上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

本科课程项目报告

UNDERGRADUATE COURSE PROJECT REPORT

基于 RISC-V 的处理器设计实现

姓　　名:	@ShabbyGayBar
学　　号:	1145141919810
课　　程:	MST3305-通用及图形处理器架构与系统
任课教师:	景乃锋
学院(系) :	电子信息与电气工程学院
开课学期:	2024年 (春季)

2024 年 6 月 16 日

目 录

第一章　项目简介

本项目实现了一个单周期以及一个5级流水线处理器。两种处理器均由 Verilog 语言编写，支持 RV32I Base Integer 指令集内的所有指令，包括:

```
add, sub, xor, or, and, sll, srl, sra, slt, sltu  // R-type
addi, xori, ori, andi, slli, srli, srai, slti, sltiu, lw, jalr  // I-type
sw  // S-type
beq, bne, blt, bge, bltu, bgeu  // B-type
lui, auipc  // U-type
jal // J-type
```

处理器顶层接口定义如表1–1所示。

表 1–1 处理器顶层接口定义

名称	输入/输出	说明
clk	输入	时钟信号
rst	输入	异步复位信号，高电平有效
inst_i	输入	输入指令
inst_addr_o	输出	指令地址输出
inst_ce_o	输出	指令使能信号输出
data_i	输入	读入的数据
data_we_o	输出	写使能控制信号
data_ce_o	输出	数据控制信号
data_addr_o	输出	读/写数据地址
data_o	输出	写出的数据

第二章　设计流程

2.1　顶层模块

流水线处理器的顶层模块划分如图2–1所示[1]，其中指令和数据存储器作为外接设备，通过接口与处理器其他单元互动，本项目中不进行实现。

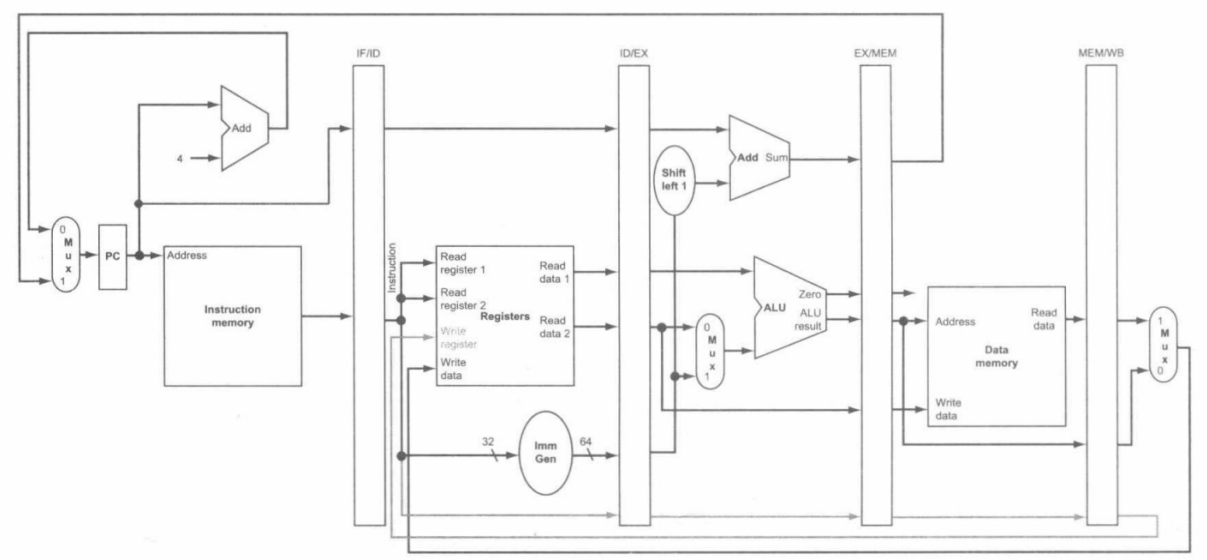

图 2–1 处理器顶层模块划分

在顶层设计中，每个流水段的组合逻辑被整合为一个模块 (.v 文件)。例如指令解码器和立即数生成器集成在 `riscv_id.v` 中，ALU 和 EX 阶段的 PC 生成单元集成在 `riscv_ex.v` 中。而寄存器被单独实现。这样的设计使得每个流水段模块可以独立测试，方便调试。

