期末复习

> 记录于最后一节课，补充于6月16日前后

选择题10*2=20分（基本概念，简单计算）

30 = 组合逻辑+时序逻辑（课堂讲得内容，给题干，设计电路、时序分析等，参考作业）

50 = 一个大题，和指令系统、单周期、流水线CPU等（老师说和往年卷很像）

---

1. 冯诺依曼结构

此处知识点，请参考详细笔记第一章-二进制表示

五大部件：运算器、存储器、控制器、IO设备

存储程序的工作方式（程序和数据都存放在存储设备中，再取出执行）

1.1 不同层次语言之间的等价转换

`高级语言`-》编译得到`汇编语言`-》通过汇编程序得到`机器语言目标程序`-》指令译码器得到`控制信号`

每条指令都和操作码和地址码组成

高级语言可以编译成不同的汇编代码，汇编程序和机器指令是1对1的（和指令也是一对一的，由`ISA`决定）；

任何语言都是要最终通过执行机器指令来实现功能

2. ISA指令集体系结构

Instruction Set Architecture

ISA的核心是指令系统，包括指令格式、操作种类，寻址方式，寄存器大小，编址方式，大端小端等等。

3. 数据的编码

数值数据三要素：进制。浮点、定点。编码

整数:

- unsigned：原码
- signed：补码
- **整数的范围：编码方式和位数有关**

浮点数:

- IEEE754
- 符号位
- 尾数：定点原码
- 指数（阶码）：使用移码，有偏移常数
- **范围：和阶码位数和基的大小有关（尾数的影响很小）**

- **精度：和尾数的位数和是否规格化有关**

BCD码

- 用二进制表示0-9

格雷码

- 相邻两个只差一位

4. 数据的宽度和存储

什么是数据的地址

大端，小端?

5. 数字逻辑基础

逻辑门:

使用Cmos晶体管（含n-mos和p-mos，一对nmos和pmos称位cmos）

与非或非需要2对cmos，非门1对cmos，与门和或门是3对。

缓冲器、传输门、三态门

不会在考试中用布尔代数相关定理进行化简

真值表和逻辑表达式来描述逻辑变量

（***即真值表到表达式要很熟悉！！***）

一定会考卡诺图化简，不考代数法化简

等效逻辑符号、电路->使用与非和或非来构建电路，因为效率更高；

使用de-morgan将所有的逻辑表达式都化简为二级**与非**表达式

6. 组合逻辑电路

数字逻辑电路有组合逻辑电路和时序逻辑电路

组合逻辑电路的输出仅仅依赖于当前的输入

组合逻辑电路可以是多级，两级电路的

重点：参考作业题目进行复习！

1. 组合逻辑的设计： 功能分析-真值表-化简-画图-评价
2. 无关项的处理很重要

三态门和高阻态非法值

典型的组合逻辑部件：译码器等等

延迟分析：

- 传输延迟：关键路径上所有元器件的传输延迟

7. 时序逻辑电路

不需要学会复现SR、D锁存器等等

但是要熟悉其符号表示和功能，比如在试卷中出现图，需要认识，能够分析电路

SR锁存器

- S为置位端、R是复位端

这里省略其他锁存器，

时序逻辑电路不仅仅依赖于当前的

Mealy型：：输出=G(现态+输出)

Moore型：输出=G(现态)

时序电路的设计：功能分析-

状态的编码考试会给定！

未用状态分析(挂起、无法自启动) 是否无论从那个状态开始进入之后都可以进入正确的状态循环

定时分析！！（clk-to-q，setup，hold最大延迟决定了时钟周期）

典型时序逻辑电路部件（不会要求画图）

7.1 存储器的结构和基本概念

> 非重点，老师讲得很快

完全不用知道实现原理

寄存器用来存储少量数据，存储器整列存储大量数据

ROM、RAM

8. 运算部件的设计

ALU的实现：

- 加法器是基础（全加器，会计算标志位ZF,CF,OF等等知识点）
- 并行进位（不需要画图）但是要知道概念

算数运算：

- 定点运算

明确不考：浮点数的运算部分

不考列竖式计算，考试不会考列竖式计算

乘法运算：关注思路、参考作业题

n*n得到2n位，如果只保留n位会溢出，如何判断移出？

除法运算：关注概念

扩充为2n%n位

9. 指令系统

如何定义每一条指令

指令含有操作码和地址码

如何对操作数进行存储？ 寻址方式

如何周而复始执行指令？ 1. 隐式地跳转到下一条 2. 显式地在指令中给出下一条指令 3. 条件测试后计算出转移地址

寻址方式

CISC和RISC的概念

指令的格式

RISC-V 有不同的指令，指令的设计

具体的指令和RTL功能： 明确强调需要掌握一些指令的功能和RTL语言

10. CPU基础

此处知识点，参考[第八章-CPU 说是]

