2009 Altera亚洲创新设计大赛

基于FPGA的MIPS32流水线处理器的设计与实现

Design and Implementation of the MIPS32 Pipeline Processor Based on FPGA

参赛院校：南京大学

参赛学生：许　佳

薛 双 百

许 信 辉

指导教师：张 泽 生

目录

一 设计概述

1.1 设计背景

MIPS（Microprocessor without interlocked piped stages，无内部互锁流水级的微处理器）是一种最早的，最成功的RISC（Reduced Instruction Set Computer，精简指令集计算机）结构的处理器之一，其机制是尽量利用软件办法避免流水线中的数据相关问题。和英特尔采用的复杂指令系统计算结构（CISC）相比，RISC具有设计更简单、设计周期更短等优点，并可以应用更多先进的技术，开发更快的下一代处理器。相对的简洁对于MIPS来说是一种商业需要，MIPS是出现最早的商业RISC架构之一。该架构得到了工业领域内最大范围的具有影响力的制造商们的支持：从生产专用集成电路核心（ASIC Cores）的厂家（LSI Logic，Toshiba，Philips，NEC)到生产低成本CPU的厂家（NEC，Toshiba，IDT），从低端64位处理器生产厂家（IDT，NKK，NEC）到高端64位处理器生产厂家（NEC，Toshiba，IDT）。

MIPS公司设计RISC处理器始于二十世纪八十年代初，1986年推出R2000处理器，1988年推R3000处理器，1991年推出第一款64位商用微处器R4000。之后又陆续推出R8000（于1994年）、R10000（于1996年）和R12000（于1997年）等型号。随后，MIPS公司的战略发生变化，把重点放在嵌入式系统。1999年，MIPS公司发布MIPS32和MIPS64架构标准，为未来MIPS处理器的开发奠定了基础。新的架构集成了所有原来MIPS指令集，并且增加了许多更强大的功能。MIPS公司陆续开发了高性能、低功耗的32位处理器内核（core）MIPS324Kc与高性能64位处理器内核MIPS64 5Kc。2000年，MIPS公司发布了针对MIPS32 4Kc的版本以及64位MIPS 64 20Kc处理器内核。

MIPS32 4KcTM 处理器是采用MIPS技术特定为片上系统(System-On-a-Chip)而设计的高性能、低电压 32位MIPS RISC 内核。采用MIPS32TM体系结构，并且具有R4000存储器管理单元(MMU)以及扩展的优先级模式，使得这个处理器与目前嵌入式领域广泛应用的R3000和R4000系列(32位)微处理器完全兼容。新的 64 位 MIPS 处理器是RM9000x2，从“x2”这个标记判断，它包含了不是一个而是两个均具有集成二级高速缓存的64位处理器。RM9000x2 主要针对网络基础设施市场，具有集成的 DDR 内存控制器和超高速的 Hyper Transport I/O 链接。处理器、内存和 I/O均通过分组交叉连接起来的，可实现高性能、全面高速缓存的统一芯片系统。除通过并行处理提高系统性能外，RM9000x2 还通过将超标量与超流水线技术相结合来提高单个处理器的性能。

MIPS系列处理器已授权相当多厂商生产，此系列产品分布相当广泛，高阶有64bit

等级用在伺服器的产品线，低阶的也有32bit等级应用在嵌入式的场合，此系列的处理器擅长使用快取及提升汇流排速率等技术来提升整体效能，Pocket PC多使用授权NEC生产的VR4100系列。可以从任何地方，如Sony， Nintendo的游戏机，Cisco的路由器和SGI超级计算机，看见MIPS产品在销售。目前随着RISC体系结构遭到x86芯片的竞争，MIPS有可能是起初RISC CPU设计中唯一的一个在本世纪盈利的。和英特尔相比，MIPS的授权费用比较低，也就为除英特尔外的大多数芯片厂商所采用。

MIPS的系统结构及设计理念比较先进，其指令系统经过通用处理器指令体系MIPS I、MIPS II、MIPS III、MIPS IV到MIPS V，嵌入式指令体系MIPS16、MIPS32到MIPS64的发展已经十分成熟。在设计理念上MIPS强调软硬件协同提高性能，同时简化硬件设计。MIPS作为RISC体系结构中最优雅的一种，虽然自身的优雅设计并不能保证在充满竞争的市场上长盛不衰，但是MIPS微处理器却经常能在处理器的每个技术发展阶段保持速度最快的同时保持设计的简洁，使其具有强劲的市场竞争力。

1.2 作品介绍

本设计选择和MIPS指令集兼容，能实现其中的21条算术运算指令，13条跳转指令，14条存取指令，8条逻辑运算指令，6条移位指令，6条数据移动指令，以及14条自陷指令，本设计致力于实现所有的定点运算指令，并扩展实现浮点运算指令。

本设计采用五级流水线架构：IF、ID、EXE、MEM和WB。EXE阶段用到的运算部件包括超前进位加法器、桶形移位器、基于基4Booth编码和Wallace树压缩的乘加器以及基于SRT的除法器。同时本设计也将给出流水线中结构、数据和控制相关性的解决方案。

本设计还采用高速缓存（cache）以及用于控制中断异常和管理cache的协处理器，兼容avalon总线。本设计的扩展功能包括超标量流水线（多发射）、分支预测器和浮点运算单元。

1.3 适用范围

本设计适用于 MIPS 系列处理器的众多应用领域：多媒体、数字电视、网络通信、家用电子和汽车电子。此外，本设计亦可用于高校中的计算机组成原理教学与实验，这也是本作品的来源。在将本作品完善后，本设计（或部分）可用于产业界 CPU 的设计与生产。

1.4 器件选择

本设计采用 DE2-70 Development and Education Board，配备了数量高达 70,000

个逻辑单元的 Altera Cyclone II 2C70，包含了 2-Mbyte SSRAM、两个 32-Mbyte SDRAM 和 8-Mbyte 闪存等大容量内存组件。同时 DE2-70 还集成了 SD 卡界面、带有 A 类和 B 类 USB 接口的 USB 主从控制器、10/100 Ethernet Controller with a connector、RS-232 transceiver and 9-pin connector 和 PS/2 mouse/keyboard connector 等接口，完全承袭了 Altera DE2 多媒体平台丰富的多媒体、储存及网络等应用接口的优点。

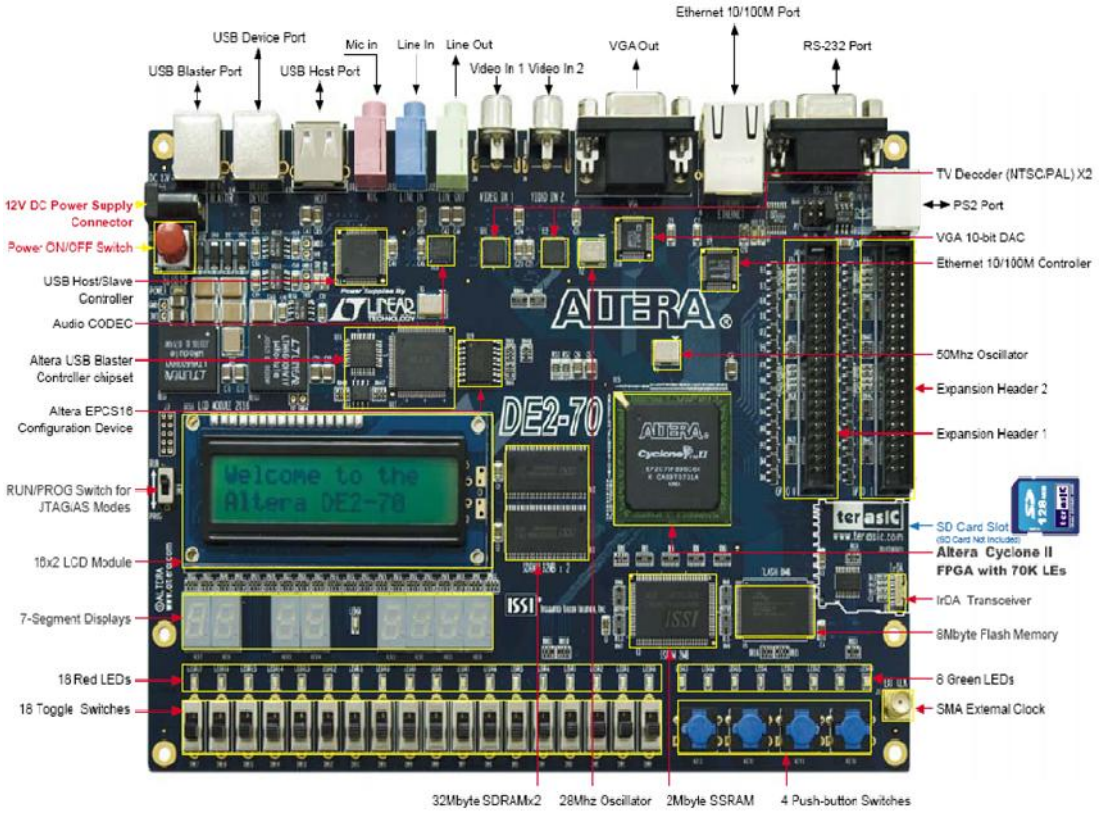

图 1　DE2-70 Development and Education Board

二 功能描述

2.1 指令集

本指令集与 The MIPS32® Instruction Set Revision 2.62 兼容，未标记星号的为基本指令，这部分指令目前的设计中已经考虑到，将会予以实现；标记星号的为扩展指令，在今后的设计中酌情添加，我们的目标是实现完整的 MIPS32 指令集。

