

Lab1

假设设计图中选择器的输入从上到下依次增大

假设控制信号0代表禁止，1代表允许

描述时直列出起作用的控制信号（蓝笔），其余保持默认值0。

用首字母缩写表示模块

自己命了些名字，在图中给出

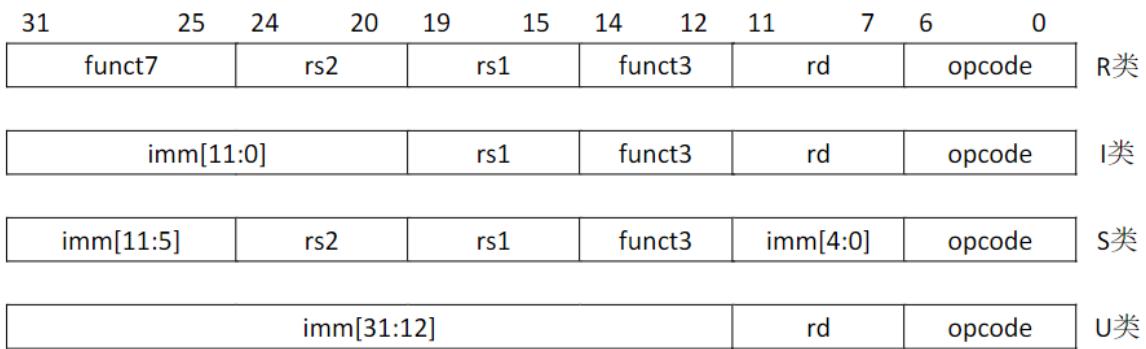
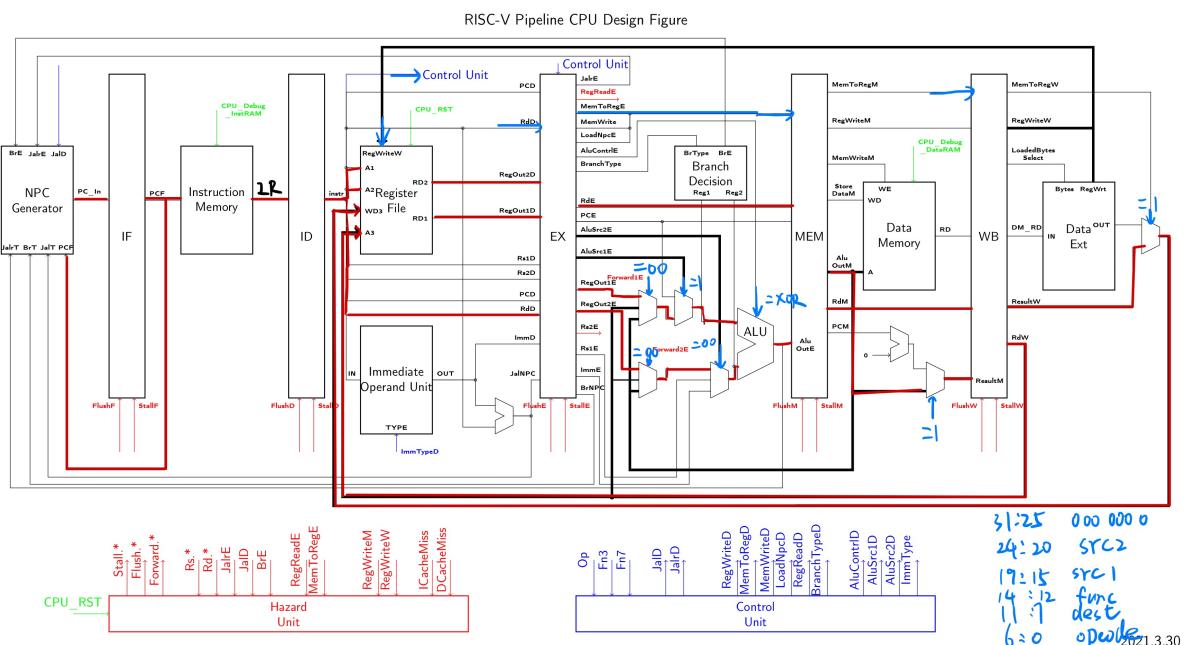


