**COD**

- 考试类型：
  - 答疑：1.11 下午两点到五点 信电楼307
  - 开卷可带A4纸，可手写正反面，可带计算器
  - 分值：60(选择)+10(判断)+30(问答)，附录有指令格式（六个格式）
- L2:
- L3:
- L3-Revision：
  - Part1: Abstraction
  - Part2: Data transfer
  - Part 3: Computer decision making
  - Part 4: Logical Instructions
  - Part 5: RISC-V assembler features
  - Part 6: Values saved to restore after function call
  - Part 7: memory allocation
- L4（10.10）
- L4-REVISION
  - 1. R
  - 2. I：
  - 3. S（used for Stores）
  - 4. B
  - 5. U
  - 6. J
  - 7. CPU performance metrics
  - errors analysis
- L5 10.17 datapath
- L5-Revision
- L6 10.24 clocking methodology & control
- L6 Revision
  - 1. Flip Flops internal delays
    - 时钟周期需要考虑如下四种情况
    - 另外
  - 2. RegFile design
  - 3. Control-- Instruction timing
  - 4. ROM based control
- L7 pipeline (10.31)
  - 1. 流水化数据通路以及控制信号
  - 2. 提高处理器性能
- L7 Revision + textbook
  - 1. Pipeline  outline
  - 2. Datapath of pipeline
  - 3. Control of pipeline
  - 4. 流水线冒险的细节
- L8 cache (11.14)
- L9 cache & virtual memory（11.21）
  - cache
  - virtual memory

# COD

- made by xzq

## 考试类型：

## 答疑：1.11 下午两点到五点 信电楼307

## 开卷可带A4纸，可手写正反面，可带计算器

## 分值：60(选择)+10(判断)+30(问答)，附录有指令格式（六个格式）

- 虚存，cache看中文书
- 指令集，流水线看英文书
- 建议看完中文书后看英文书

# L2:

1、X0无法写入，一直为0；

2、lb（符号扩展X） VS lbu(零扩展)， sb x10, 3(x10) memory做字节使能，然后merge更新

loop: x8代表基地址，addi x9,x9,4 #字节寻址所以加4

sll（逻辑移位）: 左移后移出的丢弃，右边补零，右移同理，只不过方向相反。

算术移位：**右移时根据符号位进行符号拓展**，所以不是除法；算术左移还是补零

一般默认有符号位，寄存器里面就是存（地址的）值，不用考虑那个地址上具体的内容

# L3:

1、a0-a7（X10-X17） 寄存器用于参数传递，a0可以做返回值寄存器。PC寄存器存的是下一个指令的地址，一条指令占四个字节，所以一般顺序执行PC=PC+4；sp的值是由RISC-V确定的。pc=pc+8 不需要free内存因为可以覆盖

2、汇编中的函数调用过程：jr (jump register)，函数调用jal 常用，其实是先link 再跳转。

3、开栈以16个字节为单位，16，32这样开。小端规则

jal x0 -12

# L3-Revision：

## Part1: Abstraction

指令集说通俗点就是决定CPU的指令如何具体执行，MIPS，RISC-V等都是指令集的名称。

汇编语言没有变量，操作数就是一些寄存器。而且寄存器没有类型可言。寄存器特点是直接作用于硬件，故速度非常快，而且需要事先决定好寄存器的数量，在RISC-V中，32个寄存器不多不少刚刚好。经过查阅书本，在RISC-V架构下，**doubleword代表64位；word代表32位**。一个寄存器刚好存一个word=32 bits= 4bytes

关于具体的指令，有addi,但是没有subi，因为没有必要，可以用负的立即数来代替。这里讲X0寄存器比较出色。可以使用X0来执行空指令，这点很重要。

**小端规则**很重要，具体来说：(以字节为单位)低位字节存储在低位地址，高位字节存储在高位地址。跟正常的写法差不多。

## Part2: Data transfer

注意内存和寄存器的数据流转换：load from memory to register---lw时一个代表基寄存器（指向A[0]的指针），另一个代表地址的偏移量，以**字节**为单位,所以必须是4的倍数。store from register to memory: sw指令同理。同理还有lb,sb(P34),sbu 不是很懂再看看

