Lab6: 综合实验

朱熙哲

3220103361

2024 年 1 月 10 号

目录

Chapter 1

硬件改造

1.1 数据通路

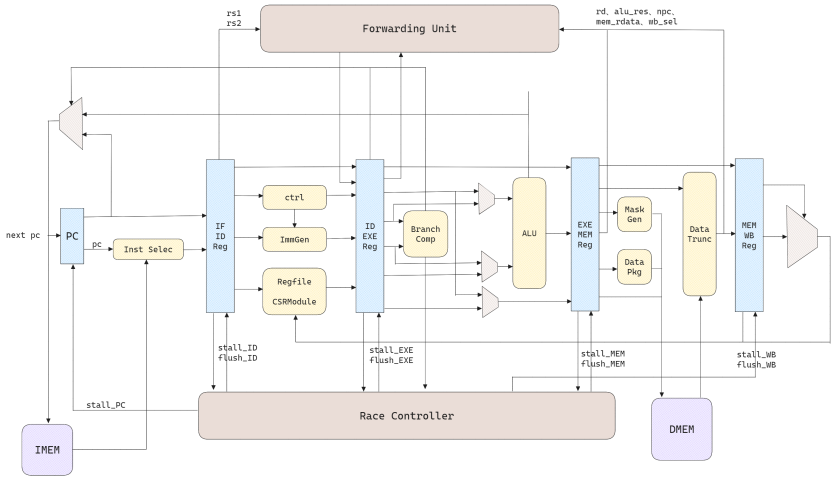

图 1.1: 数据通路

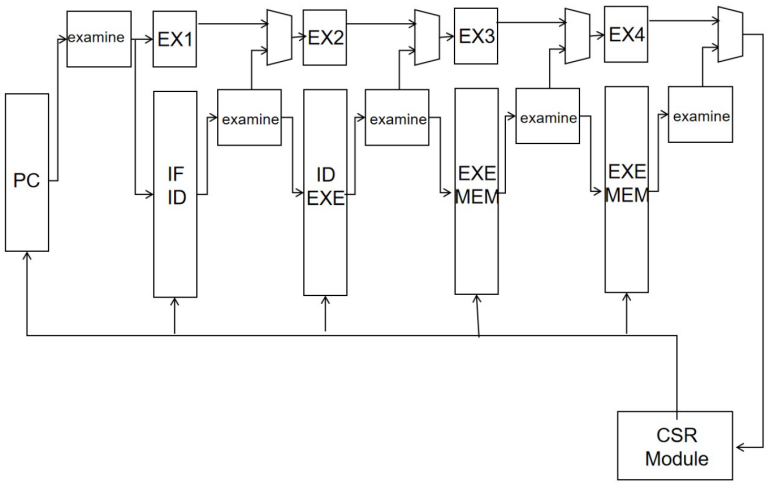

图 1.2: 异常处理

1.2 CSR 模块

1.2.1 CSRModule

已经给出，只需接线即可。

- 读写寄存器部分：
  id 阶段读寄存器，wb 阶段将写使能、写回地址、写回 csr 的值传给 CSRModule。**注意**，当 switch_mode 生效时，写使能应置零。

- 异常处理部分：
  将当前指令的 pc、valid 等传入，如发生中断或异常，需要保存当前指令并切换模式。

- 特权模式部分：
  需输出当前特权等级供异常检测模块使用，还需要输出 switch_mode 和 pc_csr，以进行模式切换和 PC 修改。

1.2.2 Forwarding 与 Register 改写

由于 CSR 中采用上升沿写回，则 Regs 也与之同步改为上升沿写回，因此 ForwardingUnit 中不仅要增加 CSR 的前递部分，也要对二者在 wb 阶段的值进行前递。

1.2.3 CSR 控制信号

```
# SYSTEM
           31-20 19-15  14-12  11-7  6-0
ecall      0x000  0     func3=0  0   opcode=0b1110011
ebreak     0x001  0     func3=0  0   opcode=0b1110011
uret       0x002  0     func3=0  0   opcode=0b1110011
sret       0x102  0     func3=0  0   opcode=0b1110011
mret       0x302  0     func3=0  0   opcode=0b1110011
csrrw      csr    rs1   func3=1  rd  opcode=0b1110011
csrrs      csr    rs1   func3=2  rd  opcode=0b1110011
csrrc      csr    rs1   func3=3  rd  opcode=0b1110011
csrrwi     csr    imm   func3=5  rd  opcode=0b1110011
csrrsi     csr    imm   func3=6  rd  opcode=0b1110011
csrrci     csr    imm   func3=7  rd  opcode=0b1110011
```

指令机械码如上，ecall、ebreak 等指令会在 ExceptExamine 模块中判断，只需解码 6 条 csr 指令和 sret、mret 指令即可。

增加 we_csr 写使能，csr_sel 选择器信号，csr_ret 特权态返回指令。ALU_MUX 中增加读取 csr 的值以写回 rd；增加 CSR_MUX 执行对 csr 的操作，直接在其中使用 rs1 或 imm 对 csr 做赋值或置位或清零操作。

csr_ret 信号会在 wb 阶段写入 CSRModule，判断是否进行异常返回。

1.3 RaceController

switch_mode 信号会接入 RaceController 中，以排空所有阶段寄存器，注意此时 stall_pc 必须置零，以使得 pc_csr 能传入 PC 中。或者可以在 PC 中将 switch_mode 的判断优先级设为高于 stall。

