考试题型

总复习

一.计算机系统概述（约5分）

- 计算机硬件的基本组成
- 计算机软件的分类

  应用软件：专门为某种应用编写的程序（如电子邮件收发软件、视频播放软件、 文字处理软件等）

  系统软件：为有效、安全地使用和管理计算机以及为开发和运行应用软件而提供 的各类软件；介于计算机硬件与应用软件之间
- 计算机系统的抽象层及其转换
- 用户CPU时间计算 (选择题)

  CPI：执行一条指令所需的时钟周期数

  指令速度计量单位 **MIPS** : 平均每秒钟执行多少**百万条**指令(记得除以10^6)
- Amdahl定律(选择填空)

二.数据的机器级表示（约10分）

- 二进制、八进制、十六进制、十进制数之间的转换
- 原码、补码、移码表示法
- 无符号整数、带符号整数表示
- 浮点数的表示？IEEE 754浮点数表示？ （若涉及特殊情况，会给出提示或解释）

  特殊序列：
- C语言中的整数、浮点数类型 (**转换出现问题**)(数据大小端)
- 数据的存储和排列顺序

  存储器按字节编址:按边界对齐

  存储器不按字节编制:不按边界对齐

三.运算方法和运算部件（约10分）

- 理解并能写出常见的汇编指令，尤其是算术运算、逻辑运算涉及的指令
- 串行、并行、带标志的加法器原理

  全加器原理
- 补码加减运算（原码乘除、补码乘除：若涉及，会给出计算规则）
- 乘除运算溢出判断，常量乘除运算

  X*Y的高n位可以用来判断溢出，规则如下：

  变量与常数之间的除运算
- 浮点数加减运算、舍入方式

  浮点数加减步骤

四.指令系统（约15分）

- 指令格式设计

  零地址指令: op

  一地址指令: op | A1

  二地址指令: op | A2 | A2

  三地址指令（RISC风格）: op | A1 | A2 | A3
- 指令执行周期
- 操作数类型、寻址方式、操作类型、操作码编码

  偏移寻址方式

  1.相对寻址

  2.基址寻址

  3.变址寻址
- 异常和中断的区别

  中断分为两种：

  异常、中断的处理
- MIPS指令的格式和寻址方式MIPS指令的格式和寻址方式
- 选择结构、循环结构的汇编指令表示 （若涉及机器代码，会给出指令格式和机器代码之间的关系）
- 过程调用的指令、执行步骤、栈和栈帧的变化(图)(例题)

  执行步骤(P调用Q)

  MIPS中的栈

  MIPS寄存器的使用约定

五.中央处理器（约15分）

- CPU执行指令过程
- CPU的基本组成？操作元件和状态元件的区别

  CPU的组成

  关于 **操作元件**

  关于 **状态元件**

- 指令周期、时钟周期
- MIPS指令格式，典型的MIPS指令以及功能描述（书本134-135页）
- 典型的MIPS指令数据通路？能够理解单周期数据通路设计，画出局部数据通路,明确控制信号取值。
- 多周期数据通路设计？能够理解多周期数据通路设计，**明确控制信号取值**(一定要知道)、指令执行状态转换图。
- 微程序控制器的基本思想、基本结构、执行、编码方式
  - 设计思路
  - 结构
  -)微指令的种类
  - 微指令的内容
  - 微操作码的编码方式
- 带异常处理的数据通路、有限状态机
  - MIPS处理器的异常处理

六.指令流水线（约15分）

- 指令流水线由哪些流水段组成、各流水段的功能？使用哪些部件？
  - **取指令(IF)**：从存储器取指令； **指令存储器、Adder**
  - **指令译码(ID)**: 产生指令执行所需的控制信号；**寄存器堆读口、指令译码器**
  - **取操作数(OF)**: 读取操作数； **扩展器、ALU**
  - **执行(EX)**: 对操作数完成指定操作； **数据存储器**
  - **写回(WB)**: 将结果写回; **寄存器堆写口**
- 典型MIPS指令的功能段划分、流水线数据通路的设计、控制信号的取值（能够理解流水段寄存器，理解流水线数据通路，但不要求自己画出数据通路）
  - 各指令对齐后的结果：
  - 各个阶段的控制信号如下：
- 结构冒险现象及其解决方法
- 数据冒险现象及其解决方法（需要深入理解转发技术、load-use数据冒险的检测和处理方法）
  - 数据冒险
  - 数据冒险的解决方法
  - 方案1：硬件阻塞
  - 方案2：软件上插入无关指令
  - 方案3：同一周期内寄存器堆先写后读
  - 方案4：利用数据通路中的中间数据：转发+阻塞
  - 转发条件为：
  - 数据冒险-方案5：编译器进行指令顺序调整来解决数据冒险
- 控制冒险现象及其解决方法 静态预测、动态预测、延迟分支
  - 控制冒险
  - 控制冒险的解决方法
  - 静态预测
  - 动态预测
  - 动态预测的流程
  - 动态预测的方法
  - 分支延迟时间片的调度
- 异常和中断引起的控制冒险、处理方法
  - 流水线数据通路处理异常的方式
  - 流水线处理异常的常见问题

七.存储器层次结构（约20分）

- 存储器的分类、主存储器的组成和基本操作、存储器的层次化结构
  - 存储器的分类
  - 主存储器的结构
  - 主存储器的性能指标
  - 存储的层次化结构
- SRAM和DRAM的区别

CPU和存储器之间的通信方式

  - SDRAM (Synchronous Dynamic Random Access Memory)
- 存储器芯片的扩展
  - 位扩展的原理
  - DRAM的内部结构
- 连续编址方式、交叉编址方式
  - 连续编址
  - 交叉编址
- 磁盘读写的三个步骤

- 磁盘存储器的性能指标
- 数据校验的基本原理 奇偶校验码 循环冗余校验码
    码字和码距
    奇偶校验码
    循环冗余校验码（CRC code）
  程序访问的局部性
    码字和码距
- 程序访问的局部性
- Cache的基本工作原理
- 直接映射、全相连映射、组相连映射（命中率、命中时间、缺失损失、平均访问时间）
    直接映射
    全相连映射
    组相连映射
        组相连映射tag的计算
    命中率、命中时间、缺失损失、平均访问时间
- 先进先出算法、最近最少用算法
    先进先出算法
    最近最少用算法
- 全写法、回写法的区别
- 虚拟存储器的基本概念
- 进程的虚拟地址空间划分
- 分页式虚拟存储器的工作原理（页表、地址转换、快表、CPU访存过程）
    页表
    地址转换
    快表

八.系统互联及输入输出组织（约10分）

系统互连、输入输出组织

- 外设的分类
- 总线、系统总线、数据线、地址线、控制线
    总线
    系统总线
- 基于总线的互连结构 （主要模块以及连接的总线）
- I/O接口的职能、通用结构
    职能
    通用结构
- I/O端口的独立编址方式、统一编址方式
    独立编址方式
- 程序直接控制I/O方式、中断控制I/O方式、DMA方式的工作原理、区别
    程序直接控制I/O方式
    中断I/O方式
    直接存储器存取方式（Direct Memory Access）（磁盘等高速外设专用）
- 中断响应、中断处理 中断优先权的动态分配
    中断系统的基本职能
    中断处理
    中断优先级
    中断屏蔽字
- 3种DMA方式：CPU停止法、周期挪用法、交替分时访问法
    CPU停止法
    周期挪用法
)交替分时访问法
- I/O子系统层次结构、每层的基本功能
    与设备无关的I/O软件
    设备驱动程序
    中断服务程序
- 用户程序、C语言库、内核之间的关系

