计算机组成原理复习

要点内容

1. 什么是总线，总线传输的特点“
   总线是连接两个或两个以上设备的通信线路
   总线传输的特点：共享传输介质
   总线包括：内部总线、外部总线、系统总线（数据总线、地址总线、控制总线）
   数据总线的宽度等于机器字长；地址总线宽度等于 $\log_2$(存储单元个数)
   2.磁盘的突发传输速率：磁盘每秒最多能传输的数据（忽略寻道时间和平均旋转延迟，只计算数据传输的速度）

第一章 计算机组成与结构简介

一、计算机组织和和架构(Organization and Architecture)

a) 计算机架构、组成、实现的概念
   i. 计算机体系结构：是那些对程序员可见的系统属性，这些属性直接影响到程序的逻辑执行
   ii. 计算机组成：实现了某种架构的操作单元以及操作单元的内部连接。组成是

架构的一种实现，对系统设计员可见

计算机架构是计算机的逻辑设计，组成则是这种逻辑设计的具体实现。

b) 具体的属性：

i. **Architectural attributes**: instruction set(指令集)、word length(字长)、I/O mechanism(I/O 机制)、addressing(地址)

ii. **Organizational attributes**: control signal、interface、memory technology、bus technology 等等对程序员透明的硬件细节

iii. **Implement attributes**: Integrated Circuits (ICs), Printed Circuits (PC) boards, Power Supplies, Chassis, Connectors and Cables，etc. 集成电路，印刷电路(PC)板，电源，机箱，连接器和电缆

c) 系列机：

i. 定义：一组架构不变，组成不同的计算机

ii. 兼容性：

1. **Upward compatibility:** program for low level computer can run over high level computer without modification
2. **Backward compatibility:** program for current computer can run over future computer without modification

二、计算机结构与功能(Stucture and Function)

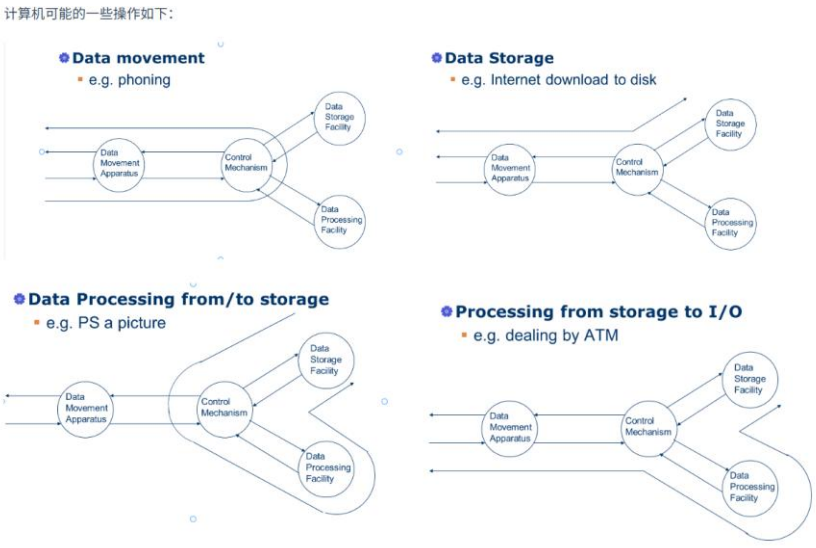
a) Computer Function:

i. Data processing：数据处理

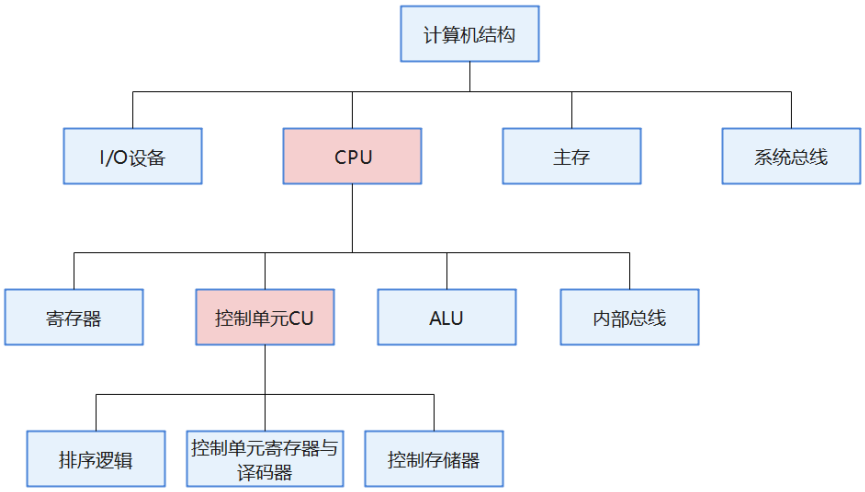
ii. Data storage：数据存储

iii. Data movement：数据传输

iv. Control ： 控制



b) Computer Structure:



第二章 计算机演变与性能

一、单位换算

- **Quantity and Unit in common use**
  - Bit -- b
  - Byte -- B： 8bit= 2^3
  - K (Hz, bytes)： 10^3 --1024＝2^{10}
  - M: Mega (bytes,Hz)： 10^6 --1024^2＝2^{20}
  - G: Giga (bytes,Hz): 10^9 -- 1024^3＝2^{30}
  - T: tera (bytes，Hz): 10^{12} --1024^4＝2^{40}
  - P: peta (bytes，Hz): 10^{15} --1024^5＝2^{50}

重点：一个字节是八比特

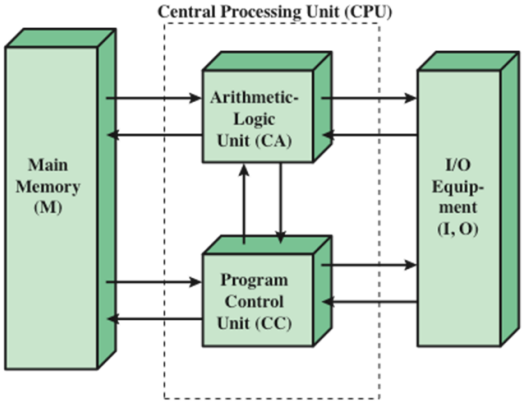
二、计算机的发展历史

a) 第一代计算机：电子管时代
b) 第二代计算机：晶体管时代
c) 第三代计算机：中小规模集成电路时代
d) 第四代计算机：大规模超大规模集成电路时代

世界上第一台计算机：ENIAC——宾夕法尼亚大学

e) 冯·诺伊曼机(图灵机/IAS)
   i. 控制器
   ii. 存储器
   iii. 运算单元
   iv. 输入输出设备(IO)

Structure of the IAS computer



三、摩尔定律(Moore's Law)
a) 内容：集成电路上可以容纳的晶体管数目在大约每经过 18 个月便会增加一倍。
b) 性能平衡：
   i. 平衡 CPU 与 DRAM(Direct Random Access Memery——直接访问随机存储器)：
   1. 改进 CPU 与 DRAM 之间的接口；
   2. 缓冲与缓存
   3. 使用多处理器配置（主处理器和协处理器）

四、Multicore，MICs，GPU
a) Multicore（多核）：在每个芯片上集成多个 CPU，在不提高时钟频率的情况下提高性能
b) MIC(Many Integrated Core，集成众核)：联合处理器的软件架构（每个芯片上的核数远多于 multicore，为了重复简单并行的任务设计）
c) GPU(Graphs Process Unit——图形处理单元)：专门在各种设备上做图像和图形相关运算工作的处理器