表 1 算术运算指令

add	加法（带溢出位）	addi	立即数加法（带溢出位）
addiu	立即数加法（不带溢出位）	addu	加法（不带溢出位）
clo	计算前导一	clz	计算前导零
div	除法（有符号）	divu	除法（无符号）

madd	乘加（有符号）	maddu	乘加（无符号）
msub	乘减（有符号）	msubu	乘减（无符号）
mul	乘法（结果写到通用寄存器）	mult	乘法（有符号）
multu	乘法（有符号）	slt	小于置一（有符号）
slti	立即数小于置一（有符号）	sltiu	立即数小于置一（无符号）
sltu	小于置一（无符号）	sub	减法（有符号）
subu	减法（无符号）		
*seb	符号扩展字节	*seb	符号扩展半字

表 2 分支跳转指令

bal	转移并链接	beq	相等转移
bgez	大于等于零转移	bgezal	大于等于零转移并链接
bgtz	大于零转移	blez	小于等于零转移
bltz	小于零转移	bltzal	小于等于零转移并链接
bne	不相等转移	j	无条件跳转
jal	无条件跳转并链接	jalr	无条件跳转并链接寄存器
jr	.寄存器跳转		
*B	无条件转移	*jalrhb	无条件跳转并链接到冒险阻塞寄存器
*jr.hb	冒险阻塞寄存器跳转		

表 3 存取控制指令

lb	取字节（有符号）	lbu	取字节（无符号）
lh	取半字（有符号）	lhu	取半字（无符号）
ll	取链接字	lw	取字
lwl	取左半字	lwr	取右半字
sb	保存字节	sc	保存条件字
sh	保存半字	sw	保存字
swl	保存左半字	swr	保存右半字
*pref	预取	*sync	同步访存
*synci	同步缓存		

表 4 逻辑运算指令

and	与	andi	立即数与
lui	取立即数的高 16 位	nor	或非

or	或	ori	立即数或
xor	异或	xori	立即数异或

表 5 数据移动指令

mfhi	从高位移	mflo	从低位移
movn	非零条件移动	movz	零条件移动
mthi	移到高位寄存器	mtlo	移到低位寄存器
*movf	浮点假条件移动	*movt	浮点真条件移动
*rdhwr	读硬件寄存器		

表 6 移位指令

sll	逻辑左移	sllv	逻辑左移变量
sra	算术右移	srav	算术右移变量
srl	逻辑右移	srlv	逻辑右移变量
*rotr	循环右移	*rotrv	循环右移变量

表 7 指令控制指令

nop	空指令		
*ehb	执行冒险阻塞	*pause	等待 LL 位来清除
*ssnop	超标量空指令		

表 8 自陷指令

break	跳出点	syscall	系统跳用
teq	相等自陷	teqi	立即数相等自陷
tge	大于等于自陷	tgei	立即数大于等于自陷
tgeiu	无符号立即数大于等于自陷	tgeu	无符号大于等于自陷
tlt	小于自陷	tlti	立即数小于自陷
tltiu	无符号立即数小于自陷	tltu	无符号小于自陷
tne	不等自陷	tnei	立即数不等自陷

2.2 处理器功能：

我们的最低目标：

- 完整的5级单流水数据通路，实现较为完整的MIPS指令集
- 异常和中断处理机制，高级缓存，虚拟内存管理机制；
- 兼容avalon总线，以便在fpga开发一些简单的应用，来验证设计的正确性。

我们的中长期目标（按计划的实现顺序排列）：

- 实现寄存器重命名机制，用于支持乱序（多）发射，开发指令级并行；
- 实现分支预测
- 运行一个开源的嵌入式操作系统

目前情况下，我们的最低目标已基本完成。如能进入复赛，我们将在暑假对该处理器进行不断完善，争取在FPGA上实现更多功能。

三 性能参数

四 设计结构

五级流水线是MIPS系列处理器最经典的流水线设计方案，它把数据通道分为上文所述的取指（IF）、译码（ID）、执行（EXE）、存储（MEM）和写回（WB）等五个流水阶段。指令在译码阶段生成所有的控制信号；4个流水段寄存器用于在指令执行的各阶段间传递必要的数据和控制信息。转发单元保证进入ALU参与运算的数据总是最“新”的。冒险检测单元在必要的时刻阻塞流水线，或者清除保存于流水段寄存器中的指令；该单元还根据流水线的状态决定下条指令的地址。