考试题型

1. 填空题（约20分）： 1分/空×20空
2. 选择题（约20分）： 1分/题×20题

3. 判断题（约10分）： 1分/题×10题
4. 简答题（约10分）： 5分/题×2题 （xx层次结构…）
5. 计算题（约40分）： 10分/题×4题 (作业题变题…)

总复习

一.计算机系统概述（约5分）

• 计算机硬件的基本组成

冯·诺依曼结构

1. 运算器
2. 控制器
3. 存储器
4. 输入设备
5. 输出设备

现代计算机：把运算器、控制器和各类寄存器等互连在一个中央处理器中(CPU)。

– 中央处理器：核心部件，用于指令执行；包含数据通路和控制器。

– 存储器：分为内存和外存。

– 外部设备和设备控制器：外设通过设备控制器连接到主机上

– 总线：传输信息的介质，用于在部件之间传输信息

• 计算机软件的分类

应用软件：专门为某种应用编写的程序（如电子邮件收发软件、视频播放软件、 文字处理软件等）

系统软件：为有效、安全地使用和管理计算机以及为开发和运行应用软件而提供 的各类软件；介于计算机硬件与应用软件之间

• 计算机系统的抽象层及其转换

功能转换： 上层是下层的抽象，下层是上层的 实现，底层为上层提供支撑环境

程序执行： 不仅取决于算法、程序编写，而且取决于语言处理系统、操作系统、 指令集体系结构、微体系结构等。

• 用户CPU时间计算 (选择题)

CPI：执行一条指令所需的时钟周期数

指令速度计量单位 MIPS：平均每秒钟执行多少百万条指令(记得除以10^6)

• Amdahl定律(选择填空)

改进后的执行时间 = 改进部分执行时间 / 改进部分的改进倍数 + 未改进部分执行时间

整体改进倍数 = 1/（改进部分执行时间比例/改进部分的改进倍数+未改进部分执行时间比例）

二.数据的机器级表示（约10分）

• 二进制、八进制、十六进制、十进制数之间的转换

表示字母:二进制B,八进制O，十进制D,十六进制H

• 原码、补码、移码表示法

原码补码略

变形补码:

$[X_T]_{变补} = 2^{n+1} - X_T \ (\text{mod}\ 2^{n+1})$

移码:

n位： $E_{移} = E_{原} + 2^{n-1}$ 或 $2^{n-1} -1$

小技巧：补码把直接符号位取反就是移码

• 无符号整数、带符号整数表示

略

• 浮点数的表示？IEEE 754浮点数表示？ （若涉及特殊情况，会给出提示或解释）

特殊序列：

- 全0阶码全0尾数: +0/-0
- 全0阶码非0尾数：非规格化(用于处理阶码下溢)
- 全1阶码全0尾数： +∞/-∞
- 全1阶码非0尾数： NaN (not a number)
- • 阶码非全0且非全1：规格化非0数　　(0.尾数 * 2^{-126})

• C语言中的整数、浮点数类型 (转换出现问题)(数据大小端)

字节(Byte) = 8 bit

大端模式，是指数据的高字节保存在内存的低地址中

小端模式，是指数据的低字节保存在内存的低地址中

• 数据的存储和排列顺序

存储器按字节编址:按边界对齐

存储器不按字节编制:不按边界对齐

三.运算方法和运算部件（约10分）

• 理解并能写出常见的汇编指令，尤其是算术运算、逻辑运算涉及的指令

• 串行、并行、带标志的加法器原理

全加器原理

$F_i = X_i + Y_i + C_i$

$C_{i+1} = (X+B)C_i + X_i \cdot Y_i$

$G_i = X_i Y_i$

$P_i = X_i + Y_i$

$C_1 = G_1 + P_1 C_0$

$C_2 = G_2 + P_2 C_1 = G_2 + P_2 G_1 + P_2 P_1 C_0$

$C_3 = G_3 + P_3 C_2 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$

$C_4 = G_4 + P_4 C_3 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0$

• 补码加减运算（原码乘除、补码乘除：若涉及，会给出计算规则）

• 乘除运算溢出判断，常量乘除运算

X*Y的高n位可以用来判断溢出，规则如下：

无符号：若高n位全0，则不溢出，否则溢出

带符号：若高n位全0或全1且等于低n位的最高位，则不溢出。

变量与常数之间的除运算

– 无符号数、带符号正整数：移出的低位直接丢弃

带符号负整数：加偏移量(2^k-1)，然后再右移k位,低位截断（这里K 是右移位数）

• 浮点数加减运算、舍入方式

浮点数加减步骤

1. 检查有无0操作数
2. 小阶向大阶对
3. 尾数加减
4. 规格化

四.指令系统（约15分）

• 指令格式设计

零地址指令: op

一地址指令: op | A1

(1) 单目运算：如：取反 / 取负等

(2) 双目运算：另一操作数为默认的 如：累加器等

二地址指令: op | A2 | A2

分别存放双目运算中两个操作数，并将 其中一个地址作为结果的地址

三地址指令（RISC风格）：op | A1 | A2 | A3

• 指令执行周期

1. (计算下条指令地址)从存储器取指令
2. 对指令译码
3. 计算操作数地址 , 取操作数
4. 计算并得到标志位
5. 保存计算结果(计算下条指令地址)

• 操作数类型、寻址方式、操作类型、操作码编码

偏移寻址方式

1.相对寻址

指令地址码给出一个偏移量(带符号数)，基准地址**隐含**由PC给出

EA=(PC)+A (例: beq)

2.基址寻址

指令地址码给出一个偏移量，基准地址**明显或隐**含由基址寄存器B给出

EA=(B)+A　(例: lw/sw)

3.变址寻址

指令地址码给出一个基准地址，而偏移量(无符号数)**明显或隐含**由变址寄存器 I 给出

EA=(I)+A

确定目标地址前,首先确定是**按字编址**还是**按字节编址**!!!!!

当按字节编址且D为单元数时，转移目标地址= (PC)+指令长度（byte）+D　(D为跳转指令偏移量)

• 异常和中断的区别

中断分为两种：

- 内部异常：在CPU执行指令时内部发生的意外事件
    - 故障：执行指令时发生的异常事件（溢出、缺页、越界、越权、越级、非法指令、除0、堆栈溢出、访问超时）
    - 自陷：执行预先设置的指令（断点、系统调用等）
    - 终止：执行指令时出现了硬件故障
- 外部中断：在CPU外部发生的特殊事件（如时钟、控制台、打印机缺纸等）

异常、中断的处理

- 检测和响应（硬件处理）
- 处理（软件完成）

当发生异常和中断时，系统将从用户态进入**内核态**来进行异常和中断的处理。

• MIPS指令的格式和寻址方式MIPS指令的格式和寻址方式

- 位宽：等宽（32位）
- 三种指令格式
    1. R-type：两个操作数和结果都在寄存器
        - |31 op 26|25 rs 21|20 rt 16|15 rd 11|10 shamt 6|5 func 0|
    2. I-type：需要用到立即数的指令
        - |31 op 26|25 rs 21|20 rt 16|15 imm 0|
    3. J-type：无条件跳转指令
        - |31 op 26|25 addr 0|

• 选择结构、循环结构的汇编指令表示 （若涉及机器代码，会给出指令格式和机器代码之间的关系）

没啥好看的

• 过程调用的指令、执行步骤、栈和栈帧的变化(图)(例题)

