Moore's Law: 18 mouths, transistors in a chip(芯片) doubles

Clock Speed: pulse frequency by clock(synchronized pulse)

CPI (Cycles per instruction): $CPI = \frac{\sum_i CPI_i \times I_i}{I_c},\ I_c = \sum_i I_i$

I_i: Each instruction's proportion of program

CPI_i: The number of cycles of each instruction

MIPS (Million Instruction per Second):$\boldsymbol{MIPS = \frac{f}{CPI \times 10^6}}$

MFLOPS (Million Floating point Operations per Second)

We better actually **measure** the time of **doing real jobs**:

SPEC marks (Standard Performance Evaluation Corporation)

Boolean Algebra Laws:

1. Commutative Law: AB =BA, A+B=B+A
2. Identity elements: 1 · A =A,0+A=A
3. Null law: 0 · A =0,1+A=1
4. 幂等定律 Idempotent law: A· A =A, A+A=A
5. Inverse law: $A \cdot \bar{A} = 0, A + \bar{A} = 1$
6. 结合律 Associative law: A· (B·C) = (A·B)·C, A+(B+C)=(A+B)+C
7. Distributive Law: A(B +C) = AB+AC, A+(BC) = (A+B)(A+C)
8. DeMorgan's Theorem: $\overline{A \cdot B} = \bar{A} + \bar{B},\ \overline{A + B} = \bar{A} \cdot \bar{B}$

Name	Graphical Symbol	Algebraic Function	Truth Table
AND	A, B → F	F = A • B or F = AB	A B | F: 0 0 | 0; 0 1 | 0; 1 0 | 0; 1 1 | 1
OR	A, B → F	F = A + B	A B | F: 0 0 | 0; 0 1 | 1; 1 0 | 1; 1 1 | 1
NOT	A → F	F = $\bar{A}$ or F = A'	A | F: 0 | 1; 1 | 0
NAND	A, B → F	F = $\overline{AB}$	A B | F: 0 0 | 1; 0 1 | 1; 1 0 | 1; 1 1 | 0
NOR	A, B → F	F = $\overline{A + B}$	A B | F: 0 0 | 1; 0 1 | 0; 1 0 | 0; 1 1 | 0
XOR	A, B → F	F = A ⊕ B	A B | F: 0 0 | 0; 0 1 | 1; 1 0 | 1; 1 1 | 0

Functional completeness: We can only use NAND/NOR to express all operations.

SOP (Sum of products) / **POS** (Product of sums)

Half Adder: $\boldsymbol{Sum} = A \oplus B; \boldsymbol{Carry} = A \wedge B$

Full Adder: $\mathbf{Sum} = A \oplus B \oplus Cin; \mathbf{Cout} = (\mathrm{A} \wedge \mathrm{B}) \vee (\mathrm{Cin} \wedge (\mathrm{A} \oplus \mathrm{B}))$

Finite Precision Number: The **size** and **number of possible representations** is **fixed**

Radix (基数) Number Systems: $(\ldots \boldsymbol{a_2 a_1 a_0 a_{-1} a_{-2}} \ldots)_r = \sum_i \boldsymbol{a_i r^i}$

Faster way to calculate value of radix number system:

$$D = \sum_{i=0}^{n} a_i \times r^i = r\left(\sum_{i=1}^{n} a_i \times r^{i-1}\right) + a_0$$
$$= r\left(r\left(\sum_{i=2}^{n} a_i \times r^{i-2}\right) + a_1\right) + a_0 = \ldots$$

Excess K (offset binary/excess code/biased representation):

We **usually** use $K = 2^{n-1}$

One's complement: For -, invert all bits, $value = 2^n - f - 1$

Two's complement: For -, invert all bits then add 1, $value = 2^n - f$

Range Expand: For -, add 1 before original bits.

