计算机组成原理

总结——图说COD

llxx@ustc.edu.cn

国产CPU

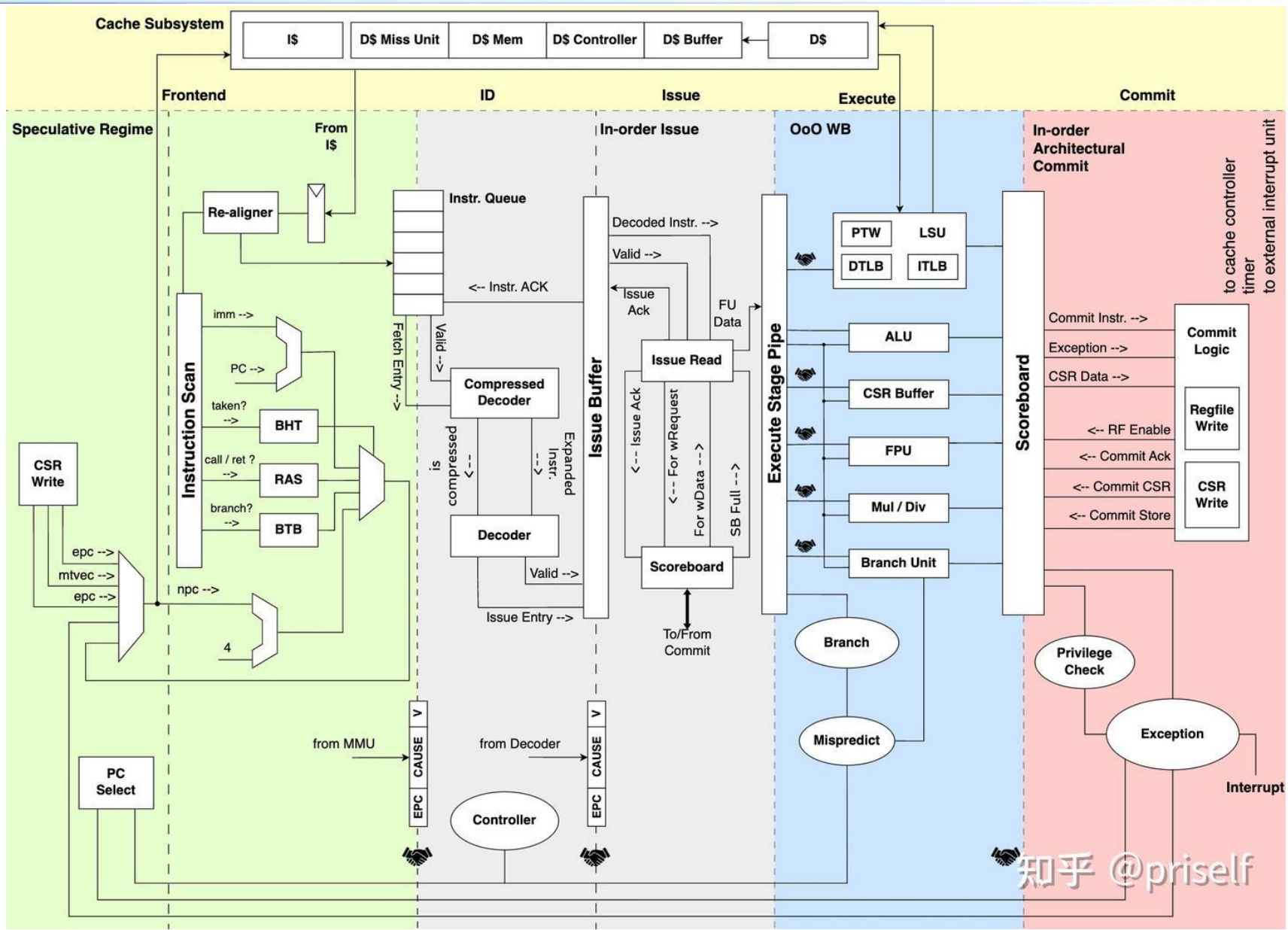
梯队	厂商	技术路线	市场份额	核心标签	主战场
两超	海光信息	x86兼容	28.4%	商业之王、CPU+DCU双轮	金融、电信、商业数据中心
	华为鲲鹏	ARM架构	23.1%	全栈生态、软硬协同	政务云、运营商、智算中心
三强	龙芯中科	LoongArch自主	~15%	100%自主、安全护城河	党政、军工、关键基础设施
	飞腾信息	ARM架构	~12%	政务根基、移动端破局	党政办公、笔记本、5G基站
	兆芯集成	x86兼容	~8%	Windows兼容、性价比	商业办公、教育、工控
一新	申威	SW64自主	~3%	超算王者、特种应用	国家超算、国防军工
黑马	RISC-V阵营	RISC-V开源	~5%	AI原生、未来可期	AI推理、边缘计算、物联网

龙芯中科，国内唯一全栈自主CPU指令集的开发者和拥有者

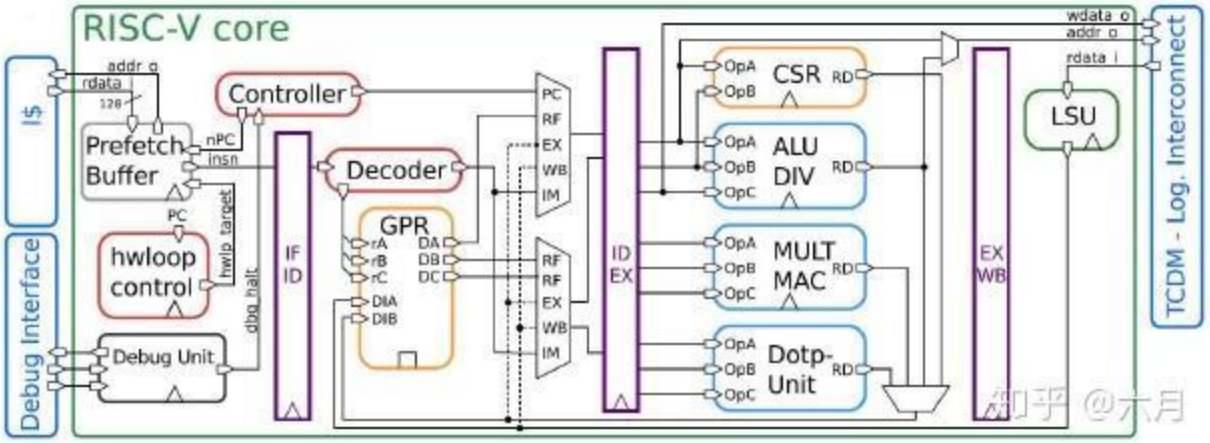
开源处理器

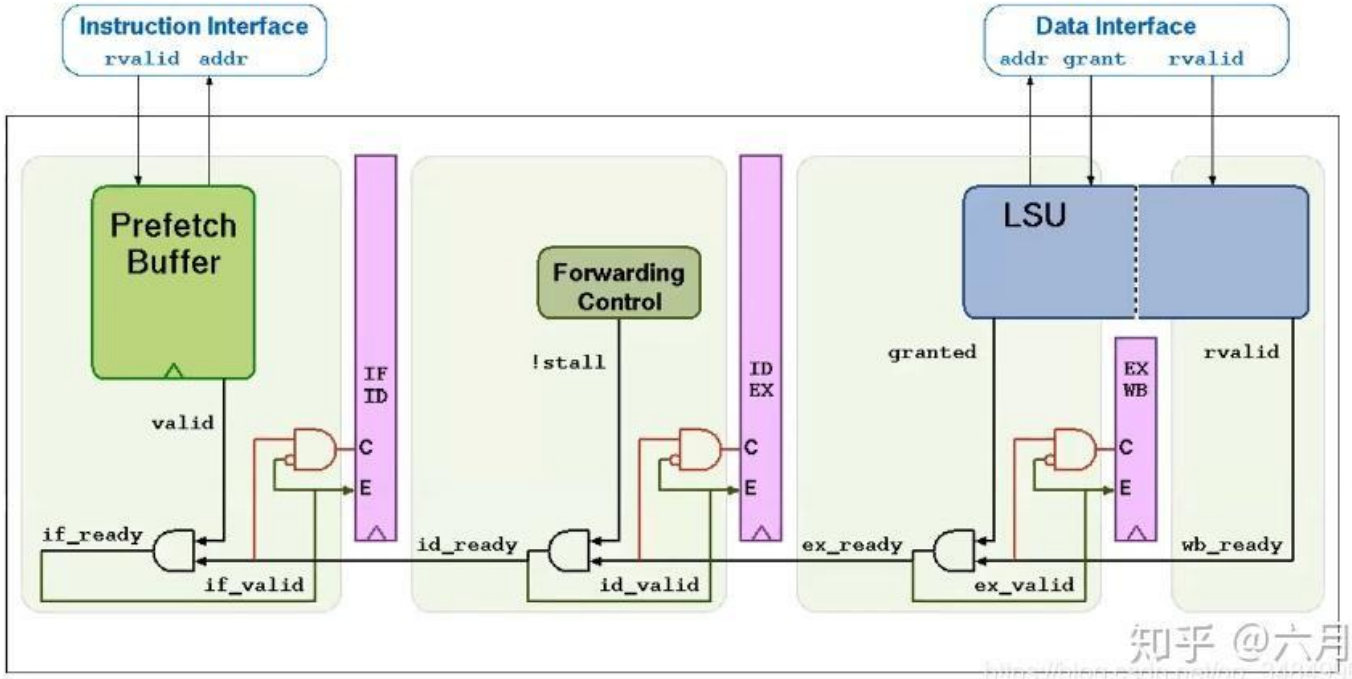
- Rocketchip generator：Chisel语言，UCB，Sifive
  - a parametric system on chip (SoC) generator
    - single **application-class** cores
      - Rocket：a single-issue, in-order core
      - BOOM：an out-of-order, superscalar core
    - cache-coherent multicore systems【例？】
      - TileLink总线
- PULP平台：并行超低功耗，嵌入式系统，Benini@ETH
  - Ariane核：InO单发射，OoO，InO提交，6段流水线
  - RI5CY核：4级InO流水线
  - PULP SOC：多核
  - PULPino SOC：单核，microcontroller-class
- 香山：中科院计算所，多核?
- 蜂鸟：microcontroller-class

Ariane CPU μArch：2024



RI5CY：取指、译码、执行、回写





PULPino： microcontroller-class

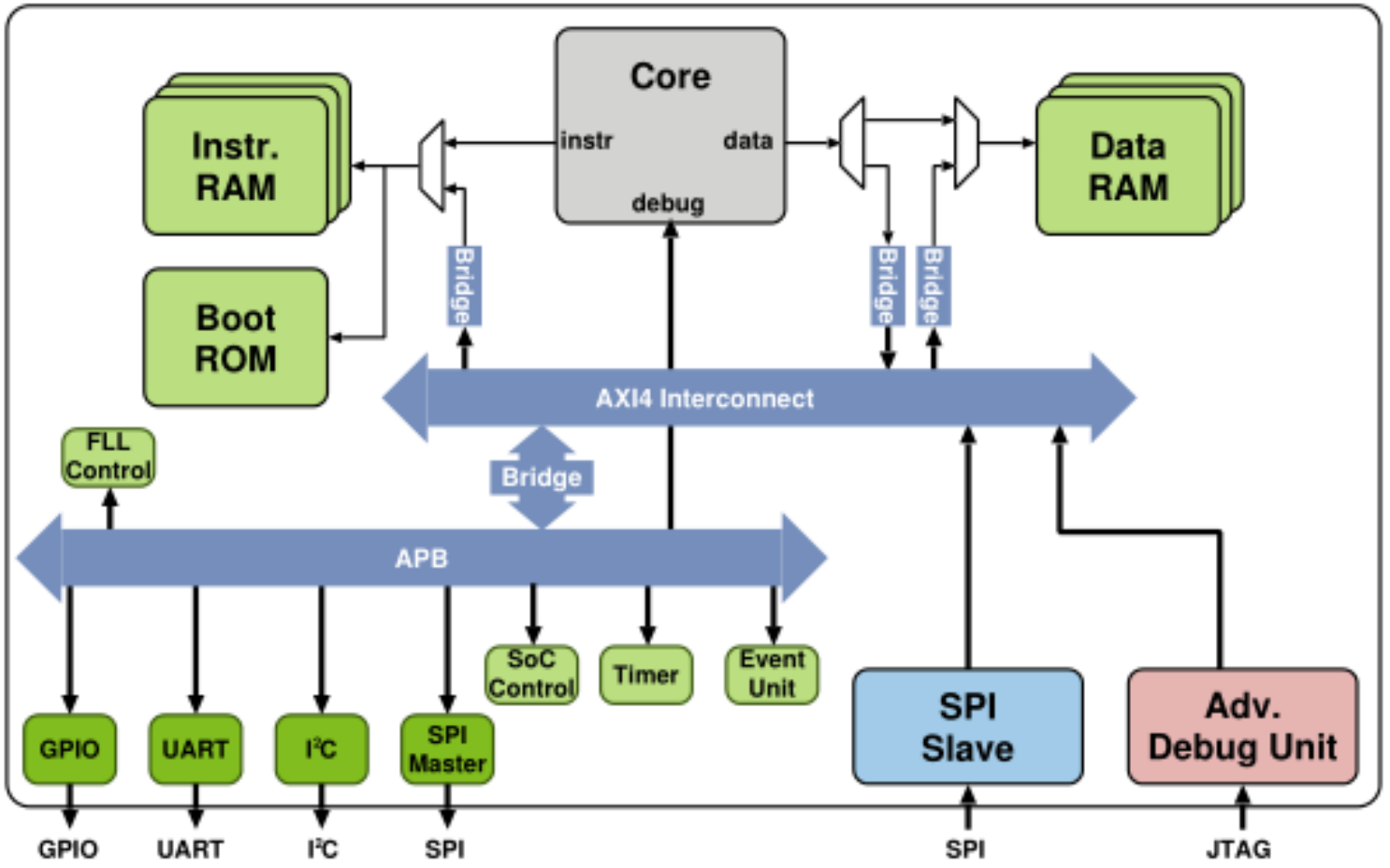
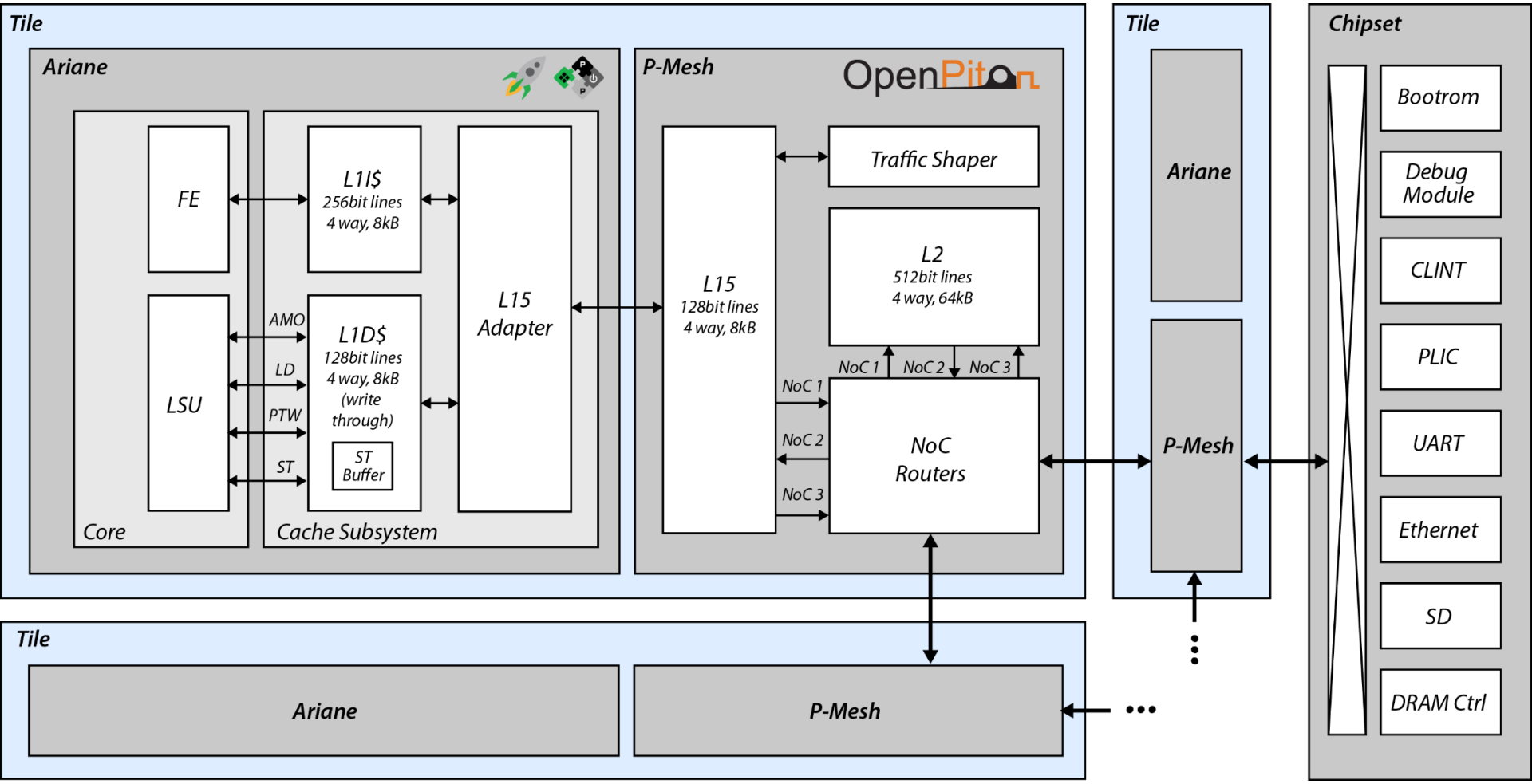


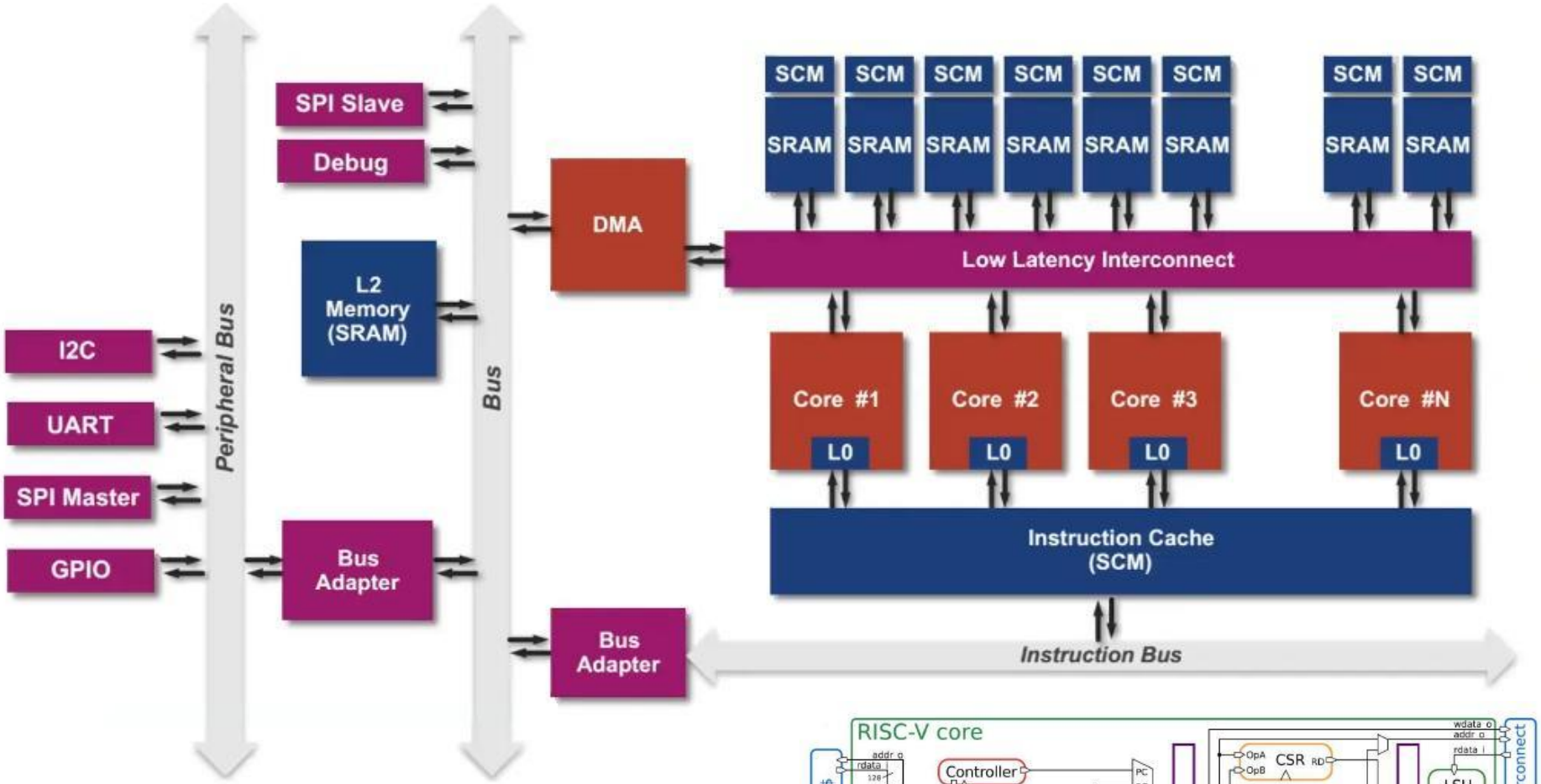
Figure 1.1: PULPino Overview.

