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).

Pg. 92: Creșterea vit. de execuție → instr. f. lungi (VLIW) → în VLIW au fost cuplate 4 instr. → apare stalling.

O mare creștere în vit. de execuție s-a făcut prin:
- extragerea în order a instr. (așa cum le-a introdus programatorul);
- executarea lor out of order (într-o ordine OK dpdv. logic, dar ≠ de ordinea dată de programator) ⇒ maximizarea utilizării resurselor hw.;
- salvarea rez. în retirement unit;
- înscrierea lor în reg. permanenți în order (WB).

Apare pb. complexității sporite a mec. de execuție a instr. → ib. să fie capabil să analizeze ordinea corectă în care să se exec. instr. out of order → mec. hw. → ocupă mai mult spațiu pe plăcuță!

Pg. 96: ii = emitere în order, finalizare în order → instr. sunt exec. în ordine, în funcție de disponibilitatea resurselor hw.; dc 2 instr. sunt emise simultan împreună, ele ib. finalizate împreună.

io = emitere în order, finalizare out of order → ib. analizat și asigurat că nu hazard WAR (execuția out of order ib. să nu pericliteze logica programului (rez. date în ordinea dorită de prog.))

ooo = emitere out of order, finalizare out of order → pp. max de complexitate a mec. de analiză → e necesar un hw. suplim. numit instruction window → instr. sunt aduse în instr. window out of order și sunt lansate în execuție în funcție de disponibilitatea unităților hw. de execuție.

Pg. 103: BHT = Branch History Table → info. despre cum s-a comportat instr. în trecut → a sărit sau nu.

BTB = Branch Target Buffer → info conține adresele la care s-a sărit la exec. anterioare ale resp. instrucțiuni de salt → dc. ajunge la instr. de salt, în ppl. se merge ultima adr. din BTB.

Planificarea structurilor ppl.:

→ de obicei struct. ppl sunt planif. aș. o treaptă să poată fi utiliz. de mai multe ori

→ Struct. statică → la (t) mom. de timp în toate treptele se găsesc în exec. elem. aceluiași tip de instr (de ex. adunare în VM).

→ Struct. dinamică → la un mom. dat în treptele structurii pot coe elemente ale mai multor tipuri de instr (de ex adunare și înmulțire în VM) → structura se adaptează singură la condiții;

→ la struct. statice, de. vreau să schimb tipul instr. care se exec., tb să termin complet ultima instr. din tipul curent, apoi să golesc complet ppl., să modific un pic structura (modif. dest. prin mai multe operații) și apoi să reîncarc ppl.;

→ coliziune: de la to introduc instr I_1, la t_1 îi trece în treapta 2 → la instr în treapta 1 introduc I_2; la t_2 I_2 tb. să treacă în treapta 2, dar I_1 tb. să reutilizeze treapta 2 → competiție ptr. resurse hw.! (pag. 116)

→ Htb. de rezervare: folosește ptr. a marca la anumite mom. de timp treptele struct ppl. care sunt ocupate cu exec. unei instr. → la to I_1 rezervă treapta 1, la t_2 și t_2 se rezervă treapta 2, la t_3 treapta 3, la t_4 din nou treapta 2, apoi struct. devine vidă;

→ Latența (latency): timpul mort = întârzierea cu care o a 2-a instr. tb. introdusă în ppl ptr. a se evita coliziunea cu instr. precedentă la anumite trepte de prelucrare; latență 0: introduc 2 instr. simultan → imposibil; pg. 116 → latență 4;

↓

ppl. în 3 trepte cu reacție → e echivalent cu un ppl cu 5 trepte secvențiale

listă a latențelor interzise → tr. 1: t_0; tr. 2: t_1 și t_2; tr. 3: t_3; tr. 4: toate

→ vector de coliziune: 5 elem.: $C_4\ C_3\ C_2\ C_1\ C_0$

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

4 3 2 1 0 = latența →

⇒ conține latențele interzise.

→ se generează diagrama stărilor: pg. 123:

- se pleacă din starea dată de vect. de coliziune;
- urm. stare se obț. deplasând spre dr. cu (2) starea anterioară și făcând SAU între rez. și vectorul de coliziune (cel inițial);

asta la pasul 2, pt. că aveam latența 2; la pas 3 shiftăm dr. cu 3 poz. (latența = 3) → pg. 123

→ latența medie minimă (MAL) = min (latențele medii)

↓

ciclul cu MAL maximizează rata de transfer prin ppl.

→ latența minimă → ar fi nevoie de un mec. care să numere (2 latențe, 3 latențe) și să introducă instr. după acele latențe (i_1 ␣␣ i_2 ␣␣␣ i_3 ␣␣ i_4 ␣␣␣ i_5...) am MAL = (2,5); e mai simplu să avem un mec. care introduce mereu instr. cu aceeași latență → latența minimă: i_1 ␣␣␣ i_2 ␣␣␣ i_3 ␣␣␣ i_4...

⇒ cu cadența asta vit. scade de 3 ori, dar ppl. f. simplu (pg. 129)

→ tot ce e valabil ptr. ppl. static e valabil și ptr. ppl. dinamic ⇒

⇒ pg. 130: 4 liste de latențe și o mtx. de coliziune ⇒

AA, BB, AB, BA — M_A & M_B

⇒ diagrama stărilor (pg. 135)