执行步骤(P调用Q)

1. 将参数放到Q能访问到的地方
2. 将P中的返回地址存到特定的地方，将控制转移到过程Q
3. 为Q的局部变量分配空间（局部变量临时保存在栈中）
4. 执行过程Q
5. 将Q执行的返回结果放到P能访问到的地方
6. 取出返回地址，将控制转移到P，即返回到P中执行

MIPS中的栈

- MIPS中的 **栈增长方向** 是 **从高地址向低地址增长**；
- MIPS栈的取数方向是 **大端序。**
- MIPS栈中，各个过程有各自的 **栈帧。**
- 每个栈帧都有自己的帧指针，用于指示栈帧开始的位置。
- 当前栈帧位于栈指针和帧指针之间。

MIPS寄存器的使用约定

- 保存寄存器`$s0 ~ $s7`的值在从被调用过程返回后仍然需要，需要由被调用着保留。
- 临时寄存器`$t0 ~ $t9`的值在从被调用过程返回后不再需要，（若需要的话可以由调用者保存）。
- 参数寄存器`$a0 ~ $a3`的值在从被调用过程返回后不再需要，（若需要的话可以由调用者保存）。
- 全局寄存器`$gp`的值不变
- 帧指针寄存器`$fp`的值用栈指针`$sp - 4`初始化
- 返回地址`$ra`的内容需要由被调用者保存

五.中央处理器（约15分）

• CPU执行指令过程

- 取指令
- PC自增
- 指令译码
- 计算主存地址
- 取操作数
- 算术 / 逻辑运算
- 存结果

以上每步都需检测“异常”，若有异常，则自动切换到异常处理程序 ； 检测是否有“中断”请求，有则转中断处理

异常和中断的差别: 异常是在CPU内部发生的 中断是由外部事件引起的

• CPU的基本组成？操作元件和状态元件的区别

CPU的组成

CPU中包含：

- 数据通路（Datapath），指令执行过程中，数据所经过的路径（及路径上的部件），是指令的**执行部件。**
- 控制器（Control），负责生成指令对应的控制信号，控制数据通路的动作；是指令的控制部件。

数据通路由 **组合逻辑元件**（操作元件） 和 **时序逻辑元件**（状态元件/存储元件） 由 **总线连接方式** 或 **分散连接方式** 连接而成，负责 **数据的存储、处理、传送**

关于 操作元件

- 加法器：不需要控制信号
- 多路选择器（MUX）：需要 **选择信号**
- 算术逻辑部件（ALU）：需要 **操作码（Opcode）**
- 译码器（Decoder）：给出多个 **输出信号**

操作元件的特点：

- 输出只取决于当前的输入
- 输出端改变和输入端改变之间有固定的逻辑门延时，不需要时钟信号。

关于 状态元件

状态元件的特点：

- 具有存储功能，在 **时钟控制下** 输入被写入到电路中，直到下个时钟信号到达。
- 时钟决定何时将输入端状态写入元件，但输出端可以随时读出
- 使用边缘触发方式定时：状态单元的值只在时钟信号的边缘处改变，每个时钟周期改变一次。
    - 上升沿触发
    - 下降沿触发

状态元件有三个指标：

- 建立时间（Setup）：在时钟信号到达前，输入信号需要维持多长时间稳定
- 保持时间（Hold）：在时钟信号到达后，输入信号还需要维持多长时间稳定
- 锁存延迟（Clock-to-Q）：从输入信号改变到输出信号改变的时间

常见的状态元件：

- 寄存器
    - 有一个写使能（Write-Enable）信号：使得时钟信号到来时输入端的值能够写入寄存器内。
- 寄存器组
    - 有三个地址线RA、RB和RW：给出需要读取和写入的存储单元地址
    - 有两个输出口busA与busB：当RA与RB传入地址后，经过一个 **取数时间**， 输出对应单元的内容到busA与busB中。
    - 有一个输入口busW：当输入信号RegWr（写使能）为真时，在下一个到来的时钟边沿将busW传来的值写入到地址在RW的寄存器中

• 指令周期、时钟周期

指令周期指CPU取出并执行一条指令的时间；各指令的指令周期各不相同。

时钟周期=锁存延迟+最长传输延迟+建立时间+时钟偏移

• MIPS指令格式，典型的MIPS指令以及功能描述（书本134-135页）

• 典型的MIPS指令数据通路？能够理解单周期数据通路设计，画出局部数据通路,明确控制信号取值。

以下为各条指令涉及的控制信号：

指令	控制信号
add/sub	ALUctr=add/sub，RegWr=1
ori(立即数0扩展)	ALUctr=or;RegDst=1(选择rd写回)；RegWr=1；ALUSrc=1(控制是否不为R型指令)
lw(需要符号扩展)	RegDst=1, RegWr=1, ALUctr=addu, ExtOp=1(立即数符号扩展), ALUSrc=1, MemWr=0(无需写入主存), MemtoReg=1(主存写入寄存器)
sw(需要符号扩展)	RegDst=x, RegWr=0, ALUctr=addu, ExtOp=1, ALUSrc=1, MemWr=1, MemtoReg=x
beq	RegDst=x, RegWr=0, ALUctr=subu, ExtOp=x, ALUSrc=0, MemWr=0, MemtoReg=x, Branch=1
jump	RegDst=ExtOp=ALUSrc=MemtoReg=ALUctr=x, RegWr=0, MemWr=0, Branch=0, Jump=1

• 多周期数据通路设计？能够理解多周期数据通路设计，明确控制信号取值(一定要知道)、指令执行状态转换图。

以下是各控制信号的列表：

- PCWr：控制是否将新地址写入PC。
- PCWrCond：（在beq指令中，与Zero信号共同）决定是否跳转到新地址。
- IorD：控制从内存中读取的的是指令还是数据。
- MemWr：控制内存写使能。
- IRWr：控制指令寄存器（记录当前指令）的写使能。
- RegDst：控制是否向Rd中写入数据。
- ExtOp：控制对立即数进行扩展的类型。
- MemtoReg：控制回写到目标寄存器的是来自内存的内容还是来自ALU的内容。
- RegWr：控制寄存器堆的写使能。
- ALUSelA：控制送入ALU的第一个操作数的来源（PC / BusA）。
- ALUSelB：控制送入ALU的第二个操作数的来源（常量（4） / BusB / immExt << 2 / immExt）。
- PCSrc：控制送入PC的值的来源（提前计算好的Target还是ALU的结果）。
- BrWr：控制是否将ALU提前计算的PC的下一个值写入Target寄存器。
- ALUOp：控制ALU进行的运算种类。

• 微程序控制器的基本思想、基本结构、执行、编码方式

设计思路

编制每个指令所对应的微程序；每个微程序由若干微指令构成，而每个微指令又由若干条微命令组成。

微程序在不同的周期执行不同的微指令（包含一系列控制信号的取值，即“微命令”）。

所有指令对应的微程序存放在一个只读存储器（控制存储器，Control Storage）中。

结构

- IR：存储当前需要查找对应微程序的指令。
- 起始和转移地址发生器：接受条件码，产生起始的微指令地址和下一条微指令的地址
- microPC：微指令计数寄存器，记录当前执行的微指令的地址
- 控制存储器：核心部件，存储所有的微程序、微指令和微命令
- microIR：微指令寄存器，存储当前的微指令
- Decoder：微指令解码器，将微指令解码成控制信号。

