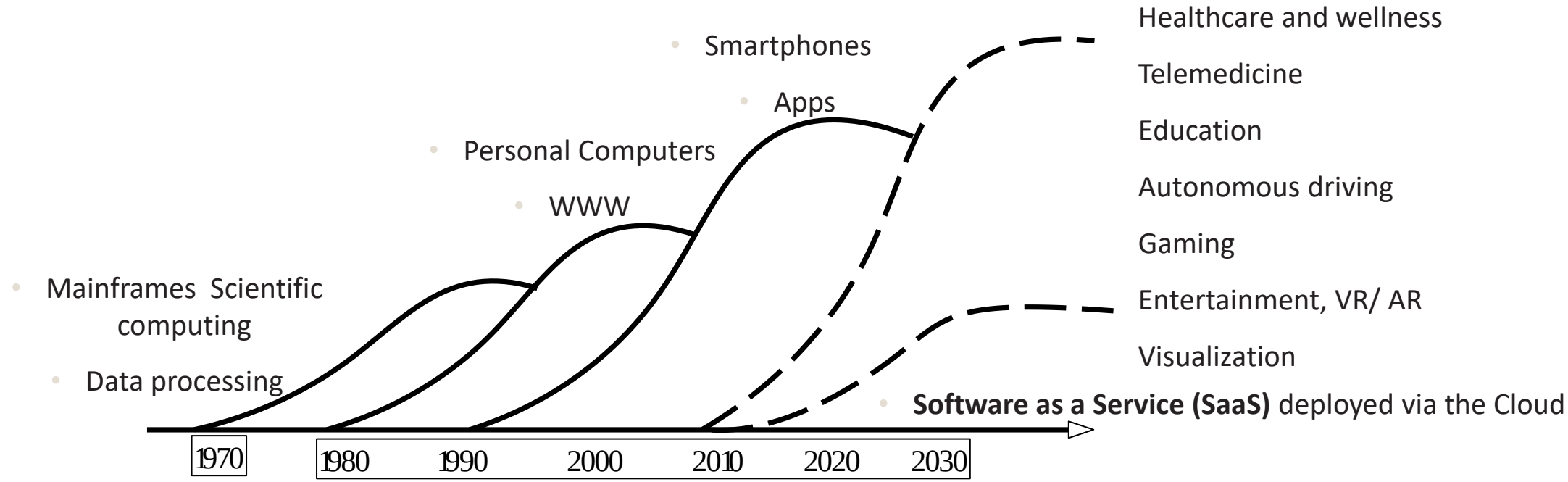
Embedded computes in Network Edge Devices



Replacing the PC is the Personal Mobile Devices (PMD)

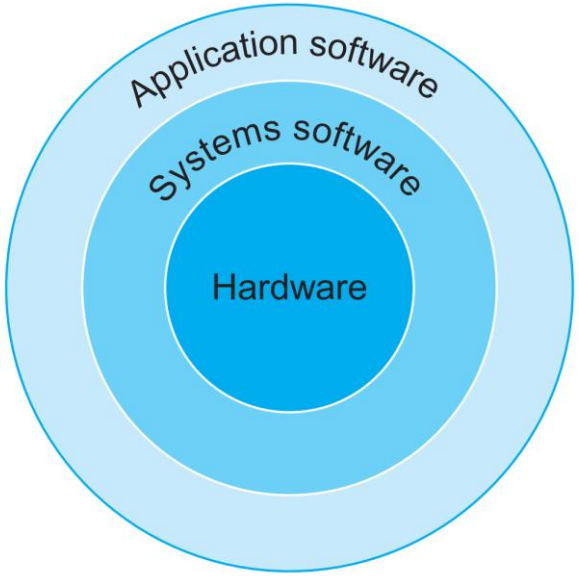
Taking over from the conventional server is **Cloud Computing**, which relies upon giant datacenters that are now known as *Warehouse Scale Computers* (WSCs).
Companies like Amazon and Google build these WSCs containing 100,000 servers

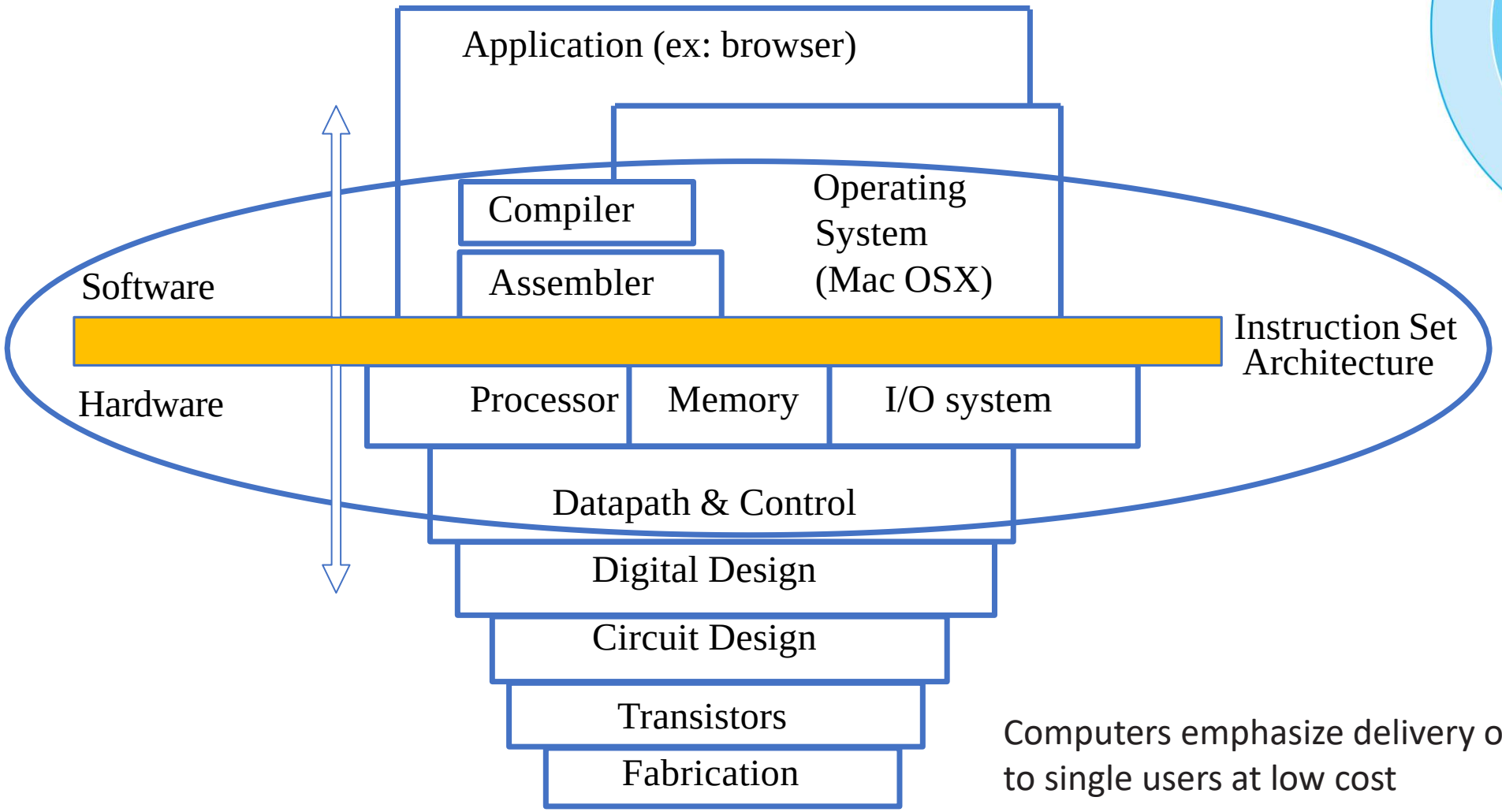
Why is Computer Architecture Exciting Today?



- Number of deployed devices continues to grow
  - Diversification of needs, architectures
  - Machine learning is common for most domains

Thinking about machine structure in Computer Architecture 1





Computers emphasize delivery of good performance to single users at low cost

Thinking about machine structure in Advanced Computer Arch – CA 2

Harness Parallelism & achieve high performance

Software

- **Parallel Requests**
  Assigned to computer
  e.g., Search Cats
- **Parallel Threads**
  Assigned to core e.g., Lookup, Ads
- **Parallel Instructions**
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- **Parallel Data**
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- **Hardware descriptions**
  All gates work in parallel at same time

Hardware

Warehouse Scale Computer

Smart Phone

Computer

Core Core

Memory (Cache)

Input/Output

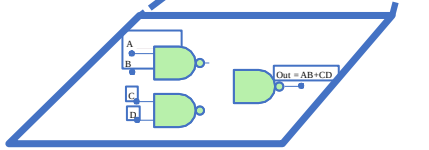
Exec. Unit(s)

Functional Block(s)

A_0+B_0 A_1+B_1

Main Memory

Logic Gates



6 Great Ideas in Computer Architecture

- Abstraction (Layers of Representation/Interpretation)
- Moore's Law
- Principle of Locality/Memory Hierarchy
- Parallelism
- Performance Measurement & Improvement
- Dependability via Redundancy

Great Idea #1: Abstraction (Layers of Representation/Interpretation)

High Level Language Program (e.g., C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compiler

Assembly Language Program (e.g., RISC-V)

```
lw   x3, 0(x10)
lw   x4, 4(x10)
sw   x4, 0(x10)
sw   x3, 4(x10)
```

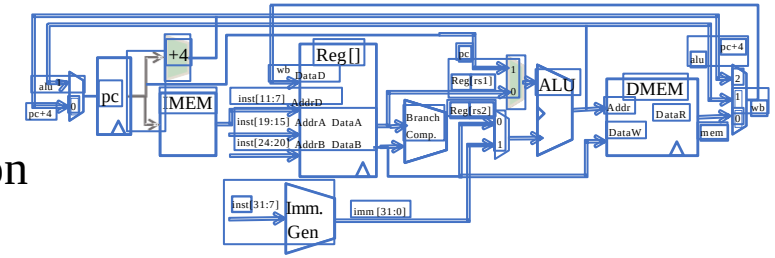
Anything can be represented as a number, i.e., data or instructions

Assembler

Machine Language Program (RISC-V)

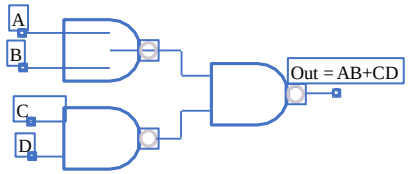
```
1000 1101 1110 0010 0000 0000 0000 0000
1000 1110 0001 0000 0000 0000 0000 0100
1010 1110 0001 0010 0000 0000 0000 0000
1010 1101 1110 0010 0000 0000 0000 0100
```

Hardware Architecture Description (e.g., block diagrams)



