计算机体系结构期末复习

L1-课程概要与体系结构基础

硬件组成与架构发展

- **不同类型计算机硬件组成**
  - **个人计算机**：以联想 ThinkCentre M8300t 台式机为例，其主机箱内包含多种 PCIe 外设插卡、内存条 DIMM、主板及内存插槽、数据线缆、中央处理器 CPU（风扇及散热片覆盖）、机械硬盘 HDD、电源模块 Power Supply、固态硬盘 SSD 及供电线缆等。这些组件协同工作，实现计算机的基本功能，如数据存储、处理和传输。
  - **高性能计算机（服务器）**：如华为 RH2285 型号机架式服务器，内部有两颗高性能 CPU（散热片覆盖）、主板、内存条 DIMM 及内存插槽、扩展板卡（PCIe FPGA）、风扇、磁盘阵列等。由于服务器需要提供高性能的计算、存储等服务，所以在稳定性、可靠性、安全性、可扩展性、可管理性等方面有较高要求，其硬件配置和架构设计都围绕这些需求展开。
  - **单板计算机（YSYX 单板计算机）**：包含 SoC 芯片、按键、选择开关、各种接口（如 FPGA JTAG 接口、外设切换开关、PL VGA、PL PS/2、PLIO 等）、晶振插座、电源模块（如电源 LDO、电源指示 LED）、烧写器切换、电平转换芯片、PS PMOD 扩展、耳机输出、音频输入、复位按键、时钟拨码开关、TF 卡插座、板载烧写器、Flash 插座、SoC UART 等众多组件。这些组件集成在一块单板上，适用于一些特定的应用场景，如嵌入式系统开发等。
- **计算机发展历程中的重要机型**
  - **ENIAC**：世界第一台电子计算机，于 1946 年诞生。它采用手动编程，通过设置开关、连接插头和插座、插拔电缆来实现程序控制。每位数值采用十进制表示，乘法运算速度为 357 次/秒，除法为 38 次/秒，有 20 个 10 位寄存器（无存储器），加法速度达 5000 次/秒。一个十进制数用 10 个电子管表示，由 40 个 9 英尺（2.74 米）高机柜组成，占地面积 167 平方米，重达 30 吨，功耗 150KW，还包含 70000 个电阻、10000 个电容、17468 个电子管、6000 个手动开关、1500 个继电器和 500 万个焊点。其设计上的不足促使了后续计算机技术的改进和发展。
  - **中国的 103 机**：中国第一台通用数字电子计算机，诞生于 1958 年。它在我国计算机发展历程中具有重要意义，标志着我国在计算机领域的起步和探索，为后续计算机技术的研究和应用奠定了基础。
  - **长城 0520C - H 台式电脑**：中国第一台国产微型计算机，于 1985 年推出，是中国计算机发展的重要里程碑，推动了计算机在国内的普及和应用。
- **冯·诺依曼结构**
  - **组成部件**：由 运算器、控制器、存储器、输入设备和输出设备五个基本部件组成。运算器负责进行算术和逻辑运算；控制器控制计算机各部件协调工作，自动执行指令；存储器用于存放数据和指令；输入设备如键盘、鼠标等用于向计算机输入信息；输出设备如显示器、打印机等用于输出计算机处理结果。
  - **工作方式**：采用“存储程序”工作方式，即将程序和数据以二进制形式存放在存储器中，计算机在运行时自动从存储器中逐条取出指令并执行。这种工作方式使得计算机的操作更加自动化和高效，成为现代计算机的基本工作原理。

- **主要思想**：计算机内部以二进制表示指令和数据，每条指令由操作码和地址码两部分组成。操作码指出操作类型，地址码指出操作数的地址。存储器不仅能存放数据，还能存放指令，且计算机能够区分数据和指令。控制器能自动执行指令，运算器能进行加、减、乘、除等基本算术运算及一些逻辑运算和附加运算，操作人员通过输入输出设备与主机通信。

