****

**《计算组成原理实验》**

**实验报告**

**（实验二）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院名称** | **：** | 计算机学院 | | | | | |
| **专业（班级）** | **：** | 21级行政1班 | | | | | |
| **学生姓名** | **：** | 黄鑫 | | | | | |
| **学 号** | **：** | 21307008 | | | | | |
| **时间** | **：** | 2023 | 年 | 1 | 月 | 11 | 日 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | **成绩** | **:** |  |
| **实验二** | **：** | **中断与异常（YstCPU Risc-V）** | | | |

**目录**

[第一章 实验目的 4](#_Toc125055543)

[第二章 实验器材 4](#_Toc125055544)

[第三章 实验任务 4](#_Toc125055545)

[第四章 实验原理与分析 4](#_Toc125055546)

[1. 中断与异常 4](#_Toc125055547)

[2. 中断控制器CLINT 5](#_Toc125055548)

[3. 定时中断发生器Timer 5](#_Toc125055549)

[4. 中断具体执行流程 5](#_Toc125055550)

[5. CSR寄存器组 6](#_Toc125055551)

[6. CSR指令 8](#_Toc125055552)

[7. 数据通路与控制线路 9](#_Toc125055553)

[8. 新增的控制信号的解释 10](#_Toc125055554)

[9. 新增的模块接口的解释 10](#_Toc125055555)

[10. CSR指令的执行流程 10](#_Toc125055556)

[第五章 实验过程与结果 11](#_Toc125055557)

[1. InstructionFetch 11](#_Toc125055558)

[2. InstructionDecode 11](#_Toc125055559)

[3. Execute 11](#_Toc125055560)

[4. CSRFile 12](#_Toc125055561)

[5. CLINT 13](#_Toc125055562)

[6. Timer 13](#_Toc125055563)

[7. 标准测试全部通过 14](#_Toc125055564)

[第六章 实验心得 15](#_Toc125055565)

# 实验目的

1. 掌握CSR 寄存器以及其操作命令；

2. 掌握中断控制器的原理和设计

3. 编写一个简单的定时中断发生器；

4. 在单周期CPU上实现中断功能

# 实验器材

电脑一台，Intellij IDEA 软件一套，Basys3板一块。

# 实验任务

lab2中的实验任务如所示：

1. EX 执行单元在处理 CSR 指令时能够正确地得到写入 CSR 寄存器的数据。
2. CSR 寄存器组可以正确支持CLINT和来自CSR指令的读写操作。
3. 定时中断发生器可以正确产生中断信号，并且实现 Timer 寄存器的 MMIO。
4. CLINT 能够正确的响应中断并且在中断结束后回到原来的执行流。

EX 执行单元的代码文件位于 src/main/scala/riscv/core/Execute.scala

CSR 寄存器组的代码文件位于 src/main/scala/riscv/core/CSR.scala

CLINT 的代码文件位于 src/main/scala/riscv/core/CLINT.scala

Timer 的代码位于 src/main/scala/riscv/peripheral/Timer.scala

在上面提到的 EX、CSR、CLINT、Timer 四个单元的相应文件里面，请在 // lab2(CLINTCSR) 注释处填入相应的代码，使其能够通过 CPUTest、ExecuteTest、CLINTCSRTest、TimerTest 测试。

