32位MIPS综合实验设计文档

原子小组

January 3, 2016

目　　录

1 引言

1.1 编写目的

在此前编写的需求文档中，已经明确了此次联合实验预期达到的目标，实验 中需要完成的各部分工作，也对实验中需要用到的关键技术做了简要的原理性说明，此次实验的前期准备工作的需求文档中基本体现。

进入实际的代码开发阶段，Verilog代码的编写需要更加详细的接口，更加精准的功能说明，更加细化的流程控制。从前的需求文档已经不足以对开发过程进行具体的指导了，需要一份更加详细的设计文档。

因此，为了指导代码的实际开发过程，编写此设计文档。

文档预期读者包括开发人员、任务提出者及其他需要使用该资源的用户。

1.2 背景

本项目的系统名称为32位MIPS处理器。

本项目任务由计算机组成原理课程刘卫东老师、李山山老师和软件工程课程白晓颖老师共同提出。

承担本项目的开发者为计33班的徐炜杰、王楠和黄欢。此外还受到了刘卫东、白晓颖、李山山三位老师的指导和王钧奕、张乐两位助教的帮助。

1.3 开发工具

本项目使用Xilinx ISE和Verilog HDL进行开发，使用ModelSim进行仿真，硬件系统为清华大学计算机系32位系统开发板，软件系统为ucore操作系统。

1.4 开发流程

1. 明确项目需求，查阅理论资料，完成初步构想，书写需求文档。
2. 根据需求文档进行设计，并对设计进行反复检验、修正，书写设计文档。

3. 根据设计文档定义完成各个模块硬件代码的书写，并进行独立调试，且过程中随时对设计进行修改。
4. 将硬件各模块协同联调，过程中可能对设计进行修改。
5. 运行ucore操作系统，完成扩展功能。

1.5 参考资料

1. 实验指导文档
2. OSLab实验参考文档
3. 贾开, 周昕宇, 李铁铮, 等. 计算机组成原理综合实验报告
4. 逆光组. 清华大学计算机组成原理32位Mips CPU教程
5. 刘卫东, 李山山, 宋佳兴, 等. 计算机硬件系统实验教程[M]. 清华大学出版社, 2013.
6. 帕特森, 亨尼斯, 康继昌, 等. 计算机组成与设计: 硬件/软件接口[M]. 机械工业出版社, 2012.
7. Sweetman D. See MIPS run[M]. Morgan Kaufmann, 2010.
8. 雷思磊. 自己动手写CPU[M]. 电子工业出版社, 2014.

2 模块设计

2.1 取指阶段

取指阶段取出指令寄存器中的指令，同时，PC递增，准备取下一条指令，包括PC、IF/ID两个模块。

2.1.1 PC模块

2.1.1.1 功能描述

给出指令地址，其中实现指令指针寄存器PC，该寄存器的值就是指令地址，对应pc_reg.v文件。
PC的取值有三种情况：

(1) 一般情况下，PC等于PC+4，每个时钟周期PC加4，指向下一条指令。

(2) 当流水线暂停的时候，PC保持不变。

(3) 如果是转移指令，且满足转移条件，那么将转移目标地址赋给PC，即PC等于转移判断的结果。

2.1.1.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
clk	1	输入	时钟信号
branch_flag_i	1	输入	是否发生转移
branch_target_address_i	32	输入	转移到的目标地址
flush	1	输入	流水线清除信号
new_pc	32	输入	异常处理例程入口地址

stall	1	输入	取指地址PC是否保持不变
pc	32	输出	要读取的指令地址
ce	1	输出	指令存储器ROM使能信号

2.1.2 IF/ID模块

2.1.2.1 功能描述

实现取指与译码阶段之间的寄存器，将取指阶段的结果（取得的指令、指令地址等信息）暂时保存，在下一个时钟周期的上升沿传递到译码阶段，对应if_id.v文件。其实现只有一个时序逻辑电路。

2.1.2.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
clk	1	输入	时钟信号
if_pc	32	输入	取指阶段取出的指令对应的地址
if_inst	32	输入	取指阶段取出的指令
stall	1	输入	取指阶段是否暂停
flush	1	输入	流水线清除信号
id_pc	32	输出	译码阶段的指令对应的地址
id_inst	32	输出	译码阶段的指令

2.2 译码阶段

译码阶段将对取到的指令进行译码：给出要进行的运算类型，以及参与运算的操作数。译码阶段包括Regfile、ID和ID/EX三个模块。

2.2.1 Regfile模块

2.2.1.1 功能描述

实现了32个32位通用整数寄存器，可以同时进行两个寄存器的读操作和一个寄存器的写操作，对应regfile.v文件。

Regfile模块可以分为四段进行理解。

(1) 第一段：定义了一个二维的向量，为32个32位寄存器。

(2) 第二段：实现了写寄存器操作，当复位信号无效时（rst为RstDisable），在写使能信号we有效（we为WriteEnable），且写操作目的寄存器不等于0的情况下，可以将写输入数据保存到目的寄存器。之所以要判断目的寄存器不为0，是因为MIPS32架构规定$0的值只能为0，所以不要写入。

(3) 第三段：实现了第一个读寄存器端口，分以下几步依次判断：

(a) 当复位信号有效时，第一个读寄存器端口的输出始终为0；

(b) 当复位信号无效时，如果读取的是$0，那么直接给出0；

(c) 如果第一个读寄存器端口要读取的目标寄存器与要写入的目的寄存器是同一个寄存器，那么直接将要写入的值作为第一个读寄存器端口的输出；

(d) 上述情况都不满足，那么给出第一个读寄存器端口要读取的目标寄存器地址对应寄存器的值；

(e) 第一个读寄存器端口没有使能时，直接输出0。

(4) 第四段：实现了第二个读寄存器端口，具体过程与第三段是相似的，不再重复解释。

注意一点：读寄存器操作是组合逻辑电路，也就是一旦输入的要读取的寄存器地址raddr1或者raddr2发生变化，那么会立即给出新地址对应的寄存器的值，这样可以保证在译码阶段取得要读取的寄存器的值，而写寄存器操作是时序逻辑电路，写操作发生在时钟信号的上升沿。

2.2.1.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号，高电平有效
clk	1	输入	时钟信号
waddr	5	输入	要写入的寄存器地址
wdata	32	输入	要写入的数据
we	1	输入	写使能信号
raddr1	5	输入	第一个读寄存器端口要读取的寄存器的地址
re1	1	输入	第一个读寄存器端口读使能信号
rdata1	32	输出	第一个读寄存器端口输出的寄存器值
raddr2	5	输入	第二个读寄存器端口要读取的寄存器的地址
re2	1	输入	第二个读寄存器端口读使能信号
rdata2	32	输出	第二个读寄存器端口输出的寄存器值

