中国科学院大学
University of Chinese Academy of Sciences

CS101

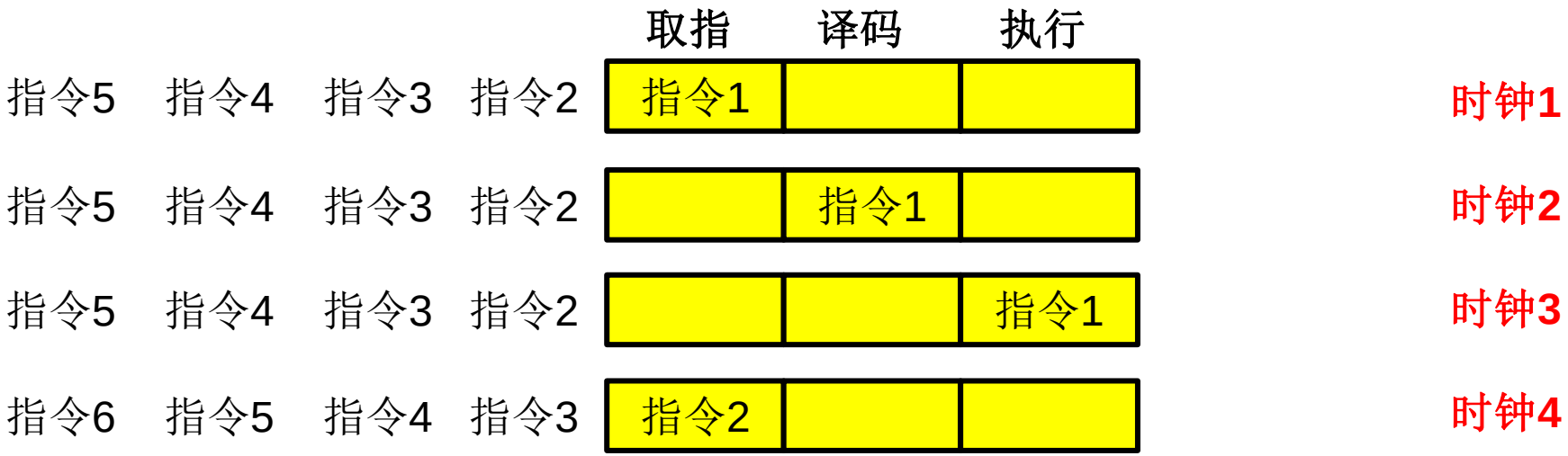
Systems Thinking

Seamless Transition-1:
The four principles of seamless transition
无缝衔接的四个原理

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

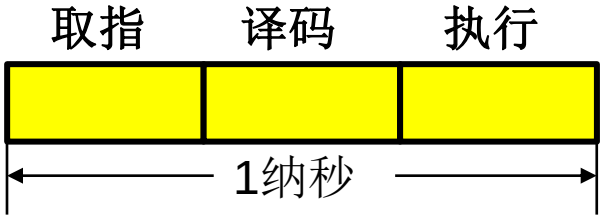
指令周期

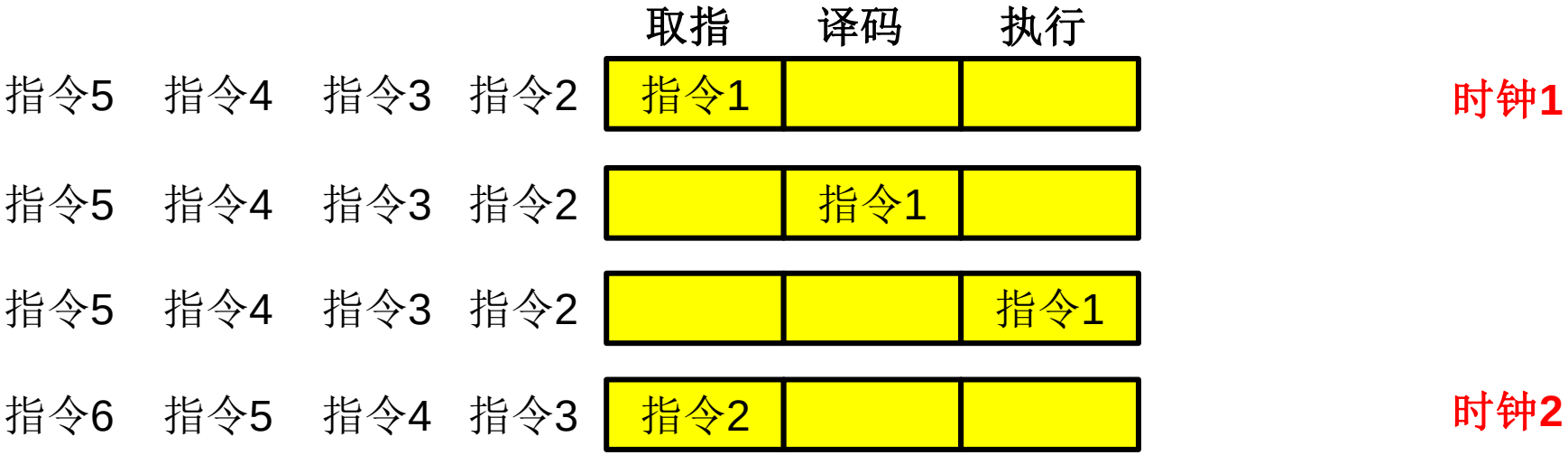
- 每个指令周期包含三个操作阶段（步骤）
  - 例如，指令MOV R1, M[R0]执行三步骤如下：
    - 取指操作：IR ← M(PC); PC←PC+1
    - 译码操作：Signals = Decode(IR)
    - 执行操作：M[R0] ← R1



指令流水线

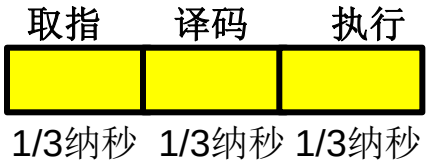
- 假设指令流水线总延时是1纳秒，流水线各阶段没有重叠（No overlap）
  - 执行完当前指令，再执行下一条指令
    - 处理器时钟周期 = 1纳秒
    - 处理器主频（frequency）= 1/(1纳秒) = 1 GHz
    - 峰值速度为1 GIPS
      (Giga Instructions Per Second)
      即每秒十亿条指令

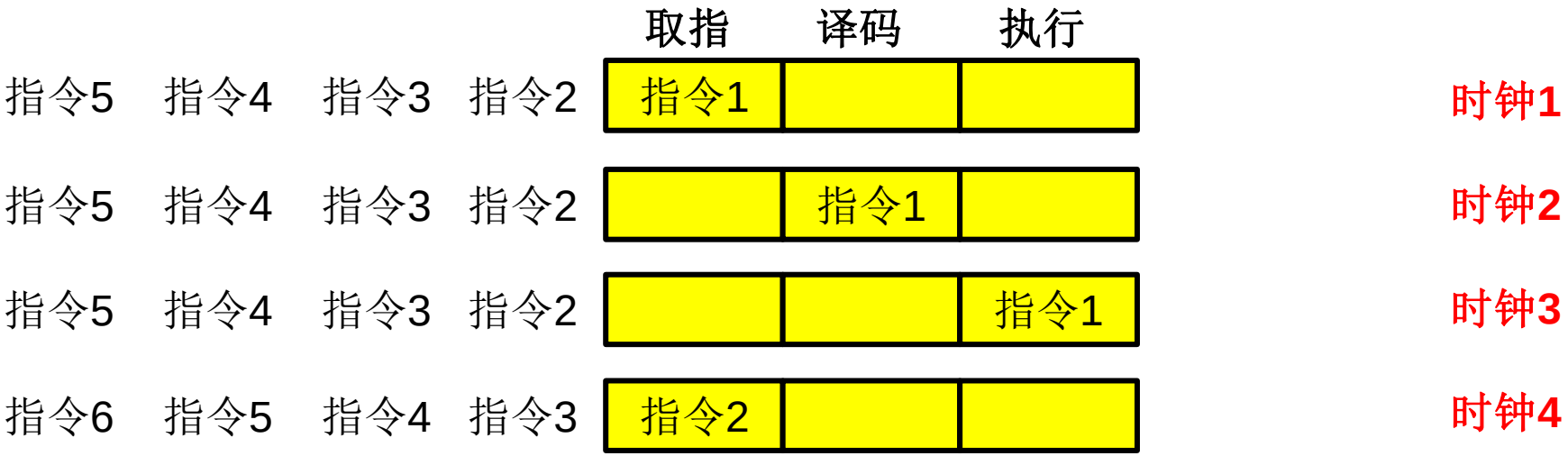




指令流水线

- 假设指令流水线总延时是1纳秒，流水线各阶段没有重叠（No overlap）
  - 执行完当前指令，再执行下一条指令
    - 处理器时钟周期 = (1/3)纳秒
    - 处理器主频（frequency）= 1/(1/3) = 3 GHz
    - 执行一条指令需要三个时钟周期
    - 峰值速度为1 GIPS (Giga Instructions Per Second)
      即每秒十亿条指令

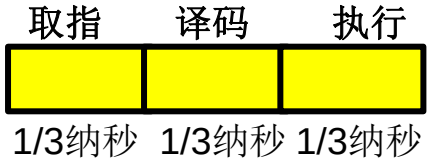




指令流水线机制总是利用重叠（overlap）

处理器主频= 3 GHz，指令流水线同时执行三条指令，平均每个时钟周期执行完毕一条指令

峰值速度：3 Giga Instructions Per Second (GIPS)，即每秒执行30亿条指令



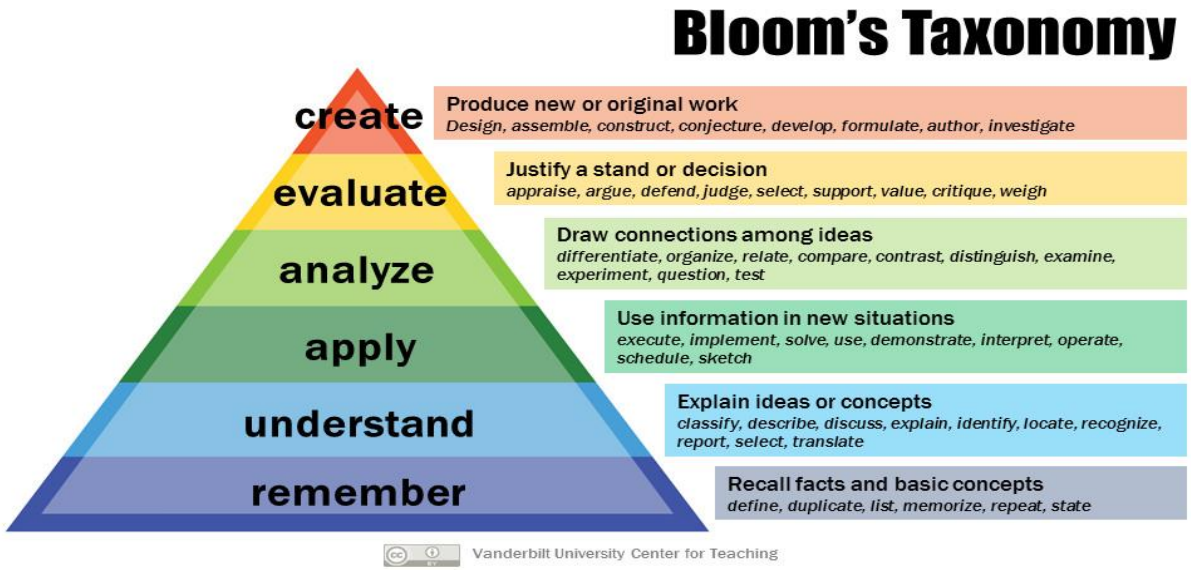
- 假设*K*级指令流水线总延时是1纳秒，则有
  - 处理器时钟周期是每级延时 = 1/*K* 纳秒
  - 处理器主频 = *K* GHz
  - 指令流水线同时执行*K*条指令，平均每个时钟周期执行完毕一条指令
  - 处理器峰值速度：*K* GIPS