取指令和译码用专用的微程序实现；其他的指令由其他微程序实现。

)微指令的种类

- 水平型微指令：将相容的微命令尽量多的安排在同一条微指令中
    - 程序短、并行度高，但编码空间利用率低，编制困难。
- 垂直型微指令：一条微指令只控制一两个微命令。
    - 编码效率高，容易编制，但程序长，无并行，速度慢。

微指令的内容

- 微操作码（生成微命令）
- 下一条微指令地址（可选）
- 常数（可选）

微操作码的编码方式

- 不译法（直接控制）
    - 微指令字很长，空间利用率低。
    - 并行控制能力强，速度快。
- 字段直接编码
    - 把微指令分成若干字段。
    - 把互斥的（不能同时进行的）微操作编码在同一字段。
    - 一条微指令最多可同时发出的微命令个数就是字段数（各个字段同时开始执行）。
    - 并行控制能力高，速度快；微指令较短。
    - 增加译码时间和线路。
- 字段间接编码
- 最小编码译法

• 带异常处理的数据通路、有限状态机

MIPS使用软件来识别中断源。

MIPS处理器的异常处理

需要在数据通路中加入两个寄存器：

- EPC：存放断点（异常处理结束后返回到的指令地址）。
    - 其中可能是正在执行的指令地址（故障：EPC = PC - 4，PC在开始执行后已经自增了），也可能是下一条指令的地址（自陷和中断，EPC = PC）。
- Cause：记录异常原因。

需要加入两个控制信号：

- EPCWr：在保存断点时有效。
- CauseWr：在发现异常时有效。
- IntCause：选择正确的异常编码值来写入Cause中。

需要加入两个异常状态：

- UndefInstr：未定义指令异常
- Overflow：数据溢出异常

故障在指令执行过程中检测。

自陷在指令译码或者条件码检测中检测。

中断在每条指令执行后检测（中断是随机发生的）

六.指令流水线（约15分）

• 指令流水线由哪些流水段组成、各流水段的功能？使用哪些部件?

取指令(IF)：从存储器取指令； 指令存储器、Adder

指令译码(ID): 产生指令执行所需的控制信号；寄存器堆读口、指令译码器

取操作数(OF): 读取操作数； 扩展器、ALU

执行(EX): 对操作数完成指定操作； 数据存储器

写回(WB): 将结果写回; 寄存器堆写口

• 典型MIPS指令的功能段划分、流水线数据通路的设计、控制信号的取值（能够理解流水段寄存器，理解流水线数据通路，但不要求自己画出数据通路）

各指令对齐后的结果：

- Load: Ifetch, Reg/Dec, Exec, Mem, Wr
- R-type: Ifetch, Reg/Dec, Exec, **nop**, Wr
- Store: Ifetch, Reg/Dec, Exec, Mem, **nop**
- Beq: Ifetch, Reg/Dec, Exec, Mem, **nop**
- J: Ifetch, Reg/Dec, Exec, **nop**, **nop**

各个阶段的控制信号如下：

- Ifetch、Reg/Dec：无
- Exec：
  - ExtOp (扩展器操作)：0 = 零扩展，1 = 符号扩展
  - ALUSrc (ALU的B口来源)：0 = BusB,1 = 扩展器
  - ALUOp(主控制器输出，用于辅助局部ALU控制逻辑来决定ALUCtrl)
  - RegDst(指定目的寄存器)：0 = Rt,1 = Rd
- Mem：
  - MemWr(DM的写信号)：1 = Store
  - Branch(是否为分支指令)：1 = 分支指令
- Wr：
  - MemtoReg(寄存器的写入源)：0 = ALU输出， 1 = DM输出
  - RegWr(寄存器堆写信号)：1 = 写寄存器

• 结构冒险现象及其解决方法

现象：同一个部件同时被不同指令所使用

解决:

1. 一个部件每条指令只能使用1次，且只能在特定周期使用
2. 设置多个部件，以避免冲突。如指令存储器IM 和数据存储器DM分开

例:

1. 将指令存储器(Im)和数据存储器(Dm)分开
2. 将寄存器读口和写口独立开来
3. 每个部件在特定的阶段被用

• 数据冒险现象及其解决方法（需要深入理解转发技术、load-use数据冒险的检测和处理方法）

数据冒险

数据冒险指后续指令执行时需要之前指令还未计算完成的数据的现象。

数据冒险有三种：

- **Read After Write**，基本流水线中经常发生。
- **Write After Read**，基本流水线中不会发生。
- **Write After Write**，基本流水线中不会发生。

数据冒险的解决方法

1. 硬件阻塞
2. 软件插入空指令
3. 合理实现寄存器堆的读写（不能完全解决）
4. 转发技术（仅能解决需要ALU结果的数据冒险，不能解决需要内存结果的数据冒险（**Load-use数据冒险**））。
5. 编译优化（不能完全解决）

方案1：硬件阻塞

缺点：控制比较复杂，需要改数据通路；指令被延迟三个时钟执行

方案2：软件上插入无关指令

• 由编译器插入三条NOP指令，浪费三条指令的空间和时间，是最差的做法。

实行方案3后有时只需要插入2条nop

方案3：同一周期内寄存器堆先写后读

寄存器**写口/读口**分别在**前/后半周期**进行操作，使写入数据被直接读出

方案4：利用数据通路中的中间数据：转发+阻塞

把数据从流水段寄存器中直接取到ALU的输入端

缺点：不能解决所有数据冒险（Load-use）。

Load-use 数据冒险的解决方法： **转发 + 阻塞**

当Load指令后的第一条指令需要使用load指令的结果时，
需要对该指令延迟一个周期。

转发条件为：

– C1(a):

EX/MEM.RegWr and EX/MEM. RegisterRd ≠ 0

and EX/MEM. RegisterRd=ID/EX. RegisterRs

– C1(b): EX/MEM.RegWr

and EX/MEM. RegisterRd ≠ 0

and EX/MEM. RegisterRd=ID/EX. RegisterRt

– C2(a): MEM/WB.RegWr

and MEM/WB. RegisterRd ≠ 0

and MEM/WB. RegisterRd=ID/EX. RegisterRs

– C2(b): MEM/WB.RegWr

and MEM/WB. RegisterRd ≠ 0

and MEM/WB. RegisterRd=ID/EX. RegisterRt

数据冒险-方案5：编译器进行指令顺序调整来解决数据冒险

• 控制冒险现象及其解决方法 静态预测、动态预测、延迟分支

控制冒险

当指令序列中出现流程控制语句时，流程控制指令会和后续的指令产生控制冒险（即如果需要跳转，那么在取出跳转的目标指令之前已经有多条指令被错误的取出）。

延迟损失时间片C：发生转移时，因为错误取出指令而给流水线带来的延迟损失。

控制冒险的解决方法

1. 硬件阻塞：清零错误取出的指令，在对应的地方插入nop指令。（效率太低）
2. 软件插入nop指令。（效率太低）
3. 分支预测
    - 静态预测（总是预测不满足）
    - 动态预测
4. 延迟分支（优化指令顺序）

静态预测

总是预测条件不满足；（可以添加启发式规则：特定情况下预测满足）。

预测失败时，将流水线中的三条错误预测指令丢弃（将控制信号值 / 指令清零）。

预测错误的代价 预测错误的代价与确定转移的时机有关：越早确定代价越小。

最早可以将 转移地址的计算 和 分支条件的判断 提前到ID段执行：

- 在ID阶段，IF / ID流水段寄存器中已经有了下一条指令PC的值和立即数
- 可以用特殊手段来判断是否为0（beq的需求）。
  预测错误的检测和处理 发生转移的条件：Branch == 1 && Zero == 1