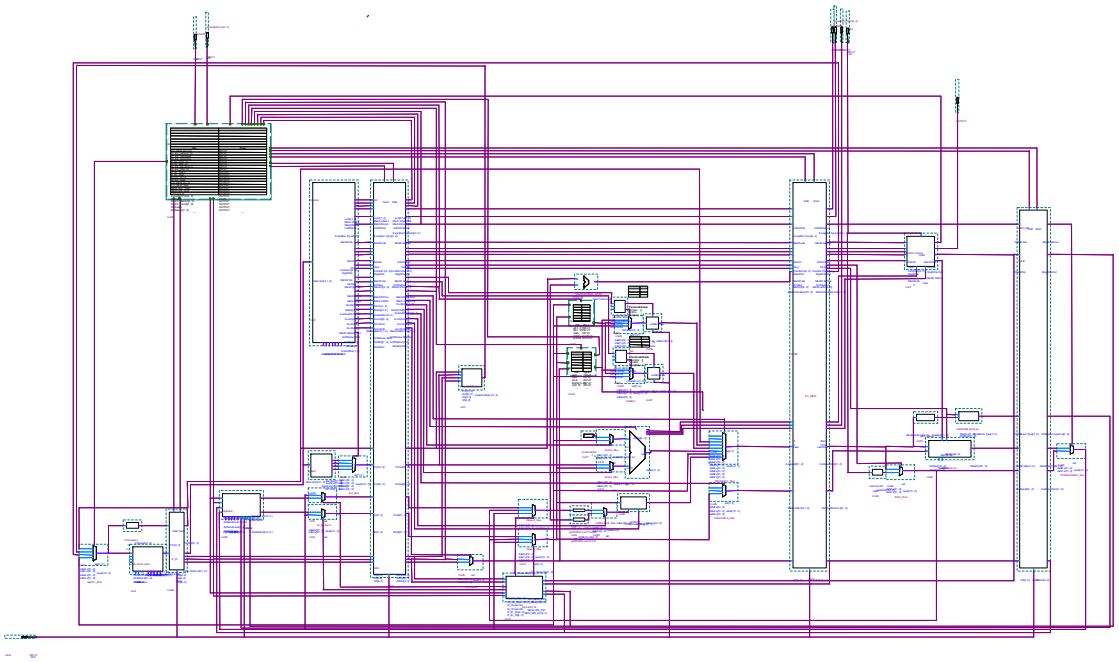

图2 MIPS32处理器数据通路图

五 设计方法

5.1 取指（IF）

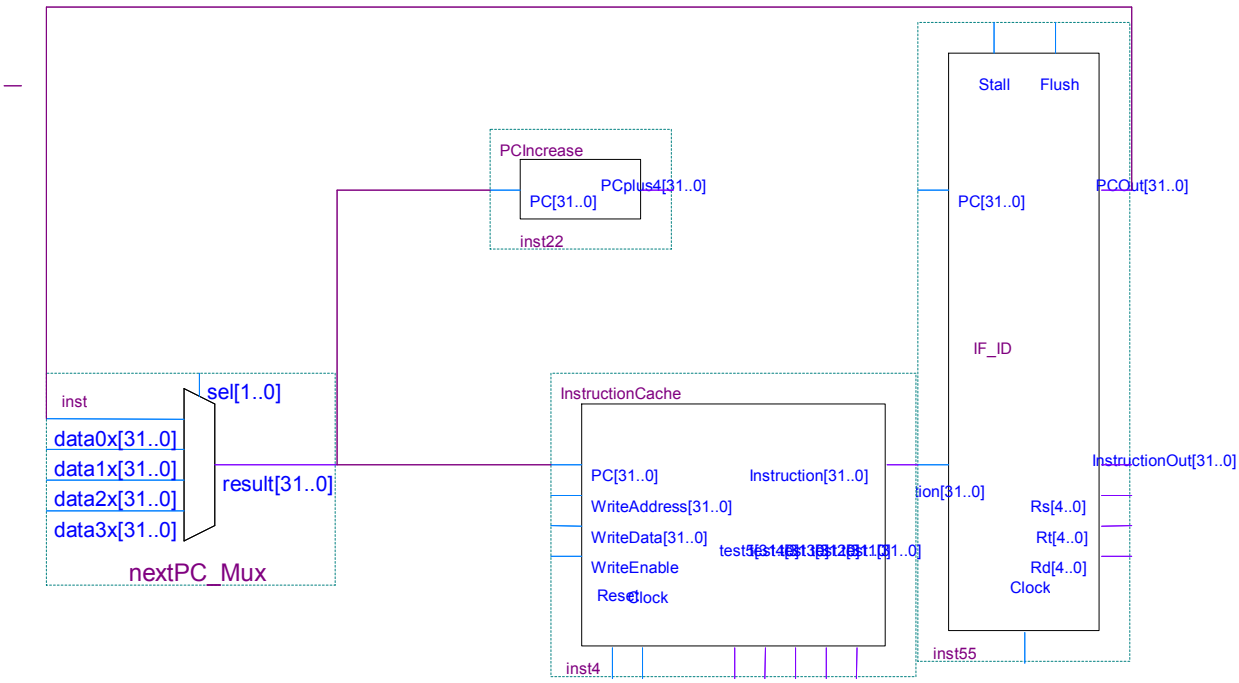

图3 取指部分数据通路

取指令阶段的主要模块及其功能如下：

- nextPC_Mux：多路选择器，选择下条指令的地址。图示为4输入，即下地址可能的来源有4个，分别是：PC+4，分支地址，跳转地址，跳转寄存器地址。加入异常处理逻辑后，输入将扩展为5个，第5个下地址为一个常量，指向异常处理程序。
- PCIncrease：加法器，计算PC+4。虽然输入输出都是32位，但实际上这32位的最低两位恒定为0，所以用一个30位加法器实现。
- InstructionCache：指令缓存。图示只是缓存的存储体，预计实现4路组相联，为此要引入存储体的标记（tag）部分。

需要特别指出的是，没有设置PC寄存器，而是从可能的下地址中选择一个地址（选择信号由冒险检测单元给出），直接送到指令Cache。这样可以消除跳转指令的分支延迟。但jump register的分支延迟依然不能避免，另外还有兼容性方面的考虑，所以实际实现中没有取消跳转指令的延迟槽。不过这种做法使分支指令的开销减少一个周期。

5.2 译码（ID）

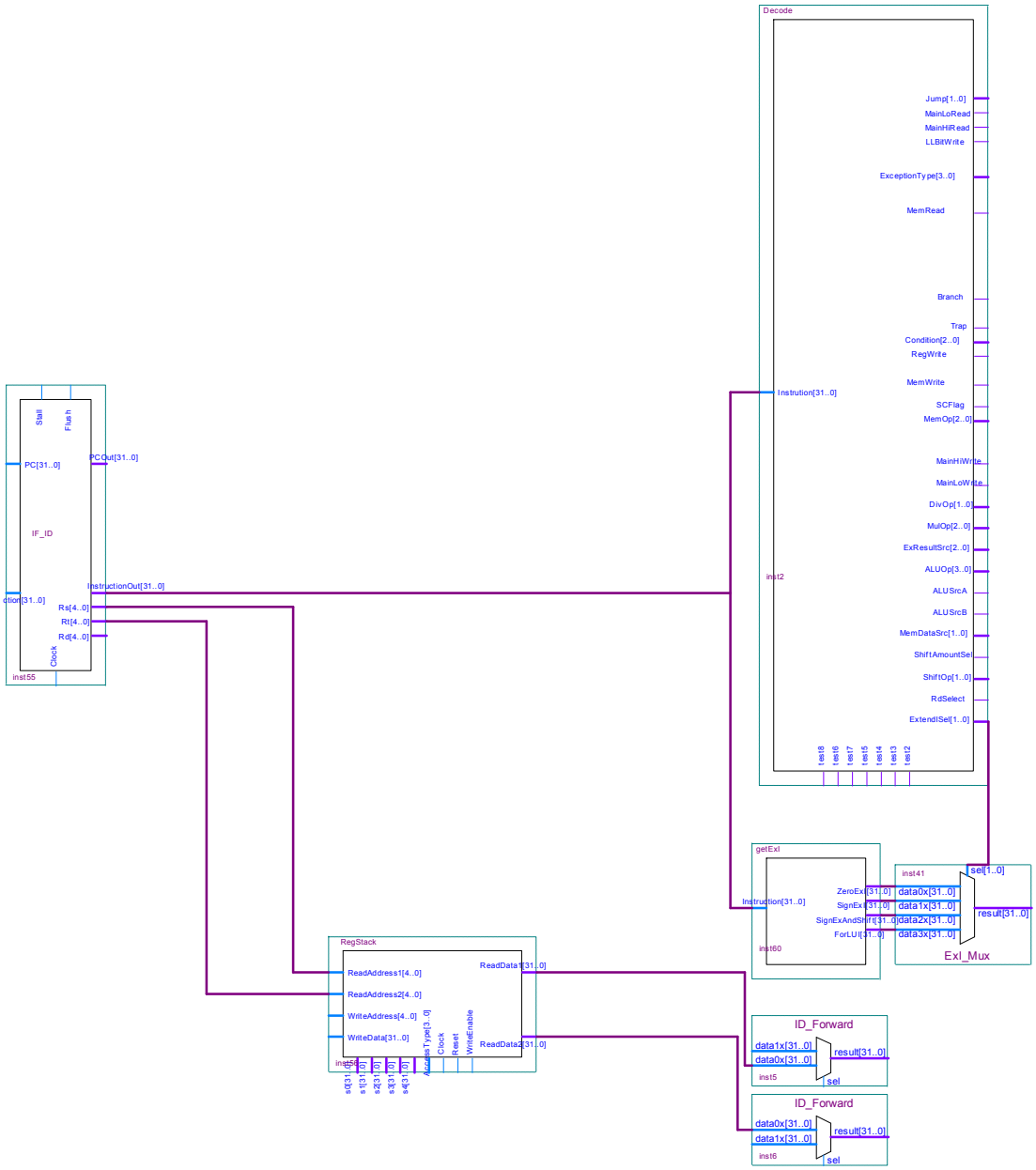

图4 译码阶段数据通路

译码阶段的主要模块及其功能如下：

- IF_ID：取指/译码流水段寄存器，在流水线上传递必要数据和控制信号。
- RegStack：32*32寄存器堆。
- Decode：控制译码器，根据指令产生控制信号。
- getExI：产生扩展的立即数，包括0扩展，符号扩展，符号扩展左移两位，扩展低16位（为了实现lui指令）。
- ExI_Mux：多路选择器，选择正确扩展的立即数。
- ID_Forward：多路选择器，用于数据转发。

5.3 执行（EXE）

在EX阶段，逻辑部件最多，耗时最长，是系统的关键路径。算术逻辑单元（ALU: arithmetic logic unit）处于EX阶段，是MIPS处理器的核心部件。ALU是CPU中完成主要的算术运算和逻辑运算的环节，它的设计直接影响到CPU的频率。本设计将给出成基本定点运算的ALU，桶形移位器、基于基4Booth编码和Wallace树压缩的乘加器以及基于SRT的除法器。

基本ALU根据控制译码器传送过来的四位ALUOp控制信号执行相应运算，输入为由二路选择器控制的两个32位操作数；输出为32位的结果，以及进/借位标志位C、小于标志位Less 、零标志位Z和溢出标志位V。主要执行的运算包括与、或、异或、或非四种逻辑运算和加法、减法（有符号），前导一、前导零、小于置一等整数运算。

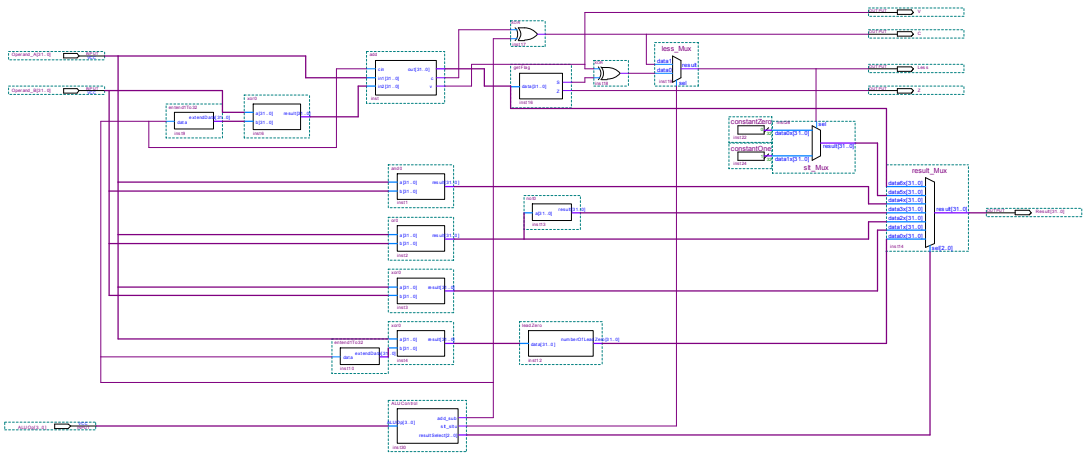

图5 基本ALU数据通路

如图5所示，ALUOp0控制加减法以及前导零（ 一） ，ALUOp1控制有无符号操作， 这样尽管用了四位ALUOp来控制ALU运算，但是可以减少额外译码。ALU控制器对四位ALUOp进行二级译码（得到三位ALUCtr，控制多路选择器的输出结果。