2.2　单周期处理器

单周期处理器作为流水线处理器的基础和蓝本，也有必要参照流水线处理器的设计思路进行模块划分，以便后续设计。与图2–1相比，单周期处理器不需要在相邻流水段之间插入寄存器，而是将所有组合逻辑之间直接相连。以下分别介绍每一流水段的设计思路和接口定义。

2.2.1 指令读取模块

指令读取 (Instruction Fetch, IF) 模块是单周期处理器中唯一需要时序控制的模块。其负责从指令存储器中读取指令，并将指令传递给下一个流水段。在单周期处理器中，指令读取模块的功能相对简单，只需实现程序计数器 (Program Counter, PC) 的自增、分支功能即可，当前指令由指令存储器接口 inst_i 提供，无需由此模块输出。程序计数器作为指令存储器的地址输入，用于指示当前指令的地址。IF 模块接口定义如表2–1所示。

表 2–1 指令读取模块接口定义

名称	输入/输出	说明	来源/去向
clk	输入	时钟信号	外部
rst	输入	异步复位信号	外部
br_i	输入	分支信号	EX
pc_i	输入	下一程序计数器	EX
pc_o	输出	当前程序计数器	指令存储器

在每个时钟上升沿，程序计数器默认会自增4，即指向下一条指令。然而当分支信号 br_i 为1时，程序计数器会跳转到 pc_i 所指示的地址。异步复位信号 rst 为1时，程序计数器为0。

2.2.2 指令译码模块

指令译码 (Instruction Decode, ID) 模块负责解析指令的操作码，识别指令类型，从指令中提取操作数，还包含了寄存器堆以供读写。另外，为了方便，本项目将图2–1中原属于执行 (EX) 模块的 ALU 输入部分的组合逻辑也整合进了译码器，使得 ID 模块的输出可直接提供给 ALU。ID 模块接口定义如表2–2所示。

除了组合逻辑外，ID 模块还包含了一个寄存器堆，用于存储32个通用寄存器的值。寄存器堆的读操作为组合逻辑，由 ID 模块的 rs1_idx_i 和 rs2_idx_i 控制，输出 rs1 和 rs2 的实时值 rs1_val_o 和 rs2_val_o；写操作为时序逻辑，由 ID 模块解析当前指令得到的 rd_we 控制，在每个时钟上升沿将 WB 模块的 rd_val_i 写入目的寄存器。

2.2.3 执行模块

执行 (Execute, EX) 模块负责执行指令，包括算术逻辑运算、移位运算、比较运算、分支跳转等。EX 模块接口定义如表2–3所示。

表 2–2 指令译码模块接口定义

名称	输入/输出	说明	来源/去向
inst_i	输入	当前指令	指令存储器
rd_val_i	输入	目的寄存器写入数据	WB
rs2_val_o	输出	rs2 寄存器值，用于 S 型指令	数据存储器
alu_op_o	输出	ALU 操作码	EX
alu_a_o	输出	第一个操作数，通常为 rs1	EX
alu_b_o	输出	第二个操作数	EX
offset_o	输出	B 型指令中的 PC 偏移量	EX
br_o	输出	分支信号	EX
zero_en_o	输出	零标志位使能，用于分支判断	EX
data_we_o	输出	数据存储器写使能	数据存储器, WB
data_re_o	输出	数据存储器读使能，同Mem2Reg[1]	WB

表 2–3 执行模块接口定义

名称	输入/输出	说明	来源/去向
pc_i	输入	当前程序计数器	IF
alu_op_i	输入	ALU 操作码	ID
alu_a_i	输入	第一个操作数	ID
alu_b_i	输入	第二个操作数	ID
zero_en_i	输入	零标志位使能	ID
br_i	输入	分支信号	ID
offset_i	输入	分支跳转偏移量	ID
br_o	输出	分支信号	IF
pc_o	输出	下一程序计数器	IF
data_addr_o	输出	数据存储器地址 (即 ALU 运算结果)	数据存储器

EX 模块主要由两部分组成：ALU 和分支跳转。

ALU 根据操作码 alu_op_i 执行运算，将结果 data_addr_o 传递给数据存储器。

分支跳转组合逻辑则在 br_i 为高 (即 ID 模块解析到 B 型指令) 时根据 ALU 的零标志位判断是否需要跳转地址，计算下一程序计数器 pc_o 并将其与分支信号 br_o 和输出给 IF 模块。

