No.

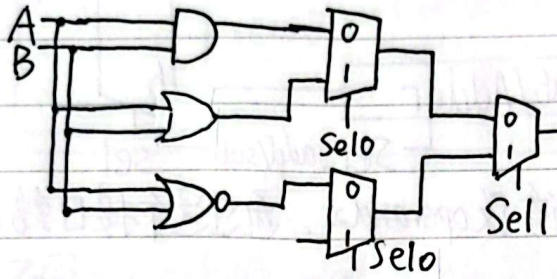
Date

CA Review: Datapath

Useful blocks:

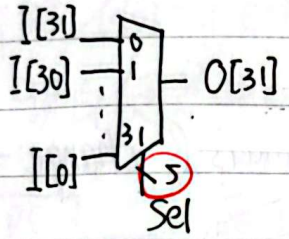
① ALU: arithmetic instructions 有: add sub slt sltu sll srl sra. 异或 或 与门 xor or and.
加减 store if less than shift.

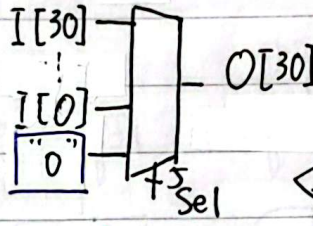
对与 xor or and:



用 SeloSel1 来控制输出哪个位运算.

对于 shift:





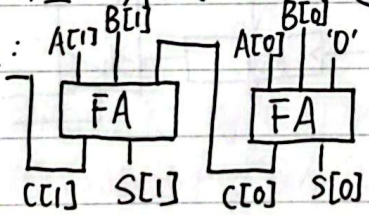
因为 SLL 就是 0 填... 用 0 填充 ↑

⇐ (left shift example).

5位 input, 因为 shift 命令就是移动 imm 低 5 位代表的十进制数.

类似可构造 srl sra 专用的 block (or: notation: C[i-1]).

对于加减: 输入 3 位: A[i] B[i] C[i] (进位) 输出 result[i] & 下一位进位。故:

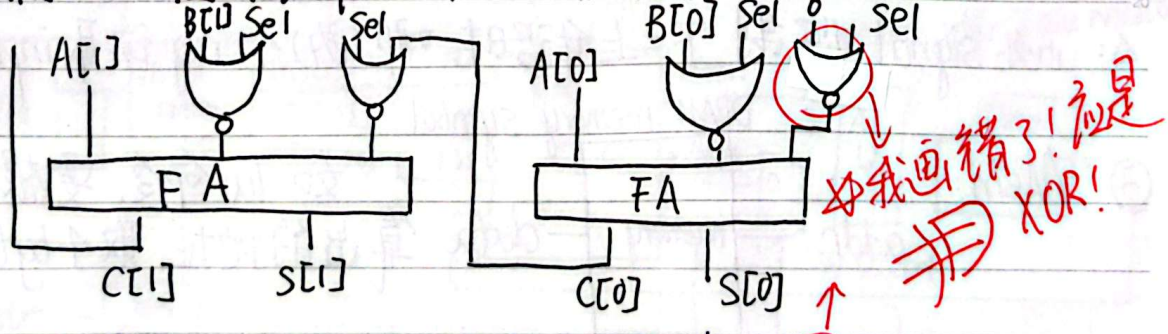


(ADDER).

而减法呢? $A - B = A + (-B) = A + \bar{B} + 1 \pmod{2^{N-1}}$

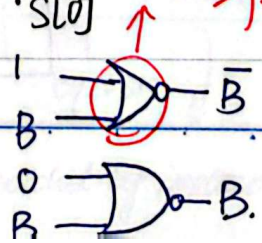
则 B[i] 输入时, 应有元件控制是否翻转.... XOR Gate! *

故:...