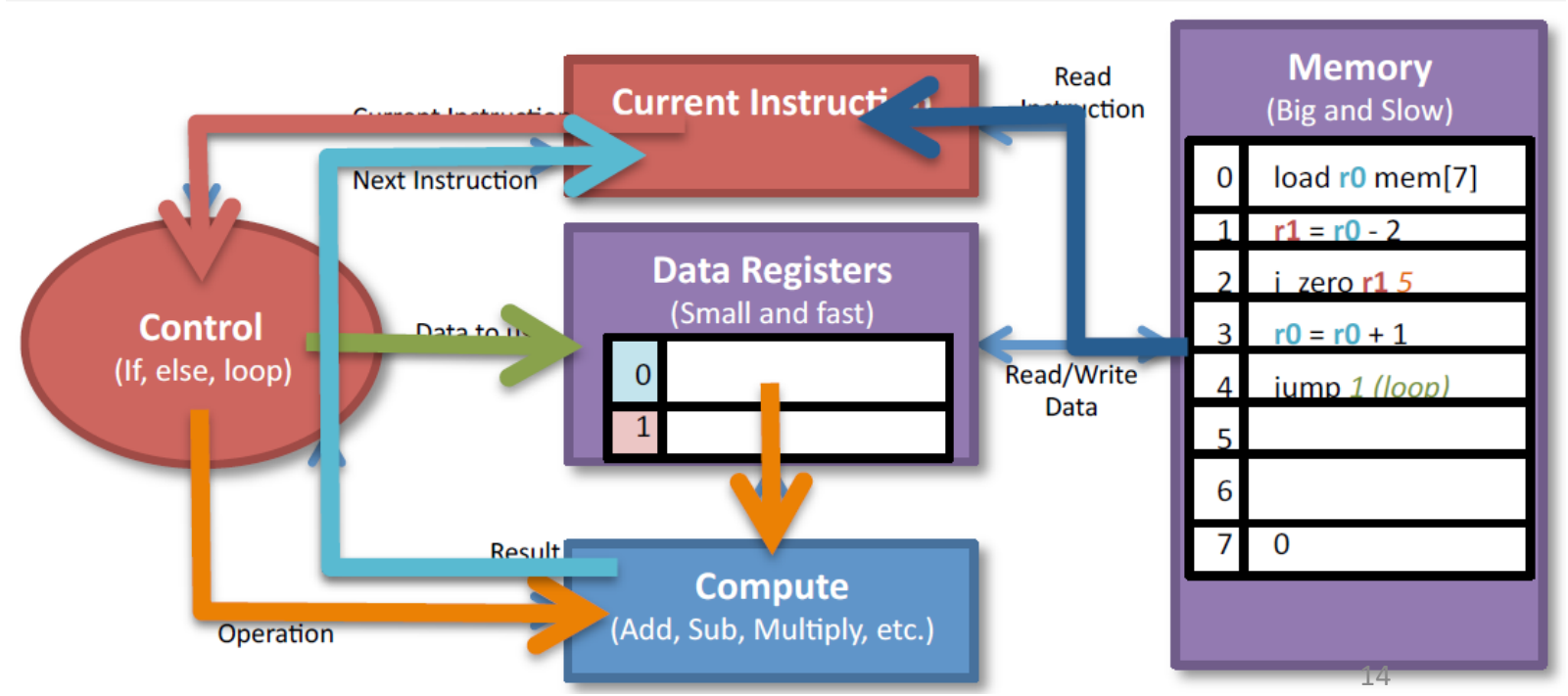
Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)

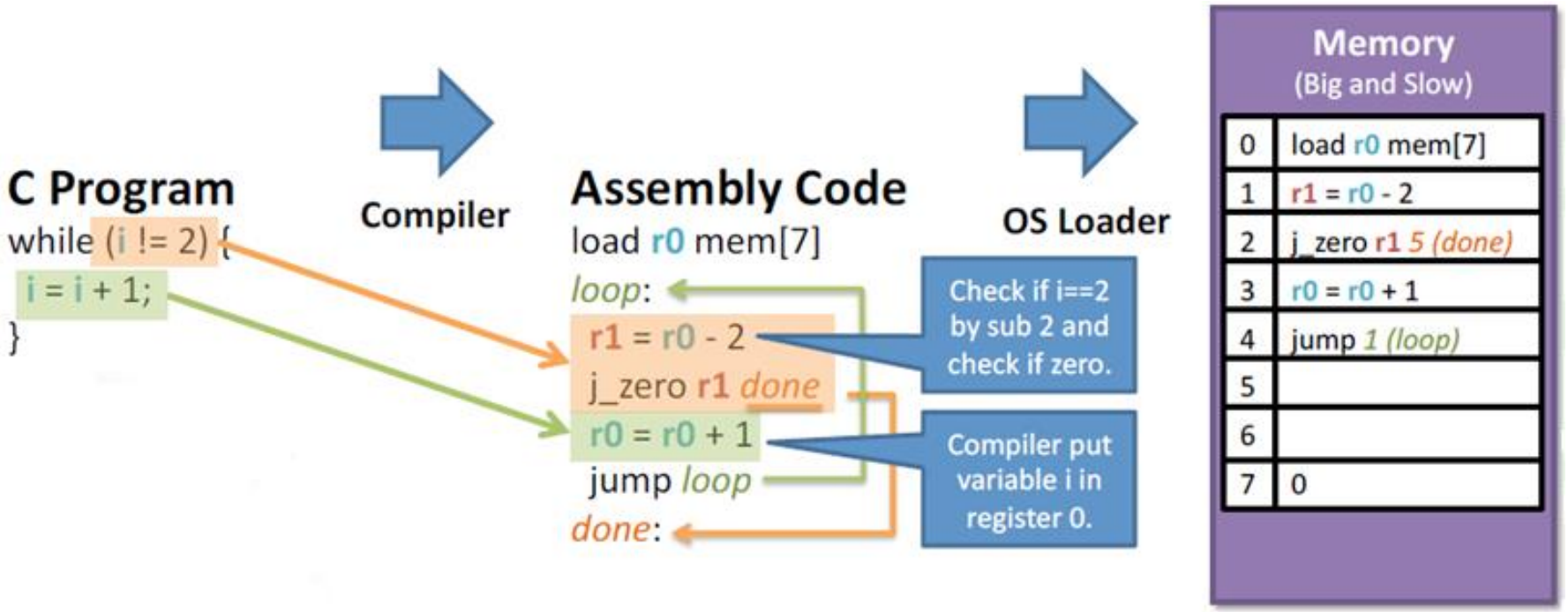


Below your program: Let's walk through the program

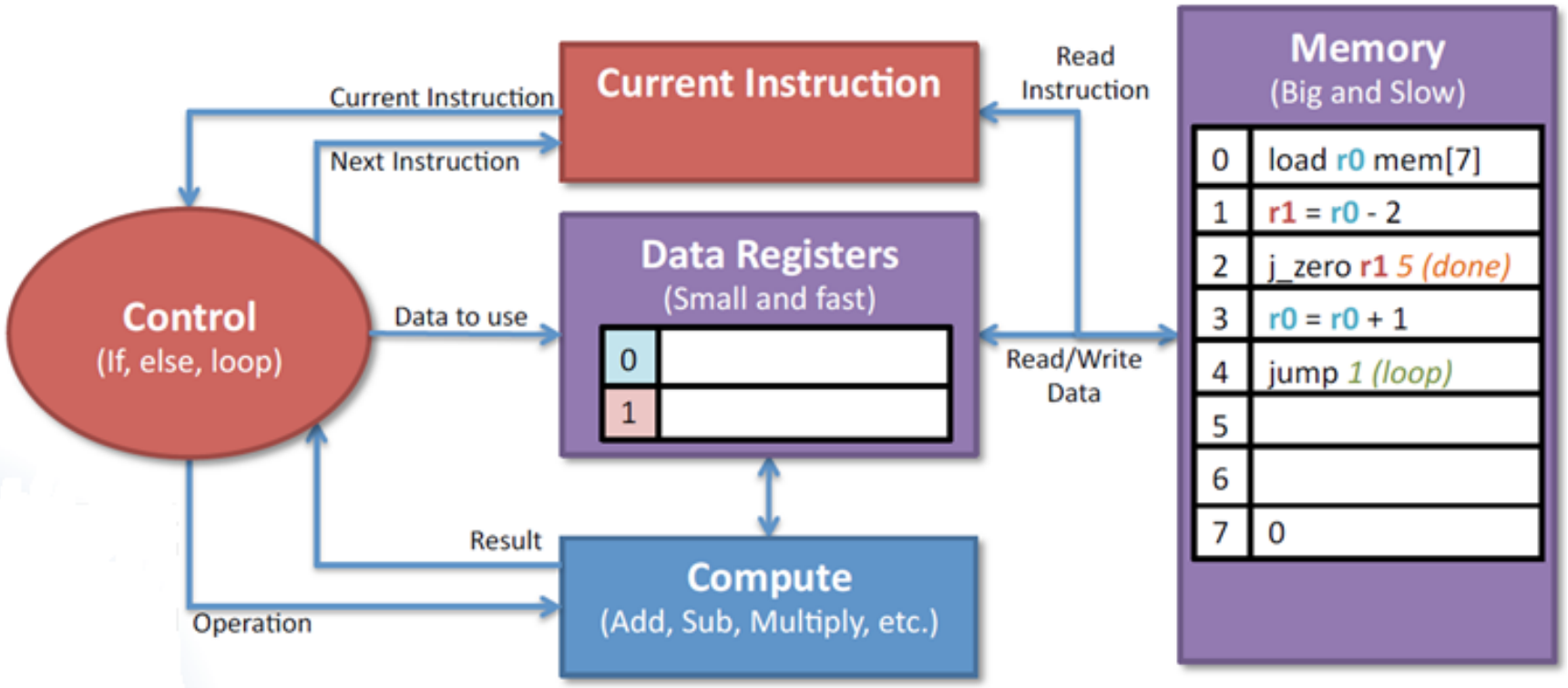
- What will the processor do?
  - **1. Load the instruction**
  - **2. Figure out what operation to do**
  - **3. Figure out what data to use**
  - **4. Do the computation**
  - **5. Figure out next instruction**
- Repeat this over and over and over…