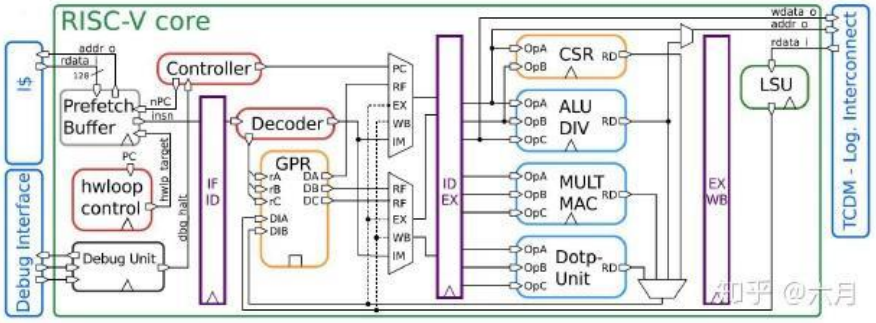
多核：cva6，多级Cache，NoC



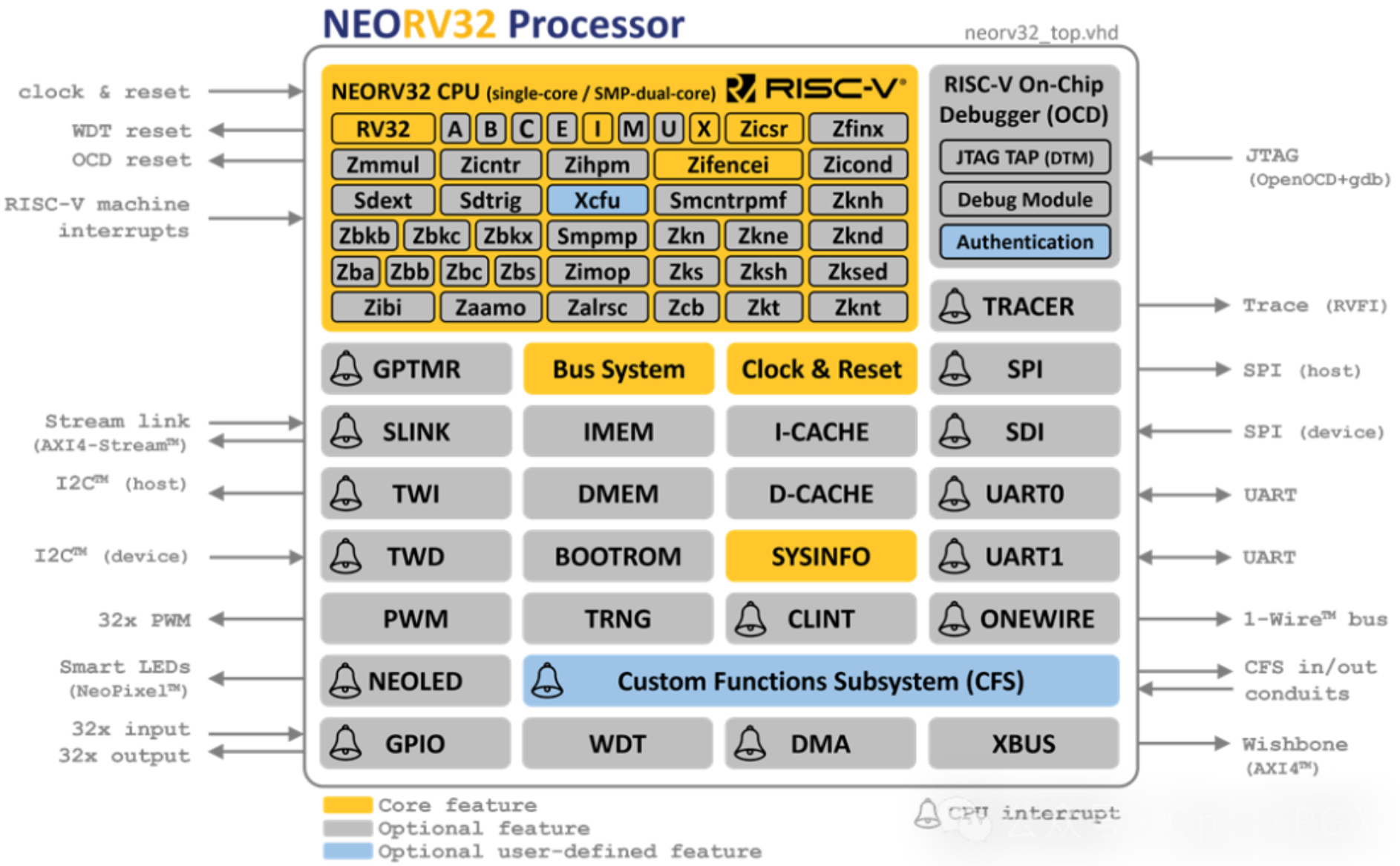
- cva6仅具有**L1Cache**，但结合开源的OpenPiton，可以拓展L2Cache

PULP项目：多核SOC

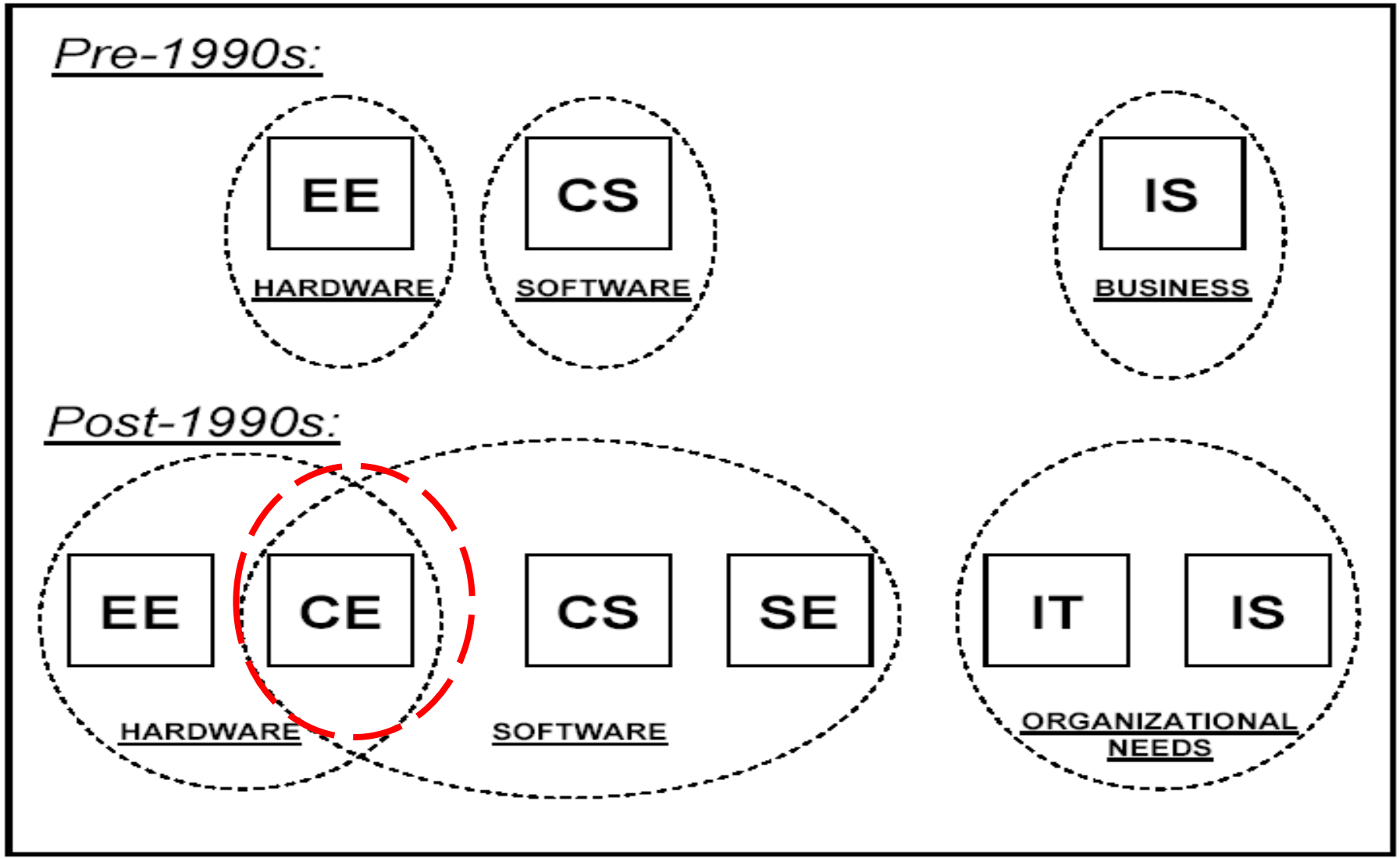




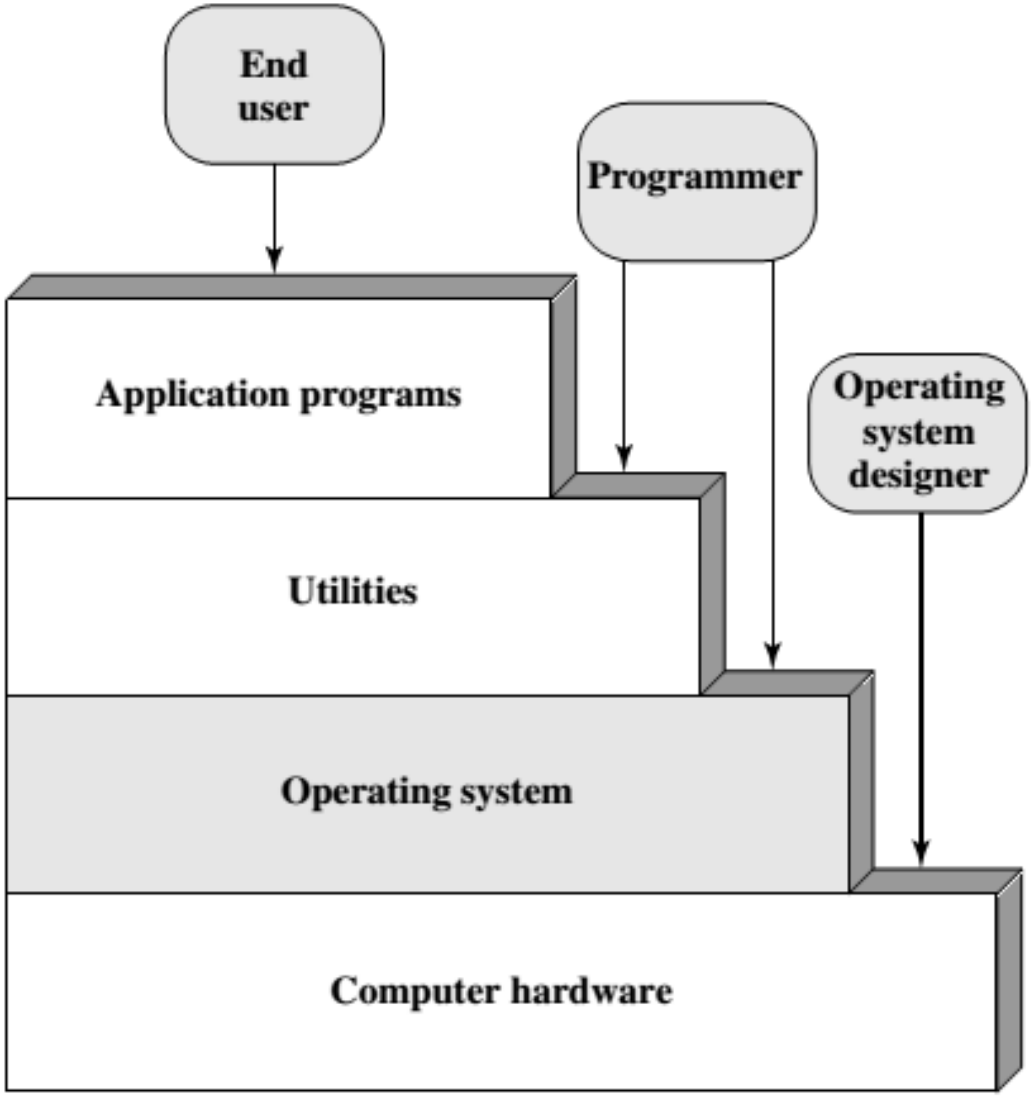
NEORV32：教科书级类微控制器SoC



计算科学（Computing）

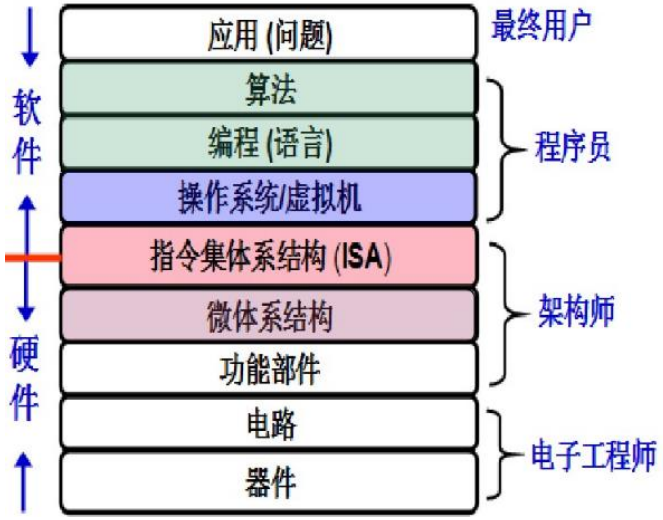


Layers and Views of a Computer System



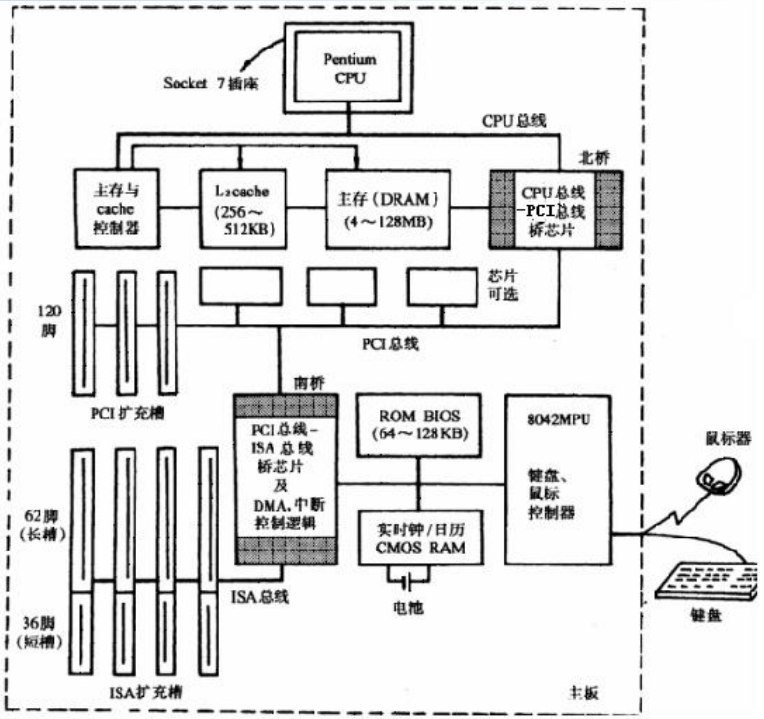
“计算机在干啥”？

计算机系统抽象层的转换

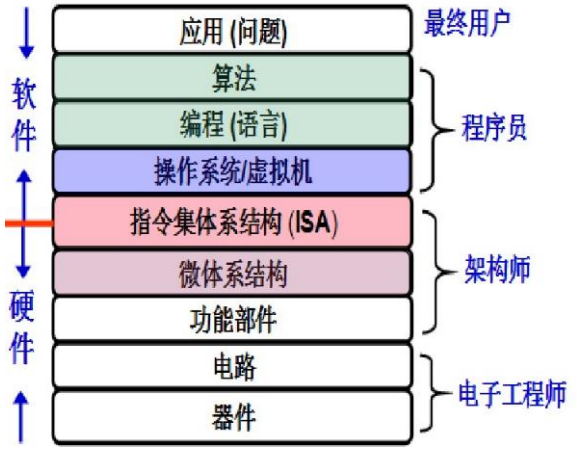


程序是如何执行的?

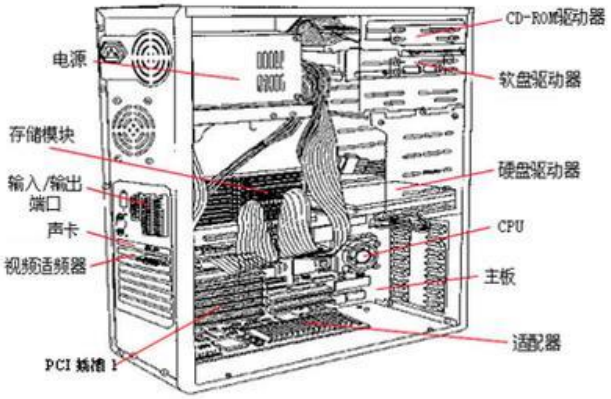
```
#include <stdio.h>
int main(void)
{    int ch;

     printf("Input a character:");

     /* read a character from
     the standard input stream */
     ch = getchar();
      putchar(ch);

     return 0;
}
```



计算机系统抽象层的转换

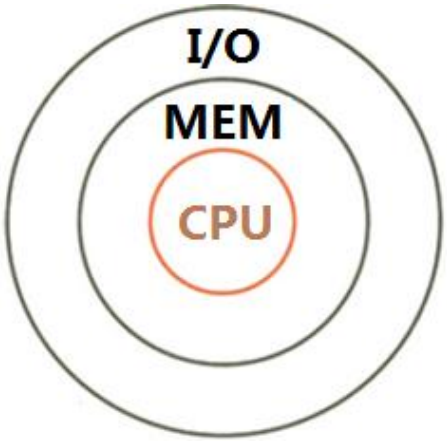


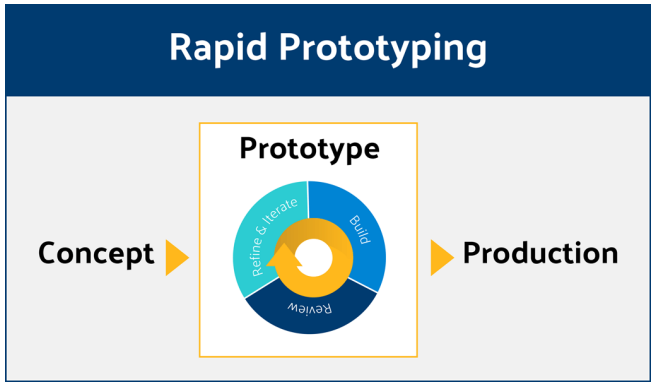
最“小”的计算机?