五、计算机性能评估
a) 时钟频率：单位时间内经过的指令周期的个数（时钟周期的倒数）
b) 处理器时间：处理器执行给定程序所需的时间

🔴 $T =$ 程序的CPU时钟周期 $*$ 时钟周期$\tau =$ 程序的CPU时钟周期/时钟速率

🔴 $T = CPI \times I_C \times \tau = CPI \times I_c$/时钟速率

CPI: 每条指令的平均周期 I_c: 指令数

受指令集体系结构、编译技术、处理器实现和内存层次结构的影响

c) 处理器速度：执行指令的速度，表示为每秒数百万条指令(MIPS)，称为 MIPS 速率：

$$MIPS\ rate = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

d) 基准测试程序：用高级语言定义的一组程序，在一个特定的应用程序或系统编程领域中，尝试提供一个计算机的代表性测试

六、Amdahl 定律

a) 表述：**对系统某部分加速时，其对系统整体影响取决于该部分重要性和加速程度。**（要想显著加速整个系统，必须提升全系统中相当大的部分的速度）

$$\text{设变量 } F_e = \frac{\text{能改进的部分}}{\text{整体}} \leq 1 \qquad S_e = \frac{\text{改进前的时间}}{\text{改进后的时间}} \geq 1$$

再设 T_0：任务在性能提升前的执行时间

改进后整个任务的执行时间为：

$$T_n = T_0(1 - F_e + \frac{F_e}{S_e})$$

加速比为：

$$S_n = \frac{T_0}{T_n} = \frac{1}{(1 - F_e) + \frac{F_e}{S_e}}$$

F_e对系统性能提升的限制作用很强

注意：加速比是$T_0 \div T_n$——加速比一定大于 1

第三章 计算机功能与互联

一、计算机的功能

a) CPU 是执行指令的组件

i. CPU 处理一条指令的时间称为指令周期

ii. 指令周期：

1. 取值周期：读取指令
2. 执行周期：执行指令
3. 间指周期：间接寻址找到指令
4. 中断周期：用来处理中断

b) 指令的读取与执行：在指令周期的开始，处理器会从内存中获取一条指令。而在一般的处理器中计数器 PC 保存了下一条要取的指令地址，没有特殊说明（跳转指令），PC 会增加，以便取下一条指令；而取到的指令会被加载进指令寄存器 IR 中，

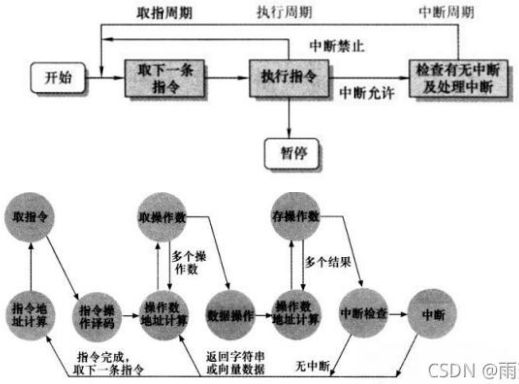
然后处理器会解析指令并执行所需的操作。

c) 中断机制(Interrupt Mechanism):

i. 概念：允许其他模块中断 CPU 执行序列的机制，从而更好地实现对紧急事件的处理

ii. 中断和指令周期：当外部设备准备好接收服务，外部设备的 I/O 模块发送中断请求信号给处理器。处理器通过挂起当前程序的操作，**跳转**服务于某个特定 I/O 设备的程序来响应，这个程序被称为**中断处理程序**，并且在设备服务完后恢复原来的执行。(注意：在中断驱动 I/O 中，中断服务程序负责处理中断信号、判断中断来源，并完成 I/O 操作的收尾工作（如设置设备状态标志、唤醒等待进程等），而真正的数据传输操作通常已经由设备或 DMA 控制器在中断前完成)

iii. 中断周期：在一条指令执行完成后，系统进入中断周期（处理器检查是否发生中断，如果没有，则进入下一条取指周期）；如果检测到有中断信号，挂起当前正在执行的程序（保存上下文——寄存器、堆栈指针、PC 中存储的下一条指令的地址），将 PC 指针设置为中断服务程序的首地址，引导 COU 执行中断服务程序。



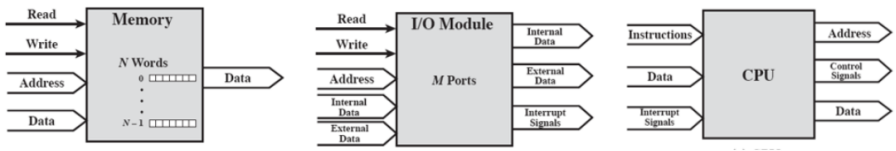
iv. 多重中断：在一个中断处理过程中（执行中断服务程序时），产生了新的中断，处理方法：

1. 禁止中断：在中断处理过程中，对新的中断加入等待队列，严格按照顺序处理
2. 定义优先级：定义中断的优先级，允许优先级高的中断引起低级中断处理程序本身被中断

二、计算机的互联结构

a) 互联结构：连接各个模块的通路

不同设备的主要输入输出所需信息交换的种类如下：



由上图可知，互连结构必须支持下列类型的传送

存储器到处理器、处理器到存储器、I/O到处理器、处理器到I/O、I/O与存储器之间(可以使用DMA直接存储器来实现)

b) 总线与总线结构:

i. 总线：是连接多个部件共享的信息传输线，是各部件共享的传输介质
ii. 总线传输的特点：某一时刻，只允许由一个部件向系统发送信息，而多个部件可以同时从总线上接受相同的信息。
iii. **总线总类**：系统总线、外围总线、内部总线
iv. **系统总线**：CPU、主存、I/0 设备(通过 I/0 接口)各大部件之间的信息传输线
1. 系统总线的分类：
a) 数据总线：传输各功能部件之间的数据信息（数据总线的宽度就是计算机 CPU 的位数(机器字长：CPU 内部一次能够处理的数据位数),eg:64 位的计算机其数据总线的宽度为 64 位）——双向
b) 地址总线：用来指出数据总线上的源数据或目的数据在主存单元的地址或 I/O 设备的地址（地址线的位数与存储单元的个数有关，地址总线的宽度决定了系统的最大容量）——单向
c) 控制总线：用来发出各种控制信号的传输线——单向
v. 总线的设计要素：
1. 总线的分类：

类型	仲裁方法	时序	总线宽度	数据传输类型
专用/复用	集中式/分布式	同步/异步	地址/数据	读/写/读-修改-写/写后读/块

2. 总线的仲裁：当多个设备同时申请使用总线传输信息时，因为总线在某一时刻只能有一个器件发送信息，所以由某种仲裁机制决定谁优先占用总线资源
a) 仲裁方式：
i. 集中式(centralized)：总线控制器（硬件）负责分配总线时间
ii. 分布式(distributed)：不存在中心仲裁器，每个请求总线的设备都具有仲裁能力，并通过协商或竞争的方式自行决定谁获得总线使用权。
3. 时序：总线上协调事件的方式
a) 同步时序：事件发生由时钟信号决定——在时钟的上升沿请求总线
b) 异步时序：事件的发生由上一个事件决定——上一个事件执行结束，马上发送下一条总线请求
c) 半同步时序

第四章 Cache

一、存储系统概述
a) 基本概念：
i. 字：存储器组织的基本单元（字长通常等于 CPU 位数——机器字长（数据总线的带宽、一个整数的数据位数））
ii. 传输单元：对于主存储器，这是指每次读出或写入存储器的位数（不一定是一个可寻址单元，也可能是块传输）
iii. 衡量存储器的三个指标：
1. 存取时间（延迟）：从地址传输给存储器的时刻到数据已经被存储或使用为止所花的时间（对于非随机存取，要包含定位存取地址的时间）

