No.

Date

CA Review: Pipeline

Performance: $\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \cdot \frac{\text{Cycles}}{\text{Instructions}} \cdot \frac{\text{Time}}{\text{Cycle}}$

简称：CPI

factors:

- ISA (RISC vs CISC)
- Program itself
- Compiler
- 编程语言

- Compiler
- Program

Microarchitecture implementation or circuit design / ISA

Eg:

Program:	Instr-type: A	B	C
CPI	2	2	4
percentage	20%	40%	40%

Program 有 10^6 instr, 以及 CPU 频率：2.5GHz

则 CPU time $= 10^6 \times$ Average CPI $\times 1/2.5\text{GHz}$.

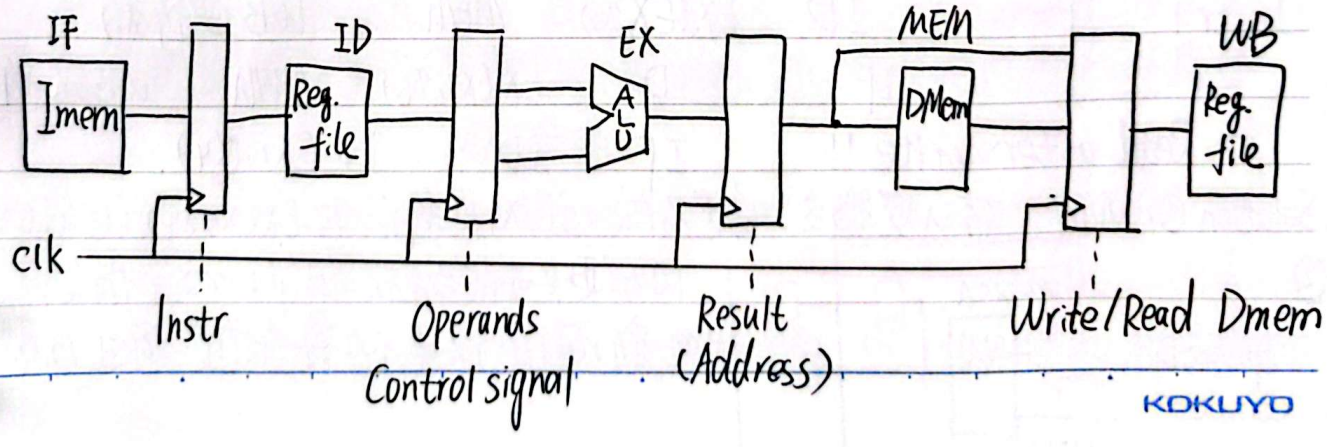
$= 10^6 \times 2.8 \times \frac{1}{2.5 \times 10^9} = 1.12 \times 10^{-3}\text{s} = 1.12\text{ms}$

如何提高 Time/Cycle，若仅是 $t_{critical\_path} \le t_{clk}$，一个path里面至多可经过5大组件 (IF. ID. EX. MEM. WB).

但若五个阶段视为5个收费站呢？而不是车必过5站，下辆车才可进？

这样，Pipelined CPU: $\max\{t_{IF}, t_{ID}, t_{EX}, t_{MEM}, t_{WB}\}$ (顶尖是 Σ).

那则需要设计"站"以让车有序过站了，⇒ reg!

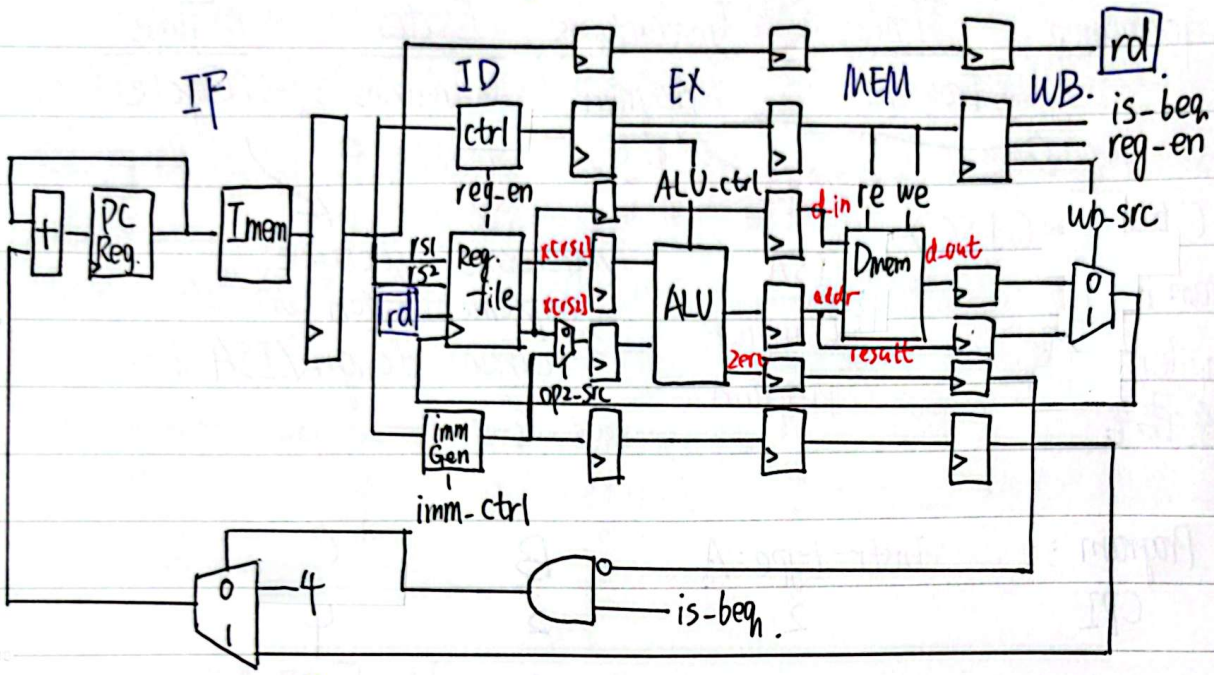


KOKUYO

No.