图 2.2: RISC-V 基本指令格式

1. 描述执行一条 XOR 指令的过程（数据通路、控制信号等）。

31	25	24	20	19	15	14	12	11	7	6	0
funct7	rs2		rs1		funct3		rd		opcode		R类



- IF
 - IR = IM[PCF]
 - PCF直传
- ID

- $RdD = \text{instr}[11:7]$
- $A1 = \text{instr}[24:20]$
- $A2 = \text{instr}[19:15]$
- $\text{RegOut2D} = RD2 = RF[A1]$
- $\text{RegOut1D} = RD1 = RF[A2]$

控制信号

- RdD 直传
- Control Unit = instr的某些位

- EX

- RdE 直传
- $\text{AluOutE} = \text{RegOut1E} + \text{RegOut2E}$

控制信号：

- $\text{Forward1E} = 00$
- $\text{Forward2E} = 00$
- $\text{AluSrc1E} = 1$
- $\text{AluSrc2E} = 00$
- $\text{ALuControlE} = \text{XOR}$
- MemToRegE 直传

- MEM

- $\text{ResultM} = \text{AluOutM}$
- RdM 直传

控制信号：

- (多路选择器) $\text{SelM} = 1$
- MemToRegM 直传

- WB

- $WD3 = \text{ResultW}$
- $A3 = RdW$

控制信号：

- $\text{MemToRegW} = 1$
- $\text{RegWriteW} = 1$

2. 描述执行一条 BEQ 指令的过程·(数据通路、控制信号等)。

31	30	25	24	20	19	15	14	12	11	8	7	6	0
imm[12]	imm[10:5]	rs2	rs1		funct3	imm[4:1]	imm[11]			opcode			

