

Computer Architecture

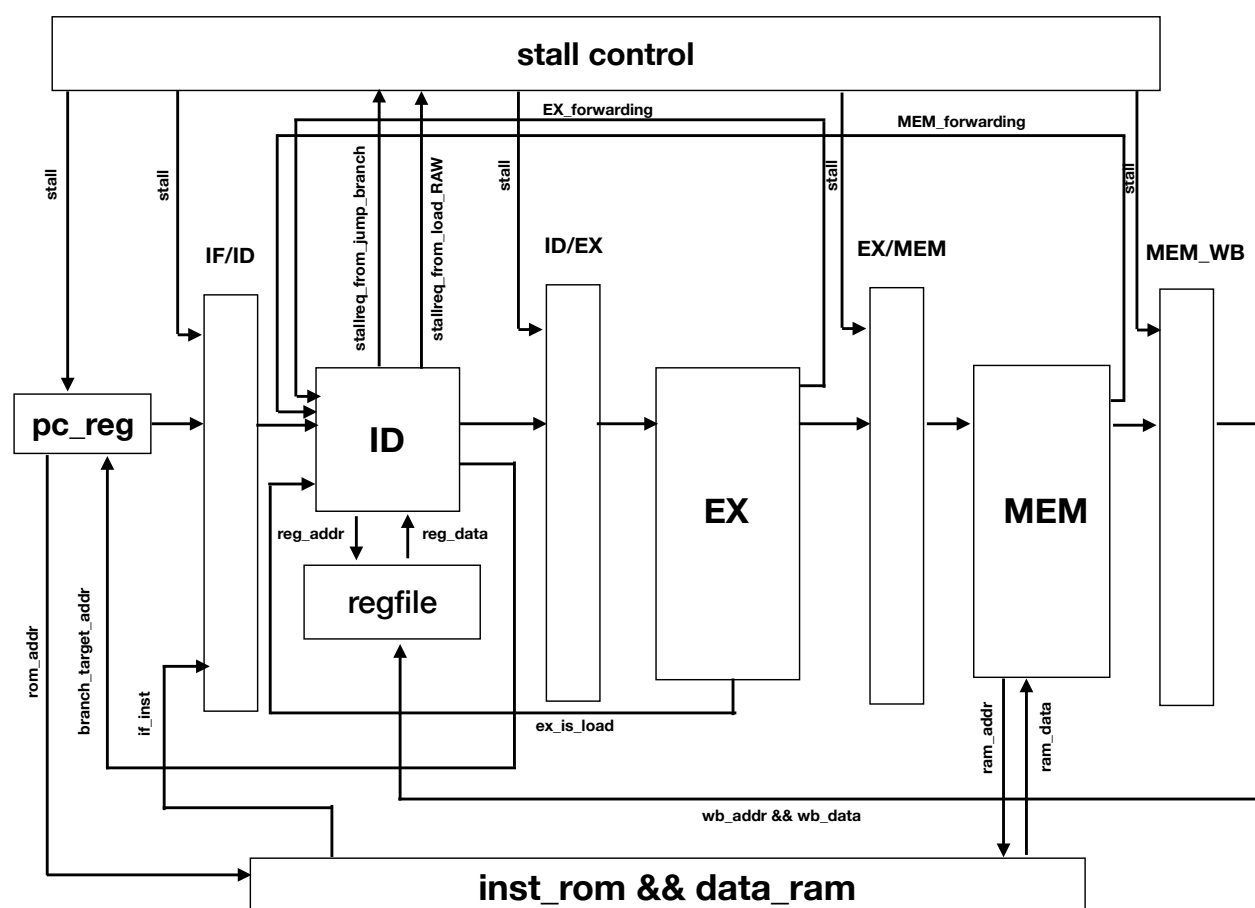
A CPU of RISC-V

刘予希

my github : <https://github.com/louisja1/CPU-RISCV.git>

2018.1.12

1 架构



我的CPU架构是五级流水，如上图所示有多个模块，其中为了实现self modifying code，将inst_rom与data_ram合并在了一起，pc_reg，id，ex，mem，wb等阶段之间信号的传导与普通的五级流水相似，其中wb阶段并没有具体的一个模块，而是直接传递信号给regfile实现写入，模块之间的四个buffer起到了传导作用以及接受时钟信号来控制数据的下传，而stall control通过接受ID阶段不同的stall request来安排相应的stall状态，并且传递给四个buffer以及pc_reg模块来实现必要的stall操作。