## Part 3: Computer decision making

branch means change of control flow; conditional branch: `beq; bne; blt; bltu; bge`; unconditional branch: `j`

**注意比较的是两个寄存器，不能是立即数比较**

Example:

```
#example of if statement
addi x11,x0,5
addi x12,x0,6
addi x13,x0,1
addi x14,x0,2  # or addi x14,x0,1
bne x13,x14,Exit
add x10,x11,x12
Exit:
```

```
#example of if-else statment
addi x11,x0,5
addi x12,x0,6
addi x13,x0,1
addi x14,x0,2
bne x13,x14,Else
add x10,x11,x12
j Exit
Else: sub x10,x11,x12
Exit:
```

```
# example of loop
add x9,x8,x0 #x9 = &A[0]
add x10,x0,x0 #sum = 0
add x11,x0,x0 #i = 0
addi x13,x0,20 # x13 = 20
Loop:
bge x11,x13,End
lw x12,0(x9)
add x10,x10,x12
addi x9,x9,4 # increment is 4 not 1!!!
addi x11,x11,1
j Loop # Do not forget!!!
End:

```

# Part 4: Logical Instructions

`and; andi; or; ori; xor; xori; sll; slli; srl; srli`分别是按位与，按位或，按位异或，逻辑左移，逻辑右移

**andi used for mask**, **no not in RISC-V**(use xor 0xffffffff); all,srl都用0来补

arithmetic shifting(`sra; srai`) : **算术右移不是除法**

## Part 5: RISC-V assembler features

简记：

```
mv rd,rs # = addi rd,rs,0
ls rd,13 # = addi rd,x0,13
```

mv用变量赋值；li用立即数赋值

PC是datapath中的一个内部寄存器，里面存储有下一条指令的地址。当顺序执行的时候，PC每次加4个单位字节

发生函数调用的时候: `jal`: jal FunctionLabel （jump的同时保存ra的值），从被调函数返回的时候：`ret = jr ra`

## Part 6: Values saved to restore after function call

stack is in memory, so need register to point to it; sp( x2 ) is the stack pointer。栈的增长是从地址大的地方往小的地方生长。所以push sp 减小，pop sp 增加。

**关于寄存器的约定**：由callee保存的寄存器sp,s0--s11都会确保函数返回的时候值不变。a0-a7由caller保存，可以作为函数的参数传进去，可以由callee随意改变同时a0,a1可以作为返回值。

最后记得除了把push的值pop出来，还要注意sp指针的位置复原。

## Part 7: memory allocation

static: 直到程序运行结束才回收，用来保存C的全局变量

heap：用malloc动态分配内存产生的变量

stack：程序运行过程中用来保存寄存器值

text segment：程序段，用来存储程序的机器码，并且有PC指针指向

# L4（10.10）

所有指令长度都是32位，funct7,funct3 分别表示7位和3位的功能码 rs2，rs1用于寄存器寻址，32个寄存器。

介绍了码表格式，指令格式尽量保持一致是他的目的。RISC-V 增加了立即数产生的复杂性。branch offset 4bytes 为单位。branch 立即数的产生异常复杂。2.提高CPU性能的操作。

没有lwu,因为不存在补位操作。

对于寄存器的约定，不同的架构约定不同。

# L4-REVISION

指令同样是以32bits的word形式存储，同时指令被分成很多个数据场。RISC-V一共有6种指令格式。

`R, I, S, B, U, J`

注意一共32个寄存器x0~x31, 通过5bits来唯一确定。

## 1. R

## 2. I：

- 与R相比，funct7（7）和rs2（5） 被imm[11:0]替代。其余不变。立即数范围[-2048, 2047]
- 立即数在被进行一次算术操作前需要被符号拓展为32位
- 比较特殊的是当发生算术或者逻辑的移位时，立即数的高7位用来判断时逻辑还是算术，低的5位用来确定具体的位数。
- load指令同样也需要注意，最前面的立即数是用来作为offset（有正负）的，与基寄存器的地址值相加形成最终的地址！而目的寄存器中存放的则是具体的数值
- LH：对load下来的halfword进行符号拓展；LHU：对。。。进行零拓展。LW不需要拓展！