增加一个控制信号：IF.Flush = （Branch & Zero）。
该控制信号取值为1时说明预测失败，需要：

- 将转移目标地址送PC
- 清除IF段取出的指令（清零IF/ID中的指令字）

此时的延迟损失时间片为1（提前了）

动态预测

将最近的转移历史记录到BHT（分支历史记录表）中，并利用最近的转移情况来预测下一次是否转移。

BHT由分支指令地址的低位作为索引。

动态预测的流程

- 根据分支指令地址在BHT中查找之前对应的分支指令的跳转情况。
- （若找到）根据历史的跳转情况进行预测；（若未找到）在BHT中加入新项，填入指令地址、转移目标地址，初始化预测位
- （若找到）根据预测位进行指令的选择。
- 根据执行结果更新预测位。

动态预测的方法

- 采用一位预测位：
    - 若预测位为1,则说明最近发生了转移，并预测下次发生转移。
    - 若预测位为0,说明最近未发生转移，预测下次不转移。
    - 若预测错误，则将预测位取反。
- 采用两位预测位：
    - 按照预测状态图来进行预测（弱转移、强转移、弱不转移、强不转移）。

分支延迟时间片的调度

将分支指令前与分支无关的指令调整到分支后执行，以填充延迟时间片

不够填充时剩下的时间片用nop填充

异常和中断会改变程序的执行流程。

如，当前的某条指令执行过程中发现异常，此时后续的数条指令已经被取到流水线中开始执行了。

• 异常和中断引起的控制冒险、处理方法

流水线数据通路处理异常的方式

- 清除发生异常的指令以及后续已经进入流水线的指令。
- 关中断（清零中断触发器）
- 保存断点到EPC
- 将异常处理程序开始地址送PC

流水线处理异常的常见问题

- 通过异常发生的流水段可以确定是哪条指令发生了异常。
    - 溢出：Ex
    - 无效指令：ID
    - 除数为0：ID
    - 无效指令地址：IF
    - 无效数据地址：Load/Store指令的Ex
- 外部中断通过中断查询来确定处理点
- 非精确中断不能提供准确的断点，由操作系统确定发生异常的指令。

- 一个时钟周期内若发生多个异常，则前面指令的异常优先级大于后面指令的异常。

七.存储器层次结构（约20分）

• 存储器的分类、主存储器的组成和基本操作、存储器的层次化结构

存储器的分类

- 按照工作性质 / 存取方式分类
  - 随机存取存储器（RAM），按地址访问，单元之间的读写时间相同。
  - 顺序存取存储器（SAM），数据按顺序写入或读出（如磁带等）。
  - 直接存取存储器（DAM），先定位到读写数据块，在读写数据块时按顺序进行（磁盘）。
  - 相联存储器（AM / Content Addressed Memory），按照内容检索存储位置。
- 按照存储介质分类
  - 半导体存储器
  - 磁表面存储器
  - 光存储器
- 按信息的可更改性分类
  - R/WM：可读可写
  - ROM：只读
- 按是否易挥发分类
  - 易失性存储器（Volatile Memory）
  - 非易失性存储器（Nonvolatile Memory）
- 按功能 / 容量 / 速度 / 所在位置分类
  - 寄存器：CPU内，速度快，容量小，存放当前的指令和数据；用触发器实现。
  - 高速缓存：CPU内，存放局部程序段和数据，容量较小；用SRAM实现，速度较快。
  - 主存储器：CPU外，存放已经启动的程序和数据，容量较大；用DRAM实现，速度一般。
  - 外部存储器：主机之外，存放暂不运行的程序、数据等，容量大；用磁表面存储器或光存储器（现代用半导体存储器）实现，速度较慢

主存储器的结构

- 地址寄存器：用于暂时存储CPU送来的地址。
- 地址译码器：用于将CPU送来的地址解码；连接 **地址线。**
- 记忆单元：用于存储数据，连接 **数据线** 。
- 读写控制电路。

地址线的宽度决定了主存储器的寻址范围：如地址线宽度为36位，则可寻址范围为 2 ^ 36 - 1个单元。

主存容量 != 主存地址空间大小!

主存储器的性能指标

- 编址方式（字节编址）
- 存储容量
- 存取时间（从CPU送出内存单元的地址码到主存将对应的数据送回CPU的时间）
- 存储周期 （**连续两次访问存储器所需的最小时间间隔**），一般大于存取时间。

存储的层次化结构

- 寄存器
- 高速缓存
- 主存储器
- 辅助存储器（如硬盘）
- 后备存储器

速度越快，容量越小，越靠近CPU.

CPU可以直接访问其内部的存储器；外部的存储器中的信息先要被取到主存中才能被CPU访问。

数据一般只在相邻层之间复制传输，且总是从慢速存储器复制到快速存储器。

• SRAM和DRAM的区别

- SRAM((静态随机存取存储器)中的数据保存在一对正负反馈门电路中。
    - 在持续供电下不需要刷新内部存储的数据，读出也不会破坏内部的数据。
    - 速度快，容量相对较小，成本较高，功耗较大。
- DRAM(动态随机存取存储器)中的数据保存在电容中。
    - 电容会持续放电，因此DRAM中的数据需要不断刷新，读出会破坏内部的数据。
    - 速度慢，容量较大，需要定时刷新，功耗较小。

因此，SRAM适合用作Cache,DRAM适合用作主存。

CPU和存储器之间的通信方式

- 异步方式
    - CPU送地址到地址线，主存进行地址译码
    - CPU发送读命令，等待存储器发送“完成”信号
    - 存储器开始读数据，完成后发送“完成”信号至CPU
    - CPU接收到“完成”信号，从数据线取数据
- 同步方式
    - CPU和主存由统一的时钟信号控制
    - 主存总是在确定的时间内准备好数据
    - CPU送出地址和信号后总是在确定的时间内取数据
    - **需要存储器芯片的支持**

SDRAM (Synchronous Dynamic Random Access Memory)

- 每步操作在系统时钟控制下进行
- 有确定的等待时间（从读命令开始到数据送到数据线的时间）
- 连续传送数个数据（2(DDR) / 4(DDR2) / 8(DDR3) / 16(DDR4) / ...）
- 多体交叉存取
- 利用总线时钟上升沿和下降沿同步传送

• 存储器芯片的扩展

存储器可以有两种扩展方式：字扩展（扩展内存长度）和位扩展（扩展位宽度）。

存储器的地址范围（地址位宽）取决于字数，如4K（4096）个单元需要12bit宽的地址线才能完整编码（2^12 = 4096）。

主存地址需要在片内地址的基础上外加存储芯片数量的低位，用来选片。

位扩展的原理

假设用8n个8bit宽度的存储芯片组成了一个64bit存储器（编号为0,1,2,…，7），读出64bit数据时，存储控制器会先传入行数i和列数j；每个存储芯片都读取并输出位于 (i, j)处的8bit数据；将这8个8bit数据按照一定顺序排列，即可得到原始的64bit数据。

