合肥工業大學

《系统硬件综合设计》
报告

学生姓名 余梓俊

学　　号 **2018211991**

专业班级 计算机科学与技术 18-3 班

指导教师 李建华　安鑫　陈田

院系名称 计算机与信息学院

2021 年 1 月 14 日

目录

1. 实验目标

设计并实现一个多周期流水 CPU

- 若干段流水、可以处理冲突
- 三种类型的指令若干条
- MIPS、ARM、RISC-V 等类型 CPU 都可以
- 下载到 FPGA 上进行验证（选）

2. 实验工具及语言

Vivado 2019.2

语言：Verilog

3. 系统设计说明

- 实现了多周期流水线 CPU
- 实现了 Tomasulo 算法，指令顺序流出，乱序执行，解决除结构冲突之外的冲突
- 实现了结构冲突时 stall
- 实现了基于阵列乘法器的乘法运算
- 基于 MIPS 指令集架构的指令格式，实现了十五条指令

4. 系统结构设计

4.1 设计背景

Tomasulo 算法是一种在乱序执行流水线 CPU 中使用的，用于对指令的顺序进行动态调度的算法。在流水线处理器中，先后执行的指令往往具有相关性，如上一条指令将一个数字写进寄存器中，下一条指令马上就要用这一个寄存器中存有的值来进行下一步的计算。这样的数据相关会导致流水线处理器中运行的冲突，这个时候可能可以通过旁路解决，但更多的时候，处理器只能够通过插入一个气泡，堵塞处理器的运行，才能够解决这一种数据冲突。由此，我们可以看出，传统顺序执行的流水线处理器，在处理具有极多数据相关的代码的时候，只能以较低的效率进行计算。

为了降低数据相关带来的处理器停顿时间，一般有两种处理方式。

静态调度 静态调度的流水线依靠编译器对代码进行静态调度，以减少相关冲突。此类调度方式，是通过程序在进行编译的时候就把相关的指令拉开距离，来减少可能产生的停顿。由于是在编译期间进行的指令调度，在程序执行阶段，指令的顺序不能够进行改变，“静态”由此而来。

动态调度 动态的指令调度是在程序的执行过程中，依靠专门的硬件对指令进行调度。该种调度方式能够处理一些编译时情况不明的相关，还能够让代码的执行效率与产生指令的编译器解构。目前许多现代的处理器都采用了这种技术。

在本文中，我们关注对指令的动态调度的算法实现。首先探讨一下指令动态调度算法的可行性，要实现指令的动态调度，也就是说我们需要做到在运行过程中，由 CPU 自行判断哪些指令能够提前运行，同时还能够做到保持数据流和控制异常行为。为了实现这个目的，一个典型的算法是记分牌算法，该算法能够做到与前文无关的指令可以尽早进入执行阶段，从而让处理器的停顿时间减少，但此算法并不能真正解决指令中常常存在的反相关和输出相关，反而，还有可能会让原本的伪相关在乱序执行下变为真相关，从而又从另一个角度增加了处理器的停顿时间。

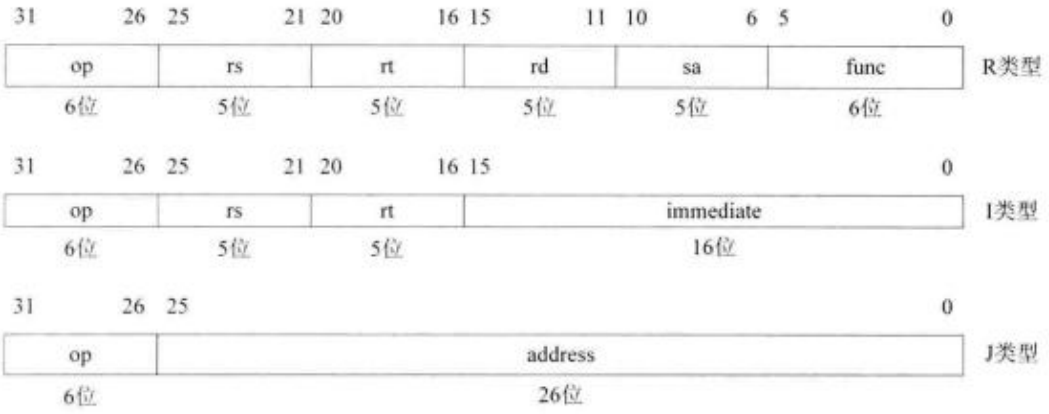
另一类实现动态调度的算法是 Tomasulo 算法，该算法解决了记分牌算法的缺陷。相比起记分牌算法，Tomasulo 算法中有两个重要的突破[2] 让它不仅能够最大限度的减少真相关带来的处理器停顿时间，同时直接解决了反相关和输出相关。这两种技术是