2. **存储周期时间**:这个概念主要用于随机存取存储器，它是存取时间加上下一次存取之前所需要的附加时间。这里附加时间用于瞬变的信号消失或数据破坏性读后的再生。需要注意，**存储周期时间是与系统总线有关，而不是与处理器相关。**
3. 传输率：数据传入或传出存储单元的速率

传输率:这是数据传入或传出存储单元的速率。对于随机存取存储器，它等于"1/周期时间"。而对于非随机存取存储器，有下列关系:

$$T_N = T_A + \frac{n}{R}$$

其中：T_N：读或写N位的平均时间

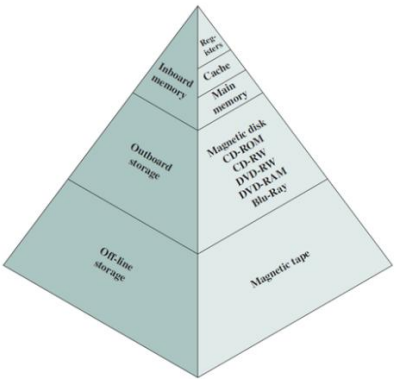
T_A：平均读取时间

n：位数

R：传输率，单位是b/s（位/秒）

b) 存储器的存取方式：
   i. Sequential(顺序存取)：从头开始，按顺序存取——访问时间取决于数据在存储器上的位置
   ii. Direct(直接存取)：每个块有唯一的地址，访问时通过跳转到附近然后顺序搜索（磁盘：访问时移动磁头到对应的扇区，在扇区内顺序搜索找到相应的字/字节）
   iii. Random(随机存取)：通过每一个地址精准定位到每一个可寻址单元（字/字节），访问时随机访问（RAM——Random Access Memery）
   iv. Associative(关联存取)：随机存储的一种

c) 存储器的层次结构——金字塔结构



二、Cache 存储器

a) Cache 中存储的是主存的部分副本，当 CPU 试图访问主存中的某个字时，首先检查这个字是否在 cache 中，如果是，则把这个字传送给 CPU;如果不是，则将**主存中包含这个字固定大小的块**读入 cache 中，然后再传送该字给 CPU（Cache 和 CPU 之间是字传输，Cache 和主存之间是块传输）
b) Cache 的原理：局部性原理
c) 主存储器有多达 2^n 个可寻址的字组成，每一个字都有唯一的 n 位地址，我们将主存看成许多定长的块，每个块有 K 个字，块数为 $M=2^n/K$
d) 而 cache 包含 m 个块，**称为行**，每行包括 K 个字和几位标记以及控制位。行的长度，不含标记和控制位，称为行大小。（Cache 的行大小等于主存的块大小）
e) 行的数量远远小于主存的块的数目。由于块数多于行数，所以单个行不可能永远的

被某个块专用，因此需要一个标记位 tag，这个通常是主存储器地址的一部分。(tag 位的计算：直接、全关联、组关联映射)

三、Cache 的设计

a) Cache 容量的选择：

i. 存储空间过小：便宜但是命中率低

ii. 存储空间过大：命中率高但是花费高、存取时间长，占用更多片上空间

b) Cache 的映射机制

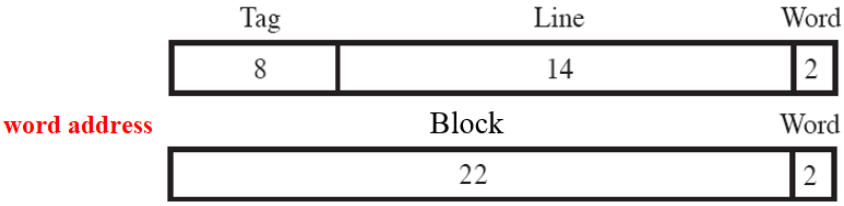
i. 直接映射：

1. 主存中的块 j 和 cache 中的行 i 有如下直接映射关系： $i=j \bmod m$，其中 m 为 cache 的行数
2. 存储器地址分为三部分：Tag、Line、Word

   其中：字号 Word 的长度由块大小决定(对块大小取对数——在某一块数据中地位到具体的字)

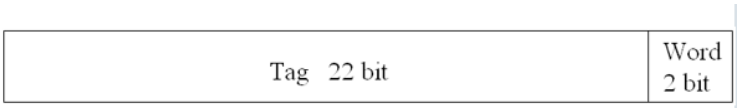
   行号 Line 的位数取决于 Cache 的行数，如 Cache 总共有 m 行，那么就需要 $\log_2 m$ 位在定位到具体的某一行

   存储器地址剩余的位数用来确定 tag 位（因为 Cache 的行数远小于主存的块数，一行可能装入不同的块，需要 tag 位定位到究竟是这些块中的哪一块）



ii. 全关联映射：

1. 允许每一个主存块装入 cache 中的任意行，此时只需要用标记位表示一个主存块。
2. 存储器地址分为两部分：Tag、Word
3. 为了确认某一块是否在 cache 中，需要对每一行中的标记进行搜寻检查



iii. 组关联映射：

1. 主存分成块，Cache 分成组，直接映射到组，全关联映射到组内具体的行

主存分成块、cache分成组，在组关联映射中，cache分为v个组，每组包含k个行，它们的关系为：$m = v \times k \; i = j \bmod v$

2. 其中：i为cache组号、j为主存块号、m为cache的行数、v为组数、k为每组中的行数

第五章 内部存储器

一、半导体存储器件

表 5-1 半导体存储器类型

存储器类型	种类	可擦除性	写机制	易失性
随机存取存储器（RAM）	读-写存储器	电可擦除，字节级	电	易失
只读存储器（ROM）	只读存储器	不可能	掩膜	非易失
可编程 ROM（PROM）			电	
可擦除 PROM（EPROM）	主要进行读操作的存储器	紫外线可擦除，芯片级		
电可擦除 PROM（EEPROM）		电可擦除，字节级		
快闪存储器		电可擦除，块级		

a) DRAM 和 SRAM

i. RAM(Random Access Memry)：随机访问存储器——通过编排的寻址逻辑，存储器的单个字直接被存取（具有易失性，一旦断电，就会丢失数据）

ii. DRAM(Dynamic Random Access Memery)：动态随机访问存储器——由电荷的有无表示 0/1——需要周期性充电刷新（访问速度较慢，常用作主存）

iii. SRAM(Static Random Access Memory)：静态随机访问存储器——不需要刷新操作（电路较复杂，费用高，访问快，常用作 Cache）