由 Sel 信号控制: 1. 则是减, B[i] 反转 (包括 0 位的进位: 0 => 1).
0. 则是加法

*:



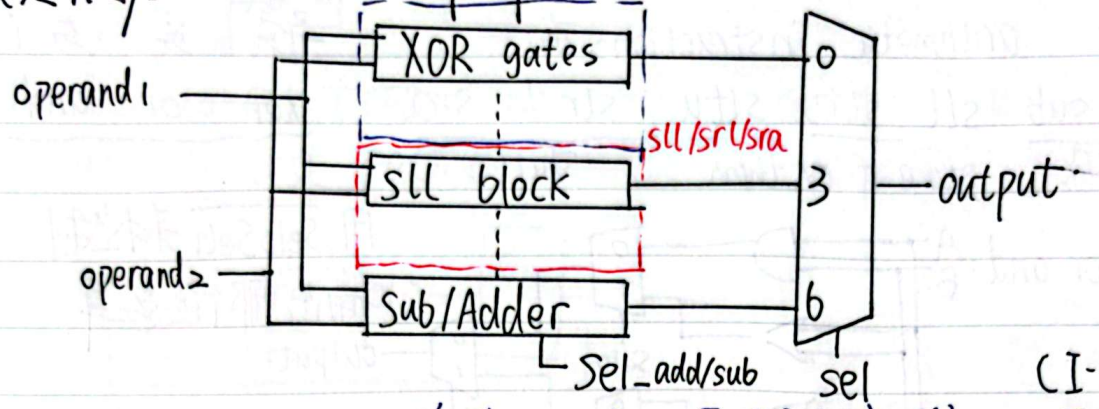
KOKUYO

No.

Date

至此，三种操作：add/sub，shift，and bit operation，都准备了block

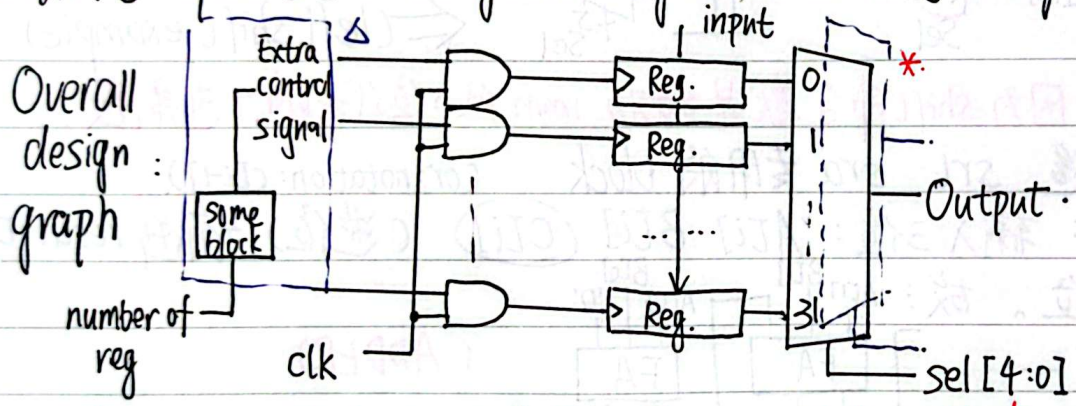
最终依据 sel 信号选择即可：xor/and/or



△：对于 immediate，只需修改 operand2，通过这个接口给 imm，而非 reg 中数即可

② Reg. ⇒ Register file.

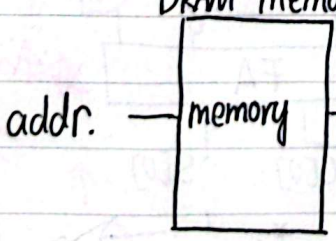
功能：provide data given register numbers & change the stored value



*：这个 32-multiplexer 可能用 5层 2-multiplexer 搭建；且一个32-plexer 输出一个 reg value，故此处应用两个 32-multiplexer！

△：此处 signal 控制下一上升沿时，哪(两)个 reg 读取 input 值

③ Mem:



如 lw 命令，要根据 offset 算出的地址，取4 byte 过来。

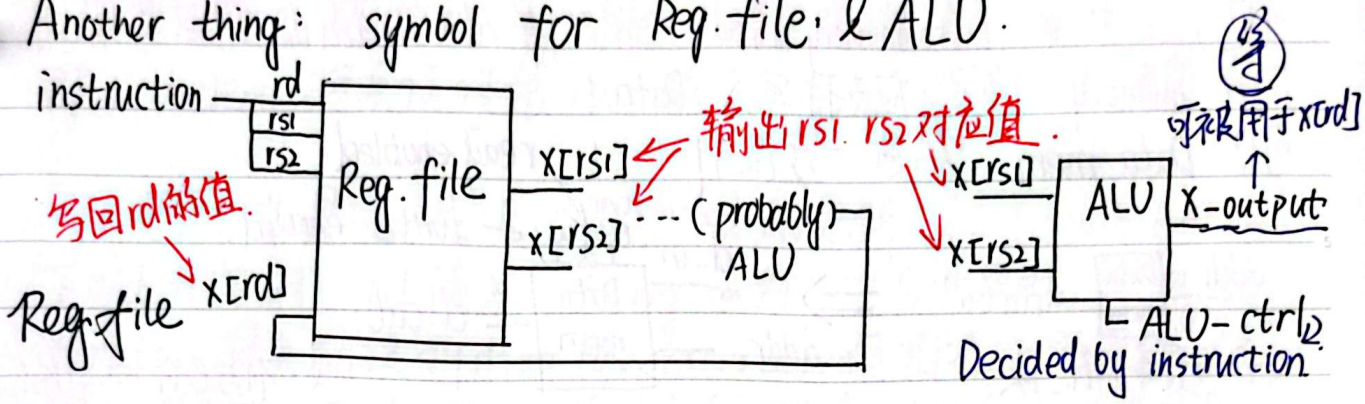
No.

Date

介绍了 useful blocks，接下来关键便是如何运用它们于指令！

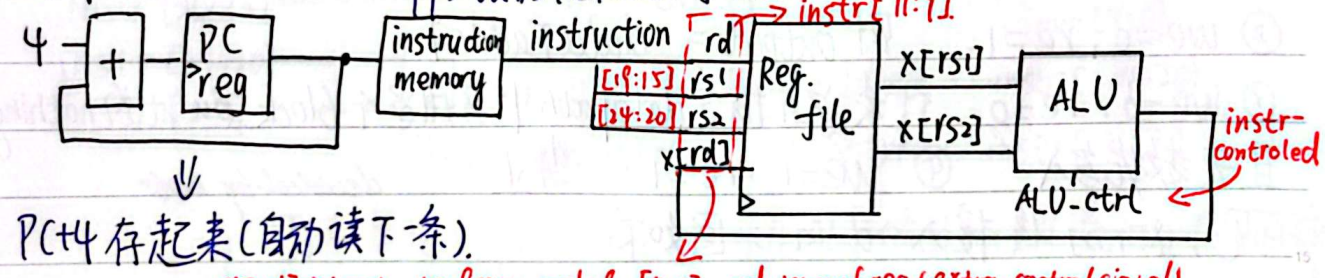
如：明显地：ALU 将会被 R-/I- type 指令使用

Another thing: symbol for Reg. file & ALU.