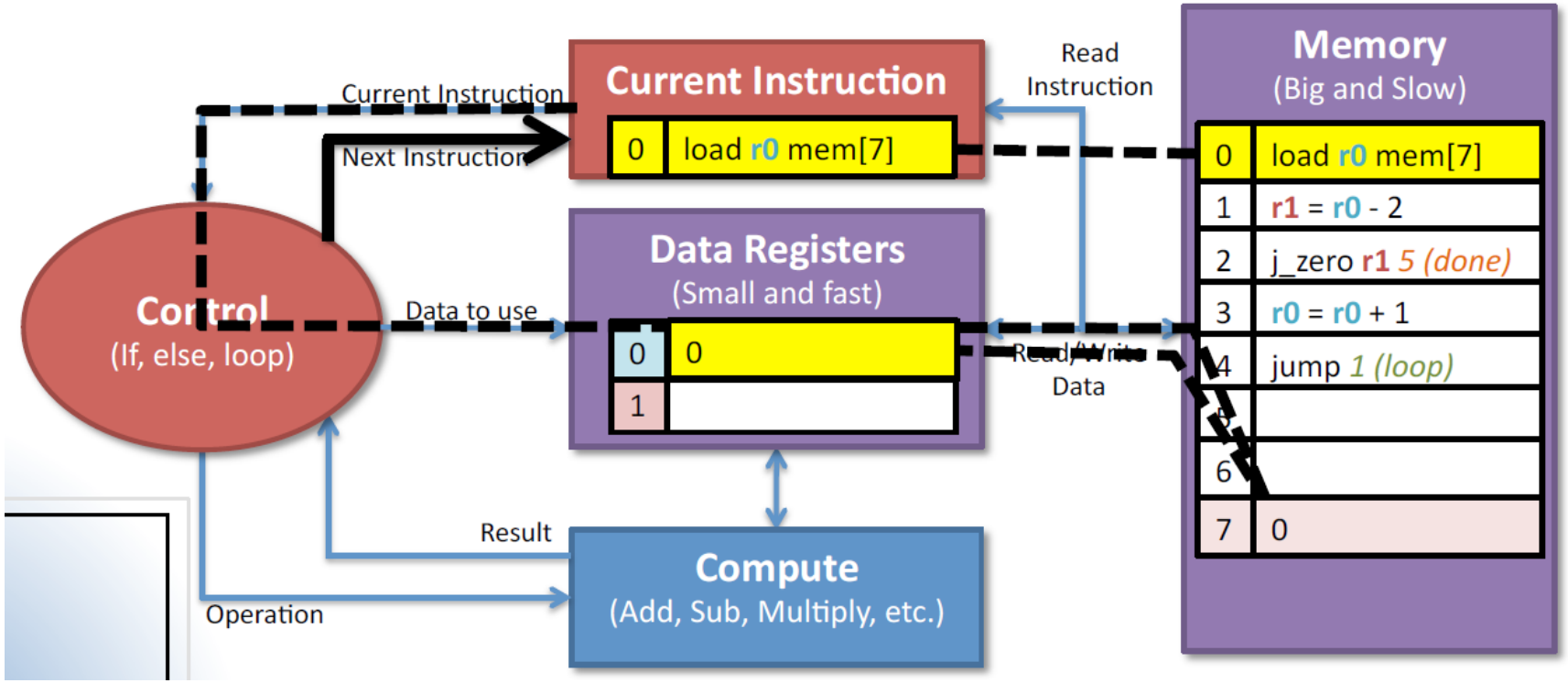
Example


C Program
while (i != 2) {
i = i + 1;
}
Compiler
Assembly Code
load r0 mem[7]
loop:
r1 = r0 - 2
j_zero r1 done
r0 = r0 + 1
jump loop
done:
Check if i==2 by sub 2 and check if zero.
Compiler put variable i in register 0.
OS Loader
Memory
(Big and Slow)
0 load r0 mem[7]
1 r1 = r0 - 2
2 j_zero r1 5 (done)
3 r0 = r0 + 1
4 jump 1 (loop)
5
6
7 0

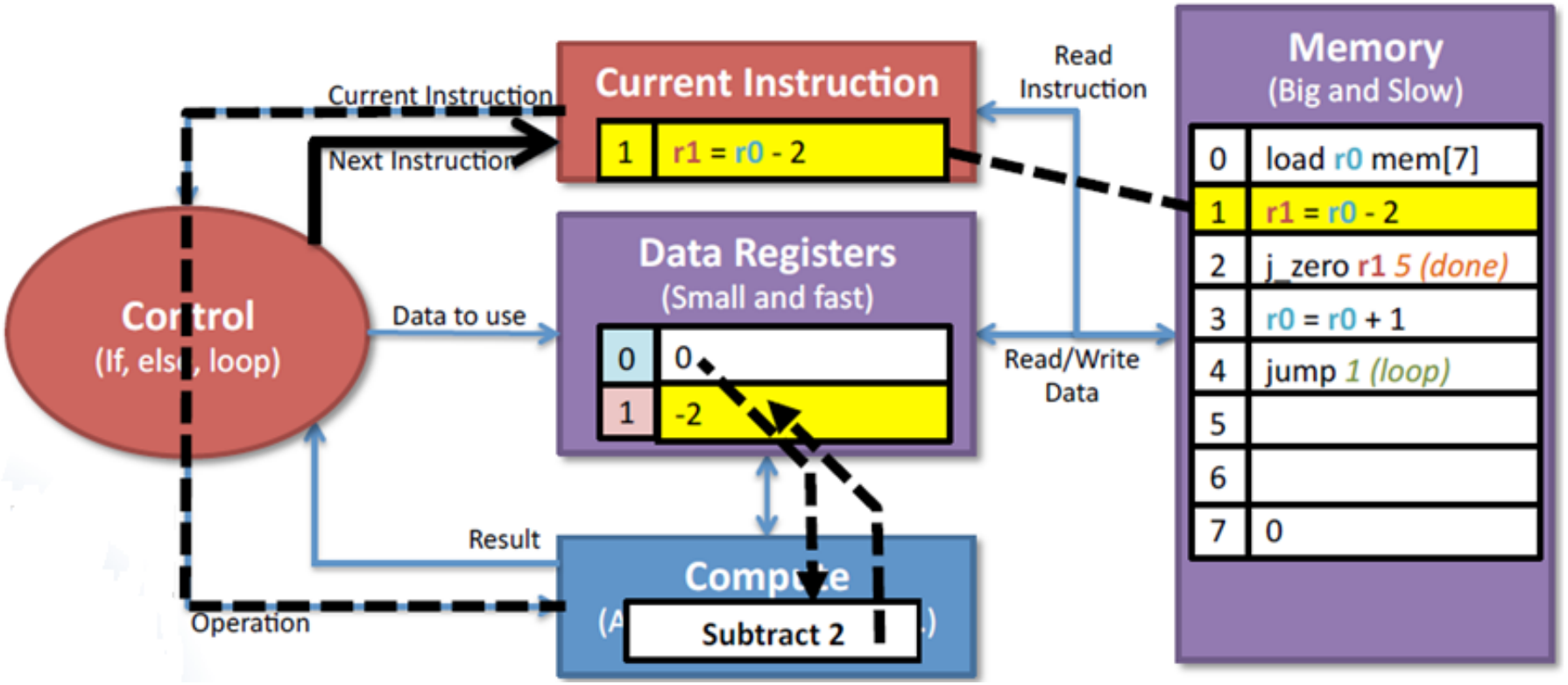
Basic processor: Memory, Control, and Compute



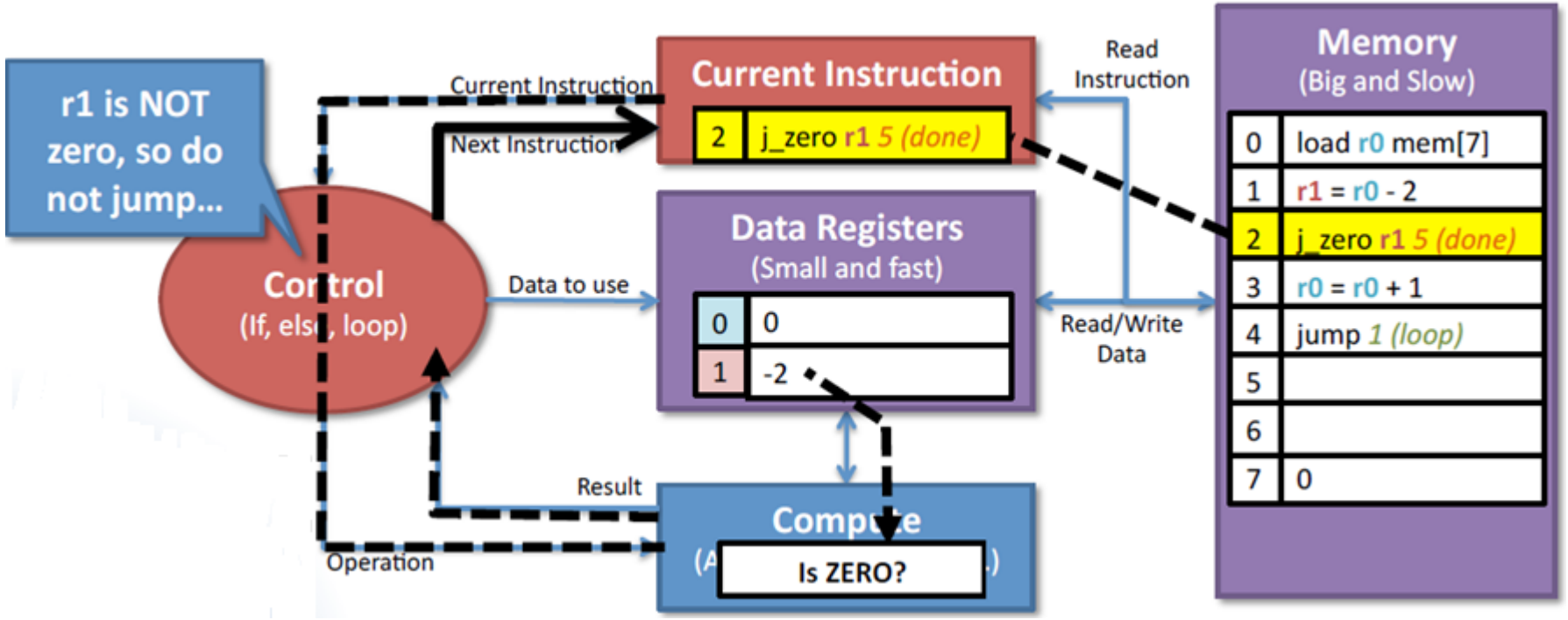
1: Load r0 (i) from memory (location 7)



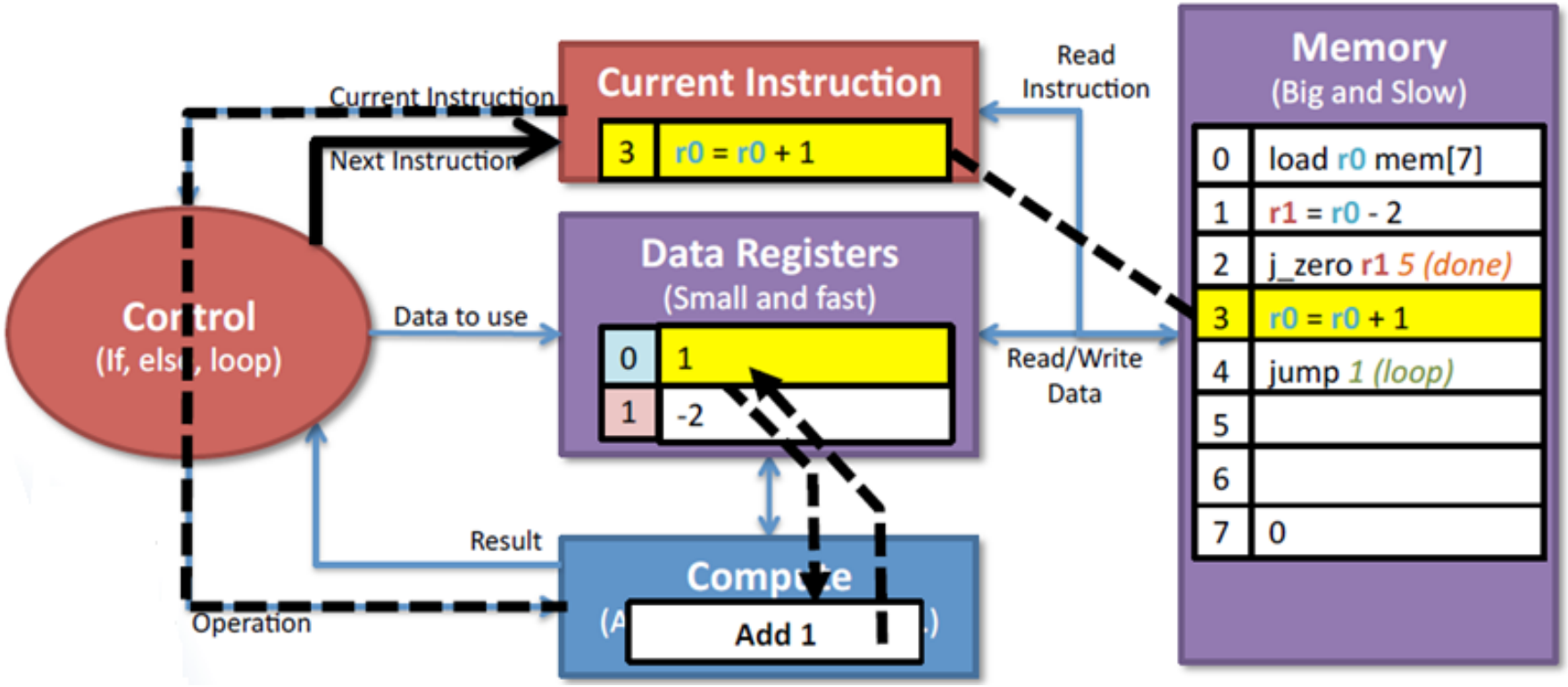
2: Subtract 2 from r0(i) to see if it is 2