回答同学们的若干问题

- 四个时间点（前三个是截止日期，可以提前提交）
  - 7月7日：提交个人作品的动态网页代码
  - 7月11日：提交最后一次作业
  - 7月14日：提交个人作品报告（可以完善动态网页代码）
    - 如改了代码，请注明完善了什么
  - 7月16日上午10-12点：期末考试
- 期末考试相关
  - 7月9日上课：期末复习
  - 期末考试内容范围
    - 整个课程，1-6章
    - 上课讲过的内容，也是期末复习重温的内容（注意：举一反三）
    - 中英文教科书都有的内容（少数例外）
    - 不会考个人作品涉及的编程内容
  - 教学团队周末值班

布鲁姆教学目标分类法应用到本课程

- 知道：能够区分概念
  - 组合电路、时序电路
- 了解：能够解决一个实例问题
- 掌握：学到了原理，能够举一反三
  - 时序电路做加法、做减法



无缝衔接

- 自动执行高阶：无缝衔接
  - 确保正确性的计算过程归纳法
    - 正确地确定第一步
    - 针对每个单一步骤，保证该步骤正确执行
    - 每一步骤执行完毕，
      正确地确定并衔接到下一步骤
  - 应对计算过程的瓶颈

- Yang's cycle principle
  扬雄周期原理
- Postel's robustness principle
  波斯特尔鲁棒原理
- von Neumann's exhaustiveness principle
  冯诺依曼穷举原理
- Amdahl's law
  阿姆达尔定律

- 自动执行难题尚未完全解决
  - 基本解决了单机自动执行难题
  - 在多台计算机上执行的计算过程如何自动执行？
  - 在人机物三元计算的万物互联网时代如何自动执行？
    - 网络思维（名字空间、拓扑、协议栈）
    - IEEE CS 2022报告：无缝智能

无缝智能尚未实现

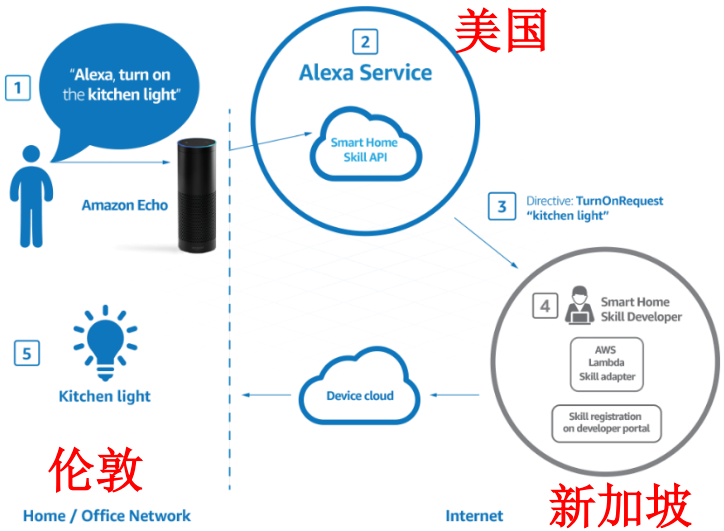
- 不只是简单的数值计算，而是智能计算
  - 计算机能够：理解、感知、认知，甚至呈现情感、美感、意识
- 无缝：用户体验流畅

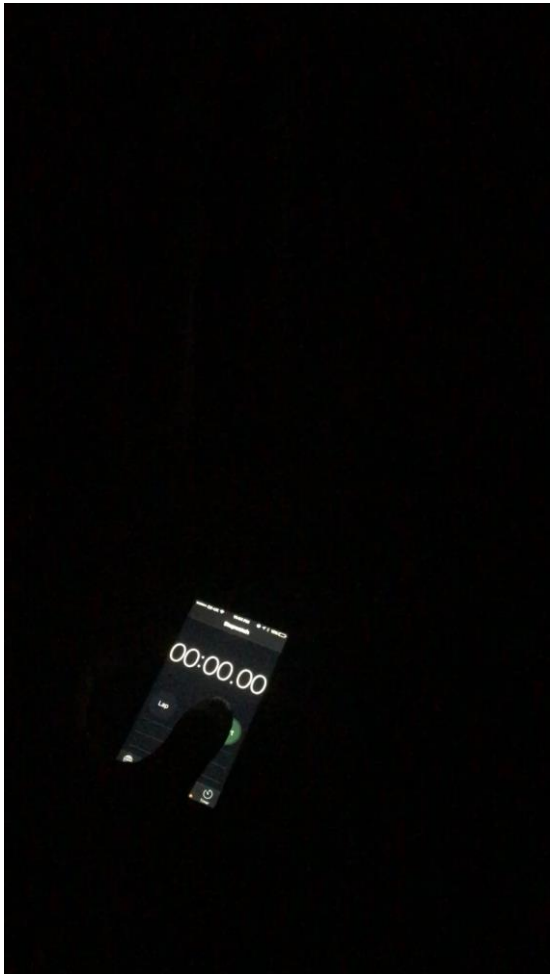
案例：执行“请开灯”命令（2017.4.13实验）

世界领先的Amazon Echo 物端系统
响应延迟：3-5秒

心理学准则：
用户体验到无缝智能
→响应延迟<0.12秒

中科院计算所
物端系统研究目标
响应延迟 < 0.1秒





扬雄周期原理
优于尼采的永恒轮回：die ewige Wiederkunft

- 汉代扬雄所著的《太玄经》
  - 《太玄经•周首》：“阳气周神而反乎始，物继其汇”
    - 宋代司马光诠释道：“岁功既毕，神化既周”
- 扬雄周期原理体现组合性（composability）方法
  - 执行完一个XX周期，周而复始执行下一个XX周期
    - 计算过程由**程序**周期组合而成
    - 程序周期由**指令**周期组合而成
    - 指令周期由**时钟**周期组合而成
  - 一个时钟周期的操作对应于一个自动机的变换
  - XX周期具备可数个类别
    - 例如，指令周期有内部周期、总线周期

Outline

- What is systems thinking?
- Three objectives of systems thinking
- Abstraction
- Modularization
- Seamless transition
  - The symphony of four principles — 无缝衔接的四条原理
  - Yang's cycle principle — 扬雄周期原理
  - Postel's robustness principle — 波斯特尔鲁棒性原理
  - von Neumann's exhaustiveness principle — 冯诺依曼穷举原理（穷尽原理）
  - Amdahl's law — 阿姆达尔定律
  - Landscape of computing systems

These slides acknowledge sources for additional data not cited in the textbook

5.1 The symphony of four principles

- 1-minute quiz
  - Q: Why can two students in San Jose and Shenzhen conduct a video talk online correctly? Please give a specific principle.
    - Why trillions of instructions can be automatically executed in a fraction of a second, across the globe, to produce correct computational results?

    - 为什么两位学生，位于硅谷和深圳，可以完成一个协同计算任务，例如视频对话？
    - 为什么他们相隔几千里，却可以在一秒之内完成协同计算任务，例如传递一个视频表情（笑容）？

    - 原理上是如何做到的？

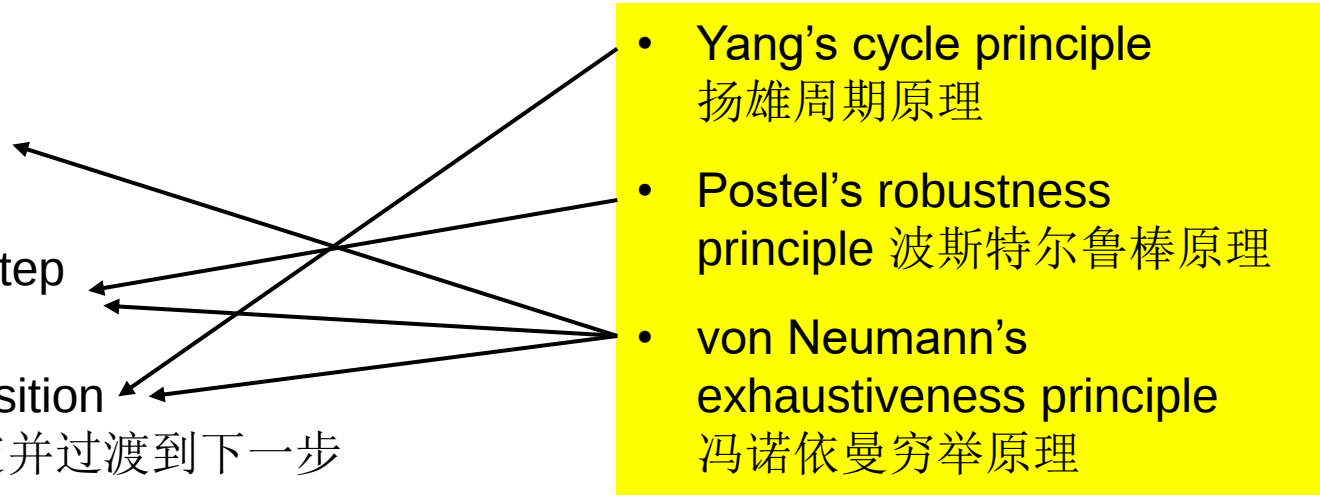
    - 先不管速度，正确性是如何保证的？（功能实现）
    - 速度是如何保证的？（性能实现）

5.1 The symphony of four principles
四条无缝衔接原理的交响乐

- 1-minute quiz
  - A. The computers involved in the video talk execute their computational processes correctly and smoothly
    多个计算机正确且无缝地执行计算过程
  - More concretely, for each computational process involved, do a computational induction (similar to mathematic induction)
    - Ensure that the first step is correctly identified
    - Ensure that any identified step (i.e., any single step) is correctly executed
    - For each step just finishing execution, ensure that the correct next step is identified and the current step correctly transition to the next step
  - 计算归纳法（只考虑一个串行计算过程）
    - 正确地确定第一步
    - 针对每个单一步骤（一个确定的步骤），保证该步骤正确执行
    - 每一步骤执行完毕，正确地确定并衔接到下一步骤
  - A step could be a program, an instruction, a gate, etc.
    - 步骤 = 程序、指令、逻辑门等 （各种粒度）

