

git with AI

```
$ git commit -m "bug fix"
```

REJECTED: Please, you can do better than this.

```
$ git commit -m "validation bug fix"
```

REJECTED: I can play this game all day long...

```
$ git commit -m "including timestamp validation \
> in the customer backend service"
```

REJECTED: Do you really believe your validation is working??? Run the tests before committing, dude!

```
$ yum remove git && yum install svn
```

REJECTED: No, no, no! fix that validation bug first.

Daniel Stori {turnoff.us}

From Word2Vec to Code Migration

Liu Qiming
Li Ke

Contents

- The idea of Word2Vec
- Applications of Word2Vec

1. The idea of Word2Vec

i.e. Representing words as discrete symbols

Words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Representing words as discrete symbols

Words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Drawbacks:

Vector dimension = number of words in vocabulary (e.g., 500,000)

Pairwise orthogonal

Word2Vec

Dense vector for word embedding
of Dimensions \ll |Vocabulary|

Word2Vec

Dense vector for word embedding

of Dimensions << |Vocabulary|

In BERT, a **float32 tensor of shape (768,)**
= 3kB

Benefits:

Allow us to calculate similarity between words

And much more!

An impression of Word2Vec

[-7.92340040e-02 4.97796200e-03 -8.23487282e-01
5.17381579e-02 -3.13413590e-01 3.31664503e-01
-4.70242426e-02 8.89038324e-01 -4.35647935e-01
-5.54904103e-01]

First ten float numbers in the last hidden layer, when
'computer organization and design' is put in to BERT(uncase
base)

BERT, **B**idirectional **E**ncoder **R**epresentations from **T**ransformers

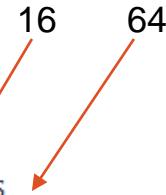
A milestone in Natural Language Processing community, most evolved derivative of Word2Vec

Word2Vec-Bert

Trained on TPUs for four days

- **BERT-Base, Uncased** : 12-layer, 768-hidden, 12-heads, 110M parameters
- **BERT-Large, Uncased** : 24-layer, 1024-hidden, 16-heads, 340M parameters
- **BERT-Base, Cased** : 12-layer, 768-hidden, 12-heads , 110M parameters
- **BERT-Large, Cased** : 24-layer, 1024-hidden, 16-heads, 340M parameters
- **BERT-Base, Multilingual Cased (New, recommended)** : 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- **BERT-Base, Multilingual Uncased (Orig, not recommended)** (Not recommended, use **Multilingual Cased** instead): 102 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- **BERT-Base, Chinese** : Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

16 64



Word2Vec

cat -- 0.11996692419052124 – dog

cat -- 0.14831775426864624 – wood

cat -- 0.1980501413345337 – motorcycle

cat -- 0.23616260290145874 – computer

cat -- 0.3551083207130432 – philosophy

cat -- 0.4030646085739136 – ‘The Abyssinian is a breed of domestic short-haired cat with a distinctive ticked tabby coat, in which individual hairs are banded with different colors.’

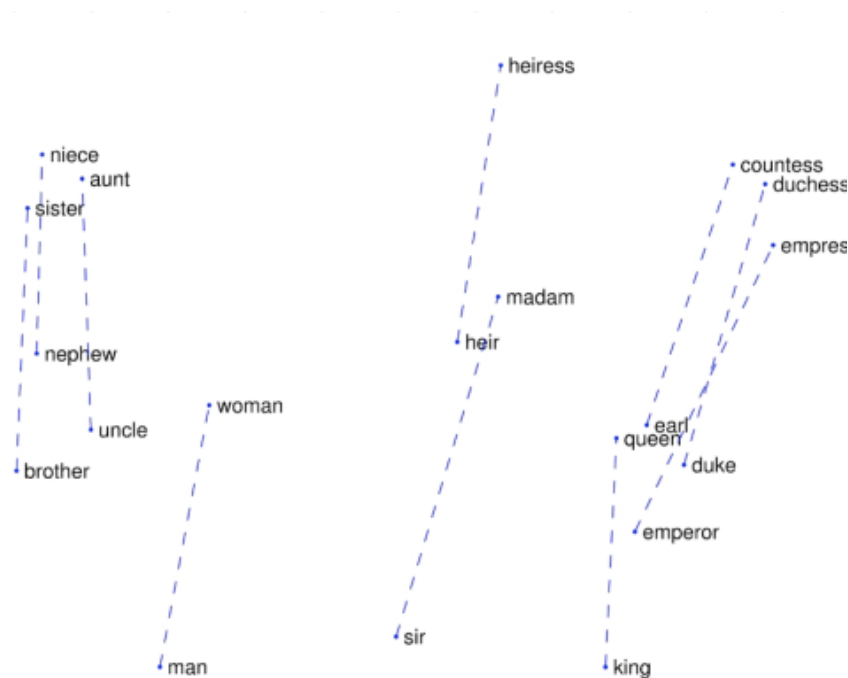
Word2Vec visualization



Similar words'
embeddings concentrate

An Improved Model of Semantic Similarity Based
on Lexical Co-Occurrence
(Communications of the ACM, 2015)

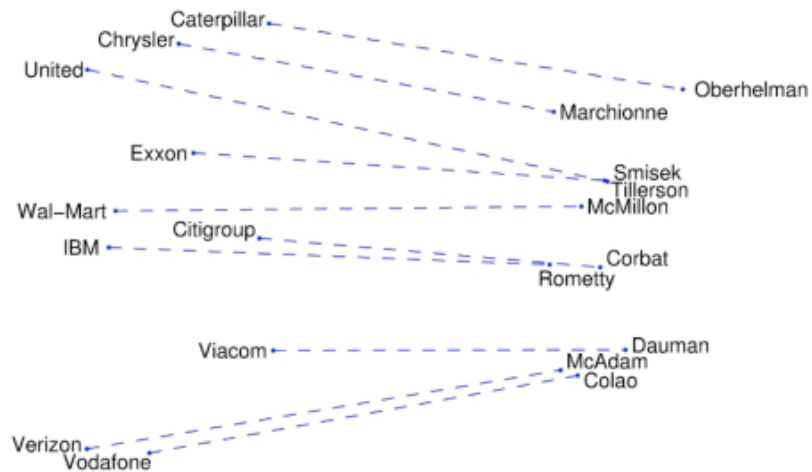
Word2Vec visualization



Lines connecting word pairs of opposite gender

GloVe: Global Vectors for Word Representation (EMNLP 2014)

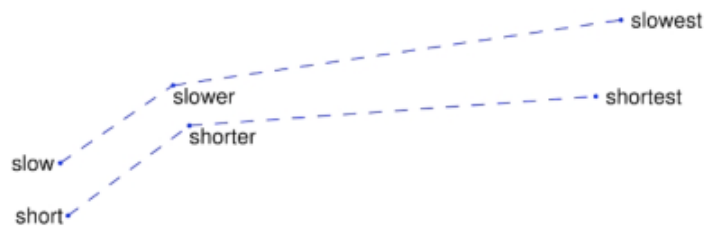
Word2Vec visualization



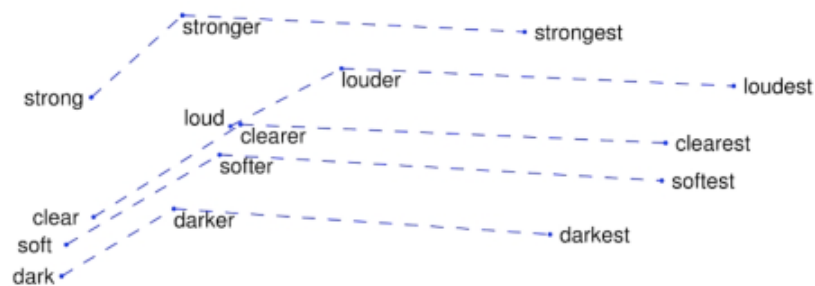
Lines connecting
companies and their CEOs

GloVe: Global Vectors for Word Representation
(EMNLP 2014)

Word2Vec visualization



Original-comparative-
superlative triples



GloVe: Global Vectors for Word Representation
(EMNLP 2014)

Further reading

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Attention is all you need

Adversarial Domain Adaptation for Duplicate Question Detection



Question



Besides natural language, where else can Word2Vec be applied?

2. Applications of Word2Vec



Trong Duc Nguyen
trong@iastate.edu



Anh Tuan Nguyen
anhnt@iastate.edu



Tien N. Nguyen
tien@iastate.edu

**MAPPING API ELEMENTS FOR CODE MIGRATION
WITH VECTOR REPRESENTATIONS**

Iowa State University, USA

<http://home.eng.iastate.edu/~trong/projects/jv2cs/>

Characterize API by its surrounding APIs

```
1. HashMap<String, Integer> dict = new HashMap<String, Integer>();
2. dict.put("A", 1);
3. FileWriter writer = new FileWriter("Vocabulary.txt");
4. for (String vocab : dict.keySet()) {
5.     writer.append(vocab + " " + dict.get(vocab) + "\r\n");
6. }
7. writer.close();
```

API Mapping



String

Integer

HashMap.put()

HashMap.keySet()



C#

string

int

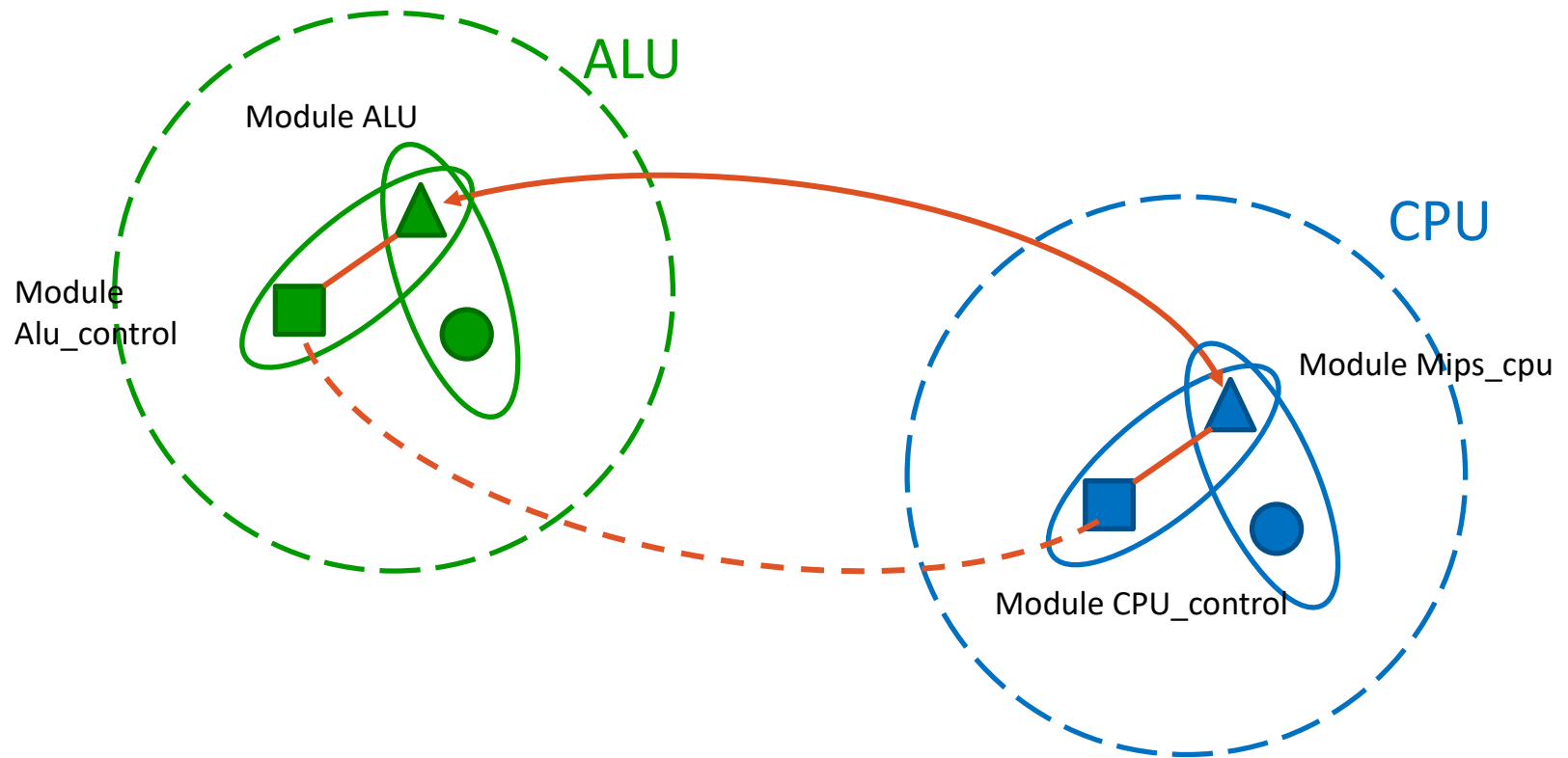
Dictionary.Add()