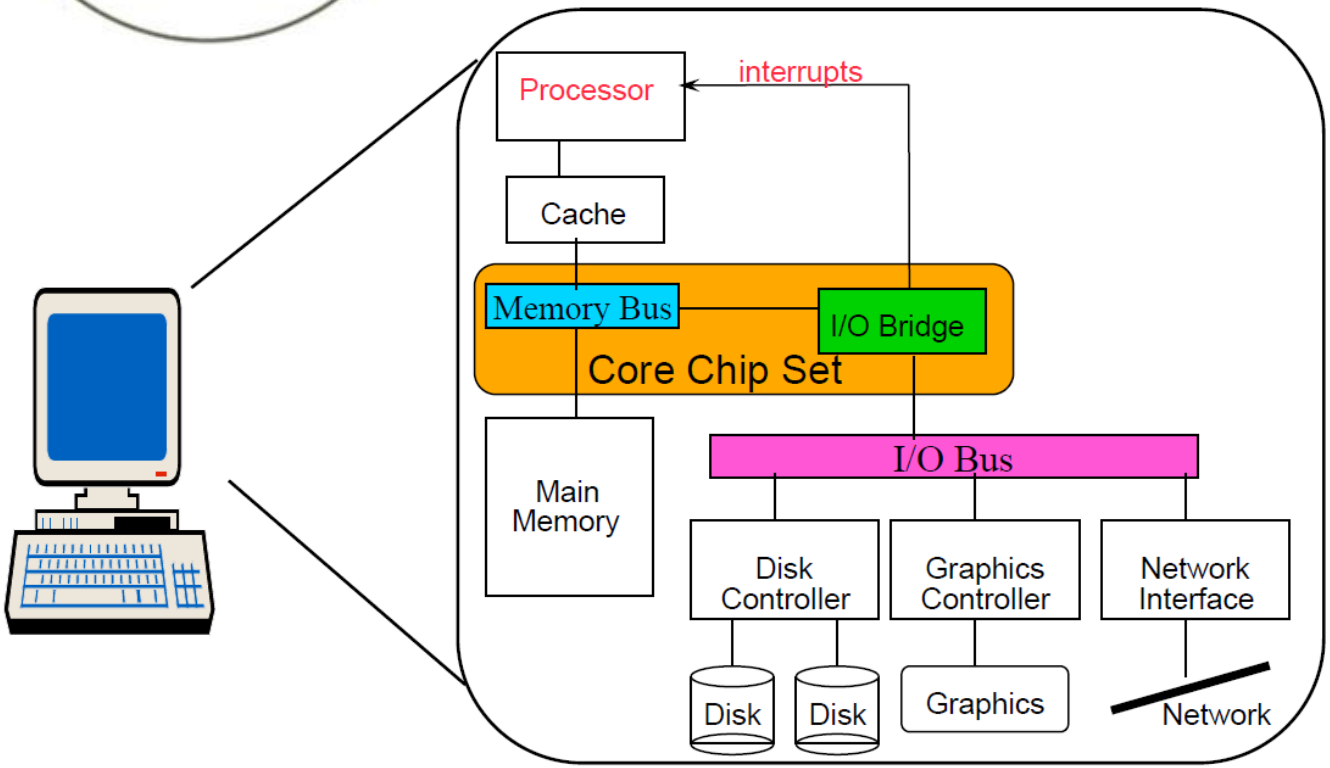

CD-ROM驱动器
电源
软盘驱动器
存储模块
硬盘驱动器
输入/输出端口
CPU
声卡
视频适频器
主板
适配器
PCI 插槽

CPU
寄存器文件
PC
ALU
总线接口
系统总线
存储器总线
I/O 桥
主存储器
“hello, world\n”
hello 代码
I/O 总线
扩展槽，留待网络适配器一类的设备使用
USB 控制器
图形适配器
磁盘控制器
鼠标
键盘
显示器
“hello, world\n”
磁盘
存储在磁盘上的hello可执行文件

ABC范式： Arch、Behavior、Constraints

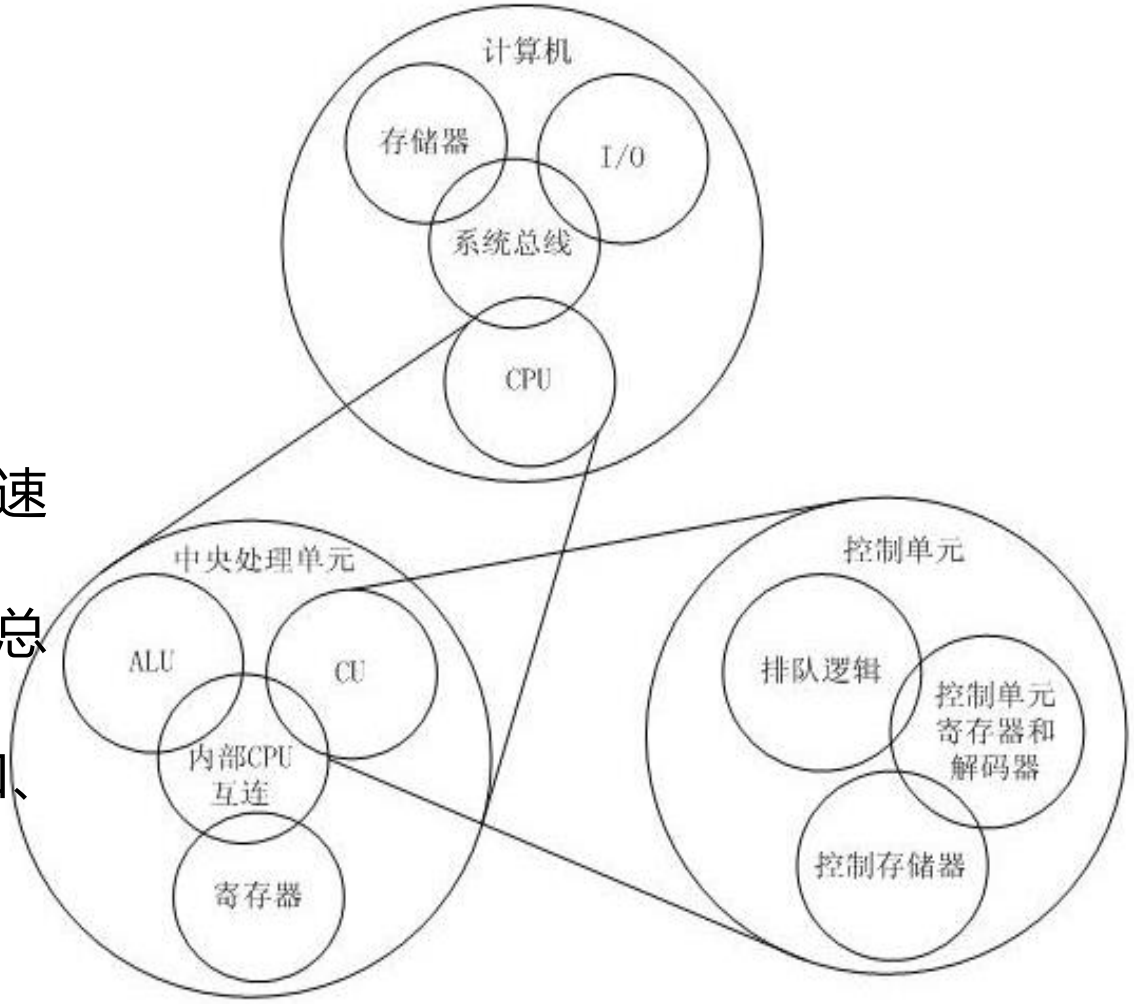






课程内容

- 中央处理器（CPU）
  - *计算机的运算方法*
  - 指令系统
  - CPU的结构
  - 控制单元设计
- 外围部件
  - **存储器**（主存储器、高速缓存、辅助存储器）
  - **系统总线**（总线性能、总线结构、总线控制）
  - **输入输出系统**（I/O接口、I/O控制方式、外设peripheral device）



计算机组成原理

指令系统ISA

程序员可见的计算机属性

指令字格式，寻址方式，指令类型

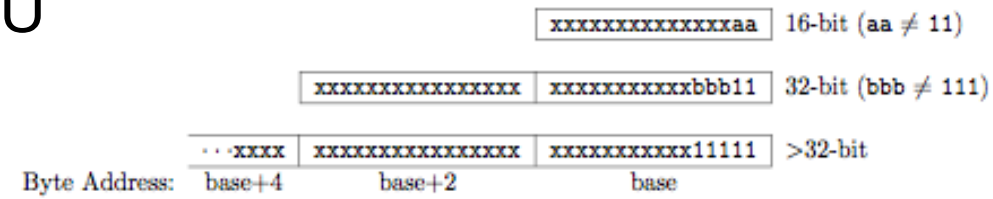
CISC，RISC，VLIW

RISC-V指令格式与操作码

图2-19，图4-16

Name (Field size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

- 指令格式：6种，基本R/I/S/U
  - 规整：Reg和Imm位置固定
  - op码与指令类型和格式绑定

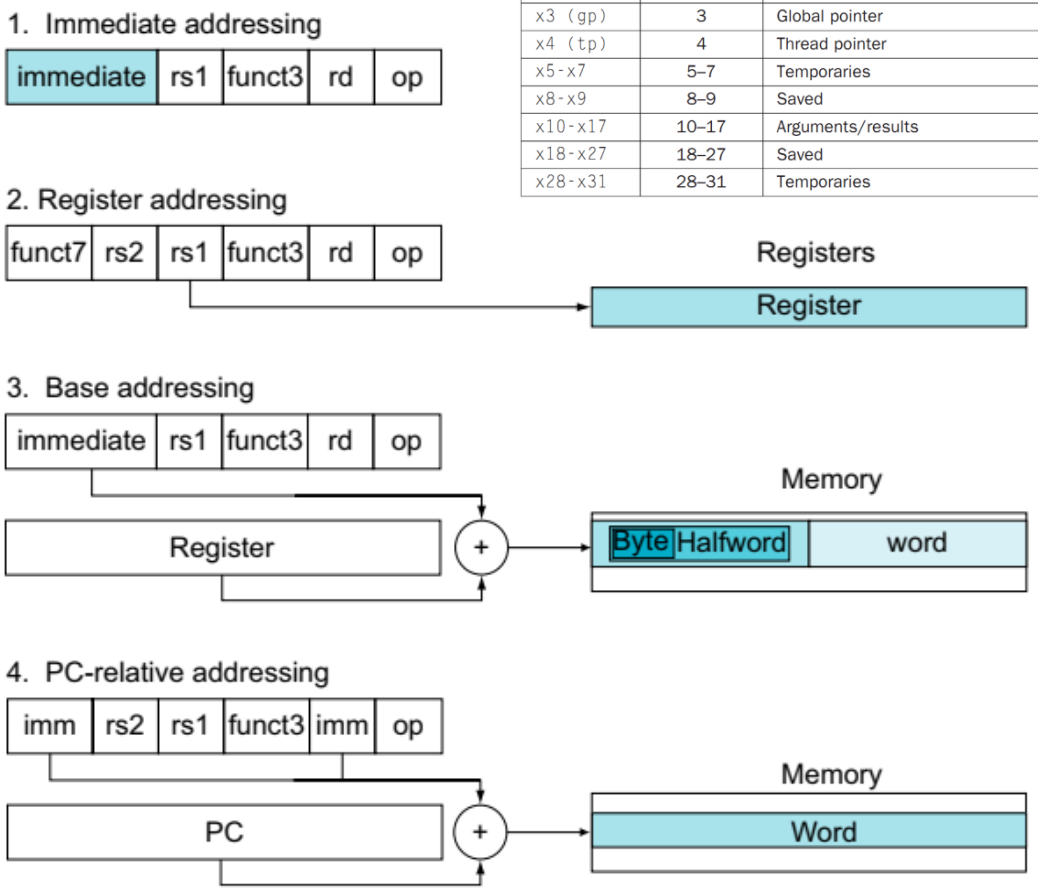


- B/J-type的立即数域（循环移位）？
  - “减少数据通路上的MUX数量和MUX端口数，改善时钟周期”，$4.4.2
- 变长指令字：Opcode的bbbaa
  - Can support variable-length instructions（RV16/RV32/RV64）

RISC-V寻址方式，图2-17

- 4种：本质3种，Imm，Reg，Base
- opr
  - 立即寻址
  - 寄存器寻址
  - 基址寻址
- 指令
  - PC相对寻址
    - beq，jal
  - 间接跳转
    - jalr x0，100(x1)
- 堆栈寻址?
  - 过程调用，syscall，中断

Name	Register number	Usage	Preserved on call?
x0	0	The constant value 0	n.a.
x1 (ra)	1	Return address (link register)	yes
x2 (sp)	2	Stack pointer	yes
x3 (gp)	3	Global pointer	yes
x4 (tp)	4	Thread pointer	yes
x5-x7	5–7	Temporaries	no
x8-x9	8–9	Saved	yes
x10-x17	10–17	Arguments/results	no
x18-x27	18–27	Saved	yes
x28-x31	28–31	Temporaries	no



RV32I 机器指令（常规）

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

RV典型内存地址空间分配

- 段式
  - DATA
  - Stack/Heap
    - 栈：自高向低
    - 堆：自低向高
  - CODE/Text
- 寻址
  - 绝对地址
  - 相对地址：基址+偏移
    - offset，displacement
    - 基址寄存器bp
- I/O port?
- 硬盘
- 网络