按照这样的方式，一次访存读出的数据分别为（0～8单元，9～15单元，…，8k ～ 8k+7单元）。

因此，若是某个数据并未对齐（起始地址非4的倍数），就需要两次访存才能读取完毕，带来额外的开销。

DRAM的内部结构

DRAM中，每行的行首会存在一个 **行缓冲**，用SRAM实现。

当系统需要读取DRAM中的数据时，会先将行数送至存储器中；存储器解码后会将该行的所有数据送入缓冲中。然后，系统将列数送至存储器中；存储器将缓冲中对应列的数据送至地址线

• 连续编址方式、交叉编址方式

多模块存储器利用多个结构完全相同的存储器并行工作来提高存储器的吞吐率。

多模块存储有两种编址方法：连续编址和交叉编址。

连续编址

连续编址方法中，主存地址的 **高位** 表示 **模块号**（第几个模块），**低位** 表示 **体内地址**（在模块中的哪里），因此也称为 **按高位地址划分方式。**

连续编址的地址在模块内连续。

编址方式如下。

模块0：[0, 1, 2, 3, ..., 15]
模块1：[16, 17, 18, ..., 31]
模块2：[32, 33, 34, ..., 47]
模块3：[48, 49, 50, ..., 63]

地址：[0 ~ 3 | 0 ~ 15]

当使用该方法访问一个连续的主存块时，存储器总是先在一个模块内访问，结束后再转移到下一个模块；无法提高存储器的吞吐率。

交叉编址

交叉编址的方法中，主存地址的 **低位** 表示 **模块号**， **高位** 表示 **体内地址**， 因此也称按地位地址划分方式。

一般采用交叉编址的多模块存储器会使用2的幂次数量的模块（实现简单）。

编址方式如下。

模块0：[0, 4, 8, ..., 60]
模块1：[1, 5, 9, ..., 61]
模块2：[2, 6, 10, ..., 62]
模块3：[3, 7, 11, ..., 63]

地址：[0 ~ 15 | 0 ~ 3]

采用交叉编址时，存储控制器会将高位的体内编码送至对应的模块中进行单独解码运算。

由于该种编码方式会使得多个模块同时工作，因此可以提高整个存储器的吞吐率。

交叉编址多模块存储器可以采用两种启动方式。

- 轮流启动
    - 如果每个存储模块读一次的位数刚好等于存储器总线的数据位数，则采用轮流启动方式。
    - 对于具有m个模块的存储器，每1/m周期启动一个存储器，可使得存储速率提高m倍。
- 同时启动
    - 如果所有存储模块读一次的位数等于存储器总线的数据位数，则采用同时启动方式。

• 磁盘读写的三个步骤

传统机械磁盘的读写数据有三个步骤：

1. **寻道操作**
    - 磁盘控制器将地址送到地址寄存器，然后产生寻道命令，令磁头定位伺服系统将磁头移动到正确的磁道上，并选择正确的磁头准备读写；完成后发出寻道结束信号给控制器，进入下一阶段操作。
2. **旋转等待操作**
    - 将扇区计数器清零；当磁盘旋转时，磁头每收到一个扇区开始脉冲便自增扇区计数器，并将计数器和地址中的扇区号进行比较。若相等，则发出扇区符合信号，进入下一阶段。
3. **读写操作**
    - 磁盘控制器的读写电路控制将数据送至写入电路进行写入，或由读出放大电路将数据送至磁盘控制器

• 磁盘存储器的性能指标

磁盘存储器有三个性能指标：

1. **存储容量**
    - 低密度存储、未格式化容量：记录面数 * 理论柱面数 * 内圆周长 * 最内道（的）位密度
    - 低密度存储、格式化容量：2 * 盘片数 * 每面的磁道数 * 每磁道的扇区数 * 512B
2. **数据传输速率**
3. **平均存取时间**
    - 平均寻道时间 + 平均旋转等待时间 + 数据传输时间

• 数据校验的基本原理 奇偶校验码 循环冗余校验码

现今的数据校验大多采用冗余校验方式（增加几位校验位，存储原信息经过某种运算的结果）。

校验的结果有三种：

- 没有错误：直接传送。

- 检测到差错但可以纠正：数据位和比较结果送纠错器。
- 检测到错误，无法纠正（不知道哪位出错）：报告错误。

码字和码距

码字：若干位代码组成的字。

码距：某编码系统中，两个不同码字之间 **不同的位**的**个数** 的 **最小值。**

码距和寻错、纠错能力之间的关系（码距1 < d <= 4）

- （码距为奇数）码距为d的编码能发现 d - 1 位错，能纠正（d - 1）/ 2 位错
- （码距为偶数）码距为d的编码能发现 d / 2 位错，能纠正 d / 2 - 1 位错

奇偶校验码

码距为2,能检验不能校正

- 实现原理：
  - 奇校验：P = b_n-1 ^ b_n-2 ^ ... ^ b_1 ^ b_0 ^ 1
  - 偶校验：P = b_n-1 ^ b_n-2 ^ ... ^ b_1 ^ b_0
  - 假设传输前的校验码为P，传输后的校验码为P'，令P* = P ^ P'
    - p*=1: 发生了奇数位错误
    - P* = 0: 没有错误 / 发生了偶数位错误
- 特点：不具有纠错能力、开销小；适合校验一字节长的代码（只有1位出错的概率相对较大），常用于存储器读写检查。

循环冗余校验码（CRC code）

模2除法

模2除法的大致流程与竖式除法相同，只是在取商的余数部分时使用的是 **异或** 而非传统的减法。

例： 10001 1001 00011

基本思想：

- 假设数据有 n 位，设为M，约定数字 k 和一个 k 次生成多项式 G(x) = a0 + a1 * x + ... + ak * x^k, a_i 取0或1，代表了k位二进制数字ak|ak-1|ak-2|...|a1|a0，设为G。
- 将M左移k位并用 **模2除法** 和G相除，得到k-1位余数rem。
- 将rem拼接在M后，得到该数据的CRC码，设为F。

这样得到的校验码F一定能（在模2除法下）被G整除。

程序访问的局部性

码字和码距

码字：若干位代码组成的字。

码距：某编码系统中，两个不同码字之间 **不同的位** 的 **个数** 的 **最小值。**

码距和寻错、纠错能力之间的关系（码距1 < d <= 4）

- （码距为奇数）码距为d的编码能发现 d - 1 位错，能纠正（d - 1）/ 2 位错。
- （码距为偶数）码距为d的编码能发现 d / 2 位错，能纠正 d / 2 - 1 位错。

• 程序访问的局部性

大量典型程序的运行情况分析表明，**在较短时间间隔内，程序产生的地址往往集中在一个小范围内。**

这种现象被称为程序访问的局部性。

- **空间局部性**：某个被访问的存储单元的邻近单元也可能在较短时间内也被访问。
  - 如：数组的各个元素在内存中连续存放，在遍历数组时数组就具有了空间的局部性。
  - 注意：数组的元素在内存中按行-列的顺序排列
- **时间局部性**：某个被访问的存储单元可能在短时间内会被多次访问。
  - 如：循环中的各条指令在短时间内被反复访问

• Cache的基本工作原理

- 在程序运行时，CPU所需要的一部分指令和数据会预先复制到缓存中。
- 当程序开始执行时，CPU先给出主存地址AD，并查询AD所在的内存块是否在缓存中。
  - 若在缓存中：直接从缓存中取信息送CPU
  - 若不在缓存中：从主存取AD单元所在的存储块，在缓存中找到一个空闲的缓存行并将该块存入缓存中，并送CPU.