Floating-point representation:

$$\boldsymbol{Value} = \pm(\mathbf{1}(\boldsymbol{or}\mathbf{0}).\boldsymbol{xxxxxxx})_2 \times \mathbf{2}^{\boldsymbol{Exponent}}$$

Sign bit: 1="-"; **Exponent**: Excess $2^{m-1} - 1$; **Significand**: xxxxxx

Expressible Numbers: we assume m-bit exponent, n-bit significand:

$2^{1-2^{m-1}} \le |range| \le (2 - 2^n) \times 2^{2^{m-1}}$(不考虑 IEEE 特殊情况)

Multiplication of 2's complement: 若乘数为负，则乘数每个 1 与被乘数相乘的结果按负数 range expand(添 1)，求和即为结果的补码表示

Floating-point Addition/subtraction:

1 Check for zeros **2** Align the significand **3** Add (Subtract) the significand **4** Normalize(right/left shift) **5** Rounding

Floating-point Multiplication: (exp_i均为 bit pattern)

$$(\pm)\boldsymbol{m_1} \times \mathbf{2}^{\boldsymbol{exp_1}} \times (\pm)\boldsymbol{m_2} \times \mathbf{2}^{\boldsymbol{exp_2}} = (\pm)\boldsymbol{m_1} \times \boldsymbol{m_2} \times \mathbf{2}^{\boldsymbol{exp_1 + exp_2}}$$

Bit pattern: $e_{result} = exp_1 + exp_2 - K$

Floating-point Division:

$$\frac{(\pm)m_1 \times 2^{exp_1}}{(\pm)m_2 \times 2^{exp_2}} = \frac{(\pm)m_1}{(\pm)m_2} \times 2^{exp_1 - exp_2}$$

Bit pattern: $e_{result} = exp_1 - exp_2 + K$

Von Neumann architecture:

Instruction data stored in a single read-write memory; Contents of memory addressed by location; Execution occurs in sequential fashions.

Word: unit of organization (memory), smallest unit to process at once.

PC-Program Counter: Holds the next instruction(address)

IR-Instruction Register: Contains the current instruction.

MAR-Memory Address Register

MBR-Memory Buffer Register (aka Memory Data Register, MDR): Contains the data returned from external memory||the data to be written to external memory, as **bridge** between CPU and external memory.

I/O A(B)R-I/O Address (Buffer) Register

IAC: Instruction Address Calculation (use PC, PC+=4)

Instruction Fetch: Fetch instruction from Memory

PC —> MAR, mem[MAR] —>IR

Instruction Execute: Execute the fetched instruction

Register and cache-fast, but small and expansive

Main memory, External memory- large and cheap, but slow

Go down the **memory hierarchy**: Cheaper, larger, slower; less frequency.

Principle of locality: 相邻/最近被引用的 memory 更容易被下次调用

Big Endian Mode: 0X2345678->12 34 56 78

Little Endian Mode: 0X2345678->78 56 34 12

CPU used to read memory word by word, but now it uses cache.

Access Method: Sequential Access, Random Access and their associate.

RAM (Random access memory): 传统 RAM 是 volatile，不供电会损失数据

SRAM (static RAM):用 logic gate 存储数据，快，无需刷新，用于缓存内存

DRAM (Dynamic RAM): 用 transistor，慢，需刷新，用于主存，便宜

ROM (Read-only memory): 数据不 volatile，用于存储 start-up program

PROM (programmable ROM): 只能被写入一次(以电子方法)

Read-mostly memory: (以下三种)

Erasable PROM:强紫外线(intense ultraviolet)擦除; **Electrically EPROM**:电

flash: 介于前两者之间, non-volatile，快，写入前成块擦除，成本低

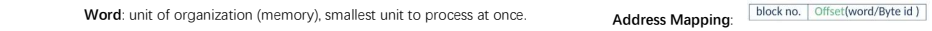
Memory Performance: Access time(一次 R/W 操作的时间), Memory cycle time(前者+传输时间), Transfer rate

Burst Mode: 当数据被访问时，相邻数据更快被传输

数据传输中可能**错误**(voltage spikes, electromagnetic interference，电源)

使用**单个奇偶校验位**(Single Parity Bit)**汉明码**(Hamming Code)

Address Mapping:



块尺寸为 b，则$Addr\%b$为块内编号，$Addr//b$为块编号->三种映射方式

Direct-Map: 1cache line 固定对应一部分 block，简单快速便宜，但易发生冲突(造成命中率下降)

Address

tag-对应的内存块标签; index-缓存行编号，对行总数取模; offset

Fully associative: tag+word, block number=Tag

每次访问数据须比对所有缓存行，慢但是冲突少

K-way set associative(上述两方法的结合):将缓存分为多个 set，每个 set 有 k 行，按直接映射匹配 set 编号, set 内按全相联映射.