# 实验原理与分析

本实验里我们在上一个实验中单周期 CPU 的基础上增加中断与异常的功能，通过增加对 CSR 寄存器的操作指令的支持来实现。相关介绍：

## 中断与异常

完成单周期 CPU 实验后，你获得了一个可以简单的按照预期指令执行的处理器。但是这个简单的 CPU 只能按照预先的程序指令一直运行，无法中途打断。

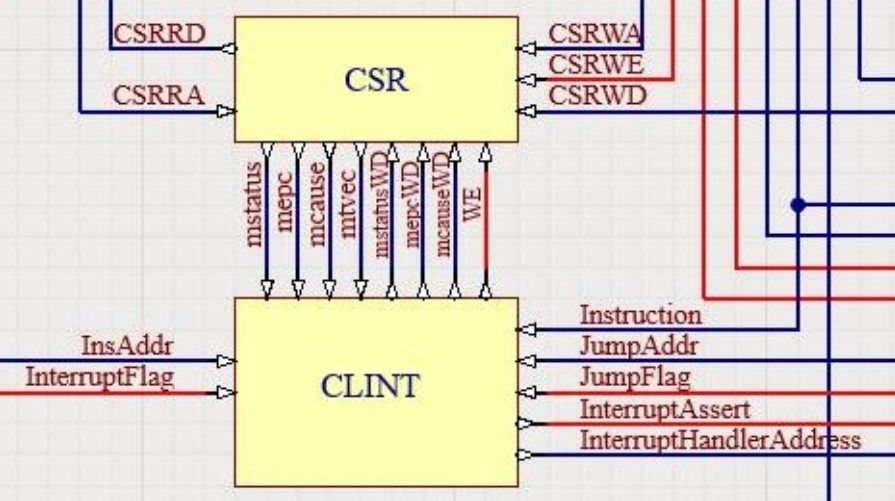
而中断的功能则是，让CPU每隔一段时间就强行暂停当前运行的程序，并跳转到操作系统设置的中断处理程序中。这便能让CPU去执行别的进程，例如用户的IO输入，或伪并行其他进程，具体执行什么中断处理程序由OS决定。这种实现中断的方式叫抢占式调度，我们的任务就是在单周期CPU上实现一个抢占式调度。

## 中断控制器CLINT

在[具体实现](#_数据通路与控制线路)中，“每隔一段时间”的功能由一个MMIO 的定时中断发生器Timer实现；而IO设备的输入则有UART单元提供；CSR寄存器组通过一个中断控制器CLINT(control interrupt)模块与两个外设Timer和UART对接。

CLINT 需要一个周期就把多个寄存器的内容修改的功能，而正常的 CSR 指令只能对一个寄存器读-修改-写（Read-Modify-Write, RMW）。所以 CLINT 和 CSR 之间有独立的优先级更高的通路，用来快速更新 CSR 寄存器的值。

如下图，其中InterruptFlag的低两位分别来自Timer和MMIO的中断信号：



## 定时中断发生器Timer

我们要实现一个 MMIO 的定时中断发生器——Timer。

MMIO 简单来说就是：在该外设中与 CPU 交互的寄存器是与内存一起编址的，所以 CPU 可以通过访存指令（load/store）来修改这些寄存器的值，从而达到 CPU 和外设交互的目的。逻辑地址的高位作为外围设备的位选信号即可，低位则用于设备内部的寻址。这里我们使用多路选择器和已有的MemoryControl实现MMIO。

Timer内部由两个控制寄存：enable寄存器和limit寄存器：

1. enable 寄存器用来控制定时中断发生器的使能，为 false 则不产生中断，映射到地址空间的逻辑地址为 0x80000008。
2. limit 寄存器用来控制定时器的中断发生间隔，映射到地址空间的逻辑地址为 0x80000004。中断发生器内部有一个加一计数器，当计数器的值到达 limit 为标准的界限时，定时器会发生一次中断信号（enable 使能情况下）。注：产生中断信号的时长没有太大关系，但是至少应该大于一个 CPU 时钟周期，确保 CPU 能够正确捕捉到该信号即可。

## 中断具体执行流程

中断的本质是切换进程，抽象地讲，由CLINT和CSRFile构成的中断器就是个进程切换器，它只在切换进程时工作，即进入中断阶段与离开中断阶段。按照状态机的思想，我们可以将中断功能分为3个状态：进入中断、离开中断、不工作（未中断或中断中）。

为了尽快处理中断，CPU 仅会将最基本的中断发生地址和中断原因等保存到 CSR 寄存器中。更复杂的功能交给中断处理程序实现。每次执行什么中断处理程序由OS决定，因此我们只需要研究3个状态下的①中断发生地址和中断原因等数据的变化，以及②跳转的相关信号与地址。

这些数据包括：中断程序地址，中断返回地址，中断开关和中断原因。具体见CSR寄存器组。

综上所述，中断的具体执行流程是：

1. CLINT获得中断信号；
2. CLINT修改3个CSR：mepc保存返回指令地址、mstatus关中断、mcause记录中断原因；
3. CLINT控制CPU跳转到mtvec指示的指令地址，去执行中断处理程序。
4. 中断处理过程中，CLINT不改变CSR内容，也不进行跳转控制，即休眠。
5. 一段时间后，CPU运行到了mret指令，开始退出中断：
6. CLINT这次只改变一个CSR：将mstatus中的中断开关重新打开；
7. CLINT控制CPU跳转返回原有程序，目标指令地址在mepc中。
8. CLINT再次休眠，直到下一次获得中断信号。