2.2.2 ID模块

2.2.2.1 功能描述

对指令进行译码，译码结果包括运算类型、运算所需的源操作数、要写入的目的寄存器地址等，对应id.v文件。其中，运算类型指的是逻辑运算、移位运算、算术运算等，子类型指的是更详细的运算类型，比如运算类型为逻辑运算时，运算子类型可以是逻辑“或”运算、逻辑“与”运算、逻辑“异或”运算等。

ID模块中的电路都是组合逻辑电路，ID模块与Regfile模块之间有接口连接。其实现可以分为三段进行理解：

(1) 第一段：实现了对指令的译码，依据指令中的特征字段区分指令。以指令ori为例：只需通过识别26-31bit的指令码是否是6'b001101，即可判断是否是ori指令，其中的宏定义EXE_ORI就是6'b001101，op就是指令的26-31bit，所以当op等于EXE_ORI时，就表示是ori指令，此时会有以下译码结果。

(a) 要读取的寄存器情况：ori指令只需要读取rs寄存器的值，默认通过Regfile读端口1读取的寄存器地址reg1_addr_o的值是指令的21-25bit，正是ori指令中的rs，所以设置reg1_read_o为1，reg1_read_o连接Regfile的输入re1，reg1_addr_o连接Regfile的输入raddr1，结合对Regfile模块的介绍可知，译码阶段会读取寄存器rs的值。指令ori需要的另一个操作数是立即数，所以设置reg2_read_o为0，表示不通过Regfile读端口2读取寄存器，这里暗含使用立即数作为运算的操作数。imm就是指令中的立即数进行零扩展后的值。

(b) 要执行的运算：alusel_o给出要执行的运算类型，对于ori指令而言就是逻辑操作，即EXE_RES_LOGIC。aluop_o给出要执行的运算子类型，对于ori指令而言就是逻辑“或”运算，即EXE_OR_OP。这两个值会传递到执行阶段。

(c) 要写入的目的寄存器：wreg_o表示是否要写目的寄存器，ori指令要将计算结果保存到寄存器中，所以wreg_o设置为WriteEnable。wd_o是要写入的目的寄存器地址，此时就是指令的16-20bit，正是ori指令中的rt。这两个值也会传递到执行阶段。

(2) 第二段：给出参与运算的源操作数1的值，如果reg1_read_o为1，那么就将从Regfile模块读端口1读取的寄存器的值作为源操作数1，如果reg1_read_o为0，那么就将立即数作为源操作数1，对于ori而言，此处选择从Regfile模块读端口1读取的寄存器的值作为源操作数1。该值将通过reg1_o端口被传递到执行阶段。

(3) 第三段：给出参与运算的源操作数2的值，如果reg2_read_o为1，那么就将从Regfile模块读端口2读取的寄存器的值作为源操作数2，如果reg2_read_o为0，那么就将立即数作为源操作数2，对于ori而言，此处选择立即数imm作为源操作数2。该值将通过reg2_o端口被传递到执行阶段。

ID模块还处理了冲突相关的部分内容。

1. 实现数据前推的方法具体如下：

   (1) 将处于流水线执行阶段的指令的运算结果，包括：是否要写目的寄存器wreg_o、要写的目的寄存器地址wd_o、要写入目的寄存器的数据wdata_o等信息送到译码阶段。

   (2) 将处于流水线访存阶段的指令的运算结果，包括：是否要写目的寄存器wreg_o、要写的目的寄存器地址wd_o、要写入目的寄存器的数据wdata_o等信息送到译码阶段。

2. load相关

   (1) 即使通过数据前推的方法，将访存阶段加载得到的数据前推，也解决不了问题，因为数据加载时，下一条指令如果是beq指令，就已经处于执行阶段了，已经进行了比较判断。

   (2) OpenMIPS解决load相关的方法是，在译码阶段检查当前指令与上一条指令是否存在load相关，如果存在load相关，那么就让流水线的译码、取指阶段暂停，而执行、访存、回写阶段继续，相当于插入一个空指令，这样处于执行阶段的加载指令会继续运行，不受影响，当其运行到访存阶段时，将加载得到的数据前推到译码阶段，然后，流水线可以继续运行。

   (3) 具体地，如果存在load相关，那么通过stallreq接口通知CTRL模块请求流水线暂停。

此外，有关转移指令的实现，如果在流水线执行阶段进行转移判断，并且发生转移，那么就会有2条无效指令，导致浪费了两个时钟周期。为了减少损失，规定转移指令后面的指令位置为“延迟槽”，延迟槽中的指令被称为“延迟指令”。延迟指令总是被执行，与转移发生与否没有关系。为了避免时钟周期的浪费，我们在译码阶段进行了转移判断：

(1) 如果处于译码阶段的指令是转移指令，并且满足转移条件，那么ID模块设置转移发生标志branch_flag_o为Branch，同时通过branch_target_address_o接口给出转移目的地址，送到PC模块，后者据此修改取指地址。

(2) 如果处于译码阶段的指令是转移指令，并且满足转移条件，那么ID模块还会设置next_inst_in_delayslot_o为InDelaySlot，表示下一条指令是延迟槽指令，其中InDelaySlot是一个宏定义。next_inst_in_delayslot_o信号会送入ID/EX模块，并在下一个时钟周期通过ID/EX模块的is_in_delayslot_o接口送回到ID模块，ID模块可以据此判断当前处于译码阶段的指令是否是延迟槽指令。