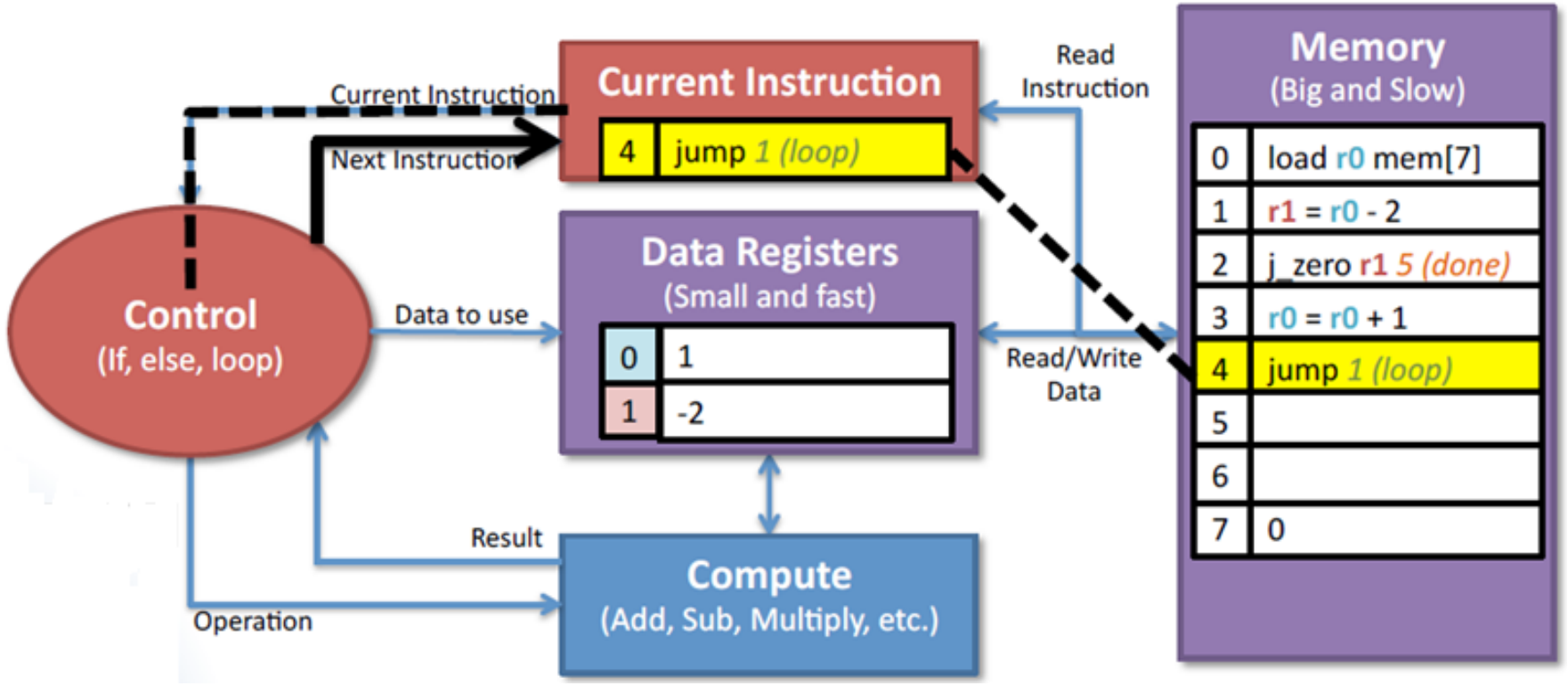
3: Check if r1 is zero 0, and jump to done if it is



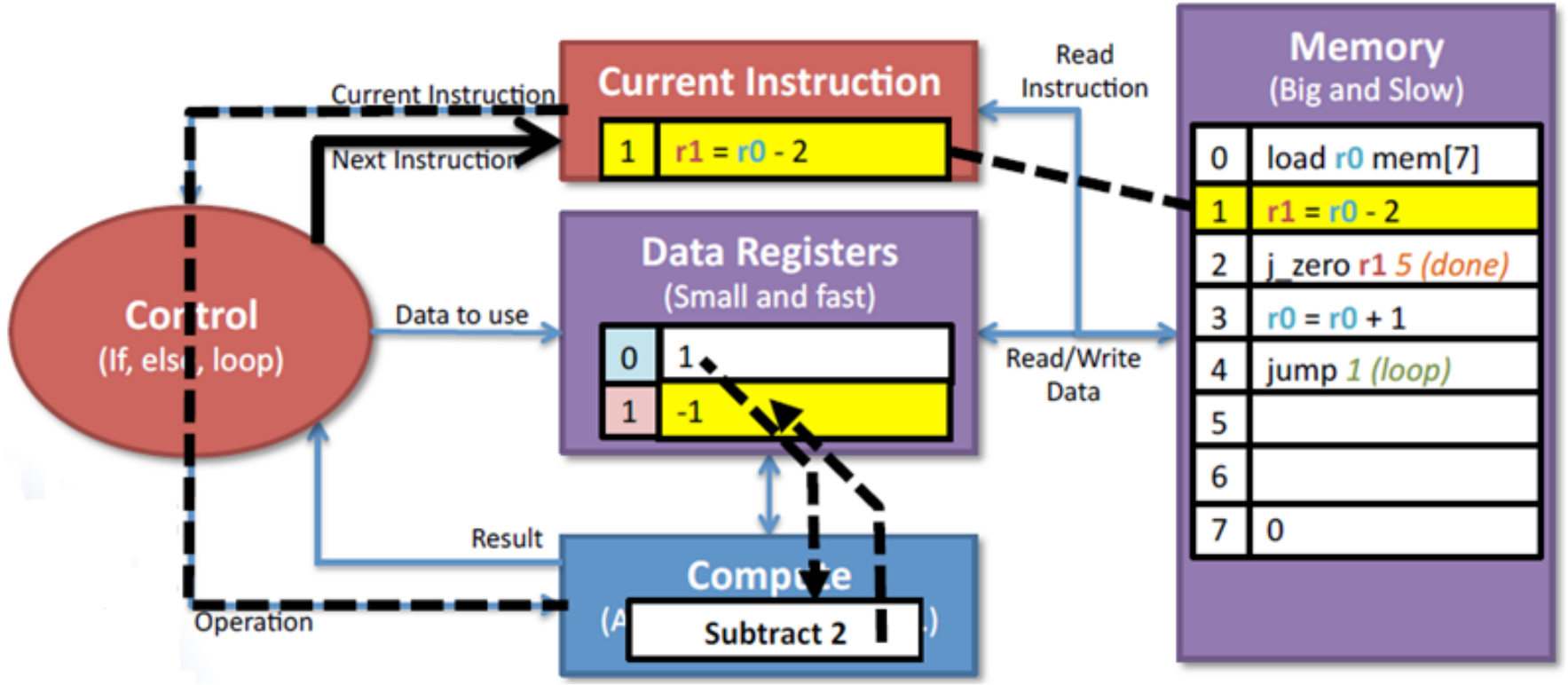
4: Increment r0 (i)