## CSR寄存器组

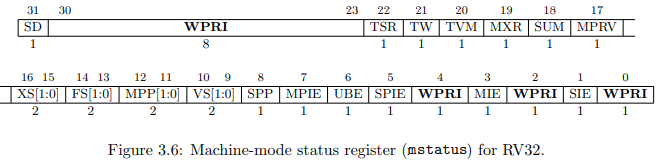
CSR (Control and Status Registers, 控制和状态寄存器) 是用来控制和保存 CPU 的其他功能的状态，例如中断使能状态、特权等级等。

而CSR寄存器组则是一组类似于 RegisterFile 的寄存器组，地址空间大小为 4096 字节，独立编址的寄存器。只有中断控制器CLINT和CSR指令才能修改CSR。

下面列出一些与中断过程相关的重要CSR：

1. **mstatus**

mstatus寄存器用于记录机器模式下的状态，其中第3位为中断开关。

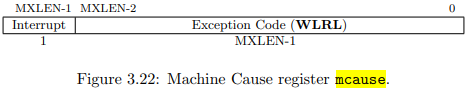


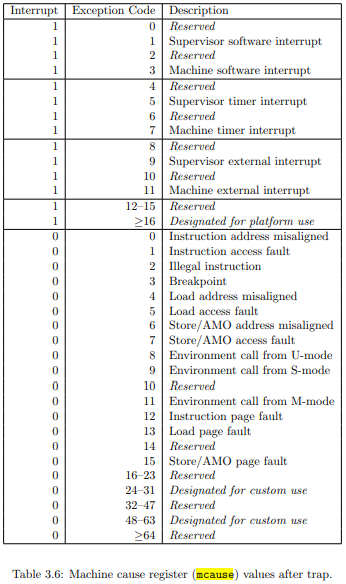
1. **mepc**

mepc寄存器保存了中断返回后需要执行的指令地址，当 CPU 响应中断时，如果 EX 阶段正在执行跳转，则设置为跳转的目标地址，否则设置为当前指令的下条指令的地址；当CPU返回中断时，mepc需要将保存的指令地址以跳转地址的方式交还给pc。

1. **mcause**

mcause寄存器保存了中断的原因。其中我们将用到的有Timer中断Machine timer interrupt和外设UART中断Machine external interrupt。





1. **mtvec**

mtvec寄存器保存了中断处理程序的地址，当CPU执行中断时， mtvec寄存器中保存的中断处理程序的地址将以跳转地址的方式交给pc寄存器。

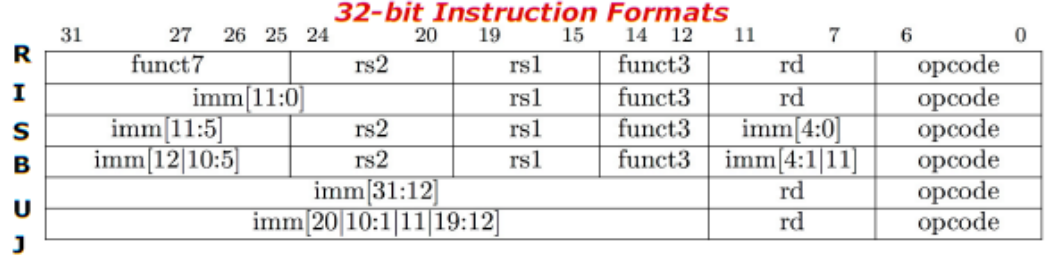
在中断发生的时候，单周期CPU需要在 CSR 寄存器写入中断相关的信息。由于CSR寄存器堆实现只有一个读写端口，因此，CPU需要多个周期才能完成 CSR寄存器的写入。在设置完 CSR寄存器后，发出控制信号，CPU跳转到mtvec中保存的指令地址，开始执行中断处理程序。