Dictionary.Keys

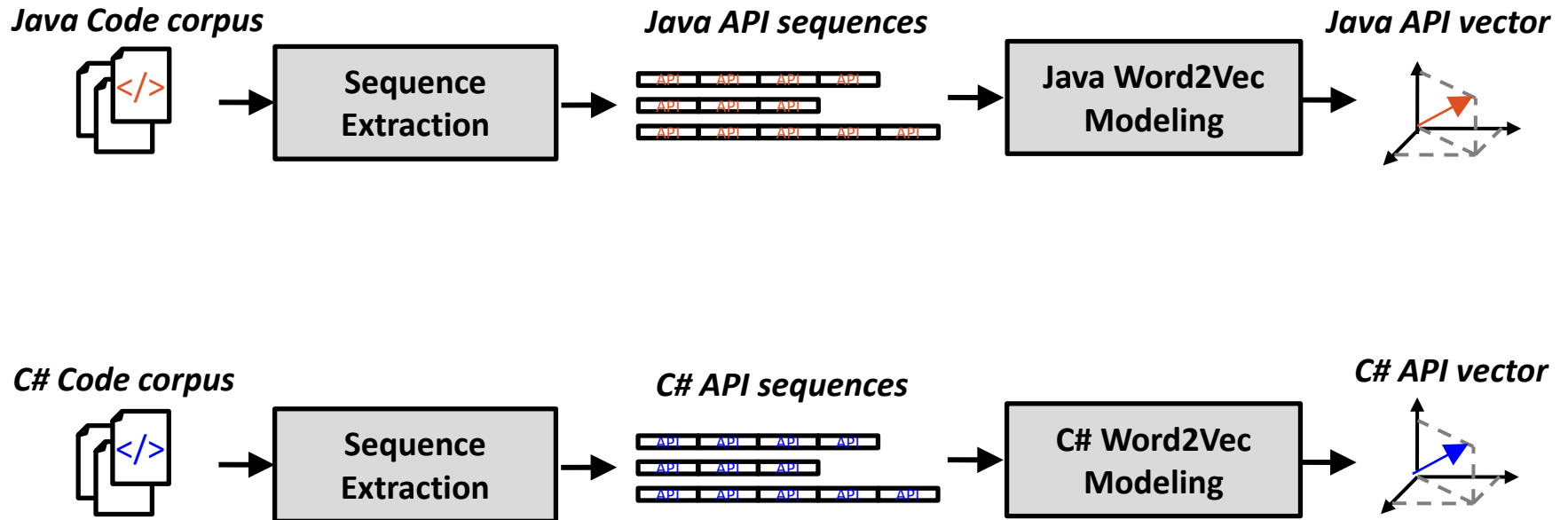
Key Insight

- Characterize APIs by their usages instead of APIs' names
 - API usages consist of the surrounding API elements
- Derive API pairs based on the relation between them

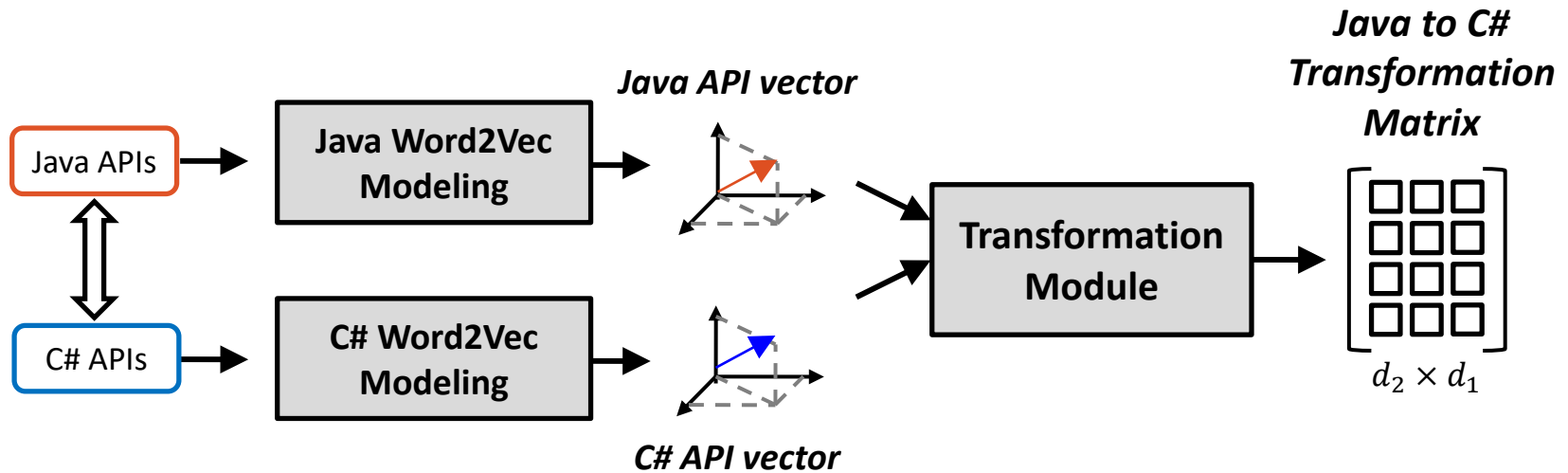
Key Idea



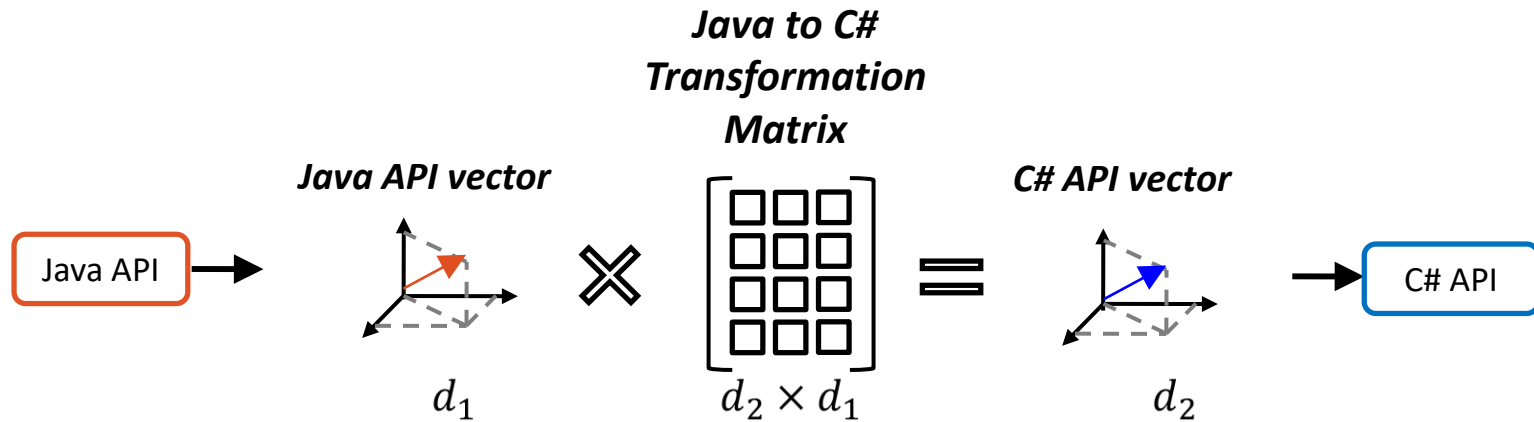
Approach



Approach



Approach



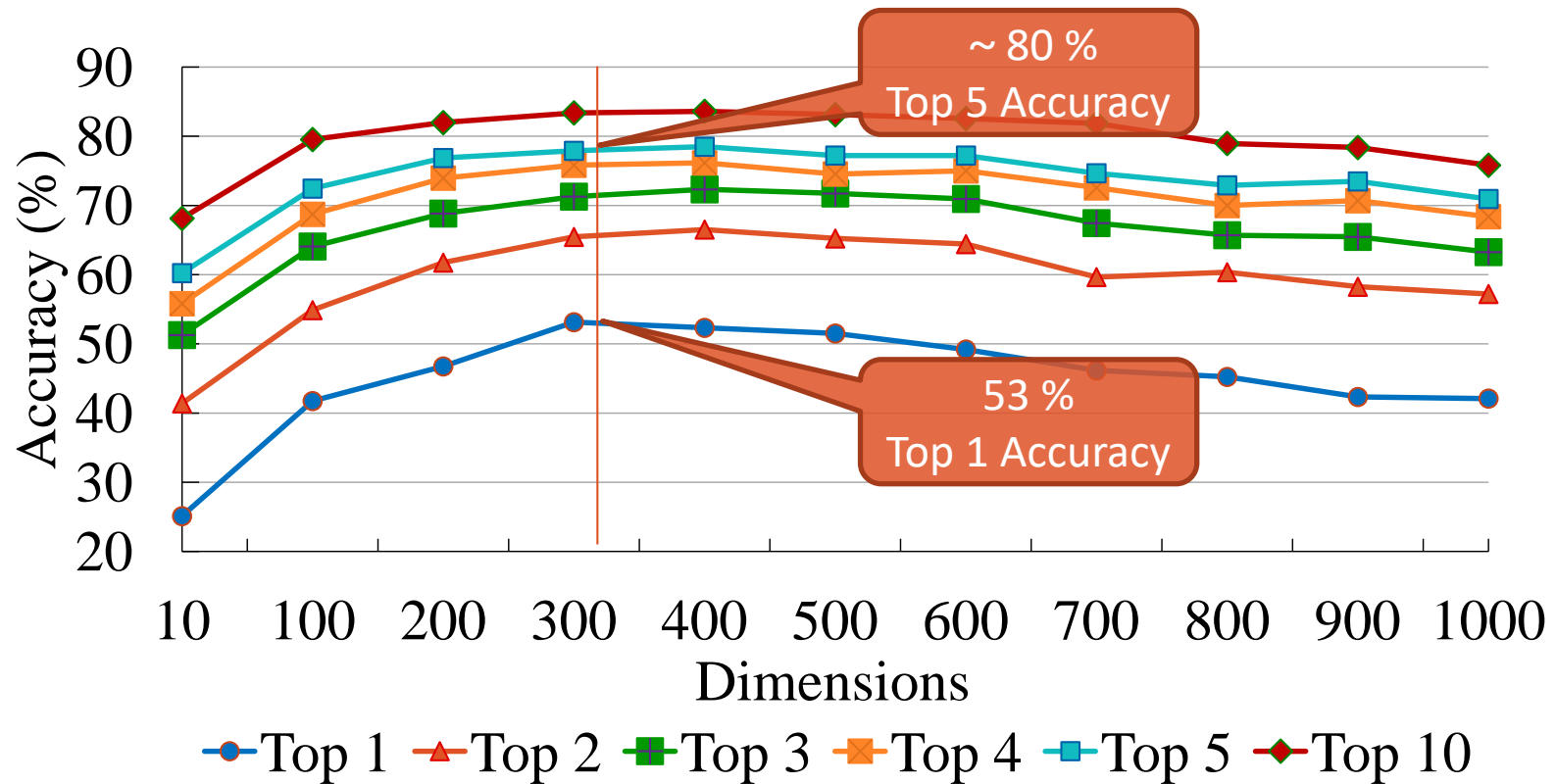
Empirical Study

- Goal:
 - The vector representation capture the relationship between API
 - The accuracy of mining the API mapping pairs
- Data collection:

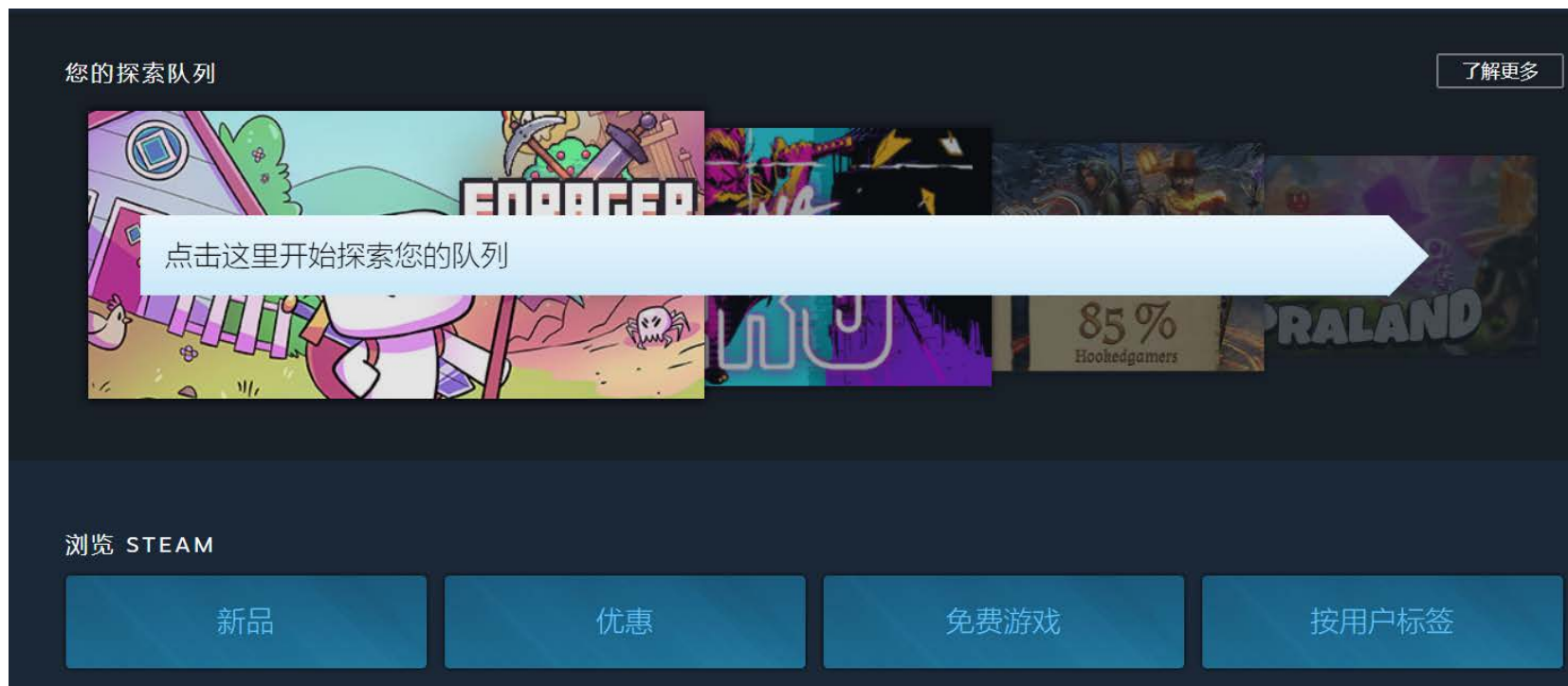
	# Projects	# Classes	# Methods	# LOCs	#Vocab size
Java Dataset	14,807	2.1 M	7 M	352 M	123 K
C# Dataset	7,724	900 K	2.3 M	292 M	130 K

- Extracted from rule-based migration tool *Java2CSharp*
<http://sourceforge.net/projects/j2cstranslator/>

API Mapping



Application





IntelliJ IDEA

Capable and Ergonomic IDE for JVM



```
public final class RandomInteger {  
    public static void main(String[] args){  
        log("Generating 10 random integers in range 0..99.");  
        Random randomGenerator = new Random();  
        for (int idx = 1; idx <= 10; idx++){  
            int randomInt = randomGenerator.nextInt(100);  
            log("Generated : " + randomInt);  
        }  
        log("Done.");  
    }  
    private static void log(String message){  
        System.out.println(message);  
    }  
}
```

Takeaways

- Word2Vec utilizes dense vectors to present each unique word to represent its linguistic information.
- Besides natural language, Word2Vec could also be used in programming languages.

Thank you!



复习:

1. 单周期与多周期处理器的设计准则
2. 控制单元实现方式: **FSM vs. Microprogramming**
 - **State register of FSM: One-hot encoding**
3. 单周期与多周期处理器的成本及性能比较
4. 流水线处理器的概念、优势和设计思路
5. **T-P-U**
 - **Tensor Processing Unit** from Google
 - **Throughput / Parallelism / Utilization in Pipelined CPU**

复习:

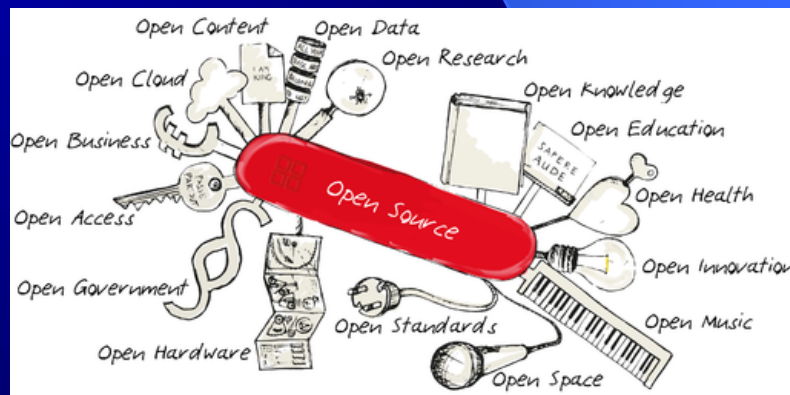
6. 按指令格式的复杂度来分, 主要有两种ISA类型的计算机

- ① 复杂指令集计算机CISC (Complex Instruction Set Computer)
- ② 精简指令集计算机RISC (Reduced Instruction Set Computer)
 - 简化的指令系统、以RR（寄存器 - 寄存器）方式工作、指令周期短
 - 采用大量通用寄存器, 以减少访存次数
 - 采用组合逻辑电路控制, 不用或少用微程序控制
 - 采用优化的编译系统, 力求有效地支持高级语言程序



7. 一些观点与看法

- ① 简单来自于规整, 《大道至简——RISC-V架构之魂》
- ② 指令在程序中出现的频率是不同的, 因此尽量加快常用操作的速度
- ③ 好的设计需要在各种因素中进行权衡
- ④ 开放指令集可降低门槛, 芯片的发展方向之一
- ⑤ 高级语言中的不同的结构对应不同类别的指令
 - 算术运算指令对应于计算语句
 - 数组或结构等数据操作时, 需要访存指令
 - 条件分支指令用于if语句和循环结构
 - 无条件转移用于Case/Switch语句结构
 - 调用指令、入/出栈指令用于过程调用



- ❑ In this paper, we study the design of the ISA for NN accelerators, inspired by the success of **RISC ISA** design principles.
- ❑ The Cambricon is a **load-store architecture** which only allows the main memory to be accessed with load/store instructions.

Table I. An overview of Cambricon instructions.

Instruction Type		Examples	Operands
Control		jump, conditional branch	register (scalar value), immediate
Data Transfer	Matrix	matrix load/store/move	register (matrix address/size, scalar value), immediate
	Vector	vector load/store/move	register (vector address/size, scalar value), immediate
	Scalar	scalar load/store/move	register (scalar value), immediate
Computational	Matrix	matrix multiply vector, vector multiply matrix, matrix multiply scalar, outer product, matrix add matrix, matrix subtract matrix	register (matrix/vector address/size, scalar value)
	Vector	vector elementary arithmetics (add, subtract, multiply, divide), vector transcendental functions (exponential, logarithmic), dot product, random vector generator, maximum/minimum of a vector	register (vector address/size, scalar value)
	Scalar	scalar elementary arithmetics, scalar transcendental functions	register (scalar value), immediate
Logical	Vector	vector compare (greater than, equal), vector logical operations (and, or, inverter), vector greater than merge	register (vector address/size, scalar)
	Scalar	scalar compare, scalar logical operations	register (scalar), immediate

B0911006Y-01 2018-2019学年春季学期

计算机组成原理

第17讲 CPU的结构和功能

CPU的结构、中断与异常、指令周期、
指令流水

主讲教师：张 科

2019年4月29日



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



中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY, CAS

第 8 章 CPU 的结构和功能

8.1 CPU 的结构

强调CPU核与核外

8.2 指令周期

8.3 指令流水

8.4 中断系统

后续集中讲解

8.1 CPU 的结构

一、CPU 的功能

1. 控制器的功能

取指令

指令控制

分析指令

操作控制

执行指令，发出各种操作命令

时间控制

控制程序输入及结果的输出

总线管理

处理异常情况和特殊请求

处理中断

2. 运算器的功能

数据加工

实现算术运算和逻辑运算

异常(Exception)和中断(Interrupt)

- 程序执行过程中CPU会遇到一些特殊情况，使正在执行的程序被“打断/中断”
 - 此时，CPU中止原来正在执行的程序，转到处理异常情况或特殊事件的程序去执行，执行后再返回到原被中止的程序处（断点）继续执行
- 程序执行被“中断”的事件有两类
 - 内部“异常”：在CPU内部发生的意外事件或特殊事件
按发生原因分为硬故障中断和程序性中断两类
硬故障中断：如电源掉电、硬件线路故障等
程序性中断：执行某条指令时发生的“例外”，如算术溢出、缺页、越界、越权、非法指令、除数为0、堆栈溢出、访问超时、断点设置、单步、系统调用等
 - 外部“中断”：在CPU外部发生的特殊事件
通过“中断请求”信号向CPU请求处理。如实时钟、控制台、打印机缺纸、外设准备好、采样计时到、DMA传输结束等
- 通常，异常指控制流中任何意外改变，中断特指来自外部事件，中断处理(Interrupt handling, 或称中断服务程序)用来处理异常和中断
- CPU周而复始执行指令
 - 执行指令过程中，若发现异常，则立即转异常处理（或作出检测标记，记录原因，并到指令最后阶段再处理）
 - 每个指令结束，查询有没有中断请求，有则响应中断