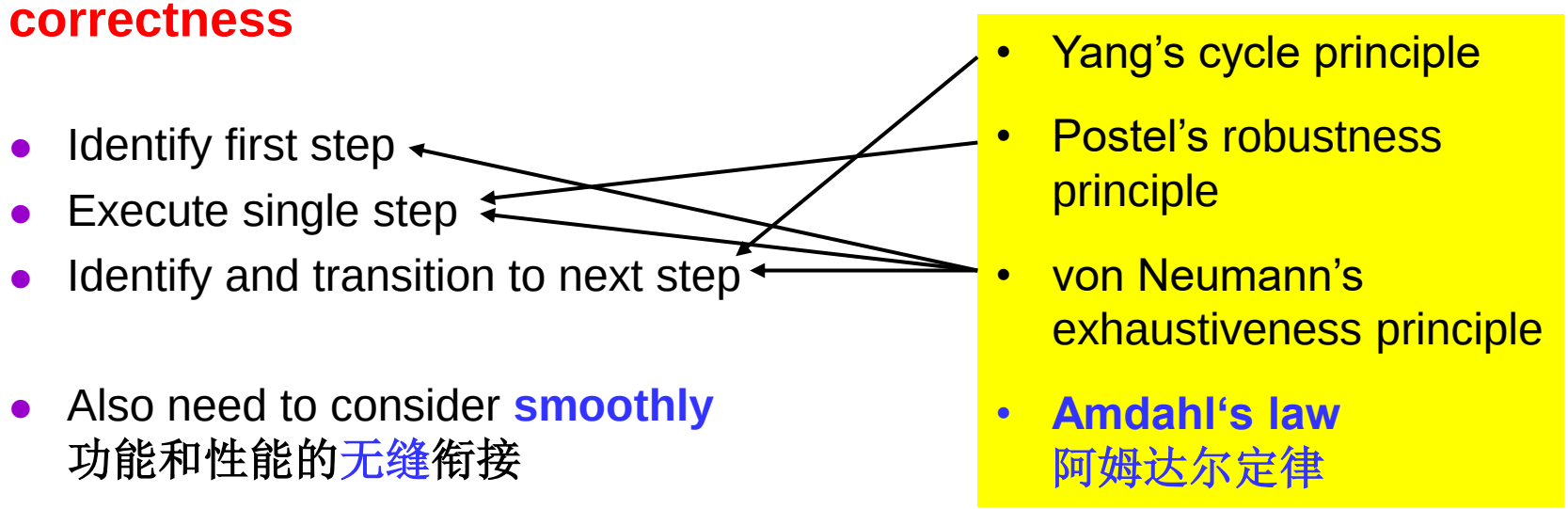
5.1 The symphony of four principles

- 1-minute quiz
  - Q: Why and how can two students in San Jose and Shenzhen conduct an online video talk correctly? Please give a specific principle.
    - Why trillions of instructions can be automatically executed in a fraction of a second, across the globe, to produce **correct** computational results?
  - A. The computers involved in the video talk execute their computational processes correctly and smoothly
  - More concretely, for each computational process involved, do a computational induction (similar to mathematic induction), to ensure **correctness**



5.1 The symphony of four principles

- 1-minute quiz
  - Q: Why and how can two students in San Jose and Shenzhen conduct an online video talk correctly? Please give a specific principle.
    - Why trillions of instructions can be automatically executed **in a fraction of a second**, across the globe, to produce **correct** computational results?
  - A. The computers involved in the video talk execute their computational processes correctly and smoothly
  - More concretely, for each computational process involved, do a computational induction (similar to mathematic induction), to ensure **correctness**



5.2 Yang's cycle principle

- In a multi-step computational process, how to ensure the seamless transition from one step to the next step?
- Yang's cycle principle
  - A system executes a computational process in a sequence of cycles. 系统按照周期执行计算过程
  - The system finishes one cycle and automatically returns to the beginning (of the next cycle), 系统执行完一个周期，自动地返回到下一个周期的开始
  - So that different computational processes preserve their respective kinds. 不同计算过程保持它们各自的特色
  - Examples of different kinds, when step=instruction
    - MOV to register instruction, MOV to memory instruction
    - ADD instruction, INC instruction
    - CMP instruction, JL instruction

《太玄经·**周首**》𝌢:
阳气周神而反乎始，物继其汇。

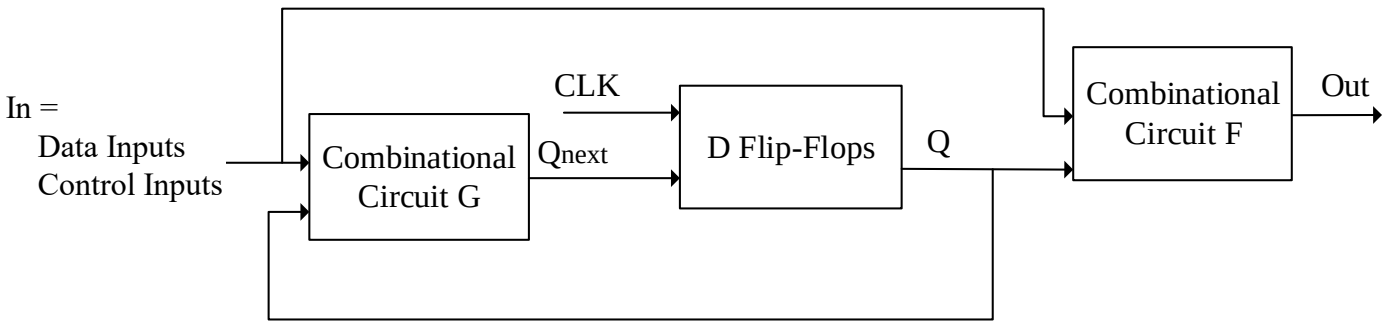
Head **Full Circle** 𝌢:
Yang qi comes full circle. Divinely, it returns to the beginning. Things go on to preserve their kinds.

扬雄，公元前2年
Yang Xiong, 2 BCE

English translation of 《太玄经·周首》adapted from
Nylan M. The Canon of Supreme Mystery by Yang Hsiung: A Translation with Commentary of the T'ai Hsuan Ching. SUNY Press, 1993.

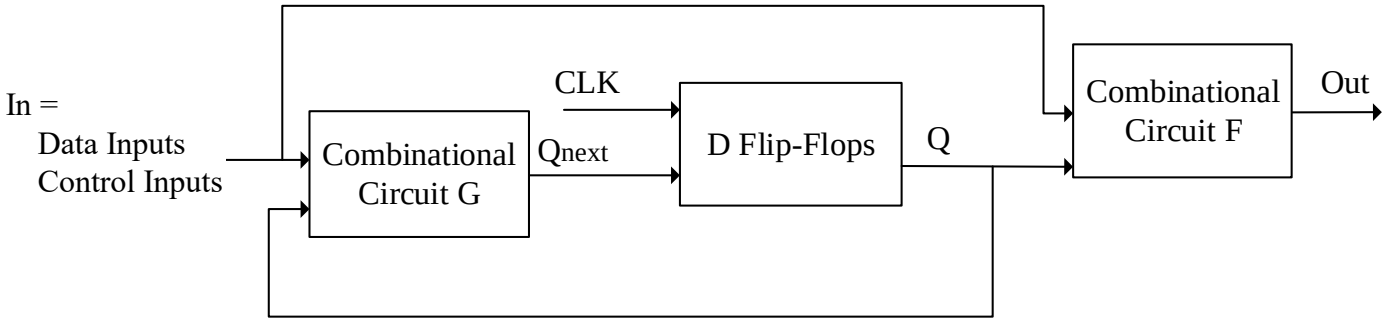
Crucial details 使用自动机的状态变量是关键

- Automatically return to the beginning of next cycle
- 怎样实现“周神而反乎始”？
  - Sequential circuit uses current state to generate next state
    第k步的Q_{next}，自动变成第k+1步的Q
    - At step *k*, the system is in state Q, = output of the D flip-flops
    - Functionality of step *k*
      - Use Q and current input In to generate Q_{next} and current output Out
    - When step *k* finishes, i.e., when CLK switches to the next clock cycle
      - Q_{next} replaces Q to become the current state via the D flip-flops, and the system returns to the beginning of step *k* + 1



Crucial details

- Automatically return to the beginning of next cycle
  - Sequential circuit uses current state to generate next state
    - At step *k*, the system is in state Q, = output of the D flip-flops
    - Functionality of step *k*
      - Use Q and current input In to generate Q_{next} and current output Out
    - When step *k* finishes, i.e., when CLK switches to the next clock cycle
      - Q_{next} replaces Q to become the current state via the D flip-flops, and the system returns to the beginning of step *k* + 1



- Use the same cycle mechanism to support diversity
  - By utilizing different control signals (control inputs)

Cycles have different granularities
周期有不同粒度

- The task of sending a WeChat message involves the executions of several programs, and consists of a sequence of program cycles 程序周期
- Execution of a program cycle consists of the executions of a sequence of instruction cycles 指令周期
- Execution of a instruction cycle consists of the executions of a sequence of clock cycles 时钟周期
- A 1-GHz processor has a clock cycle of 1 ns

- At each clock cycle, the processor performs a state transition of one or more sequential circuits

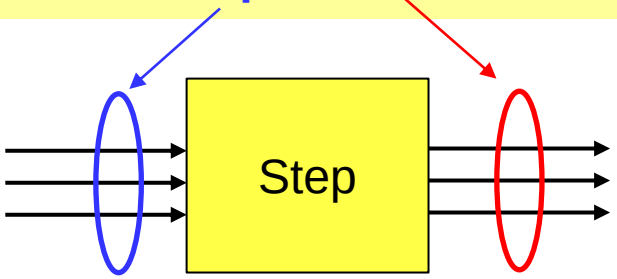
5.3 Postel's robustness principle 宽进严出原理

- Originally proposed by Jon Postel for the Internet
- Has become a systems principle
- When design, implement, and use a system, for every step,
  - Be tolerant of inputs and strict on outputs (宽进严出)
  - Be tolerant of inputs
    - System should still work when inputs deviate somewhat from "correct" values
  - Be strict on outputs
    - System should generate only "correct" outputs, not deviating from "correct" values
- Implication
  - Accumulation of errors, drifts, and distortions can often be avoided 误差、漂移、失真不会积累

TCP implementations should follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.