下面，将介绍不同 type 中的 datapath 搭建：

R-type 说白便是用 rs2 中值对 rs1 作操作最后给 rd 值



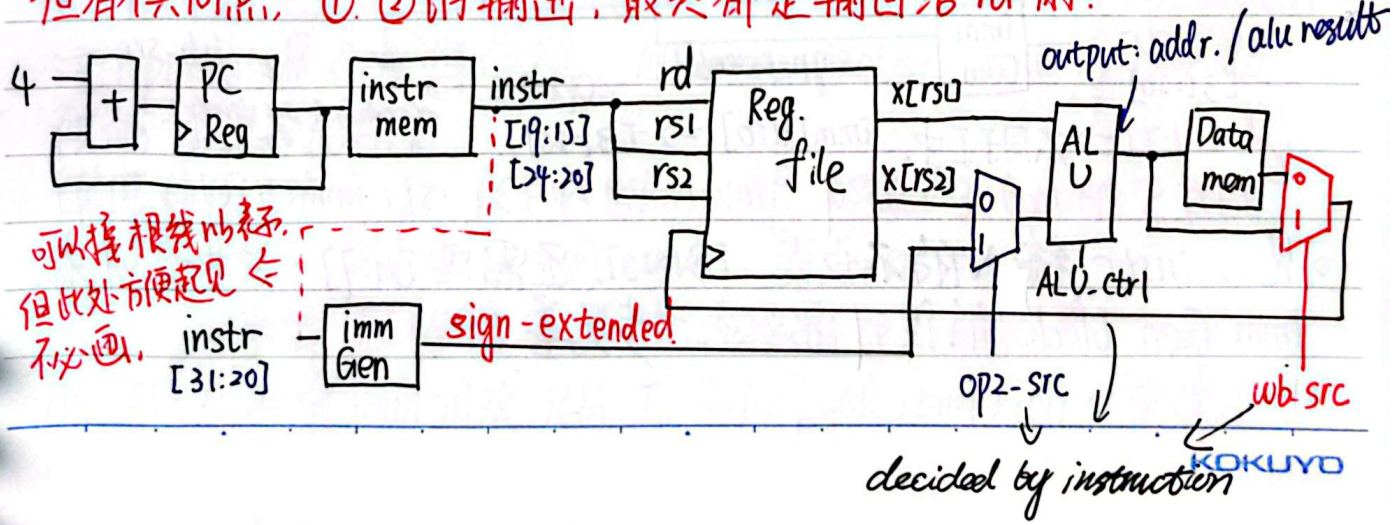
I-type: 回顾一下 I-type 与 R-type 的区别：

① I-type 中有许多操作与 R 中类似 (addi. slti, ori)，只过一 reg 一个 imm

② 除①之外，还有 l w/b/h，要从 Data memory 中拿数据

sign-extended

但有个共同点，①②的输出，最终都是输回给 rd 的！



KOKUYO

No.

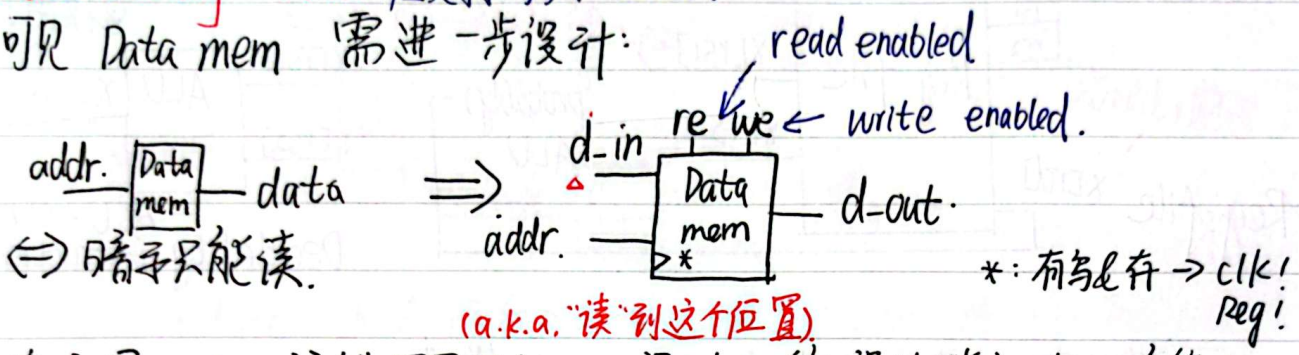
Date

S-type：在开始前，要厘清 S-type 在干什么：如 sw: rs2, imm(rs1):

store word at rs2 to memory addr. = (X[rs1] + imm).