1

6

5

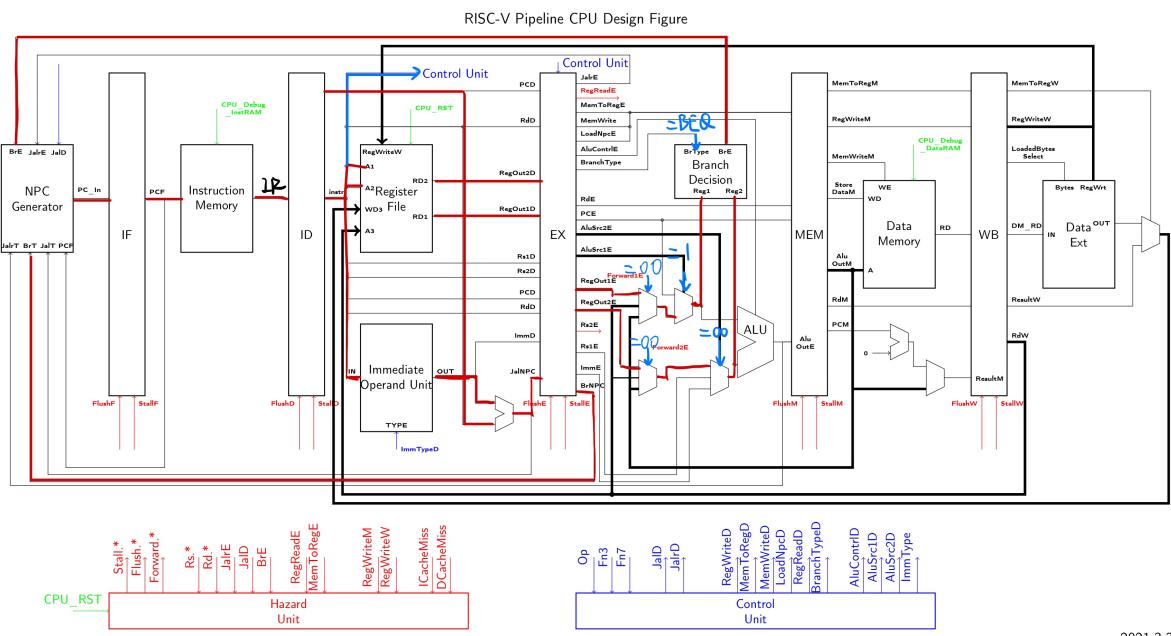
5

3

4

1

7



2021.3.30

- IF
 - IR = IM[PCF]

- ID
 - IN = instr[31] || instr[7] || instr[30:25] || instr[4:1] (其中||表示拼接)
 - OUT = IOU[IN] (分别在低位和高位补0, 扩展成32位)
 - JalNPC = OUT + PCD
 - A1 = instr[24:20]
 - A2 = instr[19:15]
 - RegOut2D = RD2 = RF[A1]
 - RegOut1D = RD1 = RF[A2]

控制信号:

- Control Unit = instr的某些位

- EX
 - Reg1 = RegOut1E
 - Reg2 = RegOut2E
 - BrE = BD[Reg1, Reg2]
 - BrT = BrNPC

控制信号:

- Forward1E = 00
- Forward2E = 00
- AluSrc1E = 1
- AluSrc1E = 00
- BrType = BEQ

3. 描述执行一条 LHU 指令的过程 (数据通路、控制信号等)。

31	20 19	15	14	12 11	7	6	0
imm[11:0]	rs1	funct3		rd		opcode	
12	5	3		5		7	

偏移量[11:0]

