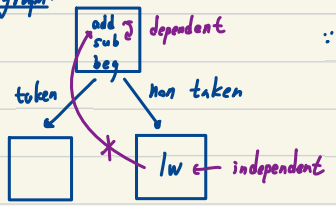
Control Dependency:

```
beg  $t0, $t1, Label
add  $s0, $s1, $s2
sub  $s7, $s4, $s5
lw   $s0, $s4, $s6
```

∵ 原先 MIPS pipeline 在 MEM stage 才決定是否要跳
當 beg 進到 MEM 後 add~lw 都已進到 pipeline
若為 taken，會造成此三個指令被 flush 之浪費

@ 由 Complier 解決：在 beg 和 add 間插入 3 個 nop 即可避免 beg 後面指令進到 pipeline
∵ conditional branch run time 才知道
∴ complier 不知道 taken or not taken

control flow graph: basic block



∵ complier 不知是否 taken
不能重排 independent 的 lw 指令到 add & sub 間消除

Ex.

```
beg $1, $2, L1
add $1, $2, $3
L1: slt $1, $2, $3
```

⇒ nop nop nop beg | IF | ID | EX | MEM | WB | ⇒

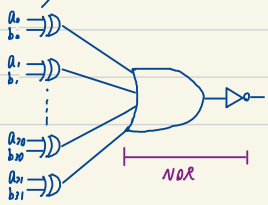
```
beg $1, $2, L1
nop
nop
nop
add $1, $2, $3
L1: slt $1, $2, $3
```

Solution 1. 透過修改 datapath，讓 beg 指令可在 ID Stage 決定是否要跳
就能減少插入 nop 數目成 1 個

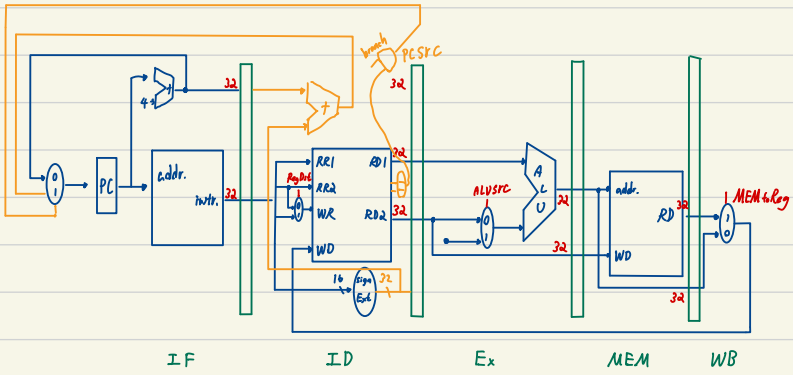
只要能在 ID stage 做到：
I. 計算跳躍目的位址
II. 比較 rs, rt register 內容是否相等

Cost 為需額外加入元件：XOR Array：32 ± 32 — Equal

設計如下：$rs = a31 a30 a29 ... a1 a0
$rt = b31 b30 b29 ... b1 b0

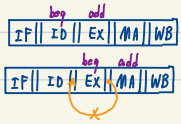


將 beq 指令改作 ID stage 決定是否要跳的 datapath 如下：（省略 FW unit & Hazard Detection Unit）



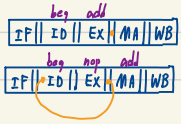
並且使用 XOR Array 判別兩 Reg 內容是否相等只需經 XOR, NOR 運算
比先前使用 ALU 做減法運算要來得快

Trade off: ∵ beq 指令在 ID stage 就要決定是否 taken
∴ 提前使用了 register 內容，會讓 data hazard 變嚴重

I.
```
add  $1, $2, $3
beq  $1, $2, L2
```


II. 若為 load-use 之 beq 情形則需 stall 2 個 clock.

∴ 需 stall 一個 clock 才能解決上面的問題

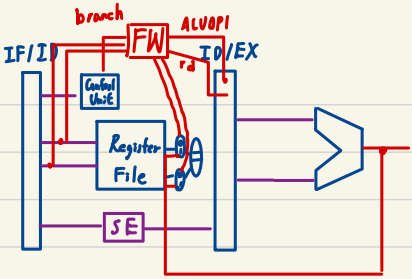
```
add  $1, $2, $3
stalled
beq  $1, $2, L2
```