(3) 如果转移指令需要保存返回地址，那么ID模块还要计算返回地址，并通过link_addr_o接口输出，该值最终会传递到EX模块，作为要写入目的寄存器的值。

2.2.2.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
pc_i	32	输入	译码阶段的指令对应的地址
inst_i	32	输入	译码阶段的指令
reg1_data_i	32	输入	从Regfile输入的第一个读寄存器端口的输入
reg2_data_i	32	输入	从Regfile输入的第二个读寄存器端口的输入
ex_wreg_i	1	输入	处于执行阶段的指令是否要写目的寄存器
ex_wd_i	5	输入	处于执行阶段的指令要写的目的寄存器地址
ex_wdata_i	32	输入	处于执行阶段的指令要写入目的寄存器的数据
mem_wreg_i	1	输入	处于访存阶段的指令是否要写目的寄存器
mem_wd_i	5	输入	处于访存阶段的指令要写的目的寄存器地址
mem_wdata_i	32	输入	处于访存阶段的指令要写入目的寄存器的数据
ex_aluop_i	8	输入	处于执行阶段指令的运算子类型
is_in_delayslot_i	1	输入	当前处于译码阶段的指令是否位于延迟槽
reg1_read_o	1	输出	Regfile模块的第一个读寄存器端口的读使能信号
reg2_read_o	1	输出	Regfile模块的第二个读寄存器端口的读使能信号
reg1_addr_o	5	输出	Regfile模块的第一个读寄存器端口的读地址信号
reg2_addr_o	5	输出	Regfile模块的第二个读寄存器端口的读地址信号
aluop_o	8	输出	译码阶段的指令要进行的运算的子类型
alusel_o	3	输出	译码阶段的指令要进行的运算的类型
reg1_o	32	输出	译码阶段的指令要进行的运算的源操作数1
reg2_o	32	输出	译码阶段的指令要进行的运算的源操作数2
wd_o	5	输出	译码阶段的指令要写入的目的寄存器地址
wreg_o	1	输出	译码阶段的指令是否有要写入的目的寄存器
branch_flag_o	1	输出	是否发生转移
branch_target_address_o	32	输出	转移到的目标地址
is_in_delayslot_o	1	输出	当前处于译码阶段的指令是否位于延迟槽
link_addr_o	32	输出	转移指令要保存的返回地址
next_inst_in_delayslot_o	1	输出	下一条进入译码阶段的指令是否位于延迟槽
inst_o	32	输出	当前处于译码阶段的指令
excepttype_o	32	输出	收集的异常信息
current_inst_addr_o	32	输出	译码阶段指令的地址
stallreq	1	输出	译码阶段请求流水暂停

2.2.3 ID/EX模块

2.2.3.1 功能描述

实现译码与执行阶段之间的寄存器，将译码阶段取得的运算类型、源操作数、要写的目的寄存器地址等结果暂时保存，在下一个时钟周期的上升沿传递到流水线的执行阶段，对应id_ex.v文件。其实现只有一个时序逻辑电路。

2.2.3.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
clk	1	输入	时钟信号
id_alusel	3	输入	译码阶段的指令要进行的运算的类型
id_aluop	8	输入	译码阶段的指令要进行的运算的子类型
id_reg1	32	输入	译码阶段的指令要进行的运算的源操作数1
id_reg2	32	输入	译码阶段的指令要进行的运算的源操作数2
id_wd	5	输入	译码阶段的指令要写入的目的寄存器地址
id_wreg	1	输入	译码阶段的指令是否有要写入的目的寄存器
stall	1	输入	译码阶段是否暂停
flush	1	输入	流水线清除信号
id_excepttype	32	输入	译码阶段收集到的异常信息
id_current_inst_addr	32	输入	译码阶段指令的地址
id_is_in_delayslot	1	输入	当前处于译码阶段的指令是否位于延迟槽
id_link_address	32	输入	处于译码阶段的转移指令要保存的返回地址
next_inst_in_delayslot_i	1	输入	下一条进入译码阶段的指令是否位于延迟槽
id_inst	32	输入	当前处于译码阶段的指令
ex_inst	32	输出	当前处于执行阶段的指令
ex_is_in_delayslot	32	输出	当前处于执行阶段的指令是否位于延迟槽
ex_link_address	1	输出	处于执行阶段的转移指令要保存的返回地址
is_in_delayslot_o	1	输出	当前处于译码阶段的指令是否位于延迟槽
ex_excepttype	32	输出	译码阶段收集到的异常信息
ex_current_inst_addr	32	输出	执行阶段指令的地址
ex_alusel	3	输出	执行阶段的指令要进行的运算的类型
ex_aluop	8	输出	执行阶段的指令要进行的运算的子类型
ex_reg1	32	输出	执行阶段的指令要进行的运算的源操作数1
ex_reg2	32	输出	执行阶段的指令要进行的运算的源操作数2
ex_wd	5	输出	执行阶段的指令要写入的目的寄存器地址

ex_wreg	1	输出	执行阶段的指令是否有要写入的目的寄存器

2.3 执行阶段

执行阶段将依据译码阶段的结果，对源操作数1、源操作数2进行指定的运算。执行阶段包括EX、EX/MEM两个模块。

2.3.1 EX模块

2.3.1.1 功能描述

依据接口关系，EX模块会从ID/EX模块得到运算类型alusel_i、运算子类型aluop_i、源操作数reg1_i、源操作数reg2_i、要写的目的寄存器地址wd_i。依据这些译码阶段的结果，进行指定的运算，给出运算结果。对应ex.v文件。

EX模块中都是组合逻辑电路，其实现可以分为两段理解：

(1) 第一段：依据输入的运算子类型进行运算，比如逻辑“或”运算。运算结果根据不同的运算类型进行保存，如逻辑操作的结果保存在logicout中，算术运算的结果保存在arithmeticres中，移位运算的结果保存在shiftres中等。

(2) 第二段：给出最终的运算结果，包括：是否要写目的寄存器wreg_o、要写的目的寄存器地址wd_o、要写入的数据wdata_o。其中wreg_o、wd_o的值都直接来自译码阶段，不需要改变，wdata_o的值要依据运算类型进行选择，如果是逻辑运算，那么将logicout的值赋给wdata_o。如果是mthi、mtlo指令，为了写HI、LO寄存器，还要给出whilo_o、hi_o、lo_o的值。

EX模块还处理了冲突相关的部分内容，具体如下。

实现数据前推的方法：将处于流水线访存阶段、回写阶段的指令对HI、LO寄存器的操作信息反馈到执行阶段，如果处于执行阶段的是mfhi、mflo指令，执行阶段的选择模块就会从中选择，确定HI、LO寄存器的正确值。

2.3.1.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
alusel_i	3	输入	执行阶段要进行的运算的类型
aluop_i	8	输入	执行阶段要进行的运算的子类型
reg1_i	32	输入	参与运算的源操作数1
reg2_i	32	输入	参与运算的源操作数2
wd_i	5	输入	指令执行要写入的目的寄存器地址
wreg_i	1	输入	是否有要写入的目的寄存器
excepttype_i	32	输入	译码阶段收集到的异常信息
current_inst_addr_i	32	输入	执行阶段指令的地址
is_in_delayslot_i	1	输入	当前处于执行阶段的指令是否位于延迟槽
link_address_i	32	输入	处于执行阶段的转移指令要保存的返回地址
hilo_temp_i	64	输入	第一个执行周期得到的乘法结果

