



**《计算机组成原理》**

**课程实验报告**

姓名： 潘腾凯

学院：信息学院

系：软件工程

专业： 软件工程

学号：37220232203786

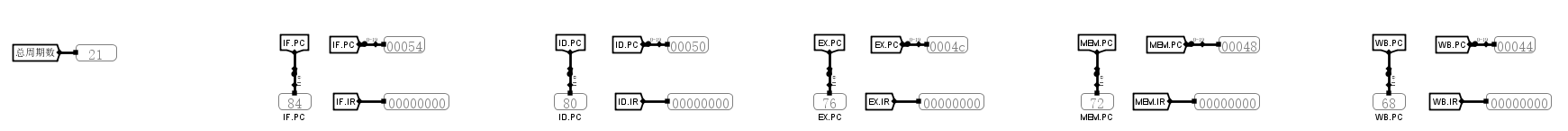
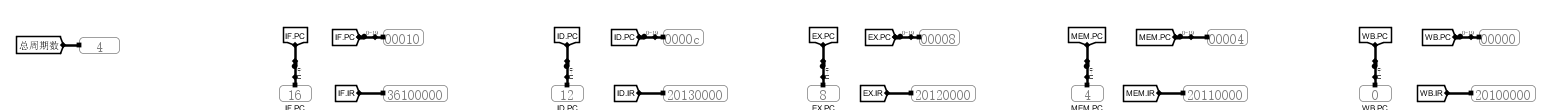
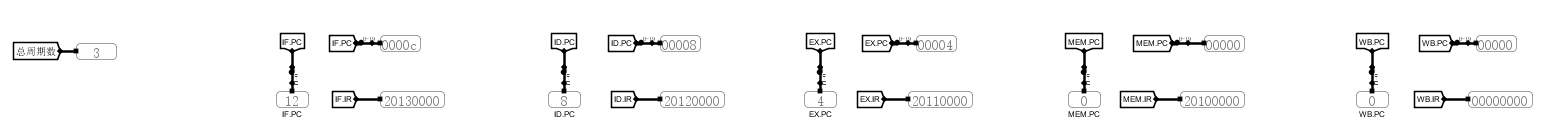
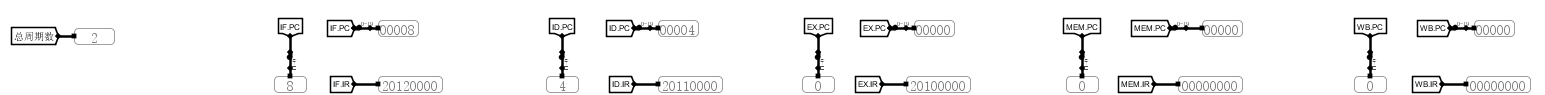
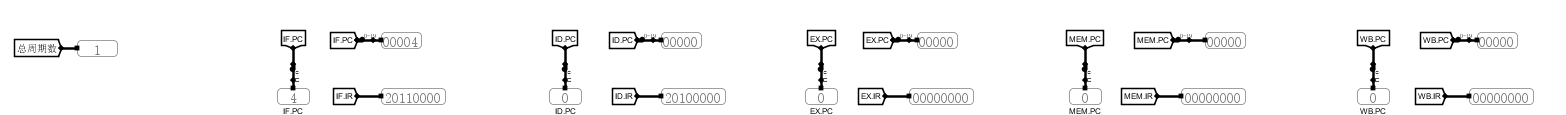
2025年5月22日

**第7次实验 流水线CPU设计**

1. **实验目的**
2. **了解理想流水线MIPS处理器设计的基本原理，在该流水线上运行测试程序。**
3. **了解气泡流水线MIPS处理器设计的基本原理，在该流水线上运行测试程序。**
4. **了解重定向流水线MIPS处理器设计的基本原理，在该流水线上运行测试程序。**
5. **了解动态分支预测流水线MIPS处理器设计的基本原理，在该流水线上运行测试程序。**
6. **实验环境**
7. **Windows系统下运行Logisim软件（需安装JDK）。**
8. **MARS 4.5汇编工具。**
9. **实验内容**