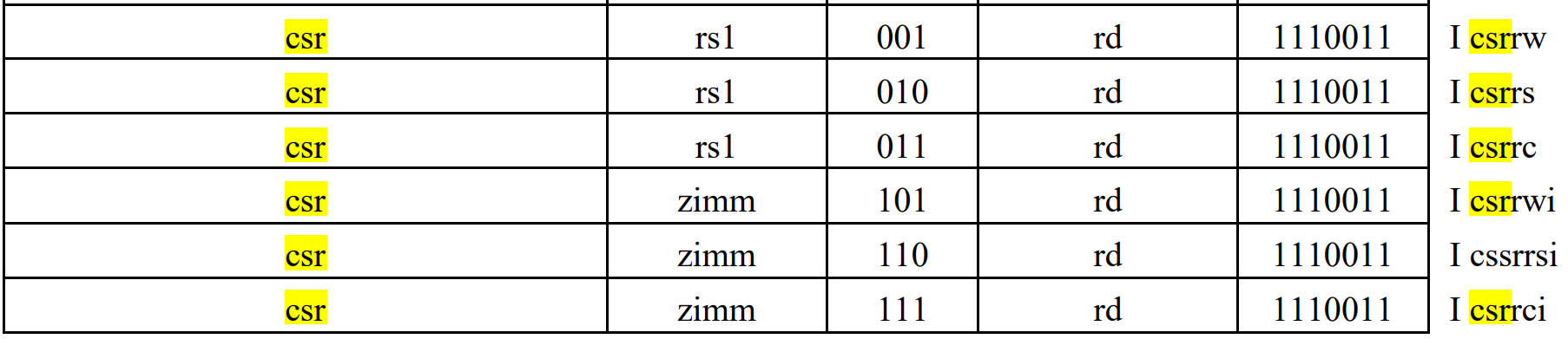
## CSR指令

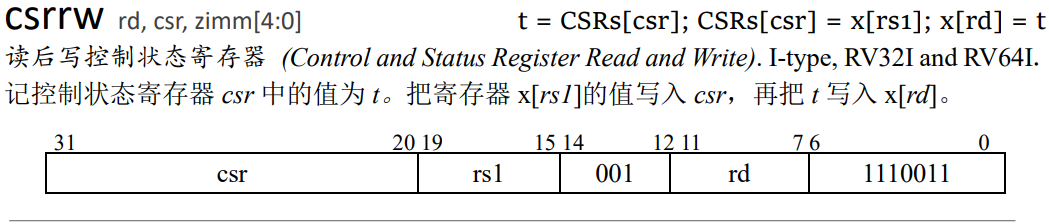
从指令手册可以看到对CSR寄存器的操作都是原子读写的。

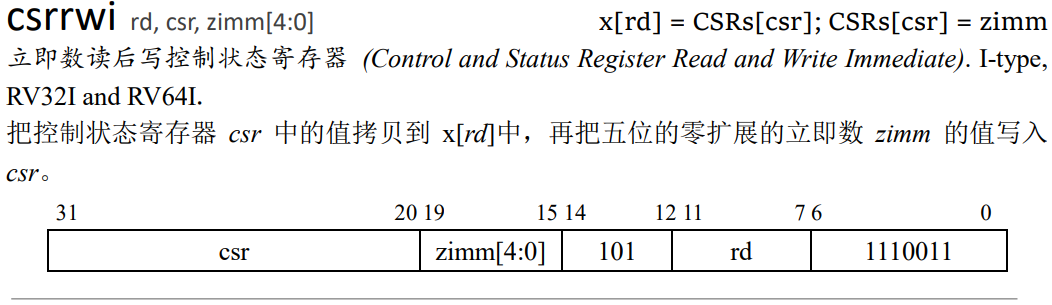
CSR指令通过ID模块译码后给出控制信号和CSR寄存器地址，对CSR组内部寄存器进行寻址，获取其内容并且修改。

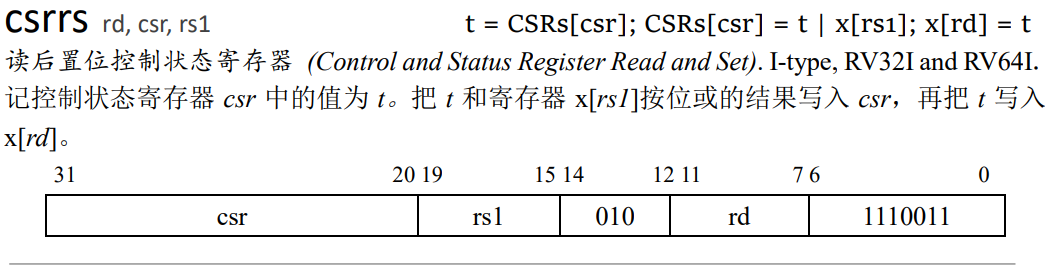
本实验新增的RV32I CSR指令共六条：csrrw、csrrs、csrrc、csrrwi csrrsi、csrrci。这些指令本质上是I-type。