可见，这里进 Data memory 的 addr. 依然是 ALU 的输出，但这里的 Data memory block 应支持写入 Data！

可见 Data mem 需进一步设计：



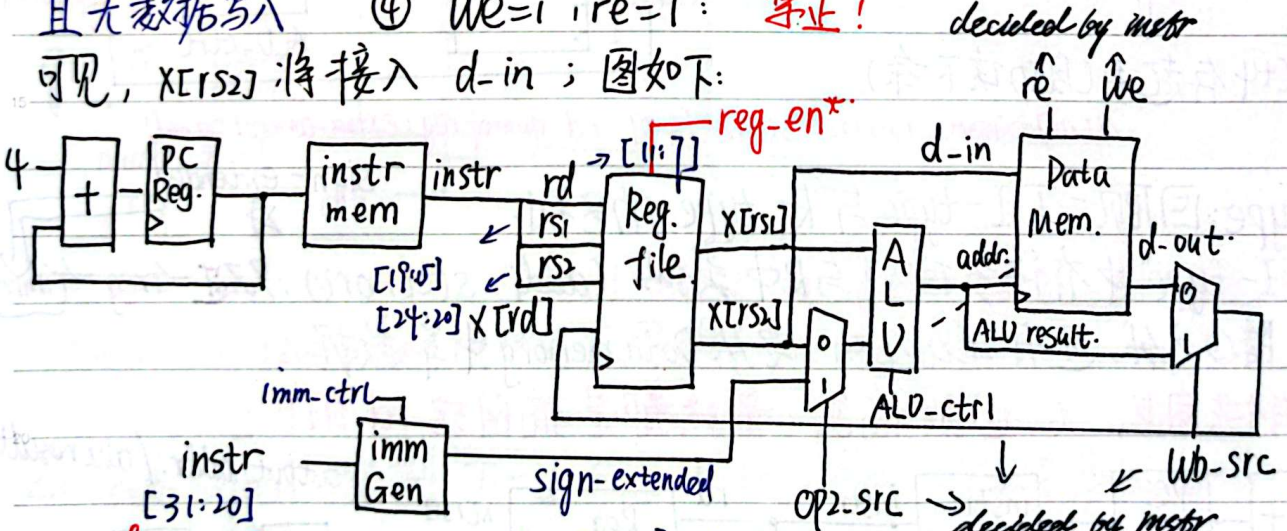
在这里 addr. 依然可用于定位，但 d-in 的设计指定了写入啥值；而 re. we 将控制此时 block 是读 or 写，i.e., 控制 block 行为：

① we=1, re=0, 则 addr. 写入 d-in 值., d-out 为 nothing

② we=0, re=1, 则 d-out = data[addr.]

③ we=0, re=0, 代表这个指令 datapath 中不用这个 block, d-out 为 nothing 且无数据写入 ④ we=1, re=1: 禁止！

可见，X[rs2] 将接入 d-in；图如下：



& [11:7] ⇒ 不同于 I 中, imm[11:0] ⇒ [31:20].

因为 S 的 machine code 中, imm[11:5] => [31:25]; imm[4:0] ⇒ [11:7]

可见，instr 输入的不仅是 [31:20], 还需要 [11:7]；这也意味着，imm Gen block 的设计需要进一步完善！△

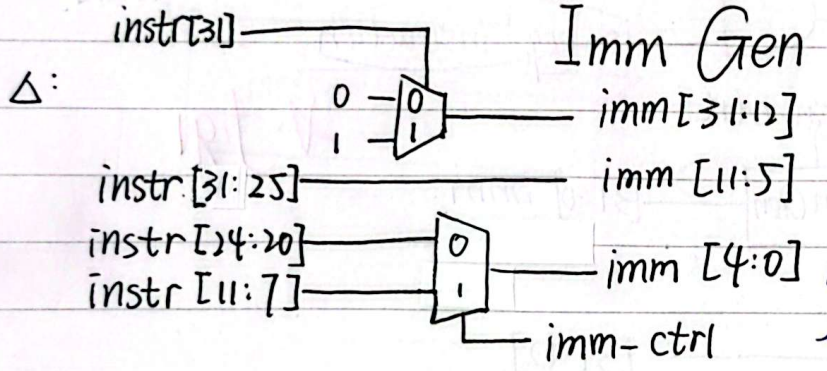
No.