性能评估与定律

- **计算机硬件技术指标**
  - **机器字长**：CPU 一次能处理数据的位数，与 CPU 中的寄存器位数有关。例如，32 位 CPU 的机器字长为 32 位，它决定了计算机在一次运算中能够处理的数据量和精度，较长的机器字长通常能提高计算效率和精度，但也会增加硬件成本和复杂性。
  - **主频**：数字硬件的操作由恒定速率时钟控制，主频决定了 CPU 的时钟周期频率。例如，4.0GHz 的主频意味着 CPU 每秒可以执行 4.0×10^9 个时钟周期。主频越高，CPU 在单位时间内可以执行的操作次数越多，但过高的主频也可能带来散热等问题，并且主频的提升并非无限制地提高计算机性能，还受到其他因素的制约。
  - **运算速度**：包括 CPI（Cycles per Instruction，执行一条指令所需时钟周期数）、IPC（Instructions per Cycle，一个周期内执行的指令数）、MIPS（每秒执行百万条指令）和 FLOPS（每秒浮点运算次数）等指标。这些指标从不同角度衡量了计算机的运算能力，例如 MIPS 常用于衡量整数运算速度，而 FLOPS 则更侧重于浮点运算性能。在实际应用中，不同的应用场景对运算速度的不同方面有不同的需求，如科学计算更关注 FLOPS，而一般的办公应用则对 MIPS 较为敏感。
- **影响计算机及集成电路发展的定律**
  - **摩尔定律**：由 Gordon Moore 提出，集成电路上可容纳的晶体管数目约每隔两年便会增加一倍，同时成本仅有极小幅度的上升。这一定律在过去几十年间一直是计算机和集成电路行业发展的重要指导原则，促使芯片性能不断提升，推动了计算机技术的快速发展，使得计算机在体积不断缩小的同时，性能得到极大提高，如个人电脑从早期的大型机逐渐演变为如今的轻薄笔记本和小型台式机。
  - **登纳德缩放比例定律**：由 Robert H. Dennard 提出，即使在集成电路芯片上放置更多的电路，冷却问题基本不变，即功耗密度保持恒定。这一定律在集成电路设计中具有重要意义，使得芯片制造商在提高芯片集成度的同时，能够较好地控制功耗，为计算机性能的提升提供了支持。但随着技术的进一步发展，该定律在一定程度上受到了挑战，如在芯片制程进入纳米级别后，漏电等问题导致功耗难以按照该定律继续保持稳定。
  - **阿姆达尔定律**：关注系统性能提升的短板问题。它指出通过使用某种较快的执行方式所获得的性能提高，受可使用这种较快执行方式的时间所占的百分比例的限制，即通过更快的处理器来获得加速会受到慢的系统组件的制约。例如，在一个应用程序中，如果只有部分代码能够并行执行，即使增加处理器核心数量，整体性能的提升也会受到不能并行部分的限制。该定律在计算机系统设计和性能优化中具有重要指导作用，提醒设计者在提升系统性能时要综合考虑各个组件的性能平衡。

$$\mathrm{ExTime}_{new} = \mathrm{ExTime}_{old} \times \left[\left(1 - \mathrm{Fraction}_{enhanced}\right) + \frac{\mathrm{Fraction}_{enhanced}}{\mathrm{Speedup}_{enhanced}} \right]$$

$$\mathrm{Speedup}_{overall} = \frac{\mathrm{ExTime}_{old}}{\mathrm{ExTime}_{new}} = \frac{1}{\left(1 - \mathrm{Fraction}_{enhanced}\right) + \frac{\mathrm{Fraction}_{enhanced}}{\mathrm{Speedup}_{enhanced}}}$$