CPU的设计决定了时钟周期宽度和CPI（Cycle per Instruction)

time= CPI *T *numOfInstrction

CPU： 数据通路+控制器

1. 数据通路
    - 包含有 存储元件和操作元件
    - 存储元件是时序逻辑
    - 状态元件是组合逻辑
2. 控制器

单周期CPU

数据通路的图需要很熟悉！！

每一条指令对应的控制信号都要知道！！

每一个指令的执行过程

每一个指令都在一个时钟周期内完成

取指令，经过clk-to-Q, PC得到新值，经过access time得到单签指令，之后计算下指令，送入PC输入端，下一个时钟周期到来时，更新PC；同时执行对应指令

多周期

不需要数据通路和控制器等原理

只需要会对比即可！！

11. 单周期和多周期的对比：

- 单周期：T为load指令的花费时间；控制信号不变；
- 多周期：不同指令的周期数不同，T是所有阶段中最长的为准；控制信号为变，使用有限状态机来描述指令执行的流程，用PLA或者微程序的方式...

12. 流水线CPU

流水线数据通路也要很熟悉！！！

指令的吞吐率：1s能够执行的指令数量；

时钟周期为：流水段寄存器的读写时间+Mem阶段的时间；吞吐率=1/ T，T为周期； 每一条指令需要n*T时间进行执行

每一个流水段的时间都是一个时钟；

每一条指令的执行时间都相同，都为阶段数* 时钟周期

单条指令的时间变长，但是吞吐率变大。

结构冒险

数据冒险

- 软件阻塞、硬件优化
- 数据转发
- 寄存器先读后写
- 编译优化
- load-use：转发+nop（或者阻塞）解决

控制冒险

在正确的目标地址取出之前有几个出现的

- 延迟分支。硬件阻塞
- 分支预测（静态和动态）
- 异常中断会引发控制冒险

13. CPU小结

CPU的功能

控制信号由控制信号生成，数据在数据通路中的流动由控制信号确定

单周期、多周期、流水线CPU的对比

我的问题：

1. **涉及到画图的内容是 从头开始画吗？(比如画出数据通路)还是电路的某一部分，还是连线？**都会有
2. **会不会涉及到 考察"为了使得datapath支持某一个指令，如何对其进行扩展"？** 会
3. 会不会考察如何根据写出汇编代码？ 不会

4. **会不会考察如何对汇编程序进行 编译优化、分支调度等？** 会

小知识点总结

> 笔者在复习的时候，发现有一些知识点容易遗忘，下面是笔者整理的一些知识点

0.1 标志信号

在加法器中我们生成了ZF、CF、OF、SF几个标志信号：

1. 如何求？ $$OF = C_n \oplus C_{n-1} = X_{n-1}Y_{n-1}\overline{C}_{n-1} + \overline{X}_{n-1}\overline{Y}_{n-1}C_{n-1}$$ $$CF = Cin \oplus Cout = Sub \oplus Cout$$
2. Signed和unsigned分别有哪些符号？

   Signed 对应有ZF、OF、SF ； Unsigned 对应有ZF、CF；
3. 如何比较两数大小？
   1. 如果是Singed，则$x < y \quad iff \quad SF \oplus OF = 1$
   2. 如果是Unsigned，那么$x < y \quad iff \quad Cout = 1$
   3. 如果询问**大于等于、小于等于呢** 和ZF信号进行组合即可，~~添双筷子的事。~~