在即将排空阶段寄存器的该拍内，寄存器写回将要完成，因此传回 Regs 的 `we_reg` 应改为 `we_reg&~switch_mode` 。

1.4 MEM 阶段数据通路修改

由于先前使用 BRAM 完成内存读写，则需要提前一拍在 EXE 阶段就发出读写内存请求，此时由于总线和 memmap 的加入，也不再需要这种方法，延后在 MEM 阶段读写内存即可，否则由于 memmap 返回的内存值从 mem 与 mmio 择一，读入 EXE/MEM 消耗的额外一拍可能会导致读取的内存错误。

1.5 异常检测模块

在每个阶段寄存器间加入异常检测模块，其中的 ExceptExamine 模块已将 InstExamine、ExceptMux、ExceptReg 三部分打包好，直接接入 PCPU 即可。接出的 except_happen 信号表示该级流水的指令出现异常，需要清空使能信号防止影响 CPU 运行，其中需要特别判断 ecall 指令，当为该指令时，指令有效值 valid 为 1，否则应置零。

异常信号会在最后传入 CSRModule 进行异常处理。

1.6 Core2Mem FSM

当 switch_mode 发生时，如果此时在进行取指或读写内存，则应等待直至 state 重新变为 IDLE，读回的值不传入 CPU，而是直接进行下一次取

指指令, 其中 switch_flush 的逻辑如下：

```
if(switch_flush) begin
    switch_flush <= (next_state!=IDLE);
end
else begin
    switch_flush <= switch_mode;
end
```

当 switch_flush 置一时，不停止 if_stall。直至 pc_csr 对应的指令返回。

Chapter 2

软件修改

2.1 快速调试

根据实验指南修改 kernel 来方便快速调试。

- 减少线程数
- 减小内存大小
- 取消内存清零
- 减少 print 的调用和输出字符串长度

2.2 环境适配

- 使用`int_mod`、`int_mul`、`int_div`函数来做整数乘除指令的软件模拟，以避免编译器编译出乘除指令，主要在 rand 函数和 printk 函数中做修改。
- S 态无法直接通过 rdtime 获得 mtime 寄存器的值，因此需要修改`sbi_set_timer`、`clock_set_next_event`等函数，或者直接删去，通

过sbi_ecall 直接将时钟中断间隔传递给 minisbi，由它设置下一次中断。

Chapter 3

仿真结果与上板结果

板验与仿真都已线下通过，此处不多展示。

Chapter 4

思考题

4.1 putc 特权态切换

- S 态 →M 态:
  putc 会调用 sbi_ecall，抛出异常，然后陷入 M 态的异常处理函数。
- M 态 →S 态:
  在 M 态的 trap 处理完成后，会执行 mret 异常返回，到 putc 函数中，也就切换回 S 态。

```
00000000802003b8 <putc>:
802003b8:   fe010113          addi    sp,sp,-32
802003bc:   00113c23          sd  ra,24(sp)
802003c0:   00813823          sd  s0,16(sp)
802003c4:   02010413          addi    s0,sp,32
802003c8:   00050793          addi    a5,a0,0
802003cc:   fef407a3          sb  a5,-17(s0)
802003d0:   fef44603          lbu a2,-17(s0)
802003d4:   00000893          addi    a7,zero,0
802003d8:   00000813          addi    a6,zero,0
802003dc:   00000793          addi    a5,zero,0
802003e0:   00000713          addi    a4,zero,0
802003e4:   00000693          addi    a3,zero,0
802003e8:   00000593          addi    a1,zero,0
802003ec:   00100513          addi    a0,zero,1
802003f0:   599000ef          jal ra,80201188 <sbi_ecall>
802003f4:   00000013          addi    zero,zero,0
802003f8:   01813083          ld  ra,24(sp)
802003fc:   01013403          ld  s0,16(sp)
80200400:   02010113          addi    sp,sp,32
```

图 4.1: putc

```
800001c8:   0d013d03          ld  s10,208(sp)
800001cc:   0d813d83          ld  s11,216(sp)
800001d0:   0e013e03          ld  t3,224(sp)
800001d4:   0e813e83          ld  t4,232(sp)
800001d8:   0f013f03          ld  t5,240(sp)
800001dc:   0f813f83          ld  t6,248(sp)
800001e0:   10810113          addi    sp,sp,264
800001e4:   34011173          csrrw   sp,mscratch,sp
800001e8:   30200073          mret
```

图 4.2: mret

4.2 IF、ID、MEM 阶段异常选择

- 如果在某一拍，三个阶段同时检测到异常，那随着流水线继续运行，必然是优先将 MEM 阶段发生的异常处理掉。这也符合逻辑，也就是优先处理更早发生异常的指令，这样后面抛出的异常可能也就被一并处理了。
- 如果是一条指令在流水线中 IF、ID、MEM 阶段都抛出了异常，那应当优先选择更早抛异常的阶段，即 IF 阶段，这也依然是优先解决更早发生的异常这套逻辑。

4.3 CSR 读写与 stall

非立即数操作的 csrrw、csrrs、csrrc 均会使用 rs1 寄存器和将 csr 存入 rd 寄存器，立即数操作也会对 rd 做赋值，因此会引入数据冒险，是可能发生 stall 的。当然，由于不涉及内存读写和控制冒险，因此这些可能产生的 stall 全部都可以通过 forwarding 机制来解决。只需额外增加 ForwardingUnit 中对 csr 的前递判断即可。

```
if (csr_addr_id == csr_addr_exe && we_csr_exe == 1)
    csr_val_idexe = csr_val_exe;
else if (csr_addr_id == csr_addr_mem && we_csr_mem == 1)
    csr_val_idexe = csr_val_mem;
else if (csr_addr_id == csr_addr_wb && we_csr_wb == 1)
    csr_val_idexe = csr_val_wb;
else csr_val_idexe = csr_val_id;
```