cnt_i	2	输入	当前处于执行阶段的第几个时钟周期
hilo_temp_o	64	输出	第一个执行周期得到的乘法结果
cnt_o	2	输出	下一个时钟周期处于执行阶段的第几个时钟周期
excepttype_o	32	输出	译码阶段、执行阶段搜集到的异常信息
current_inst_addr_o	32	输出	执行阶段指令的地址
is_in_delayslot_o	1	输出	执行阶段的指令是否是延迟槽指令
wd_o	5	输出	执行阶段的指令最终要写入的目的寄存器地址
wreg_o	1	输出	执行阶段的指令最终是否有要写入的目的寄存器
wdata_o	32	输出	执行阶段的指令最终要写入目的寄存器的值
hi_i	32	输入	HILO模块给出的HI寄存器的值
lo_i	32	输入	HILO模块给出的LO寄存器的值
mem_whilo_i	1	输入	处于访存阶段的指令是否要写HI、LO寄存器
mem_hi_i	32	输入	处于访存阶段的指令要写入HI寄存器的值
mem_lo_i	32	输入	处于访存阶段的指令要写入LO寄存器的值
wb_whilo_i	1	输入	处于写回阶段的指令是否要写HI、LO寄存器
wb_hi_i	32	输入	处于写回阶段的指令要写入HI寄存器的值
wb_lo_i	32	输入	处于写回阶段的指令要写入LO寄存器的值
whilo_o	1	输出	处于执行阶段的指令是否要写HI、LO寄存器
hi_o	32	输出	处于执行阶段的指令要写入HI寄存器的值
lo_o	32	输出	处于执行阶段的指令要写入LO寄存器的值
cp0_reg_data_i	32	输入	从CP0模块读取的指定寄存器的值
mem_cp0_reg_we	1	输入	访存阶段的指令是否要写CP0中的寄存器
mem_cp0_reg_write_addr	5	输入	访存阶段的指令要写的CP0中寄存器的地址
mem_cp0_reg_data	32	输入	访存阶段的指令要写入CP0中寄存器的数据
wb_cp0_reg_we	1	输入	写回阶段的指令是否要写CP0中的寄存器
wb_cp0_reg_write_addr	5	输入	写回阶段的指令要写的CP0中寄存器的地址
wb_cp0_reg_data	32	输入	写回阶段的指令要写入CP0中寄存器的数据
cp0_reg_read_addr_o	5	输出	执行阶段的指令要读取的CP0中寄存器的地址
cp0_reg_we_o	1	输出	执行阶段的指令是否要写CP0中的寄存器
cp0_reg_write_addr_o	5	输出	执行阶段的指令要写的CP0中寄存器的地址
cp0_reg_data_o	32	输出	执行阶段的指令要写入CP0中寄存器的数据
inst_i	32	输入	当前处于执行阶段的指令
aluop_o	8	输出	执行阶段的指令要进行的运算的子类型
mem_addr_o	32	输出	加载、存储指令对应的存储器地址
reg2_o_pc	32	输出	存储指令要存储的数据，或者lwl、lwr指令

			要写入的目的寄存器的原始值
stallreq	1	输出	执行阶段是否请求流水线暂停

2.3.2 EX/MEM模块

2.3.2.1 功能描述

实现执行与访存阶段之间的寄存器，将执行阶段的结果暂时保存，在下一个时钟周期的上升沿传递到流水线的访存阶段，对应ex_mem.v文件。其实现只有一个时序逻辑电路。

2.3.2.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
clk	1	输入	时钟信号
ex_wd	5	输入	执行阶段的指令执行后要写入的目的寄存器地址
ex_wreg	1	输入	执行阶段的指令执行后是否有要写入的目的寄存器
ex_wdata	32	输入	执行阶段的指令执行后要写入的目的寄存器的值
mem_wd	5	输出	访存阶段的指令要写入的目的寄存器地址
mem_wreg	1	输出	访存阶段的指令是否有要写入的目的寄存器
mem_wdata	32	输出	访存阶段的指令要写入目的寄存器的值
stall	1	输入	执行阶段是否暂停
flush	1	输入	是否清除流水线
ex_cp0_reg_we	1	输入	执行阶段的指令是否要写CP0中的寄存器
ex_cp0_reg_write_addr	5	输入	执行阶段的指令要写的CP0中寄存器的地址
ex_cp0_reg_data	32	输入	执行阶段的指令要写入CP0中寄存器的数据
mem_cp0_reg_we	1	输出	访存阶段的指令是否要写CP0中的寄存器
mem_cp0_reg_write_addr	5	输出	访存阶段的指令要写的CP0中寄存器的地址
mem_cp0_reg_data	32	输出	访存阶段的指令要写入CP0中寄存器的数据
ex_aluop	8	输入	执行阶段的指令要进行的运算的子类型
ex_mem_addr	32	输入	执行阶段的加载、存储指令对应的存储器地址
ex_reg2	32	输入	执行阶段的存储指令要存储的数据，或者lwl、lwr指令要写入的目的寄存器的原始值
mem_aluop	8	输出	访存阶段的指令要进行的运算的子类型
mem_mem_addr	32	输出	访存阶段的加载、存储指令对应的存储器地址
mem_reg2	32	输出	访存阶段的存储指令要存储的数据，或者lwl、lwr指令要写入的目的寄存器的原始值
ex_whilo	1	输入	执行阶段的指令是否要写HI、LO寄存器

ex_hi	32	输入	执行阶段的指令要写入HI寄存器的值
ex_lo	32	输入	执行阶段的指令要写入LO寄存器的值
mem_whilo	1	输出	访存阶段的指令是否要写HI、LO寄存器
mem_hi	32	输出	访存阶段的指令要写入HI寄存器的值
mem_lo	32	输出	访存阶段的指令要写入LO寄存器的值
ex_excepttype	32	输入	译码、执行阶段收集到的异常信息
ex_current_inst_address	32	输入	执行阶段指令的地址
ex_is_in_delayslot	1	输入	执行阶段的指令是否是延迟槽指令
mem_excepttype	32	输出	译码、执行阶段收集到的异常信息
mem_current_inst_address	32	输出	访存阶段指令的地址
mem_is_in_delayslot	1	输出	访存阶段的指令是否是延迟槽指令
hilo_i	64	输入	保存的乘法结果
cnt_i	2	输入	下一个时钟周期是执行阶段的第几个时钟周期
hilo_o	64	输出	保存的乘法结果
cnt_o	2	输出	当前处于执行阶段的第几个时钟周期

2.4 访存阶段

流水线的访存阶段包括MEM、MEM/WB两个模块。

2.4.1 MEM模块

2.4.1.1 功能描述

如果是加载、存储指令，那么会对数据存储器进行访问。此外，还会在该模块进行异常判断。对应mem.v文件。其实现为组合逻辑电路。

1. 加载指令实现思路

   加载指令在译码阶段进行译码，得到运算类型alusel_o、aluop_o，以及要写的目的寄存器信息。这些信息传递到执行阶段，然后又传递到访存阶段，访存阶段依据这些信息，设置对数据存储器RAM的访问信号。从RAM读取回来的数据需要按照加载指令的类型、加载地址进行对齐调整，调整后的结果作为最终要写入目的寄存器的数据。