2 细节简述

2.1 pc_reg

pc_reg为传统的指令存储器，起到了IF模块的作用，会接收来自ID阶段可能的分支指令造成的修改，以及stall指令。每个时钟上沿，只会修改一次。

2.2 id

id即为ID模块，接收来自IF/ID buffer的信号，对不同指令格式预处理出相应的funct7, rs1, rs2, funct3, rd, 5种不同类型的立即数，decode阶段是always (*), 将指令通过操作码等等分类为逻辑指令，算数指令，分支指令，特殊指令等等。

对于reg1, reg2均有对应的使读信号: reg1_o, reg2_o它们的类型有reg_imm, reg_read, reg_zero, reg_pc四种信号分别表示，读立即数，regfile使读信号，使0，使对应pc的值，以适应不同指令操作的读取，同时也有关于目标寄存器的信息以及使写信号，会随着流水下传直到WB阶段回写到regfile。以上实现了id模块与regfile模块的交流。

jal以及jalr指令，通过计算得到新的pc地址通过branch_target_address_o传递给pc_reg模块，以及link_addr_o下传目标寄存器需要的pc_plus_4，通过stallreq_from_jump_branch请求stall一拍。

auipc以及lui指令，通过reg1_read_o的reg_zero以及reg_pc类型来得到相应的数据，并把指令转换成add指令或者or指令下传来实现后续步骤。

branch指令，在id阶段比较reg1与reg2的值，计算出target，通过branch_target_address_o传递给pc_reg模块，通过stallreq_from_jump_branch请求stall一拍。

load指令，通过reg_read以及reg_imm，得到两个寄存器的值。

store指令，通过reg_read，得到两个寄存器的值。

op_imm指令，分辨出整数-寄存器类型指令，通过reg_read以及reg_imm，得到两个寄存器的值。

op指令，分辨出寄存器-寄存器类型指令，通过reg_read以及reg_imm，得到两个寄存器的值。

rs1以及rs2小模块根据使读信号的不同实现reg1, reg2的取值。（读regfile在同一个周期解决）

stall for load小模块，由EX阶段传来的ex_is_load表示前面一条指令是load指令，由于load指令写出的寄存器在MEM阶段才能获得，如果立即需要使用，会造成RAW，所以这里的一个特别判断，通过stall_from_load信号请求stall一拍。

2.2 regfile

读写操作，其中写操作每一个时钟上沿执行一次。

2.3 ex

算术指令，逻辑指令，位移指令，分支指令执行出相应的结果，写入wdata_o随流水线下传。load，store指令计算出相应的memory地址。

2.4 mem

load，store指令的预处理，根据word，halfword，byte的相应格式，以及具体的memory地址，计算具体取读哪里或者具体写哪里。

2.5 if_id id_ex ex_mem mem_wb

各阶段之间的buffer，每个时钟上沿传递信息，并在必要时stall。

2.6 ctrl

stallreq_from_branch_stop: if_id阶段在下个时钟周期不进行值传递，那么pc_reg会在下个时钟周期得到正确的地址，而错误的地址被冲掉了。