- 计算前导零（一），即统计寄存器中数据起始为0（1）的个数， 如果寄存器中都是0（1） ， 则结果为32。由表1可知计算前导零时，ALUOp0为0； 计算前导一时，ALUOp0为1。因此，如图1所示，首先将ALUOp0扩展为32位，再将其与操作数异或，这样就将两种运算整合到一起。然后通过32：5编码器，直接计算前导零的个数，将得到的计数高27位补零，就得到最终的结果。
- 算术运算的核心部件为超前进位加法器，其主体思想是：以尽可能短的延迟时间算出各位的进位信号，然后再按位异或。减法运算由ALUOp0控制，当ALUOp0为1时，将其扩展为32位与减数按位异或以取反，而ALUOp0作为进位信号传入加法器，这样，减数就完成了取反加一操作，然后进行加法操作就得到差。对于算术运算标志位，先对加法器得到的计算算结果进行32位按位或，然后取反

得到零标志位；加法器的溢出标志位由加法器直接计算；进/借位标志位通过将加法器计算得的进位标志位与ALUOp0异或得到。

- 逻辑运算中，与、或和异或运算，直接通过与、或和异或陈列完成；或非运算将从或阵列得到结果再通过非阵列来完成。
- 进行小于置一运算时，首先将两个操作数相减，然后再判定大小。有符号数和无符号数的判定采用不同逻辑：两个有符号数比较，Less（标志位）为V异或S的结果；两个无符号数比较，Less即为C。Less再通过二路选择器控制输出结果。

常用的移位器是用D触发器实现的，每次移一位，设定一个计数器，当计数值到零时停止移位。这种方法很是低效，速度很慢。实际上，移位过程可以看作是一个选择数据的过程，例如，左移一位的移与不移，可以看作是选择左边一位的数据和选择当前该位的数据。移2位、移4位、移8位和移16位也是同样的道理。在MIPS指令中，对32位的操作数进行移位运算只有移动1-31位有意义，移动零位即为该数据本身，而移动位数超过31位则是零。因此，我们可以用一个五位的控制信号（从高到低各位分别代表移动16位，移动8位，移动4位，移动2位，移动1位）来实现移动1-32位的操作。鉴于篇幅限制，图6只给出了8位桶形移位器的逻辑示意图。其中D0-D7为操作数，S0-S2为控制信号，Q0-Q7为移位操作后得到的结果。其中1表示左移，0表示右移；1表示算术，0表示逻辑。

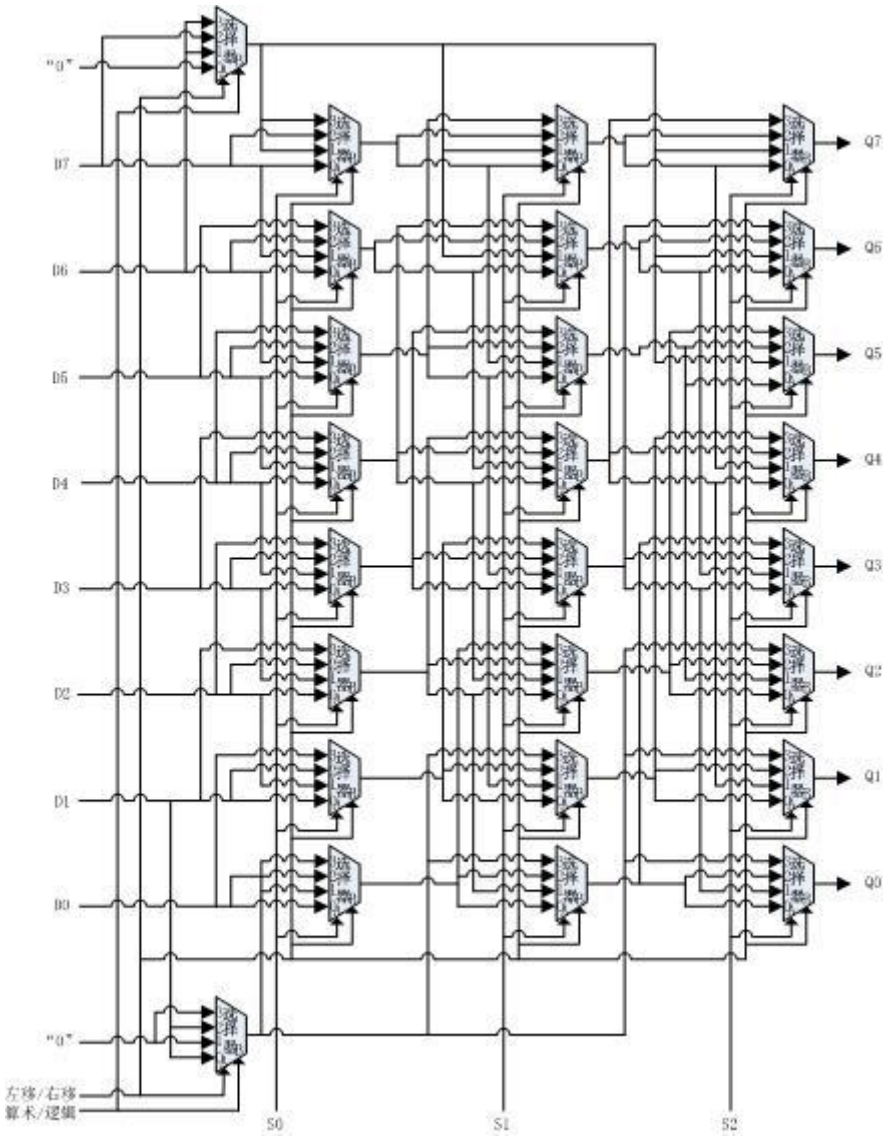

图6 桶形移位器逻辑示意图

执行阶段的主要模块及其功能如下：

- getJumpAddress：用几个字段拼接出跳转地址。
- BranchAddress_Adder：30位加法器，计算分支地址。

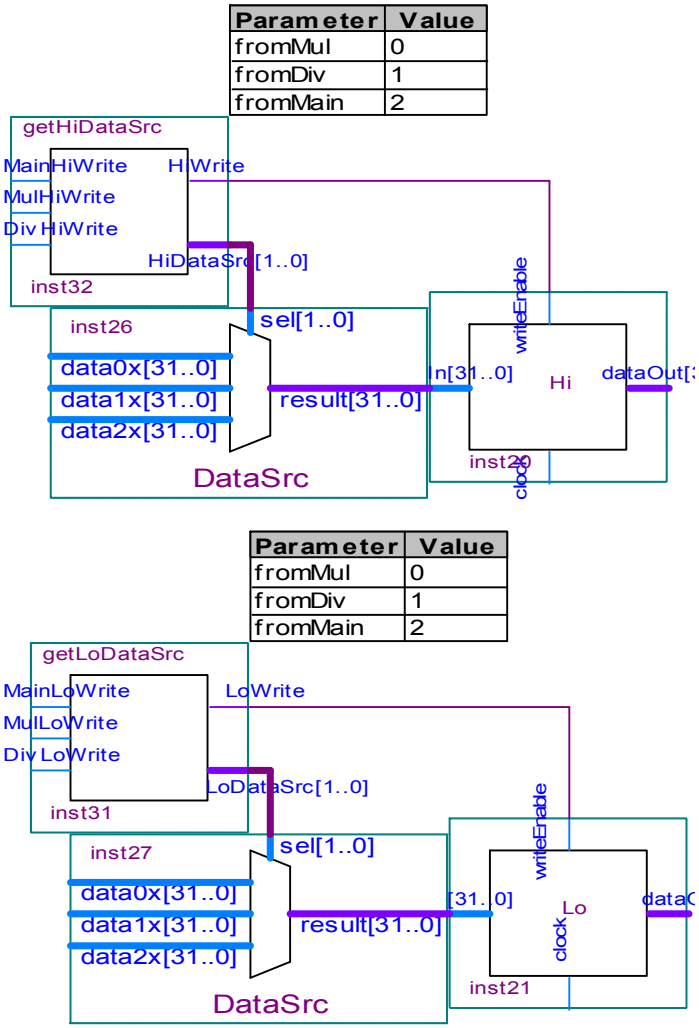

图 7 执行阶段模块图

- Lo/Hi：这两个是体系结构可见的寄存器，并为其引入了读写逻辑。由getHiDataSrc/getLoDataSrc决定写入Hi/Lo的数据的来源。可能的来源有：乘加器，除法器，主流水线（mthi/mtlo指令）。
- Div：SRT除法器。尚未加入到顶层模块中。
- Mul：乘加器。基4布斯乘法，wallace树压缩，选择进位加法。3级流水，连续的乘加操作间不用阻塞，吞吐量为一个周期一个乘加操作。为了实现这点，在乘加器的第2级末尾，已经将部分积压缩到2个。在第3级开始时，其前一条乘加指令的结果已经写入Lo和Hi，将该结果转发到第3级的开始位置，外带流水线上的两个部分积，共3个加数，作一次3-2压缩，再进行32位加法，最后是一个2选1多路选择器实现选择进位。