2.2.4 写回模块

写回 (Write Back, WB) 模块负责将执行模块的运算结果写回寄存器堆。WB 模块接口定义如表2–4所示。

表 2–4 写回模块接口定义

名称	输入/输出	说明	来源/去向
data_i	输入	数据存储器读出数据	数据存储器
data_addr_i	输入	数据存储器地址	EX
data_re_i	输入	数据存储器读使能，同Mem2Reg[1]	ID
rd_val_o	输出	目的寄存器写入数据	ID (寄存器堆)

WB 模块实际上就是一个2输入1输出的多路选择器 (Multiplexer)。根据 data_re_i 信号，选择数据存储器读出的数据 data_i 或者 EX 模块的运算结果 data_addr_i，作为写入目的寄存器的数据 rd_val_o。

2.2.5 Vivado 综合结果

2.3 流水线处理器

流水线处理器的设计基于单周期处理器，将五个流水段：取指 (IF)、译码 (ID)、执行 (EX)、访存 (MEM)、写回 (WB) 的组合逻辑分离出来，每两段之间插入寄存器，并增加了停顿与前递等数据冒险处理逻辑，如图2–1所示。

组合逻辑只需要细微的改动，寄存器的实现逻辑也较简单，此处不再赘述。以下将介绍流水线处理器中的数据冒险处理逻辑。

2.3.1 停顿

停顿 (Stalling) 是一种数据冒险处理技术。当一个指令需要读取另一条指令尚未写回的结果时，流水线处理器会在执行阶段停顿一周期，等待数据准备好。需要使用停顿的情境通常有:

1. 上一指令为 load 指令，当前的 ID 阶段却需要读取 load 指令的结果。
   此时只需要暂停 IF 和 ID 阶段的指令传递即可，下一周期 ID 模块的译码器将通过前递逻辑直接读取数据存储器的结果。
2. 分支信号激活，需要跳转到另一地址。
   此时则应该通过清零 ID-EX 寄存器插入一个气泡，实现 ID 阶段的停顿。
3. 指令或数据存储器由于读写冲突等原因暂时无法读写。
   这种情况由于顶层模块以及指令、数据存储器并没有相应的接口，因此不予考虑。

2.3.2 前递

前递 (Forwarding) 指将 EX 阶段的运算结果或 MEM 阶段的数据直接传递给 ID 阶段，是另一种数据冒险处理技术。需要前递的情境通常是当一个指令需要读取另一条指令尚未写回的结果时，因此本项目中，前递逻辑主要集中在执行阶段和译码阶段之间，以及访存阶段和译码阶段之间。

值得注意的是，教材[1]中，前递信号被传递给了 EX 模块中的多路选择器 (MUX)，将其他流水段寄存器的值作为 ALU 的输入。然而在本项目中，如第2.2.2节所述，这部分组合逻辑被整合到了 ID 模块中，因此前递信号直接取代了 ID 模块的 rs1_val_id 和 rs2_val_id。原先寄存器堆的输出 rs1_val_o 和 rs2_val_o 作为前递控制逻辑的输入，在前递信号有效时，其值将被前递的数据覆盖。

2.3.3 Vivado 综合结果

第三章 测试

3.1 代码例化层次

本项目的测试 (Testbench) 的顶层代码为 `riscv_soc_tb.v`，其中调用了处理器的顶层模块 `riscv.v`，以及指令 (inst_mem) 和数据存储器 (data_mem)，例化层次如图3–1所示。

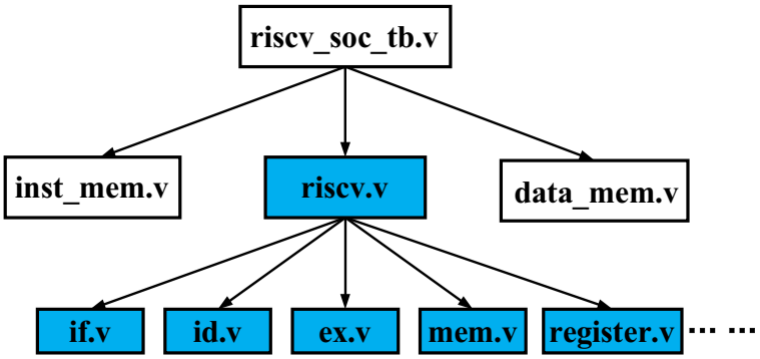

图 3–1 处理器测试代码例化层次

其中，`riscv_soc_tb.v` 和存储器的接口与代码由课程组提供，不得更改，但是指令存储器和数据存储器的内容可以自行修改。

3.2 测试程序

考虑到课程提供的测试程序 (即指令存储器内容) 涵盖的指令和数据冒险情况较少，因此本项目选择参考 cgyurgyik 的快速排序 RISC-V 汇编程序作为测试程序。该程序将一个数组 (10, 80, 30, 90, 40, 50, 70) 从小到大排序，排序结果为 (10, 30, 40, 50, 70, 80, 90) 并存储在数据存储器的地址 0x0000-0x001b 处。