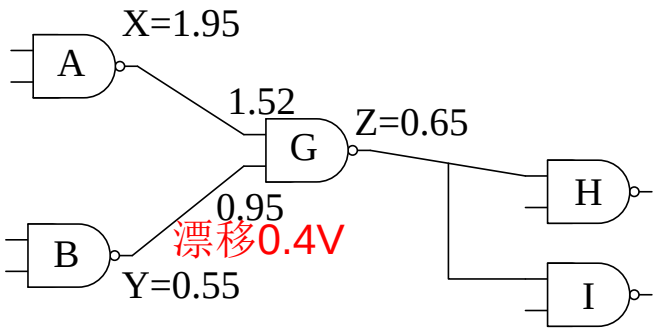
Jon Postel, 1980

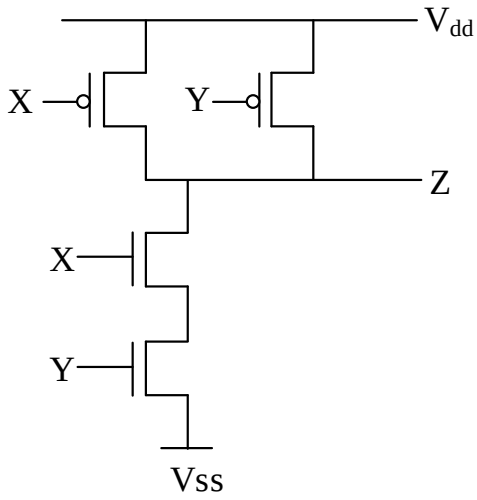
Be **strict in outputs**, and be **tolerant of inputs**



An example

- Consider gate G in the circuit of 5 NAND gates
  - It receives inputs from A, B, and outputs to H, I
    - All NAND gate have the same behavior and been implemented by a CMOS circuit
  - Naïve Design of the CMOS circuit without following Postel's robustness principle 违反波斯特尔鲁棒原理的糟糕设计
    - There is gap near the threshold voltage Vth = 0.7 Volt
      - When A=HIGH=1.95 and B=LOW=0.55 Volt, Z should be HIGH>0.7 Volt
      - However, after B drifts +0.4 to reach 0.95 Volt, Z becomes LOW, an error



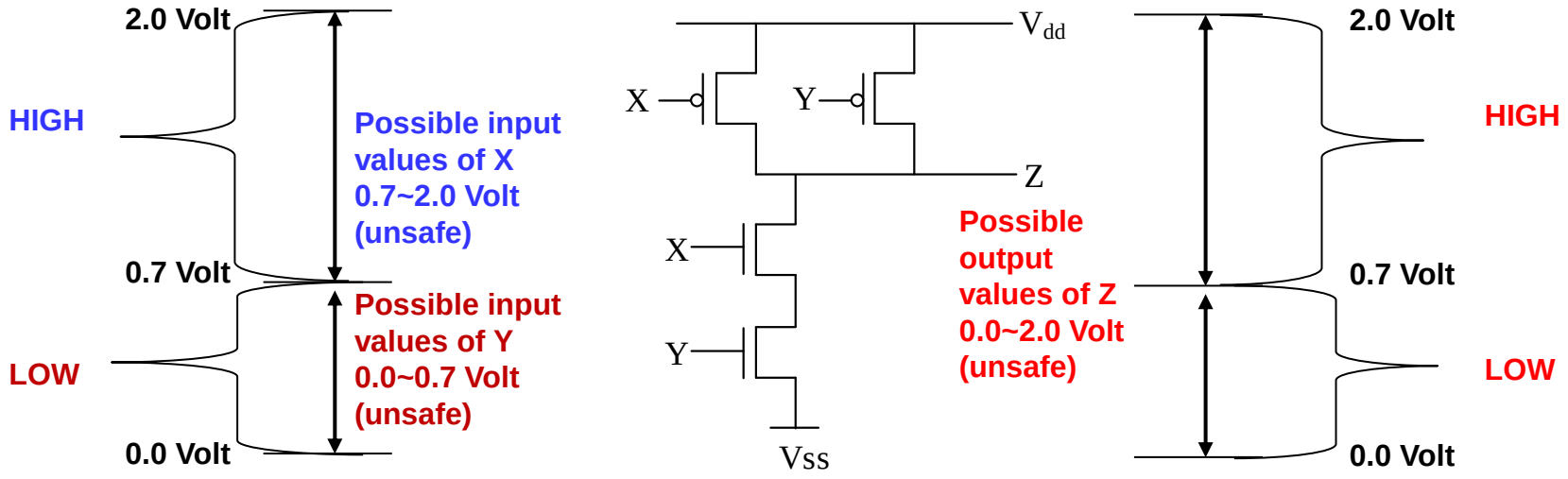


Vdd = 2 Volt 高电压
Vss = 0 Volt 低电压
Vth = 0.7 Volt 阈值电压

Naïve Design
Logic 1: > 0.7 Volt
Logic 0: < 0.7 Volt

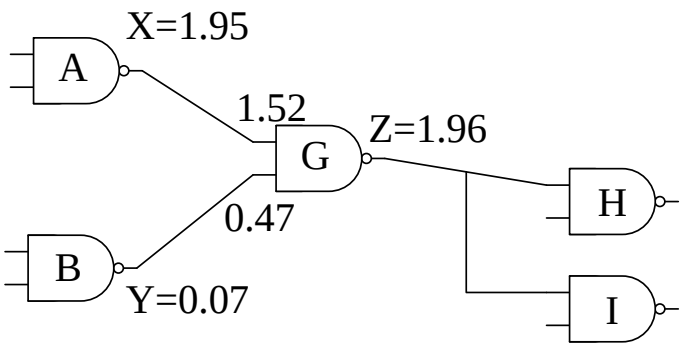
Summary of Naïve Design

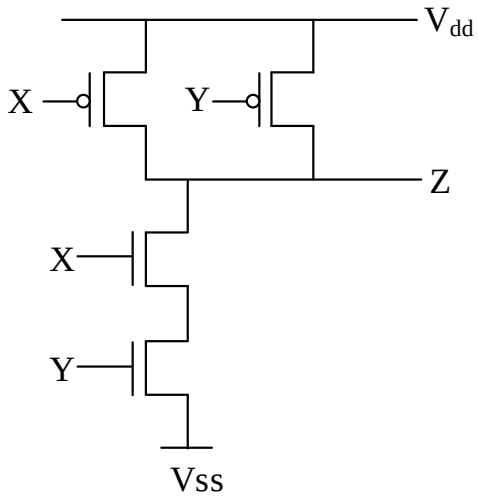
- Assume **X=HIGH** and **Y=LOW**. Then **Z should be HIGH**
  - But, could easily get the wrong result of Z = LOW
- Why?
- Treat inputs and outputs equally, and in a bad way 宽进宽出
  - Both the input side and the output side
    - have **0 minimal gap** between HIGH and LOW
    - have **unsafe margins** of 0.7 Volt for LOW and 1.3 Volt for HIGH



A better design 遵循波斯特尔鲁棒原理

- The better design following Postel's robustness principle
  - A minimal **gap of 1.0 volt at the input side**
  - A minimal **gap of 1.8 volt at the output side**
  - Note that the output of B cannot be 0.55 Volt. It has to be < 0.1 Volt
  - Let B=LOW=0.07<0.1 Volt. Even after a drifting value of +0.4 Volt, G still sees a LOW value, since B=0.47 Volt.
  - Thus, Z is HIGH with Z > 1.9 Volt





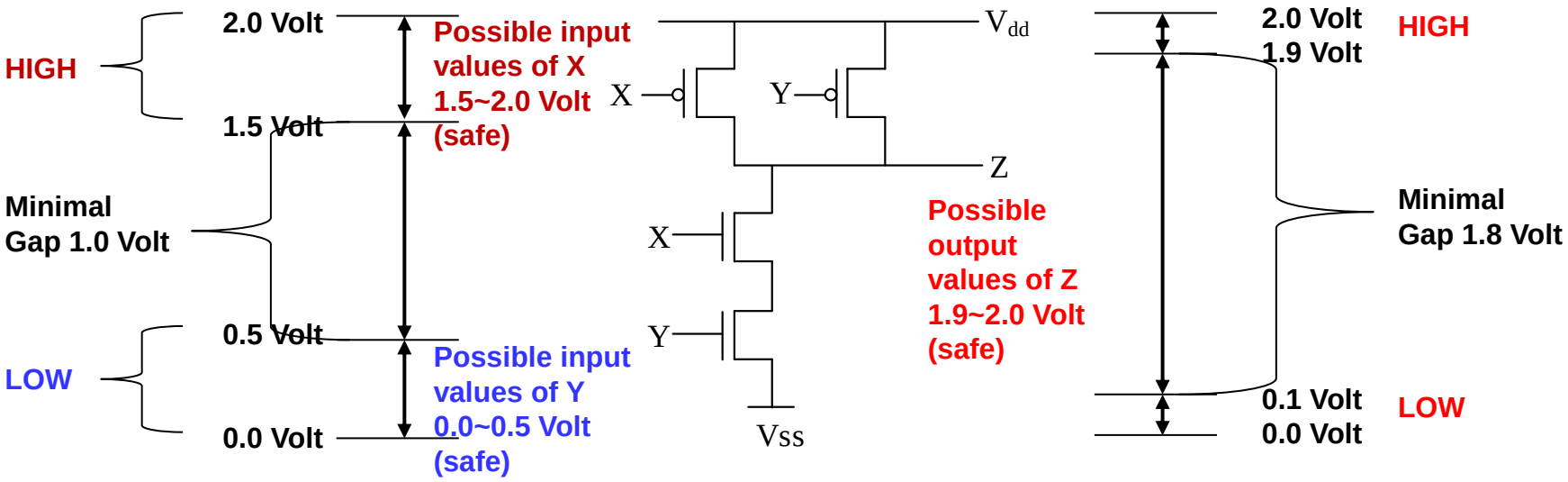
Better Design

For Input Voltages
Logic 1: > 1.5 Volt
Logic 0: < 0.5 Volt

For Output Voltages
Logic 1: > 1.9 Volt
Logic 0: < 0.1 Volt

Summary of Better Design

- Assume **X=HIGH** and **Y=LOW**. Then **Z will be HIGH**
- **Tolerance on inputs** 宽进
  - Input to a gate has a 0.5 Volt safe margin and a minimal gap of 1 Volt
    - Naïve Design: 0.7 and 1.3 unsafe margins and 0 minimal gap
- **Strictness on outputs** 严出
  - Output from a gate has a 0.1 Volt safe margin and a minimal gap of 1.8 Volt
    - Naïve Design: 0.7 and 1.3 unsafe margins and 0 minimal gap



5.4 von Neumann's exhaustiveness principle
冯诺依曼穷尽原理

- Computer must be given instructions in absolutely exhaustive detail when automatically solving a problem
  当自动解决问题时，必须给计算机指令，穷尽所有细节
- In the quote, two terms have specific meanings
  - *Operation* = Problem-solving Task
    - E.g., solving a non-linear partial differential equation
  - *Device* = Computer
    - An automatic computing system

> The instructions which govern this *operation* must be given to the *device* in absolutely exhaustive detail. …
>
> Once these instructions are given to the device, it must be able to carry them out completely and without any need for further intelligent human intervention.
>
> John von Neumann, 1945

5.4 von Neumann's exhaustiveness principle

- Computer must be given instructions in absolutely exhaustive detail when automatically solving a problem
- 1-minute quiz
  - Q: How to cover "absolutely exhaustive detail"? 怎么可能穷尽所有细节？
  - 可能有无穷多种细节，如何穷尽？
  - 使用前述**计算归纳法**
    - Identify first step
      确定第一步
    - Execute single step
      正确执行每一步（当前步骤）
      应对异常
    - Identify and transition
      to next step 确定并过渡到下一步

> The instructions which govern this operation must be given to the device in absolutely exhaustive detail. …
>
> Once these instructions are given to the device, it must be able to carry them out completely and without any need for further intelligent human intervention.
>
> John von Neumann, 1945