二、CPU 结构框图

8.1

1. CPU 与系统总线

指令控制

PC IR

操作控制

CU 时序电路

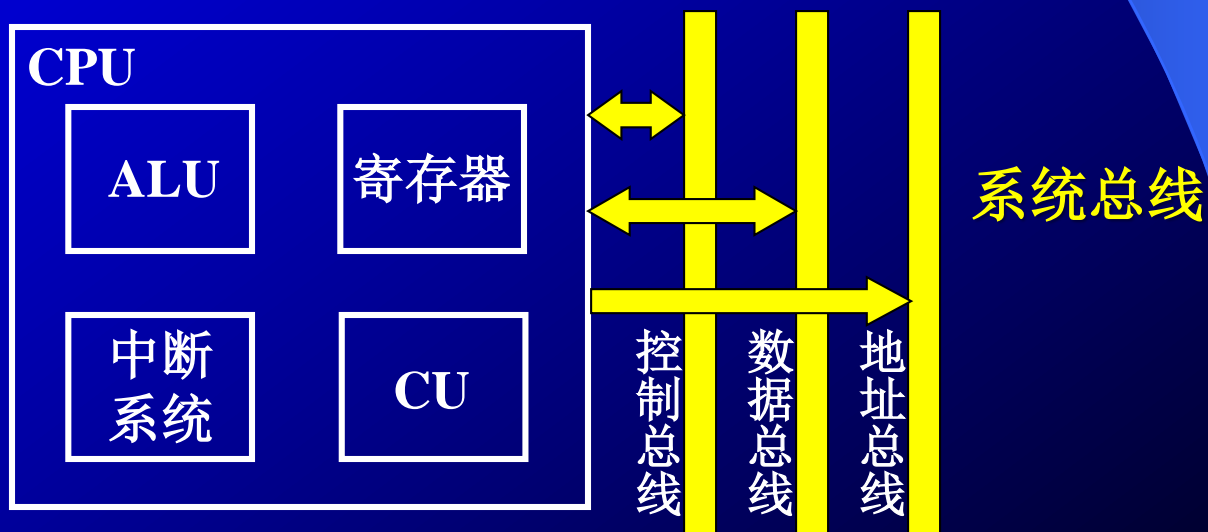
时间控制

数据加工

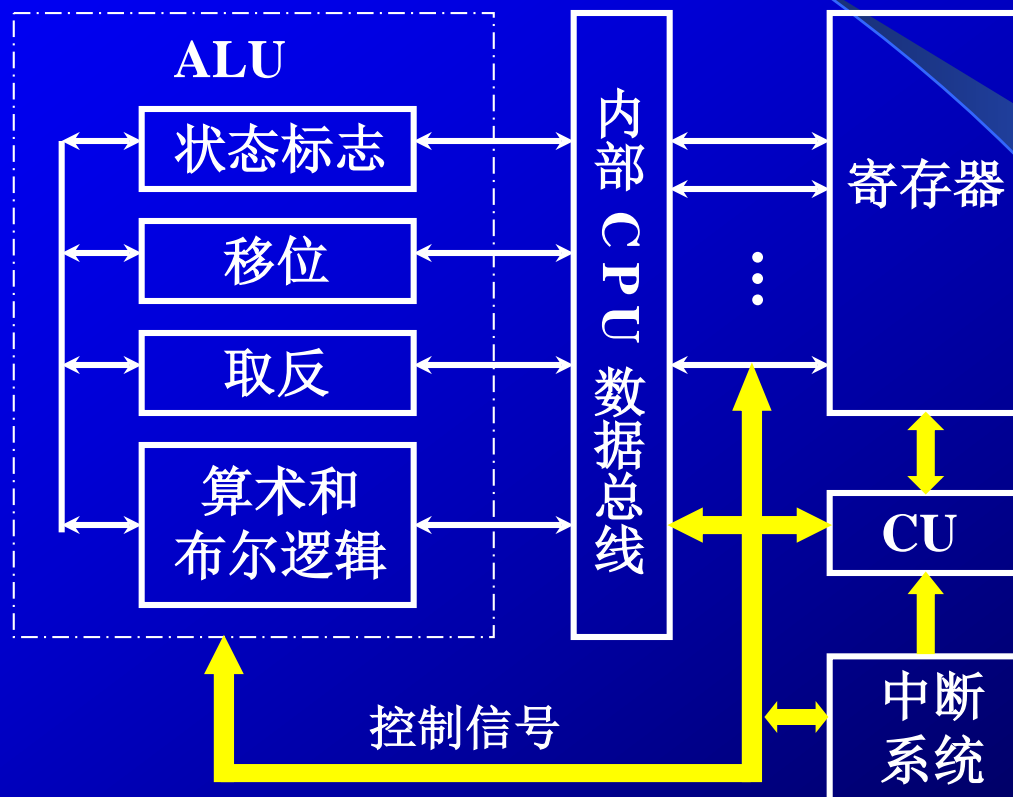
ALU 寄存器

处理中断

中断系统



2. CPU 的内部结构



三、CPU 的寄存器

8.1

1. 用户可见寄存器（CPU执行机器语言访问的寄存器）

(1) 通用寄存器

存放操作数

可作 某种寻址方式所需的 专用寄存器

如，BR/IX/SP

(2) 数据寄存器

存放操作数（满足各种数据类型）

两个寄存器拼接存放双倍字长数据

(3) 地址寄存器

存放地址，其位数应满足最大的地址范围

用于特殊的寻址方式 段基值/址 栈指针

(4) 条件码寄存器

存放条件码，可作程序分支的依据

标志寄存器

如 正、负、零、溢出、进位等

2. 控制和状态寄存器

Eg. CSR in RISC-V

8.1

(1) 控制寄存器

控制 CPU 操作，例如CPU与Mem交互信息

PC → **MAR** → **M** → **MDR** → **IR**

其中 **MAR**、**MDR**、**IR** 用户不可见

PC 用户可见

(2) 状态寄存器

状态寄存器 存放条件码

PSW 寄存器 存放程序状态字

中断标记寄存器 供中断系统使用

3. 举例 Z8000 8086 MC 68000（自学）

四、控制单元 CU 和中断系统

1. CU 产生全部指令的微操作命令序列

组合逻辑设计

硬连线逻辑

微程序设计

存储逻辑

参见 第9、10章

2. 中断系统

后续将作为独立章节讲解
(参见 第8.4章节 和 第5.5章节)

五、ALU

参见 第 6 章

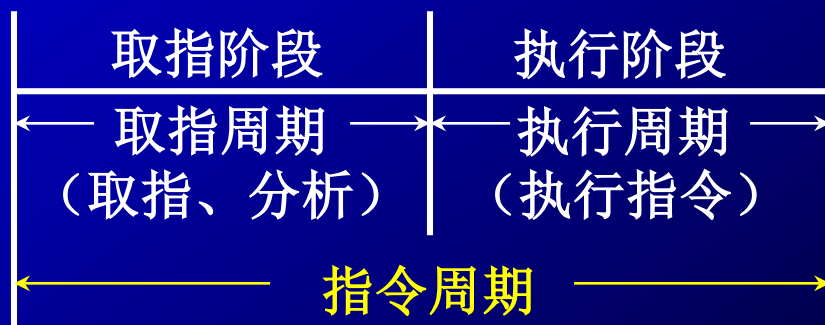
8.2 指令周期

一、指令周期的基本概念

1. 指令周期

取出并执行一条指令所需的全部时间

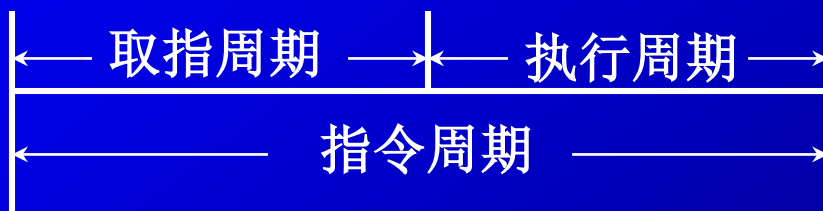
完成一条指令 { 取指、分析 取指周期
 执行 执行周期



2. 每条指令的指令周期不同



NOP



ADD mem



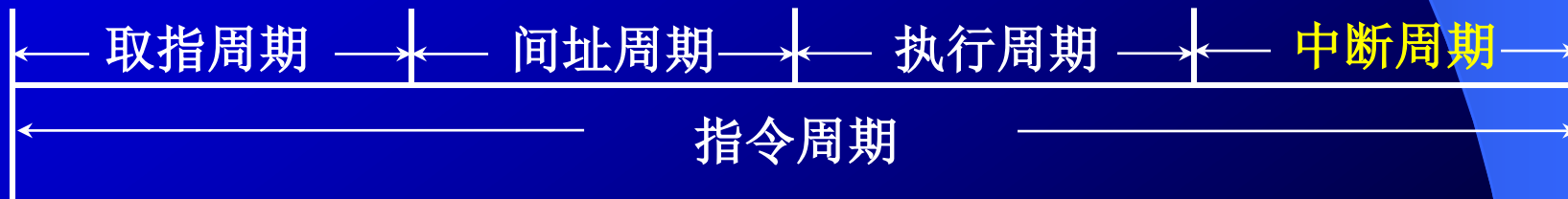
MUL mem

3. 具有间接寻址的指令周期

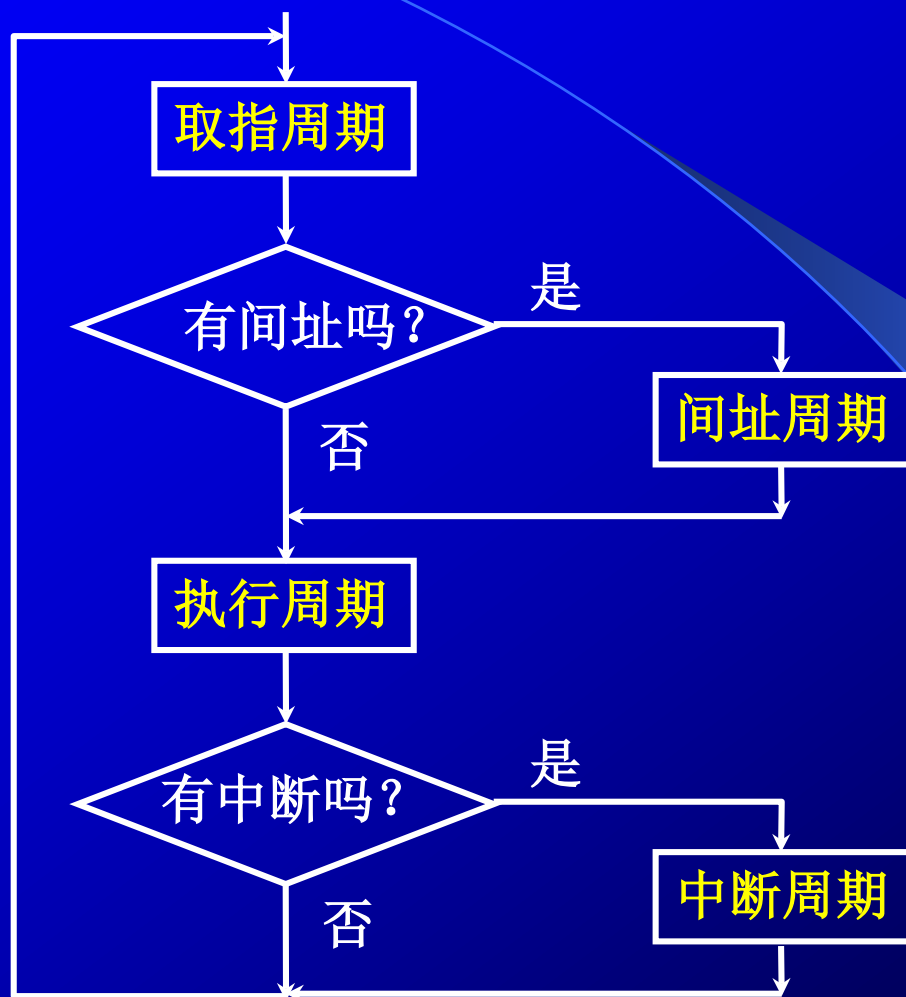
间接寻址的概念请复习7.3章节的寻址方式，教材P312



4. 带有中断周期的指令周期



5. 指令周期流程



指令周期 = 取指周期 + [间址周期] + 执行周期 + [中断周期]

6. CPU 工作周期的标志

CPU 不同的工作周期均会有访存

CPU 的4个工作周期

取指周期FE

取 指令

间址周期IND

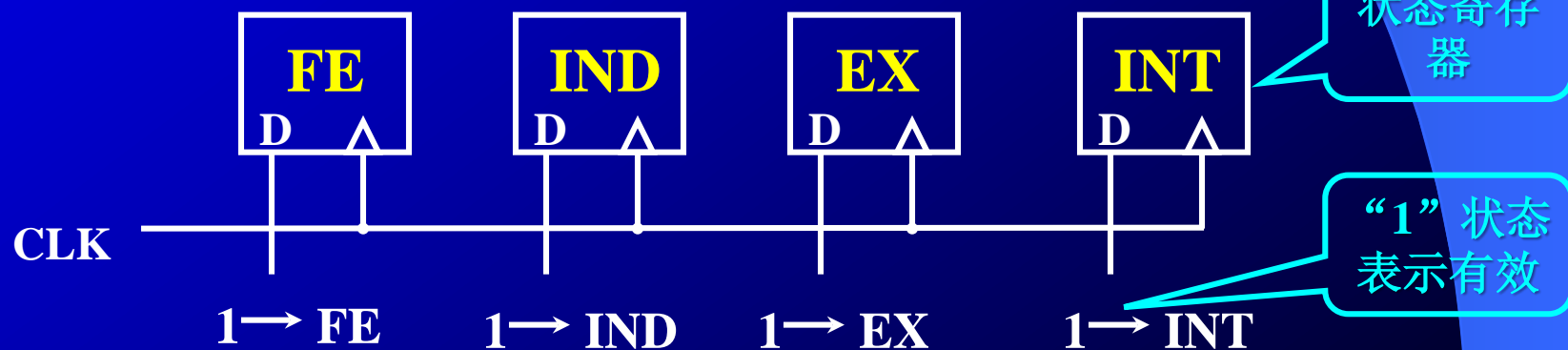
取 有效地址

执行周期EX

取 操作数(1w)