b) ROM(Read Only Memory)：只读存储器（掉电之后数据仍然存在）

i. PROM：可编程 ROM，可以通过特殊的操作完成程序写入（价格高）——只能写一次

ii. EPROM：光可擦 ROM，写入操作前，需要让芯片暴露在紫外线辐照下使所有的存储位元都被擦除，还原为初始状态，可重复进行，可多次修改，也可永久保存

iii. EEPROM：电可擦 ROM，只需要修改想要改的部分，不用全部都初始化。（以字为单位擦除）

c) Flash（闪存）：可以使用电可擦技术，擦除芯片上的某些模块（以块为单位进行擦除）

二、存储器封装

a) 半导体存储器也是封装的芯片（每个芯片的容量固定，若要满足不同的存储需求，需要进行不同的扩展）

b) 存储器扩展：

i. 位扩展：

1. 目标：增加数据总线的宽度（即每个存储单元的数据位数）

✅ **适用场景：**

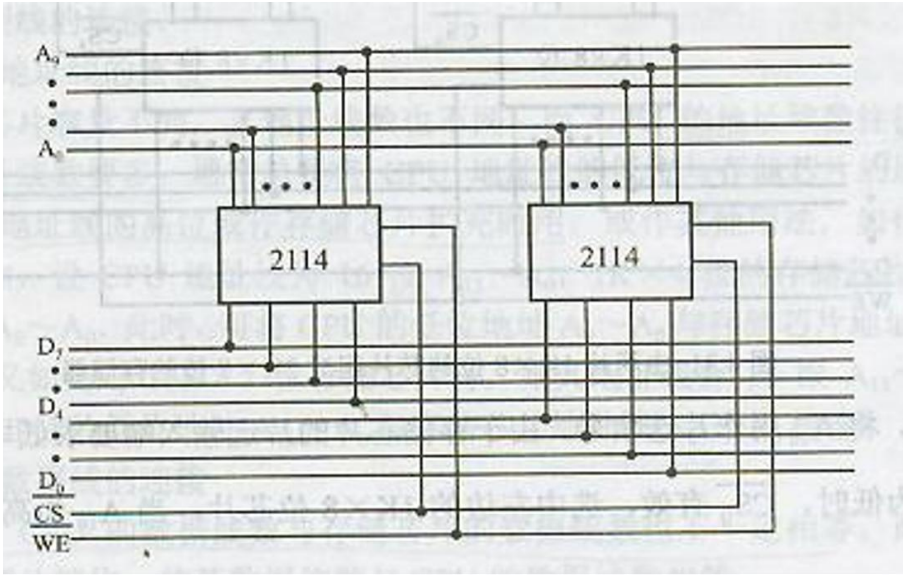
- 若某芯片是 8 位的（例如 1K×8），但系统要求每个地址返回 16 位数据；
- 就需要**横向扩展多个芯片**，让一个地址对应多个数据位。

✅ **举例：将两个 1K×8 扩展为 1K×16**

地址	芯片1 (8位)	芯片2 (8位)	合成
0	0x34	0x12	0x1234
1	...	...	...

- 同一个地址 A，两个芯片分别返回低 8 位和高 8 位；
- 控制信号同时打开两个芯片，输出组合为一个 16 位数据；
- 芯片选择相同，**地址线共用**，**数据线分别接高/低位。**

2. 实现：地址线共用，数据线串联（不同芯片的同一地址分别提供不同位的数据）



ii. 字扩展：

1. 目标：扩大可寻址单元的数量（即扩展地址空间）

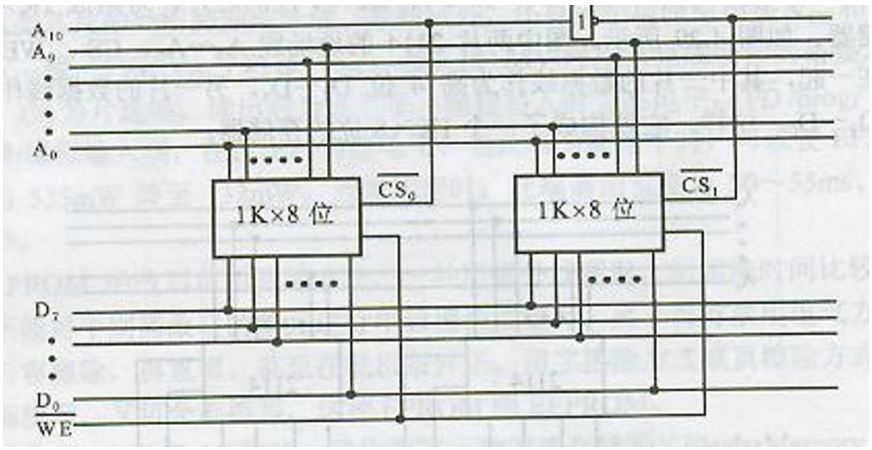
✅ **适用场景：**

- 每个芯片大小为 1K×8（1024个地址），但我们需要 4K×8 的存储；
- 就需要**纵向堆叠多个芯片**，用**不同地址段**选中不同芯片。

✅ **举例：用 4 个 1K×8 扩展为 4K×8**

- 每个芯片负责 1K（2^{10}）地址；
- 共需要 12 位地址线（$2^{12} = 4096$）；
  - A0–A9 给芯片地址输入；
  - A10–A11 用作片选地址判定，控制不同芯片的 CS（Chip Select）；

2. 实现：地址线上多一个译码器（选择低地址空间 or 高地址空间），数据线共用



iii. 混合扩展（既有字扩展又有位扩展）

第六章 输入输出

一、外设

a) 分类：
   i. 人类可读：屏幕、打印机、键盘
   ii. 机器可读：磁盘、设备控制器
   iii. 通信：网络接口
b) 磁盘驱动器：一是与 I/O 模块交换数据、控制和状态信号，二是用于控制磁盘的读/写机制

二、IO 模块

a) IO 模块的功能：
   i. 控制与定时
   ii. 通信：
      1. CPU 与 IO 模块通信
      2. IO 模块与 IO 驱动器通信
      3. 主存与 IO 驱动器通信（在 DMA 时）
   iii. 数据缓冲：更改码制为存储器能够识别的、积攒数据后一次性交给 CPU
   iv. 数据校验
b) 数据从外设到 CPU 的过程：
   i. 处理器查询外设的连接状态
   ii. IO 模块返回设备状态
   iii. 如果设备状态正常，CPU 通过 IO 模块外设发出命令，请求数据
   iv. 外设将数据按位传输到 IO 模块的缓存中
   v. IO 模块将数据传输给 CPU（或主存），具体方式：
      1. CPU 轮询（按字或字节传输）
      2. 中断驱动（按块传输）
      3. DMA（按块传输）
c) IO 通信操作：
   i. **命令译码 command decoding**: I/O 模块接受来自处理器的命令，这些命令一般作为信号发送到控制总线。例如，一个用于磁盘驱动器的 I/O 模块，可能接受 READ SECTOR (读扇区)、WRITE、SECTOR (写扇区)、SEEK (寻道)磁道号和 SCAN (扫描)记录标识等命令。后两条命令中的每条都包含一个发送到数据总线上的参数。
   ii. **数据交换 data exchange**：数据是在处理器和 I/O 模块间经由**数据总线来交换的**。外设向 CPU 发送数据时，数据通常先存储在 I/O 模块的 Buffer 中，凑满之后一次性传输出去。
   iii. **状态报告**：由于外设速度很慢，所以知道 I/O 模块的状态很重要。例如，如果要求一个 I/O 模块发送数据到处理器(读操作),而该 I/O 模块仍在处理先前的 I/O 命令而对此请求未能就绪，则可以用状态信号来报告这个事实。常用的状态信号有忙(BUSY)和就绪(READY)，还有报告各种出错情况的信号。
   iv. **地址识别**：正如存储器中每个字对应一个地址一样，每个 I/O 设备也有地址。这个地址 CPU 不会记录，由 I/O 模块记录。因此，I/O 模块必须能识别它所控制的每个外设的唯一地址 。另一方面，模块必须能进行设备通信