在此基础上，本项目稍作修改，将排序好的数组写入寄存器 x10-x16，以便在 GTKWave 中查看排序结果。

3.3 仿真流程

由于 cgyurgyik 的程序是由 RISC-V 汇编语言编写的，需要转换成指令存储器中的 32 位二进制指令。而课程组提供的转换程序不支持部分 RV32I 指令集指令，如 slli，因

此本项目采用 Kritagya Agarwal 的 *Assembly To Machine Code Translation* 工具进行转换。

最后，本项目使用 Icarus Verilog 仿真器进行仿真，通过 GTKWave 查看波形图，验证设计的正确性。

3.4 结果分析

单周期处理器和流水线处理器运行测试程序的仿真结果分别如图3–2和图3–3所示。

从波形图中可以看出，x10-x16寄存器最后的值都是排好序的数组，说明单周期处理器和流水线处理器均能正确执行快速排序的测试程序。

放大波形图3–3至19.30 ns附近，如图3–4所示，此时流水线正在执行如下指令:

```
0x0008AF03  // lw x30, 0(x17)
0x01E82023  // sw x30, 0(x16)
0x01C8A023  // sw x28, 0(x17)
0x00138393  // addi x7, x7, 1
0xFC0006E3  // beq x0, x0, LOOP
//0x00130F13  // addi x30, x6, 1, NOT EXECUTED DUE TO STALL SIGNAL
0x02C38C63  // beq x7, x12, LOOP_DONE
```

根据图3–4可以发现，当执行第2条 sw 指令时，前一条 lw 指令的结果还未读取，因此流水线处理器不得不产生 stall 信号，在此处停顿一周期，等待数据读取至 MEM 流水段的 data_i 中，并通过前递逻辑传递给 ID 流水段，从而继续执行 sw 指令。在波形图中体现为 0x01E82023 指令在 stall 信号变为 0x0011 后多占用了一个周期。

而当执行第5条 beq 指令时，由于是否跳转的判断需要在 EX 阶段，即下下个周期才能得出，因此流水线处理器在此处产生 0x0010 的 stall 信号，停顿一周期，等待判断结果。但是此时下一条指令 (addi x30, x6, 1) 已经由 IF 流水段读取，因此需要在此指令的 ID 阶段插入一个气泡，将 ID-EX 寄存器清零，使其成为 nop，实现停顿。

对比两波形图可以发现，在10 GHz时钟频率下，单周期处理器完成测试程序执行所需的时间为66.75 ns，而流水线处理器执行完成所需的时间为75.50 ns，流水线处理器的执行时间比单周期处理器更长，这似乎与流水线处理器的指令吞吐量更高的设计初衷相悖。然而，这是因为流水线处理器由于每个时钟周期内的延迟更少，可以有更高的时钟频率，而测试中这一优势并未体现，而且由于流水线处理器存在数据冒险的情况，需要停顿，因此导致流水线处理器的执行时间更长。假设流水线处理器的时钟频率提高到50 GHz，则流水线处理器的执行时间会压缩到15.10 ns，大大快于单周期处理器。