- 寄存器换名技术
- CDB 公共数据总线

后文将会详细描述算法的原理及实现。

4.2 指令集架构

即 MIPS 指令格式。



其中 OP 是指令码、func 是功能码。

（1）R 类型：具体操作由 op、func 结合指定，rs 和 rt 是源寄存器的编号，rd 是目的寄存器的编号。

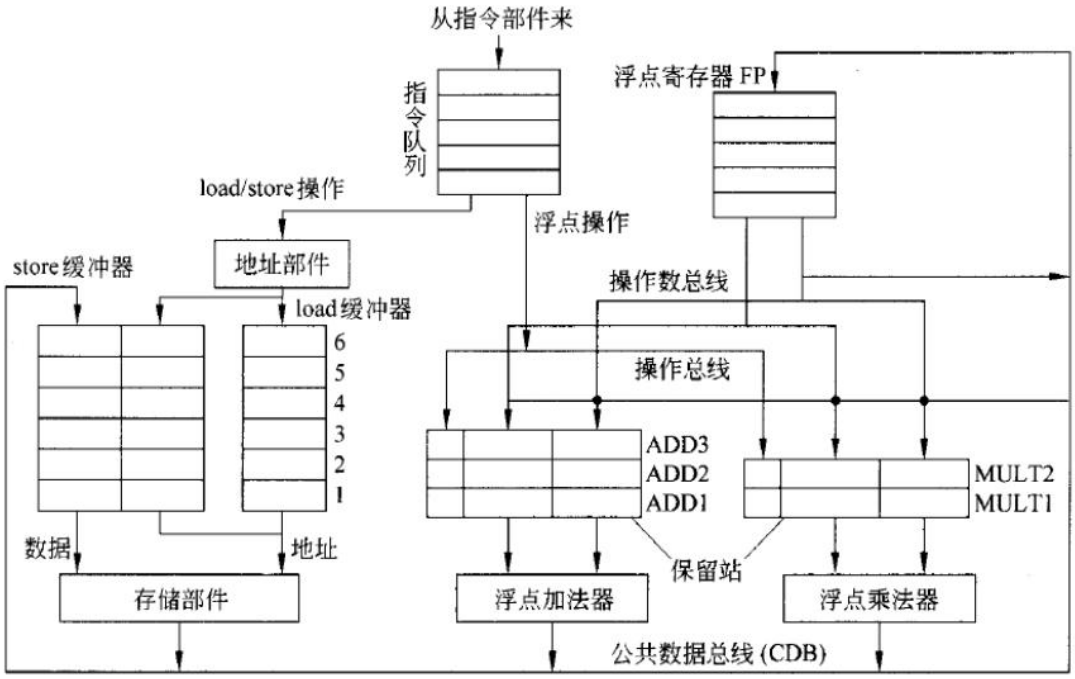
（2）I 类型：具体操作由 op 指定，指令的低 16 位是立即数，运算时要将其扩展至 32 位，然后作为其中一个源操作数参与运算。