数的表示与运算

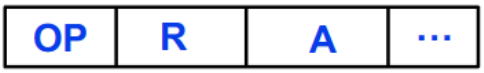
- **数的表示方法**
  - **无符号数**：寄存器的每一位均可存放数值，对于 n 位寄存器，其表示范围为[0, 2^n - 1]。例如，8 位无符号数的表示范围是 0 到 255，16 位无符号数的表示范围是 0 到 65535。无符号数在一些不需要表示负数的场景中使用，如存储内存地址、计数等。
  - **有符号数**
    - **真值与机器数**：真值是带符号的数，而机器数是符号数字化的数。例如，+0.1011 和 -1100 是真值，而 0 1011 和 1 1100 是对应的机器数（这里以某种假设的表示方式为例），其中最高位为符号位，0 表示正数，1 表示负数。
    - **原码表示法**：对于整数，当 $2^n > x \geq 0$ 时，[x]原 = 0,x；当 $0 \geq x > -2^n$ 时，[x]原 = 2^n - x。例如，x = +1110 时，[x]原 = 0,1110；x = -1110 时，[x]原 = 2^4 + 1110 = 1,1110。原码表示法简单直观，但在进行加减运算时较为复杂，需要考虑符号位的处理。
    - **补码定义**：当 $2^n > x \geq 0$ 时，[x]补 = 0,x；当 $0 > x \geq -2^n$（mod 2^n + 1）时，[x]补 = 2^n + 1 + x。例如，x = +1010 时，[x]补 = 0,1010；x = -1011000 时，[x]补 = 2^7 + 1 + (-1011000) = 10000000 - 1011000 = 1,0101000。求补码的快捷方式是当真值为负时，补码可用原码除符号位外每位取反，末位加 1 求得。补码在计算机运算中具有重要作用，它使得减法运算可以转化为加法运算，简化了硬件设计。
- **算术逻辑单元（ALU）**
  - **功能与指令关系**：ALU 是指令执行的关键部分，负责执行算术和逻辑运算。在 MIPS32 指令集中，不同的指令如 lw（load word）、sw（store word）、beq（branch on equal）、slt（set on less than）等会触发 ALU 执行相应的操作，如加法、减法、逻辑与、逻辑或等。例如，执行 lw 指令时，ALU 可能需要计算内存地址，涉及加法操作；执行 beq 指令时，ALU 要进行相等比较，可能涉及减法操作并判断结果是否为 0。
  - **设计与内部结构**：以 32 位 ALU 为例，它可以由多个 1 位 ALU 组成。1 位 ALU 可以执行 AND、OR、ADD 等基本操作，通过进位链（CarryIn 和 CarryOut）连接各个 1 位 ALU 实现 32 位数据的运算。在进行减法运算时，通过对减数进行求补操作（按位取反并加 1）来实现，同时要注意区分有符号数和无符号数运算时溢出和进借位标志的处理。
  - **溢出和零检测逻辑**：对于溢出检测，在 N 位 ALU 中，Overflow = CarryIn[N - 1] XOR CarryOut[N - 1]，即当最高有效位的进位输入和进位输出不同时，表示发生了溢出。对于零检测逻辑，通过将所有输出位进行或运算后取反来判断结果是否为 0，若结果为 0，则表示运算结果为 0。这些检测逻辑在保证运算准确性和程序控制流程中起到重要作用，例如在条件分支指令中，根据运算结果是否为 0 来决定是否进行分支跳转。
  - **Slt:Set-on-less-than**：做减法，最高符号位与 Overflow 进行异或操作之后供给最低位 ALU 的 Less 输入端
  - 溢出标志 Overflow 表示有符号数加减法操作结果是否超出范围/结果的符号是否符合预期，进借位标志 CarryOut 表示无符号数加减法操作结果是否超出范围/结果能否完整地保存

指令系统相关

- **机器指令与指令系统概念**：机器指令是每一条机器语言的语句，而指令系统是所有机器指令的集合。指令系统处在软/硬件交界面，它既被硬件设计者用于设计计算机硬件，确保硬件能够正确执行指令；又被系统程序员看到，系统程序员需要依据指令系统编写汇编程序，并且只有熟悉计算机硬件实现才能写出高效代码。指令系统的设计好坏直接影响计算机的性能和成本，例如一个设计良好的指令系统可以提高程序的执行效率，减少指令执行的时间和硬件资源的消耗；而不合理的指令系统可能导致程序复杂、执行效率低下。