**①在理想流水线MIPS处理器上运行测试程序。**

**（1）在理想流水线MIPS处理器的数据通路上运行测试程序test1-1.hex**

****

**（2）请分析test1-1.asm程序，为什么该程序的所有指令均无相关性？**

答：分析程序，无相关性原因如下：

1.数据独立性

程序通过 addi 和 ori 指令对寄存器 $s0-$s3 进行初始化和赋值，每条指令操作独立寄存器，无跨指令数据依赖。

例如：

addi $s0,$zero,0 # 初始化$s0为0

ori $s0,$s0,0 # 覆盖$s0为0（实际无需依赖前序结果）

虽然 ori 读取 $s0，但其值已由 addi 确定，且后续指令不依赖该 ori 的结果。

2.存储指令的独立性

sw 指令存储数据到内存时，基址寄存器 $s0（值为 0）固定，数据寄存器 $s0-$s3 分别来自独立的 ori 赋值，无数据依赖链。

例如：

sw $s0,0($s0) # 存$s0到地址0

sw $s1,4($s0) # 存$s1到地址4（独立于$s0）

3.控制流隔离

syscall 前插入三条无关 addi 指令，确保 addi $v0,$zero,10 与 syscall 无相关性，消除控制依赖。

4.结构无关性

指令间无资源竞争（如寄存器写端口、内存访问单元），避免结构相关。

**（3）分析理想流水线MIPS处理器的数据通路电路。**

答：数据通路整体工作流程：

1.首先在 IF 阶段取出指令，存入 IF/ID 流水寄存器。

2.指令进入 ID 阶段译码，同时从寄存器堆读取操作数，然后将指令信息和操作数等存入 ID/EX 流水寄存器。

3.EX 阶段从 ID/EX 流水寄存器获取信息，在 ALU 中执行运算，结果存入 EX/MEM 流水寄存器。

4.若涉及访存，MEM 阶段从 EX/MEM 流水寄存器获取地址等信息，访问数据存储器，结果存入 MEM/WB 流水寄存器。

5.最后 WB 阶段从 MEM/WB 流水寄存器获取数据，写回到寄存器堆 。通过这样的流程，实现指令在流水线各阶段的有序处理，提高处理器执行效率。

**（4）分析理想流水线MIPS处理器的4个流水寄存器电路（IF/ID、 ID/EX、 EX/MEM、 MEM/WB）**

答：分析如下：

1.IF/ID 流水寄存器

功能：位于取指（IF）和译码（ID）阶段之间，用来锁存 IF 阶段取到的指令信息以及程序计数器（PC）的值等。确保 ID 阶段能接收到稳定且正确的指令和 PC 相关信息，为指令译码做准备。

电路组成及原理：

数据输入输出：有 32 位指令输入 IRin32 和输出 IRout32 ，用于传递指令；32 位 PC 值输入 PCin32 和输出 PCout32 ，传递 PC 值。通过多路选择器（MUX）等电路，在时钟信号（CLK）控制下，将 IF 阶段的结果存入寄存器，供 ID 阶段使用。

控制信号： CTin16 和 CTout16 可能是 16 位控制信号输入输出，用于传递控制相关信息；传输状态信号用于表明数据是否有效；同步清零（接 RST）和使能端（stall 信号控制） ，stall 为高电平时阻止数据传输，可用于流水线暂停等情况。

2.ID/EX 流水寄存器

功能：处于译码（ID）和执行（EX）阶段间，存储 ID 阶段译码后的指令信息，如操作码、从寄存器堆读出的操作数等，为 EX 阶段执行指令提供所需数据。

电路组成及原理：

数据通路：有多组数据输入输出端口，如操作数相关端口（如 Rt1in32 等）、控制信号端口（如 CTin16 等）。通过 MUX 和寄存器等电路，在时钟作用下，将 ID 阶段处理后的信息传递给 EX 阶段。

控制逻辑：同步清零和使能控制（stall 信号），与 IF/ID 寄存器类似，用于控制数据存储和传输，保证流水线在合适时机进行数据传递，避免错误数据流入 EX 阶段。

3.EX/MEM 流水寄存器

功能：在执行（EX）和访存（MEM）阶段之间，保存 EX 阶段运算结果（如 ALU 运算结果）以及相关控制信号等，为 MEM 阶段进行访存操作提供数据，如访存地址、读写控制信号等。

电路组成及原理：

数据传输：包括 ALU 运算结果输入输出（如 ALUin32 和 ALUout32 ）、访存相关信号（如 Memin32 和 Memout32 ）等。通过一系列逻辑电路，在时钟驱动下，将 EX 阶段产生的数据和控制信息准确传递给 MEM 阶段。

控制机制：同样具备同步清零和 stall 使能控制，保证在流水线需要暂停或异常处理时，能正确控制数据传输，防止错误数据进入访存阶段。

4.MEM/WB 流水寄存器

功能：位于访存（MEM）和写回（WB）阶段之间，锁存 MEM 阶段从数据存储器读出的数据（如果是加载指令）或需要写回寄存器的结果数据，以及相关控制信号，为 WB 阶段将数据写回寄存器堆提供数据。

电路组成及原理：

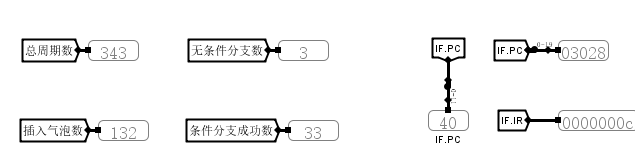
数据端口：有数据输入输出端口（如 ALUin32 、MDin32 等）用于传递不同来源的数据，控制信号端口（如 CTin16 等）传递控制信息。通过 MUX 等电路实现数据选择和存储，在时钟信号控制下，将 MEM 阶段的结果传递给 WB 阶段。

控制信号：同步清零和 stall 使能信号控制数据传输，确保在合适的时钟周期将正确的数据传递给 WB 阶段，完成指令执行的最后一步。

**②在气泡流水线MIPS处理器上运行测试程序。**

**（1）在气泡流水线MIPS处理器的数据通路上运行test2-1.hex、 test2-2.hex、 test2-3.hex、sum\_mips.hex、 fib\_mips.hex、 sort1\_mips.hex、 sort2\_mips.hex等程序，记录每个程序运行后的总周期数、插入气泡数。**

