第二次习题课

6月17日

4.1 考虑如下指令：

指令：`and rd, rs1, rs2`

解释：`Reg[rd] = Reg[rs1] AND Reg[rs2]`

4.1.1 [5]<4.3>对于上述指令，图 4-10 中的控制信号各是什么数值？

4.1.2 [5]<4.3>对于上述指令，将用到哪些功能单元？

4.1.3 [10]<4.3>对于上述指令，哪些功能单元不产生任何输出？哪些功能单元的输出不会被用到？

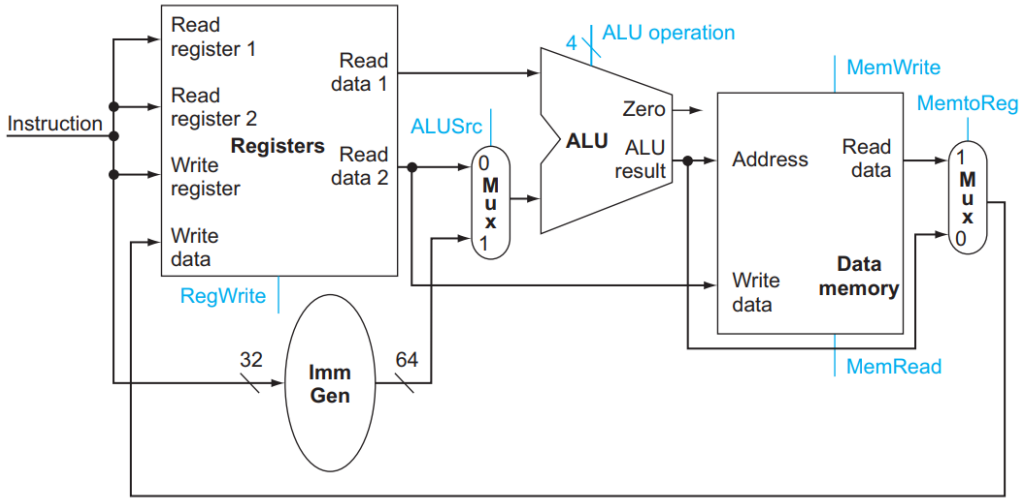

FIGURE 4.10 The datapath for the memory instructions and the R-type instructions. This example shows how a single datapath can be assembled from the pieces in Figures 4.7 and 4.8 by adding multiplexors. Two multiplexors are needed, as described in the example.

- 4.1.1

控制信号	取值
RegWrite	1
ALUSrc	0
ALU operation	AND
MemRead	0
MemWrite	0
MemtoReg	0

- 4.1.2：寄存器堆、ALU、ALUSrc 多路选择器、MemtoReg 多路选择器
- 4.1.3：
  - 都产生输出
  - 内存模块和立即数生成器的输出不会被用到

4.7 假设用来实现处理器数据通路的各功能模块延迟如下所示：

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250ps	150ps	25ps	200ps	150ps	5ps	30ps	20ps	50ps	50ps

其中，寄存器读延迟指的是，时钟上升沿到寄存器输出端稳定输出新值所需的时间。该延迟仅针对 PC 寄存器。寄存器建立时间指的是，寄存器的输入数据稳定到时钟上升沿所需的时间。该数值针对 PC 寄存器和寄存器堆。