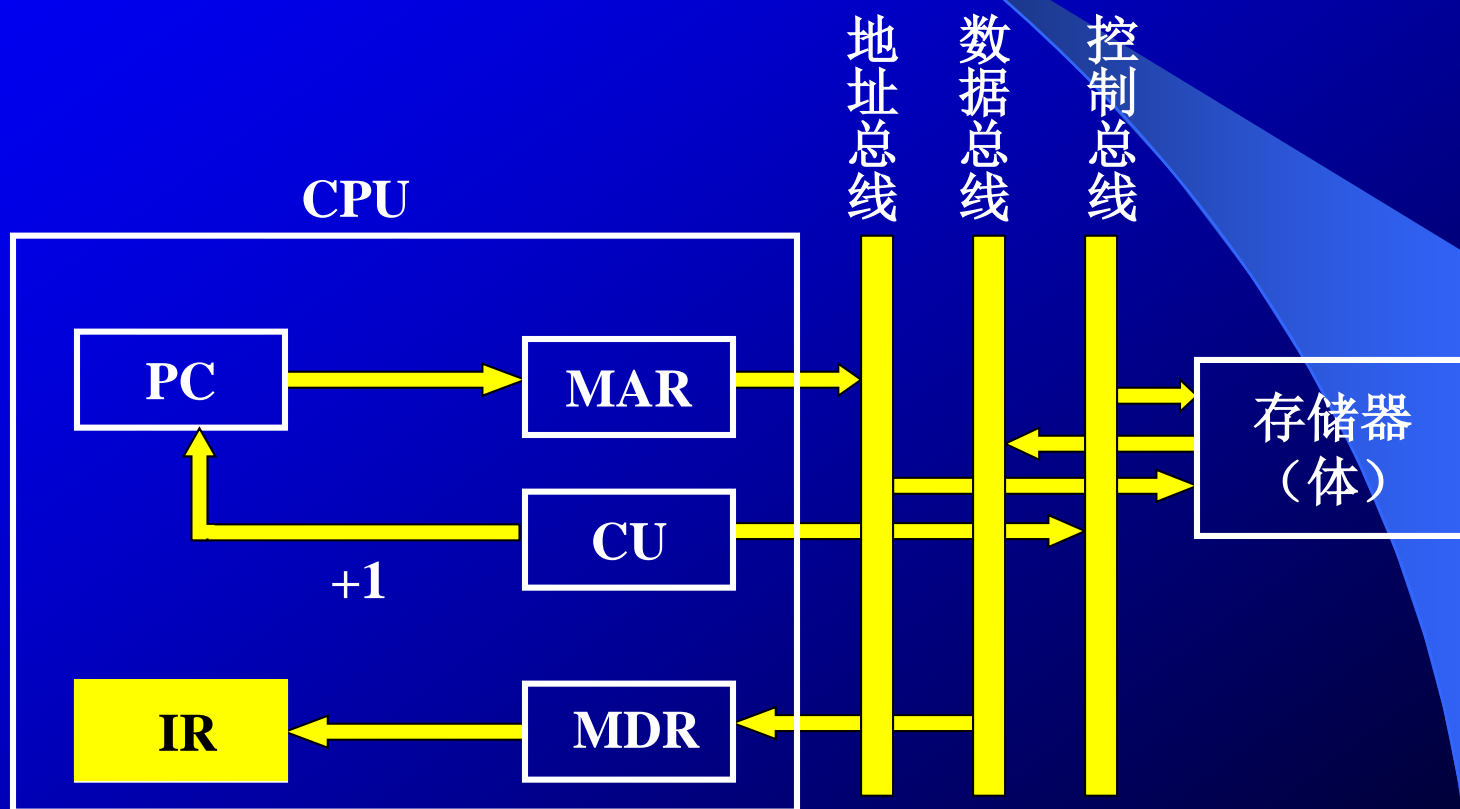
中断周期INT

存 程序断点

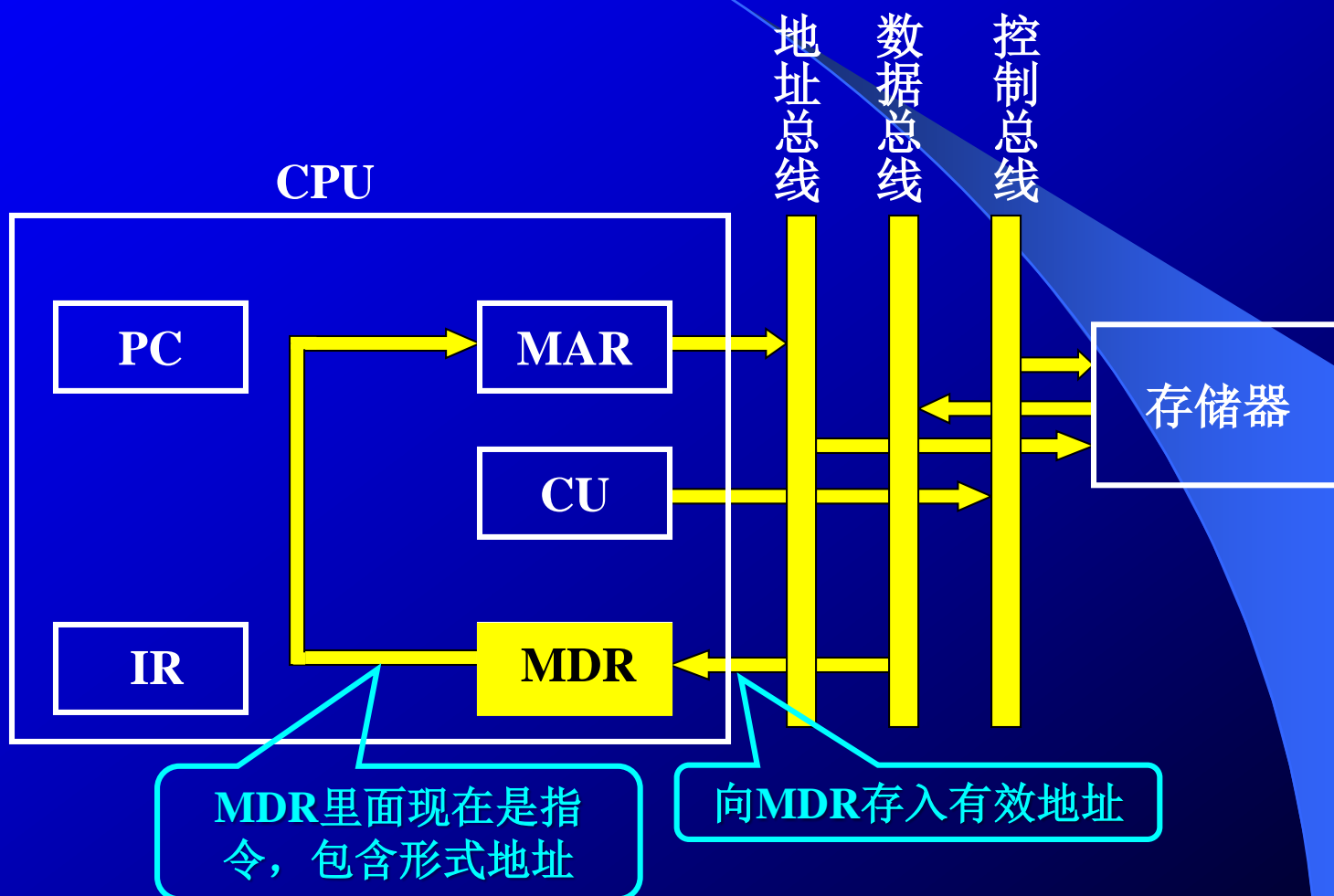


二、指令周期的数据流

1. 取指周期数据流



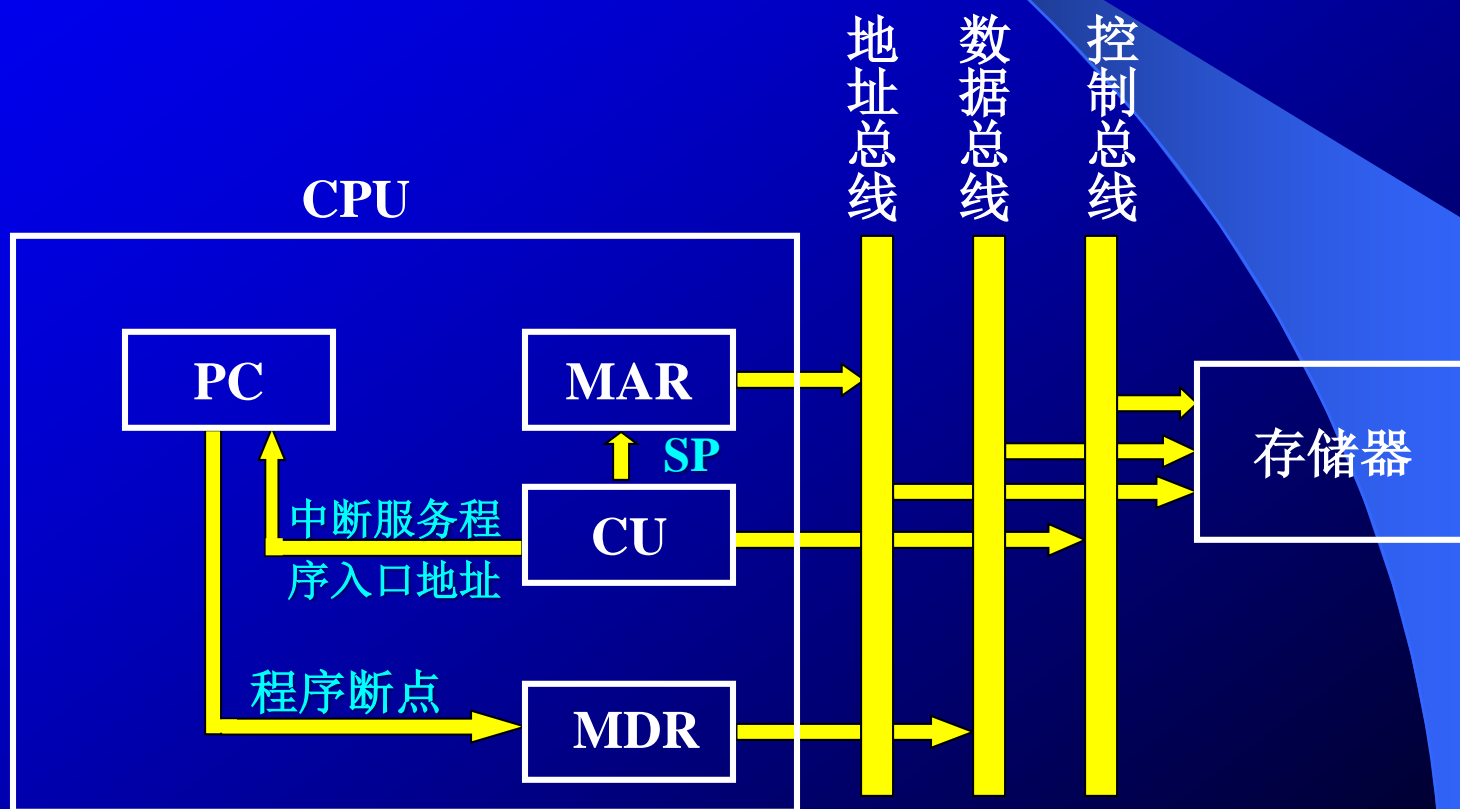
2. 间址周期数据流



3. 执行周期数据流

不同指令的执行周期数据流不同

4. 中断周期数据流



8.3 指令流水

一、如何提高机器速度

1. 提高访存速度（参见第4章）

高速芯片

Cache

多体并行

2. 提高 I/O 和主机之间的传送速度（参见第5章）

中断

DMA

通道

I/O 处理机

多总线

3. 提高运算器速度（第6章）

高速芯片

改进算法

快速进位链

• 进一步提高整机处理能力

高速器件

改进系统结构，开发系统的并行性

两个“处理部件”流水工作提高并行性



二、系统的并行性 (Parallelism)

8.3

1. 并行的概念

并行 { **并发** (Concurrent) 两个或两个以上事件在 **同一时间段** 发生
同时 (狭义Parallel) 两个或两个以上事件在 **同一时刻** 发生
时间上互相重叠