**Test2-1**



**Test2-2**



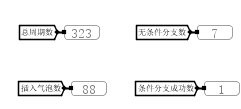
**Test2-3**



**Sum**



**Fib**



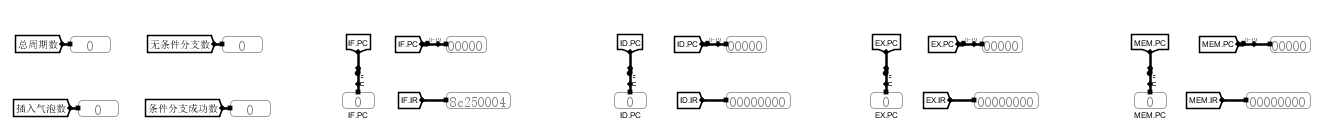
**Sort1**

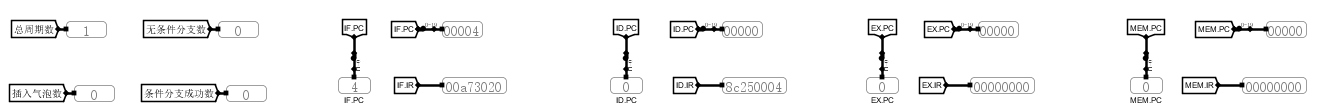


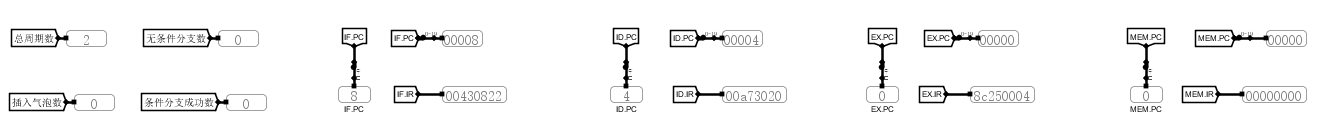
**Sort2**



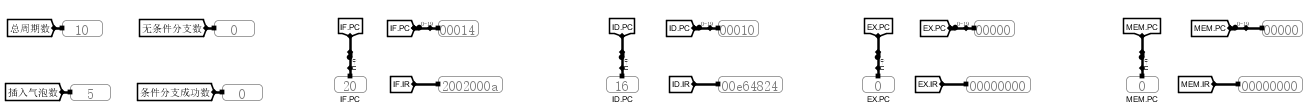
（2）在气泡流水线MIPS处理器的数据通路上运行教材P269例7.1的程序（test2-4.hex），给出该程序运行后的时空图，并与教材上的图7.20进行比较，观测是否一致？







......



与教材上的变化一致。

（3）请分析test2-1.asm、 test2-2.asm、 test2-3.asm程序。

答：

1.test2-1.asm：分支相关测试程序

功能：将寄存器 $s1 初始化为 32，通过子程序调用不断减 1，并在数码管上显示，直到 $s1 为 0。

分支指令分析：

j jmp\_next1：无条件跳转，跳过后续 3 条指令。

beq $zero,$zero, jmp\_next2：恒成立的条件跳转，等效于 j 指令，跳过后续 3 条指令。

bne $zero,$s1, jmp\_next3：因 $s1=32，跳转到 jmp\_next3，跳过后续 3 条指令。

jal jmp\_func：调用子程序，保存返回地址到 $31（$ra）。

bne $s1,$zero,jmp\_func：子程序内循环，$s1 减 1 后若非零则继续循环。

jr $31：子程序返回，跳转到 jal 保存的地址。

气泡流水线处理：

分支指令执行时，流水线通过插入气泡（NOP） stalls 后续指令，确保分支目标地址确定后再取指，避免分支相关。

程序中注释的 nop 预留位置，实际执行时由流水线自动插入气泡填充。

2.test2-2.asm：数据相关测试程序

功能：计算 0+1+2+3+4+5+6+7=28，并在数码管上显示结果。

数据相关分析：

RAW 相关：

sw $s1,0($s1) 依赖 addi $s1,$0,4 的结果。

lw $s2,0($s1) 依赖 sw 的内存写入结果。

后续 add 和 addi 指令链中，如 add $s0,$s0,$s1 依赖前序 addi $s1,$s0,1 的结果。

气泡流水线处理：

当检测到 RAW 相关时，流水线通过插入气泡 stall 后续指令，直到数据就绪。

例如，lw 指令需等待内存访问完成，期间流水线插入气泡，确保数据正确传递。

3.test2-3.asm：数据相关测试程序

功能：对寄存器进行逻辑和算术运算，并依次在数码管上显示 $s1 到 $s5 的值。

数据相关分析：

RAW 相关：

sub $s2,$s1,$zero 依赖 and $s1,$s1,$s2 的结果。

add $s3,$s1,$s1 依赖 sub 对 $s1 的更新。

or $s4,$s5,$s1 和 and $s5,$s6,$s1 均依赖 $s1 的最新值。

气泡流水线处理：

流水线在检测到 RAW 相关时插入气泡，确保前序指令结果写入寄存器后，后续指令再读取该寄存器。

例如，sub 指令执行时，流水线 stall 后续 add 指令，直到 sub 完成写回。

（4）分析气泡流水线MIPS处理器的数据通路电路。

答：

1.整体架构

气泡流水线 MIPS 处理器的数据通路涉及取指（IF）、译码 / 取数（ID）、执行（EX）、访存（MEM）和写回（WB）等阶段 ，通过一系列逻辑门、寄存器和控制信号协同工作，以解决数据冲突和控制冲突。