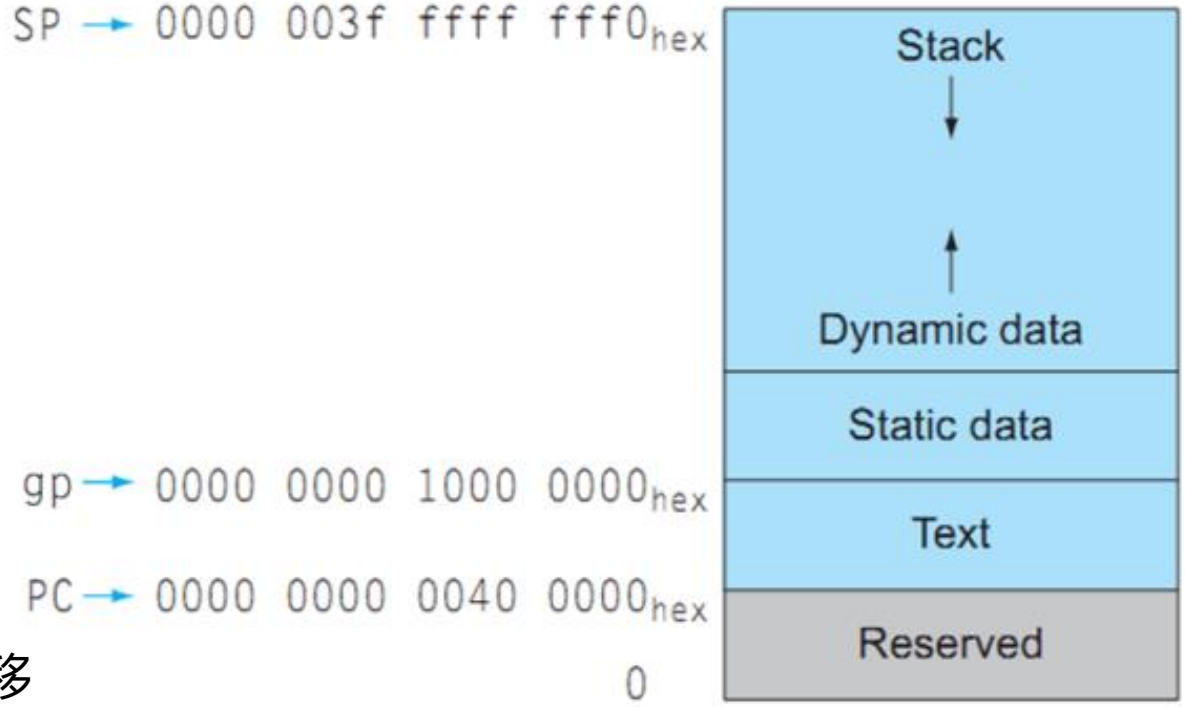


图2-13 用户地址空间划分

空间大小？

计算机组成原理

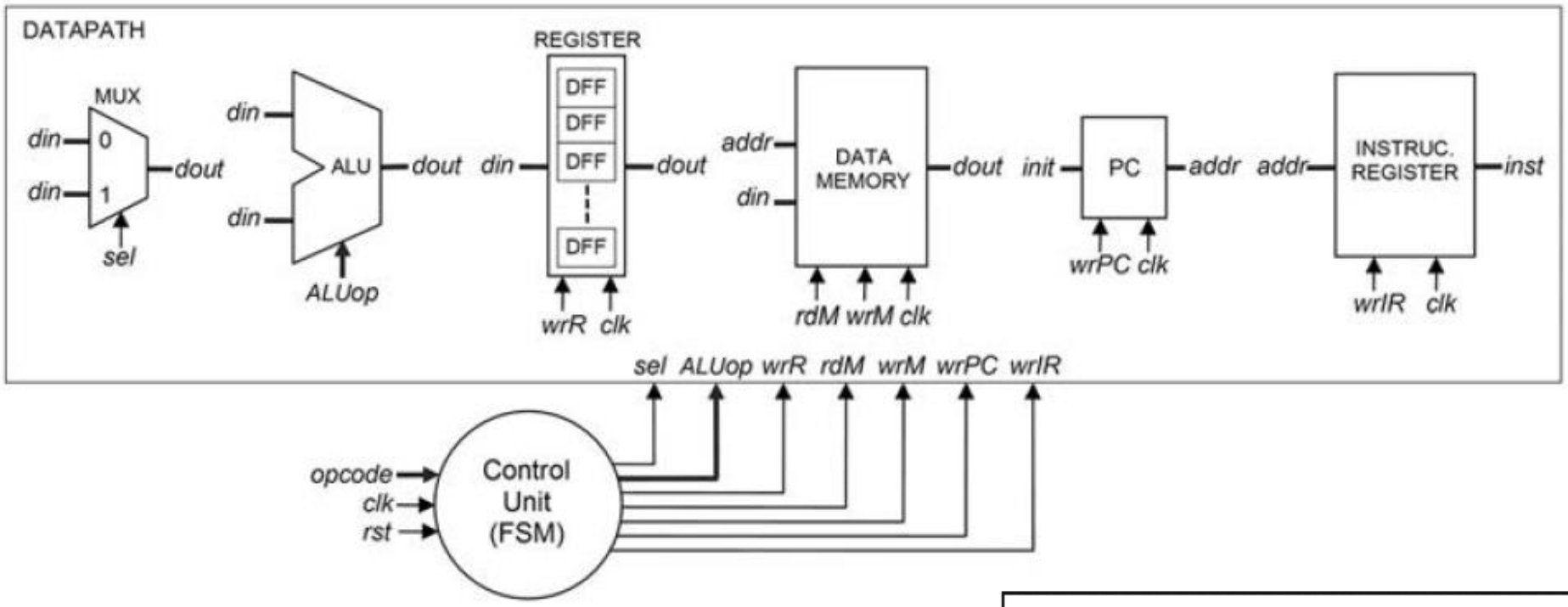
RISC CPU设计

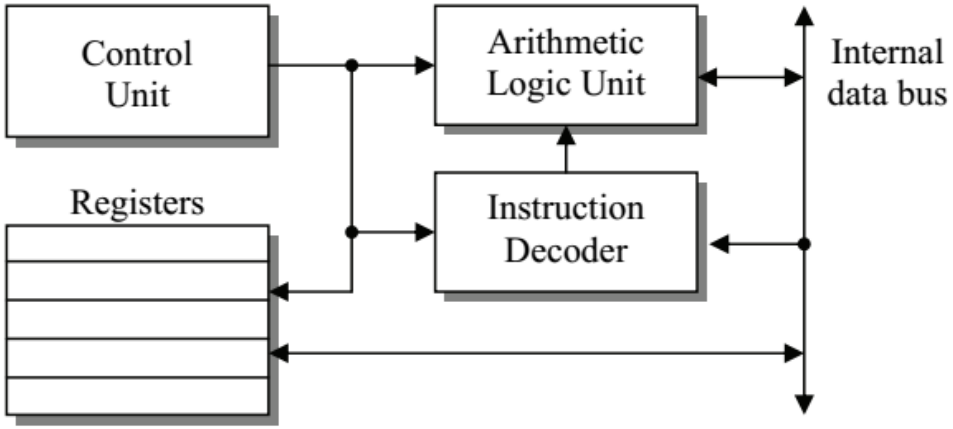
数据通路、控制器

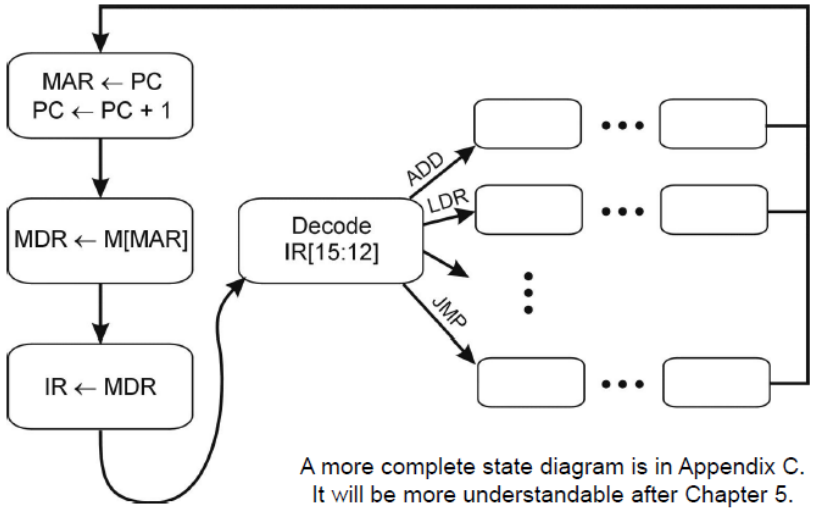
单周期、多周期、流水线

流水线hazard、stall、flush

DLX CPU设计：单周期、多周期、流水线

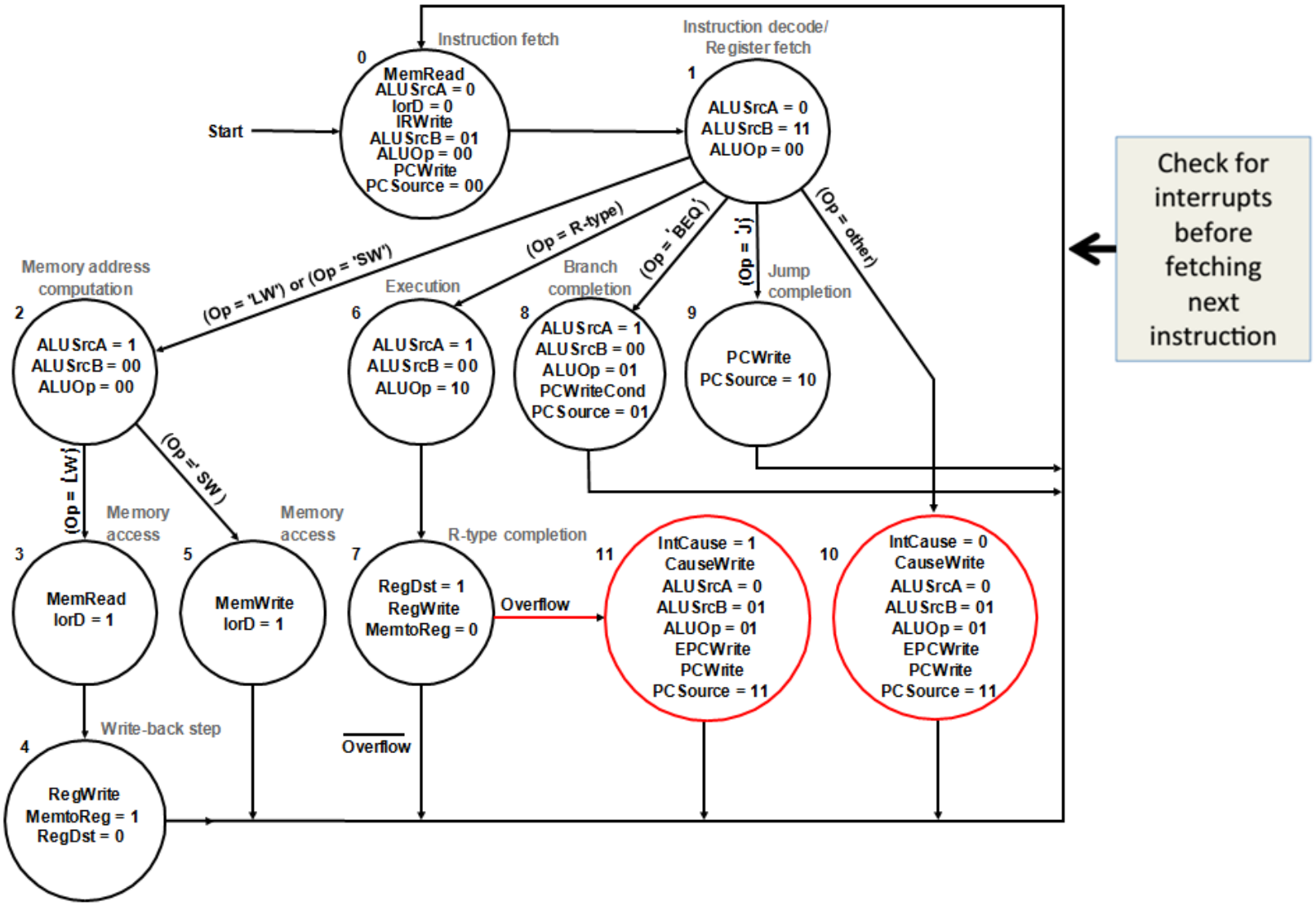




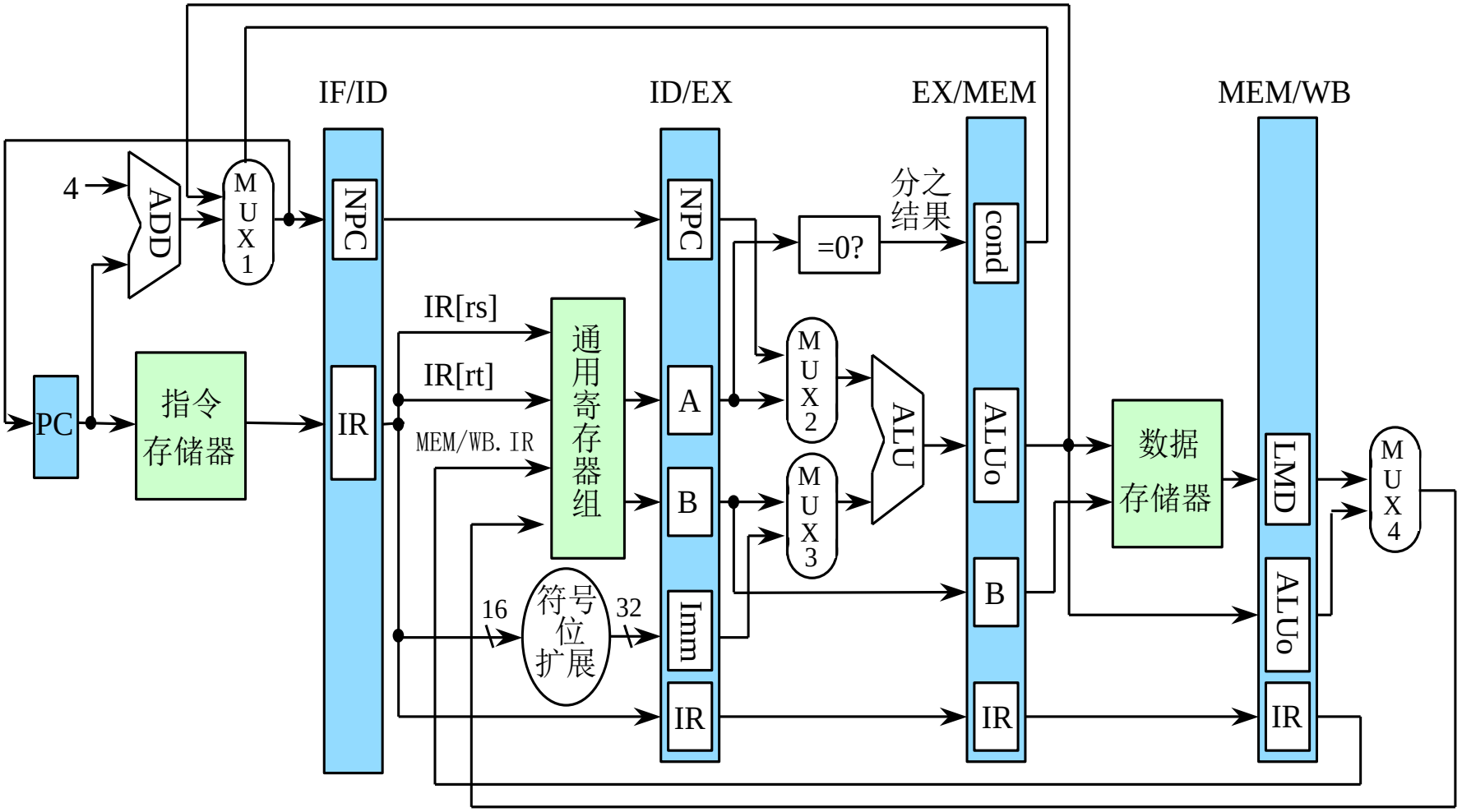


A more complete state diagram is in Appendix C.
It will be more understandable after Chapter 5.

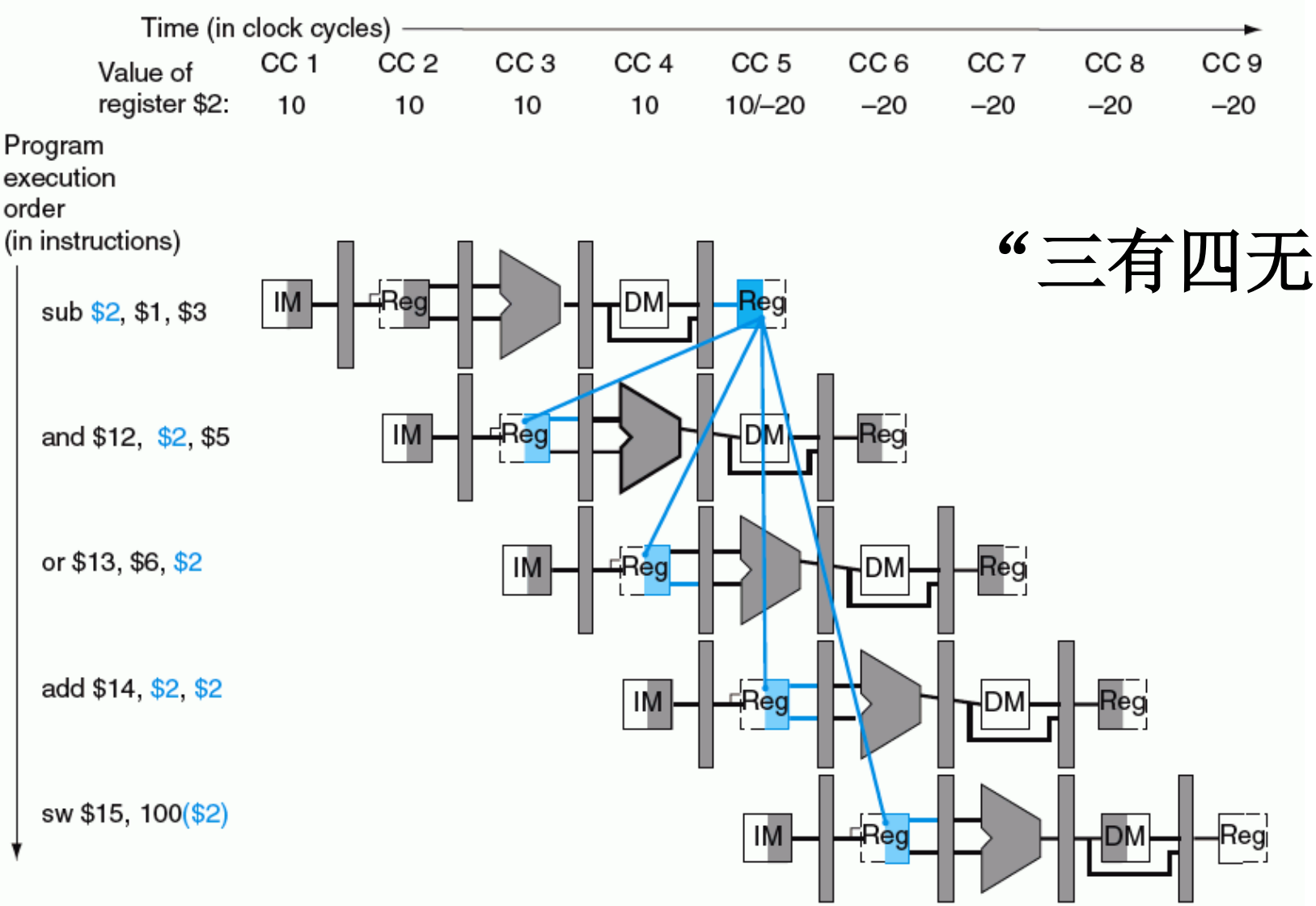
多周期实现的控制逻辑


Instruction fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
Instruction decode/
Register fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op = other)
Memory address
computation
2
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 00
Execution
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
Branch
completion
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Jump
completion
9
PCWrite
PCSource = 10
Check for
interrupts
before
fetching
next
instruction
(Op = 'LW')
(Op = 'SW')
Memory
access
3
MemRead
IorD = 1
Memory
access
5
MemWrite
IorD = 1
R-type completion
7
RegDst = 1
RegWrite
MemtoReg = 0
Overflow
11
IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
10
IntCause = 0
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
Write-back step
4
RegWrite
MemtoReg = 1
RegDst = 0
Overflow

DLX指令流水线

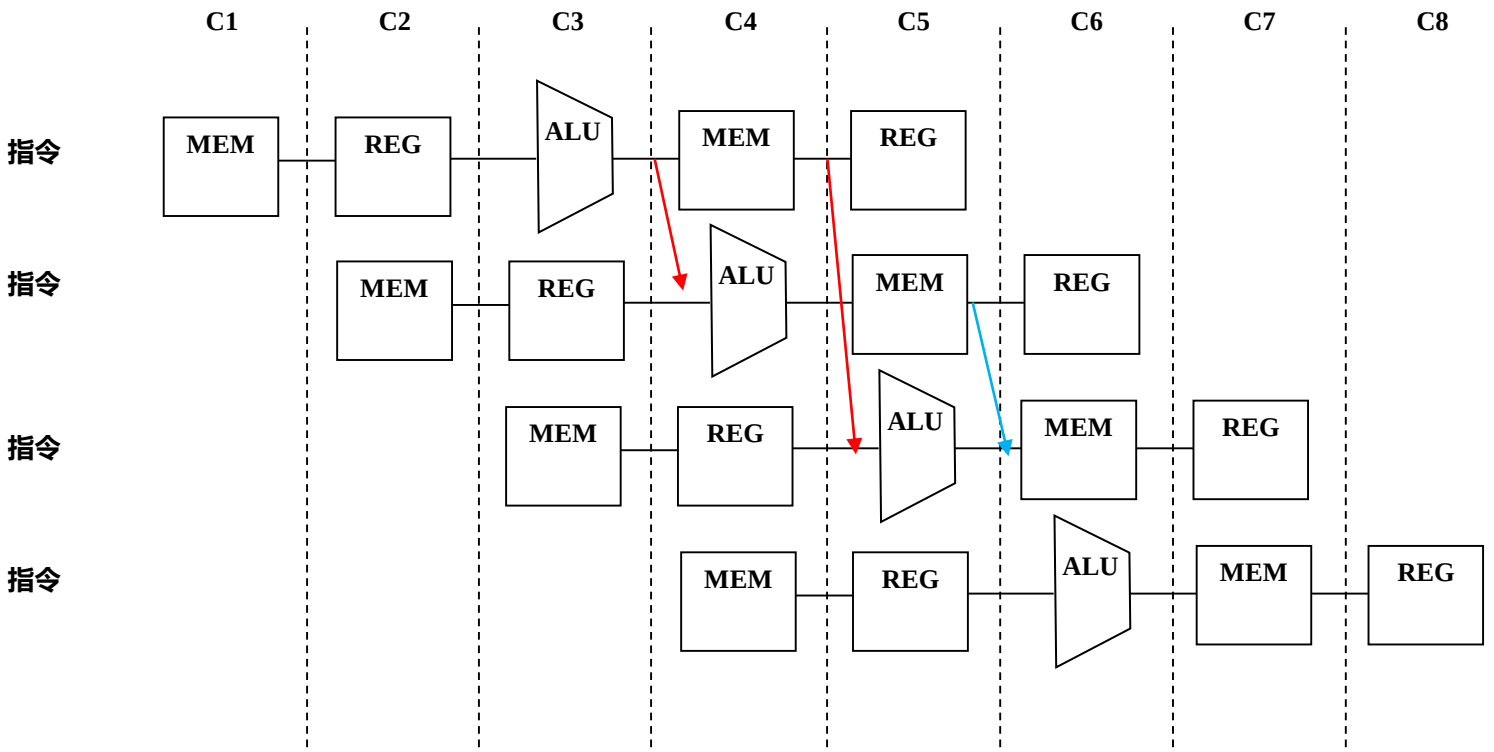

IF/ID
ID/EX
EX/MEM
MEM/WB
4
ADD
MUX 1
NPC
IR
PC
指令存储器
IR[rs]
IR[rt]
MEM/WB. IR
通用寄存器组
16
符号位扩展
32
NPC
A
B
Imm
IR
=0?
分之结果
MUX 2
MUX 3
ALU
cond
ALUo
B
IR
数据存储器
LMD
ALUo
IR
MUX 4

data hazard(RAW)



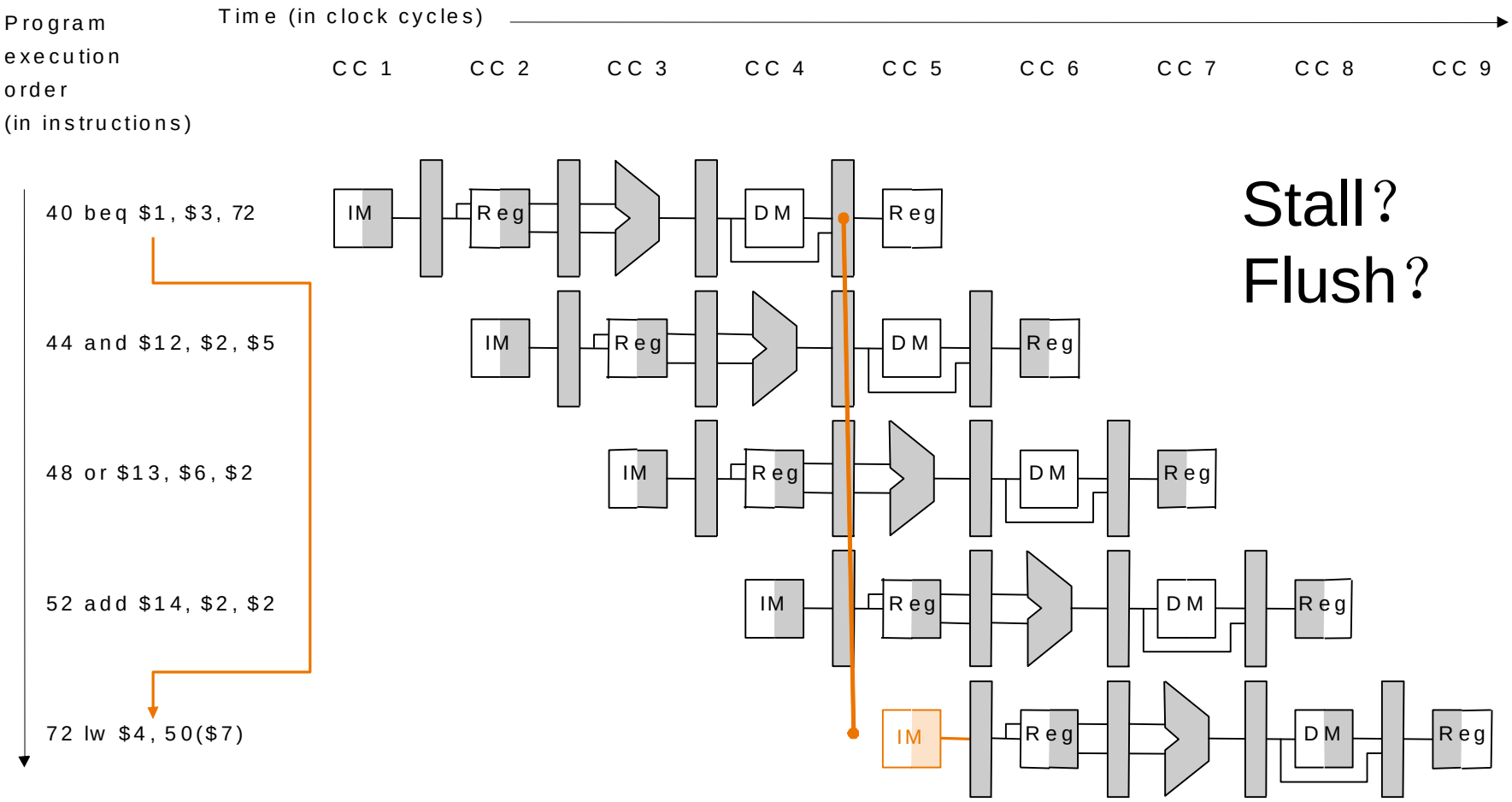
“三有四无”