0.2 指令复习

问：jalr和jal的区别是什么？

jalr的功能是存储PC+4,并且跳转到imm+R[rs]的位置；而jal是跳转到PC+imm12的位置。

问：beq为什么不能实现远距离调用函数？

因为beq是条件条状指令，不是调用指令，无法返回地址

问：返回指令是否需要指定放回地址？

如果返回地址存放在栈中或特定的寄存器中，则返回指令中可以不需要地址码；如果返回地址存放在某个通用寄存器中，则返回指令中需要给出通用寄存器编号（地址码）

> 比如我们知道使用jal指令可以实现函数调用，jal ra callsite，将PC+4存入ra寄存器（Register Adderess，在RISC架构下是寄存器x1）之后跳转到目的地址；
>
> 在目的函数结束之后，有一个返回指令，可以是jalr x0 0(ra)，即，将PC+4存入x0中，PC改为R[ra]+0，也就是之前保存的值；从而实现return。（写入X0是因为 x0的值不会被修改，**x0是硬编码0的！**）
>
> 说到X0硬编码为0还有另外一个小知识点：有的时候，我们需要在汇编代码中使用x0作为写入的对象和写出的值（读取这里面的0）； 我们在流水线CPU中知道，如果后一个指令依赖到了前一个指令所需要写入的寄存器，会通过数据转发，直接将前者的结果送入到后面指令的执行过程中（比如直接送到后者的ALU中）；但是如果这个“冲突”的寄存器是x0呢？前者写入x0为了舍弃掉这个输入，因为无意义；后者使用x0是为了用到这里的0，如果数据转发，会导致**后者得到的值变成了不确定的值。** 因此在数据转发的时候，还需要判断一下寄存器是否为0.

需要记忆的指令除了`牢九条`，还有`jalr`,`auipc`等

`jalr rd rs1 1mm12`: jump and link resigter, 功能是R[rd]<-- PC+4; PC<-- R[rs]+SEXT(imm12<<1)

`auipc rd imm20`: add upper imm to PC, R[rd]<-- PC+imm20||000H（这个表示拼接12个0上去，也可以用<<12,即左移12位);

> 突然想到一个**题目**：（在作业题和PPT中均涉及到相关问题）
>
> 如果想要将一个32位的数字赋值给一个寄存器，需要先将高32位使用`lui`指令存储到`t0`中，再把低12位用addi加进去即可；
>
> 但是，如果低12位的首位为1，那么会出现错误（addi指令会对imm12进行SEXT），所以，最好的办法是：
>
> $$lui\ \ t0\ \ offset[31:12]+offset[11]$$
> $$addi\ \ t0\ \ offset[11:0]$$
>
> （大概是这个意思）
>
> 在lui中，额外加上一个offset[11]，也就是符号位；左移12位，和低12位的符号位扩展相加，这样就可以刚好**和扩展的符号位相互抵消**（进位了）
>
> 对题目的补充： 另外一个实现是，
>
> $$lui\ \ t0\ \ Aupper20$$
> $$xori\ \ t0\ \ Alower12$$
> $$lw\ \ t1\ \ 0(t0)$$
>
> 即，直接将Apper20低位补0的数和Alower12位取异或 就可以实现将一个32位的数字存储到t1寄存器中。

0.3 数据通路扩展

下面这个是单周期CPU，支持`牢九条`指令，那么如何进行扩展使得其支持其他的指令呢?