2.关键控制信号及其作用

Stall与相关信号

Stall = DataHazard：当检测到数据相关（DataHazard ）时，产生阻塞信号Stall 。

PC.EN = ~Stall：程序计数器使能信号，Stall为高电平时（存在数据相关），PC.EN为低电平，阻止程序计数器递增，暂停取指阶段新指令的读取 。

IF/ID.EN = ~Stall：IF/ID 流水寄存器使能信号，有数据相关时，阻止当前指令从 IF 阶段流入 ID 阶段。

清零信号

IF/ID.CLR = BranchTaken：当发生分支跳转（BranchTaken 为高电平）时，清空 IF/ID 流水寄存器，防止错误指令继续执行。

ID/EX.CLR = BranchTaken + DataHazard ：出现分支跳转或数据相关时，清空 ID/EX 流水寄存器，保证流水线中指令的正确性。

3.电路各部分功能

取指阶段（IF）：程序计数器（PC）根据PC.EN信号决定是否递增，以获取下一条指令地址。

IF/ID 阶段：通过IF/ID.EN和IF/ID.CLR信号，控制指令从 IF 阶段到 ID 阶段的传输以及寄存器内容的清空。

ID/EX 阶段：依据ID/EX.CLR信号，处理分支跳转和数据相关情况下寄存器的清空操作，保证指令执行的正确顺序。

（5）分析气泡流水线MIPS处理器的数据相关检测电路。

答：

1.输入信号

1.寄存器相关信号

ID.R1、ID.R2：分别表示在译码阶段（ID）指令中用到的源寄存器rs和rt 。

EX.WriteReg：执行阶段（EX）要写入的目标寄存器编号。

MEM.WriteReg：访存阶段（MEM）要写入的目标寄存器编号。

2.控制信号

EX.RegWrite：执行阶段寄存器写使能信号，高电平时表示在执行阶段会对寄存器进行写操作。

MEM.RegWrite：访存阶段寄存器写使能信号，高电平时表示在访存阶段会对寄存器进行写操作。

3.操作码相关信号

OP：指令操作码。

Func：指令功能码。用于判断指令对源寄存器RsUsed和RtUsed的使用情况。

2.逻辑判断过程

1.检测 Rs 相关数据冲突

对于执行阶段：通过判断RsUsed（表示指令使用了rs寄存器 ）、rs≠0（ID.R1与全 0 比较，通过比较器取反实现 ）、EX.RegWrite（执行阶段写使能 ）、rs==EX.WriteReg（ID.R1与EX.WriteReg相等比较 ）这几个条件是否同时满足，若满足则对应一路输出为高电平。

对于访存阶段：判断RsUsed、rs≠0、MEM.RegWrite、rs==MEM.WriteReg这几个条件是否同时满足，满足则对应一路输出为高电平。

2.检测 Rt 相关数据冲突

对于执行阶段：判断RtUsed（表示指令使用了rt寄存器 ）、rt≠0（ID.R2与全 0 比较，通过比较器取反实现 ）、EX.RegWrite、rt==EX.WriteReg是否同时满足，满足则对应一路输出为高电平。

对于访存阶段：判断RtUsed、rt≠0、MEM.RegWrite、rt==MEM.WriteReg是否同时满足，满足则对应一路输出为高电平。

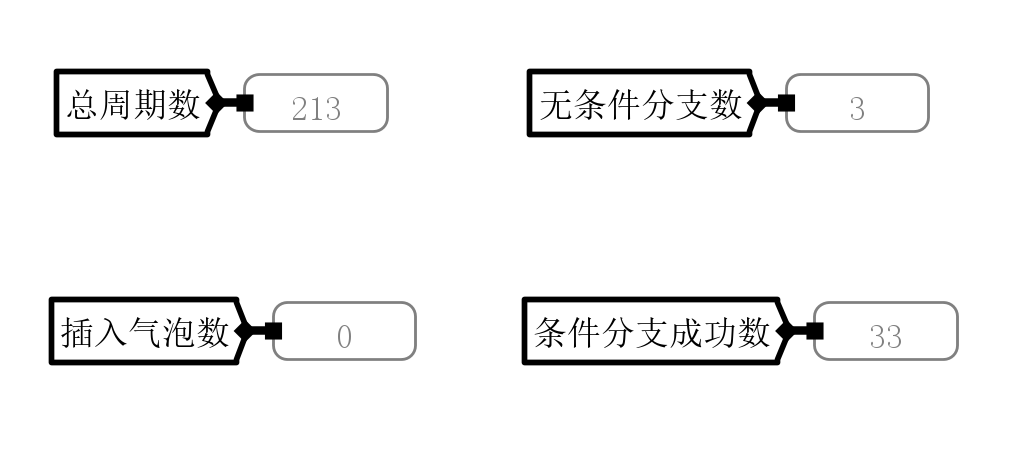
3.输出信号

DataHazard：将上述四路检测结果通过或门进行汇总，只要有一路满足数据相关条件，DataHazard信号就会变为高电平，表明检测到数据相关，通知处理器采取相应措施（如阻塞流水线等 ）来解决数据冲突问题。

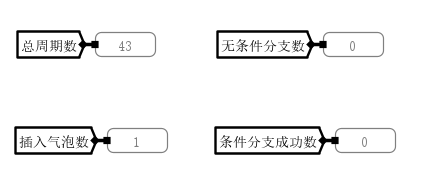
**③在重定向流水线MIPS处理器上运行测试程序。**

（1）在重定向流水线MIPS处理器的数据通路上运行test3-1.hex、 test3-2.hex、 test3-3.hex、 sum\_mips.hex、fib\_mips.hex、 sort1\_mips.hex、 sort2\_mips.hex等程序，记录每个程序运行后的总周期数、插入气泡数，并与气泡流水线对应程序的总周期数、插入气泡数进行比较，得出什么结论？

Test3-1



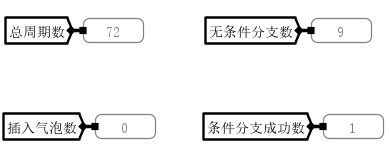
Test3-2



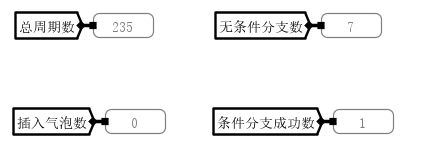
Test3-3



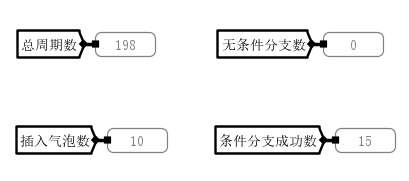
Sum



Fib



Sort1



Sort2



比较：总周期数和插入气泡数都比气泡流水线更少。

结论：重定向流水线处理器通过重定向机制减少了时间的浪费。优化处理器运行效率。

（2）在重定向流水线MIPS处理器的数据通路上运行教材P275例7.2的程序（test3-4.hex），给出该程序运行后的

时空图，并与教材上的图7.28进行比较，观测是否一致？







......