（3）J 类型：具体操作由 op 指定，一般是挑战指令，低 26 位是字地址，用于产生跳转的目标地址。

助记符	指令格式						示例	示例含义	操作及其解释
Bit #	31..26	25..21	20..16	15..11	10..6	5..0			
R-type	op	rs	rt	rd	shamt	func			
add	000000	rs	rt	rd	00000	100000	add $1,$2,$3	$1=$2+$3	rd <- rs + rt ；其中 rs＝$2，rt=$3, rd=$1
sub	000000	rs	rt	rd	00000	100010	sub $1,$2,$3	$1=$2-$3	rd <- rs - rt ；其中 rs＝$2，rt=$3, rd=$1
and	000000	rs	rt	rd	00000	100100	and $1,$2,$3	$1=$2 & $3	rd <- rs & rt ；其中 rs＝$2，rt=$3, rd=$1
or	000000	rs	rt	rd	00000	100101	or $1,$2,$3	$1=$2 | $3	rd <- rs | rt ；其中 rs ＝ $2 ， rt=$3, rd=$1
xor	000000	rs	rt	rd	00000	100110	xor $1,$2,$3	$1=$2 ^ $3	rd <- rs xor rt ；其中 rs＝$2，rt=$3, rd=$1(异或）
nor	000000	rs	rt	rd	00000	100111	nor $1,$2,$3	$1=~($2 | $3)	rd <- not(rs | rt) ；其中 rs＝$2，rt=$3, rd=$1(或非）
slt	000000	rs	rt	rd	00000	101010	slt $1,$2,$3	if($2<$3) $1=1 else $1=0	if (rs < rt) rd=1 else rd=0 ；其中 rs＝$2，rt=$3, rd=$1
I-ty	op	rs	rt	immediate					

pe							
addi	001000	rs	rt	immediate	addi $1,$2,100	$1=$2+100	rt <- rs + (sign-extend)immediate ; 其中 rt=$1,rs=$2
andi	001100	rs	rt	immediate	andi $1,$2,10	$1=$2 & 10	rt <- rs & (zero-extend)immediate ; 其中 rt=$1,rs=$2
ori	001101	rs	rt	immediate	andi $1,$2,10	$1=$2 | 10	rt <- rs | (zero-extend)immediate ; 其中 rt=$1,rs=$2
xori	001110	rs	rt	immediate	andi $1,$2,10	$1=$2 ^ 10	rt <- rs xor (zero-extend)immediate ; 其中 rt=$1,rs=$2
lw	100011	rs	rt	immediate	lw $1,10($2)	$1=memory[$2 +10]	rt <- memory[rs + (sign-extend)immediate] ; rt=$1,rs=$2
sw	101011	rs	rt	immediate	sw $1,10($2)	memory[$2+10] =$1	memory[rs + (sign-extend)immediate] <- rt ; rt=$1,rs=$2
slti	001010	rs	rt	immediate	slti $1,$2,10	if($2<10) $1=1 else $1=0	if (rs <(sign-extend)immediate) rt=1 else rt=0 ; 其中 rs＝$2，rt=$1
J-type	op	address					
halt	111111						

4.3 流水线总体思想和 CPU 模块结构



第 1 个上升沿：

取指译码，对应信息送 RegFile 和 CU，PC 自增（CU 的 isFullOut 为真时不自增），RegFile 准备好 dataOut1、2 和 labelOut1、2，CU 给保留站或队列送使能信号并通过 mux4to1 选择空闲的保留站号或者队列号送给 RegFile（尚未写入 RegFile）；

第 2 个上升沿：

若指令对应的保留站或队列未满，则当前指令写入保留站或队列中，上一阶段中 CU 送来的空闲保留站号或队列号写入 RegFile。当前指令中已就绪

的操作数，直接从 RegFile 读进来，从而与寄存器文件解耦合。对于操作数未就绪的指令，在保留站或队列中存好该数据的的最新来源（保留站号或队列号，从寄存器状态表读取），等待数据写入。操作数已就绪的一种特殊情况是此时 CDB 广播的数据恰好是该操作数，此时直接从 CDB 都进来即可。如果指令已经就绪，则从保留站向 ALU 或从队列向 memory 发请求，如果 ALU 向保留站或 memory 向队列发回了 requireAC，保留站置对应项目 busy 为 0 或者队列出队。否则继续保持请求，直到被响应。