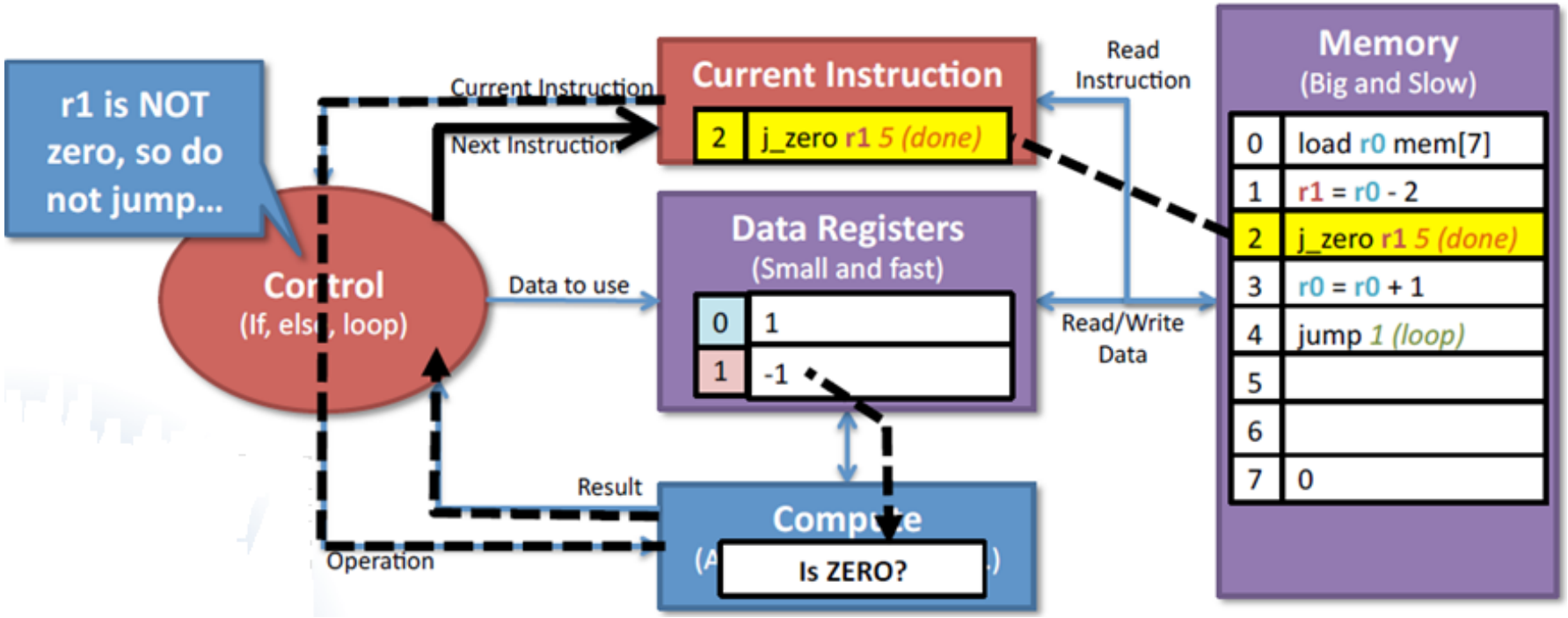
5: Continue the loop



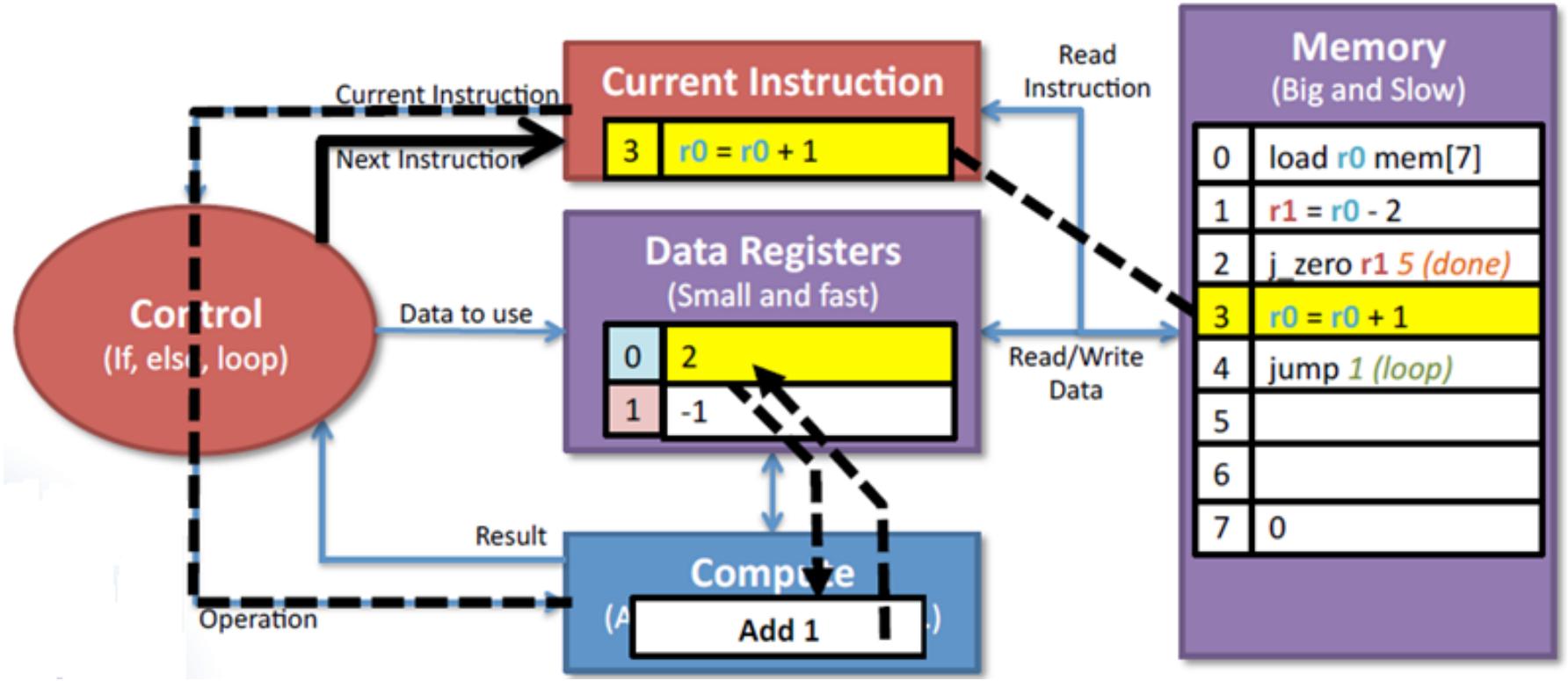
6: Subtract 2 from r0(i) to see if it is 2



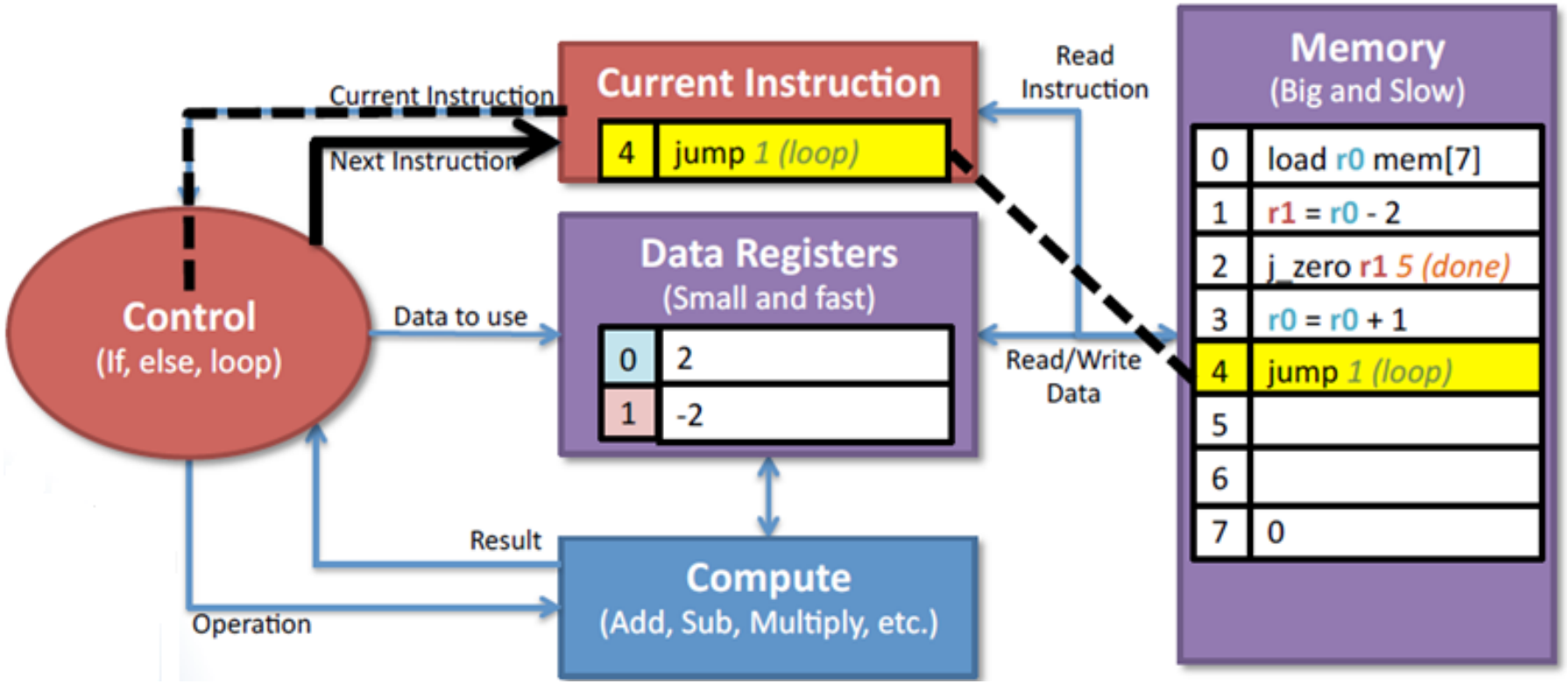
7: Check if r1 is zero 0, and jump to done if it is



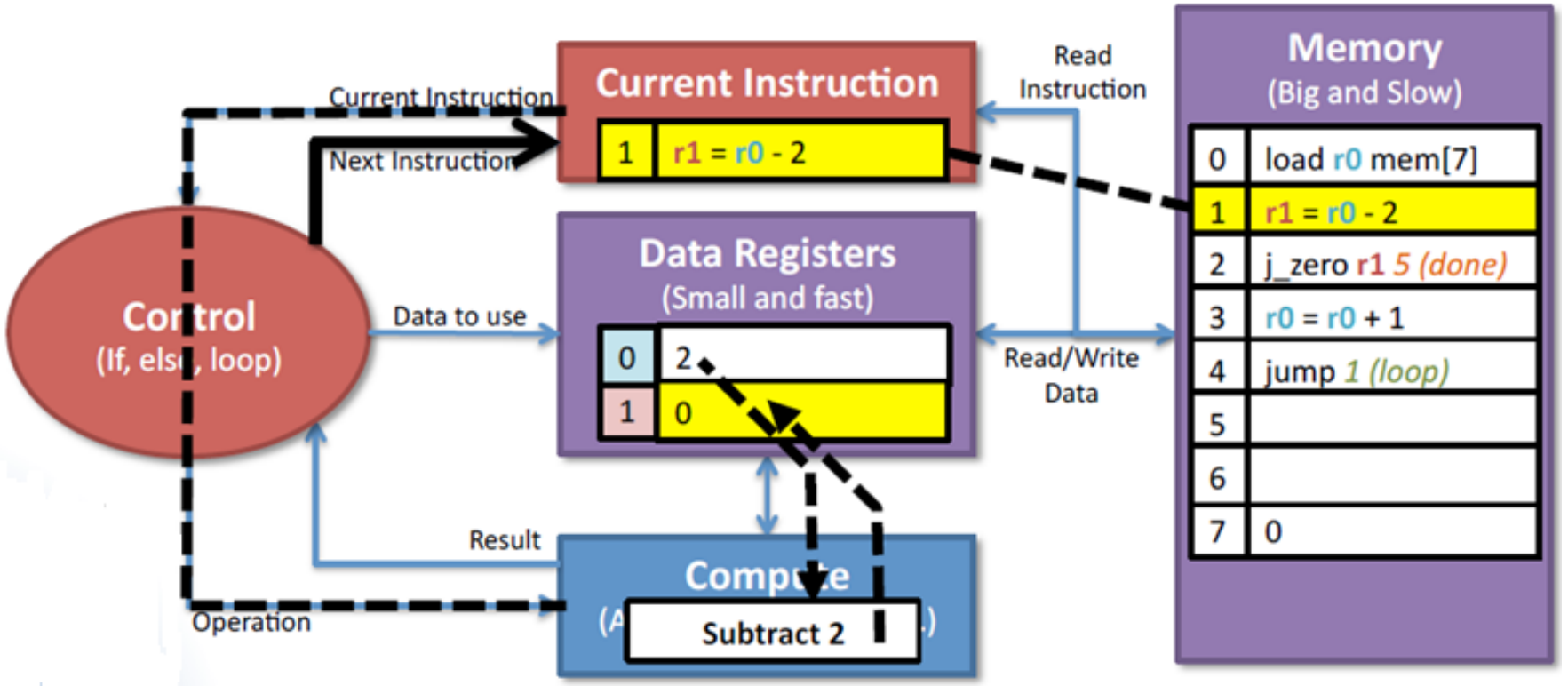
8: Increment r0 (i)