2. 存储指令实现思路

   存储指令在译码阶段进行译码，得到运算类型alusel_o、aluop_o，以及要存储的数据。这些信息传递到执行阶段，然后又传递到访存阶段，访存阶段依据这些信息，设置对数据存储器RAM的访问信号，将数据写入RAM。

OpenMIPS处理器会在访存阶段的MEM模块综合所有的异常信息、CP0寄存器的值，最终判断是否有要处理的异常。可以分成三段理解：

(1) 得到CP0中寄存器的最新值

从CP0模块传入的Status、EPC、Cause等寄存器的值并不一定是最新值，因为处于回写阶段的指令可能要写这些寄存器。这又是一种数据相关的情况，解决方法是数据前推。具体地，将回写阶段要写的CP0寄存器的信息前推到访存阶段，在访存阶段进行判断，从而得到最新值。

(2) 给出最终的异常类型

依据CP0中寄存器的值，以及译码、执行阶段收集到的异常类型，得到最终的异常类型。

首先判断当前处于访存阶段指令的地址是否为0:

如果为0，那么表示处理器处于复位状态，或者刚刚发生异常，正在清除流水线（flush为1），或者流水线处于暂停状态，在以上三种情况下都不处理异常。

如果当前访存阶段指令的地址不为0，那么可以进一步判断有没有异常、是何种异常，从而给输出变量excepttype_o赋值。

(3) 给出对数据存储器的写操作

OpenMIPS处理器要实现精确异常，也就是发生异常时，引起异常的指令及其后面已经进入流水线的指令都会失效。如果引起异常的指令是存储指令，那么要使其失效，就要停止修改数据存储器，所以在这里修改mem_we_o的赋值，如果发生异常（即变量excepttype_o不为0），那么设置mem_we_o为0，从而不会修改数据存储器。

2.4.1.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
wd_i	5	输入	访存阶段的指令要写入的目的寄存器地址
wreg_i	1	输入	访存阶段的指令是否有要写入的目的寄存器
wdata_i	32	输入	访存阶段的指令要写入目的寄存器的值
wd_o	5	输出	访存阶段的指令最终要写入的目的寄存器地址
wreg_o	1	输出	访存阶段的指令最终是否有要写入的目的寄存器
wdata_o	32	输出	访存阶段的指令最终要写入目的寄存器的值
aluop_i	8	输入	访存阶段的指令要进行的运算的子类型
mem_addr_i	32	输入	访存阶段的加载、存储指令对应的存储器地址
reg2_i	32	输入	访存阶段的存储指令要存储的数据，或者lwl、lwr指令要写入的目的寄存器的原始值
mem_data_i	32	输入	从数据存储器读取的数据
mem_addr_o	32	输出	要访问的数据存储器的地址
mem_we_o	1	输出	是否是写操作，为1表示是写操作
mem_sel_o	4	输出	字节选择信号
mem_data_o	32	输出	要写入数据存储器的数据
mem_ce_o	1	输出	数据存储器使能信号
whilo_i	1	输入	访存阶段的指令是否要写HI、LO寄存器
hi_i	32	输入	访存阶段的指令要写入HI寄存器的值
lo_i	32	输入	访存阶段的指令要写入LO寄存器的值
whilo_o	1	输出	访存阶段的指令最终是否要写HI、LO寄存器
hi_o	32	输出	访存阶段的指令最终要写入HI寄存器的值

lo_o	32	输出	访存阶段的指令最终要写入LO寄存器的值
cp0_reg_we_i	1	输入	访存阶段的指令是否要写CP0中的寄存器
cp0_reg_write_addr_i	5	输入	访存阶段的指令要写的CP0中寄存器的地址
cp0_reg_data_i	32	输入	访存阶段的指令要写入CP0中寄存器的数据
cp0_reg_we_o	1	输出	访存阶段的指令最终是否要写CP0中的寄存器
cp0_reg_write_addr_o	5	输出	访存阶段的指令最终要写的CP0中寄存器的地址
cp0_reg_data_o	32	输出	访存阶段的指令最终要写入CP0中寄存器的数据
excepttype_i	32	输入	译码、执行阶段收集到的异常信息
current_inst_address	32	输入	访存阶段指令的地址
is_in_delayslot_i	1	输入	访存阶段的指令是否是延迟槽指令
cp0_status_i	32	输入	CP0中Status寄存器的值
cp0_cause_i	32	输入	CP0中Cause寄存器的值
cp0_epc_i	32	输入	CP0中EPC寄存器的值
wb_cp0_reg_we	1	输入	回写阶段的指令是否要写CP0中的寄存器
wb_cp0_reg_write_address	5	输入	回写阶段的指令要写的CP0中寄存器的地址
wb_cp0_reg_data	32	输入	回写阶段的指令要写入CP0中寄存器的值
excepttype_o	32	输出	最终的异常类型
current_inst_address_o	32	输出	访存阶段指令的地址
is_in_delayslot_o	1	输出	访存阶段的指令是否是延迟槽指令
cp0_epc_o	32	输出	CP0中EPC寄存器的最新值

2.4.2 MEM/WB模块

2.4.2.1 功能描述

实现访存与回写阶段之间的寄存器，将访存阶段的结果暂时保存，在下一个时钟周期的上升沿传递到回写阶段，对应mem_wb.v文件。

2.4.2.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
clk	1	输入	时钟信号
mem_wd	5	输入	访存阶段的指令最终要写入的目的寄存器地址
mem_wreg	1	输入	访存阶段的指令最终是否有要写入的目的寄存器
mem_wdata	32	输入	访存阶段的指令最终要写入目的寄存器的值
wb_wd	5	输出	回写阶段的指令要写入的目的寄存器地址

wb_wreg	1	输出	回写阶段的指令是否有要写入的目的寄存器
wb_wdata	32	输出	回写阶段的指令要写入目的寄存器的值
mem_cp0_reg_we	1	输入	访存阶段的指令是否要写CP0中的寄存器
mem_cp0_reg_write_addr	5	输入	访存阶段的指令要写的CP0中寄存器的地址
mem_cp0_reg_data	32	输入	访存阶段的指令要写入CP0中寄存器的数据
wb_cp0_reg_we	1	输出	回写阶段的指令是否要写CP0中的寄存器
wb_cp0_reg_write_addr	5	输出	回写阶段的指令要写的CP0中寄存器的地址
wb_cp0_reg_data	32	输出	回写阶段的指令要写入CP0中寄存器的数据
mem_whilo	1	输入	访存阶段的指令是否要写HI、LO寄存器
mem_hi	32	输入	访存阶段的指令要写入HI寄存器的值
mem_lo	32	输入	访存阶段的指令要写入LO寄存器的值
wb_whilo	1	输出	回写阶段的指令是否要写HI、LO寄存器
wb_hi	32	输出	回写阶段的指令要写入HI寄存器的值
wb_lo	32	输出	回写阶段的指令要写入LO寄存器的值
stall	1	输入	访存阶段是否暂停
flush	1	输入	是否清除流水线