第 N+2 个上升沿（假设执行需要 N 个周期）：

保留站完成运算或队列完成存取，向 CDB 发请求，如果 CDB 发回 requireAC，将保留站或队列的 available 置为 1（执行过程中保留站和队列的 available 为 0），否则一直保持请求，直到被响应。CDB 向保留站、队列、寄存器堆广播，在任何一个上升沿，如果广播信号有效，保留站、队列和寄存器堆会监听 CDB，更新需要更新的值。

CPU 结构：

见附件“整体数据通路图.pdf”。

5. 各模块具体实现思路

5.1 CDB and CDBHelper

接收来自 mfALU，pmfALU，memory 的 data 和 label，接收来自 mfState，pmfState，momory 的 require 信号，以此选择一个有效的 data 和 label，

以及一个有效信号 EN 进行广播输出，输出到三个 Queue 和两个 ReservationStation 以及 RegFile。同时也要把选择的是哪个这一信息返回给 mfState，pmfState 和 memory。

5.2 CU

输入：

接收 Decoder 输出的 func 和 op，以及来自两个 ReservationStation 和三个 Queue（三个 queue 只需要一个信号就够了）的 isFull 信号。

输出包括：

ALUSel，即选择哪个 ALU，送给 mux4to1_4，mux4to1_4 根据该值选择空的队列号或者空的保留站号（它的其余输入即为两个保留站和队列的 writeableLabel）；

ALUop，送给两个保留站；

QueueOp，即读或写，送给三个 Queue；ResStationEN，送给两个保留站和三个 Queue；

isFullOut，判断当前指令对应的 ReservationStation 或者 Queue 的 isFull 信号是不是有效，在 top 取反后记为 labelEN 信号，再送给 regFile 的 RegWr（RegWr 为什么要 CU 的 isFull 为 0 才有效，isFull 的时候为什么不能写寄存器？见下文 RegFile 部分）和 PC 的 pcWrite（结构冲突，PC 不应再自增）；

VkSrc，代表 rt 是源操作数（R 型）还是目的操作数（I 型），在 top 中通过 VkSrc，来判断给 RegFile 的 WriteAddr 和 ReadAddr2 送 rt 还是 rd，给两个 ReservationStation 的 dataIn2、label2 送 RegFile 的 DataOut2、Label0ut2，还是送扩展后的立即数、全零的 label。

5.3 RegFile

寄存器堆不仅包括编号为 1 到 31 的（0 号的值总为 0）32 位 regData，还包括与之一一对应的编号为 1 到 31 的 regLabel（保留站号或队列号）。

端口类型	位宽	端口名称	端口说明
input	1	clk	时钟信号
input	1	nRST	清零信号
input	[4:0]	ReadAddr1	读取的寄存器号
input	[4:0]	ReadAddr2	读取的寄存器号
input	1	RegWr	寄存器是否可写
input	[4:0]	WriteAddr	写寄存器号
input	[3:0]	WriteLabel	写寄存器对应的保留站号
output	[31:0]	DataOut1	读取寄存器对应的数据
output	[31:0]	DataOut2	读取寄存器对应的数据
output	[3:0]	LabelOut1	读取寄存器对应的保留站号
output	[3:0]	LabelOut2	读取寄存器对应的保留站号
input	1	BCEN	广播的数据是否可用
input	[3:0]	BClabel	广播的数据对应的保留站号
input	[31:0]	BCdata	广播的数据
input	1	BCEN	广播是否可用
input	[3:0]	BClabel	广播数据对应的保留站号
input	[31:0]	BCdata	广播的数据

具体说明：

输入包括：

BCEN、BCdata、BClabel；

ReadAddr1，即 rs，ReadAddr2，见上文 CU 部分的 VkSrc；

RegWr，见 CU 部分的 isFullOut；

WriteAddr 见 CU 部分的 VkSrc，WriteLabel 见 CU 部分的 ALUSel；

Clk、nRet。

输出为：

DataOut1 和 DataOut2，即用 ReadAddr1 和 ReadAddr2 在寄存器堆中取出值，以及 LabelOut1 和 LabelOut2，即对应的 label。其中 Out1 会送到两个 ReservationStation 和 RS_queue，Out2 会送到两个 ReservationStation 和 RT_queue。