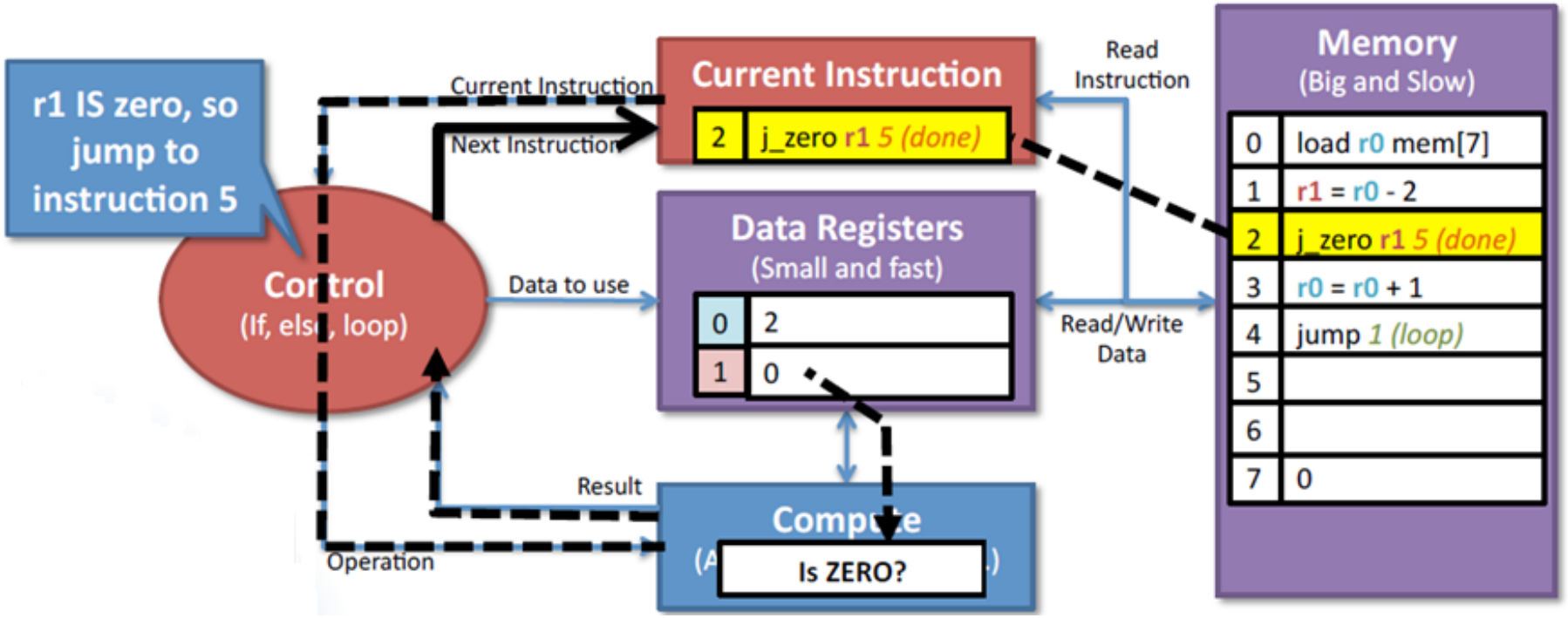
9: Continue the loop



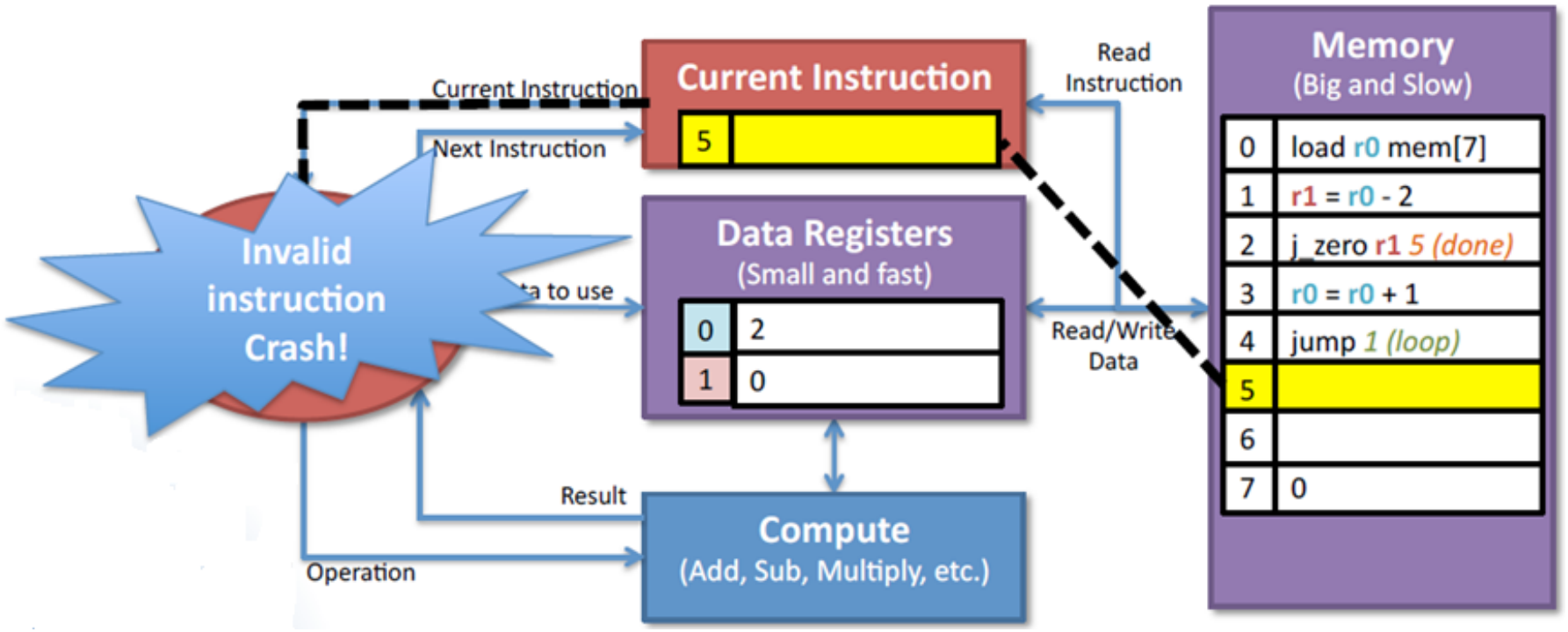
10: Subtract 2 from r0(i) to see if it is 2



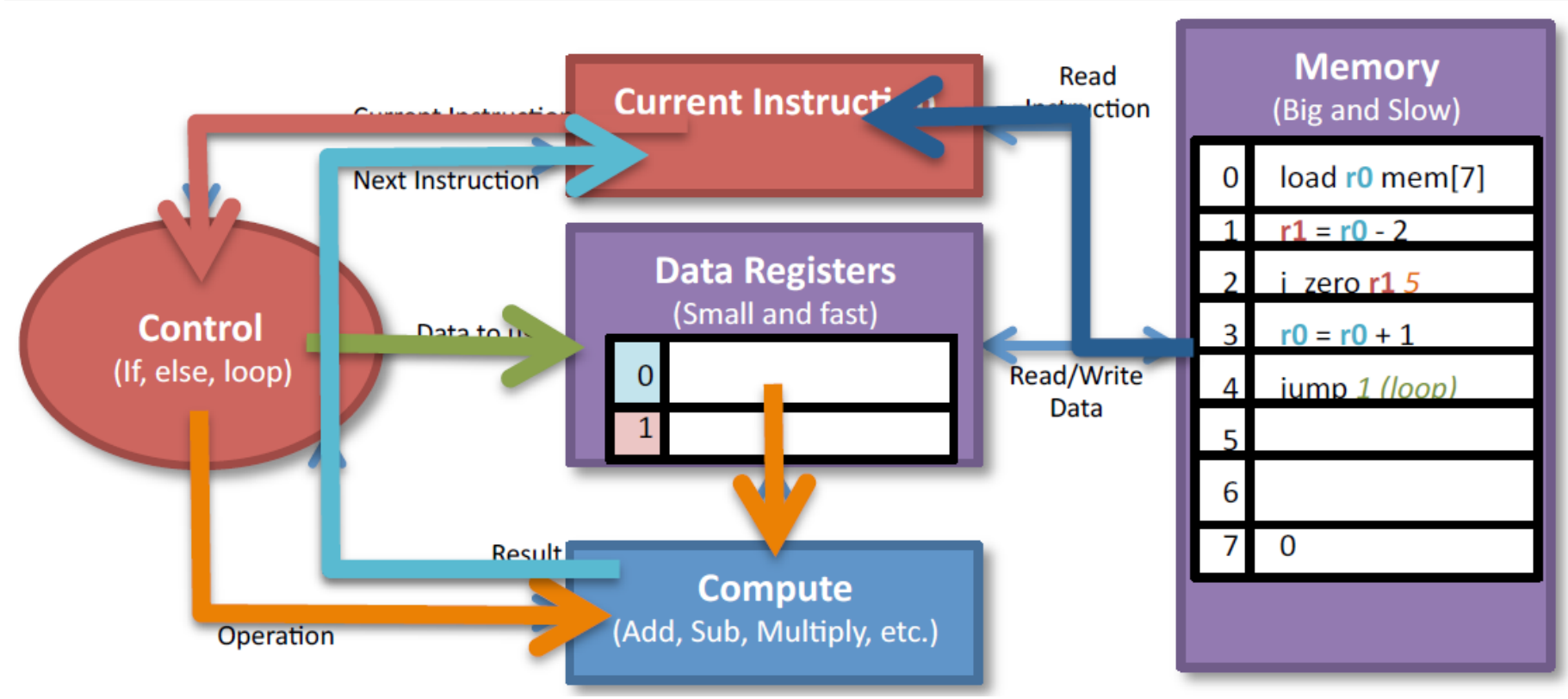
11: Check if r1 is zero 0, and jump to done if it is



12: Crash because the instruction 5 is invalid!

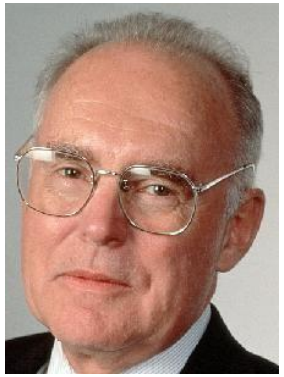


This course is about understanding the **details**, **corner cases**, **performance**, and how this all comes together in a RISC V processor.

