Hazard structural ⇒ ∃ instr., dar struct. hw. nu dispune de resursele suficiente ~~necesare~~ ai. instr. să se poată executa.

Hazard de dte. ⇒ ∃ instr. de exec., ∃ și hw. care să o execute, dar execuția instr. necesită dte. anterioare care nu ~~au fost~~ sunt disponibile încă.

Hazard de comenzi ⇒ instr. de salt ⇒ tot ce s-a introdus în ppl s-a introdus până la salt ⇒ tb. să facem Fetch.

pg. 21: când compilatorul prevede un data dependency dictează introducerea unei întârzieri în i_2 ⇒ între i_1 și i_2 tb. introdusă o întârz. de 2 . . . ⇒ se introduce NOP între i_1 și i_2

pg. 27:

WAW:
- când avem un buffer la out. în care se pun rezultatele
- când avem mai multe ppline-uri
- când compilatorul transformă programul scris de programator în lbj. de niv. înalt ⇒ se face dynamic data dependency ~~check etc~~ check

~~2 metode: Tomasulo (?): pg. 30 → pg. 39 → la var. a) am concurență ptr. RS. WAW și sumatorul și multiplicatorul vor să scrie în RS ⇒ b) e; Scoreboard~~

2 metode:
- Tomasulo: (pg. 30) ⇒ pg. 39 → la var. a) am concurență ptr. RS. WAW și sumat. și multiplicat. vor să scrie în RS ⇒ b) e mai bună
- Scoreboard: (pg. 40) → instantaneu, la un mom. de timp dat
  → se păstrează până când apar modificări, apoi se modifică.

SPM. 6

pg. 41: Scoreboard:

Met. folosește niște tbl.: { Function Unit Status / Destination Register Status }

↓

arată reg. în care se scriu informații și unitatea funcțională în care se

execută instrucțiunile

La slide 49: la mult R5, R1, R2 } sunt independente ⇒
add R2, R3, R4 }

⇒ nu am hazard pt că s-a trecut de faza citirii lui R2.

Dc. nu s-ar fi trecut de faza citirii lui R2 am fi avut hazard WAR (Write after Read) ⇒ execuția instr. ADD e întârziată ca să nu avem hazard.

Până acum am vb de hazardul fețional, ce apare în sit. când nu avem destule unități feționale care să execute instrucțiunile (toate instrucț. în U, probabil).

pg.67: Hazard de comandă (Control Hazard): poate să apară când fluxul de instr. nu mai e secvențial ⇒ salturi ⇒ pp. golirea pipeline (FLUSH).

3 grupe de instr. 'branch':
- salt condiționat: IF, CASE...
- necondiționat: CALL, JUMP...
- bucle: WHILE, FOR...

Cel mai problematic e saltul condiționat ⟶ pg.66: în ppl, i_3, i_4 și i_5 au fost aduse și procesate inutil ⇒ i_k intră în procesare abia când i_2 e în stadiul de WB ⇒ a apărut o penalizare în performanță de 3 perioade ceas (în acest ex.).

Branch prediction: ⇒ calitatea ei influențează calitatea de ansamblu a sist.
⇒ 2 metode: statică și dinamică
- statică: făcută de compilator
- dinamică: bazată pe comportarea în trecut la instr. respective

⇒ counter based branch prediction ⇒ se face salt când contorul se incrementează, nu se face salt, când contorul se decrementează.

Delayed branching ⇒ în ppl, până în momentul în care se ajunge la salt se introduce după exec. instr. independente care to exec. independent de execuția saltului ⇒ ⇒ compilatorul to să fie mai complex.
pg.69 ÷ 71

3 tipuri de μP, d.p.d.v. al paralelismului la nivel de instr.:

- superscalar → Pentium
- superppl.
- very long instr. word (VLIW)

• Superscalar ⇒ multe u. hw. care rulează instr. în ||.
⇒ e nevoie de un mec. care să sincronizeze modul în care u. fct. își primesc instr. și operanzii și modul în care e înscris rez. ⇒ metode Scoreboard și Tomasulo;
⇒ ∃ un set de regiștri numiți reg. alias → dc. instr. se referă la un reg. R2, unul din reg. alias e utilizat pe post de R2 (în alias se găsesc rez. temporare; ∃ un unic reg. R2 real, și în el se va înscrie rez. după WB).
⇒ execuție out of order ⇒ instr. sunt exec. în ordinea permisă de disponibilitatea resurselor la un mom. dat, nu în ordinea în care au fost scrise de programator
⇒ rez. sunt puse în retirement unit ⇒ instr. a fost exec. cu reg. alias, iar apoi, din retirement unit, instr. sunt înscrise în reg. reali (prin WB) în ordinea în care au fost scrise de programator.

rez. sunt puse în retirement unit ← în ordinea în care au fost obținute (out of order), dar sunt scoase din retirement unit în ordinea în care le-a ~~scris~~ cerut programatorul (WB)

pg. 251 • Superppl.: ⇒ ppl. cu m. multe trepte ⇒ vtză. mare de execuție

• VLIW: ⇒ se asociază un compilator capabil să facă o instr. lungă din juxtapunerea m. multor instr. simple; în această instr. se găsesc niște biți de comandă care . . . . execuția.

pg. 80: ILP (Instruction Level Parallelism) → permite maximizarea vitezei cu care un progr. compilat dintr-un lbj. de nivel înalt e rulat pe o mașină;

→ se utilizează fie superscalar / superppl. / VLIW, fie o combinație;

→ Hb. se ține nr. mediu de instr. pe care un μP le poate efectua simultan.

La μP RISC: { OF ≥ DEC sunt cuplate într-o lg. treaptă ppl.-ID
EX ≥ WB ——— EX

→ 3 trepte în ppl.: IF | ID | EX

Per. de ceas ptr. RISC e mai mică decât la μP CISC.

Nu putem utiliza uneori RISC → Hb. să am un compil. care să decidă perechile de instr. ce pot fi exec. în paralel → pg. 83.

pg. 84: MLP (Machine Level Parallelism) → ∃ mai multe structuri ppl. în paralel

→ Hb. să ∃ un echilibru între ILP ≥ MLP.

SPM.7 pg. 82: Procesoarele de tip RISC au instr. simple care se execută în același nr. de cicli de ceas. În fig. sunt 3 perioade de timp în care struct. ppl. se umple. La instr. de salt ppl. Hb. vidată. Instr. simple → hw. simplu → e loc ptr. plasarea altor ppl. → e redusă durata de execuție per ansamblu (teoretic).

pg. 83: În practică e posibil ca între 2 instr. (i0 și i1) să ∃ data dependency. În fig. sunt prezentate instr. care sunt rulate în U. ∃ posibilitatea ca 2 instr. să nu poată fi realizate simultan datorită lipsei resurselor.

O metodă de creștere a vit. de exec. e fol. unor instr. f. lungi (se zicem 4 op. elementare). Dc. ∃ dependență apare necesitatea de pauză (4 timpi).