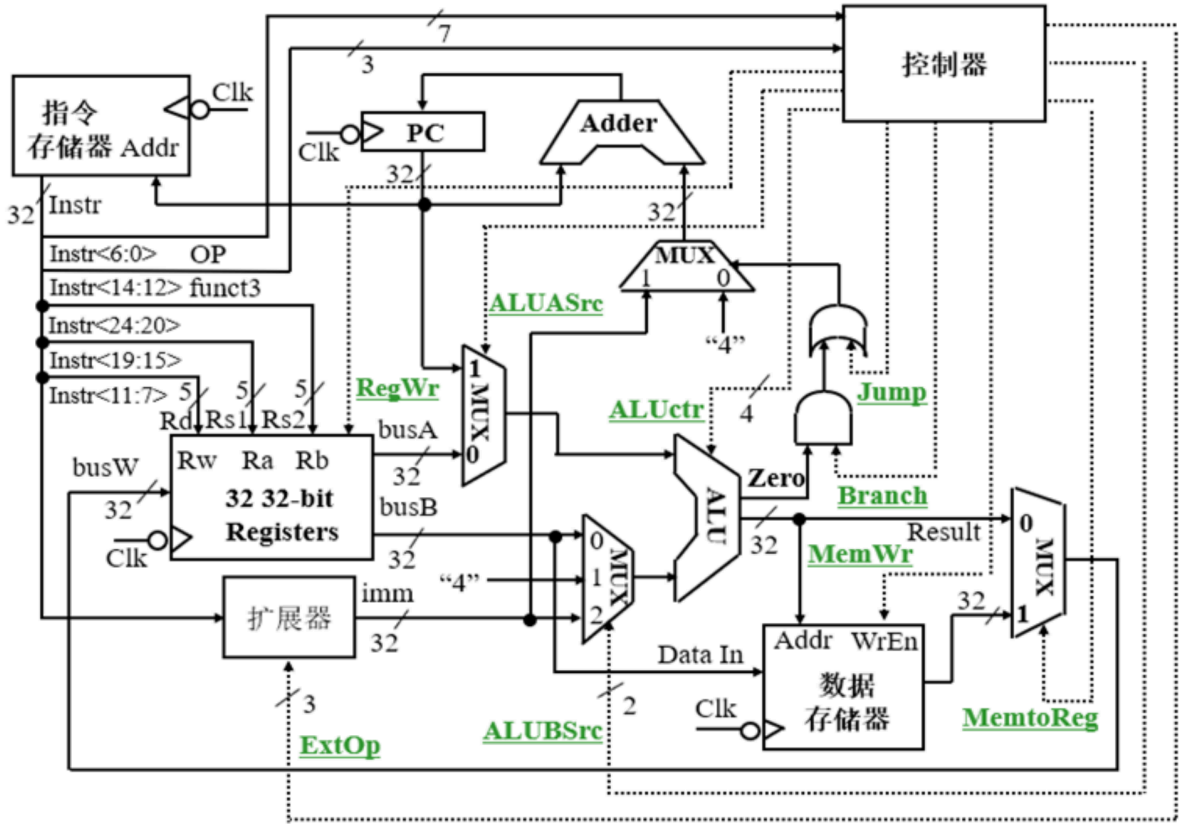

Figure 1

如何扩展以下指令?

1. auipc? 在ALUASrc选择PC作为输入即可

2. 其他的b型指令？ 可能需要func3字段作为多路选择器的选择段，将ZERO信号换成多路选择器，按照不同的b型指令，来输出对应的值；(未经验证)
3. jalr指令（jalr rd rs imm12和jal区别在于，新的PC是在rs的基础上加了一个offset，比jal范围更大）？ 将PC+4送入busW；在下地值逻辑处，添加一个多路选择器，如果是jalr指令的话，Adder的左边就输入busA而非PC；

控制器部分可能需要参考单周期CPIU实验部分内容，给相应的控制信号添加上相应的编码：

比如我想要添加jalr信号，可能控制信号要多加一个`Jalr`,只有当指令为jalr的时候才为1，使Adder的左操作数为busA, 否则左操作数为PC（和正常的一样）

下面是流水线CPU的扩展数据通路

> 没找到相关内容

0.4 性能分析

> 注意：要把握住题目中的关键信息，比如，是否考虑流水段寄存器等等；

单周期和多周期对比：

成本比较：单周期下功能部件不能重复使用；而多周期下可重复使用，比单周期省单周期指令执行结果直接保存在PC、Regfile和Memory；而多周期下需加一些临时寄存器保存中间结果，比单周期费

性能比较：单周期CPU的CPI为1，但时钟周期为最长的load指令执行时间多周期CPU的CPI是多少？时钟周期多长?

假定程序中22%为Load，11%为Store，49%为R-Type，16%为I-Type ，2%为Jump。每个状态需要一个时钟周期，CPI为多少?

若每种指令所需的时钟周期数为： Load：4；Store：4；R-Type：4；I-Type：4；Jump：3

> JUMP的CPI为3，其余的指令CPI均为4

CPI = 0.22x4+0.11x4+0.49x4+0.16x4+0.02x3 = 3.98

假设单周期时钟宽度为1，且多周期时钟周期约为单周期的1/5，则多周期的总体时间约：3.98x1/5=0.796 ； 而单周期总体时间为：1x1=1这种情况下：多周期比单周期效率高!