2.5 回写阶段

2.5.1 CP0模块

2.5.1.1 功能描述

对应MIPS架构中的协处理器CP0，实现了Index、EntryLo0、EntryLo1、BadVAddr、Count、EntryHi、Compare、Status、Cause、EPC、EBase共11个寄存器，对应cp0_reg.v文件。

具体地，首先实现了对CP0中寄存器的写操作，依据写入地址，将输入数据保存到不同的寄存器中，这是一个时序逻辑；而后实现了对CP0中寄存器的读操作，依据读取地址，将相应寄存器的值通过data_o接口输出，这是一个组合逻辑。

2.5.1.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
clk	1	输入	时钟信号
raddr_i	5	输入	要读取的CP0中寄存器的地址
int_i	6	输入	6个外部硬件中断输入
we_i	1	输入	是否要写CP0中的寄存器
waddr_i	5	输入	要写的CP0中寄存器的地址
wdata_i	32	输入	要写入CP0中寄存器的数据

data_o	32	输出	读出的CP0中某个寄存器的值
count_o	32	输出	Count寄存器的值
compare_o	32	输出	Compare寄存器的值
status_o	32	输出	Status寄存器的值
cause_o	32	输出	Cause寄存器的值
epc_o	32	输出	EPC寄存器的值
index_o	32	输出	Index寄存器的值
bad_v_addr_o	32	输出	BadVAddr寄存器的值
entry_lo_0_o	32	输出	EntryLo0寄存器的值
entry_lo_1_o	32	输出	EntryLo1寄存器的值
entry_hi_o	32	输出	EntryHi寄存器的值
ebase_o	32	输出	EBase寄存器的值
timer_int_o	1	输出	是否有定时中断发生
excepttype_i	32	输入	最终的异常类型
current_inst_address_i	32	输入	发生异常的指令地址
is_in_delayslot_i	1	输入	发生异常的指令是否是延迟槽指令

2.5.2 HILO模块

2.5.2.1 功能描述

实现寄存器HI、LO，在乘法指令的处理过程中会使用到这两个寄存器。

2.5.2.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
clk	1	输入	时钟信号
we	1	输入	HI、LO寄存器写使能信号
hi_i	32	输入	要写入HI寄存器的值
lo_i	32	输入	要写入LO寄存器的值
hi_o	32	输出	HI寄存器的值
lo_o	32	输出	LO寄存器的值

2.6 控制部件

2.6.1 CTRL模块

2.6.1.1 功能描述

控制整个流水线的暂停、清除等动作，所以不便于将其归入流水线的某一个阶段，对应ctrl.v文件。

(1) 流水线暂停机制的设计与实现：

OpenMIPS在进行异常处理和load相关发生冲突时，需要暂停流水线，一种直观的实现方法是：要暂停流水线，只需保持取指令地址PC的值不变，同时保持流水线各个阶段的寄存器（也就是IF/ID、ID/EX、EX/MEM、MEM/WB模块的输出）不变。

OpenMIPS采用的是一种改进的方法：假如位于流水线第n阶段的指令请求流水线暂停，那么需保持取指令地址PC的值不变，同时保持流水线第n阶段、第n阶段之前的各个阶段的寄存器不变，而第n阶段后面的指令继续运行。比如：流水线执行阶段的指令请求流水线暂停，那么保持PC不变，同时保持取指、译码、执行阶段的寄存器不变，但是可以允许访存、回写阶段的指令继续运行。

为此，设计添加CTRL模块，其作用是接收各个阶段传递过来的流水线暂停请求信号，从而控制流水线各个阶段的运行。具体设计如下：

CTRL模块的输入来自ID、EX模块的请求暂停信号stallreq，对于OpenMIPS而言，只有译码、执行阶段可能会有暂停请求，取指、访存阶段都没有暂停请求，因为指令读取、数据存储器的读写操作都可以在一个时钟周期完成。

CTRL模块对暂停请求信号进行判断，然后输出流水线暂停信号stall。stall输出到PC、IF/ID、ID/EX、EX/MEM、MEM/WB等模块，从而控制PC的值，以及流水线各个阶段的寄存器。

其具体实现方法如下：

(a) 当处于流水线执行阶段的指令请求暂停时（即stallreq_from_ex等于Stop），按照OpenMIPS流水线暂停机制的设计，要求取指、译码、执行阶段暂停，而访存、回写阶段继续，所以设置stall为6'b001111。

(b) 当处于流水线译码阶段的指令请求暂停时（即stallreq_from_id等于Stop），按照OpenMIPS流水线暂停机制的设计，要求取指、译码阶段暂停，而执行、访存、回写阶段继续，所以设置stall为6'b000111。

(c) 其余情况下，设置stall为6'b000000，表示不暂停流水线。

当发生异常时（excepttype_i不为0），CTRL模块会依据异常类型，给出新的取指地址（即异常处理例程入口地址），同时决定是否要清除流水线。

2.6.1.2 端口说明

接口名	宽度(bit)	输入/输出	作用
rst	1	输入	复位信号
stallreq_from_id	1	输入	处于译码阶段的指令是否请求流水线暂停信号
stallreq_from_ex	1	输入	处于执行阶段的指令是否请求流水线暂停信号
stall	6	输出	暂停流水线控制信号
cp0_epc_i	32	输入	EPC寄存器的最新值
excepttype_i	32	输入	最终的异常类型
new_pc	32	输出	异常处理入口地址
flush	1	输出	是否清除流水线

2.7 MMU

MMU单元负责进行虚拟地址到物理地址的转换。具体映射方案如下。

物理地址范围	设备	说明
$[0x00000000, 0x007FFFFF]$	RAM	共8MB
$[0x1FC00000, 0x1FC00FFF]$	引导ROM	共4KB
$[0x1E000000, 0x1EFFFFFF]$	flash	地址空间共16MB，但对于32位字，只有低16位有效
$[0x1A000000, 0x1A096000]$	VGA	
0x1FD003F8	串口数据	
0x1FD003FC	串口状态	
0x1FD00400	数码管	
0x0F000000	键盘码	

3 整体设计

3.1 CPU整体设计

本实验实现了基于标准32位MIPS指令集的子集的五级流水CPU，支持异常、中断、TLB等。

MIPS指令集的各条指令可以分解为取指（IF）、译码（ID）、执行（EX）、访存（MEM）和回写（WB）五个阶段，分别对应于本次实验需要实现的五个核心模块。模块内部采用组合逻辑电路实现，相邻模块之间的数据传输采用时序逻辑电路实现，每经过一个时钟周期，所有阶段分别将各自保存的结果交给下一个阶段，从而实现流水CPU。