Date

同时注意到：rd不能赖在reg.file中不走，直到x[rd]来！故rd也要流动！



Structure

Hazards: ① CC1 CC2 CC3 CC4 CC5

Instr: 1 IF ID EX MEM X WB X

2 - - - - - -

3 — - -

4 IF X ID X

X: Dmem与Imem 说到底，在一块内存上！

X: Reg.file 能又存又写么？有先后顺序么？ to use physical resources

可在硬件上解决问题（从解决 Data&Instr Mem issue）；从及指令 take turns

② Data.1) CC1 CC2 CC3 CC4 CC5 CC6

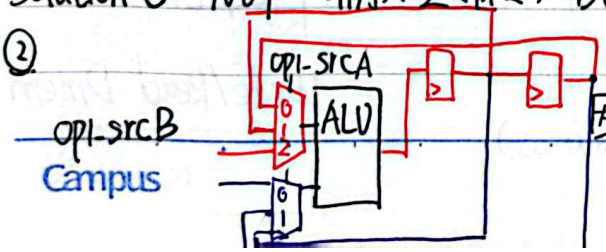
Instr.1 IF ID EX(x_2, x_3) MEM WB (更新x_1)

2 IF ID EX(x_5, x_6) MEM WB(更新x_4)

3 Read after write!! IF ID EX(x_1, x_4).

Solution ①. NOP: 输入空指令，啥也不做；让x_4更新后，再 EX(x_1, x_4). (Wait!)

②



因为冲突可能发生在相邻1or2条指令，因此新reg值但未不及存的值，通过mux

Campus

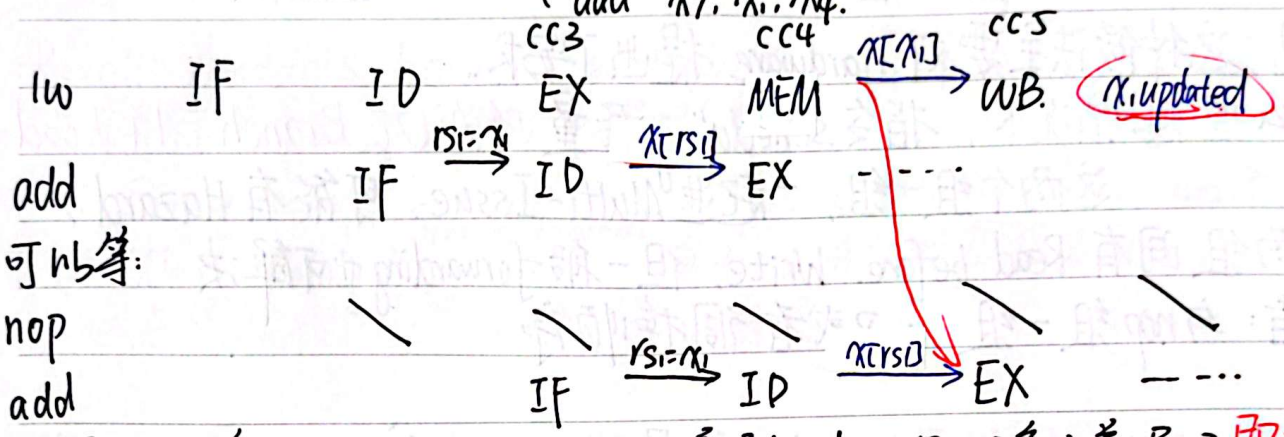
No.

Date

一言以蔽之：ALU结果值得以在1or2轮后返回Mux，在被存入Reg前就被可以利用，解决RAW干扰

而关于op1-src，大致上先可用reg_en判断：为1时，它便会注意之后是否要提前于WB而读它；而op1-src具体多少，取决于之后几条(1/2)命令后是rs1还是rs2要利用它

2). 'load delay slot'：{ lw x1, 0(x3)
add x7, x1, x4.



在隔1.2条instr时 Data Hazard 意料之中；但3条之差呢？即：能又写又读么？一般认为，这也有Hazard

或者 Code Rescheduling，将不冲突、不影响的指令先放lw后(compiler实现)

③ Control：考虑一个beq指令，若beq在3阶段后令PC跳至一个地方，那么前面pipeline中在跑的beq后几条指令岂不是全错？

一种解决方案是用nop等！待3阶段后PC给出正确address，再继续

另一种是照常：若不jump，则OK；若jump，则 额外花时间flush前四条指令所对reg. mem. 做的操作！

或者：Dynamic branch prediction: Predict if the branch will be taken based on the current record

最后，也可考虑类似于rd解决方案：forwarding to reduce delay of branches (将is_beq信号一出ALU就送过去).

KOKUYO