缓存对程序员和编译器是**透明不可见**的

有三种映射方式。

- 直接映射：每个主存块映射到Cache的固定行。
- 全相连映射：每个主存块映射到Cache的任意一行。
- 组相连映射：每个主存块映射到Cache的某个固定的组中的任意一行

• 直接映射、全相连映射、组相连映射（命中率、命中时间、缺失损失、平均访问时间）

直接映射

设缓存共有k行，则编号为i的主存块固定映射到缓存的第i % k行。

- 容易实现，命中时间短；
- 不够灵活，无法充分利用缓存空间，命中率低

采用直接映射机制的主存地址如下：

[主存标记（数据取自哪个主存块）|缓存索引（数据被缓存在哪个缓存行）|块内地址（数据在主存块内的位置）]。

块群号|Cathe行号|块内地址

例：假设缓存共16行，主存中的第17块第12单元被隐射到缓存中的第1行中，则主存地址为[0000 001 | 0010 | 0 0000 1100]。（主存标记 + 缓存索引 = 主存块序号（主存块按照缓存行的数量划分））。

直接映射的缓存内容如下：

（假设缓存有2^c行，主存有2^m块，则缓存tag共有m - c = t位——主存地址的低c位即为缓存行数。）

[有效位(1)|缓存tag(t)|缓存数据(data)]。

此处的缓存tag即为上述的主存标记

全相连映射

全相连映射的缓存内容如下：

[主存块地址 | 数据]。

访问时，直接将主存地址与缓存中的主存块地址依次比较。

组相连映射

将整个缓存划分为m组。

映射关系为：主存中第i块数据映射到缓存第i % m组中；组内使用全相联映射。

组相连映射tag的计算

设主存地址宽度为k位，块大小为2^mB, 共有2^n个缓存组（主存组）, 有2^t个主存组群，每个缓存组有p个缓存行。

则有2^k = 2^t * 2^n * 2^m（内存地址空间 = 组群数 * 组内块数 * 块大小）。

因此tag共k - n - m位。

命中率、命中时间、缺失损失、平均访问时间

命中率alpha：CPU要寻找的信息在缓存中的概率。

命中时间t_cache：在缓存中访问信息所需的时间，包含 决定是否命中的时间 和 访问缓存数据的时间。

缺失损失t_miss：当所需的信息不在缓存中时访问主存所需的时间。

平均访问时间：alpha * t_cache + (1 - alpha) * (t_cache + t_miss)

= t_cache + (1 - alpha)t_miss.

在三种映射方法中，直接映射的命中率最低，全相连映射的命中率最高；直接映射的命中时间最低，全相连映射的命中时间最高。

关联度

关联度指一个主存块映射到缓存中时，可能存放的位置个数。

- 直接映射：1
- 全相连映射：cache行数
- N路组相连映射：N

标记位和关联度正相关。因此关联度越高，标记位数越多，额外空间开销越大

• 先进先出算法、最近最少用算法

先进先出算法

总是先替换最先进入缓存的缓存行

最近最少用算法

总是将最近最少用的缓存行替换掉。

为每个cache行设立一个计数器，来记录这些行的使用情况。

- 命中时，被访问行的计数器清零；其他行的数值比之前该行计数器小的计数器 + 1。
- 未命中且当前组未满时，将新行的计数器置为0,其余计数器+1.
- 未命中且当前缓存组已满时，将计数器最大的行替换掉；该行的计数器置0,其余计数器+1

• 全写法、回写法的区别

- 全写法：同时将更改写入cache与主存单元；
- 回写法：只写入cache而不写入主存，当cache缺失时，一次将所有的更改全部写入主存。

• 虚拟存储器的基本概念

CPU通过存储器管理部件将指令中的逻辑地址（虚拟地址）转换为主存的物理地址

实质是通过页表创建虚拟空间和物理空间之间的映

• 进程的虚拟地址空间划分

- 一个程序在程序的 **链接** 阶段确定自身的虚拟地址；
- 在装入时生成页表以建立虚拟地址和物理地址的映射。
- 每个用户程序都有自己的虚拟地址空间：
    - 栈
    - 静态数据
    - 文字
    - 保留空间

• 分页式虚拟存储器的工作原理（页表、地址转换、快表、CPU访存过程）

- 分页式虚拟存储器的页大小比缓存中的数据块大得多。
- 分页式虚拟存储器采用全相连映射。
    - 缺页的开销远远大于缓存缺失的开销（需要访问磁盘）；更大的存储页和全相连映射可提高存储的页命中率。
- 通过软件来处理缺页（太慢，无法在硬件层面实现）。
- 使用 **回写** 的写策略（避免频繁写入磁盘）。
- 地址的转换用硬件实现（加速执行）。

页表

每个进程都有一个自己的页表，其中包含

- 装入位（是否已装入主存）
- 修改位（是否已经被修改，需要写回磁盘）
- 替换控制位（使用位）（说明页面的使用情况）
- 访问权限位（页面的读写权限）
- 禁止缓存位（页面是否可以装入缓存——用于保证磁盘、内存、cache的一致性）
- 实页号（存放位置）

页表的项目数由虚拟地址空间大小决定。

页表中的页有如下几种状态：

- 未分配：进程的虚拟地址空间中null对应的页
- 已分配，已缓存：有内容对应、且已经装入主存的页
- 已分配，未缓存：有内容对应，但还没有装入主存的页

地址转换

地址转换是虚拟地址被转换为物理地址的过程。

虚拟地址分为两个字段：

- 虚拟页号（高位）
- 页内地址（低位）

主存的物理地址分为两个字段：

- 物理页号（高位）
- 页内地址（低位）

变换的过程如下：

- 根据页表基址寄存器找到主存中的页表起始位置。
- 以虚拟页号作为索引找到页表内对应的页表项。
    - 对应项装入位为1：取出物理页号，和虚拟地址中的页内地址拼接形成实际物理地址。
    - 对应项装入位为0：缺页，需要操作系统处理。
        - 从磁盘读取相应内容到内存。（内存没有空间时采用类似于cache的策略淘汰和写回）
        - 当前指令的进行被阻塞，直到缺页处理完成继续执行。

可能出现的另一种异常：保护违例（存取权限和具体操作不相符）

- 显示错误信息
- 终止当前进程

快表

将经常需要的页表项存入缓存中。这种cache中的页表称为后备转换缓冲器，简称快表。

快表中的页表项内容：

- 原页表项内容

- TLB tag（虚拟页号——组相联方式下取虚拟页号的高位部分作为tag，低位部分作为索引选择TLB组）。

CPU访存过程：

- 将虚拟页号分为tag和索引，根据索引确定页表所在的TLB组。
- 将tag中的内容和TLB组中表项的内容比较。
    - 如果某项相等且有效位为1：TLB命中，通过TLB进行地址转换
    - 反之：TLB缺失，访问主存查询页表。
        - （查询页表过程）

TLB命中则页一定命中，但cache不一定命中。

页缺失说明信息一定不在主存，TLB不可能命中，cache也不可能命中

八.系统互联及输入输出组织（约10分）

系统互连、输入输出组织

• 外设的分类

按信息传输方向：

- 输入设备
- 输出设备
- 输入 / 输出设备

按功能：

- 人机交互设备
- 存储设备
- 机-机通信设备

• 总线、系统总线、数据线、地址线、控制线

总线

计算机内传输数据的公共路径。

系统总线

连接处理器、存储器、I/O模块等主要部件的总线。

通常由一组控制线、一组数据线和一组地址线构成。

- 数据线：承载信息。
    - 数据线宽度反映一次能传送数据的位数。
- 地址线：传送内存地址。
    - 地址线宽度反映最大寻址空间。
- 控制线：传输定时型号和命令信息。典型控制信号如下：