为了实现乘减操作，在布斯编码之前，根据乘减标志将乘数取反。对乘数最低2位作布斯编码时，要在最低位补1个0，从而凑齐3位作基4布斯编码。我们将这个0换成乘减标志，相当于实现对取反的乘数加1，自此即得乘数的补码，从而得到积的补码，或相反数。

为了支持有符号乘法和无符号乘法，首先对乘数高位扩展2位，共34位，经布斯编码，有17个部分积，外带布斯编码时取补导致的进位，共18个部分积。

缺憾在于wallace树在布局布线上并不友好，不过我们现在是做基于fpga的验证，所以暂时不考虑布局布线的问题。

附乘加器的顶层设计图如下：

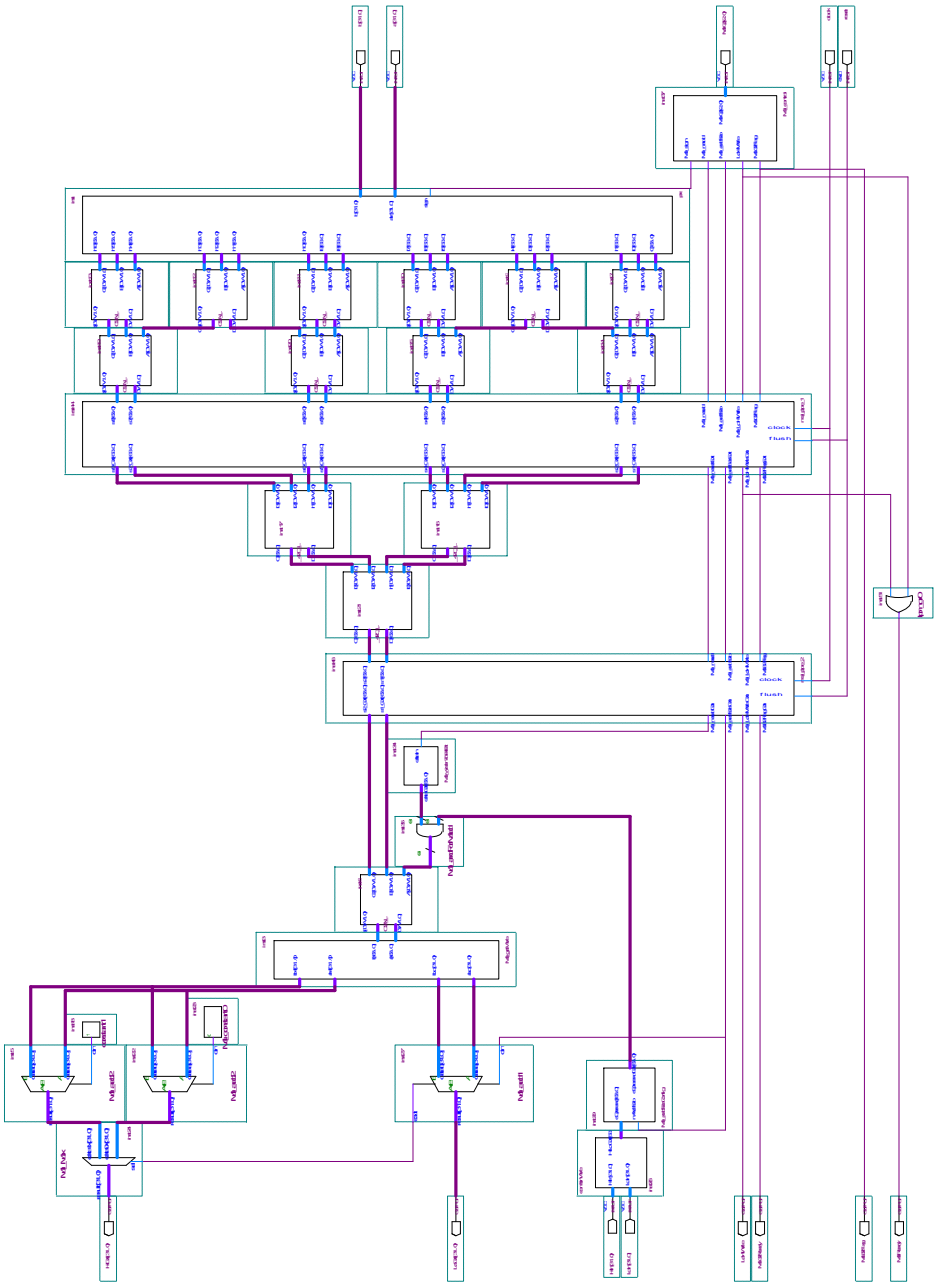

图 8 乘加器逻辑设计图

- rt_rd_Mux：选择回写的目标寄存器。选项包括rt和rd。
- fordward_Mux：多路选择器，用于转发，解决数据相关。
- shiftAmount_Mux：选择移位量来源。选项包括：指令中的shmt字段和[rs]的低5位。

- AluSrc_Mux：选择ALU操作数的来源。
- DataToMem_Mux：选择写到存储器的数据的来源。
- WBDataSrc_Mux：选择各执行部件的运算结果。

5.4 存储（MEM）

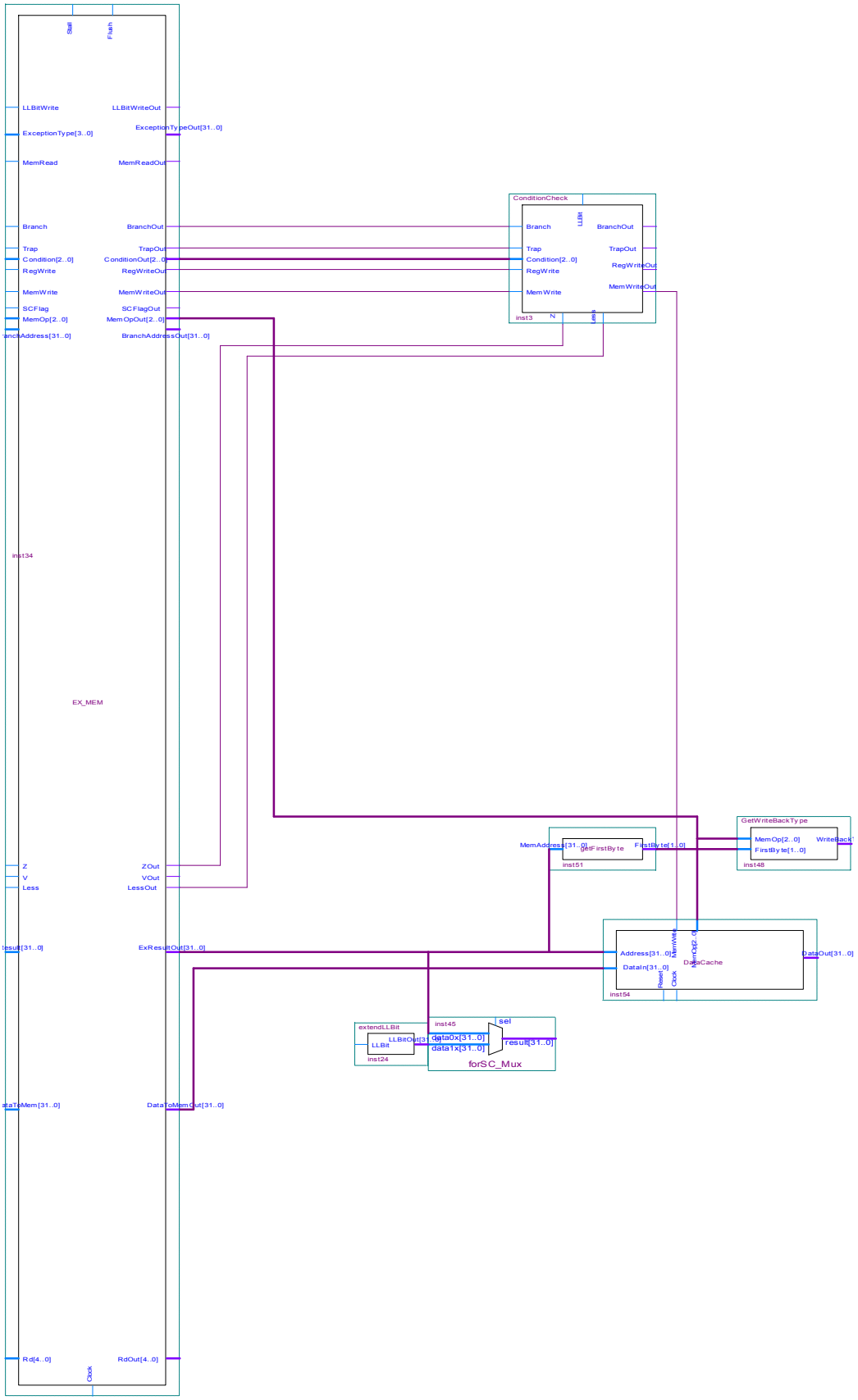
Stall
Flush
LLBitWrite
LLBitWriteOut
ExceptionType[3..0]
ExceptionTypeOut[31..0]
MemRead
MemReadOut
Branch
BranchOut
Trap
TrapOut
Condition[2..0]
ConditionOut[2..0]
RegWrite
RegWriteOut
MemWrite
MemWriteOut
SCFlag
SCFlagOut
MemOp[2..0]
MemOpOut[2..0]
BranchAddress[31..0]
BranchAddressOut[31..0]
ConditionCheck
LLBit
inst3
N
Less
inst34
EX_MEM
Z
ZOut
V
VOut
Less
LessOut
ExResultOut[31..0]
DataToMemOut[31..0]
Rd[4..0]
RdOut[4..0]
Clock
GetWriteBackType
MemOp[2..0]
WriteBack
FirstByte[1..0]
inst48
MemAddress[31..0]
getFirstByte
FirstByte[1..0]
inst51
Address[31..0]
MemWrite
MemOp[2..0]
DataIn[31..0]
DataCache
DataOut[31..0]
Reset
Clock
inst54
extendLLBit
LLBit
inst24
inst45
sel
data0x[31..0]
data1x[31..0]
result[31..0]
forSC_Mux