与教材上相比一致。

（3）请分析test3-1.asm、 test3-2.asm、 test3-3.asm程序。

答：

1.程序 1（test3-1.asm）分析

1.程序功能

该程序主要测试分支指令相关性，利用重定向流水线消除分支相关问题。程序初始将s1设为 32，接着调用子程序，在子程序里s1的值逐次减 1，并且通过数码管显示当前s1的值，直至s1变为 0。

2.分支指令测试情况

j指令：程序借助j jmp\_next1指令直接跳转到jmp\_next1标签处，这一过程不会引发分支延迟槽问题。

beq指令：beq $zero,$zero, jmp\_next2的作用类似于j指令，它会跳转到jmp\_next2标签，同样不会产生分支延迟。

bne指令：在bne $zero,$s1, jmp\_next3指令中，由于s1初始值为 32，不等于 0，所以会跳转到jmp\_next3。而在子程序里，bne $s1,$zero,jmp\_func会持续循环，直到s1变为 0。

jal与jr指令：jal jmp\_func用于调用子程序，jr $31则实现子程序的返回，这两个指令的执行也不会出现分支延迟。

3.流水线优化体现

此程序在重定向流水线的作用下，分支指令的执行不会造成流水线阻塞。像beq和bne这类指令在执行阶段就能确定跳转方向，并且能迅速更新 PC 值，有效避免了分支延迟。

2.程序 2（test3-2.asm）分析

1.程序功能

该程序的主要目的是测试数据相关情况，它计算从 0 到 7 的累加和，即 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7，结果为 28（十六进制是 1CH），最后通过数码管显示这一结果。

2.数据相关测试情况

RAW 相关（写后读）：程序里存在多处 RAW 相关情况。例如：

sw $s1,0($s1)和lw $s2,0($s1)，不过因为是对内存进行操作，所以不会引发数据冒险。

addi $s1,$s0,1之后的add $s0,$s0,$s1，这里需要s1的值。

多个addi $s1,$s1,1和后续的add $s0,$s0,$s1组合也存在这种数据相关。

重定向（转发）的运用：重定向流水线能够把 ALU 的输出结果直接转发到 ALU 的输入，从而解决上述 RAW 相关问题，无需插入暂停（Stall）。

3.流水线优化体现

在重定向流水线的支持下，数据能够直接从执行阶段（EX）或者访存阶段（MEM）转发到当前执行指令，使得程序无需额外添加nop指令，就能顺利执行。

3.程序 3（test3-3.asm）分析

1.程序功能

这个程序主要测试寄存器数据相关情况，它会对多个寄存器进行位运算和算术运算操作，然后通过数码管依次显示s1到s5的值。

2.数据相关测试情况

连续指令的数据相关：

and $s1,$s1,$s2之后的sub $s2,$s1,$zero，sub指令要读取s1的值，而这个值是由前一条and指令写入的。

sub $s2,$s1,$zero之后的add $s3,$s1,$s1，add指令同样需要s1的值。

寄存器赋值与运算：程序先给s1、s2、s5、s6赋初始值，然后进行一系列运算，最终得到的结果是：

s1 = 1

s2 = 1

s3 = 2

s4 = 5

s5 = 1

3.流水线优化体现

重定向流水线通过将前一条指令的结果直接转发到下一条指令的输入，成功解决了数据相关问题。程序中原本注释掉的nop指令在使用转发技术后就不再需要了。

（4）分析重定向流水线MIPS处理器的数据通路电路。

答：重定向（转发）流水线 MIPS 处理器数据通路，旨在通过转发技术解决数据冲突（RAW 冒险），结合控制信号处理分支冲突，让指令在流水线各阶段高效流转，提升执行效率，减少因数据依赖或分支导致的停顿。

关键模块与信号解析

① 流水线寄存器与控制信号：

IF/ID 阶段：

IF/ID.CLR = BranchTaken：分支跳转（BranchTaken 为高）时，清空 IF/ID 寄存器，丢弃错误预取指令，保证分支后指令正确流入。

IF/ID.EN = ~Stall：Stall（数据冲突需暂停）为高时，禁用 IF/ID 寄存器，暂停取指 → 译码阶段指令传递，避免错误执行。

PC.EN = ~Stall：与 IF/ID.EN 联动，Stall 有效时，程序计数器（PC）停止递增，暂停新指令取指。

ID/EX 阶段：

ID/EX.CLR = BranchTaken + LoadUse：分支跳转（BranchTaken）或发生 “Load - Use” 数据冲突（访存指令后紧接依赖其结果的指令）时，清空 ID/EX 寄存器，消除错误中间结果影响。

② 转发（重定向）路径

RsForward、RtForward：为解决 “写后读”（RAW）冲突，从执行阶段（EX）、访存阶段（MEM） 直接转发结果到当前指令的操作数。

若某指令需用的寄存器 Rs/Rt，其最新值还在 EX/MEM 阶段未写回寄存器堆，通过多路选择器选通转发路径，将 EX/MEM 阶段的结果直接送入 ALU，无需等待写回，避免流水线停顿。

③ 分支与数据冲突处理：

分支冲突：依赖 BranchTaken 信号，配合 IF/ID.CLR、ID/EX.CLR 清空流水线寄存器，快速切换指令流，减少分支延迟。

数据冲突：除转发外，若转发无法解决（如 Load 指令后紧接依赖的指令，访存未完成），则触发 Stall，暂停流水线，待数据准备好再继续。

工作流程与优化逻辑：

取指 → 译码：PC 驱动取指，指令存入 IF/ID；BranchTaken 或 Stall 控制寄存器使能 / 清空，决定指令是否流转。

译码 → 执行：ID 阶段分析指令依赖，若检测到数据冲突，优先通过 RsForward/RtForward 转发 EX/MEM 结果到 ALU；若需暂停（如 Load - Use 无法转发），触发 Stall 冻结流水线。

执行 → 访存 → 写回：EX 结果可通过转发提前供给后续指令，MEM 阶段数据也可转发；分支跳转时，及时清空 ID/EX 寄存器，保证新指令流正确执行。

（5）分析重定向流水线MIPS处理器的重定向检测电路。

答：核心功能：

重定向（转发）检测电路的作用是识别数据冲突场景，决定是否启用转发（重定向）路径，让后续指令直接使用 EX/MEM 阶段的结果，避免流水线因 “写后读（RAW）” 冲突停顿。