stallreq_from_load: 停滞pc_reg, if, id阶段。

2.7 inst_rom_data_ram

inst_rom的指令数据读入，注意到了小端序或者大端序的问题。

data_ram会把指令集也放入相应的内存位置以实现SMC，内存写操作每个时钟上沿执行一次。

3 forwarding && SMC

3.1 forwarding

解决RAW类型的data hazard，所以有EX到ID，MEM到ID的两条线。

至于load指令以接踵而来的寄存器使用造成的data hazard，通过stall一拍解决。

3.2 SMC

通过合并了inst_rom以及data_ram以实现。

4 正确性测试

4.0 说明

在data文件夹中有各个.s测试数据，及对应的vcd波形图，以下截图中波形图只有一小部分，完整的在data中对应的vcd。

截图中包含有终端输出，我在regfile写入、memory的load、store三处加了相关的display信息，以粗略地验证正确性。

使用了arith.s(from 吴章昊)来进行算术运算、位移运算、逻辑运算的正确性验证

使用了b_j.s(from 吴章昊)来进行跳转、分支的正确性验证

使用了ld_st.s(from 吴章昊)来进行load、store的正确性验证

使用了fz.s(from 范舟)来进行综合正确性测试

使用了helloworld.s(from 刘啸远、助教)来进行SMC正确性测试

4.1 \data\wzh_test\arith.s

```

.org 0x8
.global _start

_start:
# ----- Test for arithmetic ops -----
    lui x1, 0x800000    # x1          = 0x80000000
    ori x1, x1, 0x010    # x1 = x1 | 0x010 = 0x80000010

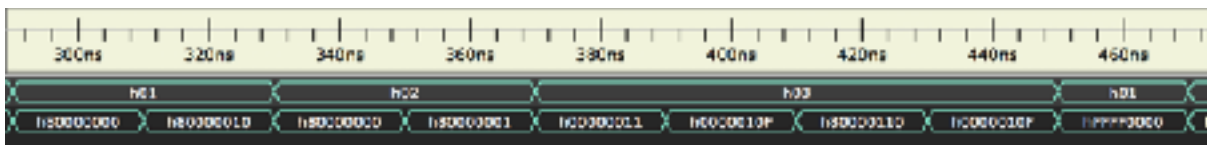
    lui x2, 0x800000    # x2          = 0x80000000
    ori x2, x2, 0x001    # x2 = x2 | 0x001 = 0x80000001

    add x3, x2, x1      # x3          = 0x00000011
    addi x3, x3, 0x0fe   # x3          = 0x0000010f
    add x3, x3, x2      # x3          = 0x80000110

    sub x3, x3, x2      # x3          = 0x0000010f

# ----- Test for cmp ops -----
    lui x1, 0xffff0     # x1          = 0xffff8000
    slt x2, x1, x0      # x2          = 1      notice: signed
    sltu x2, x1, x0     # x2          = 0      notice: unsigned
    lui x1, 0x000001     # x1          = 0x00000100
    slti x3, x1, -0x800  # x3          = 0      notice: signed
    sltiu x3, x1, -0x800 # x3          = 1      notice: signed extend and unsigned comparison

```



```

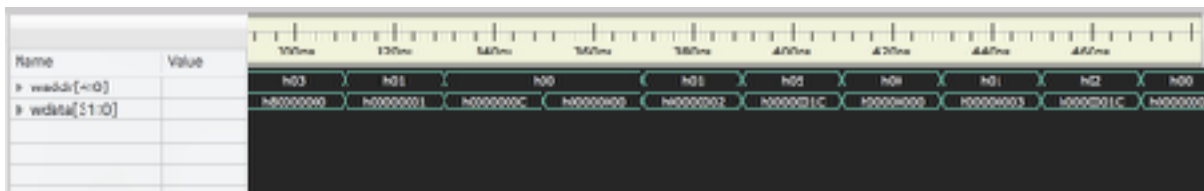
regfile : regs[ 1] <= 80000000
regfile : regs[ 1] <= 80000010
regfile : regs[ 2] <= 80000000
regfile : regs[ 2] <= 80000001
regfile : regs[ 3] <= 00000011
regfile : regs[ 3] <= 0000010f
regfile : regs[ 3] <= 80000110
regfile : regs[ 3] <= 0000010f
regfile : regs[ 1] <= ffff0000
regfile : regs[ 2] <= 00000001
regfile : regs[ 2] <= 00000000
regfile : regs[ 1] <= 00001000
regfile : regs[ 3] <= 00000000
regfile : regs[ 3] <= 00000001