图 9 存储阶段数据通路

访存阶段的主要模块及其功能如下：

- EX_MEM：执行/访存流水段寄存器。
- ConditionCheck：条件检测单元。检测指令要求的条件是否得到满足。比如对于beq，检测运算结果的零标志（Z）是否有效。
- GetWriteBackType：取得回写类型。针对lwl和lwr指令，根据访存类型和访存地址最低2位决定回写通用寄存器的哪几个字节。
- forSC_Mux：针对sc指令的一个多路选择器。如果是sc指令，则回写扩展后的LLBit，否则回写执行阶段得到的结果。
- DataCache：数据缓存。同指令缓存，图示只是缓存的存储体，预计实现4路组相联，为此要引入存储体的标记（tag）部分。

5.5 写回（WB）

写回阶段的主要模块及其功能如下：

- MEM_WB：访存/写回流水段寄存器。
- WriteBackData_Mux：选择回写的数据来源。选项包括：从内存中读取的数据和执行阶段的执行结果。

5.6 转发

模块名为FordwardUnit，转发单元，用于解决流水线上的数据相关。可能的数据相关包括：第3周期（执行）和第4周期（访存），第3周期（执行）和第5周期（写回），第2周期（译码）和第5周期（写回）。第3周期不可能要求转发第4周期的访存结果，这种潜在的数据相关由冒险检测单元事先解决，所以根本不会出现。

5.7 冒险检测

可能的冒险有（按优先级从高到低排序）：

- 异常和中断：由0号协处理器发出通知。另外0号协处理器还提供异常发生的位置信息，从而让冒险检测单元发出适当的flush信号，清除流水线中位于异常指令后的所有其他指令。
- 分支：如果分支发生，清除分支后的第二条指令，控制流转向分支地址。
- 跳转：控制流转向跳转地址。

- Load-use：加载-使用冒险，处理方案是在这两条指令间插入一条空指令。
- 乘法或除法指令：如果处于执行阶段的指令是mul，即要将32位结果回写到寄存器的乘法指令，则阻塞流水线，知道该指令执行完毕。如果是其他类型的乘法指令或者除法指令，结果只是存入Hi和Lo，不会写到通用寄存器，则不用阻塞流水线，除非遇到mfhi或者mflo指令。如果某条乘法指令或除法指令尚未执行完毕，又遇到一条乘法或除法指令，则将尚未执行完毕的指令清除，只提交新的指令结果到Hi和Lo。相邻的乘法或除法指令间，如果要得到前一条指令的结果，应该在两指令间插入mfhi和mflo，这一点由编译器和汇编程序员保证。

六 设计特点

6.1 指令的兼容性和完备性

本设计完成的指令集与 The MIPS32® Instruction Set Revision 2.62 兼容，能完成超过 80 条定点运算指令。

6.2 高效的运算部件设计

本处理器的运算部件经过精心设计，采用高效的而算法。加法运算采用超前进位加法器，移位采用桶形移位器，乘法采用基于基 4Booth 编码和 Wallace 树压缩的乘加器，除法采用基于 SRT 的除法器。

6.3 先进的体系结构

在本处理器上妥善地处理了流水线中结构、数据和控制相关性，实现了异常和中断处理机制，高级缓存，虚拟内存管理，兼容avalon总线。同时本设计拟实现寄存器重命名机制，用于支持乱序（多）发射实现指令级并行；本设计还考虑分支预测。

6.4 FPGA 验证

本设计在 quartus9.0 下，采用硬件描述语言 Verilog HDL 实现电路的逻辑设计，在仿真都正确的情况下，使用 Altera 公司的 DE2-70 开发板进行了验证。并拟在 FPGA

上实现简单的应用。

七 总结

一些体会：

我们采用自顶向下开发模式，不断分解功能模块，然后逐个构建各个基本的功能模块，由代码生成 bsf 文件；对于高层的模块，为其建立 bdf 文件，在其中用低层模块的 bsf 文件拼接得到高层模块，而非书写代码实现模块的实例化调用。这种拼接的方法比书写代码更为直观，适用于复杂模块的实现。当模块足够简单时，则用代码进行行为描述更为高效。

但使用这种方法进行开发时，最让人头疼的是模块接口的变动。因为在 quartus 中，对于那些为模块生成的 bsf 文件，只能删除对外针脚（接口），不可以添加。那么如果要给某个模块添加接口，就要重新为其生成 bsf 文件。有时出于图纸布局的需要，会改变 bsf 文件的形状以及针脚的位置，一旦重新生成 bsf 文件，这项工作就要重做。所以特此建议，在 quartus 中实现为 bsf 文件添加针脚的功能，从而可以更自由地编辑 bsf 文件。

附录：

Trap

Op[31:26]	十进制	Func[5:0]	十进制	指令	Trap
默认		默认			0
000000	0	110000	48	tge	1
		110001	49	tgeu	1
		110010	50	tlt	1
		110011	51	tltu	1
		110100	52	teq	1
		110110	54	tne	1

Trap，自陷信号，CP0 检测到此信号有效，即一个自陷异常，将通知冒险检测单元清除流水线上的其余指令，转入异常处理例程。CP0 可能还有其他附加行为，如将 CP0 中相应标志寄存器置位等，尚未研究，延后处理。

Condition[2:0]

Op[31:26]	十进制	Func[5:0]	十进制	指令	含义	Condition[2:0]	十进制
默认		默认			000	000	0
000000	0	001010	10	movz	等于	001	1
		001011	11	movn	不等于	010	2
		110000	48	tge	大于等于	100	4
		110001	49	tgeu	大于等于	100	4
		110010	50	tlt	小于	101	5
		110011	51	tltu	小于	101	5
		110100	52	teq	等于	001	1
		110110	54	tne	不等于	010	2
000100	4	xxxxxx		beq	等于	001	1
000101	5			bne	不等于	010	2
000110	6			blez	小于等于	110	6
000111	7			bgtz	大于	011	3
111000	56			sc	CP0 中 LLBit 为 1	111	7

表中列出的指令，其行为受条件约束。Condition 字段指出了条件类型。

对于自陷和分支指令，自陷或分支发生与否，与 ALU 的运算结果有关。这里的结果指 ALU 运算结果的标志。

为自陷和分支特设了自陷标志 Trap 和分支标志 Branch。这两个信号以及条件字段 Condition 被送到 ConditionCheck 单元。当 ConditionCheck 单元发现 Trap 信号有效，就检测 Condition 字段，看其自陷条件是什么，并检查 ALU 运算结果的标志，看自陷条件是否满足。满足则输出一个有效的 Trap 信号，送到 CP0。否则输出一个无效的 Trap 信号（置 0）。

Branch 标志的逻辑和 Trap 标志相同。

movz 和 movn 是条件移动指令，在条件（[rt]的值是否为 0）满足时将数据从源寄存器移动到目标寄存器。Opensparc 的文档中对此的说明是，这减少了程序中 branch

指令的数目。这句话的含义还没有完全理解。我们的实现中，没有像 Branch 和 Trap 指令那样，特设 mov 标志，而是将其集成在 Condition 字段的编码中。这是因为 Condition 字段至少 3 位，编码空间为 8，实际只用掉 5 个编码（至少要 5 个，所以 Condition 字段至少 3 位），余下三个编码中，两个用来表示 movz 和 movn。当 ConditionCheck 单元在 Condition 字段发现此种条件，就去检测零标志，以决定 RegWrite（寄存器写使能）是否有效。对于其他指令，ConditionCheck 单元不改变 RegWrite 的值。

Condition 字段的最后一个编码，用于表示 sc 指令的条件，即 LLbit 是否有效。当 ConditionCheck 单元在 Condition 字段发现此种条件，就检测 CP0 中的 LLBit。若 LLBit 有效，则 ConditionCheck 单元输出一个有效的 MemWrite 信号，允许将数据写入数据 Cache。否则 ConditionCheck 单元将 MemWrite 信号置 0（无效）。当然对于其他指令，ConditionCheck 模块不改变 MemWrite 的值。

Branch

Op[31:26]	十进制	Func[5:0]	十进制	指令	Branch
默认		默认			0
000100	4	xxxxxx		beq	1
000101	5			bne	1
000110	6			blez	1
000111	7			bgtz	1