在 nRST 下降沿，置寄存器堆和 label 堆为 0；

在 clk 上升沿，如果 RegWr 有效（如果要用的保留站或队列满了，RegWr 为 0，此时 PC 不应该自增，RegLabel 也不应修改，即结构冲突，流水线应该停下），把 WriteAddr 对应的寄存器的 regLabel 置为 WriteLabel；如果 BCEN 有效，找到与 BClabel 相同的 regLabel，将其值重置为 0，将对应的 regData 置为 BCdata。

5.4 Memory

输入包括：

WEN，由三个队列的 require 信号与得到；

dataIn1 和 dataIn2 分别来自 RS_queue 和 Immd_queue 的 dataOut；

labelIn 来自 RT_queue 的 labelOut，含义为取数指令的目标队列号，writeData 来自 RT_queue 的 dataOut(RegFile 的 Out2 是送到 RT_queue 的)，op 来自 RT_queue 的 opOut，控制读写（opOut 来自哪个 queue 都可以，统一选择 RT_queue 作为三个 queue 的代表，同理包括 labelOut、isFull、queue_writeable_label、require，不过 require 信号三个 queue 都连了出来)；

requireAC 来自 CDB 的反馈信号。

输出包括：

available 输出到三个队列的 requireAC，作为对他们送来的 require 信号的反馈；

isLastState 输出到三个队列的 isLastState,控制队列是否可以进行出队；

require 信号送给 CDB；

labelOut 和 loadData 分别送到 CDB 的 label3 和 data3。

Memory 内部包含一个 RAM 模块，通过维护内部的状态，实现了一个需要 10 个周期进行读或写的数据存储器。Memory 在 clk 的上升沿开始工作，接收来自队列的存取指令,dataIn1 和 dataIn2 分别来自 RS_queue 和 Immd_queue，将这两个值相加得到正确的存取地址。读取操作完成后 labelOut 和 loadData 送至 CDB 并向 CDB 发出 require 信号,其中 labelOut 的值就是 labelIn 的值。

5.5 ReservationStation

端口类型	位宽	端口名称	端口说明
input	1	clk	时钟信号
input	1	nRST	清零信号
input	1	EXEable	该保留站对应的 ALU 是否可执行
input	1	WEN	该保留站是否可写
input	[1:0]	ResStationDst	该保留站的编号
input	[1:0]	opCode	指令对应操作码
input	[31:0]	dataIn1	数据输入端口
input	[3:0]	label1	暴露站好输入端口
input	[31:0]	dataIn2	数据输入端口
input	[3:0]	label2	保留站号输入端口
input	1	BCEN	广播是否可用
input	[3:0]	BClabel	广播数据对应的保留站号
input	[31:0]	BCdata	广播的数据
output	[1:0]	opOut	给 ALU 输出对应的操作码
output	[31:0]	dataOut1	给 ALU 输出对应的数据
output	[31:0]	dataOut2	给 ALU 输出对应的数据
output	1	isFull	保留站是否满
output	1	OutEn	该保留站输出是否有效
output	[3:0]	ready_labelOut	保留站就绪指令的保留站号
output	[3:0]	writeable_labelOut	可写的保留站号

详细说明：

输入包括：

EXEable，该保留站对应的 ALU 是否可执行，来自对应的 mfState 或 pmfState 的 available；

ResStationDst，该保留站的编号；

WEN，来自 CU 的选择信号 ResStationEN，opCode，来自 CU 的 ALUop；

来自 RegFile 的 dataIn1、dataIn2、label1、label2；

来自 CDB 的 BCEN、BClabel、BCdata。

输出包括：

OutEN，送给对应的 mfState 或 pmfState 的 WEN 端口；

opOut，送给对应 ALU（及对应的 State）的 op 端口，dataOut1、dataOut2 送给对应 ALU 的 dataIn，ready_labelOut 就绪指令的保留站号，送给对应 ALU 的 labelIn；

writeable_labelOut，可写的保留站号，送给 mux4to1。

保留站的一部分工作见流水线总体思想的第二个上升沿部分。除此之外，在上升沿到来时，保留站还会根据 CDB 的广播数据，把相应的保留站号的 busy 重置为 0，按需把各保留站项的 Vj、Vk、Qj、Qk 更新。保留站还需要实时地（不需要在上升沿）更新 cur_addr（当前可用的空项中编号最小的那一个）和 ready_addr（哪个项的所有操作数都准备好了，也是小编号优先）。