```

4.1 \data\wzh_test\b_j.s

数据略，可在文件夹中找到，为分支跳转指令测试。

```
regfile : regs[ 3] <= 80000000
regfile : regs[ 1] <= 00000001
regfile : regs[ 1] <= 00000002
regfile : regs[ 5] <= 0000001c
regfile : regs[ 1] <= 00000003
regfile : regs[ 2] <= 0000001c
regfile : regs[ 1] <= 00000004
regfile : regs[ 1] <= 00000005
regfile : regs[ 1] <= 00000006
regfile : regs[ 1] <= 00000007
regfile : regs[ 1] <= 00000008
regfile : regs[ 1] <= 00000009
regfile : regs[ 1] <= 00000010
regfile : regs[ 1] <= 00000011
regfile : regs[ 1] <= 00000012
regfile : regs[ 3] <= 00000014
regfile : regs[ 1] <= 00000013
regfile : regs[ 1] <= 00000014
regfile : regs[ 3] <= 0000000a
regfile : regs[ 1] <= 0000000a
regfile : regs[ 1] <= 00000000
regfile : regs[ 1] <= ffffffff6
```



4.3 \data\wzh_test\ld_st.s

```

lui x3,0x0000ff
srl x3, x3, 12
sb x3, 3(x0)      # [0x3] = 0xff
srl x3, x3, 8
sb x3, 2(x0)      # [0x2] = 0xff
lui x3,0x0000dd
srl x3, x3, 12
sb x3, 1(x0)      # [0x1] = 0xdd
srl x3, x3, 8
sb x3, 0(x0)      # [0x0] = 0xdd
lb x1, 3(x0)      # x1 = 0xffffffff
lbu x1, 2(x0)     # x1 = 0x000000ee
lw x1, 0(x0)      # x1 = 0xffffedcc
nop

lui x3,0xaabb
srl x3, x3, 12
sb x3, 4(x0)      # [0x4] = 0xab, [0x5] = 0xaa
lbu x1, 4(x0)     # x1 = 0x0000aabb
lb x1, 4(x0)      # x1 = 0xffffaabb

lui x3,0x0099
srl x3, x3, 12
sb x3, 6(x0)      # [0x6] = 0x99, [0x7] = 0x99
lb x1, 0(x0)      # x1 = 0xffff9999
lbu x1, 6(x0)     # x1 = 0x00009999

lui x3,0x4455
srl x3, x3, 12
sb x3, 0x10
lui x2, 0x6677
srl x2, x2, 12
or x3, x2, x1     # x3 = 0x44556677
sw x3, 8(x0)      # [0x8] = 0x77, [0x9] = 0x66, [0xa] = 0x55, [0xb] = 0x44
lw x1, 8(x0)      # x1 = 0x44556677

```

```

regfile : regfile 31 <= 0xffff0000
Store memory[00000003] = 0xffff0000
regfile : regfile 31 <= 0xffff0000
Store memory[00000002] = 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
Store memory[00000001] = 0xffff0000
regfile : regfile 31 <= 0xffff0000
Store memory[00000000] = 0xffff0000
Load memory[00000003] = 0xffff0000
Load memory[00000003] = 0xffff0000
Load memory[00000002] = 0xffff0000
Load memory[00000002] = 0xffff0000
regfile : regfile 31 <= 0xffff0000
Load memory[00000000] = 0xffff0000
Load memory[00000000] = 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
Store memory[00000004] = 0xffff0000
regfile : regfile 31 <= 0xffff0000
Load memory[00000004] = 0xffff0000
Load memory[00000004] = 0xffff0000
Load memory[00000004] = 0xffff0000
Load memory[00000004] = 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
Store memory[00000006] = 0xffff0000
Load memory[00000006] = 0xffff0000
Load memory[00000006] = 0xffff0000
Load memory[00000006] = 0xffff0000
Load memory[00000006] = 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
regfile : regfile 31 <= 0xffff0000
Store memory[00000008] = 0xffff0000
regfile : regfile 31 <= 0xffff0000
Load memory[00000008] = 0xffff0000
Load memory[00000008] = 0xffff0000
regfile : regfile 31 <= 0xffff0000