## 3. S（used for Stores）

- 注意store和load 的区别在于store没有目的寄存器
- store为什么把rd换成立即数而不是其他的原因在于为了尽可能和其他保持一致。

## 4. B

- 指令是被存在内存中被称为Code/Text的地方
- 当前指令的地址被存在PC中
- **注意branch offset 的单位是两个bytes**，它其实是用12位的立即数来表示13位

## 5. U

- 注意LUI（写入高20位并清空低12位），配合ADDI能够创造32位的值。**注意一个意外情况**，当ADDI 的立即数最高位为1时，需要高20位减去1，因为立即数是符号拓展的。解决方案中li x10, 0xDEADBEEF 是需要软件来支持的
- AUIPC

## 6. J

- JAL自动把PC+4存到 rd 寄存器，立即数是作为PC的offset。**同样是以2bytes为单位进行offset计算**，与branch类似，同样是减少硬件开销的手段

## 7. CPU performance metrics

- performance = execution time 而我们的关注点是CPU时间，也就是跑程序的时间
- cycle：在CPU中我们通常用cycle这个术语，引申出cycle time（一个cycle多少秒）clock rate/frequency （每秒多少个cylce）
- execution time = clock cycles for program * cycle time = clock cycles for program / clock rate

- clock cycles = instructions * clock cycles per instruction
- execution time = instructions *CPI * cycle time =  instructions *CPI / clock rate
- **CPI是个平均的概念，同时CPI不能用于比较不同的ISA**
- clock rate != performance
- 平均CPI 的计算相当于把每条指令的CPI乘该指令使用的频率

## errors analysis

- sw指令基地址的offset直接就是计算出来的值，不需要做改动，且单位是byte，能被4整除

# L5 10.17 datapath

单节拍每拍执行一条指令

# L5-Revision

- S和B之间立即数的产生需要注意有点区别，因为B的话自增单位为2 bytes
- 几个指令结合上一章一起复习
  - JALR区别于JAL；JALR的PC=rs+immediate，同时原rd=PC+4；JAL的PC是通过PC=PC + offset
  - LUI：将立即数作为目标寄存器的高20位写入，低12位置0
  - AUIPC：将PC值加上立即数作为存到目标寄存器

# L6 10.24 clocking methodology & control

时钟方法：

- 边沿触发，上升沿。
- 主要三个要点
  - 正常工作对T的要求
  - memory文件
  - 寄存器的两读一写的设计
- 芯片设计需要同步的memory

# L6 Revision

## 1. Flip Flops internal delays

### 时钟周期需要考虑如下四种情况

- t_setup:时钟上升沿到来前的一段时间（需要使得被采样信号稳定一段时间）
- t_clock-to-Q: 上升沿之后需要过一段时间Q才会做出变化
- t_CL: 组合电路的延迟时间
- clock skew 考虑时钟传播到不同地点的时间不同，应当考虑最坏情况下的clock skew

### 另外

- t_hold: 时钟需要达到上升沿之后的一段时间
    - 被采样信号需要保持稳定在t_setup + t_hold 的时间内。
    - pdf上的例子需要掌握

## 2. RegFile design

- 寄存器两读一写的设计
- wd,ws.. 分别对应几根线连接

## 3. Control-- Instruction timing

- IF, ID, EX, MEM, WB

## 4. ROM based control

# L7 pipeline (10.31)

## 1. 流水化数据通路以及控制信号

- **控制信号也是伴随着数据的往后传，直到要用到控制信号的时候再发挥作用（重要），这样方便处理异常与中断**
- 双端口memory 的问题在于对于同一个寄存器既读又写不知道先做哪个，而对于寄存器文件的话**约定前半写后半拍读**。
- 结构或内容的冲突
    - 停顿，加空泡
    - forwarding转发
- 数据转发需要判断
    - 源和目的是否相同
    - 是否需要写
    - 是否是X0寄存器
- 数据的冒险
- BTB表
- 动态转移预测器要了解，因为动态比静态更加常用