- **冯·诺依曼结构机器指令规定**：指令用二进制表示，和数据一起存放在主存中。指令由操作码和操作数（或其地址码）两部分组成，操作码定义操作类型，操作数（或其地址码）指示操作源和/或目的地。例如，在一个简单的加法指令中，操作码可能表示加法操作，操作数地址码则指定要相加的两个数在内存中的位置。这种规定使得计算机能够按照预定的程序逻辑自动执行指令，实现各种计算任务。
- **数据存储与寻址方式**
  - **数据存储顺序**：包括小端方式（Little Endian）和大端方式（Big Endian）。小端方式下，LSB（最低有效位）所在的地址是数的地址，即字地址或数的地址上存放数据的 LSB；大端方式下，MSB（最高有效位）所在的地址是数的地址，字地址或数的地址上存放数据的 MSB。例如，对于整数 -65535，在内存中存储时，若采用小端方式，其 4 个字节的存储顺序为[01 FF FF FF]（从低地址到高地址）；若采用大端方式，则存储顺序为[FF FF FF 01]。有些机器两种方式都支持，可通过特定控制位来设定采用哪种方式，如 ARM、MIPS 等。不同的数据存储顺序在数据传输和处理过程中可能会产生影响，特别是在多字节数据的处理和网络通信中需要特别注意。
  - **寻址方式**：包括立即寻址、直接寻址、间接寻址、寄存器寻址、寄间接寻址、偏移寻址、堆栈寻址等。立即寻址的操作数直接包含在指令中，执行速度快但操作数幅值有限；直接寻址的有效地址等于指令中的地址码，计算简单但地址范围有限；间接寻址通过地址码指向的内存单元获取操作数地址，地址范围大但需要多次存储器访问；寄存器寻址的操作数在寄存器中，指令执行快且指令短，但寄存器数量有限导致地址范围有限；寄间接寻址结合了寄存器和间接寻址的特点，地址范围大但需要额外的存储器访问；偏移寻址通过寄存器内容和偏移量计算有效地址，较为灵活但计算相对复杂；堆栈寻址以栈顶为有效地址，指令短但应用场景有限。这些寻址方式为程序员提供了多种访问数据的手段，在编写程序时可以根据数据的特点和程序的需求选择合适的寻址方式，以提高程序的性能和效率。

假设：A=地址字段值，R=寄存器编号，
EA=有效地址，(X)=X中的内容



方式	算法	主要优点	主要缺点
立即 #	操作数=A	指令执行速度快	操作数幅值有限
直接	EA=A	有效地址计算简单	地址范围有限
间接 @	EA=(A)	有效地址范围大	多次存储器访问
寄存器	操作数=(R)	指令执行快，指令短	地址范围有限
寄间接	EA=(R)	地址范围大	额外存储器访问
偏移	EA=(R)+A	灵活	复杂
堆栈	EA=栈顶	指令短	应用有限

偏移方式：将直接方式和寄存器间接方式结合起来
有：基址BR / 变址IX / 相对(*)PC 三种
MIPS不区分基址还是变址，统一为偏移寻址方式

- **MIPS 指令系统**
  - **指令格式**：MIPS 指令长度为 32 位，有三种典型格式，即 R - type、I - type 和 J - type。R - type 格式包含 6 位操作码（op）、5 位源寄存器（rs）、5 位目标寄存器（rd）、5 位位移量（shamt）和 6 位功能码（funct），用于执行如加法（add）、减法（sub）、逻辑与（and）、逻辑或（or）、小于设置（slt）等操作；I - type 格式包含 6 位操作码、5 位源寄存器、5 位目

标寄存器和 16 位立即数，用于数据传输和条件分支指令，如 lw（load word）、sw（store word）、beq（branch on equal）等；J - type 格式包含 6 位操作码和 26 位目标地址，用于无条件跳转指令。

- **寄存器设置**：MIPS32 架构定义了 32 个 32 位通用目的寄存器（GPRs），编号从 r0 到 r31，其中 r0 硬连线为 0。还有一对特殊用途寄存器 HI 和 LO，用于保存整数乘法、除法和乘加运算的结果。此外，还有程序计数器 PC，它不是架构可见寄存器，但在指令执行过程中起到关键作用，指示下一条要执行的指令地址。这些寄存器的设置为指令的执行提供了数据存储和操作的空间，不同的寄存器在不同的指令和运算中发挥着特定的作用，合理使用寄存器可以提高程序的执行效率。

L2-复杂流水线和乱序执行(上)