4.7.1 [5] <4.4> R 型指令的延迟是多少？比如，如果想让这类指令工作正确，时钟周期最少为多少？

4.7.2 [10] <4.4> ld 指令的延迟是多少？仔细检查你的答案，许多学生会在关键路径上添加额外的寄存器。

4.7.3 [10] <4.4> sd 指令的延迟是多少？仔细检查你的答案，许多学生会在关键路径上添加额外的寄存器。

4.7.4 [5] <4.4> beq 指令的延迟是多少？

4.7.5 [5] <4.4> I 型指令的延迟是多少？

4.7.6 [5] <4.4> 该 CPU 的最小时钟周期是多少？

- 4.7.1:
  - PC -> I-Mem -> Register File read -> ALUSrc Mux -> ALU -> MemtoReg Mux -> Register File setup
  - 30 + 250 + 150 + 25 + 200 + 25 + 20 = 700 ps
- 4.7.2:
  - PC -> I-Mem -> Register File read -> ALU 计算地址 -> D-Mem read -> MemtoReg Mux -> Register File setup
  - 30 + 250 + 150 + 200 + 250 + 25 + 20 = 925 ps
- 4.7.3:
  - PC -> I-Mem -> Register File read -> ALU 计算地址 -> D-Mem write
  - 30 + 250 + 150 + 200 + 250 = 880 ps

4.7 假设用来实现处理器数据通路的各功能模块延迟如下所示：

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250ps	150ps	25ps	200ps	150ps	5ps	30ps	20ps	50ps	50ps

其中，寄存器读延迟指的是，时钟上升沿到寄存器输出端稳定输出新值所需的时间。该延迟仅针对 PC 寄存器。寄存器建立时间指的是，寄存器的输入数据稳定到时钟上升沿所需的时间。该数值针对 PC 寄存器和寄存器堆。

4.7.1 [5] <4.4> R 型指令的延迟是多少？比如，如果想让这类指令工作正确，时钟周期最少为多少？

4.7.2 [10] <4.4> `ld` 指令的延迟是多少？仔细检查你的答案，许多学生会在关键路径上添加额外的寄存器。

4.7.3 [10] <4.4> `sd` 指令的延迟是多少？仔细检查你的答案，许多学生会在关键路径上添加额外的寄存器。

4.7.4 [5] <4.4> `beq` 指令的延迟是多少？

4.7.5 [5] <4.4> I 型指令的延迟是多少？

4.7.6 [5] <4.4> 该 CPU 的最小时钟周期是多少？

- 4.7.4:
  - PC -> I-Mem -> Register File read -> ALUSrc Mux -> ALU 比较 -> AND Gate -> PC Mux -> PC setup
  - 30 + 250 + 150 + 25 + 200 + 5 + 25 + 20 = 705 ps
- 4.7.5:
  - PC -> I-Mem -> Register File read -> ALU -> MemtoReg Mux -> Register File setup
  - 30 + 250 + 150 + 200 + 25 + 20 = 675 ps
- 4.7.6:
  - 925ps

思考题：(交)

单周期处理器在一个周期内完成指令所有的微操作,思考：

--寻址方式如何实现

--周期宽度如何确定

--能否“在一个clk内完成”

--能否将两个adder合二为一

--能否将两个memory合二为一

- 寻址方式：
  - 取指地址由 PC 给出
  - load/store 使用基址加偏移寻址
  - branch 使用 PC 相对寻址
- 周期宽度：
  - 周期宽度由所有支持指令中最大的指令延迟决定，也就是关键路径决定。
- 能否“在一个clk内完成”
  - 可以，但前提是这个 clk 的周期足够长，能够覆盖最慢指令的完整组合逻辑传播时间以及目标寄存器的建立时间。如果时钟周期短于关键路径延迟，虽然电路仍会传播信号，但结果在下一个时钟沿到来前尚未稳定，处理器就可能写入错误结果
- 能否将两个 adder 合二为一/能否将两个 memory 合二为一
  - 由于所有执行路径上的部件在执行一条指令时只能使用一次，因此不能合二为一

1. 每一类指令的指令周期各含多少个时钟周期？（交）

2、分别分析R/I/S/B-type指令的多周期设计方案中每个周期所用到的功能部件（交）

- R-type: 4
  - IF: PC、指令存储器、加法器
  - ID: 寄存器堆
  - EX: ALU
  - WB: 寄存器堆
- I-type ALU: 4
  - IF: PC、指令存储器、加法器
  - ID: 寄存器堆、立即数生成器
  - EX: ALU
  - WB:寄存器堆
- I-type Load: 5
  - IF: PC、指令存储器、加法器
  - ID: 寄存器堆、立即数生成器
  - EX: ALU
  - MEM: 内存
  - WB: 寄存器
- S-type: 4
  - IF: PC、指令存储器、加法器
  - ID: 寄存器堆、立即数生成器
  - EX: ALU
  - MEM: 内存
- B-type:
  - IF: PC、指令存储器、加法器
  - ID: 寄存器堆、立即数生成器、加法器
  - EX: ALU、PC