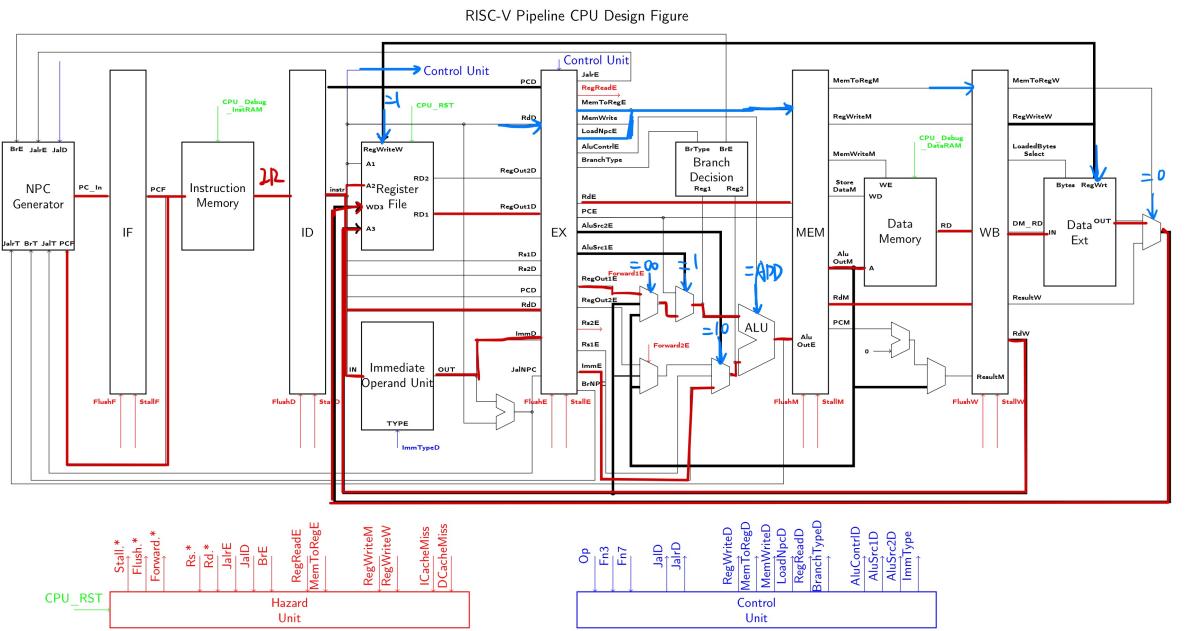
基址

宽度

dest

LOAD

LHU指令存储器中读取一个16位数值，然后将其进行零扩展到32位，再保存到rd中



2021.3.30

- IF
 - IR = IM[PCF]
 - PCF直传

- ID
 - A2 = instr[19:15]
 - RegOut1D = RD1 = RF[A2]
 - RdD = instr[11:7]
 - IN = instr[31:20]
 - ImmD = OUT = IOU[IN]

控制信号：

- RdD直传
- Control = instr的某些位

- EX
 - RdE直传
 - AluOutE = RegOut1E + ImmE

控制信号：

- Forward1E = 0
- AluSrc1E = 1
- AluSrc2E = 10
- AluControlE = ADD
- MemToRegE直传
- LoadNpcE直传

- MEM
 - RdM直传
 - A = AluOutM
 - RD = DM[A]

控制信号：

- MemToRegM直传

- WB
 - DM_RD = IN

- WD3 = OUT = DE[IN]
- A3 = RdW

控制信号：

- MemToRegW = 0
- RegWriteW = 1

4. 如果要实现 CSR 指令 (csrrw, csrrs, csrrc, csrrwi, csrrsi, csrrci) , 设计图中还需要增加什么部件和数据通路? 给出详细说明。

- CSR指令介绍

- I类

- 原子性读-修改-写控制和状态寄存器的指令

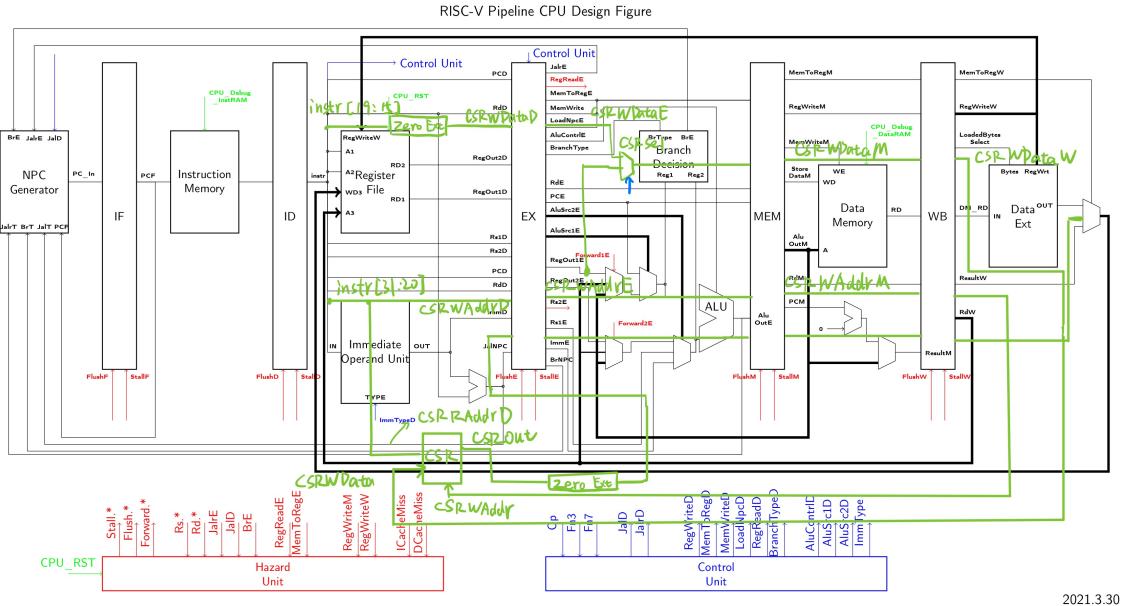
31	20	19	15	14	12	11	7	6	0
csr	rs1		funct3		rd		opcode		
12	5		3		5		7		