该文档围绕计算机体系结构中处理器相关知识展开，涵盖了从基础概念到不同类型处理器设计，再到流水线处理器关键问题及解决方法等内容，是计算机体系结构课程的重要学习资料。

1. **计算机体系结构基础与简易处理器设计**
   - **基础回顾**：包括冯·诺依曼结构运行机理、程序计数器作用、计算机性能评估、数的表示及运算、指令系统等内容，这些构成了后续处理器设计的理论基石。
   - **MIPS 寄存器与指令**：MIPS 寄存器有 zero、at、v0 - v1 等多种类型，各自具有特定功能，如 zero 恒为 0，at 为汇编程序保留等。要实现的指令涵盖算术逻辑（add、sub、and、or、slt）、控制流（beq）、存储器引用（lw、sw）等类型，且这些指令在执行过程中都需使用 ALU。
2. **处理器通用机制与部件详解**
   - **通用机制步骤**：处理器首先利用程序计数器提供指令地址，接着从内存获取指令，随后依据指令精确控制寄存器的读写以及操作的执行，这一系列步骤是处理器工作的基本流程。
   - **部件功能特性**
     - **组合部件**：像与门（$Y = A & B$）、加法器（$Y = A + B$）、多路复用器（$Y = S? I1 : I0$）、算术逻辑单元（$Y = F(A, B)$）等，其输出是输入的函数，部分组合部件的操作受时钟信号控制，在数字电路中发挥着关键作用。
     - **存储部件**：寄存器具有 N 位输入输出和写使能端，写使能信号控制数据写入，仅在时钟上升沿且写使能为 1 时更新。寄存器堆由 32 个寄存器组成，通过地址选择进行读写操作，内部通过一对多路复用器实现双读端口功能。内存有读写数据和地址总线，写操作受时钟影响，读操作在地址有效后经过短暂访问时间即可获得有效数据，类似组合逻辑。
3. **单周期处理器执行与数据通路剖析**
   - **指令执行流程**：以 add 加法指令为例，详细展示了在单周期处理器中的执行过程，包括从程序计数器获取指令地址，从寄存器堆读取操作数，经 ALU 运算后将结果写回寄存器堆等一系列操作，涉及到程序计数器更新、寄存器读写、ALU 运算和数据存储等关键步骤。
   - **各类指令通路**：分别深入介绍了 Reg - Reg 操作（如 add rd, rs, rt）、Load 操作（如 lw rt, rs, imm16）、Store 操作（如 sw rt, rs, imm16）、Branch 操作（如 beq rs, rt, imm16）和 Jump 指令的数据通路及相应控制信号。在 Branch 操作中，需读寄存器、比较操作数、计算目标地址等，控制信号由指令译码决定，如根据指令类型确定 ALU 操作、寄存器写使能、数据存储等控制信号的取值。
4. **单周期处理器控制器设计方法**
   - **译码表构建**：根据每条指令的功能，仔细分析控制信号的取值，并系统地在指令译码表中列出。例如，对于 R - format 指令，RegDst 为 1，ALUSrc 为 0 等，通过这种方式明确不同指令

与控制信号之间的对应关系。

- **逻辑表达式生成**：依据列出的指令和控制信号关系，运用逻辑推理和电路设计知识，写出每个控制信号的逻辑表达式。在 ALU 控制中，采用多层解码方式，根据 opcode 和 function code 对 ALU 操作进行解码，如 00 对应 lw、sw 的 add 操作，01 对应 beq 的 subtract 操作等，这种多层解码有助于减小主控单元规模、提高控制速度，从而优化控制器设计。