(devicecommunication)

d) 数据缓冲：用于协调 CPU/主存和外设间速度不匹配的问题。

e) 一个 I/O 模块可以连接一个或多个 I/O 设备，I/O 设备的地址由 I/O 模块识别

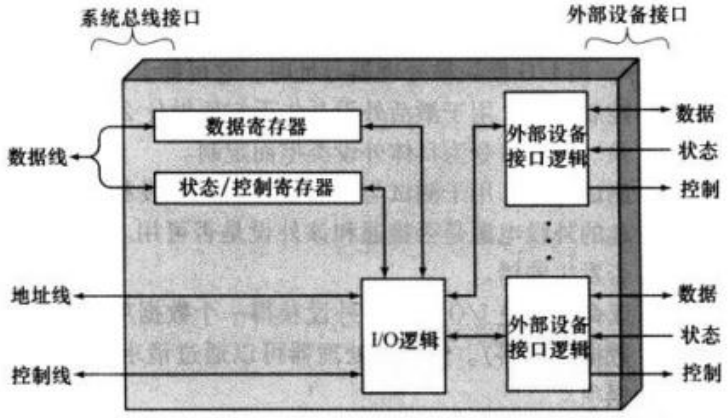


图 7-3 I/O 模块框图

f) IO 的编址方式：

i. 存储映射式编址(Memory Mapped)：将 IO 设备看成内存空间的一部分，和存储器统一编址，通过地址码区分存储器和 IO 接口（IO 不需要很多地址，但是 IO 设备的编码和存储器的地址长度相同）

ii. 独立式编址(isolated)：存储器和 I/O 分为单独的地址空间，有独立的 IO 指令，CPU 通过区分指令来区分访问 IO 还是内存

三、IO 模块决策：

a) 程序查询式 I/O：CPU 完全掌握 I/O 过程，需要事无巨细的控制 I/O 传输的每个细节（按字或字节传输）。

b) 中断驱动的 I/O：发送传输命令后，CPU 就不管 I/O 了，等传输完成后 I/O 模块通过中断通知 CPU（按字或字节传输）

c) 直接内存访问（DMA）：I/O 一次完成多于一字节的单位（比如一次传输一块）后，再通知 CPU（以块为传输单位）

第九章 计算机算数

一、算数逻辑单元(ALU):

1.ALU 是计算机是计算机实际逻辑运算的部件

2. 数据由寄存器(DR)提交给 ALU，运算结果也存放于 ALU，控制器提供控制 ALU 操作和数据传入送出 ALU 的信号；ALU 根据运算结果会设置一些标志(flag)

二、整数的表示

c) 无符号数表示：$A=\sum_{i=0}^{n-1}2^{i}a_i$

d) 符号-幅值表示（原码）:

i. 最左边的位是符号位：0 代表正数，1 代表负数

e) 补码：整数的补码=无符号数表示；负数的补码=绝对值的补码表示按位取反再+1

三、整数运算

a) 整数加减法:

i. 溢出：**若两个正数或两个负数相加，当且仅当结果的符号位变反时，发生溢出**

ii. 减法：对减数取负，与被减数进行加法运算

四、浮点数的表示

a) 表示原理:

$$\pm S\times B^{\pm E}$$

符号位：0正，1负

B：指数的底或基是隐含的，二进制数基为2不需要存储

S：有效数,小数点位置为最高位有效位的右边，即小数点左边有1位有效数。对于二进制数，有效数的小数点前只能是1，所以从小数点后开始存储有效数以节省存储空间

E：指数或者阶

实际上使用了移码

$$\pm S\times B^{E'}$$

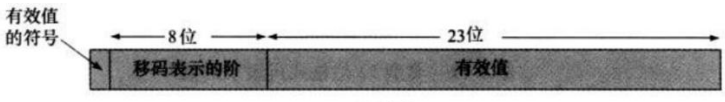
存储的指数值E'为移码，是无符号数,存储的指数值 = 真正的指数 + 偏移量(目的：正数方便比较)

通常，偏移量等于$2^{k-1}-1$,k：二进制指数的位数

一个8位字段，取偏移量为127

真实指数的范围是 [-127，128]

存储指数的范围是 [0，255]



a) 格式

$1.1010001\times 2^{10100}=0\ 10010011\ 10100010000000000000000=1.6328125\times 2^{20}$

$-1.1010001\times 2^{10100}=1\ 10010011\ 10100010000000000000000=-1.6328125\times 2^{20}$

$1.1010001\times 2^{-10100}=0\ 01101011\ 10100010000000000000000=1.6328125\times 2^{-20}$

$-1.1010001\times 2^{-10100}=1\ 01101011\ 10100010000000000000000=-1.6328125\times 2^{-20}$

b) 举例

图 9-18 典型的 32 位浮点格式

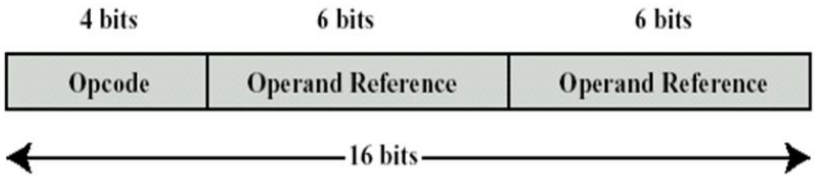
第十章 指令集：特征与功能

一、机器指令特征

a) CPU 的操作由它所执行的指令确定，这些指令称为机器指令 machine instruction；
b) CPU 能执行的各种不同指令的集合称为 CPU 的指令集 instruction set
c) 机器指令的要素：
   i. 操作码(operation code)：将要完成的操作
   ii. 源操作数引用(source operand referance)：操作所需要的输入
   iii. 结果操作数引用(result operand reference)：操作长生的结果
   iv. 下一条指令引用(next instruction reference)：执行当前指令后取下一条

10.1.2 指令表示

对于机器码，每一条指令有一个唯一的位串，格式如下：



为便于使用，普遍使用机器指令符号表示法

操作码可以用助记符表示，如 `ADD：加` `SUB：乘` 操作数也可以用符号表示，如 `ADD R,Y`

d) 指令类型：
   i. 数据处理
   ii. 数据存储
   iii. 数据传输
   iv. 控制（测试和分支指令）
e) 指令格式：
   i. 四地址指令：操作码+两个源操作数地址+一个运算结果保存地址+一个下一条指令地址
   ii. 三地址指令：操作码+两个源操作数地址+一个运算结果保存地址（隐藏下一条指令地址）
   iii. 二地址指令：操作码+两个源操作数地址（一个地址重用为源操作数地址和目标操作数地址）

二、指令的操作

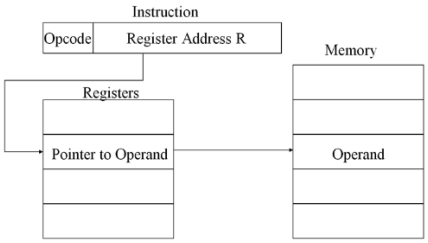
a) 数据传送(Data Transfer)：
   i. 指明源和目的操作数的位置
   ii. 指明将要传送数据的长度
   iii. 为每个操作数指明寻址方式
b) 算数运算
c) 逻辑运算
d) 转换：转换数据格式
e) 输入输出

f) 系统控制
g) 控制转移（跳跃指令）

第十一章 寻址方式和指令格式

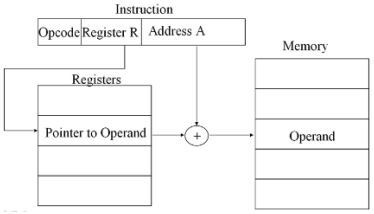
一、寻址方式：确定当前指令或下一条指令中数据地址的方法

a) 具体的寻址方式取决于 CPU 硬件以及指令的格式
b) 寻址技术：
   i. 立即寻址：操作数就是指令的一部分（对地址操作的指令），不需要内存访问
   ii. 直接寻址：地址字段包好操作数的有效地址，直接进行内存访问，不需要额外的地址计算（地址位的长度限制了可表示的地址空间有限）
   iii. 间接寻址：指令中的地址字段没有直接给出操作数的地址，而是指出了有效地址所在的存储单元的地址（先根据指令给出的地址找到存储有效地址的存储单元，再根据找到的存储单元的内容——有效地址，定位到有效地址，找到真正的操作数）——这种做法提供了更大的地址空间，缺点是需要多次访存
   iv. 寄存器寻址：指令中的地址字段给出的是寄存器的编号，可以直接取该寄存器中存储的数据（访问快速，常用于定义寄存器变量）
   v. 寄存器间接寻址：地址字段给出的是寄存器编号，操作数的有效地址存储在寄存器中



   vi. 偏移寻址：通过**基地址+偏移量**实现内存地址的计算

1. 指令要求有两个地址（基地址和偏移量，至少有一个是显式的）



2. 偏移寻址的分类：
   a) 相对寻址：EA=A+(PC)：PC 指向当前位置，A 是位移，常用于实现跳转指令
   b) 基址寄存器寻址：常用于分页或分段后物理帧的查找
   c) 变址：基址寄存器和偏移寄存器
   vii. 栈寻址：操作数放在栈顶