保留站名称	保留站编号
alu0	0100
alu1	0101
alu2	0110
mul0	1000
mul1	1001
mul2	1010
data0	1100
data1	1101
data2	1110

5.6 Queue

队列实质上也是保留站，其输入输出和行为与保留站类似。但由于存取指令的特殊性，将存取指令的保留站特化为三个队列，即 Immd_queue、RS_queue、RT_queue。

对于存取指令，队列的 require 与 memory 的 WEN 相连，memory 的 available 连回队列的 requireAC,而 memory 的 require 和 requireAC 与 CDB 交互； 对于运算指令，ReservationStation 的 OutEn 与对应的 ALU 的 WEN 相连，ALU 的 available 连回 ReservationStation 的 EXEable，而 ALU 的 require 和 requireAC 与 CDB 交互。

当队首项有效且队首项数据准备好时，require 为有效。当 memory 送来的 isLastState 有效时，说明队首指令在下一个上升沿到来时便可以处理完毕，置 poppable 为 1,队首指令出队，剩余工作交由 memory 与 CDB 交互。

在上升沿到来时，检测 CDB，更新队列中可更新的值。如果 WEN 有效（即收到来自 CU 的选择信号，新流出指令为存取指令），if poppable，出队，新指令放在出队前的最后一个有效项的位置，else，新指令放在队列中第一个空的位置；否则，如果 WEN 无效，if poppable，出队，else，什么都不做。

5.7 ALU

输入包括：

来自 ReservationStation 的 WEN、op、dataIn1、dataIn2、labelIn，来自 CDB 的 requireAC。

输出包括：

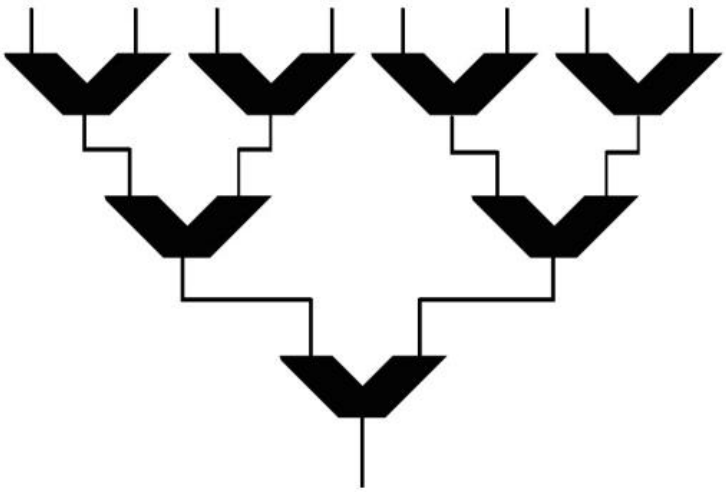
labelOut 和 result，送到 CDB。

ALU 与 ReservationStation 的交互与前述 Memory 与 Queue 的交互逻辑基本相同。

在运算结束前的一个上升沿，require 置为 1，当 require 和 requireAC 均为 1 或内部的 state 为 idle 时（前者表明 CDB 虽然 ALU 虽忙但当前结果已被 CDB 相应，下一个 clk 同样 available，后者表明就是空闲）。注意由于 require 在运算结束前的一个状态就已经置为 1，因此在保留站刚送 OutEn 信号和数据过来时，加、与、或操作运算完成，同时 require 信号置 1，向 CDB 发请求。也就是说，加、与、或操作的执行阶段只需要 0 个周期就可以完成，而减法分了两步，先取反加一，下一个 clk 再相加，因此需要 1 个周期。