消除RAW（EX段，MM段）



- 方法一：避免生成RAW序列（编译器静态调度，动态调度）
- 方法二：检测，前推，Stall（互锁interlock）
  - 三个特例：累加依赖、MEM-Copy（MM段）、Load-use（stall）

branch hazard（延迟槽、投机）



Beq：reducing the delay penalty & flush IF

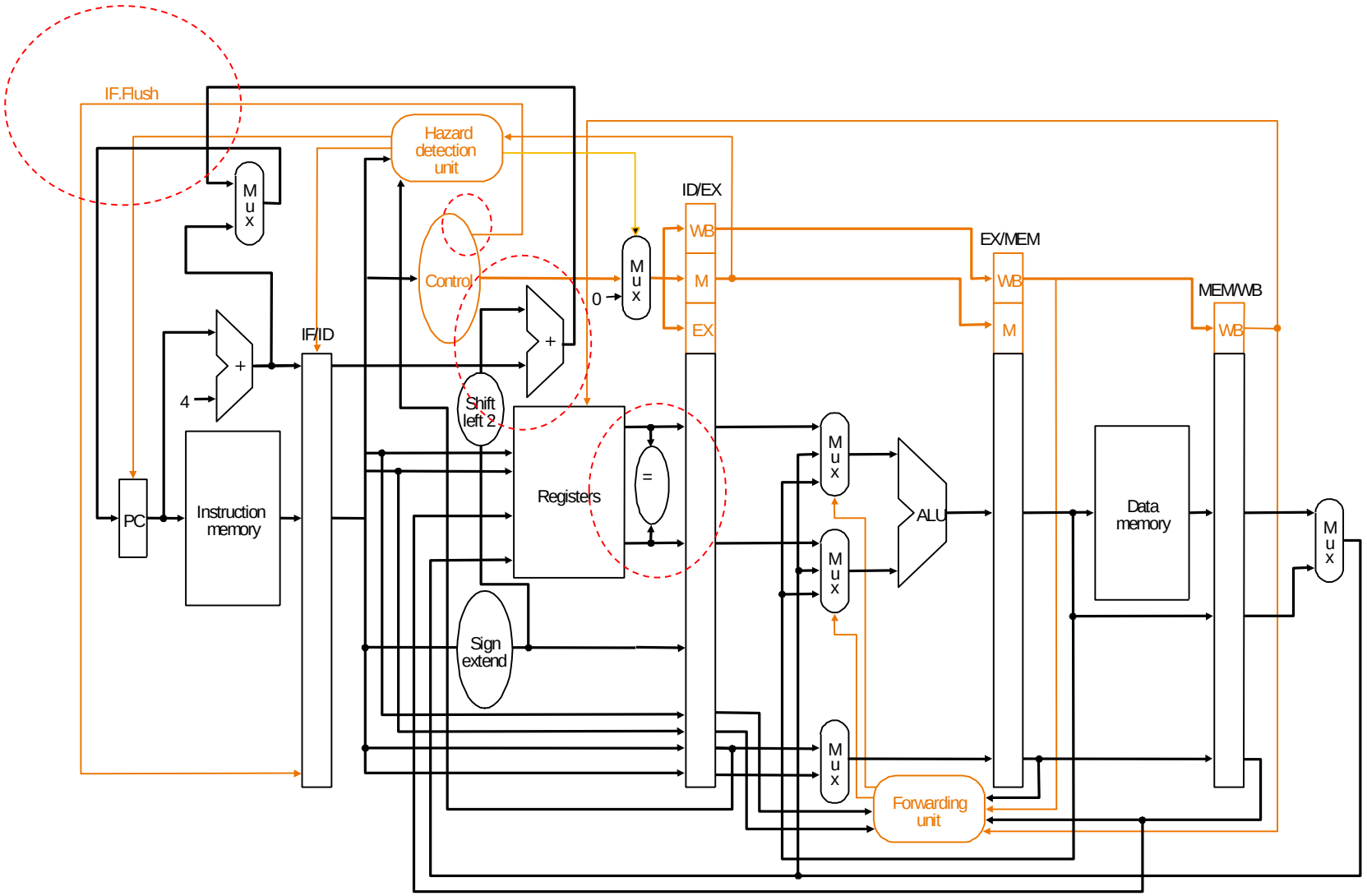
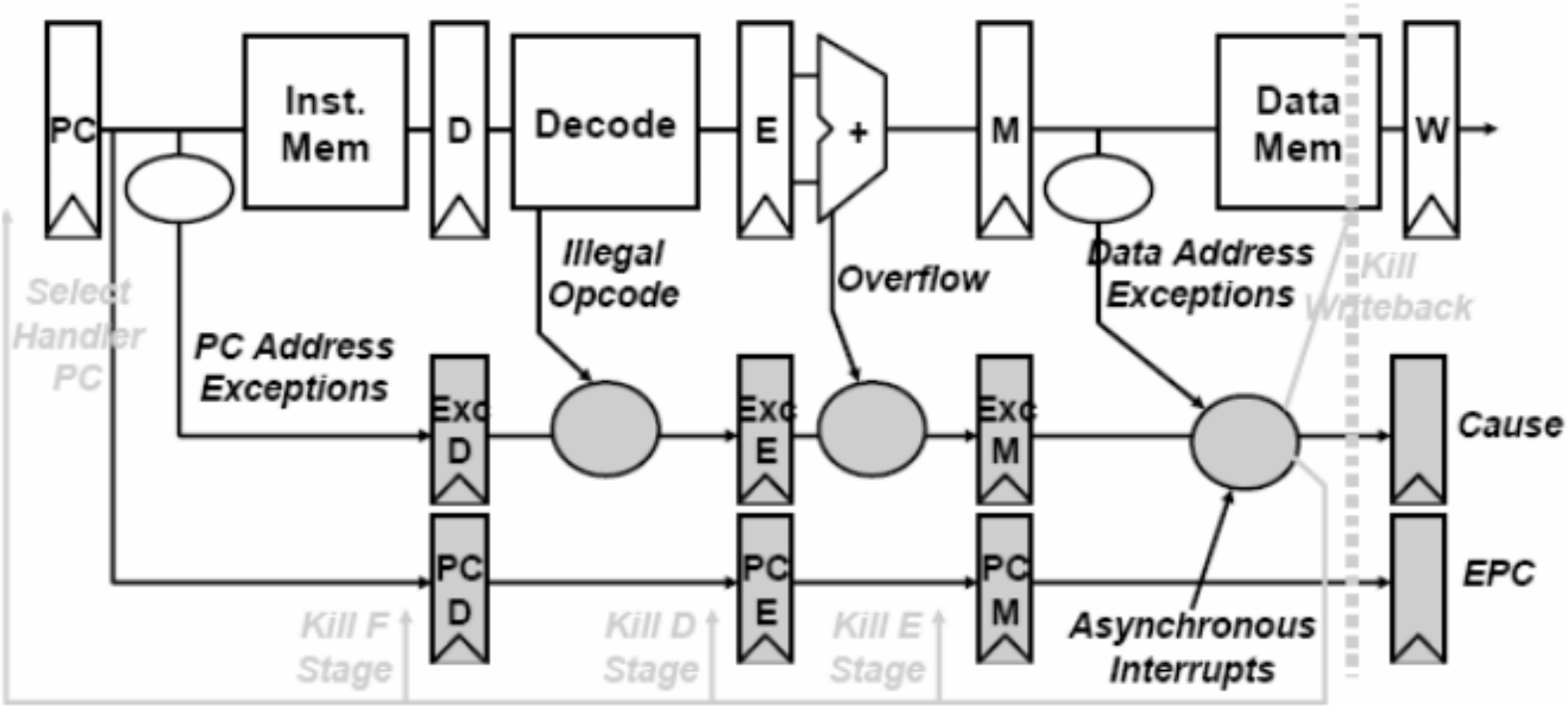


图4-65

各段产生的异常及中断处理：MIPS的策略

- 保持流水线的异常标记直到提交点（M段）
- 早期流水段的异常抑制后来的异常
- 提交点处引入外部异常（抑制其他异常）
- 如果提交点有异常或中断，则更新cause和EPC，清除所有流水段，优先级仲裁，将ISR入口赋予PC



计算机组成原理

存储系统

Cache，内存，辅存，虚存

组织，地址空间，与CPU通信

存储校验

PC机中的存储器

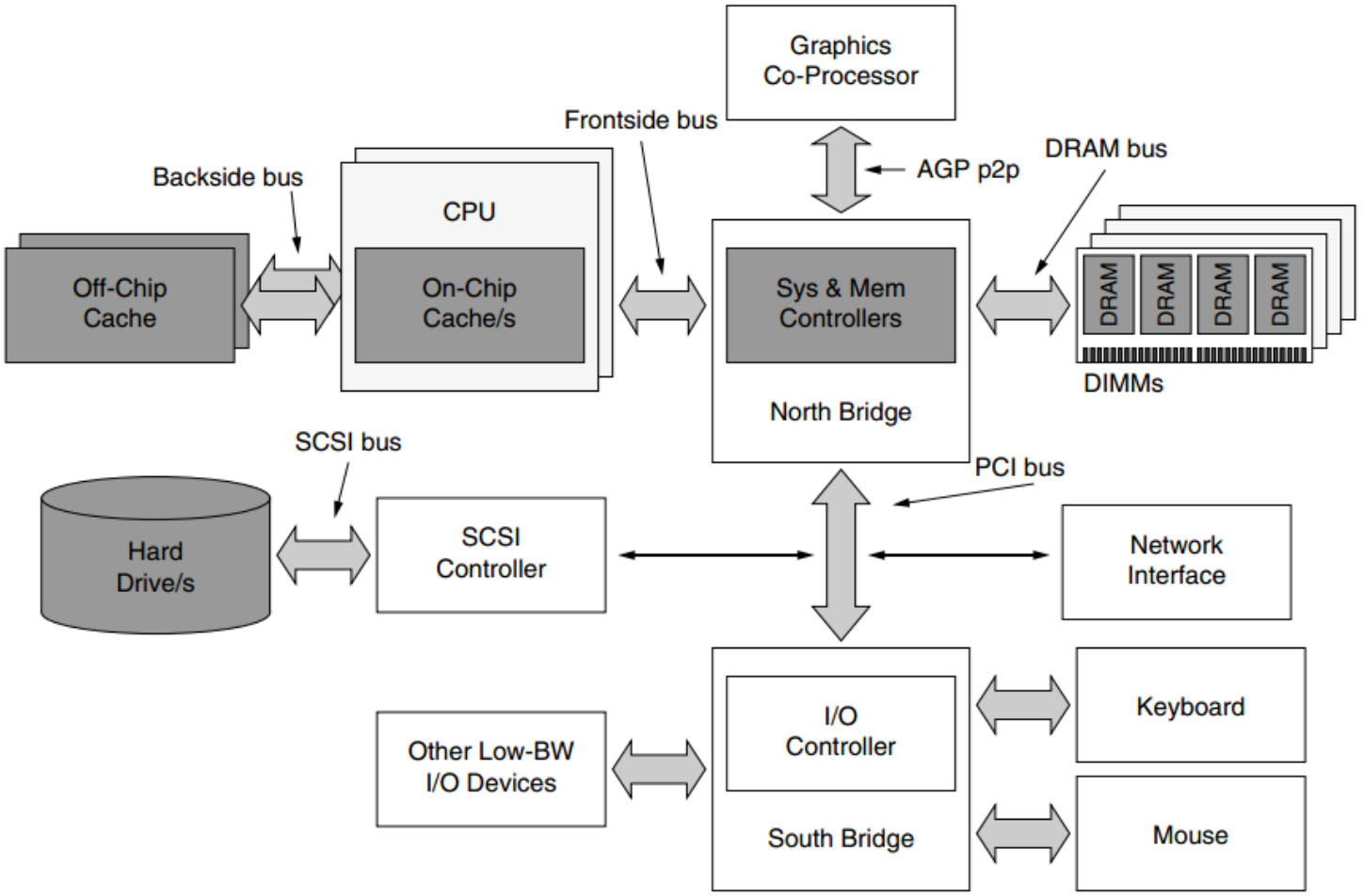
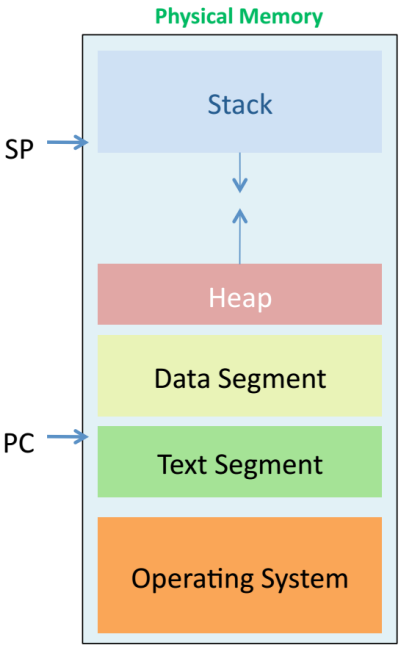
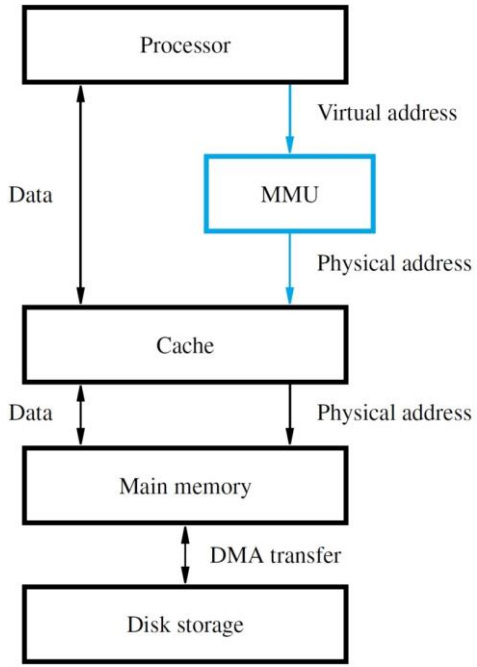
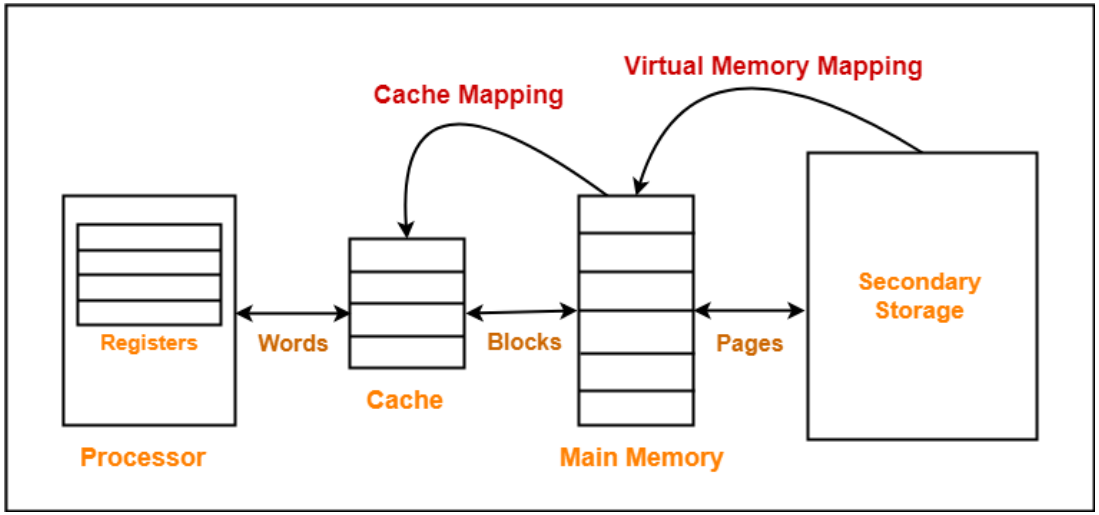


FIGURE 0v.3: Typical PC organization. The memory subsystem is one part of a relatively complex whole. This figure illustrates a two-way multiprocessor, with each processor having its own dedicated off-chip cache. The parts most relevant to this text are shaded in grey: the CPU and its cache system, the system and memory controllers, the DIMMs and their component DRAMs, and the hard drive/s.

存储系统：缓存、主存、虚存、外存

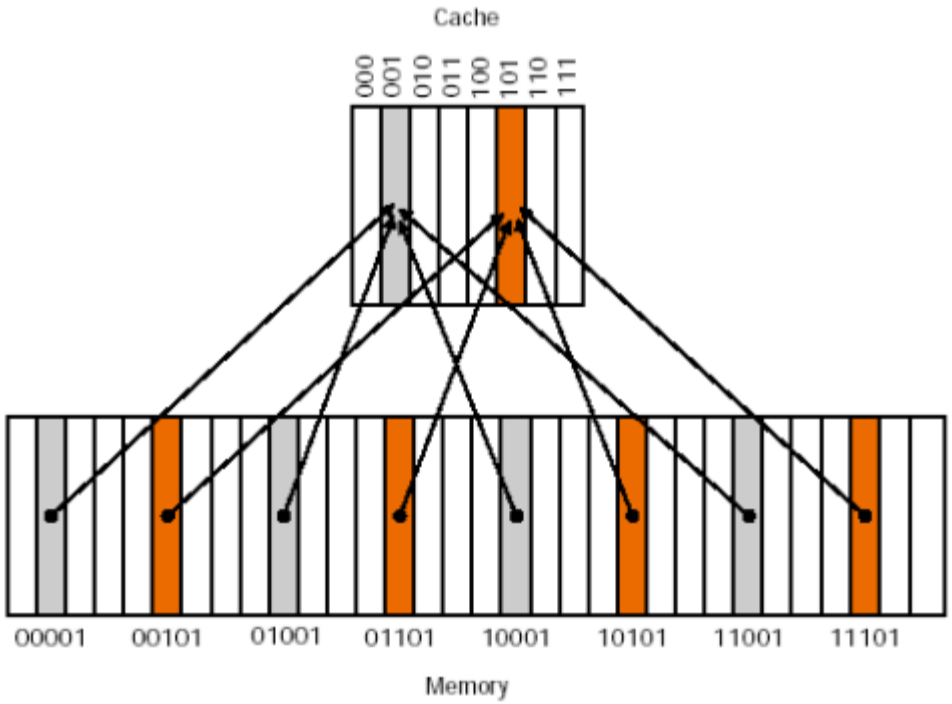


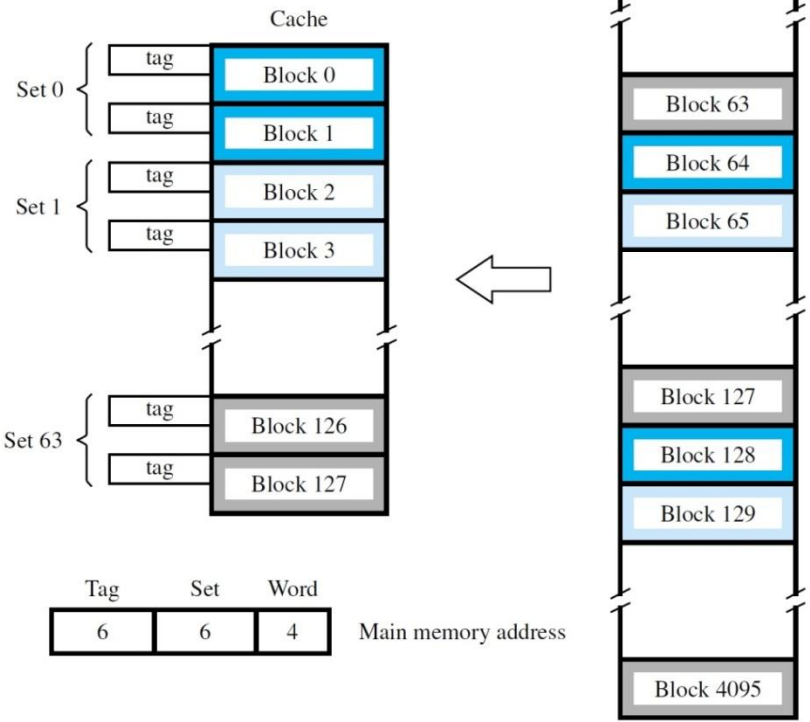




Cache: Set-Associative Mapping

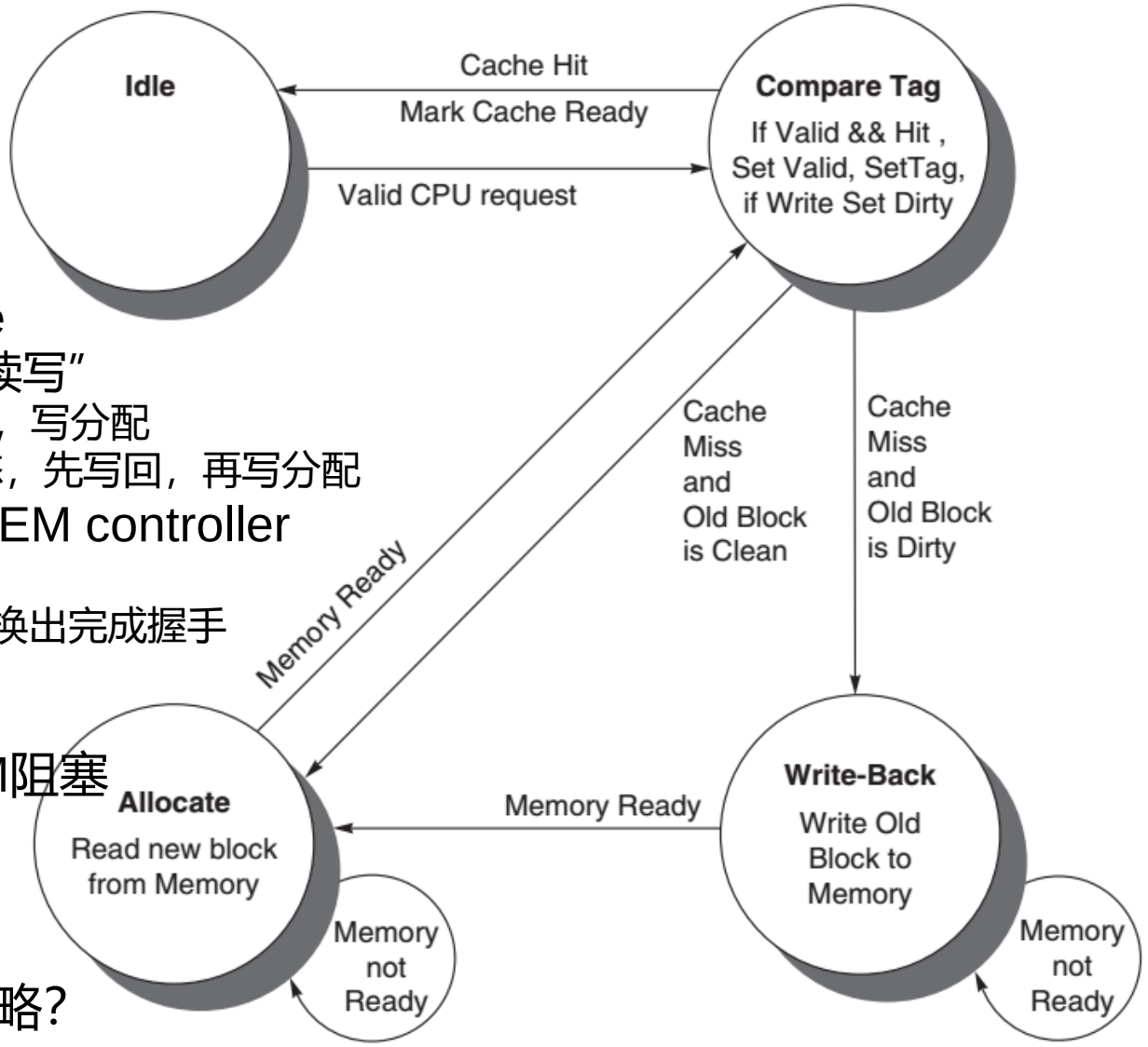
- 4Q
  - Q1: Where can a block be placed in cache? Block placement
    - Mapping: address is modulo the number of blocks
  - Q2: How is a block found if it is in cache? Block identification
  - Q3: Which block should be replaced on a miss? Block replacement
  - Q4: What happens on a write? Write strategy
- The "3C"s of cache misses types
  - Compulsory（Cold）、Conflict、Capacity、***Coherence***