此外，我们还实现了寄存器堆模块（Regfile）、HILO模块（HILO）、协处理器模块（CP0）和控制模块（CTRL）。寄存器堆模块在译码阶段实现，便于指令在译码阶段访问寄存器并得到相应的数据。HILO模块和协处理器模块在回写阶段实现。控制模块则用于控制整个流水线的暂停、清除等动作，因此不将其归入流水线中的某一个阶段。

流水线虽然能提高指令执行效率，但由此带来的冲突是不可避免的。我们对于流水线三种类型的冲突（结构冲突、数据冲突和控制冲突）分别进行了处理。

1. 结构冲突

   指令在重叠执行过程中发生硬件资源冲突，在MIPS架构中主要是指取指阶段和访存阶段同时对存储器进行访问引起的冲突。解决方法是在设计数据通路时，采用资源重复设置的方法。如果指令和数据放在同一个存储器，可使用双端口存储器，一个端口存取数据，另一个端口取指令，这两个操作可以并行操作而不引起结构冲突。

2. 数据冲突

   在同时重叠执行的几条指令中，一条指令依赖于前面指令执行结果，但是得不到执行结果造成的冲突（RAW，即Read After Write）。解决方法是在数据通路中加入流水线暂停和恢复机制（load相关），并引入数据前推技术，将执行结果提前送到其他模块，以供其他模块参考和使用。由于编译器调度容易出错，所以本项目中不在编译器上做优化。

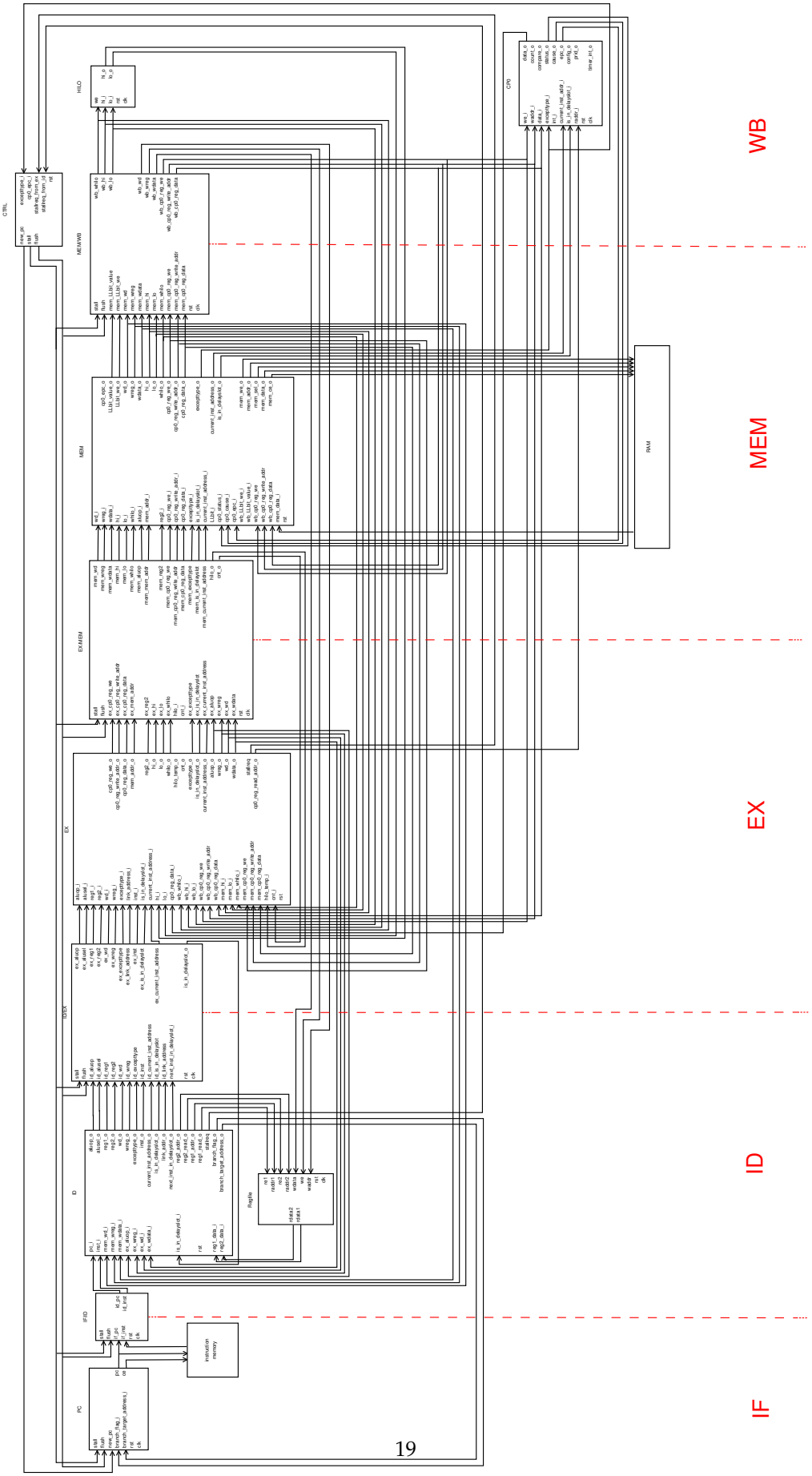
3. 控制冲突

   流水线中的分支指令或者其他需要改写PC的指令造成的冲突。解决方法是引入延时槽机制，这要求编译器在分支指令后插入空指令，对流水线的性能有一定的影响。

3.2 元件例化

见openmips.v，此文件中例化了上述所有文件，并将对应信号连接在一起。具体连接方式见数据通路图

3.3 数据通路图


IF
ID
EX
MEM
WB
PC
IF/ID
ID
ID/EX
EX
EX/MEM
MEM
MEM/WB
HILO
CP0
CTRL
RAM
instruction memory
regfile

3.4 异常处理

本次实验要求实现精确异常处理，即确定异常发生的位置，确保该指令前的指令全部执行完毕，并将该指令后的指令从流水线上擦除。

为实现精确异常，必须要求异常发生的顺序与指令的顺序相同，而在流水线处理器上，异常会在流水线的不同阶段发生，带来潜在的问题。因此，先发生的异常并不立即处理，异常事件被暂时标记，并继续运行流水线；设计一个特殊的流水线阶段，专门用于处理异常。如果某一条指令运行到了流水线的这个阶段，那么就会进行异常处理，并且当前处于流水线其余阶段的指令的异常事件都会被忽略。如此就能保证在流水线处理器中实现“按指令执行的顺序处理异常，而不是按异常发生的顺序处理异常”。