4.16 在本题中将讨论流水线如何影响处理器的时钟周期。假设数据通路的各个流水级的延迟如下：

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

同时，假设处理器执行的指令分布如下：

ALU/Logic	Jump/Branch	Load	Store
45%	20%	20%	15%

4.16.1 [5] <4.5> 在流水化和非流水的处理器中，时钟周期分别是多少？

4.16.2 [10] <4.5> 在流水化和非流水的处理器中，对于 ld 指令的延迟分别是多少？

4.16.3 [10] <4.5> 如果我们将数据通路中的一个流水级拆成两个新流水级，每一个新流水级的延迟是原来的一半，那么我们将拆分哪一级？新处理器的时钟周期是多少？

4.16.4 [10] <4.5> 假设没有停顿或冒险，数据存储的利用率如何？

4.16.5 [10] <4.5> 假设没有停顿或冒险，寄存器堆的写口利用率如何？

- 4.16.1:
  - 流水化: max(250, 350, 150, 300, 200) = 350 ps
  - 非流水化: 250 + 350 + 150 + 300 + 200 = 1250 ps
- 4.16.2:
  - 流水化: 5 * 350 = 1750 ps
  - 非流水化: 1250 ps
- 4.16.3:
  - 拆分当前最慢的流水级 ID，其延迟为 350 ps。拆成两个等长阶段。此时新处理器的时钟周期为 300 ps(MEM)
- 4.16.4:
  - 数据存储器利用率 = Load 比例 + Store 比例 = 20% + 15% = 35%
- 4.16.5:
  - 寄存器堆写口利用率 = ALU/Logic 比例 + Load 比例 = 45% + 20% = 65%

4.23 如果我们改变 load/store 指令格式，使用寄存器（不需要立即数偏移）作为访存地址，这些指令就不再需要使用 ALU（具体见习题 4.15）。这样的话，MEM 阶段和 EX 阶段就可以重叠，流水级数变为四级。

4.23.1 [10] <4.5> 流水线级数的减少会影响时钟周期吗？

4.23.2 [10] <4.5> 这样的变化会提高流水线的性能吗？

4.23.3 [10] <4.5> 这样的变化会降低流水线的性能吗？

- 4.23.1:
  - 不一定会影响时钟周期，要看合并或重叠之后新的最长流水级是否变短
- 4.23.2:
  - 可能降低 load/store 的单条指令延迟，但是否提高整体性能，要看 CPI 等是否改善
- 4.23.3:
  - 这种变化可能降低整体性能，原因是它削弱了 load/store 的寻址能力，使许多原本一条指令完成的访存操作需要额外地址计算指令
  - 这会增加动态指令条数，也可能增加寄存器压力和数据相关停顿。即使单条 load/store 的流水线级数减少，只要程序的指令数增加、CPI 变差，或者时钟周期没有同步改善，整体执行时间仍可能变长

◘思考题：（选做，不交）

- ✓--影响流水线性能发挥的因素有哪些?
- ✓--为何RV只有Load/store指令访存
- ✓--单周期、多周期的控制信号无须buffer，而流水线的控制信号需要buffer的原因
- ✓--流水线控制器的实现方式