```

4.4 \data\fb_test\fb.s

```

_start:
    ori x1, x0, 0x210 # x1 = 0x210
    ori x2, x1, 0x021 # x2 = 0x231
    slli x3, x2, 1 # x3 = 0x12001100013 = 0x462
    andi x4, x3, 0x548 # x4 = 0x01000110000 = 0x160
    ori x5, x0, 0x01a # x5 = 0x011010001316 = 0x58a
    ori x7, x0, 22 # x7 = 22 = 0x16
    ori x5, x5, x7 # x5 = 0xa1800000
    frr x7, x0, x0 # x7 = x0 = 0x14
    frr x0, x5, x7 # x0 = 0x7fff7a28
    ori x3, x0, 0x723 # x3 = 0x011000100013 = 0x723
    xori x3, x3, x4 # x3 = 0x001101000011 = 0x343
    add x6, x5, x4 # x6 = 0x7a3
    slli x7, x6, 0x7a4 # x7 = 1
    slli x8, x6, 0x7a3 # x8 = 0
    slt x8, x6, x5 # x8 = 0
    slt x8, x5, x6 # x8 = 1
    xori x9, x6, x5 # x9 = 0x4fa
    lui x10, 0x45b27 # x10 = 0x45b27000
    zuxor x11, x21:43 # x11 = 0x21c43048
es_1:
    bge x10, x11, es_2 # jump to es_2 if x1 >= x10 = 0x45b
    ori x12, x0, 0x45b # x12 = 0x45b
    ori x13, x0, 0x45b # x13 = 0x45b
    rop
    rop
    rop
es_2:
    ori x12, x0, 0x5ef # x12 = 0x5ef
    xori x13, x0, 0x173 # x13 = 0x173
    # j es_1 # jump to es_1, which backs at infinite loop
    sb x11, 2(x13) # store 0x48 to mem:0x125
    lb x14, 2(x13) # x14 = 0x48
    sb x12, 1(x13) # store 0x5ef to mem:0x124
    lb x14, 1(x13) # x14 = 0x48ef

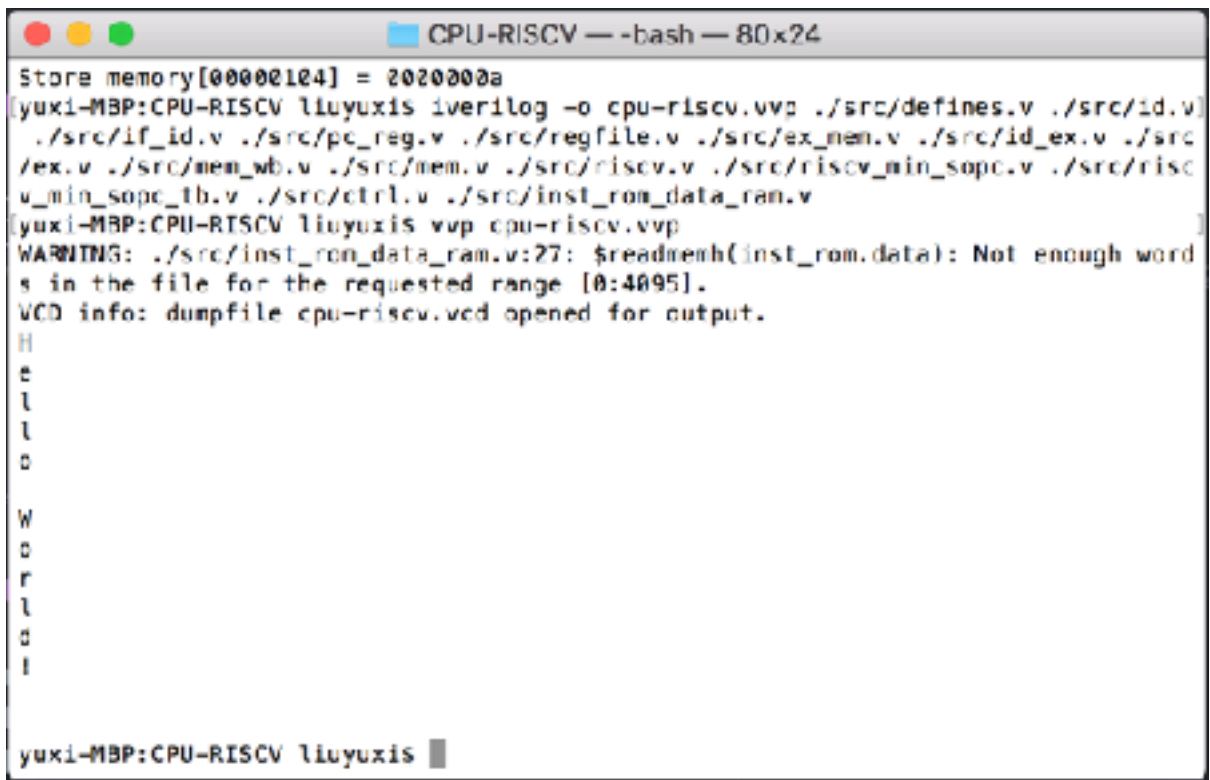
```

```

regfile : regs[ 1] <= 00000210
regfile : regs[ 2] <= 00000231
regfile : regs[ 3] <= 00000462
regfile : regs[ 4] <= 00000160
regfile : regs[ 5] <= 0000058a
regfile : regs[ 7] <= 00000016
regfile : regs[ 5] <= a2800000
regfile : regs[ 7] <= 00000014
regfile : regs[ 6] <= fffffa28
regfile : regs[ 5] <= 00000723
regfile : regs[ 5] <= 00000343