2. 并行性的等级

过程级 (程序、进程)	粗粒度	软件实现
指令级 (指令之间) (指令内部)	细粒度	硬件实现

三、指令流水原理

8.3

1. 指令的串行执行



取指令 取指令部件 完成 总有一个部件 空闲
执行指令 执行指令部件 完成

2. 指令的二级流水



指令预取

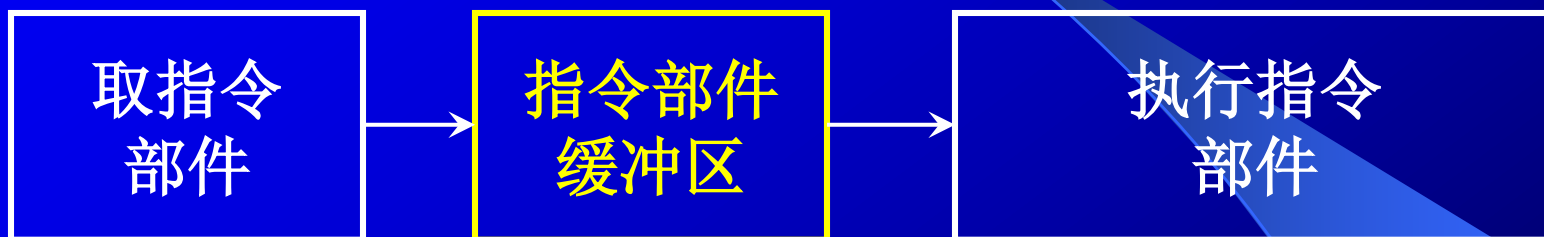
若 取指 和 执行 阶段时间上 完全重叠

整体来看 指令周期 减半 速度提高 1 倍

3. 影响指令流水效率加倍的因素

8.3

(1) 执行时间 > 取指时间



(2) 条件转移指令 对指令流水的影响

必须等 **上条** 指令执行结束，才能确定 **下条** 指令的地址，
造成时间损失

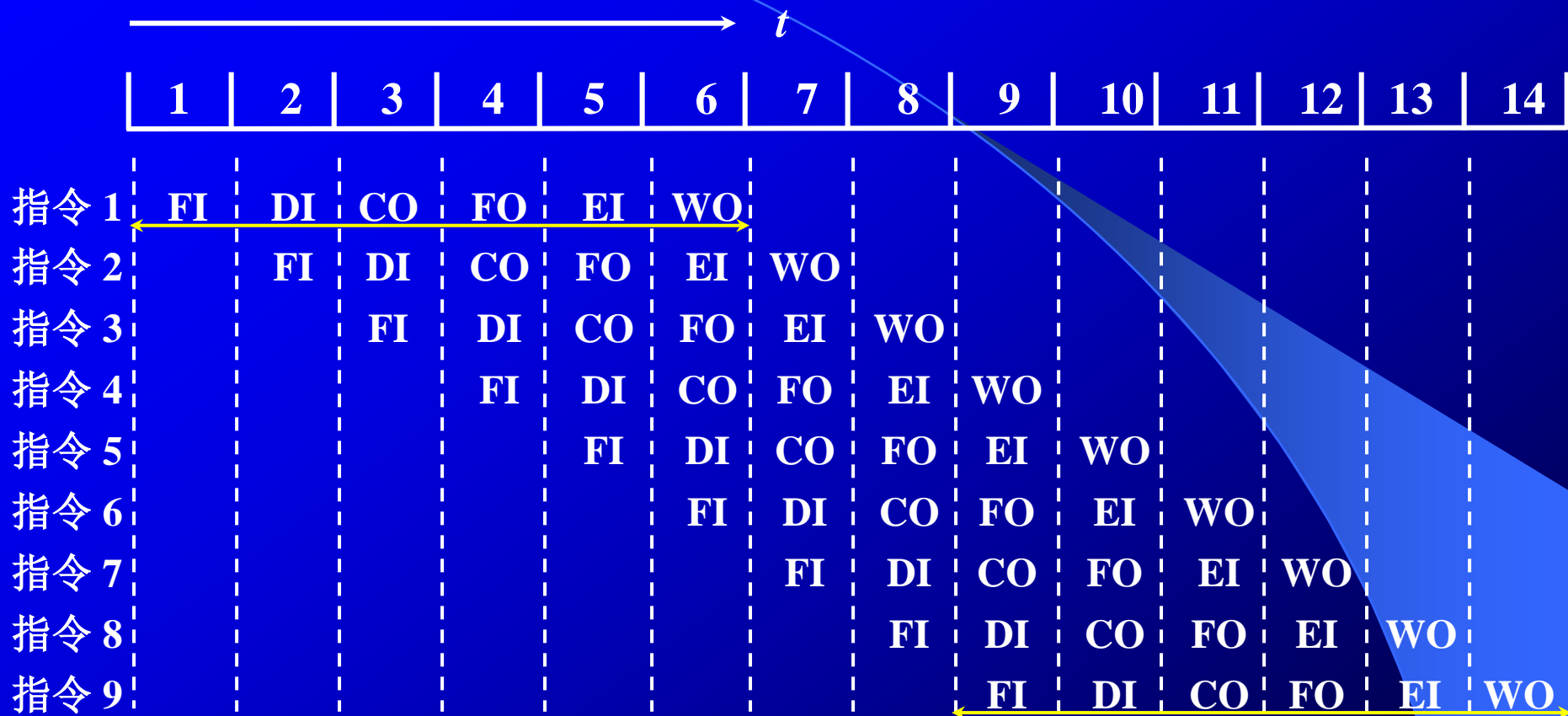
解决办法 ?

猜测法（分支预测）



4. 指令的六级流水

8.3



完成 一条指令

串行执行

六级流水

6 个时间单位

$6 \times 9 = 54$ 个时间单位

14 个时间单位 (理想状态下)

四、影响指令流水线性能的因素

8.3

程序的相近指令之间出现某种关联
使指令流水出现停顿，影响流水线效率

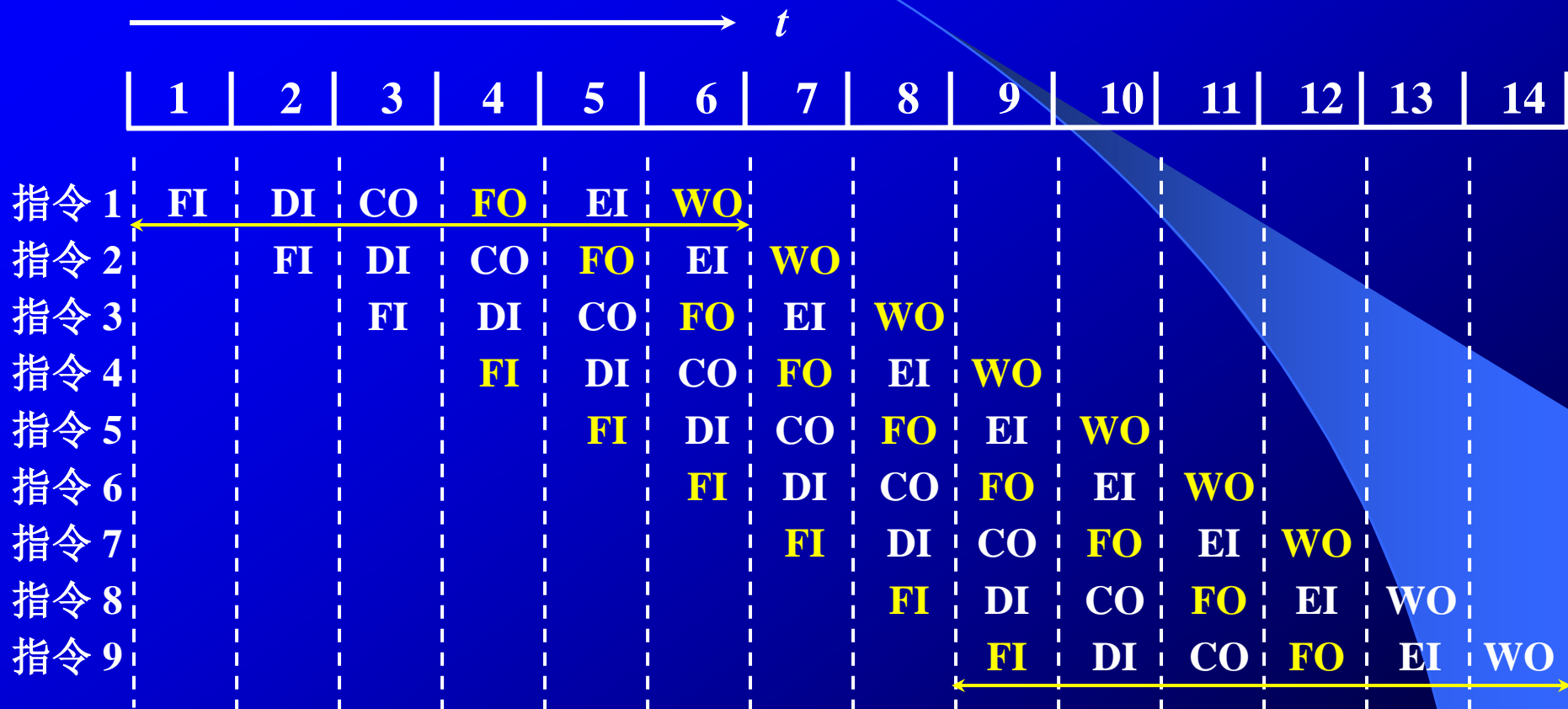


四、影响指令流水线性能的因素

8.3

Structural Hazards

1. 结构相关 不同指令争用同一功能部件产生资源冲突



指令 1 与指令 4 冲突

指令 2 与指令 5 冲突

指令1、指令3、指令 6 冲突

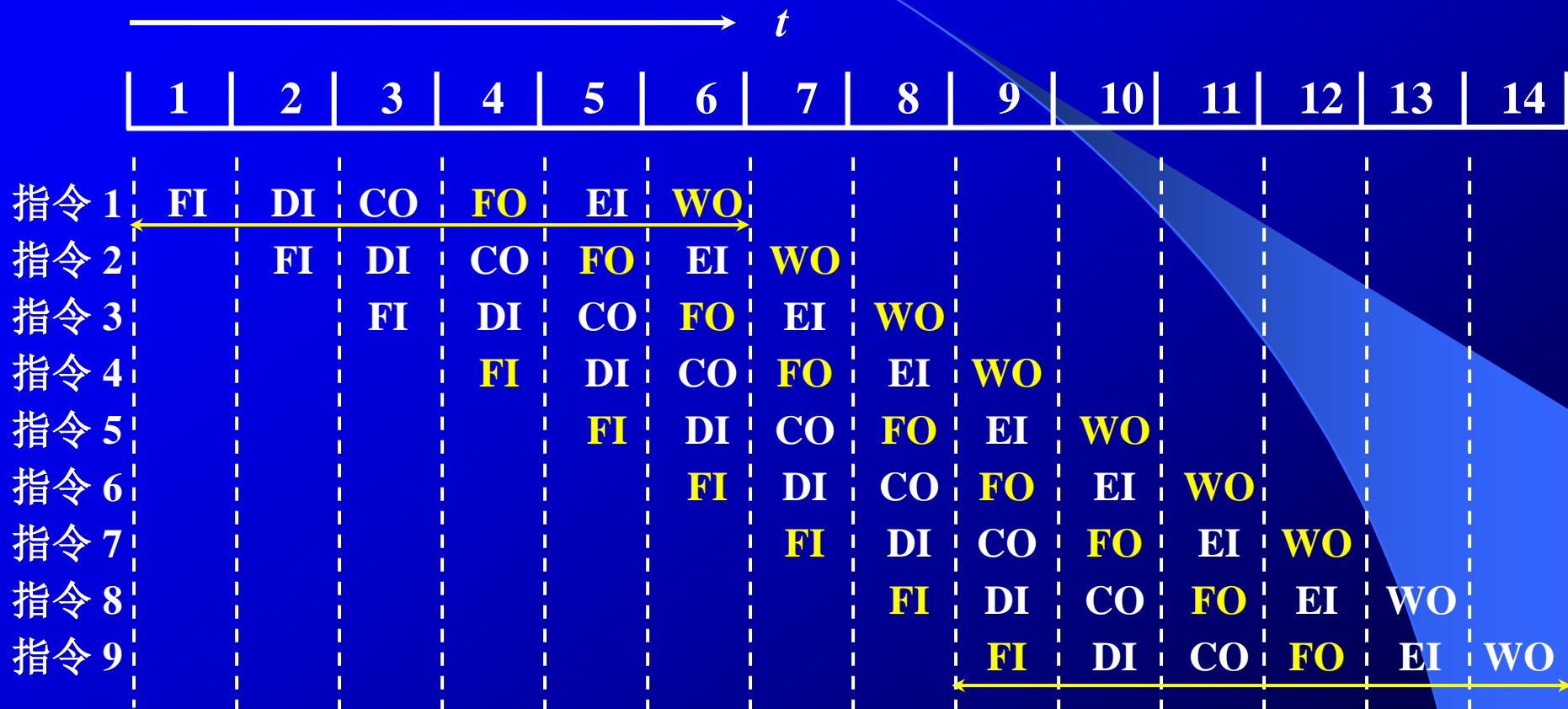
...