Question：是否可利用 forwarding 解決？

Example:
```
add  $s0, $s1, $s2
beq  $s0, $t1, Label
```
add 的 $s0 值需在 EX stage 的 ALU 運算後得到，可以在得到時同時傳給後面 beq 的 XOR Gate
同時此 FW unit 會 check：
① Control Unit 產生的 branch 信号是否為 1（ID stage 指令為 beq）
② EX stage 指令為 R-type（看 ALUOP1 == 1）

② ID stage 的 rs, rt 是不是 EX stage 的 rd 相同

Datapath 設計:



Solution 2. delayed branch:

② 由硬件解決：

(1) Static Branch Prediction：

永遠預測 Branch 為 taken 或 non-taken

ⓐ non-taken：設 branch 不發生，讓指令進到 pipeline

如果 predict 失敗，才做 flush，將已進到 pipeline 丟棄

而 Predict 動作由 Complier 完成

Example：設 branch 在 ID stage 決定是否 taken

eg

```
beq $1, $2, L1      CC2：  add  beq
                           IF | ID | EX | MA | WB
add $4, $5, $6      CC3：  slt  nop  beq
                           IF | ID | EX | MA | WB
L1：slt $7, $8, $9
```

設 $1 = $2

需有控制信號將 IF/ID flush 成全 0 為：sll $0, $0, $0 = nop

必須看 PCSrc 來決定是否要 flush

若為 1，表示要跳 (Predict 錯，flush)

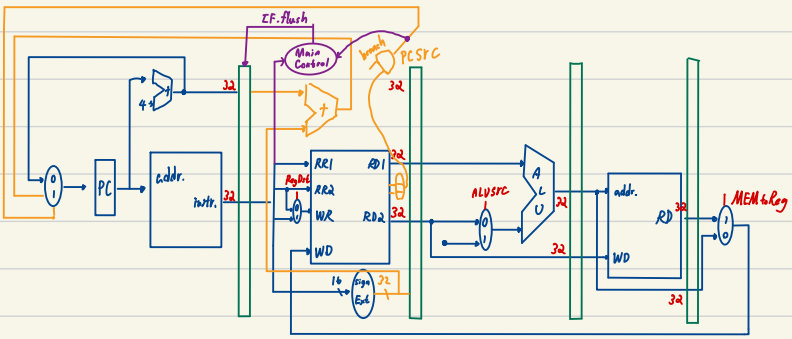
∴ 需加入一控制信號線

CC2 時，∵ 為 predict branch non taken，add 進到了 IF stage

而 CC2 結束時，beq 計算完 XOR gate 知道是否 taken

若是，會傳給 hazard detection unit，會清除 IF/ID pipeline register 為 0

加入 IF/ID. Flush 功能的 data path：



12). Dynamic Branch Prediction

和 static branch prediction 最大差別在：

會看前面 Branch 是否 taken 來 predict 接下來是否 taken

∵ 大部份情況，conditional branch 會為其一占大部份比例

e.g. for i=1 to 10 do

a = a + 1

MIPS code:

例 1.

```
    addi $s0, $0, 1
    addi $s1, $0, 10
L1: beq  $s0, $s1, Exit
    addi $s2, $s2, 1
    addi $s0, $s0, 1
    j    L1
Exit:
```

例 2.

```
    addi $s0, $0, 1
    addi $s1, $0, 10
L1: addi $s2, $s2, 1
    addi $s0, $s0, 1
    bne  $s0, $s1, L1
```

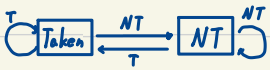
上面例 1 10次裡只有最後一次 taken ⇒ 9次 NT 1次 T

而例 2 10次裡只有最後一次 non taken ⇒ 9次 T 1次 NT

設為 static prediction，則：prediction 的 accurancy 為：0.5

但若利用 dynamic prediction 可提升至接近 0.9

a. 1-bit prediction



Example: for loop: TTTTTTTTTFTTTTTTTTTF

1-bits : FTTTTTTTTTFTTTTTTTTTF

2-bits : FTTTTTTTTTTTTTTTTTTT

b. 2-bit prediction