5.4 von Neumann's exhaustiveness principle

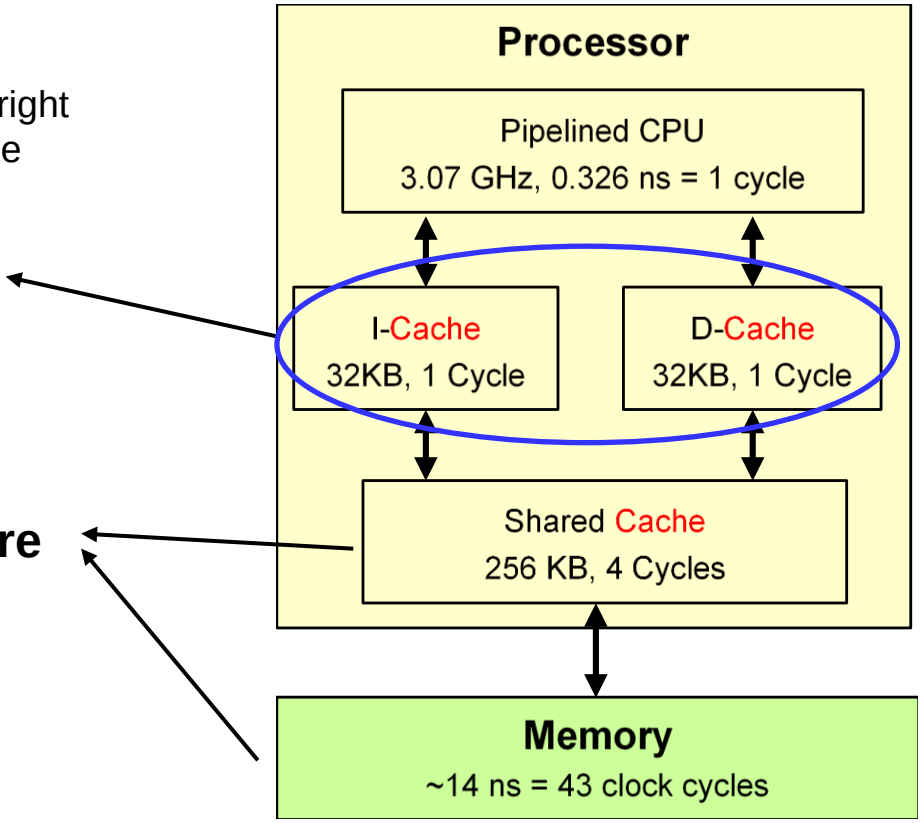
- Computer must be given instructions in absolutely exhaustive detail when automatically solving a problem
- 1-minute quiz
  - Q: How to achieve "in absolutely exhaustive detail"? List the types of instructions, and give an example for each type
  - A: Answers to more fundamental questions 必须考虑正常执行和异常执行
    - Where and what is the first instruction, when the computer power is turned on?
      开机之后执行的第一条指令在哪里？是什么？
    - How to determine the next instruction to execute?
      如何确定下一条指令？
    - What types of exceptions are there, to normal execution of programs?
      如何保证当前指令正确执行？
      没有正确执行，只有两种可能：（1）没有正确设计实现；（2）出现异常
      有哪些异常类别？如何应对？

The first instruction to execute
when the computer power turns on
开机后执行的第一条指令：在哪里？是什么？

- Example with an x86 processor **x86处理器例子**
  - Where: the first instruction to execute is at memory address 0xFFFFFFF0 在哪里？处理器规定的地址空间的某个高位地址
  - What: a jump instruction, e.g., JUMP 000F0000 是什么
    - Address 000F0000 contains the entry instruction for the BIOS code
      跳转指令：跳转到BIOS（最底层系统软件）的入口地址
- Why? 什么道理？
  - Computer starts by executing the BIOS firmware code
    - To initialize the computer and to load the operating system
      计算机开机后先执行固件，做初始化工作（如自检），然后载入系统软件
  - Using a jump instruction upfront increases flexibility
    跳转指令增加灵活性
    - E.g., if we want the computer to start by executing another firmware code BIOS-2 at entry address 000FA000,then change
    - Address 0xFFFFFFF0 to hold JUMP 000FA000

Three ways to determine the next instruction to execute 确定下一条指令的方法1

- The earliest method is linear sequencing in Harvard Mark I computer, the *Automatic Sequence Controlled Calculator*
  - Instructions are linearly sequenced
    下一条指令紧跟当前指令
    - There is no jump. Next instruction is located right after current instruction on the instruction tape
  - Storing data and code separately
    (This is called **Harvard architecture**)
    哈佛体系结构
    - Still widely used in the cache units of modern computers. A processor has separate instruction cache and data cache
  - In contrast, the **Princeton architecture**
    普林斯顿体系结构
    uses a single cache or memory
    to store both data and instructions
- Modern computers use both



Three ways to determine the next instruction to execute

确定下一条指令的方法2、3

- The ENIAC method
  ENIAC方法
  - Every instruction holds the address of the next instruction
    当前指令指明下一条指令的地址
    - Used by the revised version of the ENIAC computer

- Modern computers mostly use the PC mechanism: the address of the next instruction to execute is stored in the program counter (PC)
  当代计算机采用“程序计数器机制”
  PC存放下一条指令的地址

Deal with exceptions to normal execution
应对异常

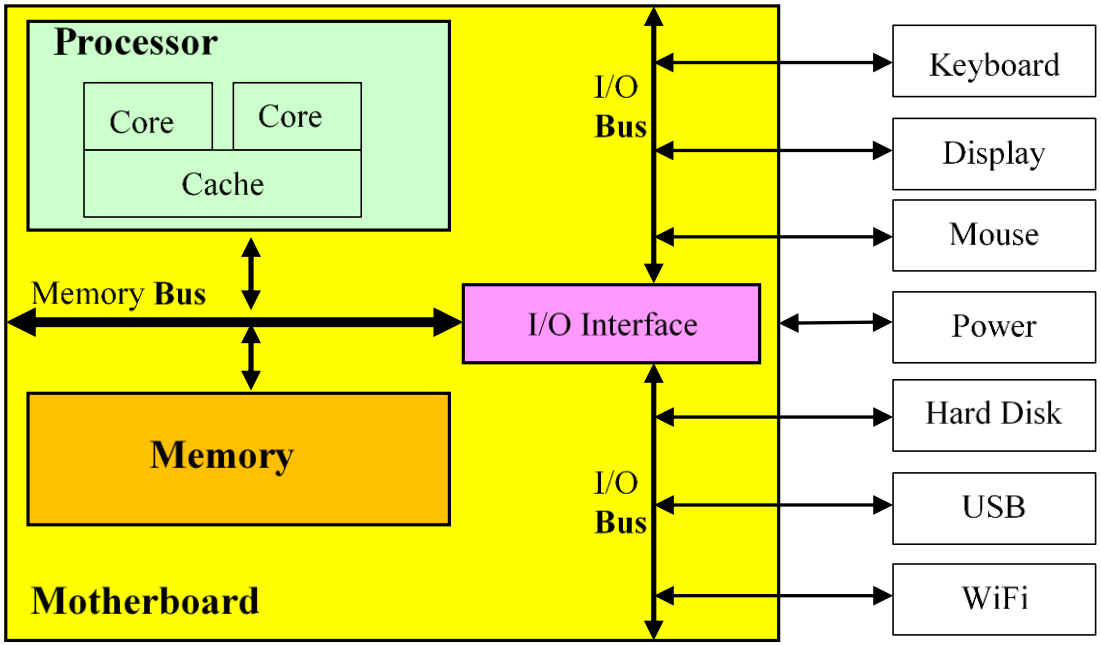
- 同学们已经体验过了Go语言编程中的异常
  - 读文件可能出错
- We have seen exceptions in programming, e.g.,

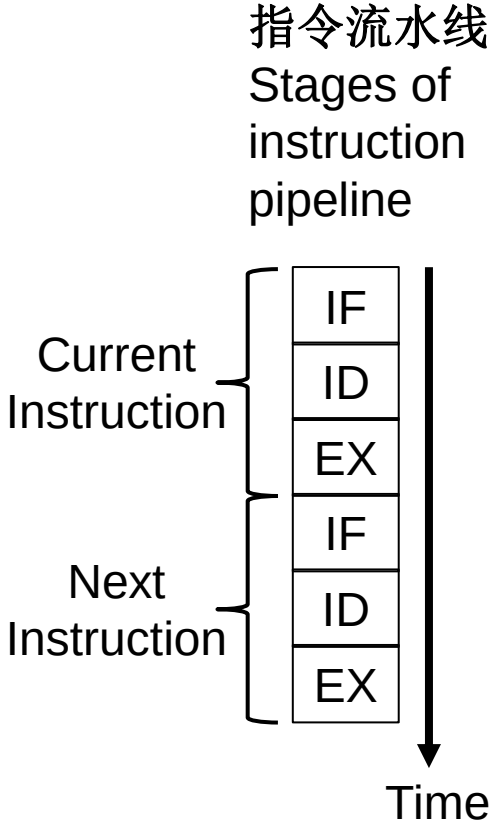
  in the Text Hider project, the statement
  - p, _ := ioutil.ReadFile(“./Autumn.bmp”)　例如，文件不存在

  should really be
  - p, error := ioutil.ReadFile(“./Autumn.bmp”)
    if error != nil {
    …// put exception-handling code here　出错了，执行异常处理代码
    }
    … // no error; continue normal execution　无错，继续正常执行

Three types of exceptions are supported by computer hardware 三种异常

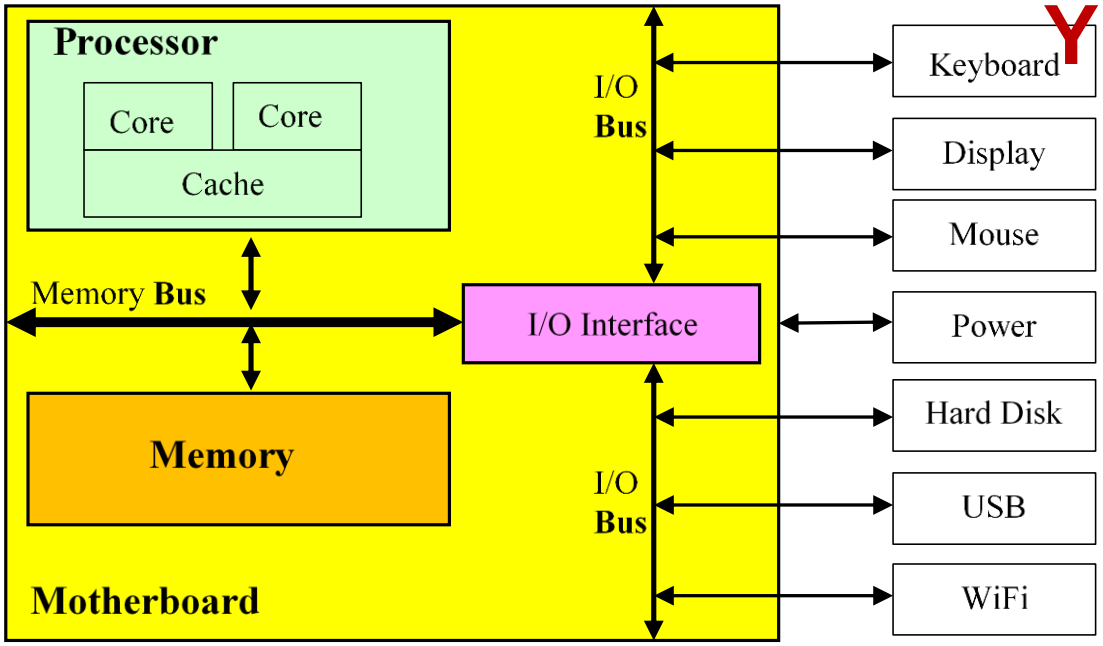
- In normal execution (without exception), the current instruction finishes and continue to execute the next instruction
  正常情况：执行完当前指令，继续执行下一条指令
- 异常情况：（1）不执行完当前指令；（2）不执行下一条指令；
  - Interrupt 中断
  - Hardware error 硬件出错
  - Machine check 保底异常