四、影响指令流水线性能的因素

8.3

Structural Hazards

1. 结构相关 不同指令争用同一功能部件产生资源冲突



解决办法

- 停顿
- 指令存储器和数据存储器分开
- 指令预取技术（适用于访存周期短的情况）

2. 数据相关

Data Hazards

8.3

不同指令因重叠操作，可能改变操作数的 读/写 访问顺序

- 写后读相关 (RAW)

顺序执行In-order的CPU只可能出现RAW相关

SUB R_1, R_2, R_3 ; $(R_2) - (R_3) \rightarrow R_1$

ADD R_4, R_5, R_1 ; $(R_5) + (R_1) \rightarrow R_4$

- 读后写相关 (WAR)

STA M, R_2 ; $(R_2) \rightarrow M$ 存储单元

ADD R_2, R_4, R_5 ; $(R_4) + (R_5) \rightarrow R_2$

- 写后写相关 (WAW)

乱序执行Out-of-order的CPU
可能出现三种数据相关

MUL R_3, R_2, R_1 ; $(R_2) \times (R_1) \rightarrow R_3$

SUB R_3, R_4, R_5 ; $(R_4) - (R_5) \rightarrow R_3$


解决办法 • 后推法(停顿) • 采用 旁路技术(Forwarding)

3. 控制相关

Control Hazards (Branch Hazards)

8.3

由转移指令引起



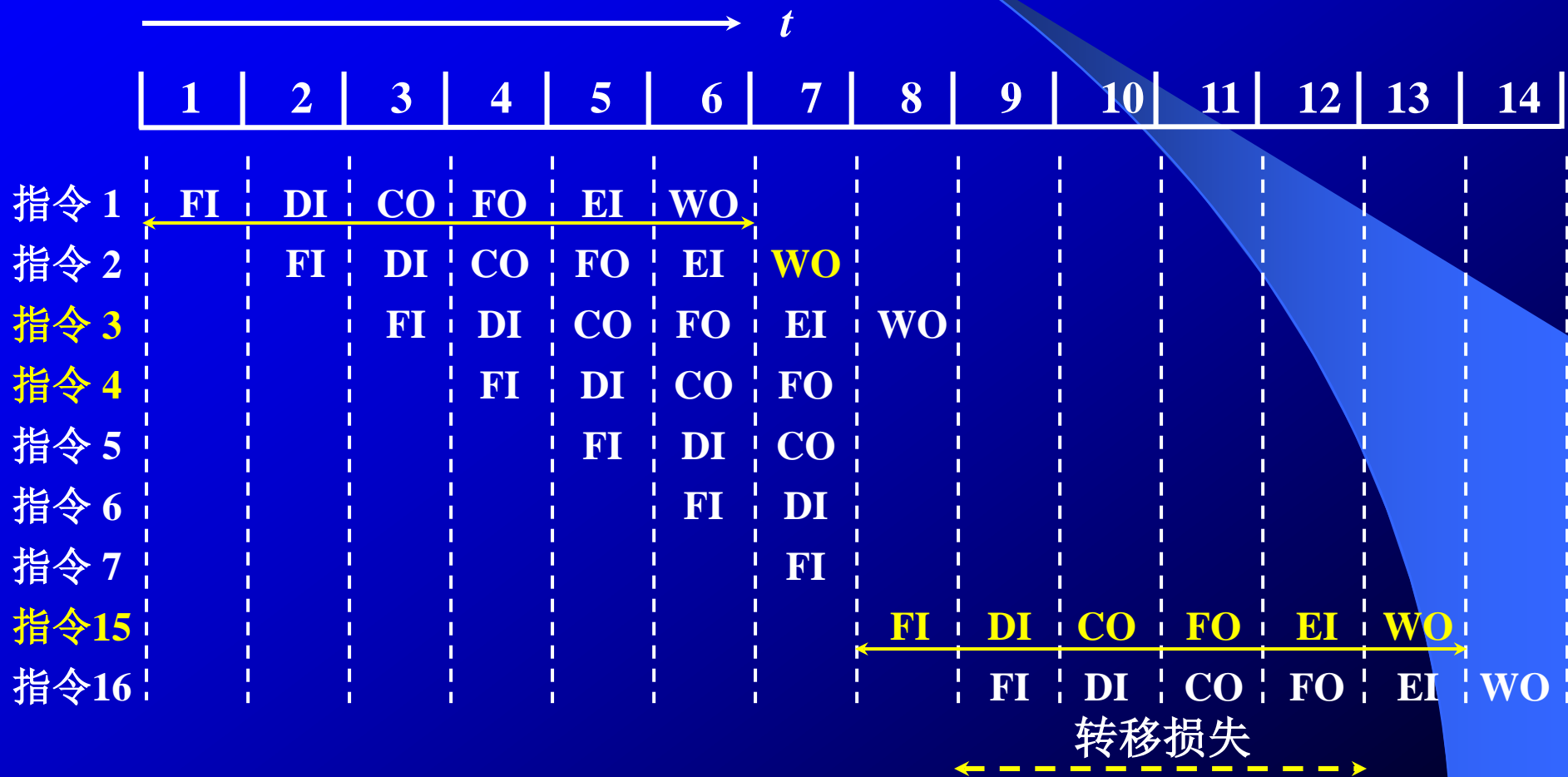
LDA # 0
LDX # 0
M ADD X, D
INX
CPX # N
BNE M
DIV # N
STA ANS

BNE 指令必须等
CPX 指令的结果
才能判断出
是转移
还是顺序执行

3. 控制相关

8.3

设 指令3 是转移指令



五、流水线性能

8.3

1. 吞吐率

单位时间内流水线所完成指令或输出结果的数量

设 m 段的流水线各段时间为 Δt

- ## • 最大吞吐率

$$T_{pmax} = \frac{1}{\Delta t}$$

- ## • 实际吞吐率



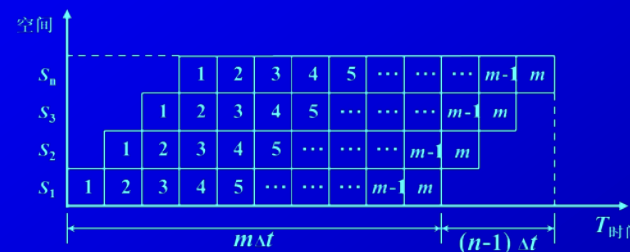
连续处理 n 条指令的吞吐率为

$$T_p = \frac{n}{m \cdot \Delta t + (n-1) \cdot \Delta t}$$



2. 加速比 S_p

8.3



m 段的流水线的速度与等功能的非流水线的速度之比

设流水线各段时间为 Δt

完成 n 条指令在 m 段流水线上共需

$$T = m \cdot \Delta t + (n-1) \cdot \Delta t$$

完成 n 条指令在等效的非流水线上共需

$$T' = n \cdot m \cdot \Delta t$$

则

$$S_p = \frac{nm \cdot \Delta t}{m \cdot \Delta t + (n-1) \cdot \Delta t} = \frac{nm}{m + n - 1}$$

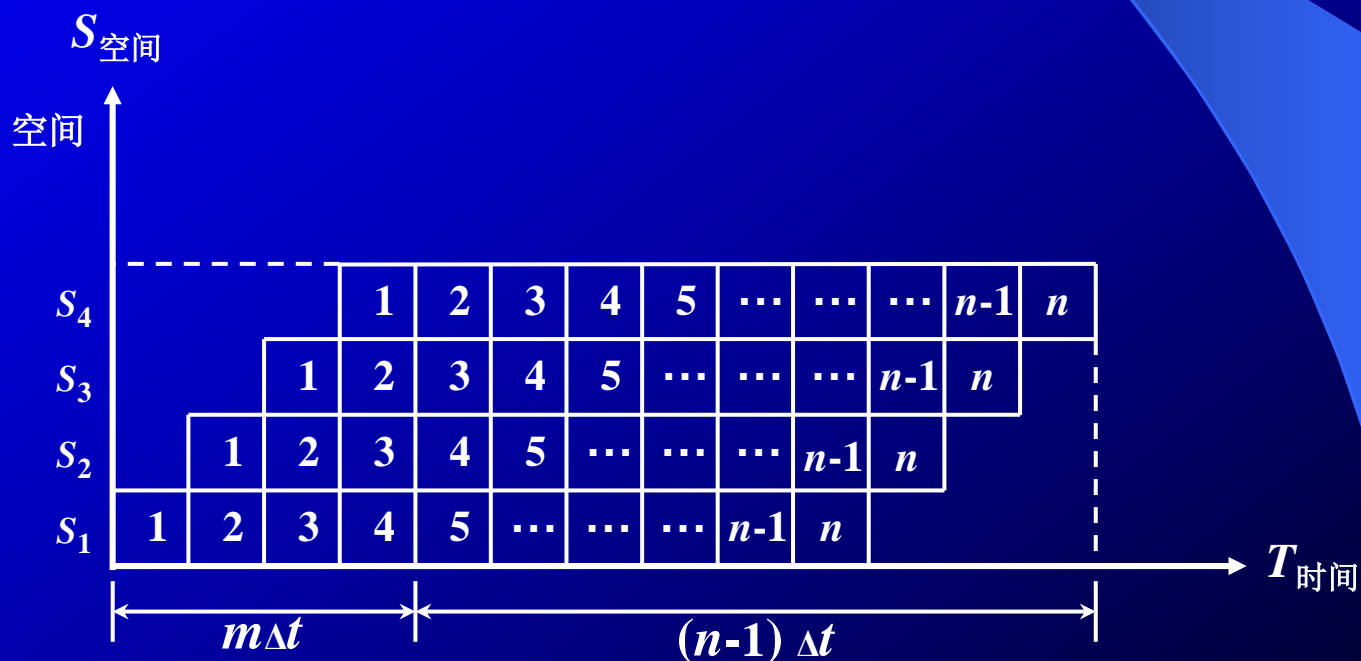
3. 效率

8.3

流水线中各功能段的 **利用率**

由于流水线有 **建立时间** 和 **排空时间**

因此各功能段的 **设备不可能** 一直处于 **工作** 状态



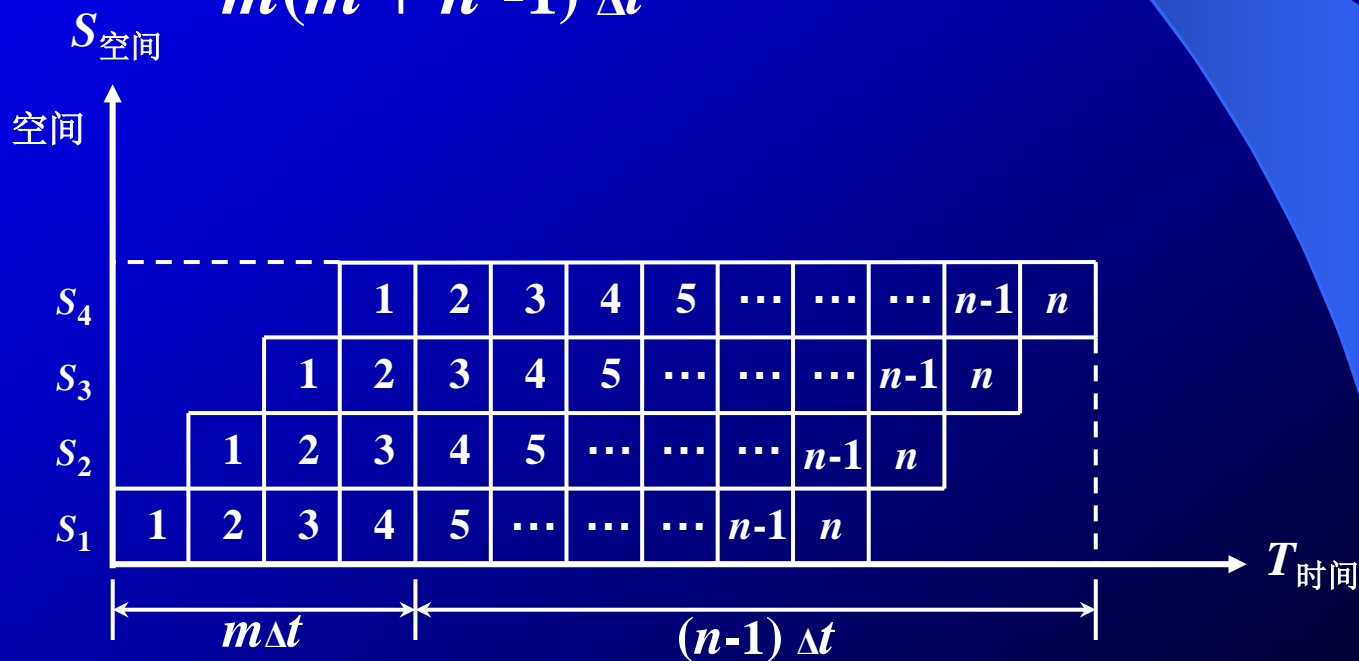
3. 效率

8.3

流水线中各功能段的 **利用率**

效率 = $\frac{\text{流水线各段处于工作时间的时空区}}{\text{流水线中各段总的时空区}}$

$$= \frac{mn\Delta t}{m(m+n-1)\Delta t}$$



六、流水线的多发技术

8.3

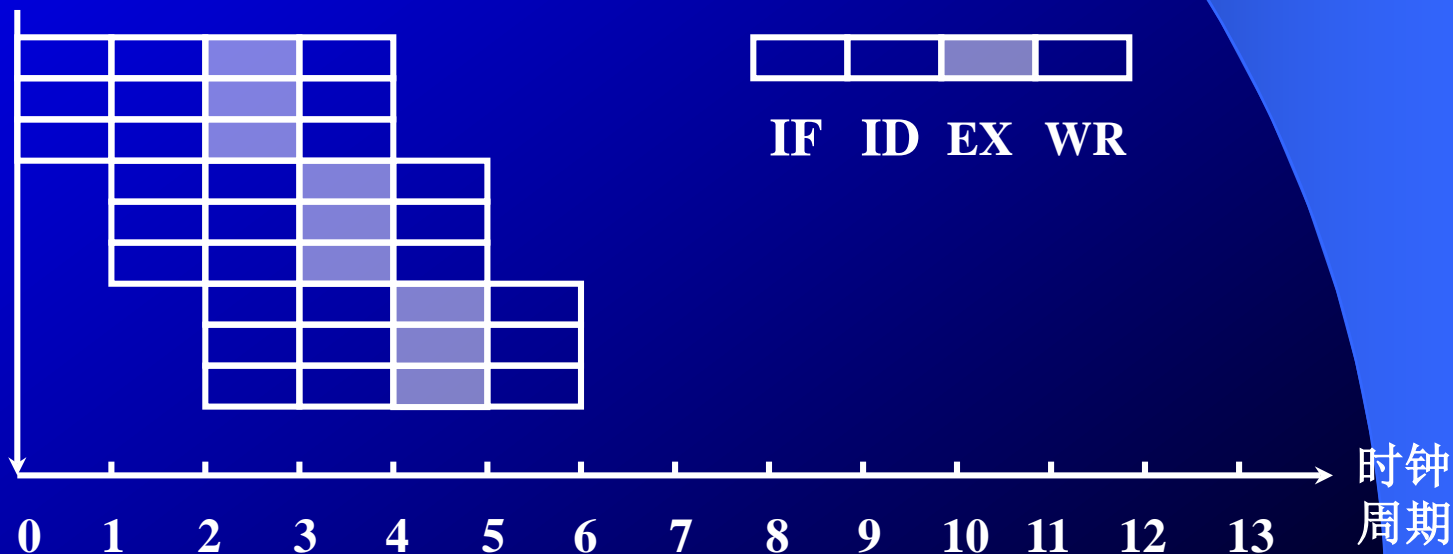
1. 超标量技术 (Superscalar)

