0. Pipeline

Multi-Cycle Processor Cycle time:

CT_{multi} =max(T_k) (CPI: cycle per instruction)

max(T_k) = longest stage duration among the N stages

$Time_{multi}$ = I × $\boldsymbol{Average\ CPI}$ × CT_{multi}

Pipeline Processor: 最快加速 N 倍 (N = pipeline stage)

$CT_{pipeline}$ = max(T_k) + T_d （ max(T_k) *和上面一样*）

T_d = Overhead for pipelining (e.g. pipeline register delay)

$Time_{pipeline}$ = (I + N − 1) × (max(T_k) + T_d)

1. Cache: Tag | Index | Block Offset

$$\text{Index} = \left(\frac{\text{地址}}{\text{块大小}}\right) \mod \text{组数。}$$

mod: 取模%

块大小(block offset) 隐含在题中: e.g. two-word blocks ➔ 块大小是 2 word, 1word = 4bytes ➔ 2*4(word) = 8 bytes, **block offset 位数**是 log2 8=3 bits。**Index 位数**: log2 (set 数)

Tag 位数 = 32 – Offset 位数 – Index 位数

$$\text{Tag} = \left\lfloor \frac{\text{Memory Address}}{\text{Number of Cache Blocks}} \right\rfloor$$

Cache hit：要 index 与 tag 都相同才算是 hit，否则是 miss。

$$\text{Hit Rate} = \frac{\text{Cache Hits}}{\text{Total Memory Accesses}}$$

分母为访问次数

平均访问时间 AMAT=hit rate * cache 时间 + (1-hit rate)*主存访问时间

2-way Set-Associative Cache：缓存总共有 **16 words**，分为 **4 个 Set（组）**，每组有 **2 个 block**，**2words/block**。

若目标组中的两条路都已占满，**替换策略**决定替换数据：**LRU**（最近最少使用）：替换组内最久未使用的数据块。**FIFO**（先进先出）：替换最早被加载的数据块。

数据加载策略：Write Through：每次写数据时，立即写回主内存。优点：主内存始终是最新的。缺点：写操作较慢。**Write Back：**只有当缓存块被替换时才写回主内存。优点：写操作更快。缺点：主内存可能不是最新数据。

缓存大小计算:

$$\text{Cache Size} = (\text{Block Size} \times \text{Number of Blocks}) + (\text{Tag Size} \times \text{Number of Bloc}$$

循环结构与缓存题目特点：多次循环访问同一块数据。**E.g.** 第一次循环未命中，后续命中。缓存容量是否足以保存循环中所有数据。

2. Process and Scheduling

进程的五种主要状态(Process States)：1.New（新建）：刚创建的进程。2.Ready（就绪）：已准备好运行，但等待 CPU 调度（e.g.刚被新创建、被抢占、I/O 完成）。3.Running（运行中）：正在执行的进程。4.Waiting（等待）：等待 I/O 等事件完成。5.Terminated（结束）：已完成执行或被终止的进程。Suspended（挂起）：是扩展状态，通常因系统资源不足或人为操作导致进程暂时移出内存。

$$\text{CPU利用率} = \frac{\text{CPU忙碌时间}}{\text{总时间}} \times 100\%$$

CPU 有时停下来等 IO

每次 fork() 调用都会使当前的进程数量翻倍，在 n 次循环后，总进程数量为 2**n

调度算法的比较：1.按每种算法规则排序；2.计算总等待时间；3.平均等待时间=总等待时间/程序总数

Scheduling Algorithms (调度算法分类)

- **Non-Pre-emptive (非抢占式):**

  1.First Come First Serve (FCFS): 按到达时间顺序调度。简单但可能有长等待时间(Convoy Effect)。**2.Shortest Job First (SJF):** 优先短任务，减少平均等待时间。需提前知道运行时间，可能导致饥饿。

- **Pre-emptive (抢占式):**

  1.Round Robin (RR): 时间片轮转，公平分配。适合交互式任务，时间片过大可能退化为 FCFS。**2.Priority Scheduling:** 根据优先级调度，高优先级抢占低优先级。缺点: 低优先级任务可能饥饿。**3.Shortest Remaining Time First (SRTF):** 优先剩余时间最短任务。动态抢占，延续 SJF 特点。

- **Multilevel Queue (多级队列调度):** 根据优先级将进程分配到不同队列。不同队列可能有不同的调度算法。

- **Multilevel Feedback Queue:** 动态调整队列优先级，结合抢占式和非抢占式算法。

Context Switching (上下文切换): 从一个进程切换到另一个进程时，保存当前状态并加载新进程的状态。**保存的内容:** 1.CPU 寄存器、程序计数器、堆栈指针。2.进程控制块(PCB)内容：进程 ID、状态、资源信息等。

$$\text{平均等待时间 (AWT)} = \frac{\sum(\text{开始时间} - \text{到达时间})}{\text{进程数}}$$

在就绪中等待时间

$$\text{平均周转时间 (ATT)} = \frac{\sum(\text{完成时间} - \text{到达时间})}{\text{进程数}}$$

从提交到完成时间

$$\text{平均响应时间 (ART)} = \frac{\sum(\text{首次 CPU 分配时间} - \text{到达时间})}{\text{进程数}}$$