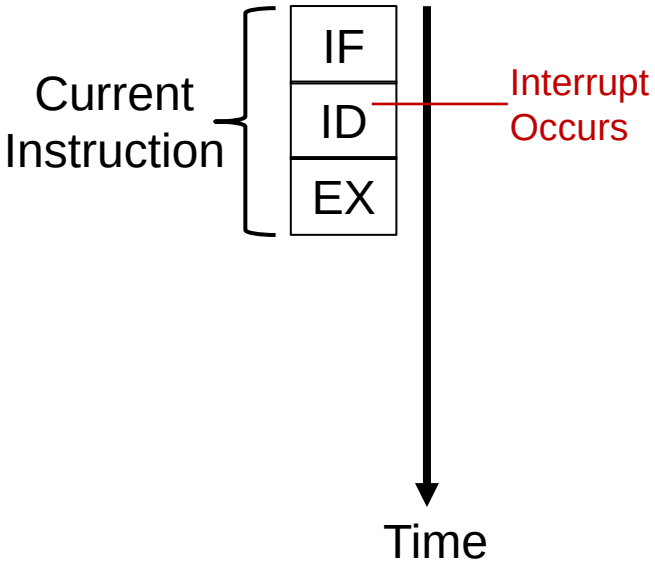




5.4.1 Interrupt handling 中断处理

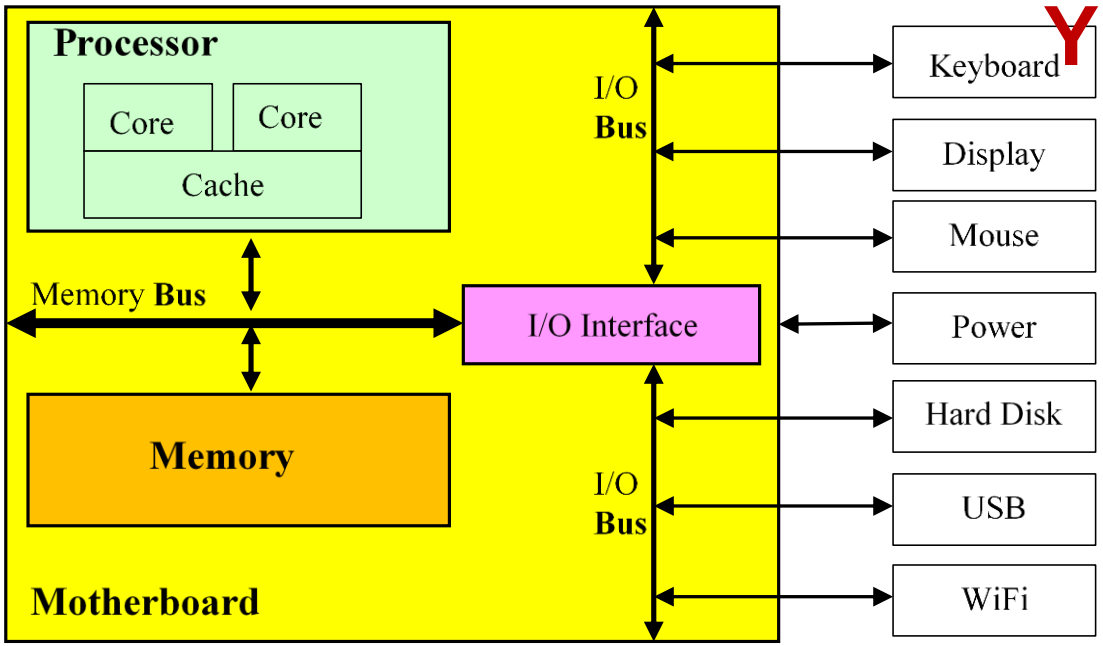
- When an interrupt occurs, e.g.,
  - When the user punches key 'Y' on the keyboard 用户敲了'Y'键 while the processor is executing the instruction decode stage
- What should the processor do?
  - Should it immediately take an exception-handling action?
  - Should it finish the current instruction first?

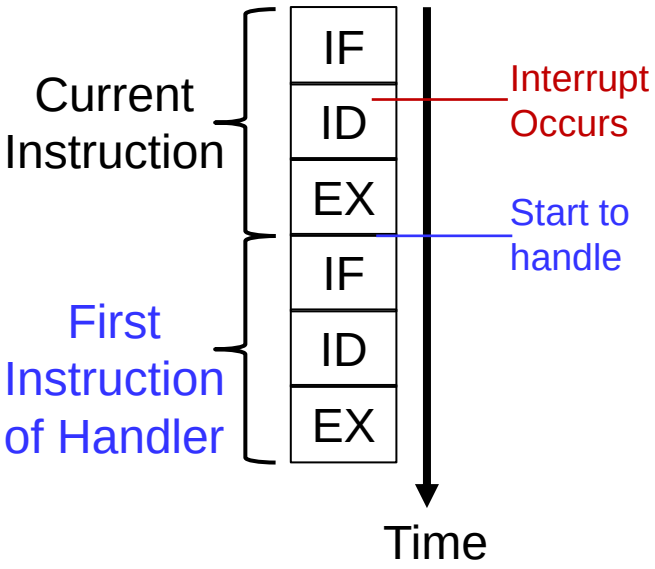




Interrupt handling

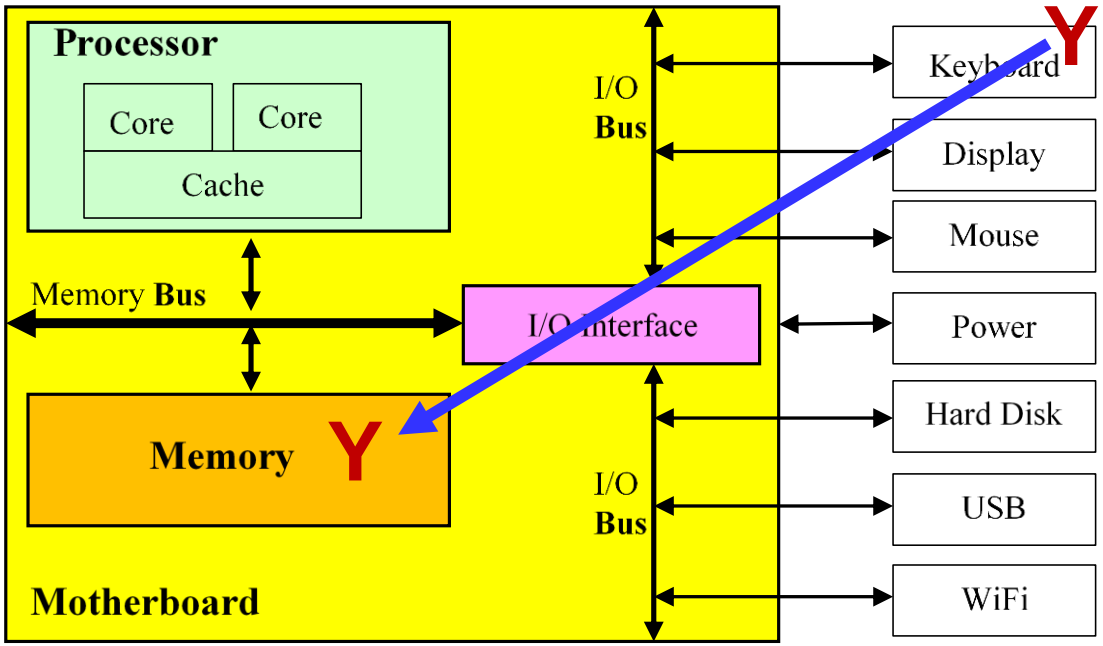
- 继续执行完毕当前指令，然后跳转到中断处理程序的入口
- The processor finishes the current instruction and jumps to an interrupt handling subprogram to handle the interrupt
  - 中断处理程序称为 interrupt handler

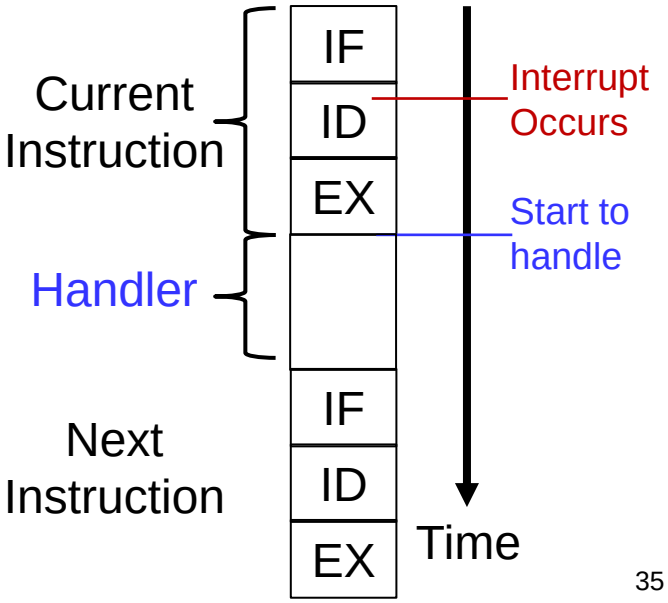




Interrupt handling

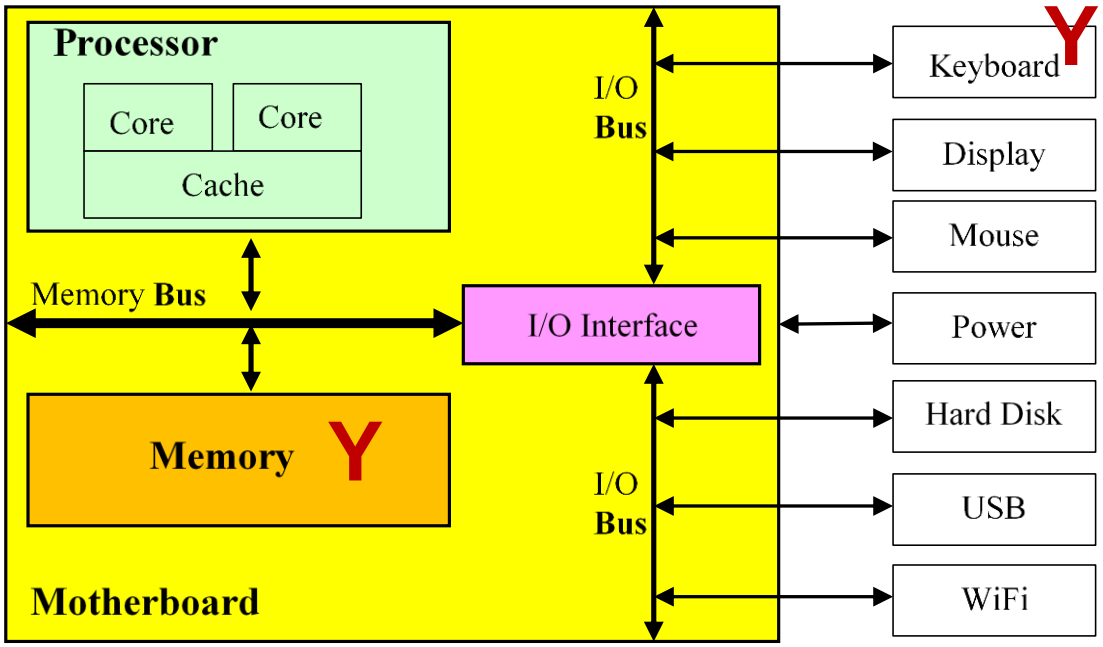
- The processor finishes the current instruction and jumps to an interrupt handling subprogram to handle the interrupt
  - such as **copying the punched key value to memory**
    执行中断处理程序，包括将敲入的键值'Y'拷贝到内存