> 源自课件

单周期和流水线对比

假设在单周期处理器中，各主要功能单元的操作时间为：

存储单元（取指令、存取存储器里的数据）：200ps

ALU和加法器：100ps 寄存器、寄存器堆（读/写）：50ps

假设MUX、控制单元、PC、扩展器和传输线路都没有延迟，不考虑任何特殊情况（延迟），则单周期和流水线的实现方式相比，哪个更快？吞吐率呢?

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400 ps
Load word	200	50	100	200	50	600 ps
Store word	200	50	100	200	0	550 ps
Branch	200	50	100	0	0	350 ps
Jump	200	0	100	0	50	350 ps

- 不再需要关心指令种类占比。CPI都是1。
- 单周期，时钟周期600ps，吞吐率1/600ps = 1.67×10^9指令/秒
- 流水段寄存器延时50ps，最长阶段200ps，所以时钟周期是250ps（如果说忽略流水段寄存器延时，就是200ps），吞吐率1/250ps = 4×10^9指令/秒，是单周期的大约2.4倍

Figure 2

> 注： 1. 显然没考虑冒险 2. 吞吐率一般用

单周期、多周期、流水线CPU比较

- **单周期、多周期、流水线三种方式比较**

假设各主要功能单元的操作时间为：

- **存储单元：200ps**
- **ALU和加法器：100ps**
- **寄存器堆（读 / 写）：50ps**

假设MUX、控制单元、PC、扩展器和传输线路都没有延迟，指令组成为：25%取数、10%存数、52%ALU、11%分支、2%跳转，则下面实现方式中，哪个更快？快多少？

（1）单周期：每条指令在一个固定长度的时钟周期内完成

（2）多周期：每类指令时钟数为取数-5，存数-4，ALU-4，分支-3，跳转-3

（3）流水线：每条指令分取指令、取数/译码、执行、存储器存取、写回五阶段

（假定没有结构冒险，数据冒险采用转发处理，分支延迟槽为1，预测准确率为75%；无条件跳转指令的更新地址工作也在ID段完成。不考虑流水段寄存器延时，不考虑异常、中断和访存缺失引起的流水线冒险）

Figure 3

解：CPU执行时间 = 指令条数 x CPI x 时钟周期长度

三种方式指令条数都一样，所以只要比较CPI和时钟周期长度即可。

各指令类型要求的时间长度为：

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400 ps
Load word	200	50	100	200	50	600 ps
Store word	200	50	100	200	0	550 ps
Branch	200	50	100	0	0	350 ps
Jump	200	0	100	0	50	350 ps

对于单周期方式：

时钟周期将由最长指令来决定，应该是load指令，为600ps

所以，N条指令的执行时间为600N(ps)

对于多周期方式：

时钟周期将取功能部件最长所需时间，应该是存取操作，为200ps

根据各类指令的频度，计算平均时钟周期数为：

CPU时钟周期=5x25%+4x10%+4x52%+3x11%+3x2%=4.12

所以， N条指令的执行时间为4.12x200xN=824N(ps)

Figure 4

对于流水线方式：

Load指令：当发生Load-use依赖时，执行时间为2个时钟，否则1个时钟，

故平均执行时间为1.5个时钟；

Store、ALU指令：1个时钟；

Branch指令：预测成功时，1个时钟，预测错误时，2个时钟，

所以：平均约为：.75x1+.25x2=1.25个；

Jump指令：2个时钟（总要等到译码阶段结束才能得到转移地址）

平均CPI为：1.5x25%+1x10%+1x52%+1.25x11%+2x2%=1.17

所以， N条指令的执行时间为1.17x200xN=234N(ps)

Figure 5

0.5 冒险

冒险的分类：

- 控制冒险：后续指令在转移目标地址产生之前已经被取出
- 结构冒险：同一部件被不同指令所使用
- 数据冒险：后面的指令使用到了前面还没有生成的数据

冒险的解决方案：

- 数据冒险：
  - 硬件阻塞、软件插入无关指令

- 同一周期先写后读（对于后面指令读了前面指令写的寄存器问题，只能将原先空三个指令改为空2个指令）
- 转发（无法解决load use冒险）
- 编译优化（解决一下load-use冒险）