Branch，分支指令标志，用于标识当前指令为分支指令。将经由 Condition Check 单元确定分支条件是否满足，然后送往冒险检测单元，用于产生 PCSrc。详见 Condition[2:0] 字段的说明。

LLBitWrite

Op[31:26]	十进制	Func[5:0]	十进制	指令	LLBitWrite
默认		默认			0
110000	48	xxxxxx		ll	1

LLBitWrite ，为 ll 指令特设的标志。ll(Load Linked Word)的行为与 lw 类似。只是 ll 指令要将 CP0 内的 LLBit 置 1。此信号作为 LLBit 的写使能，只对 ll 指令该信号有效。

SCFlag 信号已经删除。

Op[31:26]	十进制	Func[5:0]	十进制	指令	SCFlag
默认		默认			0
111000	56	xxxxxx		sc	1

sc(Store Condition)的行为：若 CP0 中的 LLBit 为 1，则写存储器且向寄存器堆回写 1；否则不写存储器且向寄存器堆回写 0。

注：

该字段可由 Condition 字段在 MEM 阶段生成。即 Condition 为 111 时，sc 为 1，否则 sc 为 0。所以这个冗余信号在译码阶段可以省去。详见 Condition[2:0]

MemOp[2:0]

Op[31:26]	十进制	Func[5:0]	十进制	指令	MemOp[2:0]	十进制
默认		默认			000	0
100000	32			lb	011	3
100001	33			lh	001	1
100010	34			lwl	101	5
100011	35			lw	000	0
100100	36			lbu	100	4
100101	37			lhu	010	2
100110	38			lwr	110	6
101000	40			sb	011	3
101001	41			sh	001	1
101010	42			swl	101	5
101011	43			sw	000	0
101110	46			swr	110	6
110000	48			ll	000	0
111000	56			sc	000	0

MemOp，该信号指出访存的类型，它还与访存地址的低两位，共同指出回传给寄存器堆的 4 个字节中,哪些字节要回写。编码空间中各具体码值的意义,见文档 MemOp.doc。

MemWrite（高电平有效）

Op[31:26]	十进制	Func[5:0]	十进制	指令	MemWrite
默认		默认			0
101000	40			sb	1
101001	41			sh	1
101010	42			swl	1
101011	43			sw	1
101110	46			swr	1
111000	56			sc	1
111001	57			swc1	1
111010	58			swc2	1

MemWrite，内存的写使能信号，仅对 store 指令有效。

RegWrite

Op[31:26]	十进制	Func[5:0]	十进制	指令	RegWrite
默认		默认			0
000000	0	000000	0	sll	1
	0	000010	2	srl	1
	0	000011	3	sra	1
	0	000100	4	sllv	1
	0	000110	6	srlv	1
	0	000111	7	srav	1
		001001	9	jalr	1
	0	001010	10	movz	1
	0	001011	11	movn	1
	0	010000	16	mfhi	1
		010010	18	mflo	1
		100000	32	add	1
		100001	33	addu	1
		100010	34	sub	1
		100011	35	subu	1
		100100	36	and	1
		100101	37	or	1

		100110	38	xor	1
			39	nor	1
			42	slt	1
			43	sltu	1
000011	3			jal	1
001000	8		xxxxxx	addi	1
001001	9			addiu	1
001010	10			slti	1
001011	11			sltiu	1
001100	12			andi	1
001101	13			ori	1
001110	14			xori	1
001111	15			lui	1
011100	28	000010	2	mul	1
011100	28	100000	32	clz	1
011100	28	100001	33	clo	1
100000	32			lb	1
100001	33			lh	1
100010	34			lwl	1
100011	35			lw	1
100100	36			lbu	1
100101	37			lhu	1
100110	38			lwr	1
110000	48			ll	1
111000	56			sc	1

RegWrite，寄存器堆的写使能信号，也即回写信号。对于有数据回写的指令有效，包括 R-Type，I-Type，mov，load 等指令有效。

MemRead

Op[31:26]	十进制	Func[5:0]	十进制	指令	MemRead
默认		默认			0
100000	32	xxxxxx		lb	1

100001	33	xxxxxx		lh	1
100010	34	xxxxxx		lwl	1
100011	35	xxxxxx		lw	1
100100	36	xxxxxx		lbu	1
100101	37	xxxxxx		lhu	1
100110	38	xxxxxx		lwr	1
110000	48	xxxxxx		ll	1

MemRead，内存读取标志。实际上内存（这里指数据 Cache）并不设读使能信号，该信号主要用于回写数据的选择（回写的数据可能来自内存或者是执行阶段时得到的结果），还用于告知 CP0,当前指令要读取内存，让 CP0 检测 Cache 是否命中，是否有地址错误等。

Jump[1:0]

Op[31:26]	十进制	Func[5:0]	十进制	指令	说明	Jump[1:0]
默认		默认			不跳转	00
000000	0	001000	8	jr	跳转到Rs寄存器中的地址	10
	0	001001	9	jalr	跳转到Rs寄存器中的地址	10
	2			j	跳转到立即数扩展后的地址	01
	3			jal	跳转到立即数扩展后的地址	01

Jump，为跳转指令特设的标志。冒险检测单元将会检测这个标志，以决定下条指令的地址。跳转的目标可能是[rs]，也可能是指令中的 26 位立即数左移两位，拼上 PC+4 的高 4 位得到的地址。故 jump 标志 1 位不够，至少两位。

link 表示将返回地址（PC+8）写回寄存器！jal 的返回地址一定存入 31 号寄存器，但在指令中没有显示的 rd 字段，因为指令的低 26 位全部作为立即数，生成跳转地址。没有 rd 字段意味着回写的目标寄存器有 3 种选择:rs,rt 和 31 号寄存器。也就是说，

RdSelect 字段应该设成 2 位。

jalr 的返回地址存入指定的 rd 或默认的 31 号(此时指令中的 rd 字段被设为 31)。

jalr 指令格式:

jalr rs;（此时 rd 默认为 31 号）

或者：

jalr rd,rs;

MainHiWrite

Op[31:26]	十进制	Func[5:0]	十进制	指令	MainHiWrite
默认		默认			0
000000	0	010001	17	mthi	1

MainHiWrite，主流水线对寄存器 Hi 的写请求信号。

getHiDataSrc 模块接收各单元的写请求，并决定写寄存器 Hi 时，写入的数据来自哪里，包括主流水线（mthi 指令）、乘法单元和除法单元。

MainLoWrite

Op[31:26]	十进制	Func[5:0]	十进制	指令	MainLoWrite
默认		默认			0
000000	0	010011	19	mtlo	1

MainLoWrite，同 MainHiWrite。

HiRead

Op[31:26]	十进制	Func[5:0]	十进制	指令	HiRead
默认		默认			0
000000	0	010000	16	mfhi	1

HiRead，寄存器 Hi 的读请求信号，只对于 mfhi 指令有效。冒险检测单元会监听该信号。当冒险检测单元发现该信号有效时，将检查乘除法单元，判定当前是否有乘除法（不包括只保留 32 位结果的乘法指令 mul）在运行。有，则将流水线停顿，直到乘法或除法指令运行完毕，即 Hi 中的内容变得可用，确保 mfhi 指令取到的是乘除法单元最终的运算结果。

LoRead

Op[31:26]	十进制	Func[5:0]	十进制	指令	LoRead
默认		默认			0
000000	0	010010	18	mflo	1

LoRead，同 HiRead。

DivOp[1:0]

Op[31:26]	十进制	Func[5:0]	十进制	指令	说明	DivOp[1:0]
默认		默认			不是除法	00
000000	0	011010	26	div	有符号除法	01
000000	0	011011	27	divu	无符号除法	10

DivOp，除法单元的操作吗。由于要区分操作数有无符号，以及当前指令是否除法，故该信号至少 2 位。

MulOp

Op[31:26]	十进制	Func[5:0]	十进制	指令	说明	MulOp[2:0]
默认		默认			不是乘法	000
000000	0	011000	24	mult		001
000000	0	011001	25	multu		010
011100	28	000000	0	madd		011
		000001	1	maddu		100
		000010	2	mul		101
		000100	4	msub		110
		000101	5	msubu		111

MulOp，乘法单元的操作码。乘法单元可能执行的操作有：乘法（有/无符号），乘加（有/无符号），乘减（有/无符号），只保留 32 位结果且结果不写入 Lo 与 Hi 的特殊乘法指令 mul。

MemDataSrc[1:0]

Op[31:26]	十进制	MT[25:21]]	Func[5:0]	十进制	指令	说明	MemDataSrc[1:0]
默认			默认			来自 Rt	00
111001	57		xxxxxx		swc1	来自 FPU	01
	58				swc2	来自 COP2	10

MemDataSrc，写入内存的数据可能的来源有：[rt]、CP0 和浮点单元。故设此选择信号。

ExResultSrc[2:0]