- 影响流水线性能的因素：
  - 流水线性能取决于时钟周期、CPI、指令数；冒险、停顿、预测失败、访存延迟和流水级不均衡都会让性能低于理想值
- RV 只有 Load/Store 指令访存:
  - 如果 ALU 指令也能直接访问内存，一条指令可能既要读内存又要运算，甚至还要写内存，流水线阶段会变得不均衡，控制也更复杂
  - RISC-V 采用 Load/Store 架构，是为了让指令行为规整、硬件简单、流水线更容易高效实现
- 流水线控制信号需要 buffer:
  - 单周期处理器中，一条指令在一个周期内完成
  - 多周期处理器中，同一条指令分多个周期执行，但同一时刻通常只处理一条指令
  - 流水线中同一时刻有多条指令在不同阶段执行，每条指令需要的控制信号不同
- 流水线控制器:
  - 在 ID 阶段统一译码，生成该指令后续阶段所需的控制信号，然后把控制信号分组放入流水线寄存器，随指令逐级传递

4.22 [5]<4.5>对于如下的 RISC-V 的汇编片段：

```
sd    x29, 12(x16)
ld    x29, 8(x16)
sub   x17, x15, x14
beqz  x17, label
add   x15, x11, x14
sub   x15, x30, x14
```

只有一个存储器，同时存储指令和数据

4.22.1 [5]<4.5>请画出流水线图，说明以上代码会在何处停顿。

4.22.2 [5]<4.5>是否可通过重排代码来减少因结构冒险而导致停顿的次数？

4.22.3 [5]<4.5>该结构冒险必须用硬件来解决吗？我们可以通过在代码中插入 NOP 指令来消除数据冒险，对于结构冒险是否可以相同处理？如果可以，请解释原因。否则，也请解释原因。

4.22.4 [5]<4.5>在典型程序中，大约需要为该结构冒险产生多少个周期的停顿？（使用习题 4.8 中的指令分布）。

指令	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
I1: sd	IF	ID	EX	MEM	WB							
I2: ld		IF	ID	EX	MEM	WB						
I3: sub			IF	ID	EX	MEM	WB					
I4: beqz				stall	stall	IF	ID	EX	MEM	WB		
I5: add							IF	ID	EX	MEM	WB	
I6: sub								IF	ID	EX	MEM	WB

- 4.22.2:
  - 只要有 ld 或 sd，它们进入 MEM 阶段时就会和取指冲突。代码重排最多改变冲突出现在什么时候，但不能让一个单端口存储器同周期完成两次访问
- 4.22.3:
  - NOP 本身也是一条指令，也需要从存储器取出来，插入 NOP 不会增加存储器端口，也不会消除 IF 和 MEM 同时访问存储器的问题
- 4.22.4:
  - ld, sd 占 36%
  - 如果有 N 条指令，则总共会有 0.36N 个周期的停顿

4.25 考虑如下循环：

```
LOOP: ld    x10, 0(x13)
      ld    x11, 8(x13)
      add   x12, x10, x11
      subi  x13, x13, 16
      bnez  x12, LOOP
```

如果使用完美的分支预测（即没有控制冒险带来的流水线停顿），流水线中没有使用延迟槽，采用硬件前递解决数据冒险，分支指令在 EX 阶段判断是否跳转。

4.25.1 [10]<4.7>给出该循环中前两次循环的流水线执行图。

4.25.2 [10]<4.7>标注出没有进行有用操作的流水级。当流水线全负荷工作时，所有五个流水级都在进行有用操作的情况多久会出现一次？（从 subi 指令进入 IF 阶段开始计算，到 bnez 指令进入 IF 阶段结束。）

- 4.25.2：五个流水级都在有用操作的情况不会出现

指令	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
I1: `ld x10, 0(x13)`	IF	ID	EX	MEM	WB											
I2: `ld x11, 8(x13)`		IF	ID	EX	MEM	WB										
I3: `add x12, x10, x11`			IF	ID	**stall**	EX	MEM	WB								
I4: `subi x13, x13, 16`				IF	**stall**	ID	EX	MEM	WB							
I5: `bnez x12, LOOP`						IF	ID	EX	MEM	WB						
I1': `ld x10, 0(x13)`							IF	ID	EX	MEM	WB					
I2': `ld x11, 8(x13)`								IF	ID	EX	MEM	WB				
I3': `add x12, x10, x11`									IF	ID	**stall**	EX	MEM	WB		
I4': `subi x13, x13, 16`										IF	**stall**	ID	EX	MEM	WB	
I5': `bnez x12, LOOP`												IF	ID	EX	MEM	WB