总结

方式	算法	主要优点	主要缺点	操作数位置	应用
立即寻址 **Immediate Addressing**	操作数=A	无存储器访问，速度更快	操作数范围有限	操作数在指令中	定义和使用常数
直接寻址 direct addressing	有效地址（EA）=地址字段（A）	只要求一次存储器访问无需额外计算就可得出有效地址	受限于地址字段长度，只能提供有限的地址空间	操作数在地址地段指向的地址中	
间接寻址 indirect addressing	EA=（A）	更大的地址空间	速度较慢	操作数在地址地段指向的存储器中	中断服务程序入口寻找
寄存器寻址	EA=R	无存储器访问	地址范围有限	操作数在地址地段指向的地址中	定义常用变量
偏移寻址 displacement addressing	EA=A+(R)	灵活	复杂	操作数在基地址和偏移间接结合指向的地址中	分页或分段
栈寻址stack	EA=栈顶	无存储器访问	应用有限		

二、指令格式

a) 指令长度

i. 影响因素：内存大小、内存组织、总线结构、CPU 复杂性、CPU 速度

ii. 指令长度应当等于数据总线宽度或其整数倍数、指令长度应当等于字符长度或定点数长度的整数倍

第十二章 CPU 结构与功能

一、CPU 的组成部分

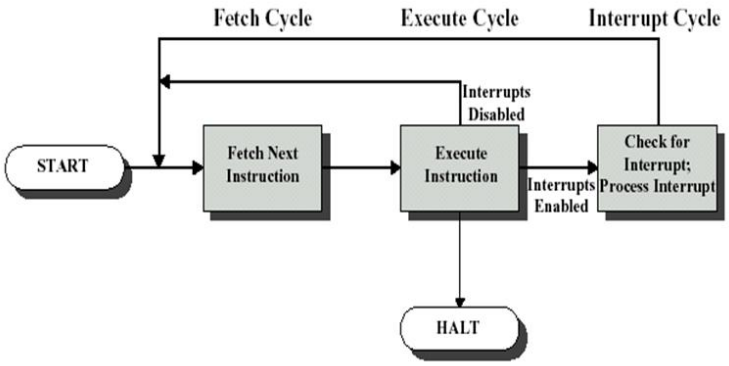
1. CPU 的组成部分：ALU、CU、寄存器
2. 指令执行的过程
   i. 取指令
   ii. 解释指令（指令的译码）
   iii. 取操作数
   iv. 处理数据
   v. 写回数据
3. 寄存器的作用：协调 ALU 这个组合逻辑电路与其他时序电路的矛盾，临时存放一些数据（将要处理的数据或者下一条指令的位置、临时存放 ALU 计算出来的结果）

二、 寄存器组织

1. 寄存器：CPU 中专门用于暂存数据的一小块空间（最贵、访问速度最快）
2. 寄存器分类：
   i. 用户可见寄存器(user-visible register)：允许编程人员使用的寄存器（定义的寄存器变量）——减少对内存的访问
      a) 通用目的寄存器：暂存操作数和结果
      b) 数据寄存器：用于存放参与特定运算的操作数（整数、浮点数）
      c) 地址寄存器：用于存放内存地址（堆栈指针、全局变量）
      d) 条件码寄存器：用于存放最近一次逻辑运算的结果，作为条件转移指令是否跳转的判断依据
   ii. 控制与状态寄存器(Control and status registers)：对用户透明，有 CPU 或操作系统控制使用（用于控制或表征 CPU 状态，用户程序无权直接访问）
      a) 程序计数器 PC(Program Counter)：存放下一条指令的地址
      b) 指令寄存器 IR(Instruction Register)：存放最近取来的指令
      c) 存储器地址寄存器 MAR(Memory Address Register)：将要读/写的存储器的地址
      d) 存储器缓冲寄存器 MBR(Memory Buffer Register)：将要被写入存储器的数据或是刚刚从存储器读来的数据
      e) 程序状态字 PSW(Program Status Word)：一个或一组寄存器，有时会包含条件码寄存器，包含程序当前的一些状态信息（符号、进位、溢出、中断）

三、指令周期

1. 定义：指令周期的是完成一条指令所要经历的全部过程
2. 指令周期的内容：
   a) 取指周期：获取指令、指令译码 （必须）
   b) 间址周期：如果涉及到间接寻址，这是就需要在间指周期内找到有效地址及其真实的操作数（非必须）
   c) 执行周期：真正完成指令的执行（必须）
   d) 中断周期：在一条指令执行结束后，检查是否发生中断，如果有中断，则引导 PC 指向中断服务程序的首地址（非必须）



四、 指令流水线

1. 指令预取：可以在译码和执行当前指令时获取下一条指令
2. 指令流水线：将一个指令指令周期划分为六个阶段：
   a) 取指令 FI(Fetch Instruction)：从内存中读取一条指令
   b) 译码指令 DI(Decode Instruction)：指令译码
   c) 计算操作数 CO(Calculate Operand)：计算源操作数的有效地址
   d) 取操作数 FO(Fetch Operand)：从存储器中取出操作数，放在数据寄存器中
   e) 执行指令 EI(Execute Instruction)：执行制定的操作
   f) 写操作数 WO(Write Operand)：将结果保存到内存中

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

即使某些指令不需要六个流水段，也需要走一次对应的段消磨时间，不能跳过某个指令的某个执行阶段。

   g) 流水线性能：
      （1） **流水线的周期长度**指的是流水线中**每一个阶段统一使用的时钟周期长度**。这个时钟周期由**流水线中最慢的阶段决定**。
      （2） CPU 的时钟周期长度 = 流水线的周期长度

   h) 流水线冲突：
      （1） 资源冲突（结构冲突）(resource hazard)：两条（或多条）已进入流水线的指令需要使用相同资源（指令在访问相关资源时，必须按照串行顺序执行，不能使用流水线并行进行了）

(2) 数据冲突(data hazard)：对一个操作数位置的访问出现冲突（解决方法：解决方式：流水线气泡。等待上一条指令完成写入，在此之前一直等待，什么都不做。）