## 2. 提高处理器性能

- 提高时钟频率
- 流水线
- 超标量的处理器（了解），多发射

# L7 Revision + textbook

## 1. Pipeline outline

- 指令执行时间（流水线化）= 指令执行时间（非流水线化）/ 流水线的级数

    - 但由于流水线各级并非完全平衡，加上一些额外开销，所以实际情况流水线实际获得的加速比小于流水线级数
- 注意流水线可能会引入一些额外的开销
- 流水线所带来的性能的提升是通过提高吞吐率来实现的，而不是通过减少单条指令的执行时间！
- 流水线的冒险
    - 结构冒险：硬件不支持多条指令在同一时钟周期内执行（eg: IMEM, DMEM）
    - 数据冒险：一个操作必须等到另一个操作结束后才能执行
        - 解决方案：数据转发
- **流水线阻塞（又称气泡）**
- 分支冒险：
    - 解决方法一：阻塞
        - 解决方法二：预测（计算机采用的方法）
        - 静态预测
            - 动态预测，花费一定代价来存储历史记录
        - 解决方法三：延迟决定，先做不需要预测的事（MIPS书上提到）

# 2. Datapath of pipeline

- 将数据通路分为五部分，采用五级流水线
    - IF：取指令
    - ID：指令译码，读寄存器堆
    - EX：指令执行或地址计算
    - MEM：数据内存访问
    - WB：写回
- 熟练掌握数据的流动以及**流水线寄存器(相当重要)**的输入输出
    - PC可以看作在指令集层次可见的寄存器，**在异常发生时，PC的内容必须被保存下来，但是流水线寄存器的内容可以被丢弃**
- **重要特性**：数据通路中的每一个逻辑元件，如指令内存、寄存器读取端口、ALU、数据内存以及寄存器写入端口都只能在流水线的单级中使用，否则会产生结构冒险
- lw指令必须把**寄存器号从沿着流水线寄存器一路往下保存**，最后再将写入寄存器号返回到寄存器堆

# 3. Control of pipeline

- 不需要单独的写信号的情况（每个时钟周期内都会做的事）：PC写信号，四个流水线寄存器的写信号
- 五组控制线：
    - 取指令：不需要控制信号
    - 指令译码/**读取寄存器堆**：不需要控制信号
    - 指令执行/**地址计算**：RegDst，ALUOp, ALUSrc
    - 内存访问: Branch, MemRead, MemWrite
    - 写回: MemtoReg, RegWrite

- 需要注意的是流水线方式的数据通路**不改变控制线的意义**，但是我们需要将9条控制线按流水线步骤进行**分组**
- 在指令译码阶段创建控制信息，并且在**拓展流水线寄存器使之包含控制信息**
- tip:在一个时钟周期的前半段和后半段分别进行写入和读取寄存器操作为目的时防止数据冒险

## 4. 流水线冒险的细节

- 检测数据冒险与转发对应的条件以及解决冒险的控制信号（硬件上对应**转发单元以及转发单元控制的转发多路复用器**）：
  - EX冒险：需要看第二遍
  - MEM冒险：需要看第二遍
- 数据冒险以及阻塞：在转发需要转发单元和转发多路复用器的前提下还需要一个**冒险检测单元**
  - 阻塞的具体做法：保存PC以及阻塞之前部分的流水线寄存器的值，阻塞的后半段执行nop
  - nop 的具体做法是：将9个控制信号均置0，不会对任何寄存器和存储器进行写入操作
- 分支冒险：
  - 假定分支不发生，若发生，丢弃指令。丢弃指令的具体操作：将最初的控制信号置0（与阻塞一致）+ 当分支到达MEM时必须清楚前面IF，ID，EX阶段的指令
  - 缩短分支的延迟（提前分支指令的决策）（需要看第二遍）：计算分支的目的地址+判断分支指令的跳转条件
  - 动态分支预测：实现方法是分支预测缓存