regfile : regs[ 6] <= 000007a3
regfile : regs[ 7] <= 00000001
regfile : regs[ 8] <= 00000000
regfile : regs[ 8] <= 00000000
regfile : regs[ 8] <= 00000001
regfile : regs[ 9] <= 000004fa
regfile : regs[10] <= 45b27000
regfile : regs[11] <= 21c43048
regfile : regs[12] <= 000005ef
regfile : regs[13] <= 00000124
Store memory[00000125] = 00000048
Load memory[00000125] = xxxx48xx
Load memory[00000125] = xxxx48xx
regfile : regs[14] <= 00000048
Store memory[00000124] = 000005ef
Load memory[00000124] = xxxx48ef
Load memory[00000124] = xxxx48ef
regfile : regs[14] <= 0000048ef
regfile : regs[15] <= 0000048ef
Store memory[00000126] = 03430000
Load memory[00000124] = 034348ef
Load memory[00000124] = 034348ef
regfile : regs[15] <= 034348ef
regfile : regs[17] <= 034340f0
Store memory[00000128] = 21c43048
Load memory[00000128] = 21c43048
Load memory[00000128] = 21c43048
regfile : regs[16] <= 21c43048

```

4.5 \data\helloworld.data



```
Store memory[00000104] = 2020000a
[yuxi-MBP:CPU-RISCV liuyuxis iverilog -o cpu-riscv.vvp ./src/defines.v ./src/id.v]
./src/if_id.v ./src/pc_reg.v ./src/regfile.v ./src/ex_mem.v ./src/id_ex.v ./src
./ex.v ./src/mem_wb.v ./src/mem.v ./src/riscv.v ./src/riscv_min_sopc.v ./src/risc
v_min_sopc_tb.v ./src/ctrl.v ./src/inst_rom_data_ram.v
[yuxi-MBP:CPU-RISCV liuyuxis vvp cpu-riscv.vvp ]
WARNING: ./src/inst_rom_data_ram.v:27: $readmemh(inst_rom.data): Not enough word
s in the file for the requested range [0:4095].
VCD info: dumpfile cpu-riscv.vcd opened for output.
H
e
l
l
o

W
o
r
l
d
!

yuxi-MBP:CPU-RISCV liuyuxis █
```

(注释了load、store、regfile处的\$display)

5 感谢 & 参考资料

感谢在写CPU过程中一起讨论、互相帮助的各位同学。

参考资料 《自己动手写CPU》