Address:

结合了上述两方式的优点

Replacement Algorithm:

FIFO-简单，慢; LRU-复杂，命中率高; RR(Random replace)简单，不稳定

Write Strategy (为保持 cache 与 memory 数据一致性):

Write through: 当写入 cache 时同时写入主存，慢,但数据安全

Write back: 当 cache 中数据被移除时，写回主存，快，但不易 I/O

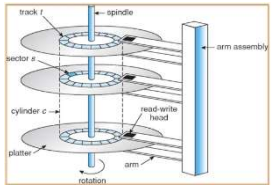
多重缓存:

$$\begin{aligned} &average\ access\ time \\ &= hit\ rate \times hit\ time + miss\ rate \times average\ time\ (when\ cache\ miss) \\ &= hit\ time + miss\ rate \times miss\ penalty \end{aligned}$$

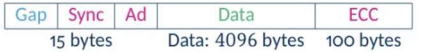
Miss penalty: extra time when cache miss occurs

Split caches: 缓存指令与缓存数据物理上分开且固定大小，冲突少/灵活

Unified Caches: 复杂但是灵活



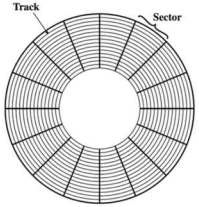
Each sector:



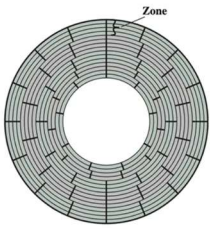
Gap: separates sectors; **Sync**: indicate 后为信息域& timing alignment

Ad(Address Mark 地址标记) **ECC**(Error Correction Code，错误校正码)

- Constant angular velocity (CAV)



- Multiple zone recording (MZR)



Disk access time: $Seek + Rotational\ delay + Data\ Transfer$

Transfer time: $t_T = \frac{b}{N} \times \frac{1}{r}$, b: 传输字节数; N: 磁盘字节数; r: 每秒旋转圈数

Solid State Drives(SSD): Semiconductor, Flash Memory，高 IO 性能，

Durability 耐用(no mechanical wear)，便宜，快

RAID: Redundant Array of Independent (Inexpensive) Disks，独立磁盘冗余阵列，将多个物理硬盘驱动器组合成一个逻辑硬盘. 通过条带化(Striping)

RAID 级别	数据处理方式	容错能力	性能特点	主要优点	主要缺点
RAID 0	条带化(Striping)	无	读写速度最快	最大化性能提升	任何磁盘故障导致全部数据丢失
RAID 1	镜像(Mirroring)	完全容错	读取速度较快，写入速度取决于写入镜像的过程	数据冗余，高可靠性	成本高，磁盘利用率低
RAID 2	条带化加专用汉明码校验	单个磁盘故障可恢复	-	专为需要纠正多位错误的应用设计	实际应用中很少使用，因为需要特定数量的磁盘
RAID 3	字节级条带化加专用奇偶校验盘	单个磁盘故障可恢复	高读写传输速率	高效率的数据传输速率	写操作性能受限于奇偶校验盘
RAID 4	块级条带化加专用奇偶校验盘	单个磁盘故障可恢复	高读取速度，写入受制于奇偶校验更新	允许独立访问任意单个磁盘	写操作瓶颈在奇偶校验盘
RAID 5	块级条带化加分布式奇偶校验	单个磁盘故障可恢复	优秀的整体性能，尤其是随机读取	分布式奇偶校验提高写入性能	复杂度增加，重建期间性能下降
RAID 6	块级条带化加双重分布式奇偶校验	可承受两个磁盘同时故障	优于RAID 5的写入性能	更高的数据冗余度	更复杂的编码解码过程，更多的计算资源需求

恢复磁盘故障：校验盘 P 的每位=所有盘的对应位异或和

asynchronous communication(异步通信)

I/O Module：Interface to the processor and memory. **Need** Control, Timing, Processor/Device Communication, Data buffering

External Devices: Human readable and Machine readable.