# L8 cache (11.14)

- 很重要，考试的大部分分数
- 主关联百分百搞懂

# L9 cache & virtual memory（11.21）

## cache

- **数制标准**：k代表1000，ki代表1024
- cache的成本比cpu的成本高
- cache存在地方很多不止内存，还有文件，网页，数据库等等

## virtual memory

- 分时运行

# REVISION+TEXTBOOK（cache+VM）

## 1. 基本概念

- 存储器技术：SRAM，DRAM，磁盘
- 块：层次结构中存储或不存储信息的最小单元称为块
- **低层向高层（高层速度更快）**发出数据请求：命中/缺失；命中率/缺失率

- 命中时间（访问高层存储器所需要的时间，包括判断的时间）/缺失损失（包括把相应的块从底层搬运到高层的时间+把信息块传送给处理器的时间）
- **重要！！！**：block size = line size = 2^b(b为字节偏移的位数，若一个内存中有4 words，那么16bytes，所以b等于4)

# 2. 块定位方式

## （1）直接映像

直接映像高速缓存（给主存每个word分配一个高速缓存**地址**最简单的方法：依据主存地址进行分配）

- 标记（高速缓存中）：高位拿来做标记，解决一对多的问题
- 有效位: 用来判断**有没有数据**
- 一定要熟练根据数据量计算高速缓存的大小？？？12
- 给出字节地址，求出对应的高速缓存中的哪一块？？？

## （2）全关联（适用于块数量较少）

## （3）组关联（重要）

- 作用：降低缺失率
- 块号mod（高速缓存中组的个数）得到对应的组，然后可以放到该组的任何位置
- n路组关联 = 关联度为n = 一个组内有n块
- 地址中的下标值用来选择包含所需地址的**组**，所以说关联度每增加1倍就使得下标减少一位而标记增加1位

# 3. 替换块的选择

## （1）最近最少使用法（LRU）

- 对2路组关联：在每组里单独设置一位用来指示哪个单元刚被访问过

# 4. 多级cache的性能评估

- 作用：多级cache降低缺失损失
- 定义：用较大的二级高速缓存来处理一级高速缓存的缺失，从而降低缺失损失
- 看书！！！ 还没搞懂

# 5. 高速缓存的性能评估

- 一个程序花费的总的时钟周期数为：处理器周期和存储器停顿的时钟周期
    - 存储器停顿周期取决于缺失率和缺失损失
    - 组关联降低缺失率
- 。。。。

# 6. 虚拟存储器

## （1）概念

- 定义：主存作为通常由磁盘实现的辅助存储器的高速缓存。

- 作用：允许多个程序之间有效安全地共享内存
- 本质：虚拟内存实现程序地址空间到物理地址的转换，这种转换处理加强了程序之间地址空间的保护
- 页：单位：一块被称为一页（**虚拟存储器和主存储器都被分成页**）
- 缺页：虚拟存储器的访问缺失被称为**缺页**（缺页引起控制权的转移，该转移由异常处理机制处理）
- 地址变换
- 如今，由虚拟存储器控制的存储器层次结构的两层分别是**DRAM和磁盘**
- 虚拟存储器的地址划分
    - 虚页号(virtual page number) 页内偏移(page offset)
    - 虚页号=>实页号（物理页号）
    - 物理页号构成物理地址的高位部分
    - 页内偏移不变，构成物理地址的低位部分，页内偏移域的位数决定了页的大小
    - 题目给出虚拟地址的大小是包含页内偏移的