◦	source/dest	source	CSRRW	dest	SYSTEM
	source/dest	source	CSRRS	dest	SYSTEM
	source/dest	source	CSRRC	dest	SYSTEM
	source/dest	zimm[4:0]	CSRRWI	dest	SYSTEM
	source/dest	zimm[4:0]	CSRRSI	dest	SYSTEM
	source/dest	zimm[4:0]	CSRRCI	dest	SYSTEM

- CSRRW (Atomic Read/Write CSR) 指令原子性的交换CSR和整数寄存器中的值。CSRRW指令读取在CSR中的旧值，将其零扩展到XLEN位，然后写入整数寄存器rd中。rs1寄存器中的值将被写入CSR中。
- CSRRS (Atomic Read and Set Bit in CSR) 指令读取CSR的值，将其零扩展到XLEN位，然后写入整数寄存器rd中。整数寄存器rs1中的值指明了哪些CSR中的位被置为1。rs1中的任何为1的位，将导致CSR中对应位被置为1，如果CSR中该位是可以写的话。CSR中的其他位不受影响（虽然当CSR被写入时可能有些副作用）
- CSRRC (Atomic Read and Clear Bit in CSR) 指令读取CSR的值，将其零扩展到XLEN位，然后写入整数寄存器rd中。整数寄存器rs1中的值指明了哪些CSR中的位被置为0。rs1中的任何为1的位，将导致CSR中对应位被置为0，如果CSR中该位是可以写的话。CSR中的其他位不受影响
- 对于CSRRS指令和CSRRC指令，如果rs1 = x0 (寄存器x0硬编码为0，即读出来总是0，写进去总是被丢弃)，那么指令将根本不会去写CSR，因此应该不会产生任何由于写CSR产生的副作用（译者注：某些特殊CSR检测是否有人尝试写入，一旦有写入，则执行某些动作。这和写入什么值没什么关系）。注意如果rs1寄存器包含的值是0，而不是rs1 = x0，那么将会把一个不修改的值写回CSR（译者注：这时会有一个写CSR的操作，但是写入的值就是旧值，因此可能会产生副作用）。
- CSRRWI指令、CSRRSI指令、CSRRCI指令分别于CSRRW指令、CSRRS指令、CSRRC指令相似，除了它们是使用一个处于rs1字段的、零扩展到XLEN位的5位立即数 (zimm[4:0]) 而不是使用rs1整数寄存器的值。如果zimm[4:0]字段是零，那么这些指令将不会写CSR，因此应该不会产生任何由于写CSR产生的副作用。

- 说明

-



- 添加的部件

- ID段: CSR寄存器、对CSR寄存器读出的值进行零扩展、对instr[19:15]进行零扩展
- ID/EX段、EX/MEM段、MEM/WB段都各自增加一个段间寄存器，用以存放CSR写回数据（可能来自instr[19:15]的零扩展，也可能来自rs1寄存器中的值）、CSR写回地址(CSRWAddr) 和从CSR读出并零扩展的值 (ZeroExt[CSROut])
- ID/EX段: CSR部件 (寄存器)
- Ex段: CSRSel二路选择器

- 数据通路

- 将instr[19:15]进行零扩展，放入ID/EX的段间寄存器
- CSRSel二路选择器，用来选择写回CSR的数据，要么来自rs1寄存器中的值 (instr[19:15]作为寄存器地址)，要么来自instr[19:15]的扩展。选择器的输出分别传递到EX/MEM段、MEM/WB段的段间寄存器，最后在WB段写回CSR
- instr[31:20]作为CSR的输入地址，读出CSR寄存器的值，并进行零扩展，再分别传递到ID/EX段、EX/MEM段、MEM/WB段的段间寄存器，最后接到WB段的多路选择器上，从而写回rd对应的寄存器
- instr[31:20]作为CSR的写回地址，分别传递到ID/EX段、EX/MEM段、MEM/WB段的段间寄存器，最后接到CSR