阵列乘法器：

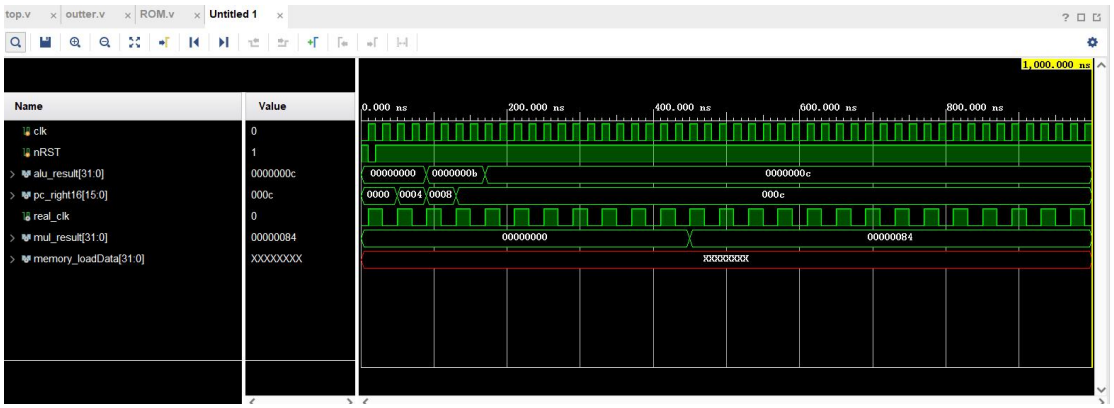
在 ALU 的设计中，乘法器的设计利用阵列乘法器加速定点数乘法。采用 32 + 16 + ... + 1 = 63 个简易的加法电路，按 5 层的方式排布成阵列，并行地计算乘法。将乘法的运行时间缩短至 5 个 CPU 时钟。



6. 各模块具体实现代码

代码见附件。

7. 仿真测试



乘法测试结果：0x0b * 0x0c = 0x84。可以看到，3 条指令顺序流出，不会因为乘法指令用到前两条的结果而停顿，第二条指令结果写回后 6 个周期之后（乘法需要 5 个周期），乘法结果产生。

更多的测试文件见附件。

8. 实验总结

8.1 项目亮点

项目完成了基于 Tomasolu 算法的支持乱序执行的 CPU 的实现。传统 Tomasulo 教材资料只给出了算法的软件模拟实现或伪代码实现，网络上关于该算法的 verilog 实现也较少。本文创新性地根据该算法的理论，从模块设计，时序设计等不同角度，独立进行分析、设计该算法硬件层面的实现。

I. 体系结构特点

该 CPU 整体体系结构特点为：框架流水线，局部并行化，部件多周期。

流水线 总框架大致可以分指令发射，执行，广播等三个阶段。各个阶段流水执行。即每个时钟周期（几乎）保证有一条指令被执行，一份数据被广播。

并行化 多个 ALU 并行地执行数据。一旦指令的操作数准备完毕，即可从保留站发射到 ALU 处。各个 ALU 的运算独立进行，互不干涉。

多周期 部件多周期更符合实际情况，本设计中各个执行单元都有 state 部件用于控制状态。所有执行和存储器件都在各个周期内分步骤完成。

II. 硬件实现队列

利用硬件实现队列（ReservationStaion + Queue）。注意到并解决了所有的所有的难点。包括

计算空余位置号 利用组合电路正确计算队列中的空余位置号；

分配保留站号 每次新指令进队时，正确地分配唯一的保留站号；

处理广播冲突 当进队的指令中的保留站号恰好为正在广播的保留站号时，队列能正确地将广播中的数据替换指令的数据，再写进队列里；

正确判断“伪满” 若队列已满，但下一个周期到来时队列能发射一条指令，则队列实质上仍可以接受指令，并没有处于真正满的状态。本设计能正确识别“伪满”现象，最大限度保证指令流动。