(3) 控制冲突(control hazard)：流水线对转移指令预测错误时，就会出现控制冲突

处理转移预测错误的五大方法：

处理转移预测错误的五大方法：

- 多指令流
- 预取转移目标
- 循环缓冲
- 转移预测
- 延迟转移

a) **多指令流**：保留两条指令流水线，第二条流水线仅仅在遇到分支语句时启动，把每个转移目标放到一个流水线中执行，这样就可以并行执行两个转移目标的语句。

b) **预取转移目标**：前面提到，预测错误时最怕的是错误的访问了内存或者外存，因为访存花费的时间非常长。预取转移目标就是在译码一条指令，得知它是转移指令后，先把转移指令目标位置的指令预取来，放到寄存器中。

c) **循环缓冲**：增加一个小的缓冲区，存放最近访问到的 n 条机器指令，如果发生了转移，CPU 在访存寻找指令前，先在缓冲区中搜索指令（**局部性原理**）。

d) **转移预测**：在遇到条件转移指令，在转移前，先预测最可能进入的分支。

i. 预测转移永不发生(如果取下一条指令会触发页故障（缺页），则暂停执行，直到条件转移指令执行完毕，得到是否要转移的信息，防止无效的访问外存)

ii. 预测转移一定发生

iii. 依据操作码预测：有些指令会大概率引发跳转，对这种指令就认为它一定会跳转（比如调用函数，函数返回）；有的指令只有小概率引发跳转，这些指令就认为永不会跳转。

iv. 转移-不转移切换：根据之前执行转移指令时，转移的多还是不转移的多，对下一次转移指令做预判

v. 根据转移历史表预测：建立一个 cache（称为转移历史表，BTB/BHT），根据记录的信息，决定是否转移

e) **延迟转移**：插入 NULL 操作占据时拍，不让流水线提前执行，访问内存或者外存。相当于什么都没做，浪费了时间

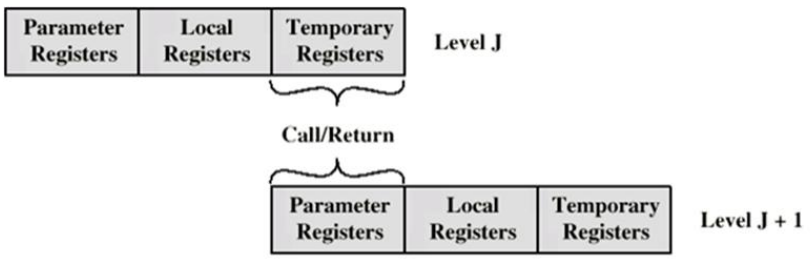
第十三章　精简指令集计算机(RISC)

一、指令执行的特征

1. 执行的操作：
    a) 赋值语句在程序中出现的频率最高
    b) 条件语句出现的频率也比较高
    c) 过程调用/返回操作的耗时最多
2. 操作数：主要是简单的标量变量，很大一部分是局部标量变量（优化的方向主要是对局部标量变量进行快速的访问和存取）
3. 过程调用：过程调用所设计的数据传递的量不大，绝大多数都是对局部变量的访问
4. 结论：优化最常用、最耗时的操作才能更好地支持高级计算机语言

二、大寄存器组(large register file)

1. 寄存器的特点：
    a) 速度快（速度快于主存和 Cache）
    b)地址短（寄存器的数量少，地址编码所需要的位数少）
    c)更靠近 CPU
2. 计算机中寄存器的分配：
    a)软件解决：依靠编译器分配寄存器，将寄存器分配给那些在给定时间内使用最多的变量
    b)硬件解决：配置更多的寄存器。使得更多的变量存储在寄存器中
3. 寄存器窗口：
    a) 使用一大组寄存器——很好的减少对内存的访问需求
    b) 寄存器中对局部变量的存储：将局部标量变量存储在寄存器中，以减小内存访问；每次过程（函数）调用都会改变“局部变量”具体所指；原有过程的局部变量必须被送到存储器（栈），以腾出寄存器空间供现有正在执行的过程的局部变量来使用
    c) 寄存器窗口的使用：使用多个小的寄存器组，每个小组指派给一个不同的过程（父/子今年成）。过程调用时自动地切换来使用**不同的但大小固定**的寄存器窗口，而不再在存储器保存寄存器内容，相邻过程的窗口是（部分）重叠的，以允许参数传递



4. 环形缓冲组织：将所有的寄存器组按照环形的方式组织起来（指针首尾相接），在程序调用过程中，一旦发现所有的窗口都在使用，则会产生一个中断，并将最早的窗口交换到内存中（N 个窗口仅能用于 N-1 个过程的直接调用）
5. 全局变量：给全局变量单独分配一个寄存器窗口（独立于环形寄存器，但是和环形窗口寄存器统一编址）

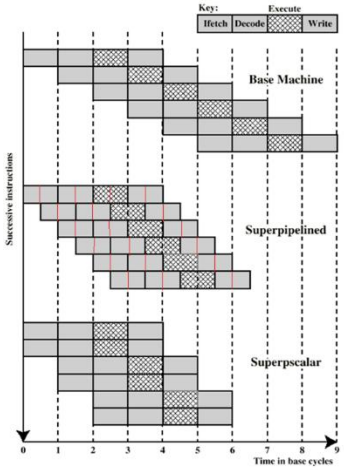
三、基于编译器的寄存器优化

1. 寄存器优化的目的：**尽可能在寄存器而不是在主存中为多数计算机保持操作数，并且减少装载和保存操作(访存)**
2. 优化的方式：**使用时间上不重叠的符号寄存器可共享同一物理寄存器，**如果某一时段物理寄存器用尽，则某些变量仍需放回到存储，如果某一时段物理寄存器用尽，则某些变量仍需放回到存储器
3. 优化任务的本质——判定在程序的任何给定时间点，什么样的量应指派给寄存器中——**图着色法**
4. **图着色法:**节点代表符号寄存器(逻辑寄存器),若两个符号寄存器的使用时间有重合部分,则这两个节点之间有一条边,尝试用 n 中颜色给图书上色,(n 是物理寄存器的个数),如果问题无法成功解决,则不能上色的寄存器内容必须保存到内存中.

第十四章 指令集并行性和超标量处理

一 概述

1. 超标量(superscalar)方法的本质:在不同的流水线中并行地执行指令(允许指令能以不同于原来程序顺序的次序执行)
2. 超标量是指令在不同流水上并行地执行,每个流水线都有若干个功能单元,都可以进行指令流水,是真正意义上的并行
3. 超标量与超流水:
   a) 超流水:将流水线上的任务划分的更细(在一个流水线周期内完成两个任务,一个流水段同步于外部一个更快的时钟)



4. 超标量的限制因素:
   a) 真实数据相关性:先写后读的相关性(存在同步关系的两条指令不能使用超标量技术并行)
   b) 过程相关性:
      (1) 由于分支跳转指令的存在,使得后续指令是否应该被发射变得不确定
      (2) 优于变长指令的存在,使得取指令时对下一条指令的定位变得比较复杂
   c) 资源冲突:多个指令(流水线)竞争使用同一资源(存储器)------解决方案:复制资源
   d) 输出相关性:在程序中有先后顺序的两条指令对同一个变量进行操作,选择使用超标量可能破坏原来的时序关系(写写相关)------解决方案:寄存器重命名
   e) 反相关性:先读后写相关性

二 超标量设计

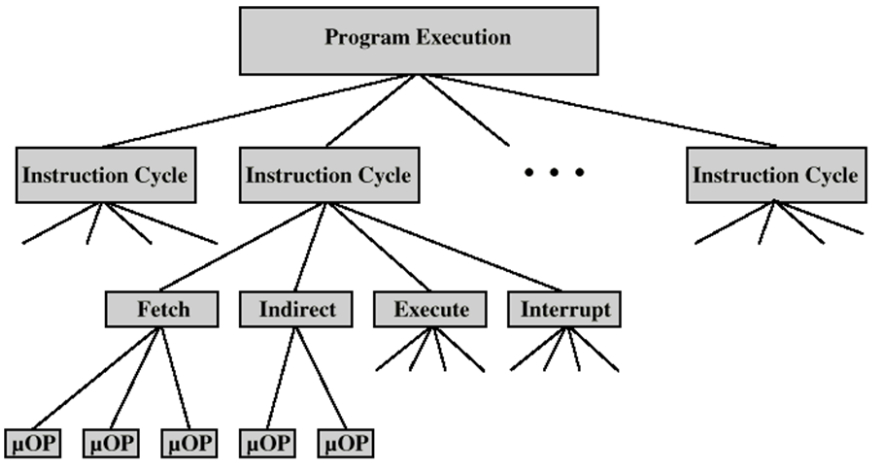
1. 指令发出(instruction issue)策略:
   a) 按序发出按序完成: 严格的按顺序执行顺序发出指令, 并以同样的顺序写结果
   b) 按序发出乱序完成
   c) 乱序发出乱序完成:将译码部件与执行部件解耦合(加入一个窗口缓存已经译码的指令,当检测到有可用的执行部件时,随机从窗口中抽取一条已经译码的指令且没有冲突或阻塞进行执行)---------可以解决真实数据相关,过程相关,资源冲突的问题.
2. 寄存器重命名:针对输出相关和反相关,可以采用寄存器重命名的方法解决: 将不同指令使用的同名寄存器映射到不同的物理寄存器上去.
3. 超标量中提高性能的三种技术:资源复制,乱序发出,寄存器重命名.