Programmed I/O：处理器直接控制 I/O，但在等待时会浪费 CPU 周期

Interrupt-driven I/O: 处理器发出 I/O 命令后继续执行其他进程，当 I/O 完成时，I/O 模块会中断 CPU，CPU 暂停当前程序处理中断。

Interrupt Processing:

Hardware side

1. Device issue an interrupt signal to processor
2. Processor finish current instruction execution
3. Processor check for interrupt, find one, send an interrupt acknowledge (INTA) signal to device.
4. Device remove interrupt.
5. Processor need to remember the current position of the program before jumping to the interrupt servicing routine. This is done by putting the PC and the flag registers (Processor Status Word PSW) to the stack.
6. Processor then load PC with address of interrupt routine

Software side

1. The Interrupt routine will save those registers into the stack which have been used by the interrupt routine (so that they can be recovered).
2. The Interrupt routine perform required action, e.g. Reading data from device.
3. When finished, the interrupt routine will restore the saved registers.
4. The last instruction executed by the interrupt routine is a Return from Interrupt instruction (RETI), which will restore the PSW and PC from the stack, in the reverse order that they were saved.

DMA (Direct Memory Access):

IOP (Input-Output Processor): steal cycles from CPU:

IOP issues signal 给 CPU 使其与 buses 断连，并 elongate clock

Procedure/Function: self-contained 程序, economy&modularity.

返回的地址储存在 top of system stack, LIFO

General/Dedicated purpose registers:通用/专用 寄存器

PSW (processor status word) or flag register: 用于条件判断

有两种数据储存: **Numeric**：整数或浮点数;

Non-numeric: (character)ASCII or Unicode/Binary data (bit-map)

Addressing methods:符号约定:

A-地址字段内容;R-寄存器;EA-effective 地址;(X)-位置 X 的内容

立即寻址 Immediate Addressing: $op = A$　Literal mode(小立即数);

Immediate Operand in the Next Word-对于大立即数

Direct Addressing (Absolute Addressing)$EA = A$

Indirect Addressing: $EA = (A)$

Register Addressing: $EA = R$

Register Indirect Addressing: $EA = (R)$

Displacement Addressing: $EA = A + (R)$

Stack Addressing: $EA = top\ of\ stack$

Table 13.1 Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

Assembly Language Programming:

Each lines has 4 fields: [label] mnemonic operand list comment

#114-立即数; [1234]-内存地址; R1(4)-偏移量;

OS 提供的服务：Program Creation, Execution, IO Access, File System Management, System Access, Error detection and response.

Protection Scheme: user mode/Kernel(Supervisor) mode

Multi-tasking and Time-sharing System: 每个进程(process)被分配固定的 time slice，每用完一个 slice 会切换，若在执行 IO 会主动释放 CPU

Virtual memory：Physical 地址：用于实际访问内存的地址;

Logical 地址：程序使用的地址空间，不一定对应物理内存的地址

MMU (Memory Management Unit):基于局部性原理工作，并通过地址映射将程序的逻辑地址转换为物理内存地址.

Page：逻辑物理地址空间被划分为 fix size 页面，每进程有自己的逻辑地址空**PTE** (Page Table Entries):V- Valid bit, page 是否在内存中;

P- Protection information(权限); D-Dirty bit， 页面是否已被修改

Demand Paging：当程序启动时，内存空，读数据才加载 page.