5. **多周期处理器设计要点**
   - **设计动机与优势**：为解决单周期 CPU 因最长指令周期决定时钟周期而导致的效率问题，多周期 CPU 将指令执行分解为多个较小任务，每个任务占用一个时钟周期，不同指令所需周期数不同，且可复用功能单元，如 ALU 和内存，有效提高了硬件资源的利用率。
   - **结构与控制实现**：在结构上，多周期 CPU 使用单一内存单元处理指令和数据，采用单一 ALU，并在主要功能单元后添加寄存器暂存输出结果，直到后续时钟周期使用。其控制单元通过有限状态机（FSM）实现，包括取指、译码、执行等多个状态，状态转换基于输入和当前状态，控制信号输出大多依赖当前状态（如 Moore FSM），这种设计方式能够灵活地控制指令执行流程，提高处理器的适应性和效率。
6. **处理器设计步骤与资源利用分析**
   - **设计步骤概述**：在确定指令集架构（ISA）后，处理器设计依次经过以下关键步骤：首先用 RTL（Register Transfer Language）详细分析每条指令的功能；接着根据指令功能确定所需元件并规划时钟方案；然后将数据通路进行互连，构建起数据传输的通道；再确定每个元件所需控制信号的取值，并汇总形成指令与控制信号关系表；最后依据该表得出每个控制信号的逻辑表达式，进而设计出控制器电路。在这个过程中，涉及到数据通路和控制器的设计，数据通路包含操作元件（由组合逻辑电路实现）和存储（状态）元件（由时序逻辑电路实现），两者相互协作，共同完成处理器的功能。
   - **资源利用对比**：在资源利用方面，单周期 CPU 以最长的 Load 指令执行时间确定时钟周期，当执行其他短指令时，在时钟周期后半阶段会整体处于闲置状态，造成硬件资源浪费。多周期 CPU 内部部件并非在每一拍都处于忙碌状态，存在一定的空闲时间。相比之下，流水线 CPU 旨在提高吞吐量和硬件资源利用率，但在实际应用中，由于指令执行的复杂性，难以达到理想的流水线状态，存在如指令执行阶段不均衡、资源冲突等问题，需要进一步的优化和改进。
7. **流水线处理器核心知识**
   - **理想与实际流水线**：理想流水线的目标是在增加少量成本（主要是硬件成本）的情况下显著提高吞吐量，其特点是重复相同且独立的操作，并且这些操作可均匀划分为无资源共享的子操作。然而，在实际的指令流水线中，由于指令的多样性和复杂性，难以满足这些理想条件。实际流水线中，非流水线版本和 k - stage 流水线版本的吞吐量受寄存器延迟影响，成本随寄存器数量增加而增加，如非流水线版本的吞吐量公式为 $BW = 1 /(T + S)$（S为寄存器延迟），k - stage 流水线版本为 $BW_{k - stage}=1 /(T / k + S)$；成本方面，非流水线版本为 $Cost = G + R$（R为寄存器成本），k - stage 流水线版本为 $Cost_{k - stage}=G + Rk$。
   - **设计问题与停顿原因**：流水线处理器设计面临诸多挑战，需要平衡各阶段工作，确保每个阶段的工作量和执行时间合理。同时，要妥善处理数据、控制和资源依赖问题，例如数据依赖包括 Flow 依赖（真数据依赖，即写后读 RAW）、Anti 依赖（读后写 WAR）和 Output 依赖（写后写 WAW），其中 Flow 依赖必须保证程序语义正确，而 Anti 和 Output 依赖是由于架构寄存器数量有限导致的。此外，还需处理长延迟操作和异常中断等情况。停顿是流水线运行中的常见问题，原因包括资源依赖（如寄存器文件、内存和功能单元的资源冲突）、数据依赖和控制依赖等，这些问题会导致流水线停止运行，影响处理器性能，需要通过相应的技术手段加以解决。
   - **数据依赖处理策略**：针对数据依赖问题，有多种处理方法。Anti 和 Output 依赖相对较易处理，可通过在最后阶段按程序顺序写入目的地来解决。对于 Flow 依赖，有五种基本处理方

式：一是检测并等待数据在寄存器文件中可用；二是在软件层面检测并消除依赖，无需硬件检测；三是检测并通过转发或旁路数据给相关指令，其核心思想是在结果产生后直接将数据送到需要的功能部件，而非经过寄存器文件，实现时需要增加依赖检查逻辑、数据转发路径和冲突检测电路来控制选通器；四是预测所需值并进行推测执行，然后验证结果；五是采用细粒度多线程技术，无需检测依赖。这些方法各有优缺点，在实际应用中需要根据具体情况选择合适的策略来优化流水线性能。