PIPELINING E HAZARD

→ **PROBLEMI DI CONFLITTI**: • strutturali, tentativo di usare la stessa risorsa nello stesso momento;

• sui dati, tentativo di utilizzare un risultato prima che sia pronto; (data hazard)

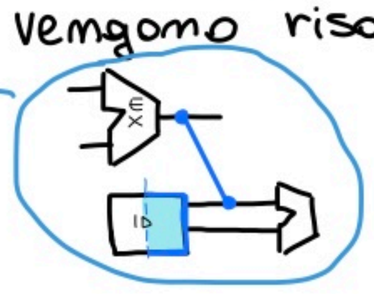
• sul controllo, tentativo di prendere una decisione sulla prox istruz. da eseguire prima che una condiz. venga verif; (control hazard)

→ i **conflitti strutturali** sono immediatam. risolti nell'arch. MIPS:
→ Mem Dati divisa da Mem istr;
→ banco registri usato sia in modo scritt. che lettura nello stesso ciclo di clock;

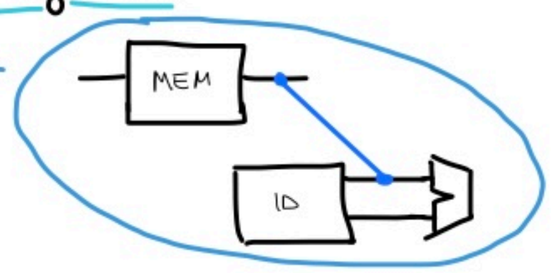
→ i **conflitti sui dati** vengono risolti tram propagazione registri:

dipend. che vanno 'indietro' nel tempo

tra istr R/R ho **prop. EX/EX**



tra istr LOAD/R ho **prop. MEM/EX**



→ i **conflitti di controllo** possono risolversi solo attendendo l'esec. della fase di decodifica che permette l'esec. di fetch dell'istr successiva:

↳ **branch untaken**, la cond è falsa quindi posso proseguire senza aspettare la fine dell'istr; Y+4 = PC+4 istr. succ.

↳ **branch taken**, deve inserire un ciclo di mezzo per permettere a istr di terminare correttamente; Z = PC+4+1*4

→ la prop. EX/EX e MEM/EX viene realizzata in 2 modi:

tecniche SW:
• **inserimento istr. nop**, ovvero non-operation;
• **scheduling istr**, ovvero riordino in base alle dipendenze e se nec. utilizzo delle nop;

tecniche HW:
• **inserimento bolle**, ovvero stalli tra stadi della pipeline; → se ne occupano PCWrite e IF/DWrite
• **propagaz. dati in avanti**, porto avanti EX e MEM per lasciarle libere ad altre istr;
↳ **forwarding** o bypassing ← propago il dato alle prox istr che ne hanno bisogno;

→ se aggiungo multiplexer a ing. ALU posso prelevare da qualsiasi registro i dati in ing senza dover usare stalli per conflitti dati;

→ se una istr. nello stadio di EX ha bisogno di un dato non ancora scritto in WB è nec. portare il dato all'ing corretto della ALU:

COPPIE CHE CREANO CONFLITTO:

1 → EX/MEM.regrd = ID/EX.regrs
2 → EX/MEM.regrd = ID/EX.regrt
} → **hazard in EX** → prop. sse RegWrite è asserito nello stadio del conflitto

3 → MEM/WB.regrd = ID/EX.regrt
4 → MEM/WB.regrd = ID/EX.regrs
} → **hazard in MEM**

↳ 3 tipi di **dipendenza** dati: • WAR, legge un dato che modifica nell'istr succ;
• WAW, scrive uno stesso dato 2 volte in 2 istr div;
• RAW, scrive un dato e poi lo legge; → non crea conflitto

→ per risolvere i **conflitti di controllo** (ovvero esec. salto) ci sono due casistiche:
→ **senza predizione**, aggiungo nop o stalli per attendere fase MEM in

cui so risultato verifica;

→ con predizione, ovvero o predico untaken branch: eseguo come se il fosse falsa e nel caso sia vera pongo a 0 i segnali di controllo delle tre istr succ;

BRENCH PREDIC. statica (ALWAYS TK / ALWAYS NOT TK)

→ pipeline ottimizzata, sposto decisione salto da MEM anticipando calcolo indr e valutazione del confronto così da inserire 1 solo stallo (o nop);

pred. dinamica

sposto sommatore da EX a ID e comparatore a ID

con necessità di propag. fino a EX di PCsrc e Branch e in alcuni casi conflitti su dati risolvibili solo da stalli

→ PRESTAZIONI PIPELINE:
- T(P) = tempo CPU per prova P
- Cicli(P) = n° cicli di clock CPU per P
- CK = periodo di clock
- FREQ = $\frac{1}{CK}$ freq. ciclo

↳ $T(P) = Cicli(P) * CK = Cicli(P)/FREQ$

↳ $CPI(P) = (NI(P) + Stalli(P) + 4)/NI(P)$ prestazione pipeline