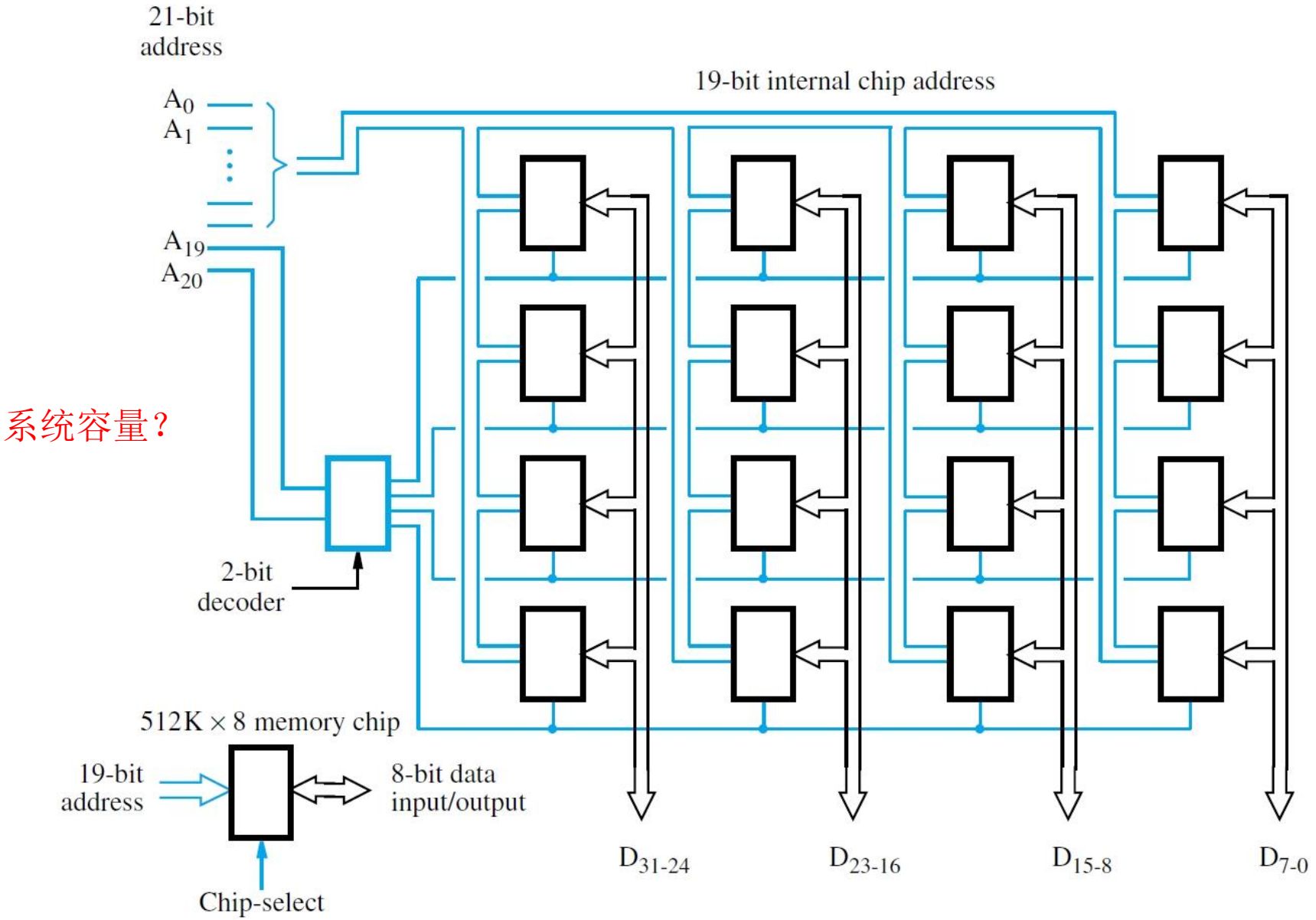


a Directmapped Cache Controller

- 多周期
- 读写操作策略
- CompTag
  - 命中：完成读写cache
  - Miss： “先换入，再读写”
    - 非脏：Allocate状态，写分配
    - 脏：Write-Back状态，先写回，再写分配
- Cache Controller<-> MEM controller
  - 半同步?
    - MEM Ready：换入换出完成握手
- 与CPU同步：IF或MEM阻塞
  - 周期数=?
- 流水化?
- 写透？组相联？替换策略?



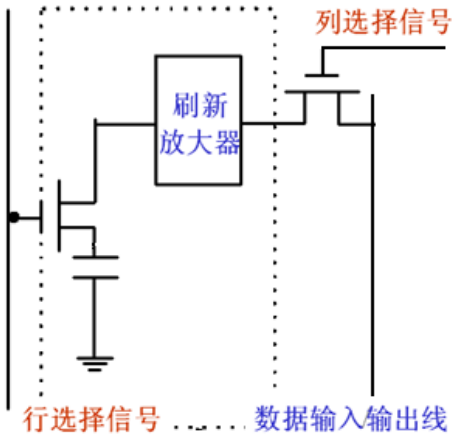
内存组成：cell类型，位/字扩展，交叉编址

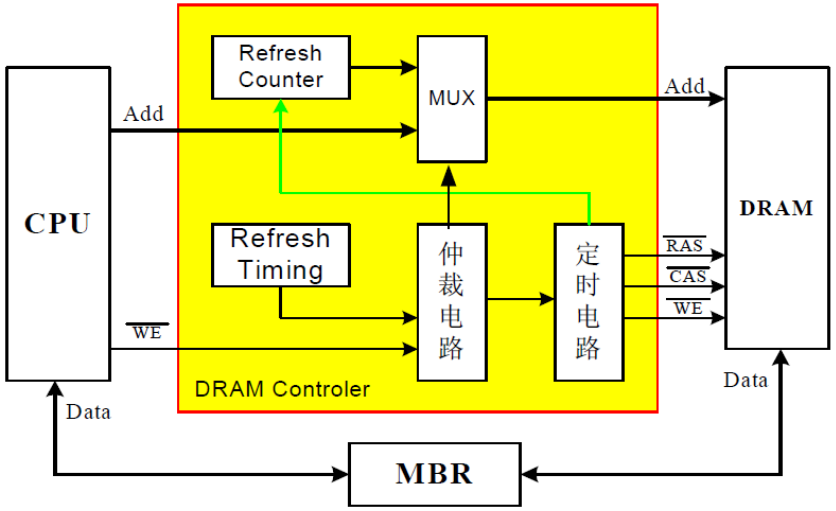


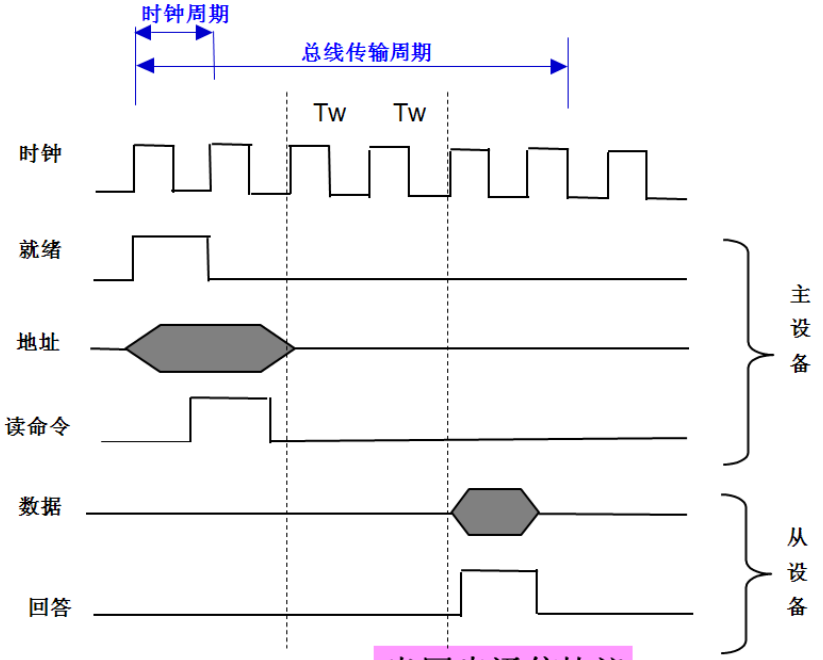
系统容量？

内存操作与访存过程

- DRAM操作：RAS/CAS，Read/Write
  - Precharge/Recharge/Refresh
- 同步方式
  - Async SRAM/DRAM
  - SSRAM/SDRAM
- 传输方式：单字模式，突发模式

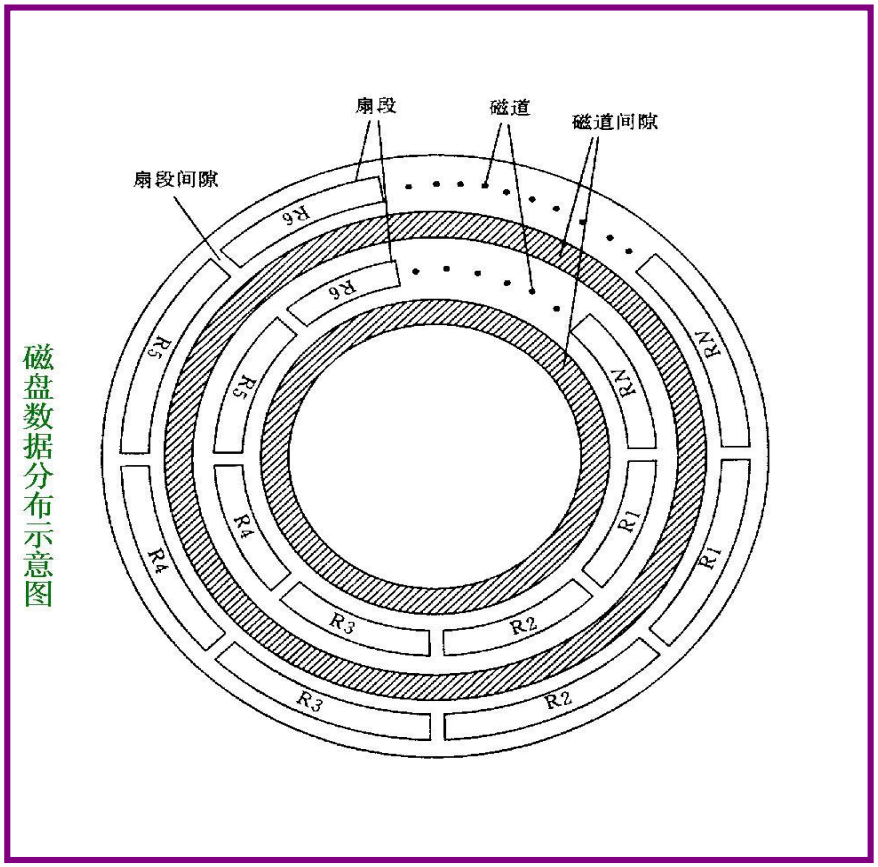


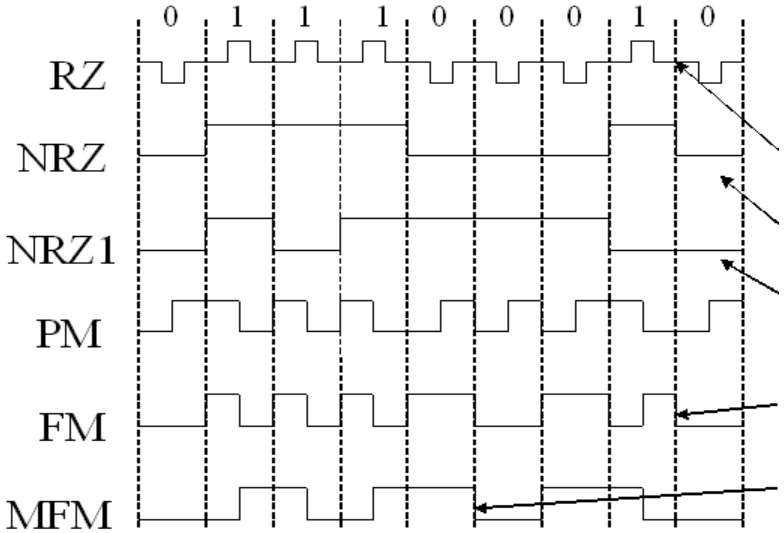




半同步通信协议

硬盘：编码方式，编址方式（物理/逻辑）

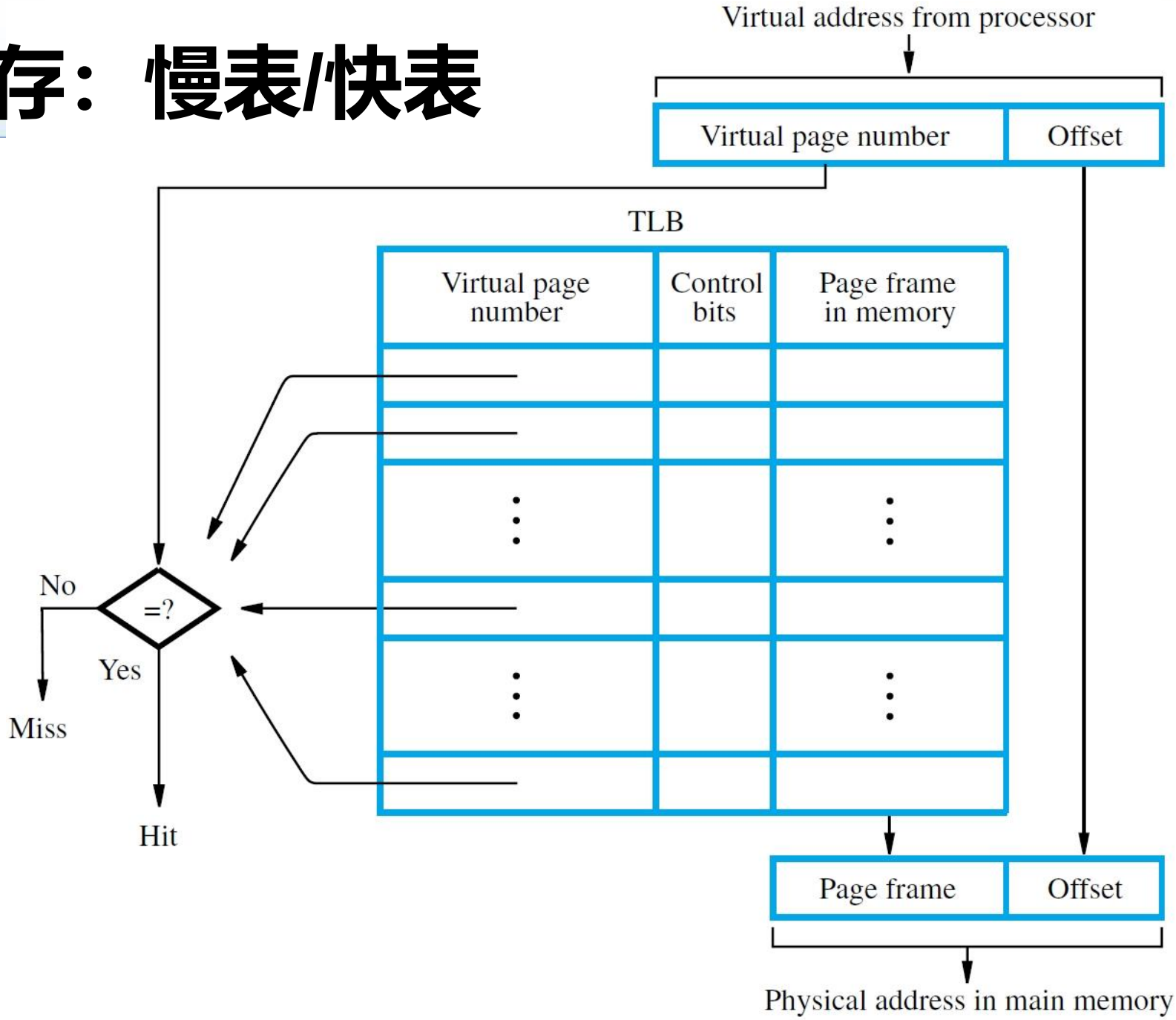




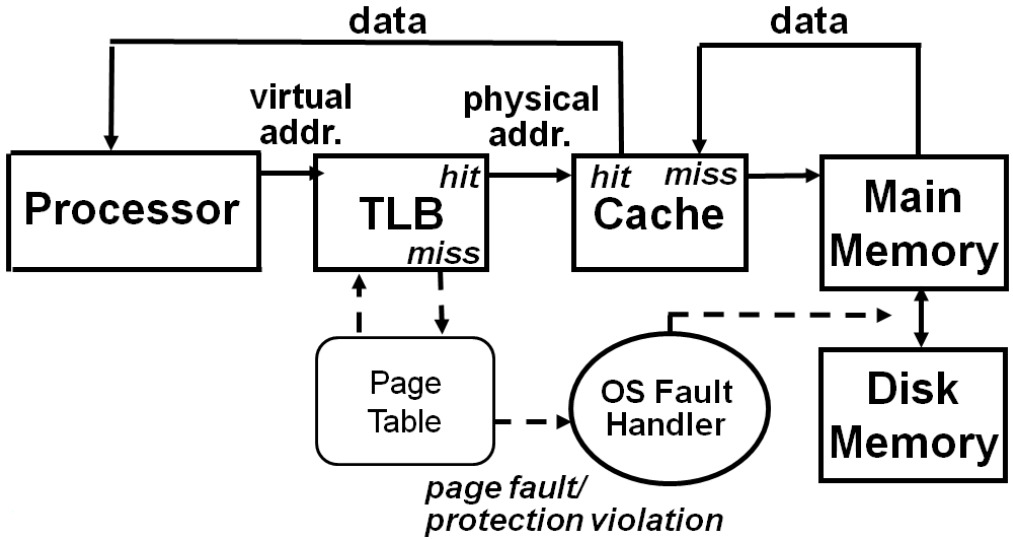
自同步能力？

台号	柱面号	磁头号	扇段号

虚存：慢表/快表


Virtual address from processor
Virtual page number
Offset
TLB
Virtual page number
Control bits
Page frame in memory
No
=?
Yes
Miss
Hit
Page frame
Offset
Physical address in main memory

