μP utiliz. regularea.

Memory Oriented μPv:

→ 2 μPs: un μP scalar și o struct ppl. ce face parte dintr μPv;

→ μP scalar extrage instr. din mem. și vede dacă e o instr. scalară sau una vect
  → scalară: o execută;
  → vectorială: o dă la UC vectorială ⇒ ea generează adresele vectoriale;

SPM.14 pg. 11: Memory Oriented Vector Processor

μPs vectoriale au 2 sect.: scalară, vectorială

μP scalar detect. ce instr. e scalară și o execută, în care instr. e vect., și o trimite la μP vect.

2 o sg. mag. de dte., o sg. mag. de adrese, dar aceasta e multiplicată prin buffere.

Unit. de comandă vect. primește instrucț. la solicitarea unității scalare; aceste instr. sunt decodificate . . .

Register oriented

Info. despre . . . . și . . . sunt stocate în regiștri, de unde pot fi luate repede de struct. ppl., iar comunicația e făcută de un disp. suplimentar și e compensată de . . .

Manipularea dtor în conj. cu mem. cache se face în blocuri.

Caracteristici:

→ aceste μP au o viteză mai mare → cost mai mare
  → de la început se proiectează așa . . .

→ prezența regiștrilor ⇒ crește viteza de operare; μP vectoriale sunt prevăzute cu seturi de instr. speciale, instr. orientate asupra operării cu vectori;

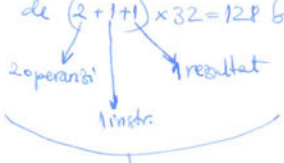
→ m. multe unități funcționează în ∥ ⇒ creșterea vitezei de prelucrare.

! Multiplicarea nr. de unități în ∥ NU duce la micșorarea timpului de execuție → date succesive accesate simultan.

Proiectarea memoriei (pg. 22): bandwidth = nr. mediu de cuv. accesat într-o unit. de timp → rata de comunicare cu memoria. Sistemul de memorie e compus din module.

pg. 24: Nr. cititi cei 2 operanzi pe cele 4 structuri ppl. și instrucțiunea, care e comună tuturor unit. ppl. → (2×4+1) elem. citite, și fiec. elem. are 32b ⇒ (2×4+1)×32 = 288 biți.

Lățimea magistralei ⇒ ptr. 4 ppl. cu instr. și date, pe 32 biți sunt necesare 4 magistrale de (2+1+1)×32 = 128 biți



e adv. că e aceeași instr. pe toate μPs la un mom. dat, dar fiecare μP își citește care e acea instr. !!!

config. mem. → module multiple ce permit acces simultan
→ registre și cache între μPs și mem. principală

pg. 29: mem. cu 3 porturi → mem. poate fi accesată prin 3 porturi, dar se fol. la un mom. dat portul 1 SAU portul 2 SAU portul 3, dar SAU nu înțeles ca un SAU exclusiv.

- portul 3 e port de write ⇒ înscrierea rez.
- porturile 1 și 2 sunt porturi read
- 1 sg. port lucrează la un mom. dat

Parțialul s-a mutat pe 30 noiembrie (Joi)!
Materia rămâne aceeași!!!

pg. 31: la to nu poate face nimic ptr. că la sf. lui to se introduce a_0. La fel și la t_1 ——— 12 ———
b_0. La t_3 începe procesarea în ppl, stage 1. Dc, ppl are 3 stages, la t_5 are loc pri înscriere în memorie: $w_0(C_0)$.

La t_1 și t_2 s-au emis cuvinte de memory read ptr a se evita a_1 și b_1 ⇒ la t_3 intră o nouă instr. la procesare în stage 1.

Ideal ar fi fost dc a_0 și b_0 ar fi intrat simultan în stage 1. Practic nu se poate (coliziune) ⇒ se face o întârziere.

pg. 32: am redus nr. de module de memorie ⇒ a e introdus cu întârziere, b e introdus cu întârziere și rez. e scris cu întârziere (ptr. a se evita coliziunile) ⇒ pri rezultat, W_0, apare la a 12-a per. de ceas!!! ⇒ struct. ppl. se umple de cca. 2 ori mai greu.

A 2-a instr. va deși tot la un tact după prima. Cadența se păstrează. ⇒ Tb. făcut un compromis cost/performanță ⇒ ~~nu interesează~~ ~~scade~~ m. multe memorii sunt mai costisitoare ⇒ ne interesează viteza. ~~Este~~ f. mare sau e ok și cu o viteză ~~mai mică~~ ?

pg. 33: → pe cât sunt m. multe trasee concurente memorie–μP band-width crește, dar crește și complexitatea hw-ului
→ gradul de multiplicitate al struct. ppl. ⇒ e greu de det. → trial & error;
→ manipularea secțiunilor de cod în legătură cu salturile ⇒ creștere mare a vitezei!

Împe cât timp se poate menține performanța de vârf și menținerea unui nivel de perf. maximă

Programarea UPV. pg. 37:

→ expandarea unui scalar într-un vector:
- în bucla 1 (cea din stg. slide-ului) ⇒ fiecare instanță a buclei tb. făcută separat (tb. calculat x)
- x e tr. într-un vector ⇒ toate secv. buclei pot fi făcute în paralel ⇒ ppl. UPV. !!!

→ loop unrolling:
- for-ul nu se putea exe. în ||, și mai avea nevoie și de branch prediction (for e o buclă!);
- expandarea buclei în elem. componente ⇒ calcul ||;

→ loop fusion (jamming):
Vrem să calculăm elem. lui M = Z + Y + P
- 2 bucle ⇒ BAD
- contopim buclele în una sg. → cele 2 instr. se pot realiza în ||. chaining ⇒ poate să facă adunări cu 3 operanzi ⇒ even better.

→ loop distribution:
o buclă poate conține elem. indep. și elem. de data dependency
⇒ se împarte bucla în 2 sect;
- dependentă de dte,
- indep. ⇒ instr. pot fi realizate în ||, ptr. că sunt indep. de dte

→ forțarea max. de muncă în bucla interioară ⇒ am o buclă int. cu 1000 de întreg. și una ext. care o repetă pe cea int. de 10 ori, de ex..

//

Pb.: 1) Adunare, înmulțire cu scalar, etc.
10) NU !!! (apar întârzieri ⇒ ~~...~~ controlul UPs și leg. cu mem.)

//

Marți ⇒ ce e mai imp. din materia de parțial