    - 时钟：用于总线同步
    - 复位：初始化所有设备
    - 总线请求：发出信号的设备要使用总线
    - 总线允许：接到信号的设备允许使用总线
    - 中断请求
    - 中断回答：某个中断请求已被接受
    - 存储器读
    - 存储器写
    - I/O读
    - I/O写
    - 传输确认：数据已被接受 / 已送总线

性能指标

- 总线宽度
- 总线工作频率（等于时钟频率的1 / 2 / 4倍，取决于一个时钟周期传送多少数据）
- 总线带宽（最大数据传输速率）
    - （对于同步总线）总线带宽 = 总线宽度 * 总线时钟频率 / 传送一次数据所用时钟周期
- 总线寻址能力（可寻址地址空间）
- 总线定时方式
- 总线传送方式
- 总线负载能力

• 基于总线的互连结构 （主要模块以及连接的总线）

- I/O总线
    - USB控制器、接口
        - USB设备
    - 网卡
    - 磁盘控制器
        - 磁盘
    - PCI接口
        - PCI设备
    - 南桥芯片
        - 北桥芯片
            - CPU（前端）总线
                - CPU总线接口
            - 存储器总线
                - 主存储器
            - 显卡

• I/O接口的职能、通用结构

职能

- 数据缓冲（匹配二者工作速度）
- 错误 / 状态监测（状态寄存器）
- 控制和定时（接受控制 / 定时信号）
- 数据格式转换
- 和主机、设备通信

通用结构

###

- 数据缓冲寄存器（传送数据）
- 状态 / 控制寄存器（记录设备的状态信息；接受CPU的控制信号）
- 地址译码、I/O控制逻辑

• I/O端口的独立编址方式、统一编址方式

将I/O的端口和主存空间统一编址，将主存空间分出一部分地址为I/O端口编号。

独立编址方式

单独将I/O端口编号，成为一个独立的I/O地址空间（需要专门的I/O指令）。

• 程序直接控制I/O方式、中断控制I/O方式、DMA方式的工作原理、区别

程序直接控制I/O方式

- 无条件传送：定时对简单外设进行数据传送
- 条件传送（轮询）：操作系统主动查询I/O数据
    - I/O设备将自己的状态放在状态寄存器中，操作系统定时查询该寄存器的内容

中断I/O方式

如果一个IO设备需要CPU的干预，则通过中断请求来通知CPU.

CPU中断当前程序的执行，调用中断处理程序执行。

处理结束后，返回被中止的程序继续执行。

直接存储器存取方式（Direct Memory Access）（磁盘等高速外设专用）

IO设备直接和主存进行大批量数据交换。

需要专门的DMA控制器控制总线来完成数据传送：

- 外设准备好数据，向DMA发DMA请求信号
- DMA控制器向CPU发出总线请求信号
- CPU让出总线，DMA控制总线进行传输；不需要CPU干涉

• 中断响应、中断处理 中断优先权的动态分配

中断系统的基本职能

- 记录中断信号
- 自动响应中断请求
- 自动判优（判断中断优先级，响应优先级最高的中断）
- 保护被中断程序的断点和中断现场
- 中断屏蔽（实现多重中断的嵌套执行）

中断处理

- 中断检测
- 中断响应（调出响应的中断处理程序），此时禁止中断
- 中断处理
    - 不同中断源的中断处理程序不同。
    - 具体的多重中断处理过程：
        - 准备阶段：（禁止中断，不允许被中断打断）
            - 保护现场和旧屏蔽字
            - 查明中断原因
            - 设置新屏蔽字
            - 开中断
        - 本体阶段（具体的中断处理）（此时允许中断，可以被新的优先级更高的中断打断）
        - 恢复阶段：（禁止中断，不允许被中断打断）
            - 关中断
            - 恢复现场和旧屏蔽字
            - 清理中断请求
            - 开中断
            - 中断返回
    - 单重中断不允许处理时被新的中断打断，因此不需要设置中断屏蔽字。

中断优先级

- 中断响应优先级：由 **查询程序 / 排队线路**决定的优先权；反映 **多个中断同时请求时选择哪个响应。**
- 中断处理优先级：由 **各自的中断屏蔽字** 来动态设定，反映 **该中断和其他中断之间的关系。**

中断屏蔽字

每个中断都有自己的中断屏蔽字，用来说明该中断不能和哪些中断同时处理。

处理优先级高的中断会屏蔽处理优先级低的中断。

例：中断a, b, c, d的响应优先级为a>b>c>d, 处理优先级为a>d>c>b, 若同时发生a, c, d中断，且c中断处理程序会引发b中断，那么中断的处理顺序为：

- 按照响应优先级先响应a中断；因为a中断的处理优先级也是最高，因此无法处理其他中断，直到a中断处理结束。
- 按照响应优先级响应c中断；c中断的处理优先级低于d中断，因此在c中断的处理过程中同时开始d中断的处理。由于b中断的处理优先级最低（被c屏蔽），因此不处理b中断。
- d中断的处理结束。

- c中断的处理结束，响应b中断。
- b中断的处理结束，中断处理过程结束。

• 3种DMA方式：CPU停止法、周期挪用法、交替分时访问法

CPU停止法

DMA传输时，CPU脱离总线并停止访问主存，直到DMA传送一块数据结束。

- 控制简单，适用于传输率很高的外设
- CPU在传输过程中基本无法工作
- 主存周期没有被充分利用（存储周期快于I/O设备的数据准备时间）

周期挪用法

DMA传输时，CPU让出一个总线周期，在该周期内由DMA控制总线来访问主存，传送完数据后释放总线。

- 既能及时响应I/O请求，CPU和主存效率又较高（适用于I/O设备的读写周期大于主存周期的情况）
- 每次访存都要申请、占用、释放总线，增加传输开销。
- I/O设备要求DMA传送时，可能会碰到三种情况：
    - CPU不需要访问主存：不发生冲突
    - CPU正在访问主存：需要等到存储周期结束
    - CPU正要访问主存：发生访存冲突，此时 **DMA的优先级高于CPU**。

)交替分时访问法

每个存储周期分为两个时间片，分别给CPU和DMA.

• I/O子系统层次结构、每层的基本功能

I/O软件被组织为四个层次。层次越低，则越接近设备，远离用户。

- 用户层I/O软件（IO函数调用、系统调用）
- 和设备无关的操作系统I/O软件
- 设备驱动程序
- I/O中断处理程序

后三者属于操作系统的一部分。

从用户I/O软件切换到内核I/O软件的唯一办法是系统调用（自陷）。

与设备无关的I/O软件

一般包含以下内容：

- 设备驱动程序统一接口
    - 将设备抽象为文件，和文件具有相同接口
- 缓冲处理
    - 管理I/O对于内核缓冲区的使用
- 错误报告
    - 向用户报告在内核态发生的错误
    - 直接返回编程错误（如请求了不可能的I/O操作等）

- 打开、关闭文件
- 逻辑块大小处理

设备驱动程序

略

中断服务程序

略

• 用户程序、C语言库、内核之间的关系

用户程序通过某种I/O函数 / I/O操作符（C语言库函数）来请求I/O操作；

（I/O函数、I/O操作符）通过操作系统内核提供的系统调用来实现I/O。