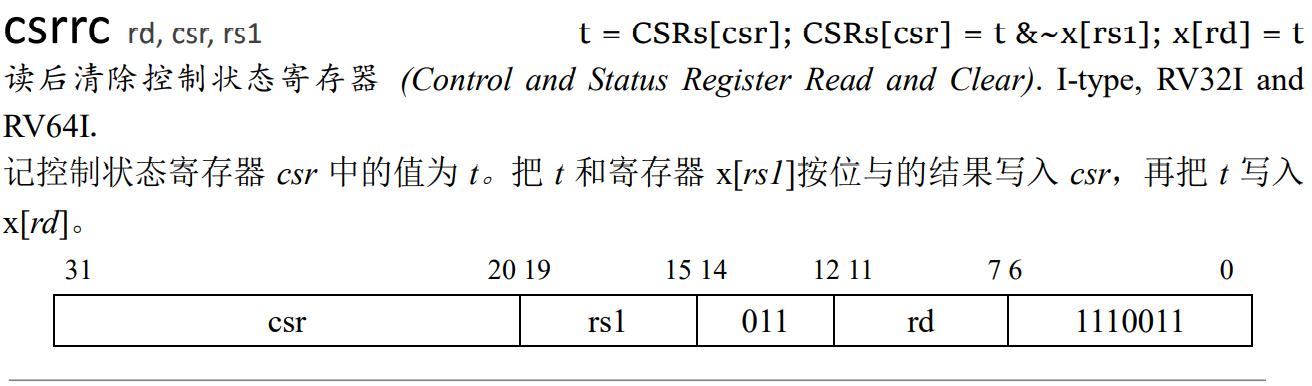




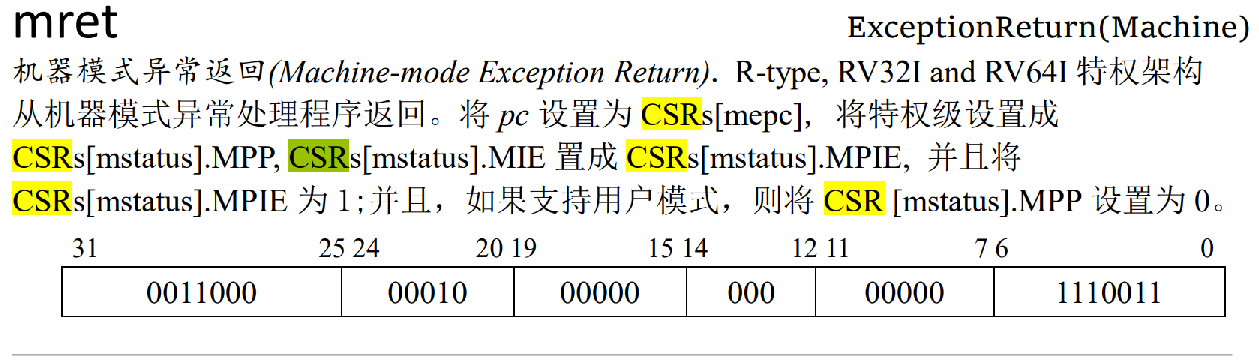




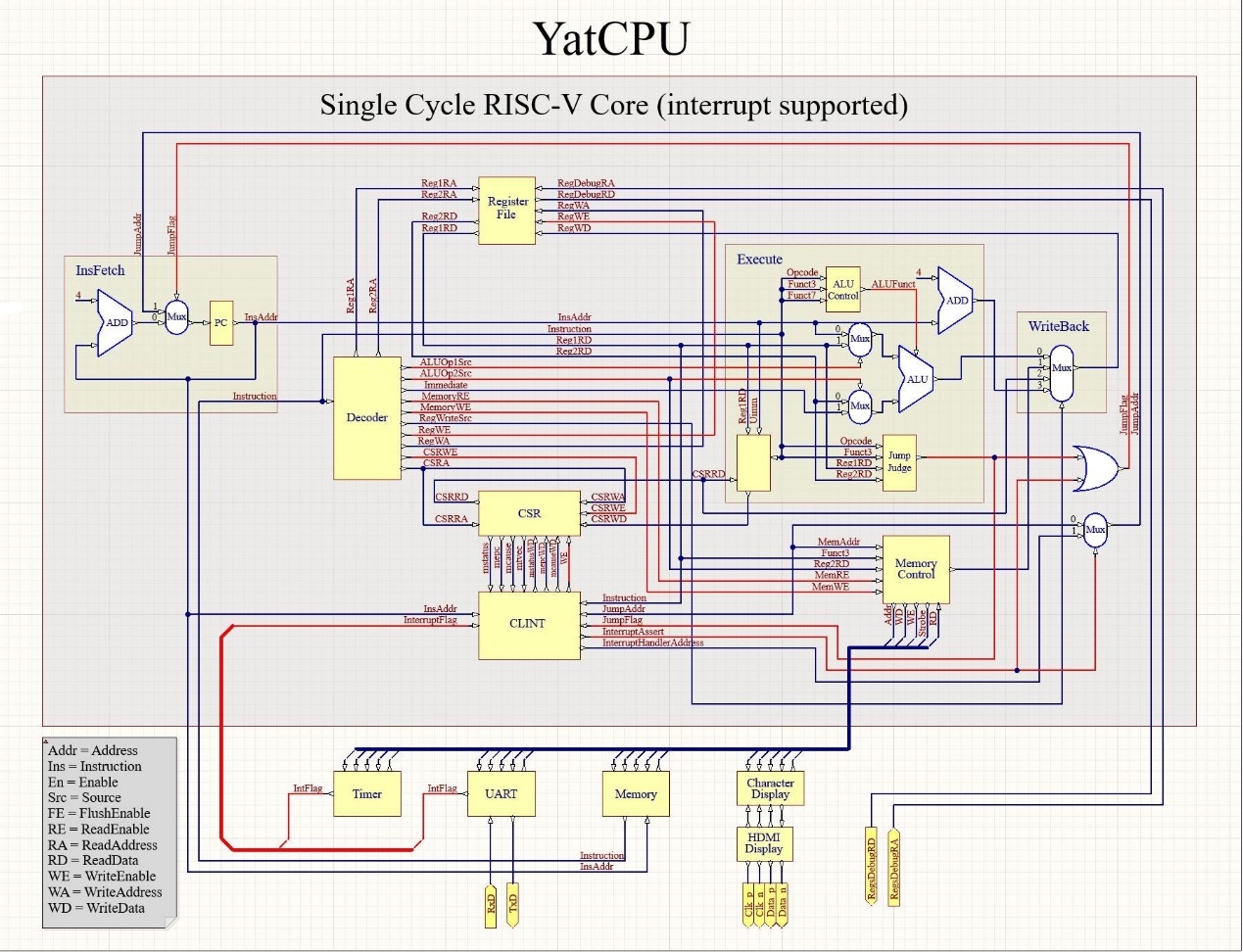




此外，



## 数据通路与控制线路



上图有一些接线错误，下面是更正：

1. InsFetch模块JumpAddr线和和CLINT的JumpAddr线，它们另一端应该连接Execute模块中的一个JumpAddr计算单元的输出，该单元在图中没有画出（不过我在lab1中的实验报告中描述了这个模块）。
2. Memory Control模块的MemAddr线应该连接上ALU的输出。
3. Memory Control模块的Funct3线和CLINT的Instruction线，它们的另一端应接在Decoder输出的Instruction线上。

新增的控制信号的解释

|  |  |
| --- | --- |
| 控制信号 | 含义 |
| InterruptFlag | 来自外设的中断信号，第0位来自Timer，第1位来自UART |
| InterruptAssert | 由CLINT发出的跳转信号，用于响应/退出中断 |
| CSR与CLINT之间的WE | CLINT对CSR的写使能 |
| CSRWE | CSR指令对CSR的写使能 |

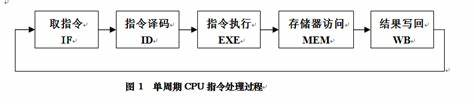
新增的模块接口的解释

|  |  |  |
| --- | --- | --- |
| InterruptAssert Address | 由CLINT发出的跳转目标地址，用于响应/退出中断 | |
| CLINT与CSR之间的 | mstatus | 当前CSR中的mstatus寄存器的值 |
| mcause | 当前CSR中的mcause寄存器的值 |
| mepc | 当前CSR中的mepc寄存器的值 |
| mtvec | 当前CSR中的mtvec寄存器的值 |
| mstatusWD | CLINT对mstatus的写数据 |
| mepcWD | CLINT对mepc的写数据 |
| mcauseWD | CLINT对mcause的写数据 |
| Uimm | CSR指令中5位无符号数的零拓展结果 | |
| CSRA | CSR指令对CSR的地址（读或写） | |
| CSRRA | CSR指令对CSR的读地址 | |
| CSRWA | CSR指令对CSR的写地址 | |
| CSRRD | CSR指令对CSR的读出数据 | |
| CSRWD | CSR指令对CSR的写入数据 | |