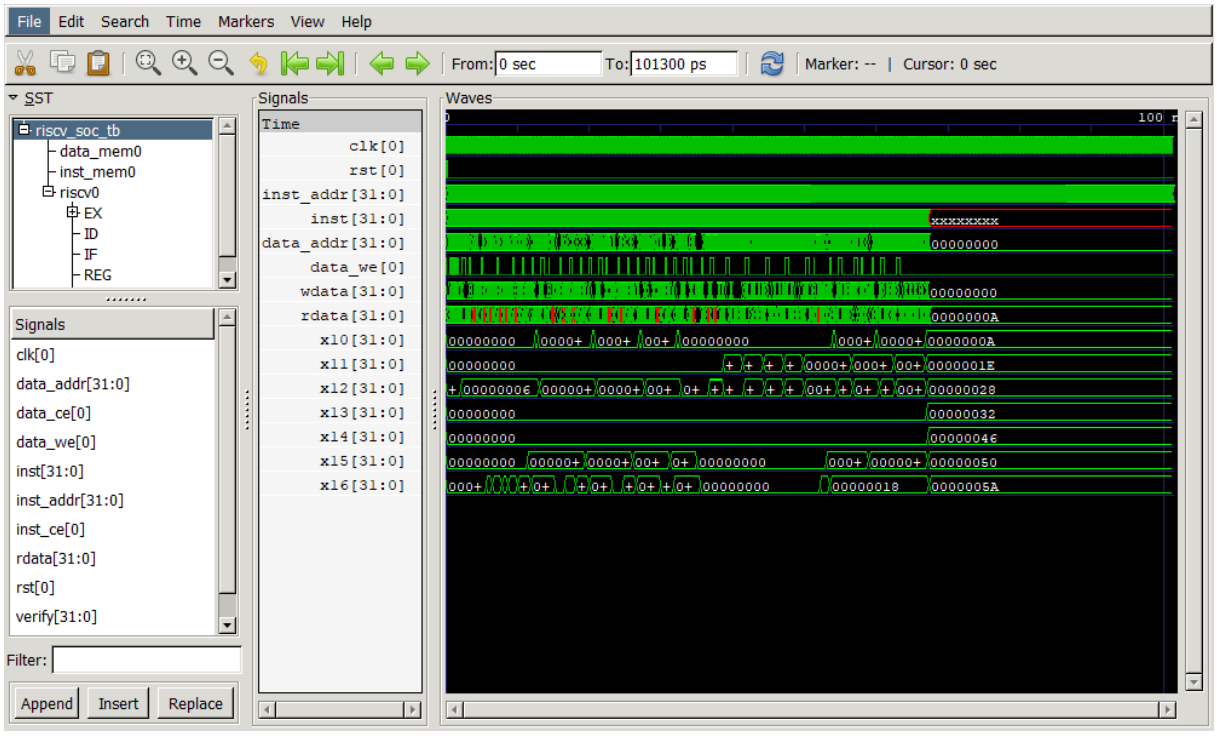

图 3–2 单周期处理器波形图

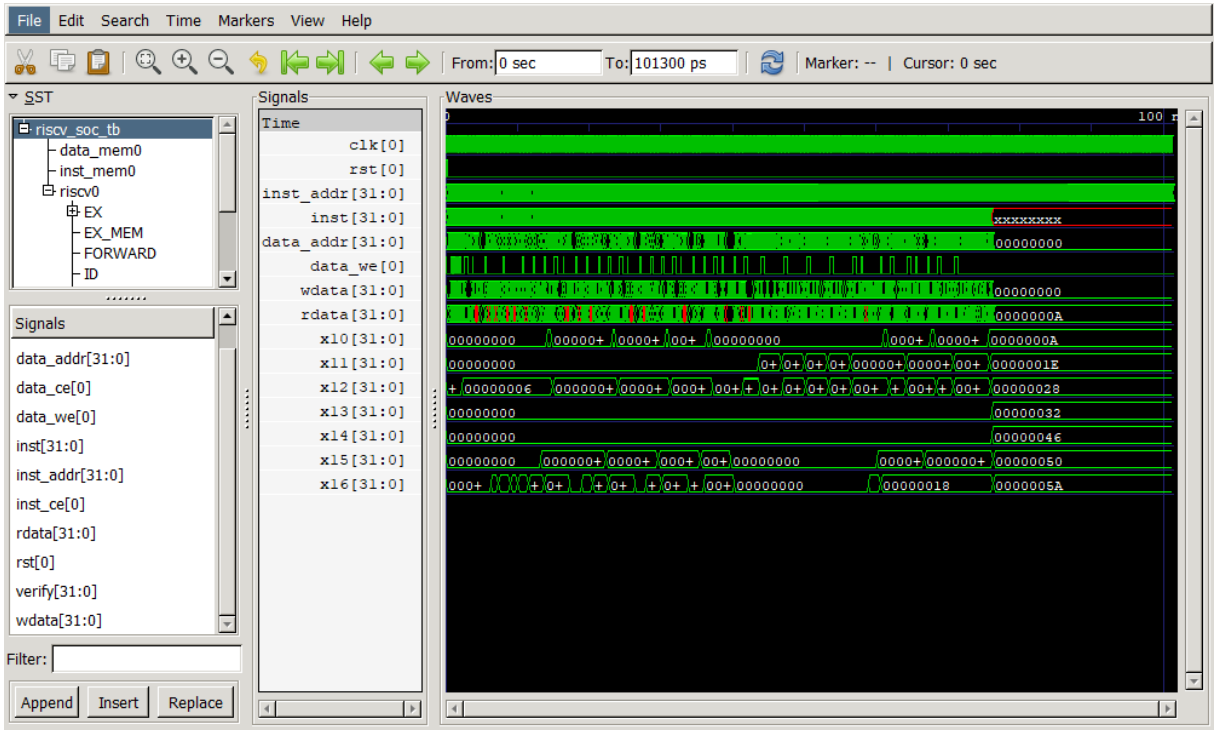

图 3–3 流水线处理器波形图

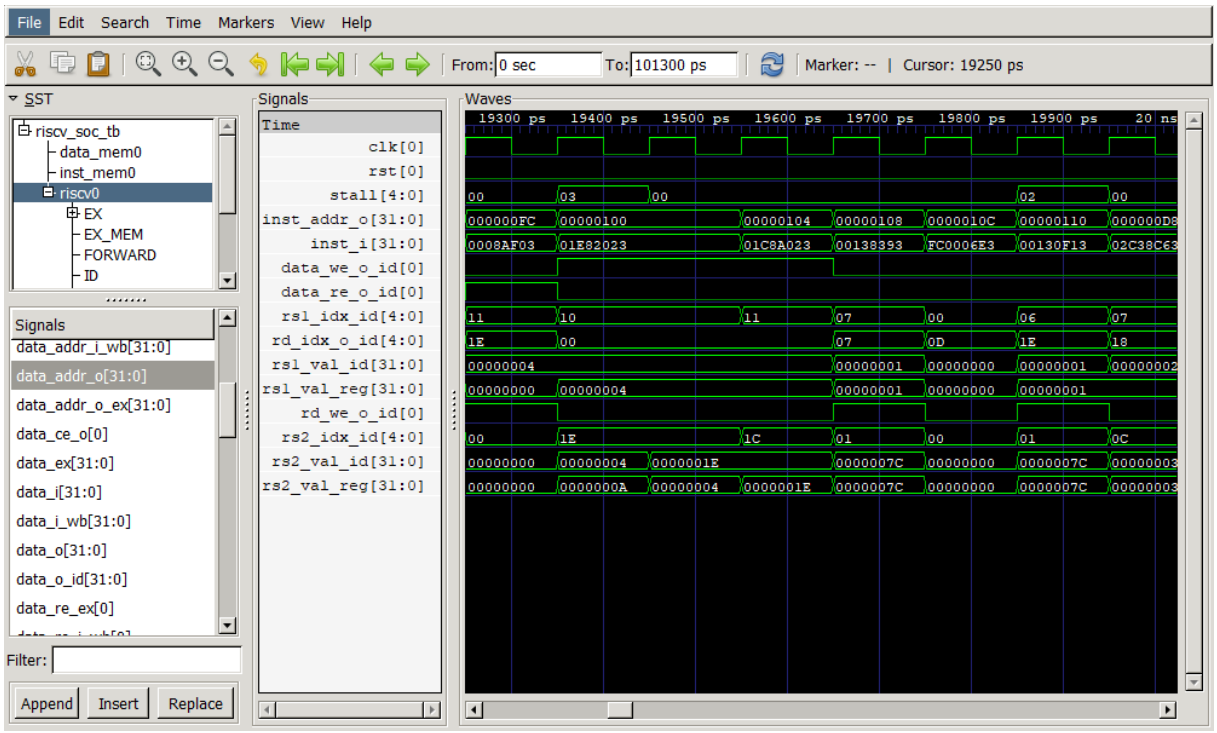

图 3–4 流水线处理器停顿示意图

第四章　总结

本项目实现了一个单周期处理器和一个5级流水线处理器，支持 RV32I Base Integer 指令集内的所有指令。通过仿真测试，验证了两种处理器的功能正确性。流水线处理器的设计使得其在时钟频率更高的情况下能够有更高的性能，但是由于数据冒险的存在，可能导致流水线处理器的延迟更长。

本项目已发表至 GitHub。

参考文献

[1] PATTERSON D A, HENNESSY J L. Computer Organization and Design, Fifth Edition: The Hardware/Software Interface[M]. 5th. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013.