关键信号与逻辑

① 输入信号

寄存器编号：ID.Rs（当前指令源寄存器 Rs）、ID.Rt（当前指令源寄存器 Rt）、EX.WriteReg（执行阶段要写的寄存器）、MEM.WriteReg（访存阶段要写的寄存器）。

控制信号：EX.RegWrite（执行阶段写寄存器使能）、MEM.RegWrite（访存阶段写寄存器使能）、EX.MemRead（执行阶段访存操作标识，用于检测 Load - Use 冲突 ）。

指令类型：通过 OP、Func 编码判断指令是否使用 Rs/Rt（即 RsUsed、RtUsed，表示当前指令是否依赖 Rs/Rt 寄存器 ）。

②转发逻辑（以 RsForward 为例）

电路通过组合逻辑判断 “当前指令的 Rs 依赖，是否可从 EX 或 MEM 阶段转发结果”，规则对应代码：

if (RsUsed & (rs≠0) & EX.RegWrite & (rs==EX.WriteReg#))

RsForward=2 // 从 EX 阶段转发（结果已算好，直接用）

elif (RsUsed & (rs≠0) & MEM.RegWrite & (rs==MEM.WriteReg#))

RsForward=1 // 从 MEM 阶段转发（访存完成，结果可用）

else

RsForward=0 // 无需转发，从寄存器堆读

RsForward=2：EX 阶段结果直接送 ALU，避免等写回寄存器堆。

RsForward=1：MEM 阶段结果送 ALU，同理减少停顿。

③ Load - Use 冲突检测

LoadUse 信号专门检测 “Load 指令后紧接依赖其结果的指令”，逻辑：

LoadUse = RsUsed & (rs≠0) & EX.MemRead & (rs==EX.WriteReg#)

+ RtUsed & (rt≠0) & EX.MemRead & (rt==EX.WriteReg#)

若 LoadUse=1，说明 Load 指令的结果还在访存（未到写回阶段），转发也无法解决，需触发 Stall 暂停流水线，等数据准备好再继续。

工作流程

识别依赖：通过 RsUsed、RtUsed 判断当前指令是否用 Rs/Rt 寄存器。

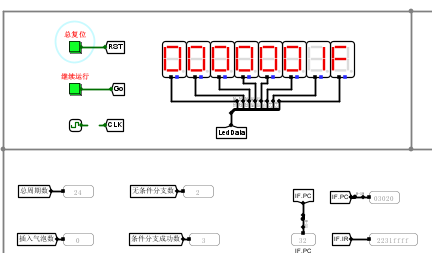
检测转发条件：对比 ID.Rs/Rt 与 EX.WriteReg、MEM.WriteReg，结合 EX.RegWrite、MEM.RegWrite，决定是否从 EX/MEM 转发。

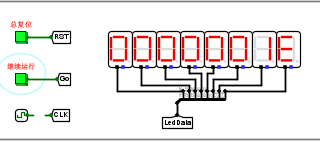
处理特殊冲突：若检测到 LoadUse，直接触发停顿，避免错误执行。

**④在动态分支预测流水线MIPS处理器上运行测试程序。**

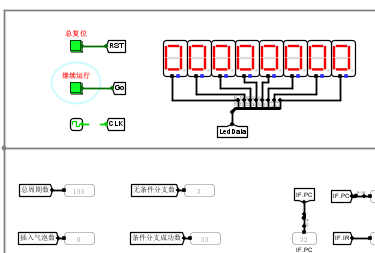
（1）在动态分支预测流水线MIPS处理器的数据通路上运行test4-1.hex、 test4-2.hex、 test4-3.hex、 test4- 4.hex、 sum\_mips.hex、 fib\_mips.hex、 sort1\_mips.hex、 sort2\_mips.hex等程序，记录每个程序运行后的总周期数、插入气泡数，并与气泡流水线、重定向流水线对应程序的总周期数、插入气泡数进行比较，得出什么结论？

① test4-1.hex程序执行后，过一段时间，数码管显示1F，之后减1、显示1E，一直到0，程序结束。

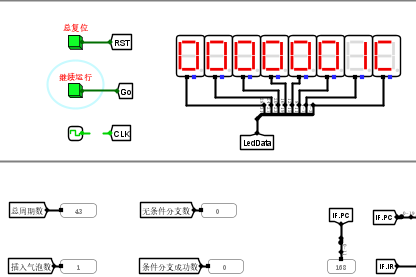




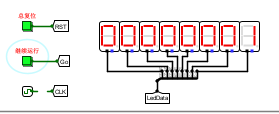
......

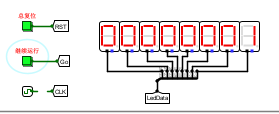


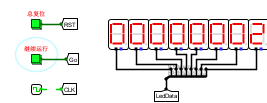
② test4-2.hex程序执行后，过一段时间，数码管显示1C。

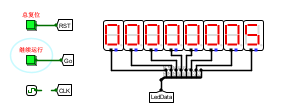


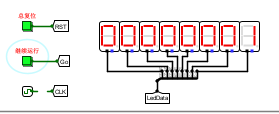
③ test4-3.hex程序执行后，过一段时间，数码管依次显示： 1、 1、 2、 5、 1。