## CSR指令的执行流程

CSR指令的执行流程大致与普通R指令（例如add）相同，区别在于将MEM阶段取消了，而ID、EX阶段都要访问CSR。

下面以csrrs指令为例：



（1） InsFetch是公共取指模块。

（2）在InsDecode阶段中，译码器Decoder译出zimm、CSRA与CSRWE。接着ID阶段还要完成读寄存器组的任务，从RegFile中读出Reg1Data交给EX阶段；从CSRFile中读出CSRRD，交给EX阶段和WB阶段。

（3） EX阶段中，一个新的单元将5位的zimm零拓展为uimm。同时该单元由funct3知当前为csrrsi指令，故将CSRRWD赋值为Reg1Data | CSRRD并写入CSR。

|  |  |
| --- | --- |
| funct3 | CSRWD |
| csrrw | reg1\_data |
| csrrs | CSRRD | Reg1Data |
| csrrc | CSRRD & （~Reg1Data） |
| csrrwi | uimm |
| csrrsi | CSRRD | uimm |
| csrrci | CSRRD & （~uimm） |

（4） WB阶段中的四路选择器的第2个输入是CSRRD，而此时的RegWriteSrc将会选择CSRRD。

# 实验过程与结果

有了上面的铺垫后，补充代码实现支持中断和CSR指令的功能就很容易了。

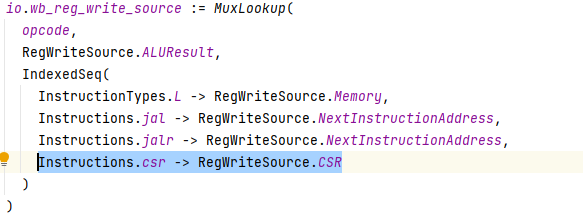
## InstructionFetch

公共取指模块无需修改。

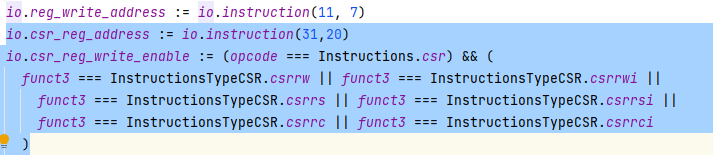
## InstructionDecode

ID 译码单元需要添加识别 CSR 指令的功能，即根据CSR指令的语义和编码规范，产生相应的传给其它模块的控制信号与数据。

1. 在RegWriteSrc中新增一行：



1. 增加2个CSR相关输出



## Execute

正如前面的分析，我们只需增加一个零拓展与CSRRWD输出单元：