III. 通过边界条件测试

通过了所有的边界条件测试样例。在算法的实际设计中，受到硬件时序的约束，会产生极其多的边界条件。 例如

指令队列流出 指令队列需要判断当前指令所在的保留站是否满。当满时，指令无法流出；

广播与写入 当前广播的信号，恰好对应着当前写入信号的保留站号。

此时器件应能正确捕获广播，避免遗漏；

执行单元的状态转换　当执行单元（例如 ALU）将运算执行完毕时，它需要考虑以下几种状况：CDB 总线是否忙碌，保留站是否仍发来请求。ALU 的附属器件 state 模块需要对其进行分析，判断其接下来进入的状态；

CDB 繁忙　CDB 是所有“写”操作的唯一总线。当多个器件同时企图写总线时，将会引发冲突，此时需要一个优先译码器决定哪个执行器件的输出可以被广播。被拒绝广播的器件必须阻塞等待，直到 CDB 总线接受广播。

IV. 代码风格良好

本项目有着良好的代码风格，以确保代码的可读性和可维护性，以备之后项目的进一步发展。

采用宏定义增强可读性　将所有常量写入头文件中，便于管理。所有常数都用宏定义代替，增强可读性；

generate 语法　当大量产生相同器件，例如阵列乘法器，或进行相同的连线时，采用 verilog 2001 标准中加入的 generate 语法，以达到效率、准确地描述硬件的效果。

8.2 项目不足

我们对于该算法的实现中，由于时间的限制，并不能将 CPU 的性能做到最佳，以下是我们发现了该 CPU 可能存在的一些不足，并思考了相应的解决方案。

• ALU 数量不足，并行化仍有待提高，在有多条加法语句同时需要执行的时候还可以提高并行化（而且这样的情形存在的概率不低）。因此，我们可以设置多个加法器，多个乘法器，当有多条相同运算的指令同时流入到保留站中时，并且有多条相同运算的指令同时就绪，这些指令就可以在一个时钟周期内并行完成，进而降低了 CPU 的停顿时间，提高了效率。

• 保留站数量的设置并没有达到最佳。保留站的设置数量，关系到能够流入的指令的数量，进而影响到 CPU 能够并行执行的指令数量，CPU 的效率也会受到影响。不难发现，保留站设置得越大，每个周期内保留站内有指令就绪的概率就越高，此时 CPU 内工作的 ALU 数量就越大，并行化程度就越高。但从另一方面来看，部件的增加，带来的是硬件的复杂度以及功耗的增加（作为 CPU 的设计，不仅要考虑仿真时的速度，更重要的是在硬件上运行时的复杂度及功耗），这两者之间的权衡关系，目前以我们的能力还很难把握。

• CDB 公共数据总线的设计仍可优化。目前的设计是每一个时钟周期 CDB 接受一个 ALU 的计算结果并广播出去，这样的设计并非有问题，但效率的降低，源于我们的 CDB 没有设置结果缓存的区域。试想一下这样的情况，多个 ALU 同时完成运算，但是此时只能够一个一个广播出去，此时等候的 ALU 就会处于停顿状态，增加了 CPU 的停顿时间。要解决这个问题，我们可以在 CDB 中设置一个硬件队列，这样一来，ALU 一旦计算完毕，直接将结果放进 CDB 中的队列中就自动进行下一次运算，不会再由于无法广播而增加 CPU 的停顿时间。

• 指令数量有限。仅实现 15 条指令且几乎全部都为 R 类和 I 类指令，没有实现任何跳转和分支指令。由于跳转和分支需要清空保留站和队列，更关键的是 Tomasulo 算法不像前瞻执行一样顺序写回，而是乱序执行，无法保证写回的顺序，可能一条分支指令想要跳转时，该分支指令后面的指令已经流出执行并写回。因此分支指令实现难度较大，我们没有实现。

8.3 心得体会

一步一步，从无到右，从上学期的单周期 CPU，跨过普通多周期 CPU，直接实现了基于 Tomasulo 算法的流水线 CPU，本次课设综合了计算机组成原理以及计算机系统结构和 verilog 等一系列知识，整个过程下来的收获颇多。

由于期末精力有限，报告撰写多有不足，例如仿真测试部分无法进行更多详细的描述，还望批阅老师谅解。