### （2）特性

- 考虑到写操作时间太长，在VM中不采用写通过而是采用**写回**
- 页的存放和查找：借助页表，页表寄存器（指出页表在主存中的位置，该寄存器指向页表首地址）
- 缺页关键词：
    - 控制转移，异常处理，交换区，LRU
- TLB（加速地址变换）**TLB是页表的cache**
    - 需要注意的是查找TLB缺失分为两种情况：一种是一次TLB缺失；另一种则是一次缺页
        - TLB缺失只需要创建缺失的TLB表项
        - 一次缺页就需要把控制权转移给操作系统
    - 因为TLB是一个高速缓存，所以TLB需要有标记域
    - 目前采用多级TLB，IMEM和DMEM都采用TLB和页表

## 12.5

- 4 个 mode（user mode;kernel mode ） 特权指令
- exception 机制
- 系统调用：改变控制流

## 12.19

## 重点：

### 一、SMD

- 熟悉分类方法（SIMD，MIMD...）

### 二、FPU

- 浮点操作严格按照步骤来

- **需要注意的是**幂指数小的跟着大的来，小的进行移动

# 12.25 习题课

# 1.2 网课观看

## 1. 多核处理模型

- 各自独立的资源：数据通路+一级二级cache
- 共享的资源：主存+ 三级cache

## 2. 多线程

- 硬件支持，硬件线程1与2
- 逻辑线程

## 3. 线程级并行技术（TLP）

- 线程：指令序列，有自己的指令寄存器、处理器状态
- 多核：
  - 物理CPU
  - 逻辑CPU
  - 超标量（SMT）
- OPENMP为操作系统分配的软件线程
- 问题：
  - 同步问题：有限个访问者同时请求共享资源中的同一个元素（类比跑步竞赛，结果会不确定）
    - 解决方法：上锁，锁值置0解除锁
    - 但是还是会有同步的问题，但目前所学知识无法解决

## 4. 硬件同步

- 原子读，原子写
- openMP critical section 实现临界区，临界区一次只有一个指令可以进入
- 死锁：
  - 解决方法：银行家算法

## 5. SMP 共享内存多处理器

- 如何共享数据：所有处理器核心共享地址空间
- 每个核有私有cache
- cache一致性准则
- 避免伪共享的发生

- cache的缺失：
  - 强制缺失：冷启动
  - 容量缺失：
  - 冲突缺失：
  - 一致性缺失：和其他处理器之间为了保持cache一致性而导致的缺失，又称通信缺失

# 1.2 review

- 期末考试的难度在cache上
- 多处理器（可能考）
- 蓝色的重点考查
- CPI对于相同的ISA才能比较！！！
- 流水线关于冲突的选择和判断：三种冲突（特别是**数据依赖**和控制依赖）
- 基础概念：流水线只能增大吞吐率不能减少每条指令的延迟

# REVIEW

## cache：

- block在cache里就是数据，一个直接映像高速缓存所需的总位数是 2^n(下标个数)*（block位数+标记域位数+ 有效位位数）
- tag的位数随着关联度的增加而增加
- overhead-经费；开销
- 注意cache miss的三种情况：
  - 容量miss是因为cache本身太小，无法存储所有的数据。这时尽管采用最佳的替换策略仍旧无法避免miss
- 要求熟练计算一个高速缓存的总位数
- block 又名line
- 会计算高速缓存的性能（还没看过）
- 有m项的全相联高速缓存可简单看成一个m路组相连高速缓存
- 一般观察示意图，块/组 用行来表示； N路的路用列来表示，举个例子，共8个block，2路组相连，得到四行，两大列
- **一般来说题干出现cache的大小指的是cache中数据的总大小**
- dirty位用于写回策略，并且每个block分配一个dirty位。
  - **如果条目为脏，需要先将数据先回内存再替换条目**
- 关于写策略，见英文书P386:
  - write buffer的本质就是一个队列，存储着被等待写到主存中的数据。在把数据写到cache和buffer后，程序便可继续执行。当buffer中的数据全部被写到主存后，buffer被free
  - write生成的速度大于主存接收他们的速度，这时write buffer无济于事；相反的情况，可以通过增加buffer的深度来减少stall的频率