## CSR

正如[前面的分析](#_CSR寄存器组)，只有中断控制器CLINT和CSR指令才能修改CSR。



此外，CSR还需要输出4个寄存器的内容给CLINT



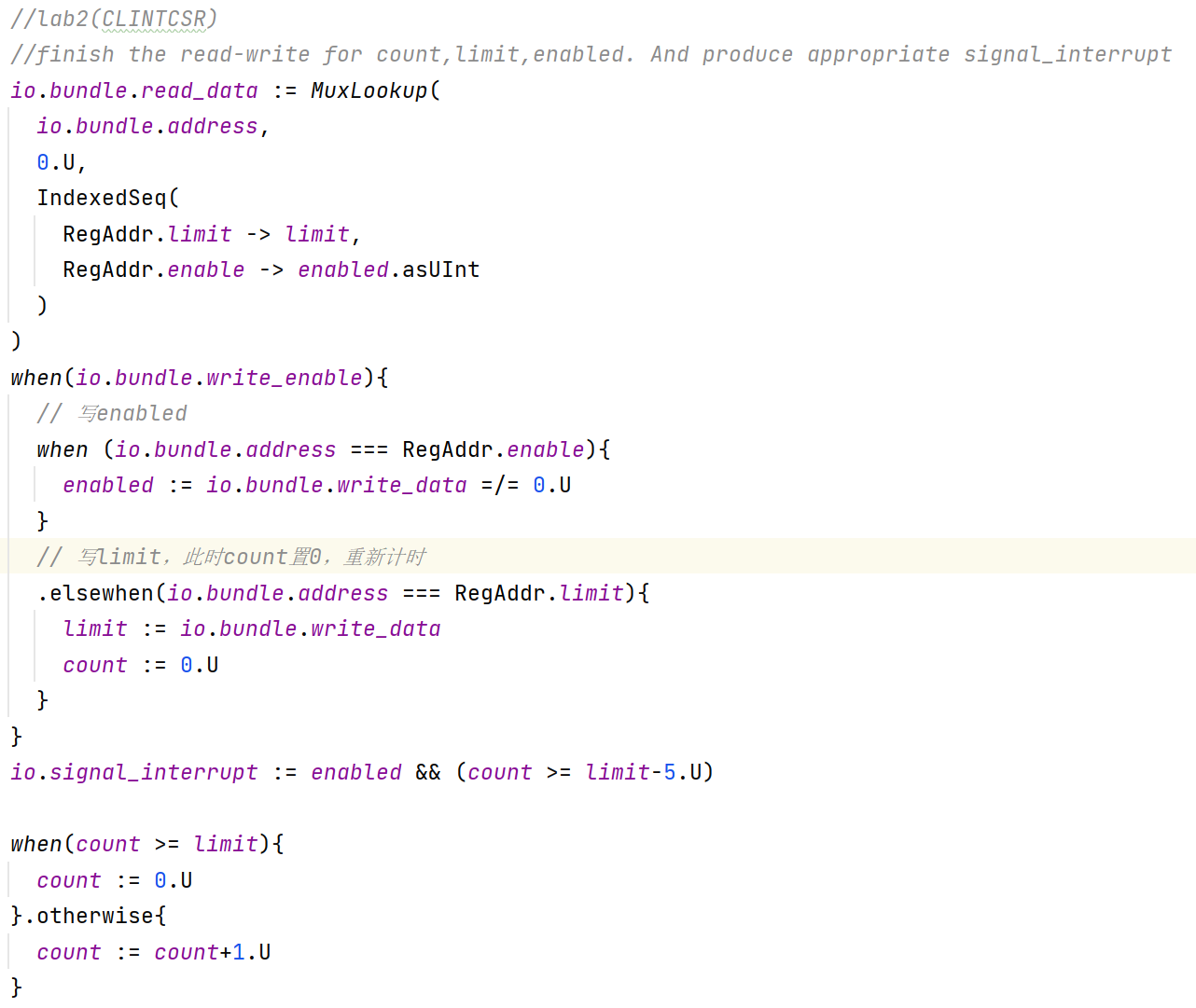
## CLINT

正如[前面的分析](#_中断具体执行流程)，实现CLINT的状态机：



## Timer

正如[前面的分析](#_定时中断发生器Timer)，设置两个控制寄存器enable与limit的关系，并则加上MMIO的功能：

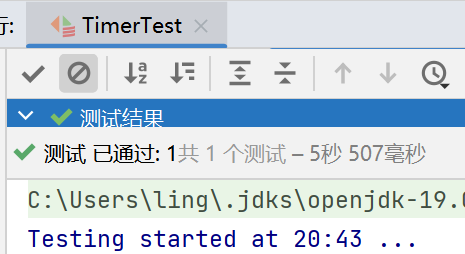


## 标准测试全部通过

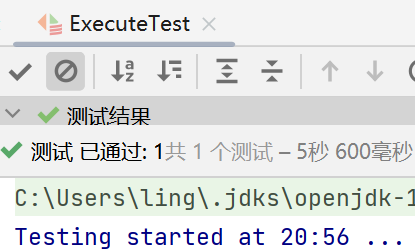
CLINT和CSR



TimerTest



ExecuteTest



CPUTest



# 实验心得

现在来捋一下整个lab2。lab2的任务可分为两部分，第一个是增加CSR指令支持（CSR就是那6条指令），第二个就是增加中断支持，通过实现一个CLINT。两者之间影响较小，将两个任务分开理解会更好理解。

增加CSR指令支持。首先是找出那6条指令的详情，拿一条作为例子捋清楚执行流程（这是在lab1学到的方法）即可。

增加中断支持，通过实现一个CLINT。把CLINT当做状态机理解，分成三个状态：进入中断、退出中断、正在中断或未中断（两者是一个状态）。接着，研究各个状态CLIINT的工作，其实很简单，也是两部分：跳转与CSR寄存器。研究跳转就要先捋清楚中断的执行流程（看吧又是捋流程）；CSR寄存器也不难：mepc、mcause、mstatus、mtvec，四个寄存器查手册弄清楚用途即可（看吧又是查手册）。

以上。

p.s.很高兴看到我在[github](https://github.com/SYSU-SCC/yatcpu-docs/issues/13)上反馈的[数据通路与控制线路图片](#_3._典型指令的具体细节)错误得到了更正。