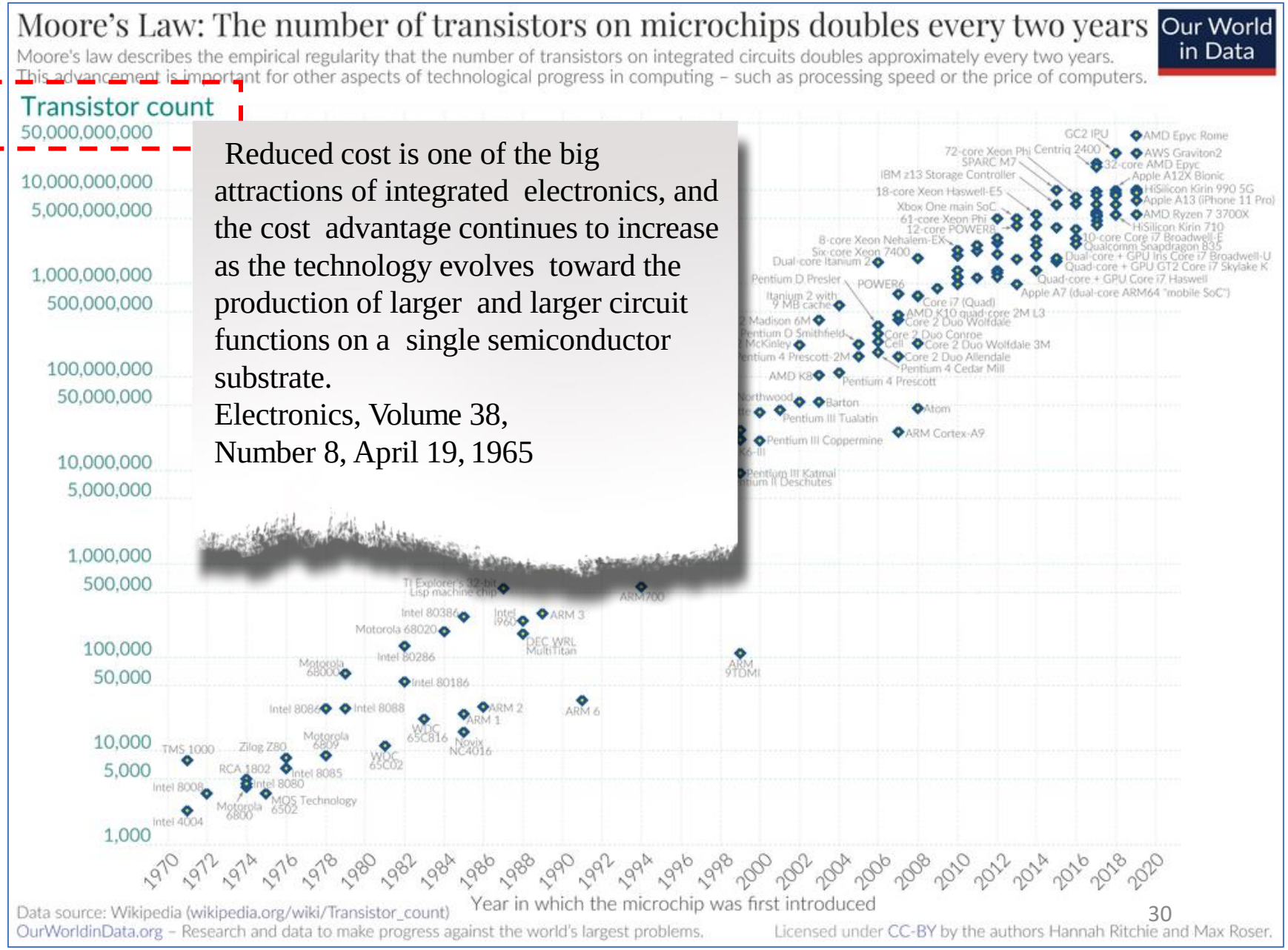


Great Idea #2: Moore's Law

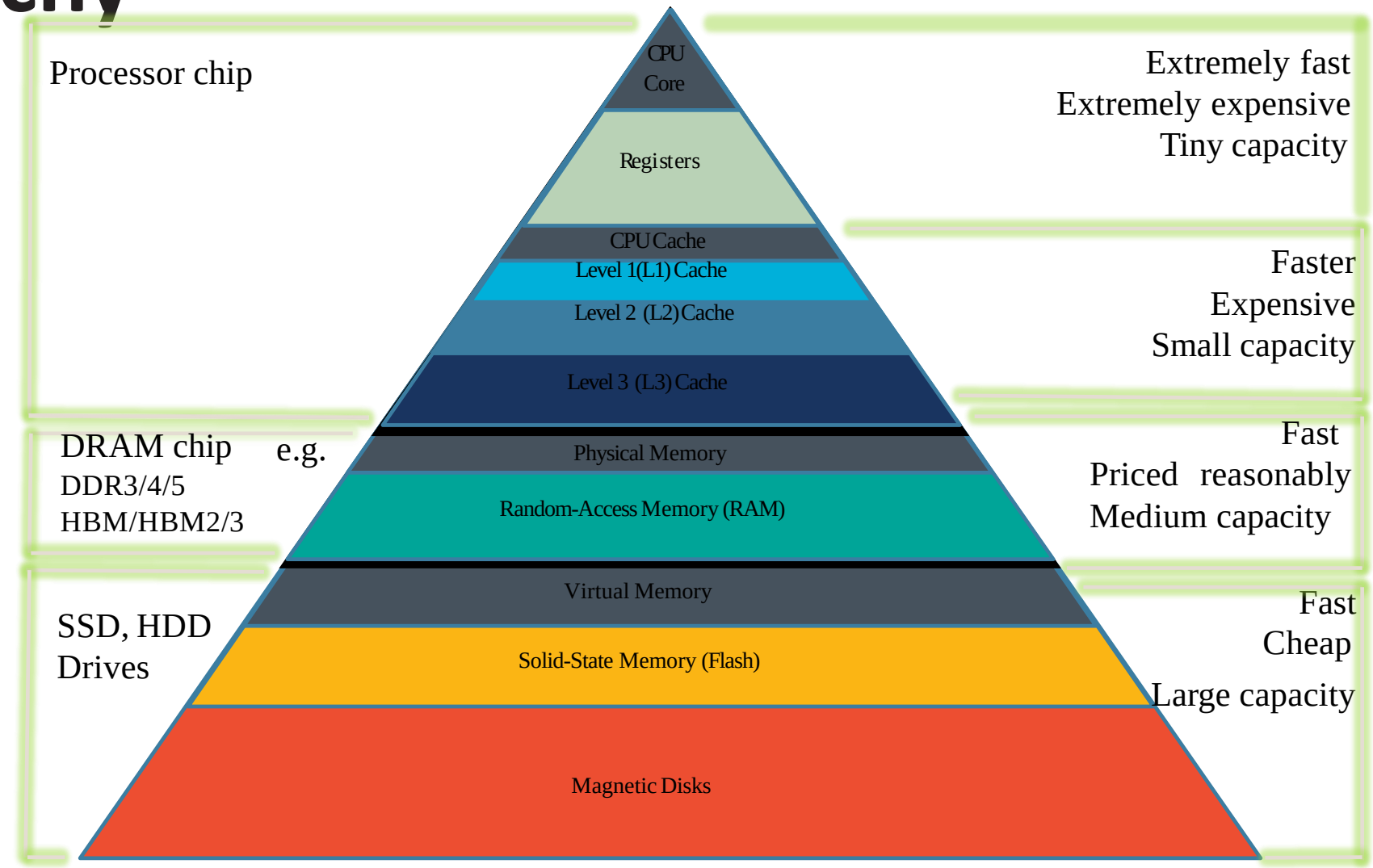
It states that integrated circuit resources double every 18–24 months



Gordon Moore
Intel Cofounder
B.S. Cal 1950!