> 1. Load-use冒险没法通过转发技术解决，因为load一定要到M段才能把得到结果，而后面的一个指令此时一定处于Exec阶段，需要这个数据；矛盾无法调和。
> 2. **数据冒险处理最佳方案："转发" + "Load-use阻塞"**

- 控制冒险
  - 阻塞、空指令
  - 减小延迟损失片

> 没有任何的优化（在M段才把PC进行更新）C=3
>
> 如果在Ex段就根据Zero信号和Target来更新PC，那么C=2
>
> 如果将target的计算和zero计算都前移到ID阶段，那么C=1
>
> Reference：(26 封私信 / 19 条消息) [读书笔记]CSAPP：13[B]处理器体系结构：流水线 - 知乎

  - 分支预测（静态、动态）

> 分支预测计算CPI的时候注意，只有B型指令可以预测，且B型指令的CPI为1+C*p，p为预测错误的概率

  - 延迟分支（将前面的指令往后移）

> 类似之前解决load-use冒险的编译优化，也是调整指令的顺序；
>
> 把分支指令前面的与分支指令无关的指令调到分支指令后面执行，以填充延迟时间片（也称分支延迟槽Branch Delay slot），不够时用nop操作填充；

0.6 CPU中异常的处理

> 不知道会不会考，但是我记得相关的知识点在几个章节里面都有

外部异常、内部中断；

CPU对异常的处理流程：

1. 保存断点和程序状态
2. 识别程序异常状态并且转到异常状态进行执行

多周期的异常处理：

> 图中添加了两个状态对应异常处理的状态；新增的寄存器`Cause`，`EPC`分别存储异常原因和断点信息

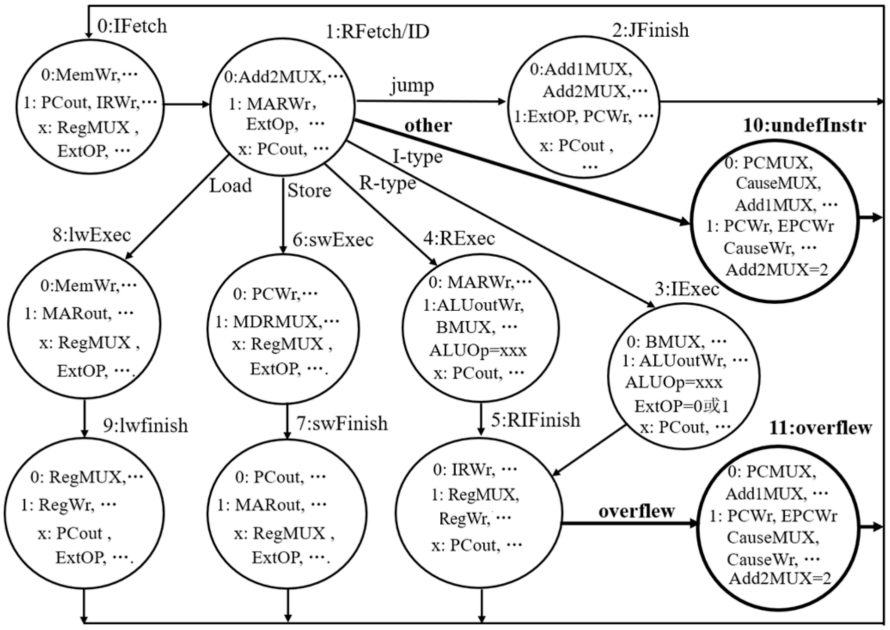

Figure 6

流水线的异常处理：

异常和中断会引起`控制冒险`；

处理方法：

- IF段前先检查是否有外部中断：如果有就执行中断逻辑。
- 在每一个流水段中添加对应可能出现的内部异常检测逻辑，比如EX段添加OVERFLOW的检测；
- 2021真题："该数据通路能否检测溢出异常？在溢出异常发生时，结果是否会写入目的寄存器",可以，因为流水段中有Overflow信号。

> 见课本270页；

0.7 The End

> 本课程最后一页ppt

Figure 7

考前提醒

1. 地址范围不是指令范围
2. 注意判断需要插入几个nop指令、阻塞几个周期时，需要考虑**是否有寄存器先写后读！否则需要分类讨论**