从提交到响应

$$\text{吞吐量} = \frac{\text{完成的进程数}}{\text{总时间}}$$

单位时间内完成的进程数量。

3. Inter-process Communications

P( )(Wait/Proberen)：检查信号量的值。如果信号量值大于 0，则将其减 1，表示当前线程已进入关键区。如果信号量值为 0，则阻塞当前线程，直到信号量被释放。

V( )(Signal/Verhogen)：增加信号量的值，表示释放关键区资源。如果有线程因信号量为 0 而被阻塞，则唤醒其中一个线程。

互斥锁(Mutex)：锁只有两种状态。锁定(Locked)： 资源被占用，其他线程需要等待。解锁 (Unlocked)： 资源空闲，允许其他线程获取锁。（临界区保护）

4. Memory Management

Bit map 每个位表示一资源的状态：**1** 表示资源空闲或**可用**；**0** 表示资源已**被占用**。

Physical Address: (Segment, Offset) 合法地址：Segment + Offset <= Base + Length

Allocate Policy(动态分区)：首次适配（**First Fit**）：从头开始寻找第一个满足需求的分区。最佳适配（**Best Fit**）：寻找大小最接近需求的分区。最差适配（**Worst Fit**）：寻找最大的分区，以保留较多的碎片空间。

Buddy System 分块规则(Quick Fit)：总内存划分为 2 的幂次方大小的块。如果请求大小不能正好匹配现有块，则分裂为更小块，直到满足需求。**释放规则：**如果一个块被释放，且它的“伙伴”也未被分配，则它们可以合并成更大的块。伙伴是指同一级的、地址连续的两个块。

Internal Fragmentation occurs when the allocated memory block is larger than the actual memory required, leading to unused space within the block. This type of fragmentation happens in **fixed-size allocation units**.

External Fragmentation occurs when there are many small free memory blocks scattered across the memory, but they are non-contiguous and cannot satisfy larger memory allocation requests. This type of fragmentation occurs in **dynamic partitioning**.

5. File Systems

文件储存：• 连续空间，文件的数据紧密相连，读写效率高，一次磁盘寻道就可以读出整个文件。前提：必须先知道一个文件的大小，这样文件系统才会根据文件的大小在磁盘上找到一块连续的空间分配给文件。文件头(Linux incode)里需要指定「起始块的位置」和「长度」。**但是有「磁盘空间碎片」和「文件长度不易扩展」的缺陷。•** 非连续空间存放方式两种：**「链表方式」离散的，非连续的**，可以**消除磁盘碎片**，提高磁盘空间利用率，同时**文件长度可以动态扩展**。链表可分为「**隐式链表**」和「**显式链表**」两种形式（是否将指针集中放在一个表中）。**「索引方式」**每个文件都有「索引数据块」，里面是指向文件数据块的指针列表，就像书的目录；文件头需要包含指向「索引数据块」的指针，可以通过文件头知道索引数据块的位置，再通过索引信息找到文件剩下对应的数据块。

块 (Block): 数据存储的基本单位，每块大小固定（如 2048 bytes）。块号 (Block Number): 每个块的唯一编号，用于指示数据存储位置。

多级索引 (Multi-level Index):直接指针 (Direct Pointer): 直接指向数据块。一级间接指针: 指向一个块，该块存储指向数据块的指针。二级间接指针: 指向一个块，该块存储一级间接指针。三级以此类推

文件最大大小=各级指针数据大小累加

32bits 的指针= 4 bytes，每 block 2048 bytes(块大小)➔每 block 指针数=2048/4 = 512

n 级间接指针数据大小=每块指针数^n ×块大小 (直接指针算是 n=0 级)

分区最大的大小=最大块数×**块大小(题目)**

最大块数 =2^块号位数

FAT16 使用 16 位表示块号，最多支持 2^16=65536 个块（最大块数），要减去初始块

块号 k 范围 = [k×块大小, (k+1)×块大小−1]

从字节偏移计算块号: 块号=字节偏移/块大小；

计算跨越的块，偏移范围：[起始字节,起始字节+读取字节数−1]；**块号范围：**覆盖该字节范围的所有块号。

1KB=1024bytes;1 MB=1024 KB=1024×1024=1,048,576 bytes;1GB=1024MB=1024×1024×1024=1,073,741,824bytes

6. Operating Systems

Monolithic Kernel（整体内核）将所有操作系统核心服务都集中在内核模式运行。服务之间的通信直接通过函数调用实现，性能较高。一旦某个服务发生崩溃，可能导致整个系统失效。例如 Linux 和 Windows。

Microkernel（微内核）将内核功能最小化，仅保留核心功能，而其他服务（如文件系统和驱动程序）运行在用户模式。即使某个服务崩溃，也不会直接影响到其他服务或内核本身。频繁的模式切换可能带来一定的性能开销。扩展性较好，例如 QNX 和 MINIX。

slt $t0, $t1, $t 若 $t1 < $t2，将 1 存储到 $t0，否则存储 0。**slti** $t0, $t1, imm 若 $t1 < imm，将 1 存储到 $t0。**beq** $t0, $t1, label 若 $t0 == $t1，跳转到 label。**bne** $t0, $t1, label 若 $t0 != $t1，跳转到 label。**j** label 无条件跳转到 label。

Single cycle 时间周期由最长指令确定，全部指令共享这一时间周期；Multiple 每条指令分为多个阶段(IF, EX, MEM 等), 最短阶段决定时钟周期；Pipeline 多指令同时处于不同阶段。

指令统计 = 循环外指令(初始化, lw sw 等) + 循环内指令*次数