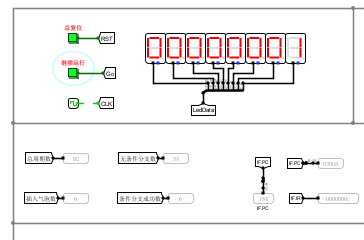




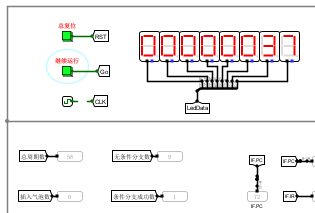




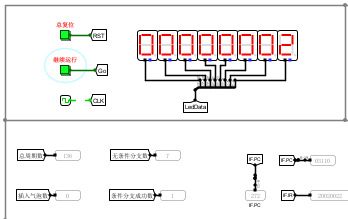
④ test4-4.hex程序执行后，过一段时间，数码管显示： 1。



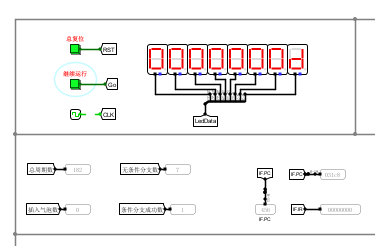
⑤ sum\_mips.hex程序执行后，过一段时间，数码管显示37H（55）。



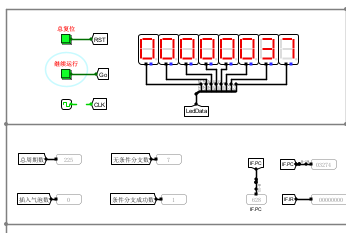
⑥ fib\_mips.hex程序执行后，过一段时间，数码管依次显示： 0、 1、 1、 2、 3、 5、 8、 d、 15、 22、 37。



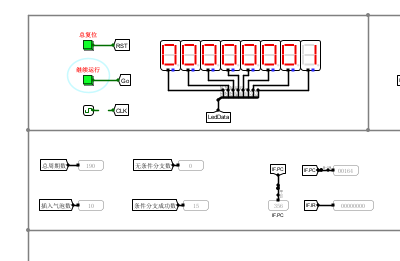
......



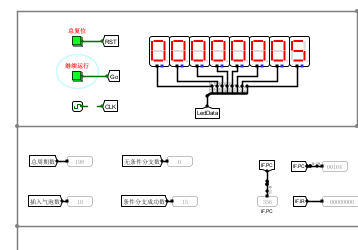
.......



⑦ sort1\_mips.hex该程序执行后，过一段时间，数码管依次显示5、 4、 3、 2、 1。



⑧ sort2\_mips.hex该程序执行后，过一段时间，数码管依次显示1、 2、 3、 4、 5。



（2）请分析test4-1.asm、 test4-2.asm、 test4-3.asm、 test4-4.asm程序。

答：

1.程序 1（test4-1.asm）分析

1.程序功能

此程序的主要功能是测试分支指令的相关性，借助动态分支预测流水线来消除分支相关问题。程序一开始把s1的值设为 32，随后调用子程序，在子程序里s1的值会逐次减 1，并且通过数码管显示当前s1的值，直到s1变为 0。

2.动态分支预测的运用

j指令：像j jmp\_next1这类无条件跳转指令，预测起来比较简单，直接跳转即可，不会造成流水线阻塞。

beq与bne指令：

beq $zero,$zero, jmp\_next2的作用类似于j指令，会直接进行预测跳转。

bne $zero,$s1, jmp\_next3由于s1初始值为 32，不等于 0，所以会预测跳转。

在子程序中，bne $s1,$zero,jmp\_func会进入循环，动态预测器能够迅速学习到这个循环模式，减少预测错误的情况。

子程序调用与返回：jal jmp\_func和jr $31指令的执行依赖于返回地址栈（RAS），动态预测器可以利用 RAS 来准确预测返回地址。

2.程序 2（test4-2.asm）分析

1.程序功能

该程序的目的是测试数据相关情况，它计算从 0 到 7 的累加和，即 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7，结果为 28（十六进制是 1CH），最后通过数码管显示这一结果。

2.数据相关与转发机制

程序中存在多处 RAW（写后读）相关情况，例如：

addi $s1,$s0,1之后的add $s0,$s0,$s1。

多个addi $s1,$s1,1和后续的add $s0,$s0,$s1组合。

不过，由于采用了转发（旁路）机制，数据能够直接从执行阶段（EX）或者访存阶段（MEM）转发到当前指令的 ALU 输入，所以不需要插入暂停（Stall）。

3.动态预测的作用

虽然这个程序主要测试数据相关，但动态分支预测对于循环结束的预测（如最后一次不跳转）会有帮助，能够减少分支延迟。

3.程序 3（test4-3.asm）分析

1.程序功能

此程序着重测试寄存器数据相关情况，它对多个寄存器进行位运算和算术运算操作，之后通过数码管依次显示s1到s5的值。

2.数据相关与指令序列

指令间的数据依赖：

and $s1,$s1,$s2之后的sub $s2,$s1,$zero，sub指令需要读取s1的值，而这个值是由前一条and指令写入的。

sub $s2,$s1,$zero之后的add $s3,$s1,$s1，add指令同样依赖s1的值。

动态转发的优化：动态分支预测流水线结合转发机制，能够把前一条指令的结果直接转发给下一条指令，避免了数据冒险，使得程序无需插入nop指令就能正确执行。

4.程序 4（test4-4.asm）分析

1.程序功能

该程序的主要功能是全面测试动态分支预测流水线，它通过一系列的跳转指令来填充分支预测缓冲区（BPB），并且验证预测器在不同场景下的性能。

2.动态分支预测测试

BPB 的载入过程：程序中连续使用j指令（如j jmp\_next1到j jmp\_next8），每次跳转都会在分支预测缓冲区（BPB）中新增一个表项，最终 BPB 会包含 8 个表项。

循环预测情况：

addi $s1,$s1,-1和bne $s1,$zero,jmp\_next2构成了一个循环，初始时循环 5 次。动态预测器能够快速学习到这个循环模式，提高预测准确率。