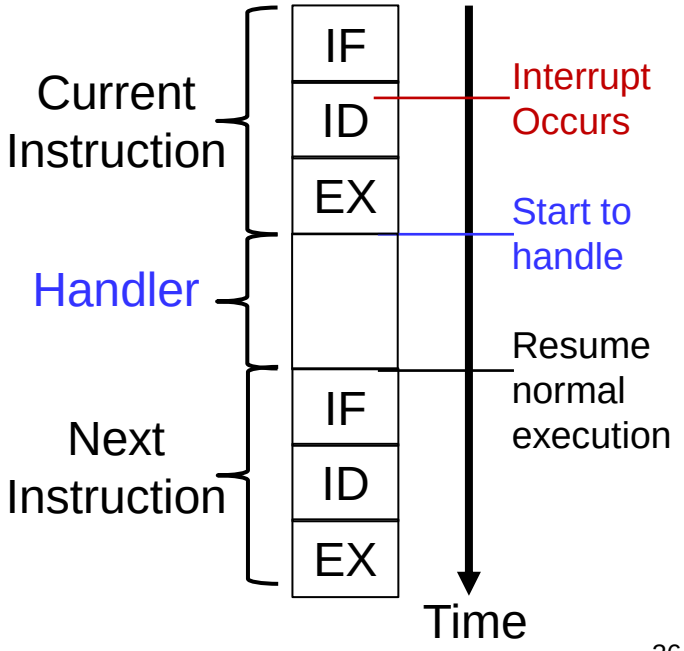




Interrupt handling

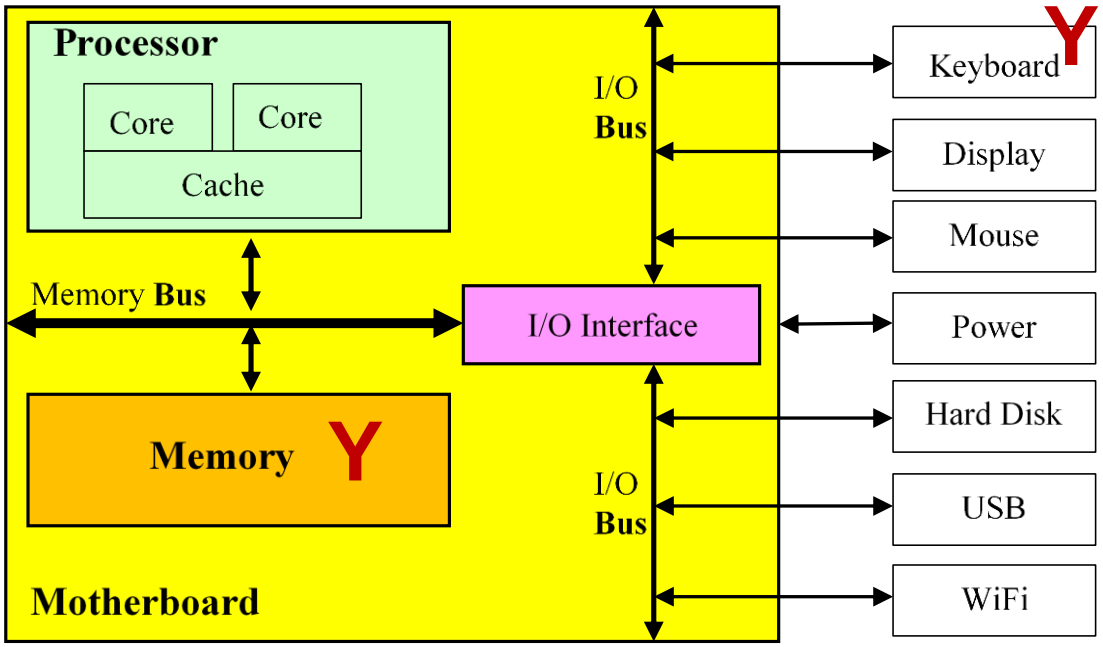
- When an interrupt occurs, the processor finishes the current instruction and jumps to an interrupt handling subprogram to handle the interrupt
  - such as **copying the punched key value to memory**
- Then, the processor resumes normal execution
  - by executing the original next instruction
    退出中断处理程序，恢复正常执行原来程序的下一条指令

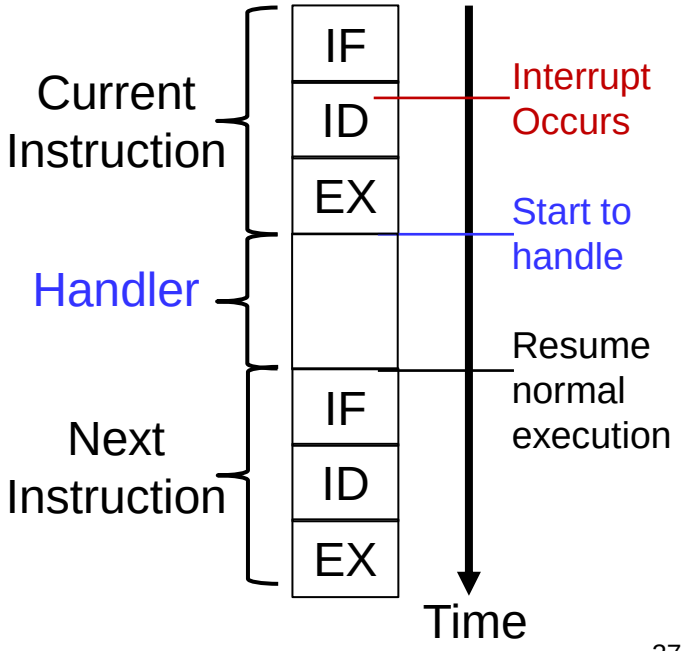




Interrupt handling

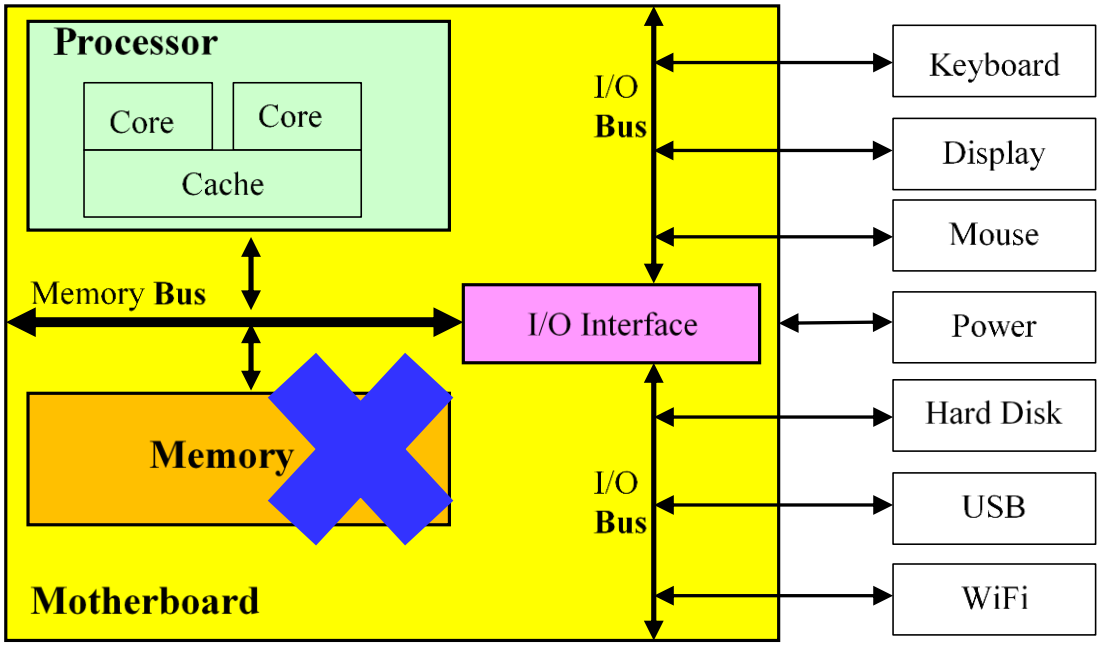
- When an interrupt occurs, the processor finishes the current instruction and jumps to an interrupt handling subprogram to handle the interrupt
  - such as **copying the punched key value to memory**
- Then, the processor resumes executing the next instruction
  - Q: how does the processor know the address of the next instruction?
  - A: need to **save context** before executing handler
    进入中断处理程序前，必须保存上下文，知道返回后该执行哪条指令

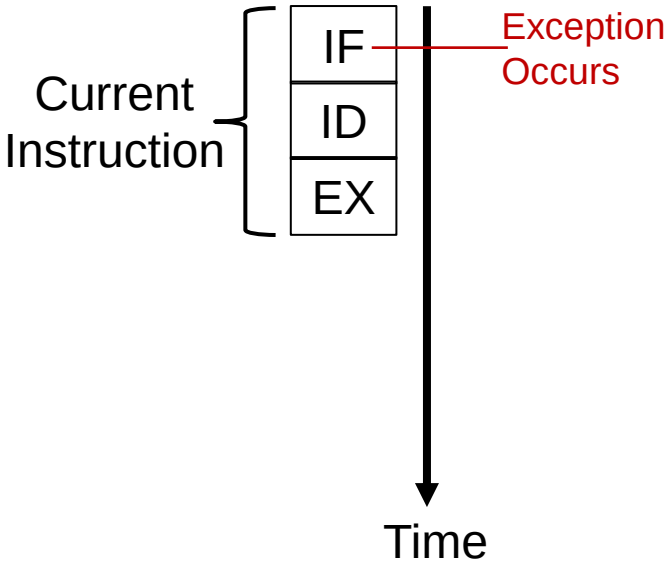




5.4.2 Hardware error handling 硬件出错

- When a hardware error occurs, e.g.,
  - When the memory becomes faulty and generates a hardware error exception 例如，内存条坏了
- What should the processor do?
  - Should it immediately take an exception-handling action?
  - Should it finish the current instruction first?