- 中断周期要完成的微操作
  - 保存返回地址 EPC
  - 记录中断/异常原因 Cause
  - 关中断或修改状态寄存器 Status
  - 把 PC 改为异常处理入口地址

- 多周期状态机中，出现溢出的指令是否会把错误结果写回
  - 不会，如果 ALU 在 EX 阶段发现溢出，那么这条指令不能继续进入正常写回状态。控制器应该转入异常处理状态，异常信号要及时改变控制状态，禁止错误写回

- 多周期状态机中如何响应中断
  - 多周期中同一时刻只执行一条指令，所以通常可以在某些安全边界检查中断，例如：一条指令结束后或每个状态结束时或 IF 开始前若检测到中断

- 顺序执行中断“精确”；流水线中断“精确”或“非精确”可选
  - 顺序执行：顺序执行 CPU 一次只处理一条指令，所以中断/异常点很清楚，异常之前的指令都完成异常之后的指令都没开始
  - 流水线：流水线 CPU 中，同一时刻有多条指令在不同阶段，如果某条指令异常，要保证这条指令之前的指令已经完成，本条指令及其之后的指令不会改变机器状态；如果机器允许异常点不那么清楚，比如异常发生时后面的某些指令已经改了状态，就叫非精确中断。现代通用处理器通常尽量提供精确异常

- 精确中断为什么提交点是 M 段
  - 到 MEM 阶段时，前面的指令已经更靠后或即将完成；后面的指令还没有提交最终状态；此时比较容易统一处理异常、冲刷后续指令、保存 EPC/Cause。M 段是一个方便判断“哪些指令该提交、哪些该取消”的边界

- EPC 和 Cause 应该在哪个阶段写？异常检测电路在哪里？
  - 异常检测电路分布在可能产生异常的阶段(IF、ID、EX、MEM)
  - 但 EPC 和 Cause 往往不会在每个阶段立即随便写，而是统一在异常提交/处理阶段写，常见是 MEM 附近，这样更容易保证精确异常

- MIPS 异常返回指令 eret 如何实现
  - 把 PC 恢复到 EPC 保存的返回地址。清除异常级标志，重新允许正常中断/异常处理。所以 eret 可以看作一种特殊跳转指令，只不过跳转目标来自 EPC，而不是普通寄存器或立即数

- 异常与中断同时发生，优先级如何？
  - 同步异常优先于异步中断
  - 异常和当前正在执行的指令直接相关，比如溢出、非法指令、访存错误；如果不先处理，机器无法确定这条指令是否正确完成
  - 中断是外设来的异步请求，通常可以稍微延后一个或几个周期处理

- 分支延迟槽中的指令发生异常，EPC = ？
  - EPC 设置为分支指令的地址
  - Cause 寄存器中的 BD（Branch Delay）位置为 1

- 对比中断、异常、过程调用

对比维度	中断	异常	过程调用
请求时间	随机、不可预测（外部事件）	指令执行时同步发生	程序设计时确定
响应时间	延迟响应（指令周期结束时）	立即响应（"马上"处理）	立即响应（指令执行）
同步/异步	异步（与当前程序无关）	同步（由当前指令引起）	同步（程序主动调用）
断点保存	PC保存到栈或特定地址	PC-4保存到EPC寄存	返回地址保存到链接寄存器或栈
现场保护	通用寄存器、PSW（由ISR完成）	相关寄存器（由处理程序完成）	调用者/被调用者保存寄存器
返回点	被中断指令的下一条指令	异常指令（可恢复）或下一条	CALL指令的下一条指令
中断周期	需要（中断隐指令完成）	不需要（同步处理）	不需要
系统状态	用户态→内核态，关中断	用户态→内核态	同一特权级（通常）