Date

*：同时，reg file block 加入了额外的 control signal：reg-en

why？假设 sw 之后，d-out 为 nothing；虽然 S-type 无 rd，但 I[11:7] 处依然是5位，且传给 reg. file（但其实这5位是 imm[4:0]！）那么，nothing 将会误传给 rd！故要 reg-en 控制 wb 决策！

那么可见，若是 lw 或 R-type，rd 的更新将在下-clk 上升沿；下-上升沿时，rd 读入（if reg-en 为1），与此同时，PC reg 给出 instr，将会陆续拆出新 rd，rs1，rs2；但前者进程更快，故新 reg 已来时，旧 rd 已读入，已准备好被读了（如上一个 rd 是现在 rs1）



* 还有 bne. blt. bge；可让 ALU 伸出 portal，并 instr 控制发挥

B-type：eg. beq rs1, rs2, L (imm/label)，若 X[rs1] == X[rs2]，则前往 L（PC+imm/label；但机器码用 imm/label-PC）

如何简便地在上一个版本的图（S-type）中修改，以支持 B-type？

回想 ALU 中计算过 X[rs1] - X[rs2] 的！可以用一个大或门判断32-bit结果是否为0！因此，ALU 可以额外出一个 zero portal.

这个 portal 是否发挥作用，应由 instruction machine code 提供的信息决定。

发挥作用时：
$$\begin{cases} PC = PC+4, & \text{if zero} \neq 0 \\ PC = PC+imm, & \text{if zero} = 0 \end{cases}$$

可见：原来的 PC+4 简单逻辑也需要进一步完善

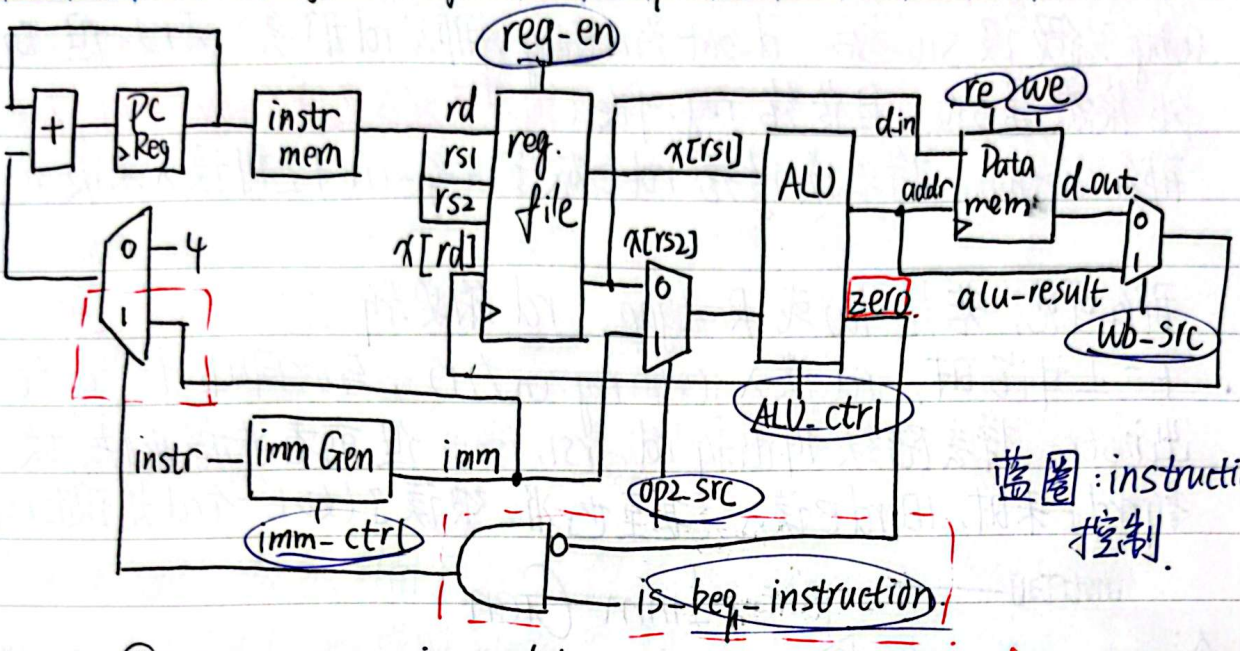
而 B-type 中 imm 组成又与 I. S 不同，故 imm Gen 也要改