特殊分支指令测试：

beq $s0,$s0,jmp\_next9是一个始终跳转的分支。

bne $s1,$s1,jmp\_next10是一个始终不跳转的分支。动态预测器需要对这些特殊情况进行准确预测。

子程序调用测试：jal func和jr $ra用于测试返回地址栈（RAS）的功能，确保子程序能够正确返回。

总结：

动态分支预测相比静态预测（如总是预测跳转）更为智能，它能够根据分支的历史行为来调整预测策略，从而减少分支预测错误带来的损失。

（3）分析动态分支预测流水线MIPS处理器的数据通路电路。

答：动态分支预测流水线 MIPS 处理器数据通路，在基础流水线架构上，加入动态分支预测模块（如预测器、分支目标缓冲 BTB 等），通过预测分支方向，提前预取指令，减少分支停顿；同时保留转发（重定向）、停顿控制，解决数据冲突与预测错误后的流水线刷新。

关键模块与信号解析

① 流水线寄存器与控制信号

IF/ID 阶段：

IF/ID.CLR = BranchTaken：分支实际跳转（BranchTaken 为高）时，清空 IF/ID 寄存器，丢弃错误预取的指令流。

IF/ID.EN = ~Stall：Stall（数据冲突需暂停）为高时，禁用 IF/ID 寄存器，暂停取指 → 译码阶段的指令传递。

PC.EN = ~Stall：与 IF/ID.EN 联动，Stall 有效时，程序计数器（PC）停止递增，暂停新指令取指。

ID/EX 阶段：

ID/EX.CLR = BranchTaken + PredictErr：分支实际跳转（BranchTaken）或预测错误（PredictErr，预测方向与实际不符）时，清空 ID/EX 寄存器，刷新错误执行的中间指令，保证流水线正确流转。

②动态分支预测相关模块

预测器与 BTB（分支目标缓冲）：

取指阶段（IF）通过 PC 索引 BTB，预存分支指令的目标地址和预测方向（如是否跳转），提前预取目标指令流，减少分支延迟。

译码 / 执行阶段（ID/EX）验证实际分支方向，若与预测不符（PredictErr=1），触发 ID/EX.CLR 清空流水线，重新从正确分支地址取指。

预测验证与修正：执行阶段对比 “预测方向” 和 “实际分支结果”，若不一致，标记 PredictErr，驱动流水线刷新，同时更新 BTB 中的预测记录，提升后续预测准确率。

③ 转发（重定向）路径

RsForward、RtForward：与重定向流水线逻辑一致，检测数据冲突（RAW 冒险），从 \*\* 执行阶段（EX）、访存阶段（MEM）\*\* 直接转发结果到当前指令的操作数，避免因等待寄存器写回而停顿。

工作流程与优化逻辑

取指 → 译码：

PC 驱动取指，同时 BTB 依据 PC 预测分支方向，预取目标指令；若检测到 Stall（数据冲突），暂停 PC 递增和指令流转。

分支实际跳转时，IF/ID.CLR 清空预取的错误指令。

译码 → 执行：

分析指令依赖，通过 RsForward/RtForward 转发 EX/MEM 结果，解决数据冲突；若遇 Load - Use 等无法转发的冲突，触发 Stall。

验证分支预测结果，若预测错误（PredictErr=1），ID/EX.CLR 清空流水线，重新从正确地址取指，同时更新预测器。

执行 → 访存 → 写回：

EX 结果可通过转发提前供给后续指令；MEM 阶段完成访存，结果也可转发。

分支结果确认后，若与预测不符，及时刷新流水线，保证指令流正确。

（4）分析动态分支预测流水线MIPS处理器的BHT（分支历史表）电路。

答：核心功能：分支历史表是动态分支预测的核心存储结构，用于记录分支指令的历史执行行为（如是否跳转），通过 “历史预测未来”，让处理器提前预取指令，减少分支停顿。

关键电路模块解析

①存储阵列（核心表项）

结构：由多个存储单元组成，每个单元对应一条分支指令（或通过 PC 索引的分支位置），存储分支历史信息（如 2 位饱和计数器、跳转记录位等 ）。

作用：通过历史执行结果（如 “最近两次跳转”），预测当前分支是否会跳转，指导取指阶段预取指令。

②索引与读写逻辑

PC 索引：用分支指令的 PC（程序计数器）值作为地址，查询 BHT 中对应的历史记录。

写操作：分支指令执行完成后（确认实际是否跳转），更新对应表项的历史信息（如计数器递增 / 递减、翻转跳转标记 ）。

读操作：取指 / 译码阶段，依据 PC 索引 BHT，快速获取预测结果（是否跳转），驱动指令预取。

③预测与更新电路

预测逻辑：根据 BHT 表项的历史数据（如 2 位计数器：00/01 预测不跳转，10/11 预测跳转 ），输出预测方向（PredictTaken 或 PredictNotTaken ）。

更新逻辑：分支指令执行后，对比 “预测方向” 和 “实际结果”，若预测错误，调整表项计数器（如饱和计数器加 1 / 减 1 ），修正历史记录。

工作流程

取指阶段：

分支指令的 PC 送入 BHT，索引对应表项，读取历史预测信息（如是否跳转）。

处理器依据预测结果，预取 “跳转目标地址” 或 “顺序下一条地址” 的指令流。

执行阶段：

分支指令实际执行，确定是否跳转（BranchTaken 信号）。

对比 “预测方向” 和 “实际结果”，若不一致（PredictErr），触发流水线刷新（清空错误预取指令）。

更新阶段：

根据实际跳转结果，更新 BHT 对应表项的历史记录（如调整饱和计数器），为下次预测提供更准确的依据。

**请按照实验课件的要求，验证测试程序，将运行结果黏贴到实验报告中，给出相关的文字说明，并对有关的程序和电路原理进行分析。**

1. **实验报告提交**
2. **实验报告命名为：学号+姓名+第7次实验报告.pdf。**
3. **将实验报告上传到FTP上，第7次实验报告提交截止时间（2周内）：2025年6月4日晚上24点。**
4. **实验报告上传到：/上传作业/曾文华/计组（1班）（2024-2025第2学期）/第7次实验。**