Fast response，但是会大量 page faults，使用 prepaging 解决缺点

Cache	Virtual memory
Block	Page
Block Size	Page Size
Block Offset	Page Offset
Miss	Page Fault
Tag	Virtual Page Number

Address Translation: 程序通过 CPU bus 内存，提取 logical page number，查找对应 PTE-若页面在内存，则获取 physical frame number 并附加页内偏移量；若不在，则生成一个 page fault，通过 IO 从 hard disk

Page Fault 处理: 从硬盘加载 page，在物理内存中找到空闲页面，无则择一替换(replace,判断是否写回 HD)，更新 PTE

Page Replacement: FIFO or Least Recently Used (LRU)

Write Strategy：通常采用 **Write back**，因为写入硬盘慢/成本高

TLB (Translation Lookaside Buffer)：每次虚拟内存引用可能导致两次物理内存访问：获取 PTE && 获取数据。使用 buffer 加速

结合缓存与虚拟内存: 获得物理地址后，使用该地址访问缓存内存，并根据情况处理缓存未命中问题。(在地址转换之后)确保高效的数据访问流程

Processor Organization: 处理器通过 **Data Movement** 与 **Data Processing / Transformation** 执行指令. 数据存储在 register 中，用控制单元(Control Unit)生成的控制信号来协调操作.

Instruction Execution Cycle: FI (Fetch Instruction)-**DI** (Decode Instruction)-**CO** (Calculate Operand Address)-**FO** (Fetch Operands)-**EI** (Execute Instruction) -**WO** (Write Operand)

指令流水线 (Instruction Pipeline): 目标是提高吞吐量(Throughput)

现代 RISC 架构：FI, DI, EX(执行), MEM(访问内存), WB(写回)

理想的 pipe:仍需要 5stage，但是平均一个 cycle 只需要 1 指令(CPI=1)

流水线冒险(Pipeline Hazards):

资源冲突(Resource/Structural Hazard)硬件资源被多个阶段同时请求解决

方案：增加更多 resource; separate data/instruction cache (split cache)

数据冒险(Data Hazard)当前指令依赖于前面指令的结果，但尚未写回

类型：RAW(Read After Write，最常见)**WAR** and **WAW**

方案: 优化指令顺序; **转发(**Forwarding, ALU 输出直接到下一个指令输入),

控制冒险(Control Hazard)条件分支指令导致无法确定下一条指令地址

方案：**静态预测**(Static Prediction)：总是预测不跳转或跳转。

延迟槽(Delay Slot)：在分支指令后插入不影响判断结果的指令。

动态预测(Dynamic Prediction)：根据历史记录预测，准确率可达 90%+。

simple branch prediction-预测错误就改变，可能会导致 **frequently** 误判.

双错误阈值机制: two consecutive wrong prediction to change decision.

处理器性能公式与影响因素

$$Execution\ Time = Instruction\ Count \times CPI \times Clock\ Cycle\ Time$$

IC：程序所需指令数量; CPI(Clocks)：每条指令平均所需时钟周期数。

Clock Cycle Time(T)：每个时钟周期的时间长度(即 1 / 主频)。

提升处理器性能的关键技术

1. 更有效的指令流水线(Pipelining)减少 CPI，提高吞吐量(idea CPI=1)
2. 多执行单元(Multiple Execution Units)同时执行多条指令(提高 IPC)
3. 更大寄存器文件(Register File)减少内存访问次数 → 提高速度。
4. 精简指令集(RISC)简化指令格式 → 减少 eliminates microprograms

Characteristics of Modern Processors: 1 所有指令都是 register-register type(除 LOAD, STORE 外); 2 指令 fixed 长度与 format,简单; 3 少指令和寻址方式; 4hardwired 替代 microprogrammed control; 5 使用 instruction pipelining, extensive SW and HW techniques;

指标	RISC I	VAX-11 / M68000 / Z8002
指令数量增加	20%~30%	—
CPI	显著降低（2~4倍）	较高（如 VAX CPI=8）
时钟周期时间	更短	更长

Characteristics of RISC: 同 CMP1235+optimizing(优化) compiler