虚存：MMU结构与过程



TLBmiss异常、缺页异常

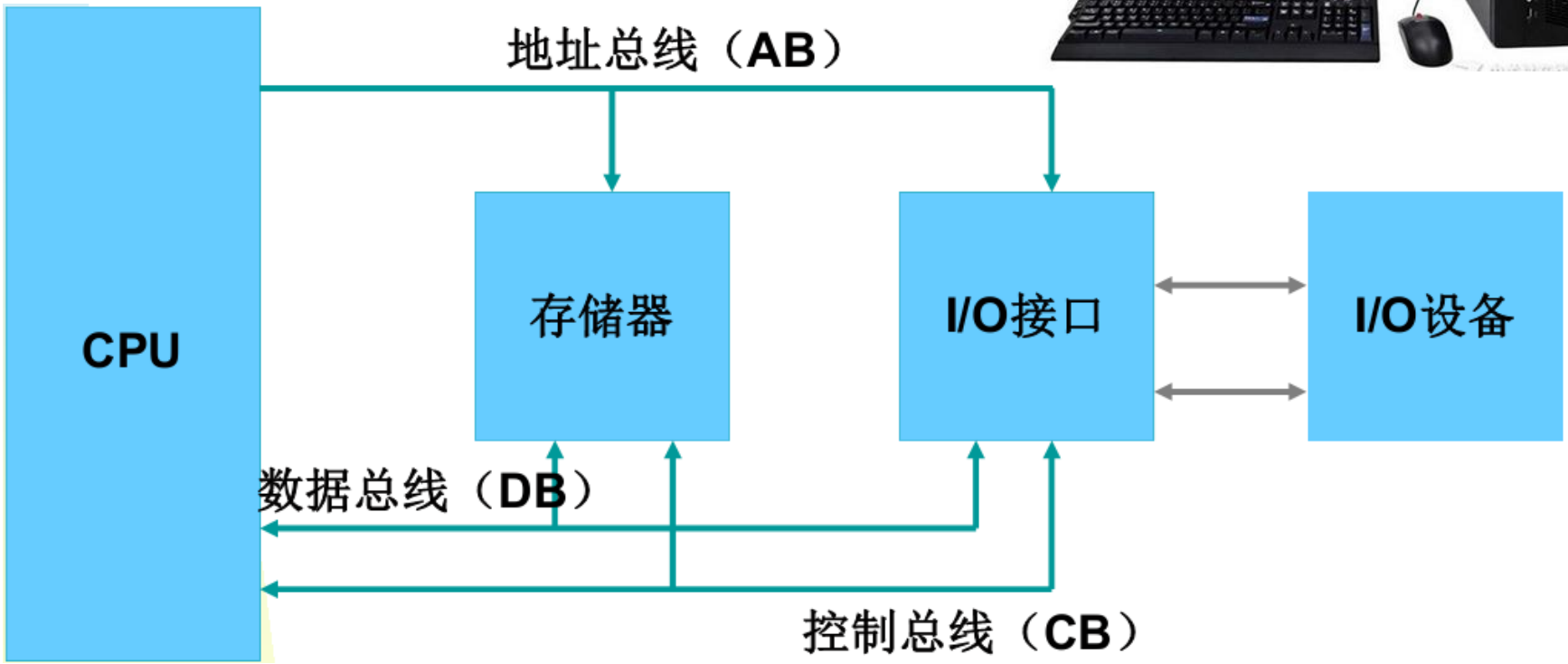
TLB	Page table	Cache	Possible? If so, under what circumstance?
hit	hit	miss	Possible, although the page table is never really checked if TLB hits.
miss	hit	hit	TLB misses, but entry found in page table; after retry, data is found in cache.
miss	hit	miss	TLB misses, but entry found in page table; after retry, data misses in cache.
miss	miss	miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
hit	miss	miss	Impossible: cannot have a translation in TLB if page is not present in memory.
hit	miss	hit	Impossible: cannot have a translation in TLB if page is not present in memory.
miss	miss	hit	Impossible: data cannot be allowed in cache if the page is not in memory.

计算机组成原理

总线

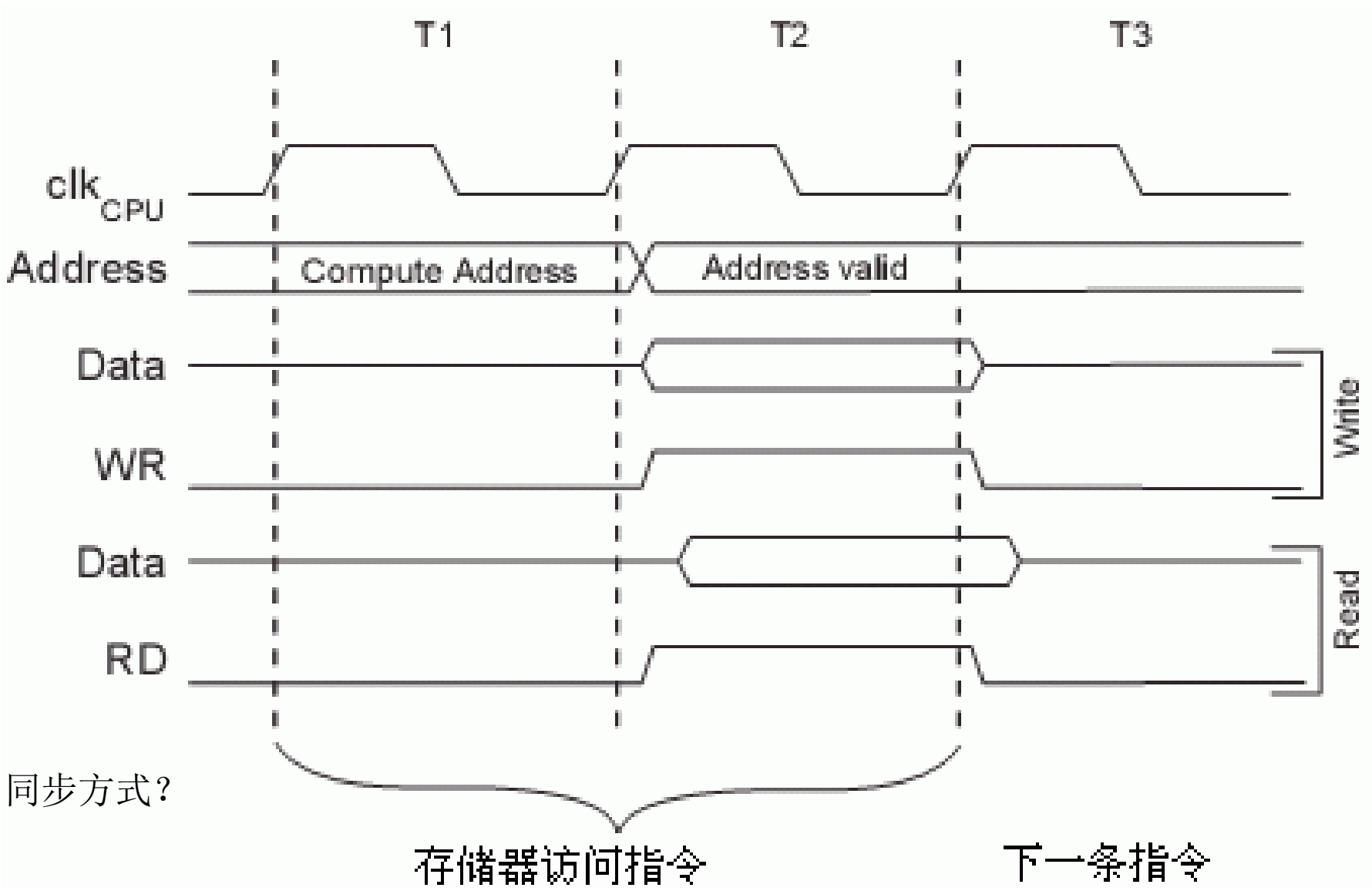
共享，竞争，同步

系统拓扑结构：基于总线



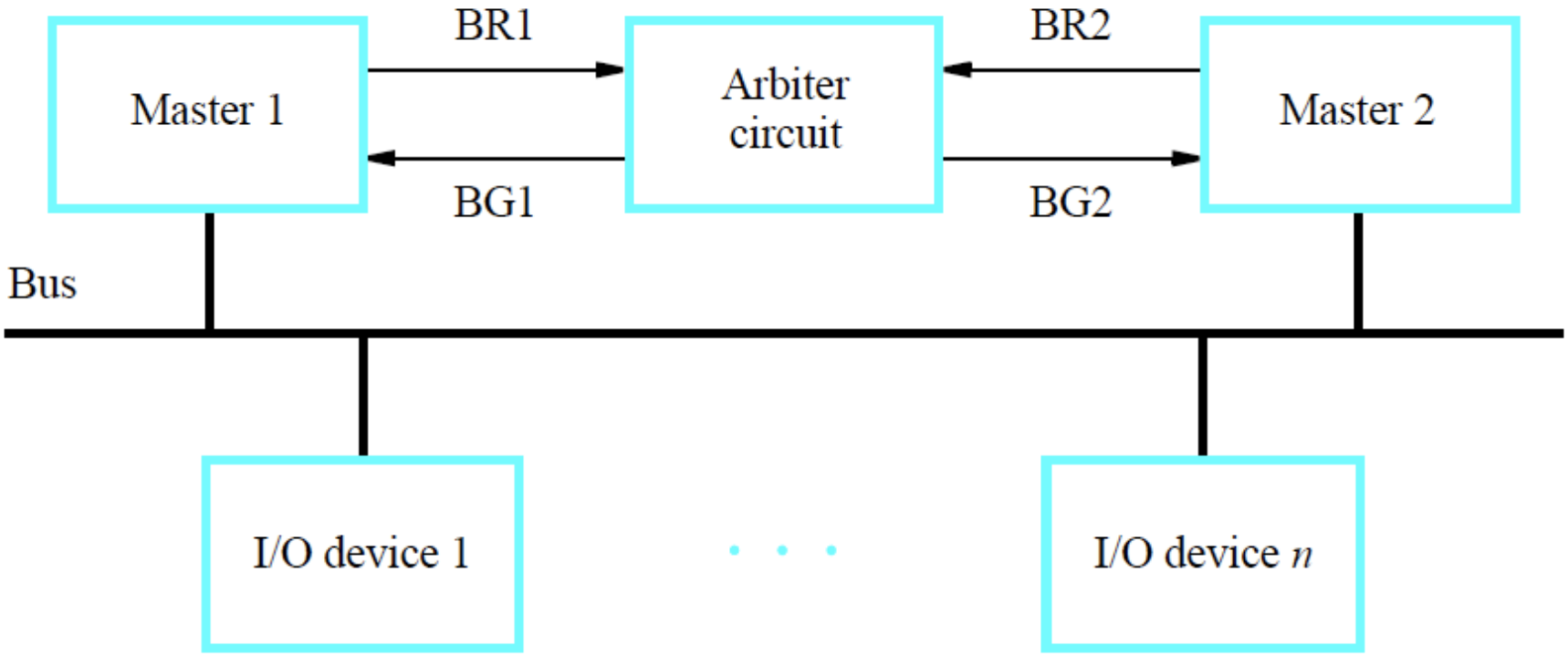
- 结构（单总线、多总线），传输过程（同步、仲裁），总线标准

总线通信过程：总线周期



同步方式？

总线使用权分配：仲裁方式?