ERET指令（异常返回指令）既要清除Status寄存器的EXL字段，从而使能中断（即设置Status寄存器的EXL字段为0，表示不再处于异常级），还要将EPC寄存器保存的地址恢复到PC中，从而返回到异常发生处继续执行。

以下是一些可能用到的中断和异常的情况。

异常号	异常名	描述
0	Interrupt	外部中断、异步发生，由硬件引起
1	TLB Modified	内存修改异常，发生在Memory阶段
2	TLBL	读未在TLB中映射的内存地址触发的异常
3	TLBS	写未在TLB中映射的内存地址触发的异常
4	ADEL	读访问一个非对齐地址触发的异常
5	ADES	读访问一个非对齐地址触发的异常
8	SYSCALL	系统调用
10	RI	执行未定义指令异常
11	Co-Processor Unavailable	试图访问不存在的协处理器异常
23	Watch	Watch寄存器监控异常

下表列出可能用到的中断号。

中断号	设备
0	系统计时器
1	键盘
3	通讯端口COM2
4	通讯端口COM1

中断/异常处理过程如下：

1. 保存中断信息，主要是EPC，BadVaddr，Status，Cause等寄存器的信息

   EPC:存储异常处理之后程序恢复执行的地址。对于一般异常，当前发生错误的指令地址即为EPC应当保存的地址（如果发生异常的指令在延迟槽中，那么设置EPC寄存器的值为该指令的地址减4）；而对于硬件中断，由于是异步产生则可以任意设定一条并未执行完成的指令地址保存，但在进入下一步处理之前，该指令前的指令都应当被执行完。

   BadVAddr：捕捉最近一次地址错误或TLB异常（重填、失效、修改）时的虚拟地址。

   Status：将EXL位置为1，进入kernel模式进行中断处理。

   Cause：记录下异常号。

   EntryHi：TLB异常时，记录下BadVAddr的部分高位。

2. 根据Cause中的异常号跳转到相应的异常处理函数入口
3. 中断处理
4. 通过调用ERET指令恢复现场，返回EPC所存地址执行并且将Status中的EXL重置为0表示进入user模式。

实现思路:

在流水线的各个阶段收集异常信息，并传递到流水线访存阶段，在访存阶段统一处理异常信息。

在流水线访存阶段，处理器将结合协处理器CP0中相关寄存器的值，判断异常是否需要处理，如果需要处理，那么转移到该异常对应的处理例程入口地址，清除流水线上除回写阶段外的全部信息（回写阶段的指令要继续执行），同时修改协处理器CP0中相关寄存器的值。

如果是ERET指令，那么转移到EPC寄存器保存的地址处，同时，也要清除流水线上除回写阶段外的全部信息，修改协处理器CP0中相关寄存器的值。

其中，清除流水线上某个阶段的信息，实际上就是讲该阶段中的所有寄存器设置为初始值即可。

具体步骤如下:

1. 在可能发生异常的位置实现对异常的记录。

   (1) 访存时可能发生ADEL，ADES，TLBM，TLBL，TLBS，Watch异常

   (2) 译码后可能发生RI，SYSCALL，Co-ProcessorU异常

2. 实现对中断的记录。

   (1) 硬件产生中断时将信息写入CP0寄存器

3. 根据异常记录信息判断是否产生异常。
4. 进入异常处理流程。

4 总线

为了实现CPU与外设的通信而加上了总线的设计。设计思想是CPU将选择的外设、传给外设的地址和数据输出给总线，总线将这些信号传给对应外设，外设再将CPU需要的信号通过总线传回。为了处理结构冲突，在总线上加入了仲裁器，来决定在当前处理周期进行什么操作。

4.1 仲裁器

总裁器是为了总线为了处理结构冲突而增加的一个部件，主要作用是确定总线在当前周期的操作应当是取指还是访存。

在优先级上，我们设定取指的优先级最高，访存读次之，访存写最次。根据该周期进行的操作，分为以下几种情况处理:

1. 只进行取指操作

   在这种情况下不需要暂停流水线，直接执行取指即可。

2. 只进行取指和访存读操作

   在这种情况下需要在访存阶段插入气泡，访存读行为需要等待取指行为结束后才能进行。在这个过程中流水线需暂停，直到访存读操作完成流水线才可以继续。

3. 只进行取指和访存写操作

   在这种情况下需要在访存阶段插入气泡，访存写行为需要等待取指行为结束后才能进行。在这个过程中流水线需暂停，直到访存写操作完成流水线才可以继续。

4. 需进行取指、访存读和访存写操作

   在这种情况下需要在访存阶段插入气泡，使得取指、访存读、访存写行为能够依次执行。在这个过程中流水线需暂停，直到访存写操作完成流水线才可以继续。

5. 仲裁器收到了来自CPU的暂停请求

   这种情况是由于CPU遇到了load相关问题，需要令数据前推而产生的。load相关问题是指CPU在load类指令后遇到了跳转指令从而导致的数据冲突问题。在CPU的模块中，我们为了处理这个冲突，会在流水线中插入一个气泡，从而使跳转指令的译码晚一个周期，令load类指令能完成访存，从而在访存阶段得以将数据前推回译码阶段。

   对仲裁器而言，当遇到来自CPU的暂停请求时，就意味着仲裁器需要跳过取指阶段，直接执行访存读。

4.2 外设处理

选定外设后，总线给出地址、数据和读写使能，外设则将数据写入对应地址或从对应地址取出数据传回总线。当读/写操作完成后会给出ack信号，传回仲裁器中。对仲裁器而言，如果没有来自外设的ack信号，便会暂停流水线，直到收到ack信号为止。

以下是外设的说明。

1. RAM

   为了加快从RAM里取指的速度，将RAM的读操作压缩在一个周期内完成。实验证明在25M时钟下可正确运行，在50M时钟下会取指错误。RAM写则是两个周期。

   RAM可读可写。

2. ROM

   ROM只用于取指令，里面的指令直接写在代码里。当需要从中取指时，只需一个周期便可从对应地址取出指令。在运行ucore的时候ROM里放的是bootloader的指令。

   ROM只可读不可写。

3. FLASH

   由于FLASH断电可保存信息，所以提前将ucore的内容写进FLASH中。在bootloader时，CPU将存放在FLASH里的ucore指令取出，写进RAM中。由于FLASH读操作较慢，所以不使用CPU主频，而是接50M时钟。在50M时钟下，FLASH读需要八个周期。

   FLASH只可读不可写。

4. UART

   我们参考了fpga4fun中处理串口的代码。串口也不使用CPU主频，而是接50M时钟。

   UART可读可写。