- 每个时钟周期内可 **并发多条独立指令**
配置多个功能部件

- **不能调整** 指令的 **执行顺序**

通过编译优化技术，把可并行执行的指令搭配起来

指令序列



2. 超流水线技术 (Super-pipelining)

8.3

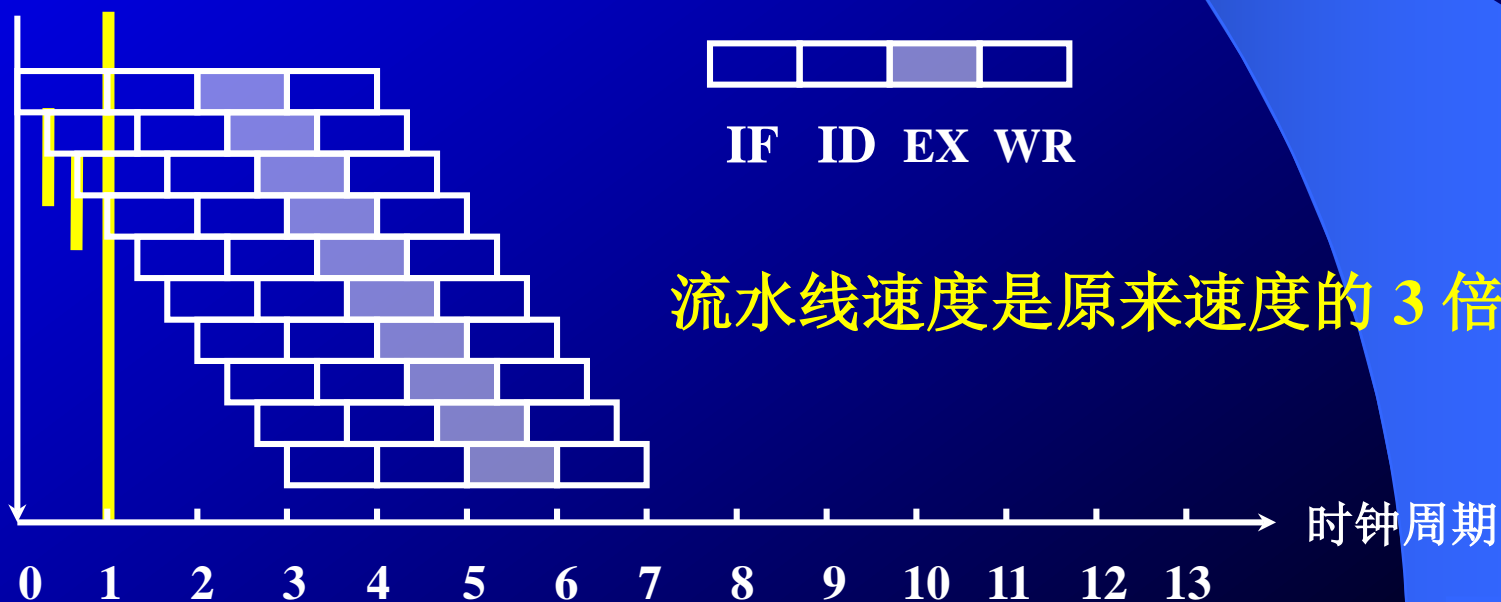
- 在一个时钟周期内再分段（3段）

在一个时钟周期内 一个功能部件使用多次（3次）

- 不能调整 指令的执行顺序

靠编译程序解决优化问题

指令序列

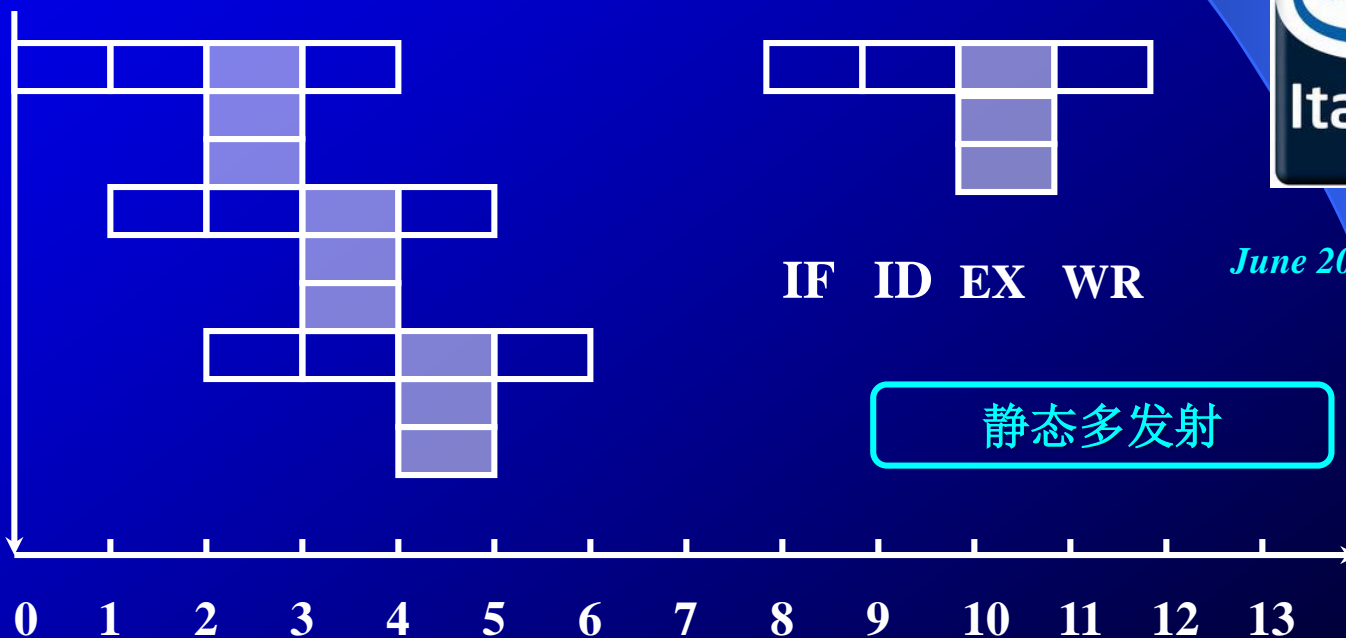


3. 超长指令字技术(VLIW-Very Long Instruction Word)

8.3

- 由编译程序 **挖掘** 出指令间 **潜在的** **并行性**，
将 **多条** 能 **并行操作** 的指令组合成 **一条**
具有 **多个操作码字段** 的 **超长指令字**（可达几百位）
- 采用 **多个处理部件**

指令序列



IA-64

June 2001~Feb 2017

时钟周期

动态多发射

- 动态多发射流水线的执行模式
 - 顺序发射、顺序执行
 - 顺序发射、乱序执行
 - 乱序发射、乱序执行
- Pentium 4 动态多发射流水线
 - 超流水、超标量、动态调度、乱序发射、乱序完成
 - 20级以上超流水线、3发射超标量、2个队列动态调度、126条微操作同时执行
- 体系结构课程继续讲授

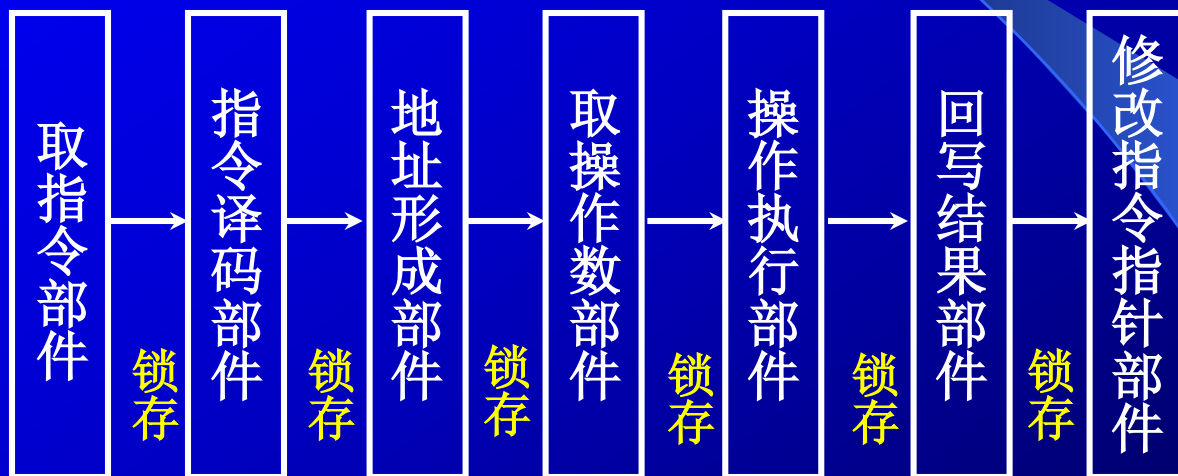


七、流水线结构

8.3

1. 指令流水线结构

完成一条指令分 **7 段**，每段需一个时钟周期



若 **流水线不出现断流**

1 个时钟周期出 **1** 结果

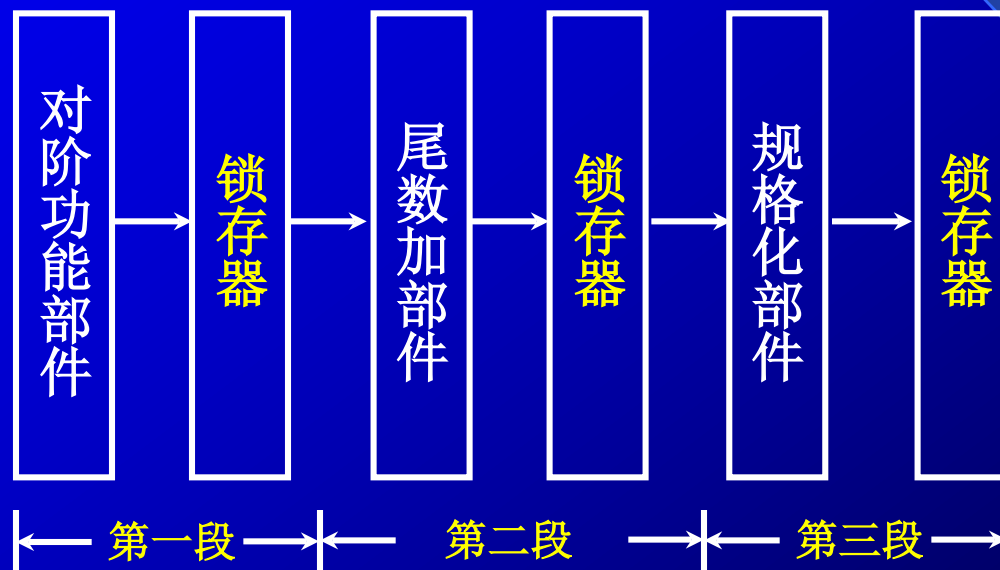
不采用流水技术

7 个时钟周期出 **1** 结果

理想情况下，**7 级流水** 的速度是不采用流水技术的 **7 倍**

2. 运算流水线

完成 浮点加减 运算 可分
对阶、尾数求和、规格化 三段



分段原则 每段 操作时间 尽量 一致

作业

- 习题： 8.4, 8.8, 8.11, 8.12
- 本次作业提交截止时间：
 - 请于5月6日上课前提交
- 复习教材第8章，预习第9章



Q & A ?



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



中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY, CAS