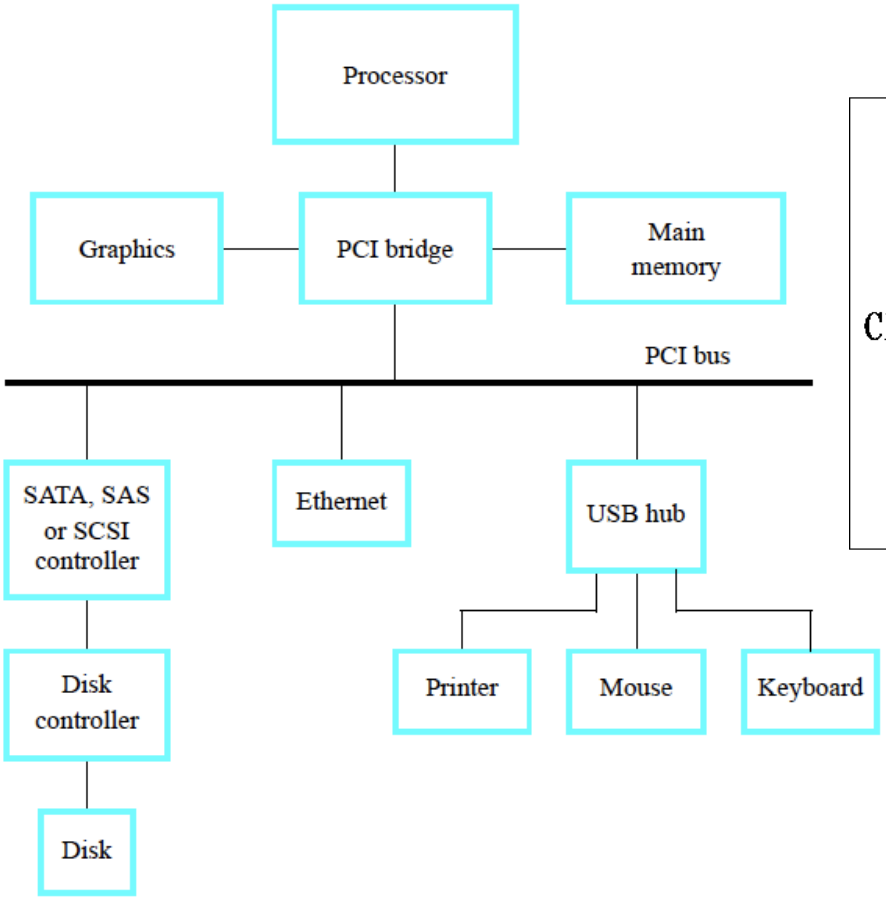
计算机组成原理

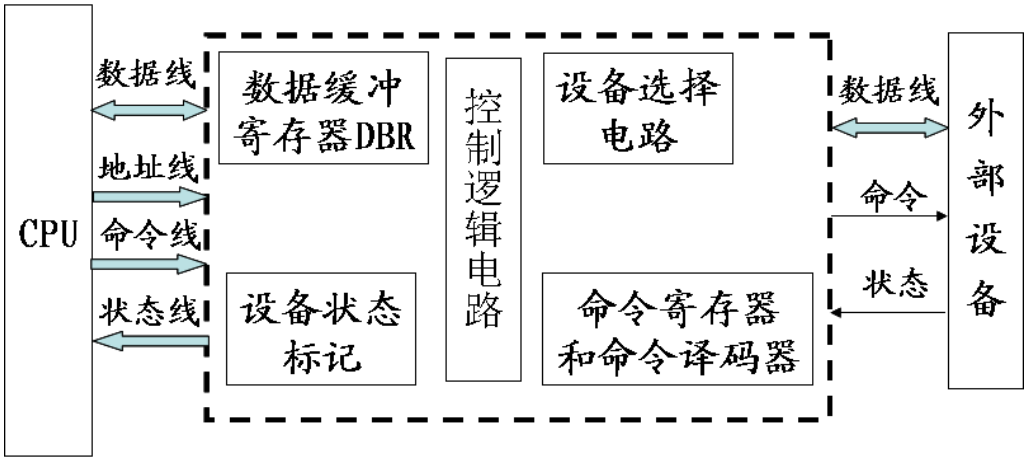
I/O

接口（端口）地址映射方式

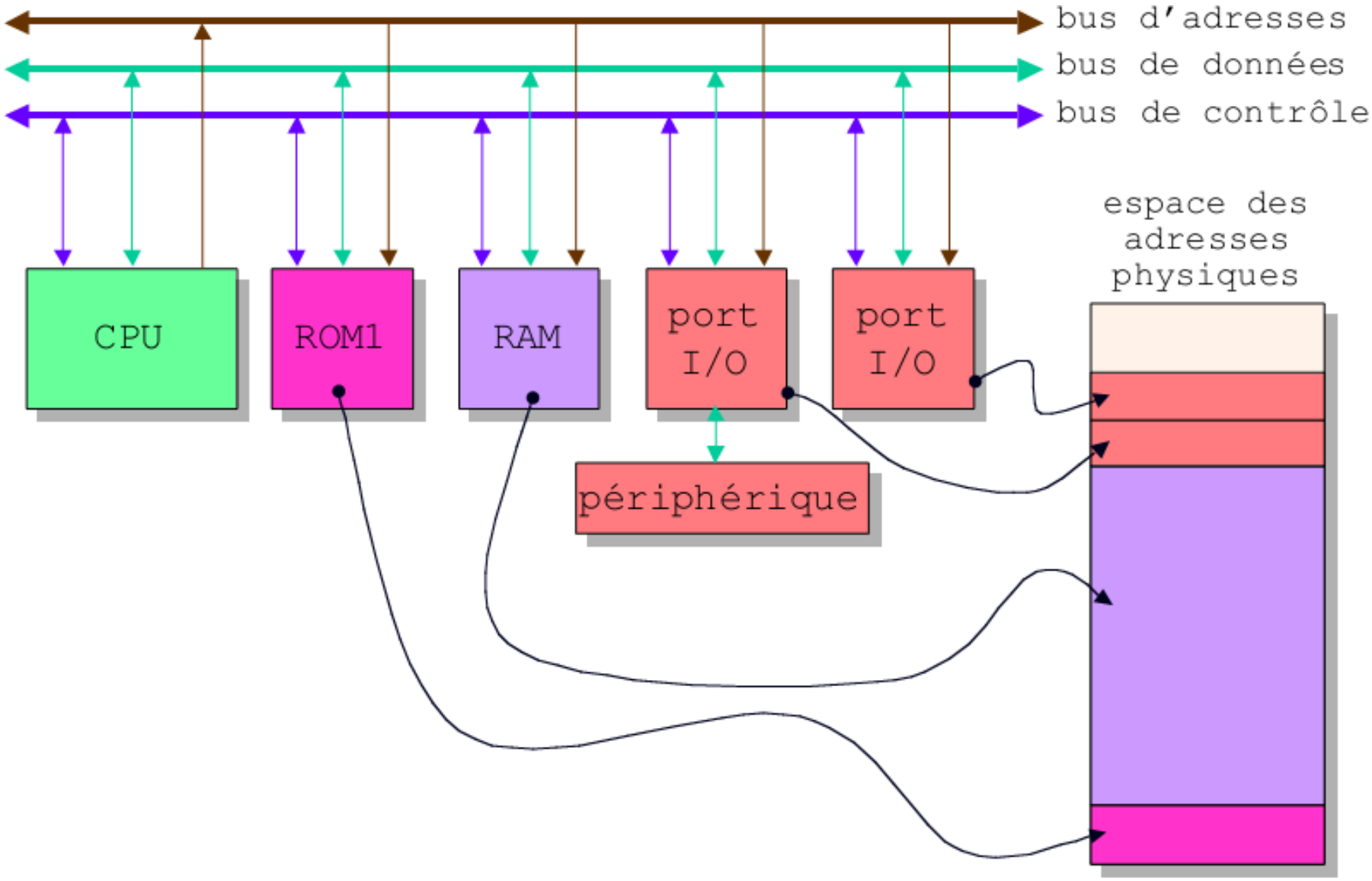
程序控制，中断，DMA

I/O Interface

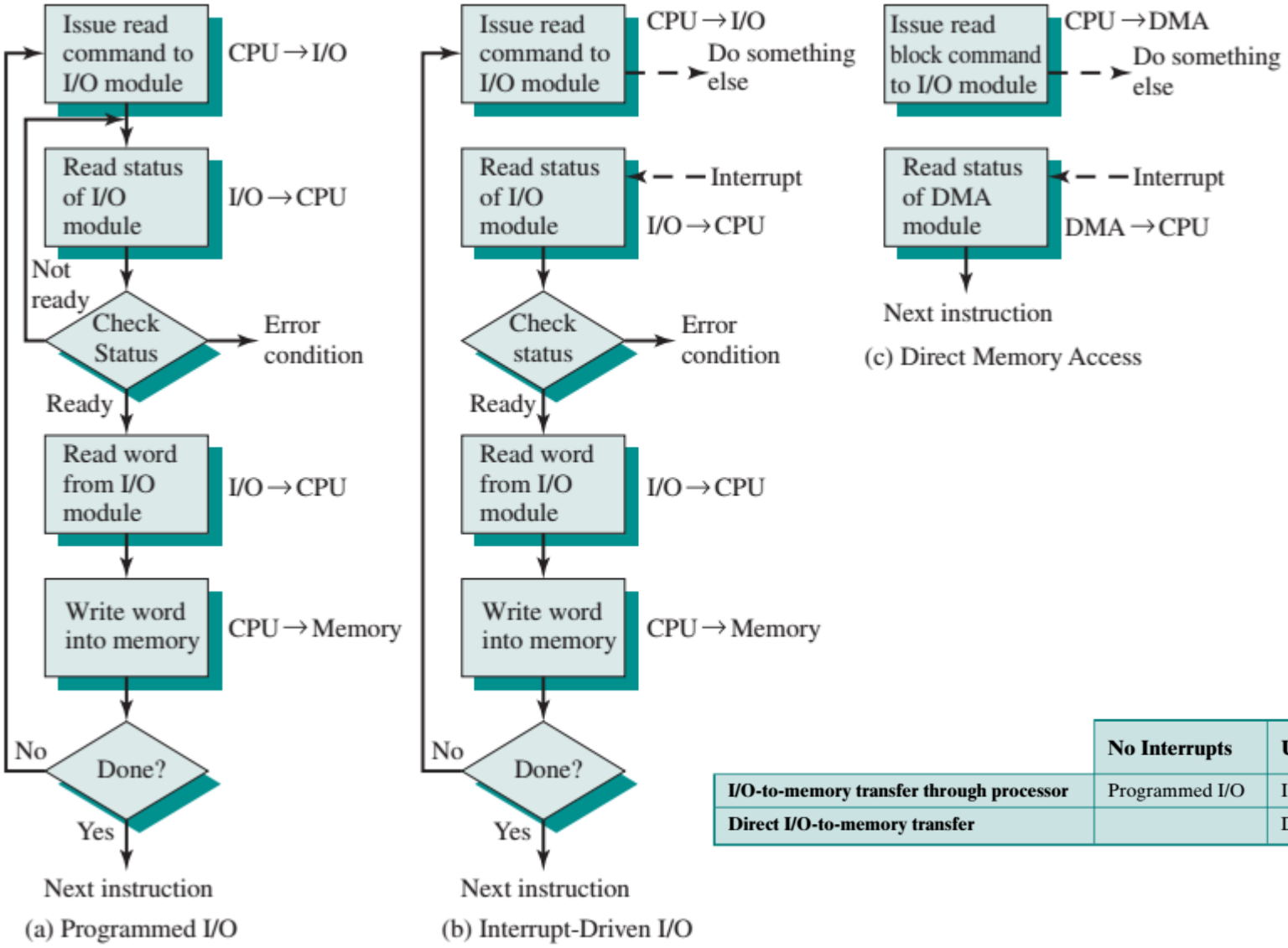




I/O Port Addressing: MMIO, PMIO

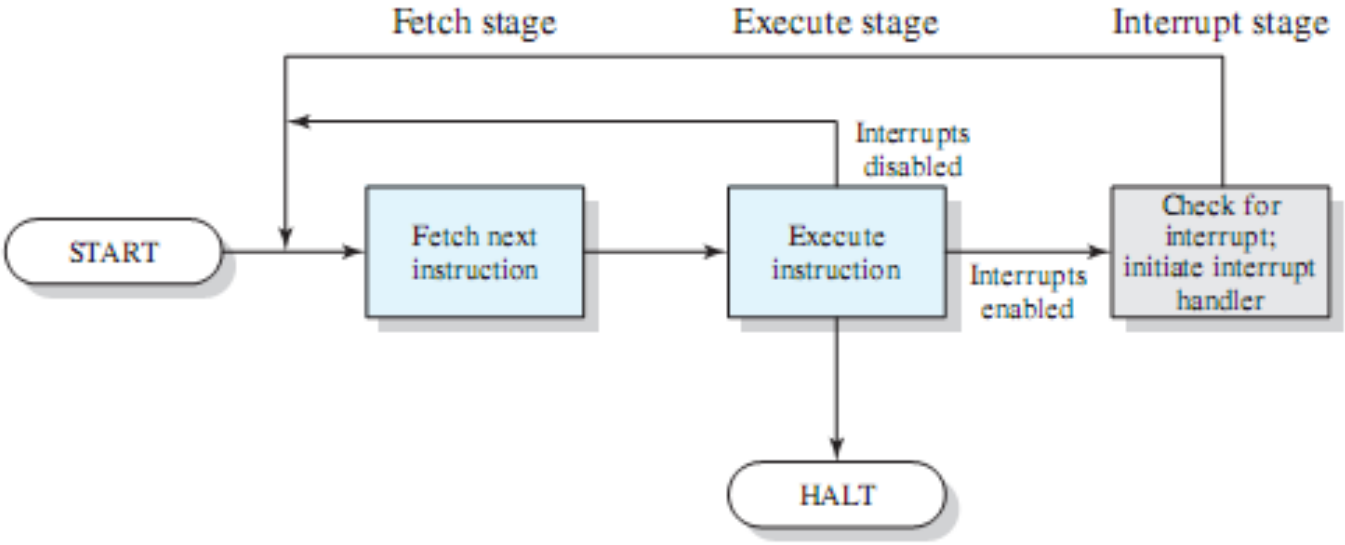


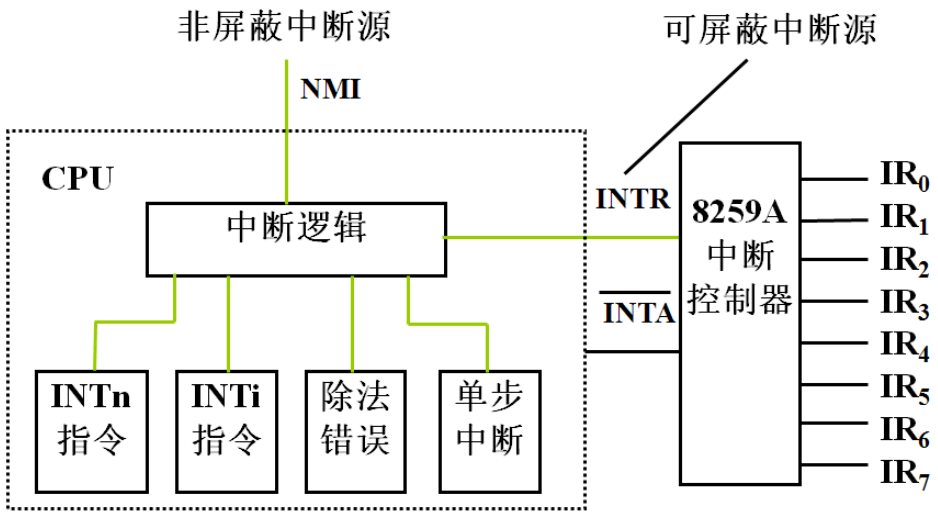
Three Techniques for Input of a Block of Data

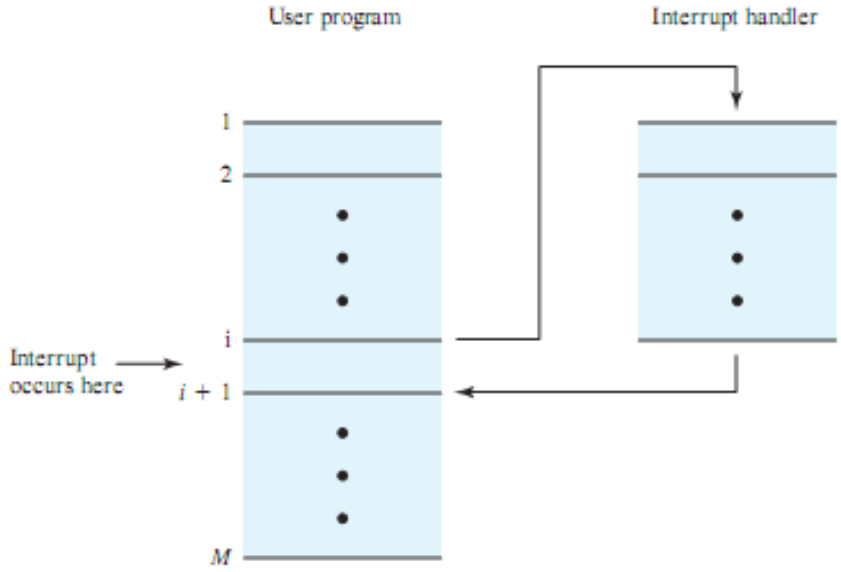


	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Instruction Cycle with Interrupts

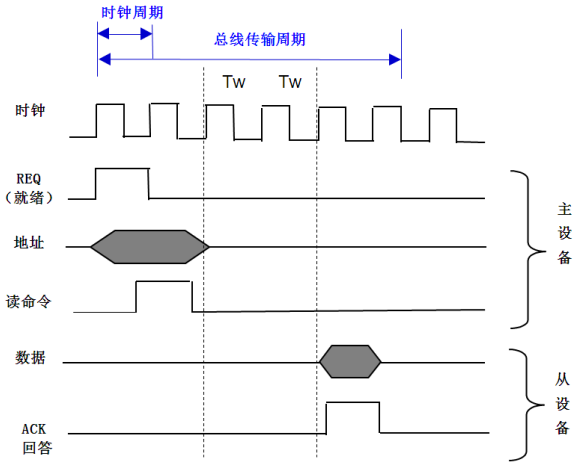


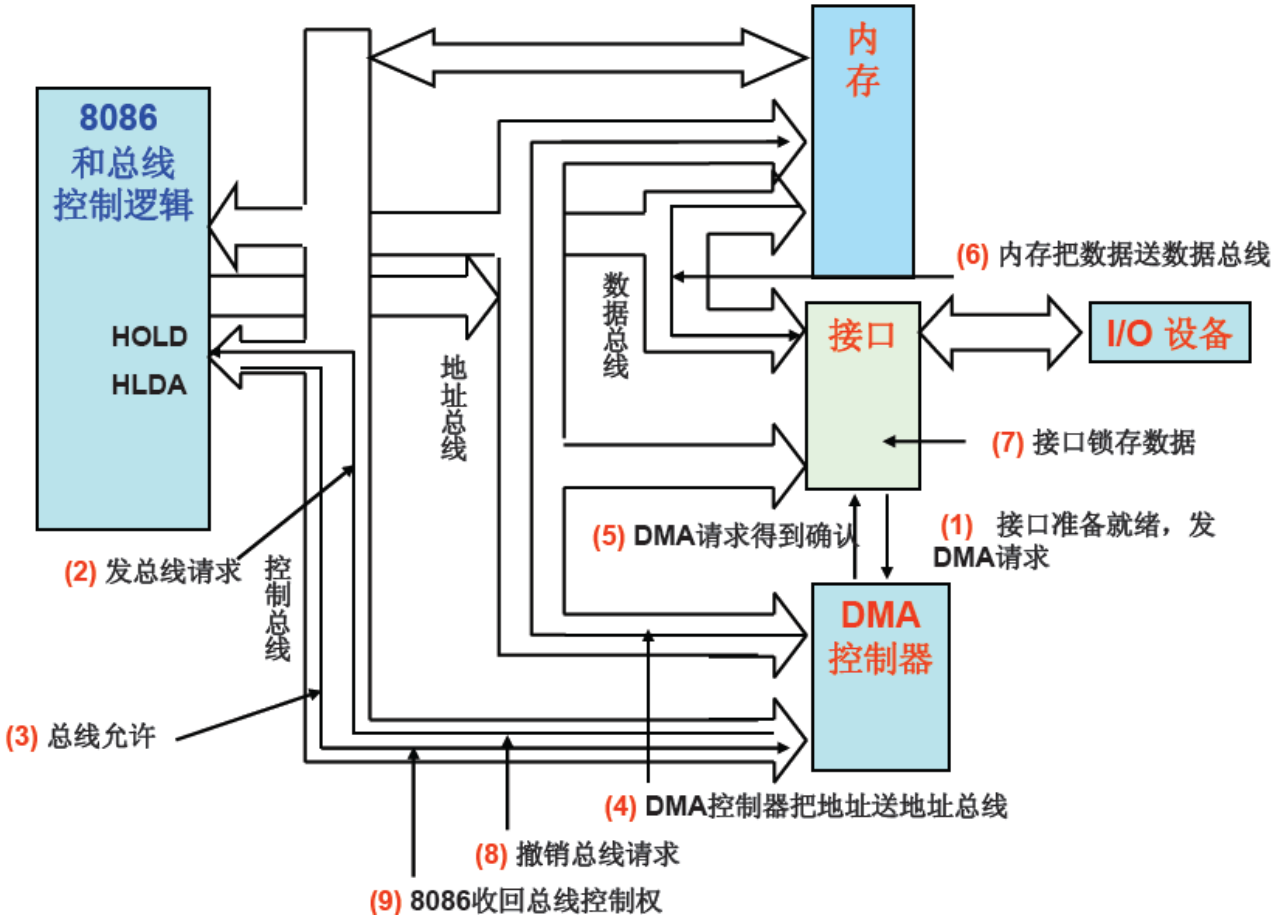




DMA控制I/O：CPU暂停

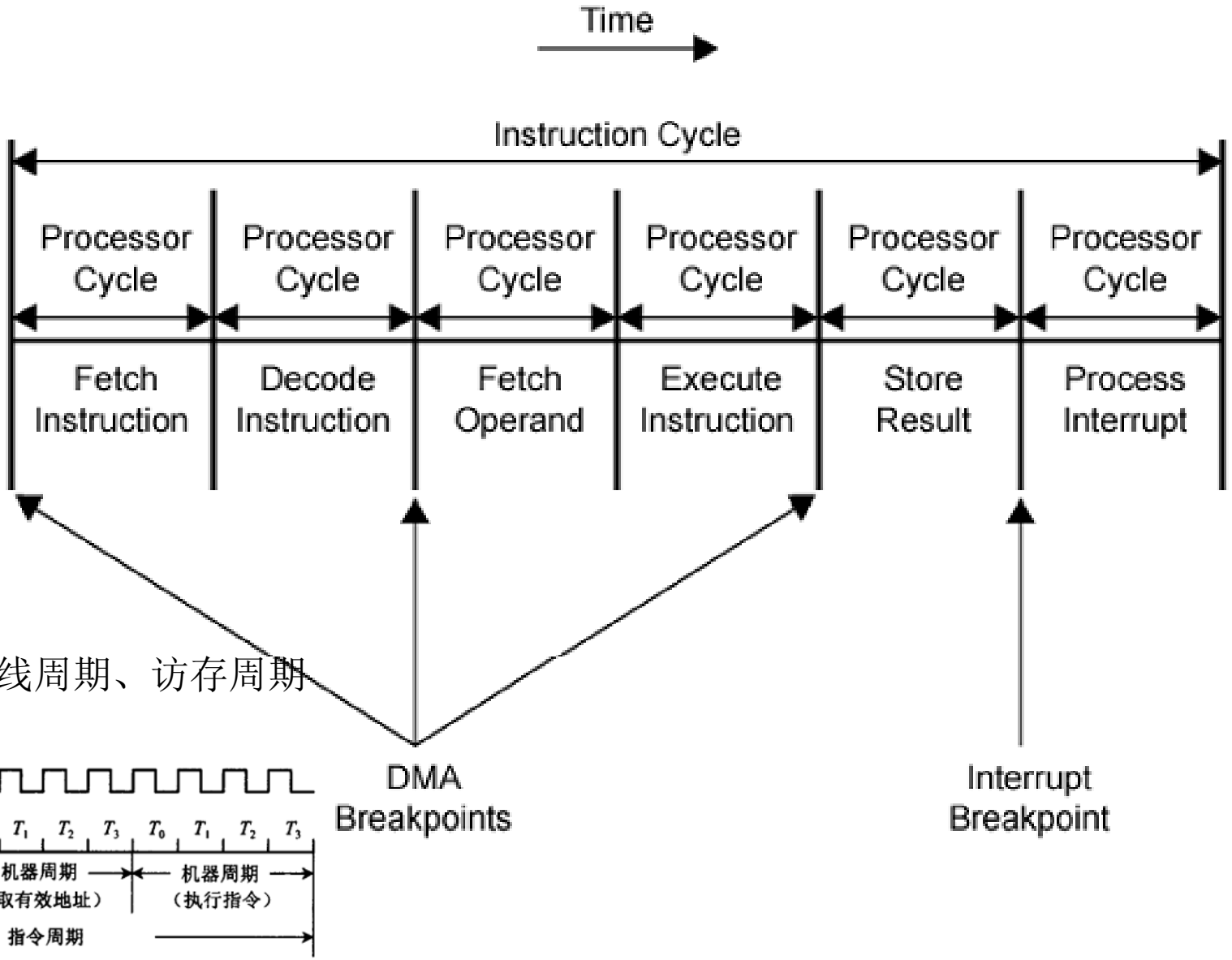
- I/O类型
  - 内存与外设
  - 内存与内存
  - 外设与外设





- 例：设备->内存
  - 类型：单字节，块传输，请求传输
  - 步骤：预处理（CPU），数据传输（DMAC），后处理（ISR）
    - CPU负责总线仲裁

DMA and Interrupt Breakpoints



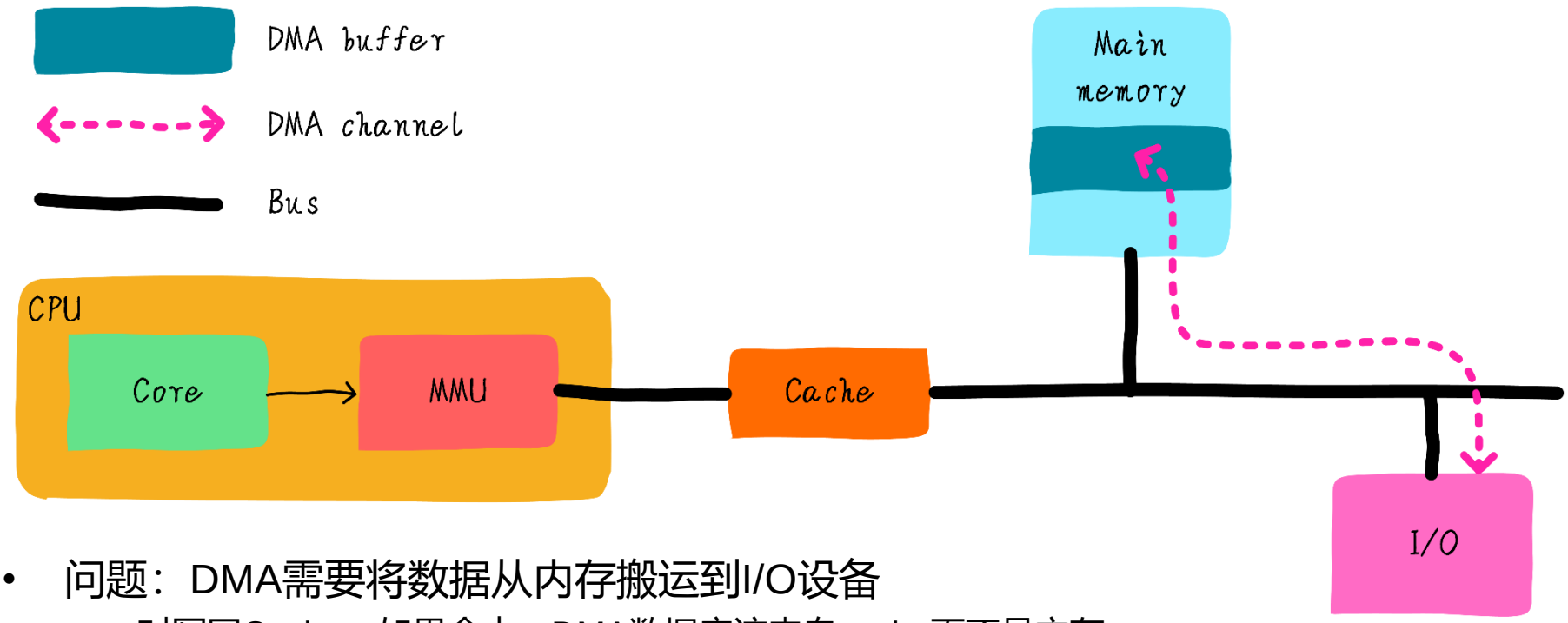
指令周期

机器周期：

CPU周期、总线周期、访存周期

时钟周期

DMA与Cache一致性：“I/O一致性”



- 问题：DMA需要将数据从内存搬运到I/O设备
  - 对写回Cache，如果命中，DMA数据应该来自cache而不是主存
  - DMA Buffer的Cacheline对齐问题
- 一致性维护
  - uncachable DMA buffer
  - 软件维护：DMA前进行Cache Flush/Invalidate
  - 总线监视技术：Cache控制器监视总线上的每一条内存访问，检查是否命中
    - PIPT Cache：用物理地址（Physical Address）作为索引（Index）和标签（Tag）的缓存

非典型问题☺

- 最小的计算机系统?
  - 需要哪几条指令？需要哪几种寻址方式？由哪些部件构成?
- C语言的计算机模型?
  - int i 在哪儿？全局变量与局部变量差异?
  - getchar()的执行过程?
- C语言、编译器、OS、COD的关系?
- 函数调用过程与中断响应过程的异同?
- 计算机现在在干啥?
- 总线的副作用?
- CPU如何访问RAM?
- Cache的副作用?
- 中断的副作用?
- DMA的副作用?
- 如何确定一个程序的执行时间?

Thank You