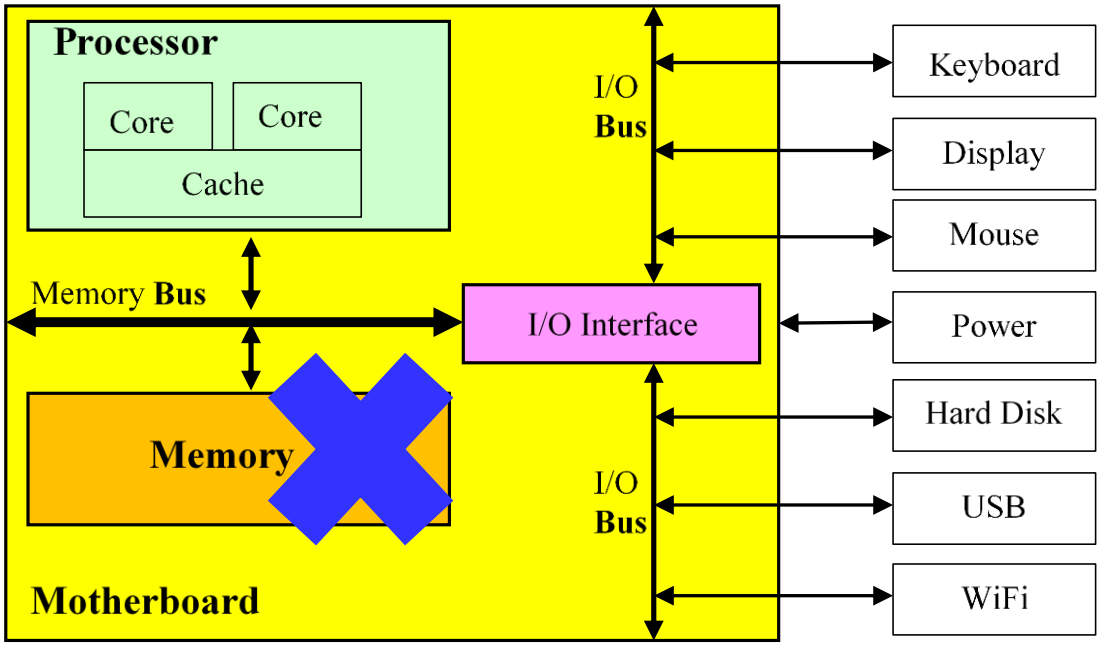


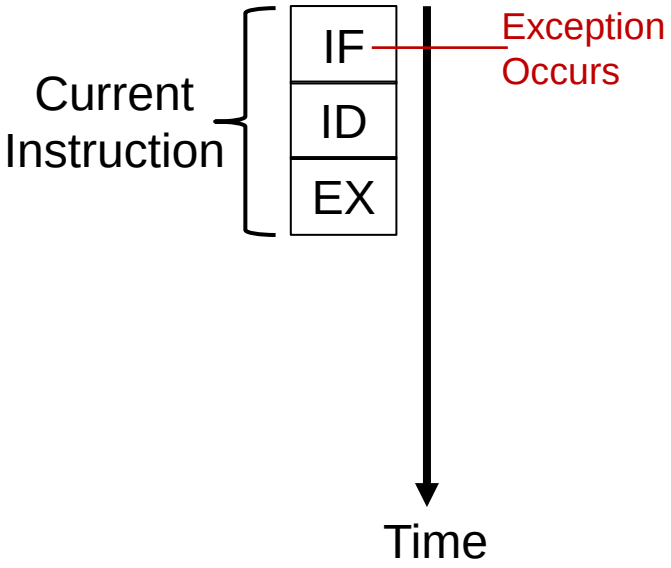


Hardware error

- When the memory becomes faulty and generates a hardware error exception 内存条坏了
  - Should the processor finish the current instruction first?

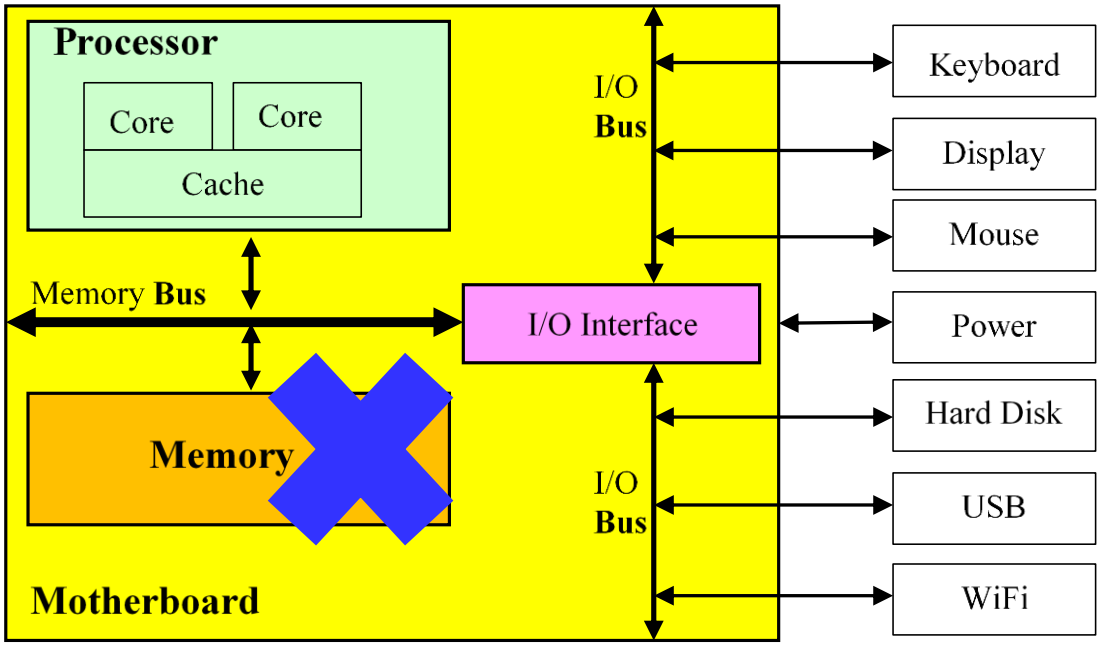
  还能够先完成当前指令再“中断”吗？
  - No, because the IF stage cannot be finished 不行！指令都取不回来
    - The instruction cannot be fetched from memory 完不成**取指（IF）**操作

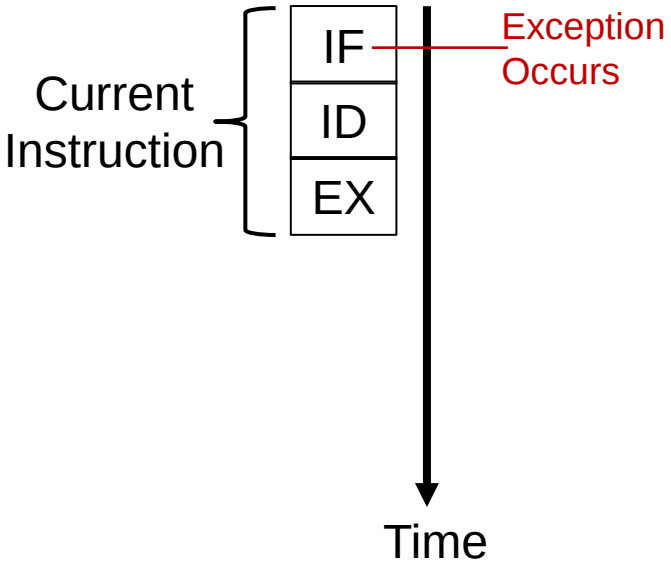




Hardware error

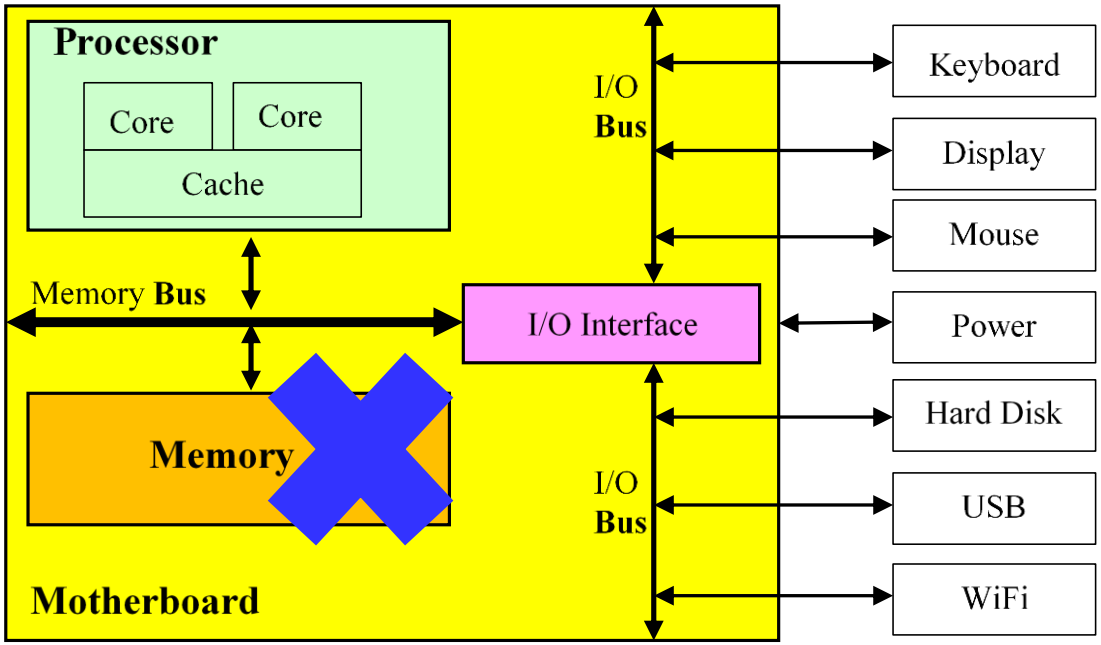
- When the memory becomes faulty and generates a hardware error exception
  - Should the processor immediately take an exception-handling action?
  - Yes, it executes an exception-handling sequence of steps without depending on the memory
    立即执行异常处理程序，无需访问内存

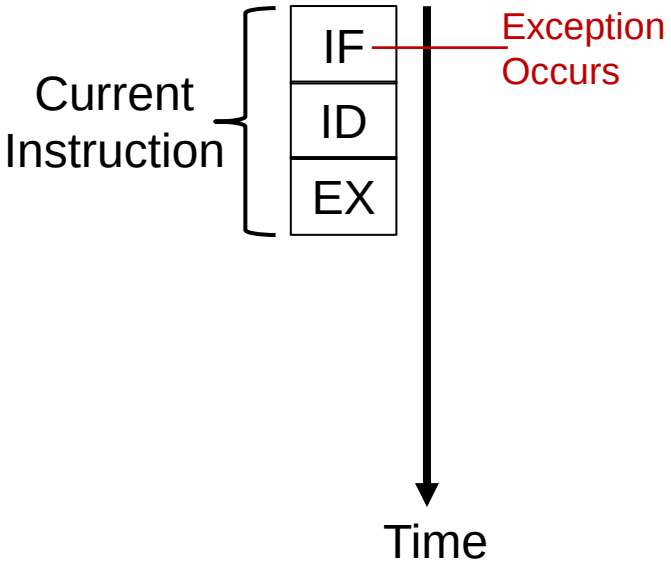




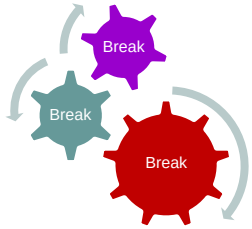
5.4.3 Machine check 保底异常

- This is the "all other" exception, for exhaustiveness
  - Typically, an unrecoverable hardware error
- Example
  - While executing an exception-handling sequence of steps for the memory fault, the sequence experiences another error
  - The system generates minimal diagnostic information and crashes
    输出最基本的诊断信息，宕机





Take-Home Messages



- Seamless transition of within a computational process is achieved by solving three problems
  - via the symphony of four principles
- Yang's cycle principle, types of cycles
  - The system finishes one cycle and automatically returns to the beginning of the next cycle, so that steps of the computational process preserve their respective kinds
- Postel's robustness principle
  - Each step should be tolerant of inputs and strict on outputs
- von Neumann's exhaustiveness principle
  - Computer must be given instructions in absolutely exhaustive detail when automatically solving a problem
  - To be exhaustive, the system designer must consider first step, next step, normal (correct) execution of current step, and exceptions
  - Exceptions include interrupts, hardware errors, machine checks