Op[31:26]	十进制	Func[5:0]	十进制	指令	数据来源	ExResultSrc[2:0]	十进制
默认		默认				xxx	
000000	0	000000	0	sll	shift Unit	111	7
		000010	2	srl	shift Unit	111	7
		000011	3	sra	shift Unit	111	7
		000100	4	sllv	shift Unit	111	7
		000110	6	srlv	shift Unit	111	7
		000111	7	srav	shift Unit	111	7
		001001	9	jalr	PC+8	101	5
		001010	10	movz	Rs	100	4
		001011	11	movn	Rs	100	4
		010000	16	mfhi	Hi	010	2
		010010	18	mflo	Lo	011	3
		100000	32	add	ALU	110	6

		100001	33	addu	ALU		
		100010	34	sub	ALU		
		100011	35	subu	ALU		
		100100	36	and	ALU		
		100101	37	or	ALU		
		100110	38	xor	ALU		
		100111	39	nor	ALU		
		101010	42	slt	ALU		
		101011	43	sltu	ALU		
000011	3			jal	PC+8		
001000	8			addi	ALU		
001001	9			addiu	ALU		
001010	10			slti	ALU		
001011	11			sltiu	ALU		
001100	12			andi	ALU		
001101	13			ori	ALU		
001110	14			xori	ALU		
001111	15			lui	ALU		
011100	28	100000	32	clz	ALU		
		100001	33	clo	ALU		
100000	32			lb	ALU		
100001	33			lh	ALU		
100010	34			lwl	ALU		
100011	35			lw	ALU		
100100	36			lbu	ALU		
100101	37			lhu	ALU		
100110	38			lwr	ALU		
101000	40			sb	ALU		
101001	41			sh	ALU		
101010	42			swl	ALU		
101011	43			sw	ALU		
101110	46			swr	ALU		
	48			ll	ALU		
	49			lwc1	ALU		

	50			lwc2	ALU		
	56			sc	ALU		
	57			swc1	ALU		
	58			swc2	ALU		
				mfc0	CP0	000	0
				mfc1	浮点	001	1

ExResultSrc，执行阶段取得的结果可能的来源有：ALU，移位单元，链接地址 PC+8，Rs（条件移动指令 mov），Hi，Lo，浮点单元，CP0，共 8 个，故该信号 3 位。

ALUSrcA

Op[31:26]	十进制	Func[5:0]	十进制	指令	说明	ALUSrcA
默认		默认			[rs]	0
001111	15	xxxxxx		lui	常数 0	1

ALUSrcA ，ALU 的第一个操作数的选择信号。仅对于 lui（取立即数高半字）指令该位置 1，选常数 0 作为 ALU 的第一个操作数。对于其他指令，该位置 0，选[rs]作为 ALU 的第一个操作数。

ALUSrcB

Op[31:26]	十进制	Func[5:0]	十进制	指令	来源	ALUSrcB
默认		默认			Rt 内容	0
001000	8			addi	Immediate	1
001001	9			addiu		1
001010	10			slti		1
001011	11			sltiu		1
001100	12			andi		1
001101	13			ori		1
001110	14			xori		1
001111	15			lui		1
100000	32			lb		1
100001	33			lh		1
100010	34			lwl		1

100011	35			lw		1
100100	36			lbu		1
100101	37			lhu		1
100110	38			lwr		1
101000	40			sb		1
101001	41			sh		1
101010	42			swl		1
101011	43			sw		1
	46			swr		1
	48			ll		1
	49			lwc1		1
	50			lwc2		1
	56			sc		1
	57			swc1		1
	58			swc2		1

ALUSrcB ，ALU 的第二个操作数的选择信号。可选项有：[rt]和扩展的立即数。默认值为 0，选[rt]。对于 I-Type 指令和访存指令，该位置 1，选扩展的立即数。其中访存指令要用扩展的立即数来计算访存地址。

ALUOp[3:0]

Op[31:26]	十进制	Func[5:0]	十进制	指令	说明	ALUOp[3:0]	十进制
默认		默认				xxxx	
		001010	10	movz	符号减	0001	1
		001011	11	movn	符号减	0001	1
		100000	32	add	加	0000	0
		100001	33	addu	加	0000	0
		100010	34	sub	符号减	0001	1
		100011	35	subu	符号减	0001	1
		100100	36	and	与	0100	4
		100101	37	or	或	0110	6
		100110	38	xor	异或	1000	8

		100111	39	nor	或非	1001	9
		101010	42	slt	slt	0101	5
		101011	43	sltu	sltu	0111	7
			48	tge	符号减	0001	1
			49	tgeu	sltu	0111	7
			50	tlt	符号减	0001	1
			51	tltu	sltu	0111	7
			52	teq	符号减	0001	1
			54	tne	符号减	0001	1
000100	4	xxxxxx		beq	符号减	0001	1
000101	5			bne	符号减	0001	1
000110	6			blez	符号减	0001	1
000111	7			bgtz	符号减	0001	1
001000	8			addi	加	0000	0
001001	9			addiu	加	0000	0
001010	10			slti	slt	0101	5
001011	11			sltiu	sltu	0111	7
001100	12			andi	与	0100	4
001101	13			ori	或	0110	6
001110	14			xori	异或	1000	8
001111	15			lui	加	0000	0
011100	28	100000	32	clz	前导 0	0010	2
011100	28	100001	33	clo	前导 1	0011	3
100000	32	xxxxxx		lb	加	0000	0
100001	33			lh	加	0000	0
100010	34			lwl	加	0000	0
100011	35			lw	加	0000	0
100100	36			lbu	加	0000	0
100101	37			lhu	加	0000	0
100110	38			lwr	加	0000	0
101000	40			sb	加	0000	0
101001	41			sh	加	0000	0
101010	42			swl	加	0000	0
101011	43			sw	加	0000	0

101110	46			swr	加	0000	0
110000	48			ll	加	0000	0
110001	49			lwc1	加	0000	0
110010	50			lwc2	加	0000	0
111000	56			sc	加	0000	0
111001	57			swc1	加	0000	0
	58			swc2	加	0000	0

ALUOp，ALU 的操作码，用于确定 ALU 执行何种运算。可选的 ALU 操作减表中倒数第三列“说明”。特别指出的是，其中的 tgeu 和 tltu 两条 trap（自陷）指令，由于只关心运算结果的标志，所以不在编码中设置“无符号减”的选项，而是复用 sltu（set on less then unsigned），即可得到正确的标志。

RdSelect[1:0]

Op[31:26]	十进制	Func[5:0]	十进制	指令	说明	RdSelect[1:0]
默认		默认			默认选 Rd	00
000011	3			jal	选常量 31	10
001000	8			addi	选 Rt	01
001001	9			addiu		01
001010	10			slti		01
001011	11			sltiu		01
001100	12			andi		01
001101	13			ori		01
001110	14			xori		01
001111	15			lui		01
100000	32			lb		01
100001	33			lh		01
100010	34			lwl		01
100011	35			lw		01
100100	36			lbu		01
100101	37			lhu		01
100110	38			lwr		01
110000	48			ll		01

	56			sc		01

RdSelect，回写的目标寄存器的选择信号。可选的回写目标有：rd（默认值），rt（针对 I-Type 和访存指令），以及 31 号寄存器（针对 jal 指令，因为指令中没有显示指出回写目标，而是采用默认的 31 号寄存器）。

ShiftAmountSel

Op[31:26]	十进制	Func[5:0]	十进制	指令	移位量选择	ShiftAmountSel
默认		默认			任意	x
000000	0	000000	0	sll	shamt	0
		000010	2	srl	shamt	0
		000011	3	sra	shamt	0
		000100	4	sllv	Rs 低 5 位	1
		000110	6	srlv	Rs 低 5 位	1
		000111	7	srav	Rs 低 5 位	1

ShiftAmountSel，移位量的来源选择。针对移位指令。可选的移位量来源有：指令的 shamt 字段，或者 rs 的低 5 位。

ShiftOp[1:0]

Op[31:26]	十进制	Func[5:0]	十进制	指令	类型	ShiftOp[1:0]
默认		默认			任意	xx
000000	0	000000	0	sll	逻辑左移	
		000010	2	srl	逻辑右移	
		000011	3	sra	算术右移	
		000100	4	sllv	逻辑左移	
		000110	6	srlv	逻辑右移	
		000111	7	srav	算术右移	

ShiftOp，移位单元的操作码。可选的移位操作见表中倒数第 2 列“类型”。

ExtendI[1:0]

Op[31:26]	十进制	Func[5:0]	十进制	指令	说明	ExtendI[1:0]

默认		默认			符号扩展	00
001111	15	xxxxxx		lui	低 16 位添 16 个 0	11
001100	12			andi	0 扩展	01
001101	13			ori	0 扩展	01
001110	14			xori	0 扩展	01
000100	4			beq	符号扩展左移两位	10
000101	5			bne	符号扩展左移两位	10
000110	6			blez	符号扩展左移两位	10
000111	7			bgtz	符号扩展左移两位	10

ExtendI，立即数的扩展方式选择。默认选符号扩展，这对于 I-Type 和访存指令，移位指令（用到指令低 16 位中的 shamt 移位量字段）等都是正确的选择，故这些指令不用在表中列出。零扩展针对逻辑指令，符号扩展左移两位针对 Branch 指令（用于计算分支地址），低 16 位添 16 个 0 这种扩展方式针对 lui（取立即数高半字）指令。