第十五章控制单元操作

控制器负责向处理器外部发出控制信号，从而控制与存储器或 I/0 模块间的数据交换;控制器还需要向处理器内部发送控制信号，以在寄存器间移动数据，并引发 ALU 完成指定功能以及其他内部调整操作。控制器的输入包括指令寄存器、标志和来自外部的控制信号(例如中断信号等)。

一、微操作(micro-operation)

1. 定义：每个指令周期（取指周期、执行周期）都是由若干个子周期构成的，每个子周期又包含若干个小步骤(steps)，这些 step 就被称为微操作，微操作是处理器的原子操作(atomic-operation)



2. 取指周期涉及的微操作:
   (1) 从 PC 中取出将要执行指令的地址，放在 MAR 中
   (2) 将 MAR 上的地址送到地址总线
   (3) 控制单元 CU 向内存发送读命令，内存读取数据总线上的内容
   (4) 内存将读到的内容放在数据总线上，将数据送入 MBR
   (5) 将 MBR 中的指令送入 IR 中，清空 MBR，为下一次数据传送做准备
   (6) 取值结束后 PC+1

- t1: MAR ← (PC)
- t2: R ←1, MBR ← (memory)
  PC ← (PC) +I
- t3: IR ← (MBR)

- t1: MAR ← (PC)
- t2: R ←1, MBR ← (memory)
- t3: IR ← (MBR)
  PC ← (PC) +I

(tx = 时间单位/时钟周期，一个时钟脉冲)

3. 间接寻址周期涉及的微操作
   (1) 从 IR 的低 n 位获得操作数的逻辑地址，将这个逻辑地址送入 MAR 中
   (2) 将 MAR 的地址放在地址总线上
   (3) CU 发送内存读命令，内存从地址总线上获得逻辑地址
   (4) 内存将逻辑地址单元内存储的真实地址放在数据线上，送进 MBR
   (5) 将 MBR 中存储的真实地址更新到 IR 的地址位，清空 MBR
   原来 IR 的地址位存储的简介地址，经过间址周期，IR 地址位存的是真实地址

4. 执行周期涉及的微操作
   （1） 将 IR 中的地址位（存储真实数据的物理地址）放进 MAR 中
   （2） 将 MAR 的内容放在地址线上
   （3） CU 发送内存读命令，内存根据数据物理地址找到真实操作数
   （4） 内存将操作数放在数据总线上，将数据送入 MBR 中
   （5） 最后还要将 MBR 中的数据送到用户可见的通用寄存器（数据寄存器）上。
5. 中断周期涉及的微操作
   （1） 将 PC 指针的内容写入到 MBR 中
   （2） 将保存断点的内存地址写入 MAR 中
   （3） 根据 MAR 中的地址，将 MBR 中的内容写入到对应的存储空间
   （4） 将中断服务程序的首地址写入到 PC 中
   后面接取指周期

6. 指令周期：为区分 CPU 处于哪个周期，设置有 2 位的寄存器，称为指令周期代码 **Instruction Cycle Code （ICC）**

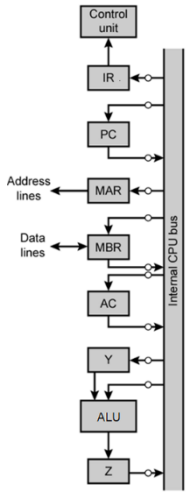
二、处理器控制

控制器主要完成亮相基本任务：

（1） 排序(sequencing)：根据正被执行的程序，控制器使 CPU 以恰当的顺序一步步完成一系列微作
（2） 执行(execution)：控制器使每个微作得以完成

1. 控制器的控制信号（控制线路的通断）：
   （1） 输入信号：时钟、各种标志(flag)、指令寄存器(IR)、来自控制总线的控制信号
   （2） 输出信号：CPU 的控制信号、对总线的控制信号
2. 处理器内部组织
   ALU 是一个组合逻辑电路（一有输入就会产生结果），无法满足系统的时序需求，一次需要两个新增寄存器(Y 寄存器和 Z 寄存器)分别存储 ALU 的输入和输出，以此来满足系统的时序要求

t_1：MAR←(IR(地址))
t_2：MBR←内存
t_3：Y←(MBR)
t_4：Z←(AC) + (Y)
t_5：AC←(Z)



三、硬连线实现

控制器的实现分为两种：硬连线式和微程序式实现

1. 硬连线式实现中，控制器必须是组合逻辑电路
2. 控制器输入：
   (1) 标志与控制总线信号——每一位都有特定的含义
   (2) 指令寄存器：不同指令的操作码引起控制器发出不同的控制信号，从而完成不同的操作
   (3) 时钟信号：系统总时钟信号，保证控制信号的有序发送

第十六章 微程序控制

一、微指令

1. 微程序控制语言(microprogramming language)：指定微操作的语言，每行描述一个时间内出现的一组微操作，成为一条微指令（**微指令的每一位基本都有明确的控制意义**，这些控制信号最终驱动 CPU 内部各部件协调工作）
2. 微程序（固件）：一组微指令的集合，决定了机器指令在 CPU 中的具体执行过程
3. 控制字(control word)：微指令的二进制表示，用于控制每根控制线的开关
4. 微指令的类型：
   (1) 垂直微指令：每个微指令指定要执行的单个微操作（并行能力有限）
   (2) 水平微指令：每个微指令指定要并行执行的多个微操作——每个控制信号都占据控制字中的**独立一位（或字段）**，控制信号之间**彼此独立、可同时激活**
5. 微程序控制器：
   (1) 微程序控制器的组成：
      a) 控制存储器：存有一组微指令
      b) 控制地址寄存器：存储下一条微指令的地址
      c) 控制缓冲寄存器：存放从控制存储器中取来的微指令，准备发送控制信号

第十七章 并行处理

一、多处理器组织

1. 单指令单数据(Single Instruction Single Dat——SISD)：单一处理器执行单一指令流来操作保存于单一存储器上的数据。
2. 单指令多数据流(SIMD)：一条机器指令控制几个处理部件对不同数据进行操作，每个处理部件有一个相关的数据存储器
3. 多指令多数据流(MISD)：一系列数据被发送到一组处理器，每个处理器执行不同的指令序列。这种结构从来没有在商业上被实现过。
4. 多指令多数据流(MIMD)：一组处理器同时执行不同的指令，对不同的数据集进行操作

二 对称多处理器(Symmetric Multiprocessor)

1. 定义：有两个或者更多功能相似的处理器
2. SMP 的处理器共享一个主存和 IO 设备，通过总线互联在一起（紧耦合）
3. 系统被一个集中式操作系统控制，由 OS 控制不同作业之间的数据交互