- 控制信号:

- 控制CSRSel二路选择器的信号

5. Verilog 如何实现立即数的扩展?

- 不同类型的指令采用不同的扩展方式，下面举例说明

- I-type

- ADDI
- ```
assign Imm[31:0] = {20{instr[31]}, instr[31:20]}
```

- J-type

- JAL
- ```
assign Imm[31:0] = {12{instr[31]}, instr[19:12], instr[20],  
instr[30:21], 1'b0}
```

- B-type

- BEQ, BNE
- ```
assign Imm[31:0] = {20{instr[31]}, instr[7], Inst[30:21], instr[11:8],
1'b0}
```

- S-type
  - `assign Imm[31:0] = {20{instr[31]}, instr[31:25], instr[11:7]}`
- U-type
  - LUI
  - `assign Imm[31:0] = {instr[31:12], 12'b0}`

## 6. 如何实现 Data Memory 的非字对齐的 Load 和 Store?

---

- Load: 分两次读取即可
  - 先读取 `{addr[32:2], 2'b0}`, 即低位字节所在地址, 然后将低位字节写入寄存器
  - 再增加addr, 得到高位字节所在地址, 然后将高位字节写入寄存器
- Store: 同上分两次写入寄存器即可

## 7. ALU 模块中, 默认 wire 变量是有符号数还是无符号数?

---

- 无符号数

## 8. 简述BranchE信号的作用。

---

- 用来指示Branch类型的指令是否跳转
- Branch Decision模块会根据传入BrType和Reg1、Reg2的值判断当前Br指令是否成功跳转, 并输出对应的控制信号BranchE: 如果跳转成功, PC变为跳转后的地址; 如果跳转失败, 则PC为  $PC_{(old)} + 4$

## 9. NPC Generator 中对于不同跳转 target 的选择有没有优先级?

---

- Br和Jalr的优先级最高, Jal的优先级第二高, PC+4的优先级最低
- Br和Jalr在EX段计算并返回给NPC Generator信号, 而Jal在ID段就计算并返回给NPC Generator信号。显然后计算的优先级更高, 因为如果Br或Jalr与Jal的信号同时到达, 则Br或Jalr指令是Jal指令的前一条指令, 显然靠前的指令优先级高
- 显然只有当Br、Jalr、Jal信号都无效时, PC+4才会生效, 即顺序执行

## 10. Harzard 模块中, 有哪几类冲突需要插入气泡, 分别使流水线停顿几个周期?

---

- 数据冲突-RAW相关: LW后跟使用寄存器的指令, 停顿一个周期

## 11. Harzard 模块中采用静态分支预测器, 即默认不跳转, 遇到 branch 指令时, 如何控制 flush 和 stall 信号?

---

- 对于Branch指令, 设PC为当前Branch
  - IF: 取指令
  - ID: 译码, 此时知道是Branch指令。由于默认不跳转, 故不需要Stall, 默认顺序执行Branch之后的指令 ( $PC + 4$ )。并计算Branch的目标地址
  - EX: 通过Branch Decision比较两个输入得到是否跳转。此时IF段为 $PC + 8$ 对应的指令。如果跳转信号有效, 说明 $PC + 4$ 和 $PC + 8$ 的指令不应执行, 故置FlushD和FlushE为有效, 清空对应的段

间寄存器，从而开始执行分支成功部分的第一条指令；如果跳转信号无效，说明跳转失败，则只需继续执行下去，Flush信号均置为无效即可

## 12. 0号寄存器值始终为0，是否会对forward的处理产生影响？

---

- 会
- 如果不加判断，转发过程中可能会将0号寄存器的值设置成非0。故转发前需要判断转发的目的寄存器是不是0号寄存器