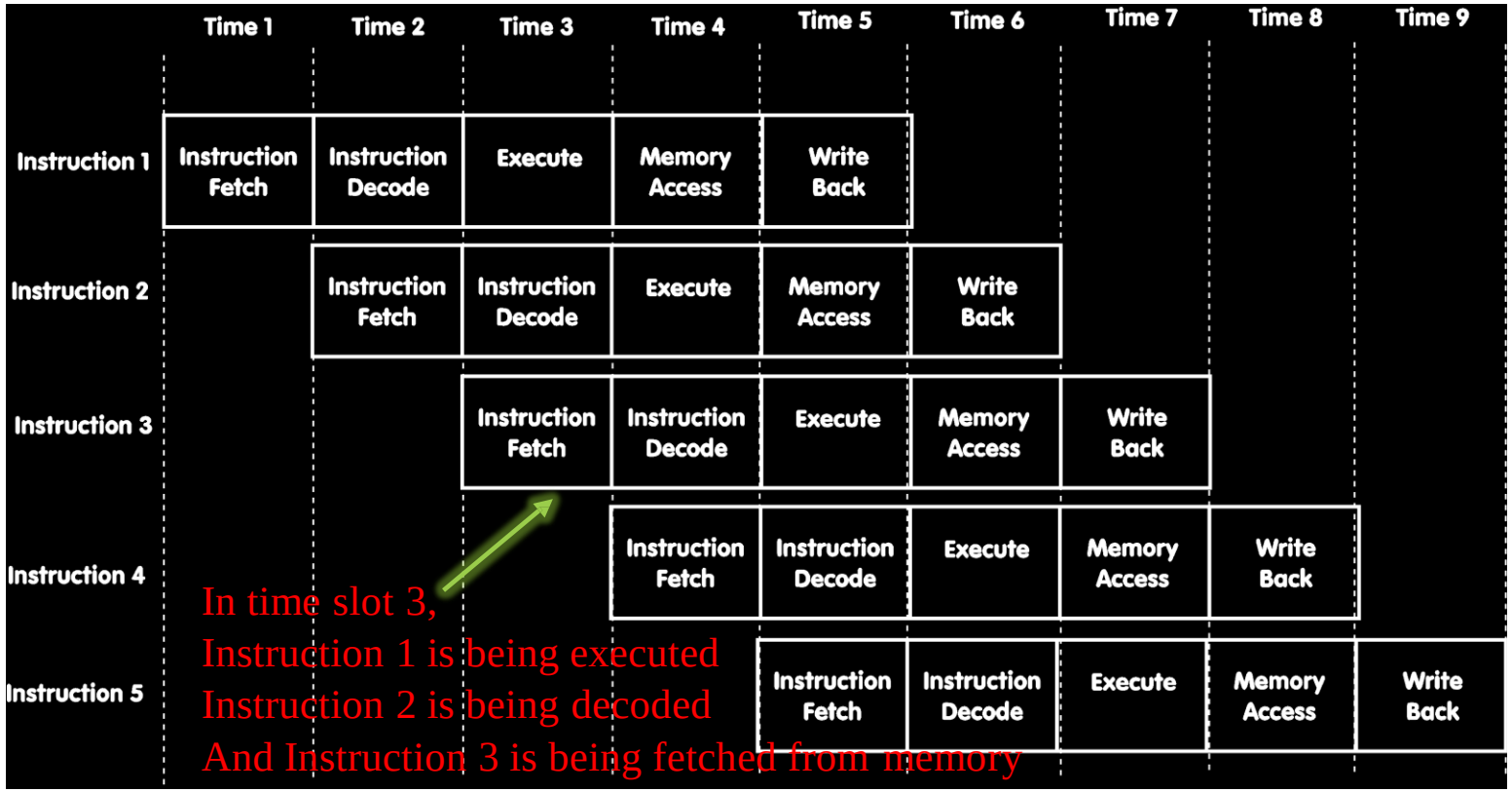


Great Idea #3 : Principle of Locality/Memory Hierarchy



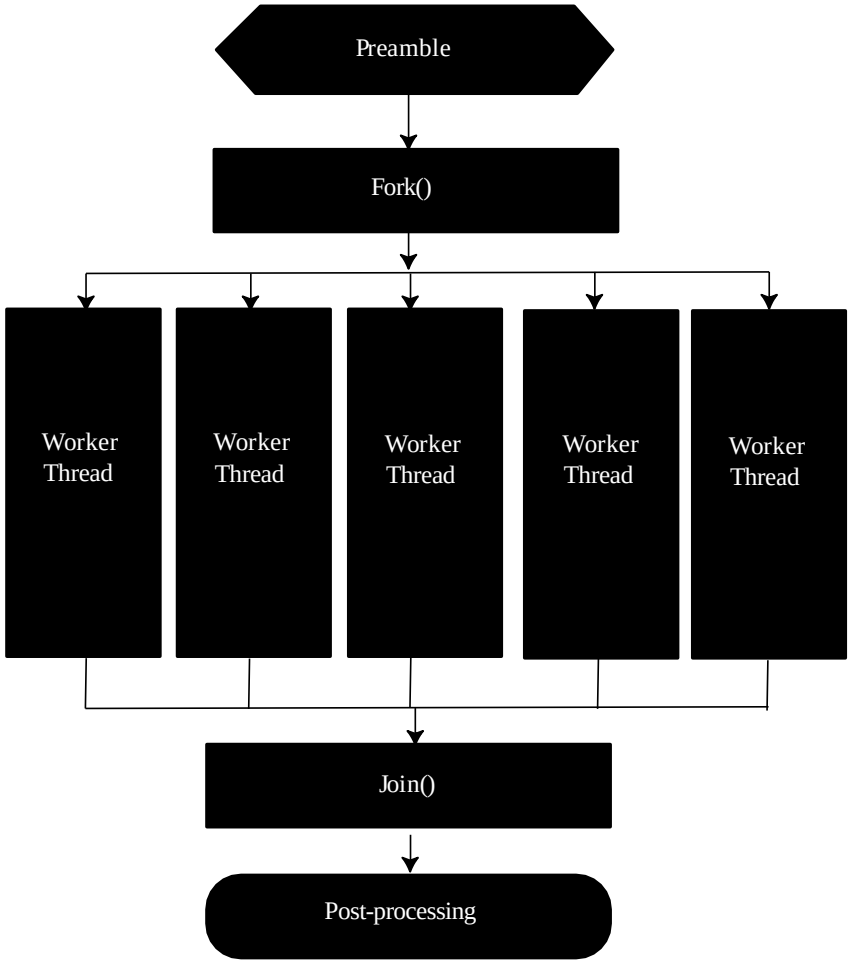
Great Idea #4: Parallelism (1/3)

Performance via Pipelining



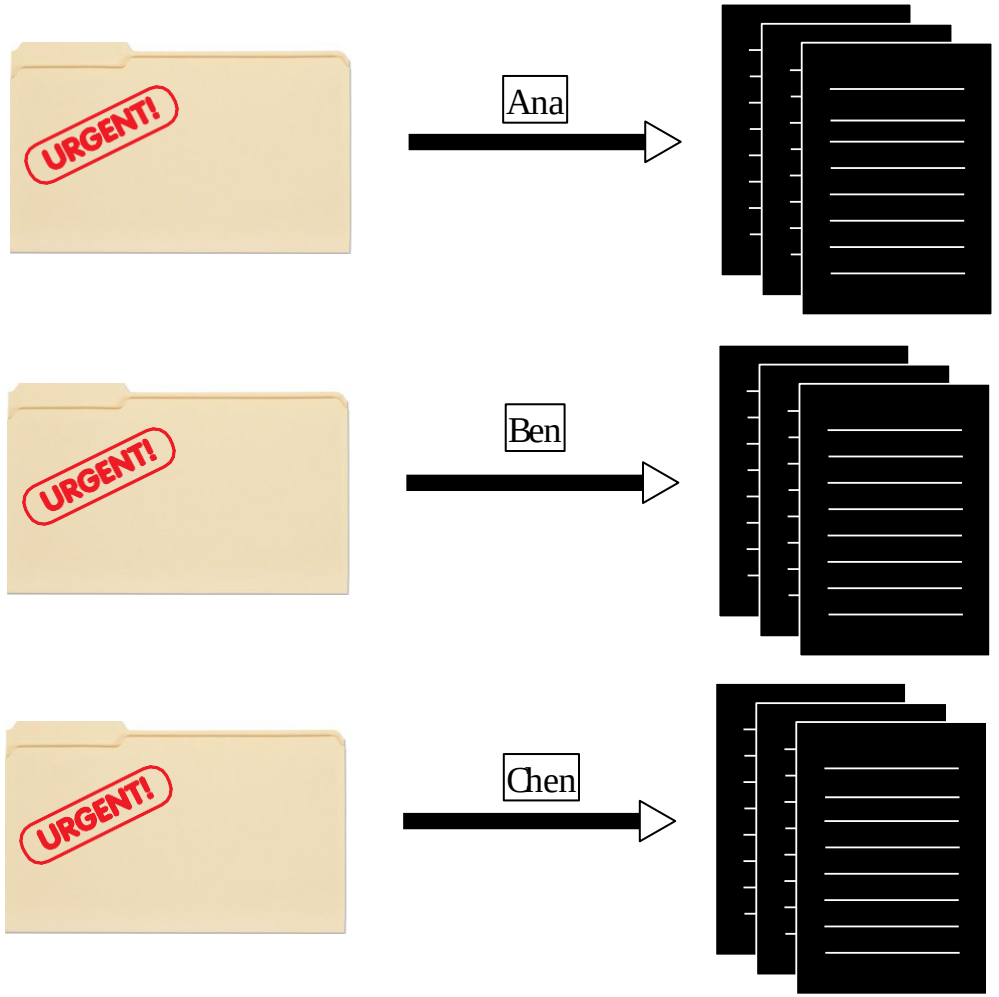
Great Idea #4: Parallelism (2/3)

Parallel Threads



Great Idea #4: Parallelism (3/3)

Parallel Processors



Caveat! Amdahl's Law

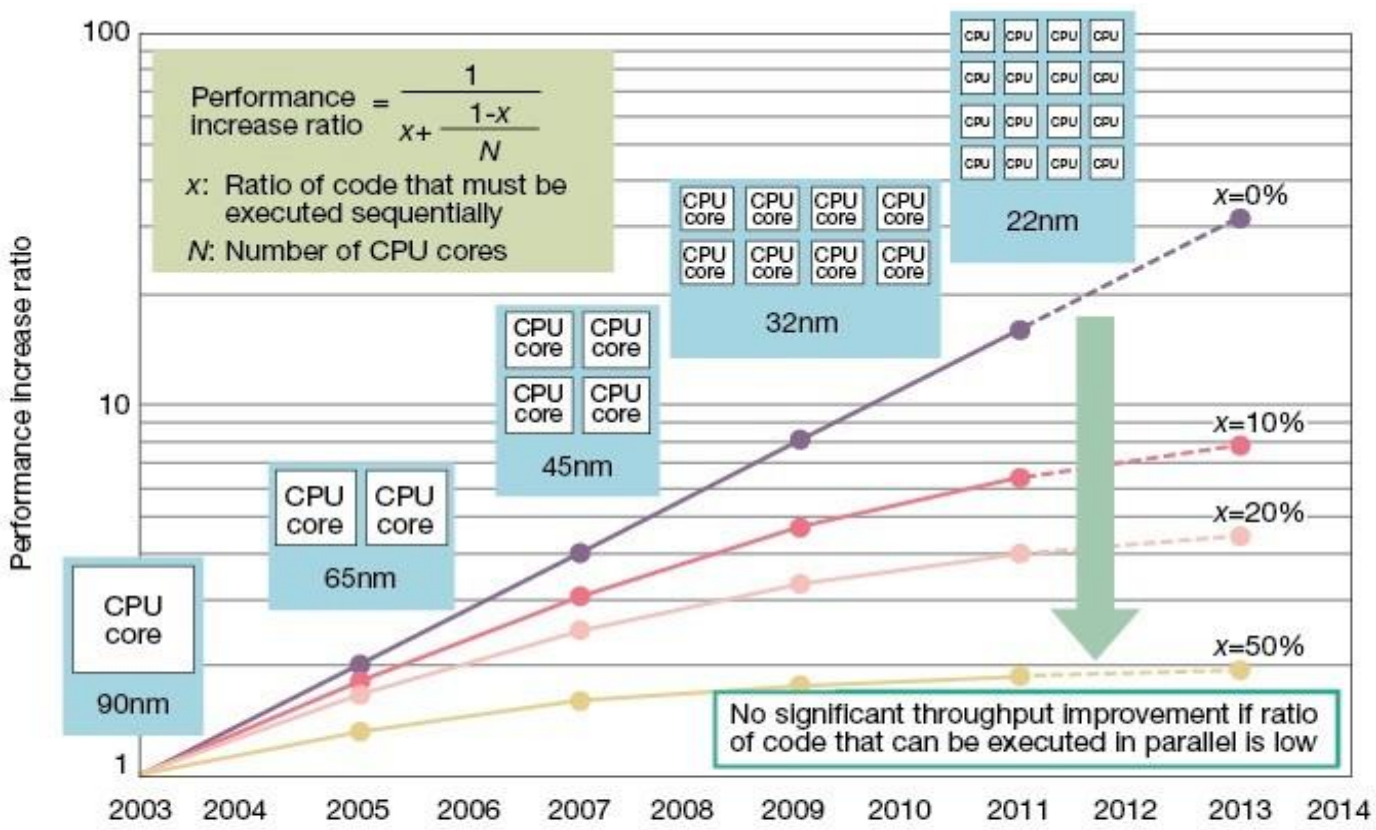


Fig 3 Amdahl's Law an Obstacle to Improved Performance Performance will not rise in the same proportion as the increase in CPU cores. Performance gains are limited by the ratio of software processing that must be executed sequentially. Amdahl's Law is a major obstacle in boosting multicore microprocessor performance. Diagram assumes no overhead in parallel processing. Years shown for design rules based on Intel planned and actual technology. Core count assumed to double for each rule generation.

Gene Amdahl

Computer Pioneer

Great Idea #5: Performance Measurement and Improvement

- How hardware and software affect performance ?

Hardware or software component	How this component affects performance
Algorithm	Determines both the number of source level statements and the number of I/O operations executed
Programming language, compiler, and architecture	Determines the number of computer instructions for each source level statement
Processor and memory system	Determines how fast instructions can be executed
I/O system (hardware and operating system)	Determines how fast I/O operations may be executed

Great Idea #5: Performance Measurement and Improvement

- Matching application to underlying hardware to exploit:
  - Locality
  - Parallelism
  - Special hardware features, like specialized instructions (e.g., matrix manipulation)
- Latency/Throughput
  - How long to set the problem up and complete it (or how many tasks can be completed in given time)
  - How much faster does it execute once it gets going
  - Latency is all about time to finish

Great Idea #6: Dependability via Redundancy

- Redundancy so that a failing piece doesn't make the whole system fail
- Applies to everything from datacenters to storage to memory to instructors
  - Redundant datacenters so that can lose 1 data center, but Internet service stays online
  - Redundant disks so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - Redundant memory bits of so that can lose 1 bit but no data (Error Correcting Code/ECC Memory)

Homework

- Reading Technologies for Building Processors and Memory (Section 1.5 #page 24) and writing the report