KOKUYO

No.

Date

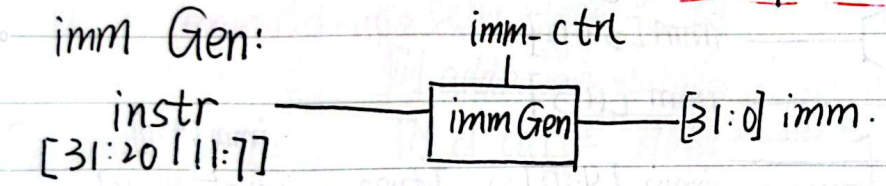
以下是支持 R.I-type & bge 的 datapath 图：

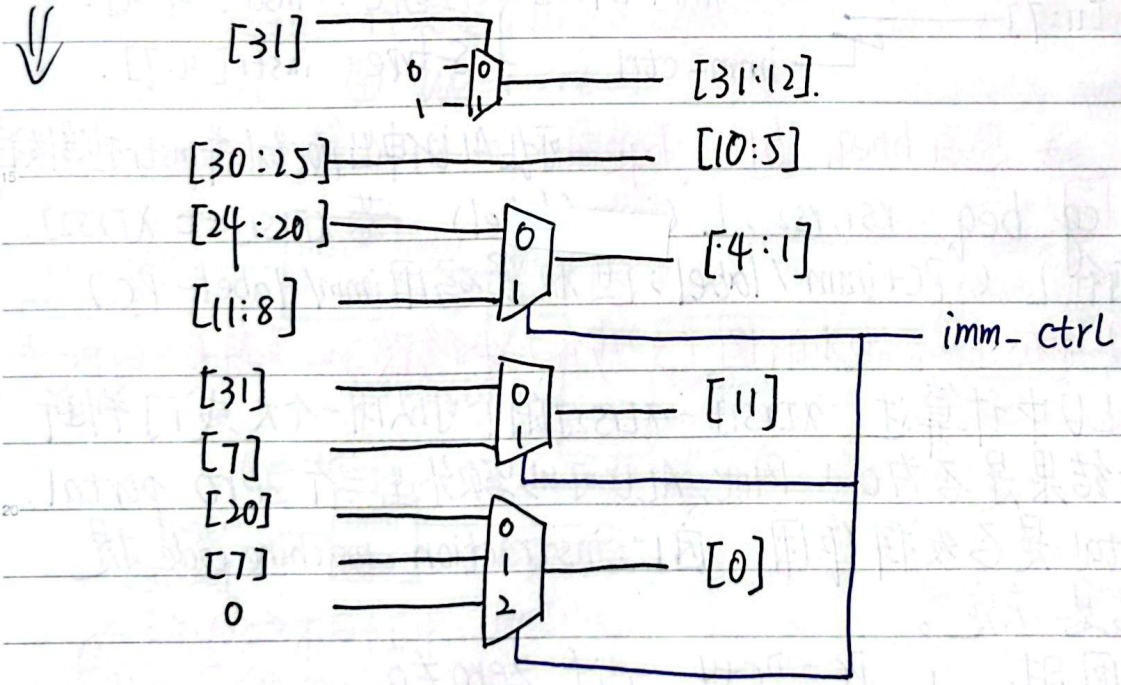


蓝圈：instruction 控制

★fig1

imm Gen:





Campus