13. tétel első része:

„Modern processzor megoldások
(futószalag elv, hazard, sorrenden kívüli végrehajtás, spekulatív végrehajtás, szuperskalár processzorok, VLIW processzorok, vektor processzorok)”

A processzorok működésük során alapvetően egy ún. fetch-execute ciklust ismételgetnek. Ezt a lépéssorozatot egy egyszerű megközelítésben 5 fázisra bonthatunk.

1. Instruction fetch (IF): A következő gépi kódú utasítás beolvasás arról a memóriacímről, amit az programszámláló éppen tartalmaz.
2. Instruction decoding (ID): A beolvasott gépi kódú utasítás értelmezése. Mi a művelet? Hol vannak az operandusok? (Melyik regiszterben vagy mely memóriacímen? Milyen címzési módot kell használni, hogy kell egyáltalán meghatározhassuk a memóriacímüket?) Hová kell majd menteni az eredményt?
3. Operand fetch (OF): Az operandusok betöltése a műveletvégző egység (pl. ALU) bemeneti regisztereibe a megfelelő címzési módot használva.
4. Execution (EX): A művelet végrehajtása, mely során az operandusok értékei alapján meghatározásra kerül a kívánt eredmény (pl. az ALU kimeneti regiszterében).
5. Write back (WB): A kapott eredmény elmentése arra a memóriacímre vagy általános célú regiszterbe, amelyet a gépi kód definiált. A programszámláló regiszter tartalmának aktualizálása (általában a végrehajtott utasítás hosszával való növelése).

A modern processzorokban ez nem szekvenciálisan működik (szembe menve a Neumann elvekkel). Ennek az az oka, hogy az egyes fázisok megvalósítása külön-külön áramkörökben történik és soros végrehajtás esetén mindig csak egy-egy áramkör dolgozna, a többi pedig tétlen lenne.

Futószalag-elvű végrehajtás

A hatékonyság növelése érdekében alkalmazhatjuk a futószalag-elvű (pipeline) végrehajtást, mint az ILP (utasítás szintű párhuzamosság) egyik lehetőségét. Ekkor egyszerre több elemi (gépi kódú, nem a magasszintű programozási nyelvek esetén használatos, hanem assembly szintű) utasítás van végrehajtás alatt, de mindegyik másik fázisban. Például míg az egyik utasítás operandusai betöltődnek, addig dekódolhatjuk a következő utasítást és közben elkezdhetünk betölteni egy harmadikat és így tovább. Így tehát minden lépésben (órajelciklusban) befejeződik egy utasítás, bár mindegyiknek a végrehajtása több, mint egy órajelnyi időt igényel. Egyes rendszerekben egy utasítás végrehajtása akár 20-30 fázisra is felosztható.

Bizonyos esetekben gondot az, hogy egy utasítás hamarabb elkezdődik, mint ahogy az őt megelőző befejeződne. Az ilyen helyzetet **hazard**nak nevezzük. Ennek 3 típusa van:

- Adat hazard: Például egy olyan adatra van szükségünk, amely még nem állt elő (adat függőség) vagy egy előálló adat felül akar írni egy olyat, amire még szükségünk lesz (név függőség).

- Strukturális hazard: Ekkor két utasítás végrehajtásának egy-egy részlépése architekturális okok miatt nem hajtható végre párhuzamosan.
- Vezérlési hazard: Egy döntési helyzetben (elágaztatás, ciklusszervezés) még nem ismert a feltétel értéke, amikor már a következő utasításokat el kell kezdeni feldolgozni. Nem tudjuk azonban, hogy melyeket. (Az igaz vagy a hamis ágat? A ciklusmagot vagy a ciklus utáni részt?)

Az ilyen helyzetek kezelésére különböző technikákat alkalmazhatunk:

- Pipeline buborék (bubble): A futószalagra egy „buborékot” helyezünk. Ezáltal az üresjárat által az egyik utasítás „várakozik”, amíg meg nem szűnik veszélyes szituáció, azaz a hazard. A végrehajtás ugyan lassul kicsit, de nem annyira, mintha egyáltalán nem lenne futószalag.
- Operandus/eredmény továbbítás (bypassing): Bizonyos esetekben (pl. adatfüggőség) a szükséges késleltetés mértékét (buborékok számát) csökkenthetjük azáltal, hogy a EX fázisban előálló eredményt egyből elérhetővé tesszük az OF fázisban nem csak a WB fázis után.
- Regiszter átnevezés (renaming): A névfüggőség elkerülhető lenne végtelen sok regiszterrel, de néha elegendő az is, ha megfelelően használjuk a rendelkezésre álló regisztereket. Az ilyen technikát alkalmazó processzorban az értékek nem biztos, hogy abba a regiszterbe kerülnek, amelyet a programozó/assembler megadott, így csökkentve a hazardok számát.
- **Sorrenden kívüli végrehajtás** (Out-of-order execution, OoOE): A programban szereplő gépi kódú utasítások sorrendjének megváltoztatása egyfajta futási idejű hardveres újrafordítás során. A processzor egy kis méretű utasításablak használatával előre beolvassa a következő néhány utasítást és a végrehajtásukat úgy ütemezi, hogy lehetőleg ne alakuljon ki hazard, de a végeredmény változatlan maradjon. Így a végrehajtás megközelítőleg buborék mentes maradhat, de jóval bonyolultabb processzori áramköröket igényel.
- **Elágazásbecslés** (branch prediction): Vezérlési hazardok elkerülésére használhatjuk. A processzor megpróbálja megbecsülni, hogy melyik ágat kell majd végrehajtania és annak az utasításait helyezi a futószalagra. Ha helyes volt a becslés, akkor nem volt szükség buborékra, helytelen becslés esetén az eredmények eldobásra kerülnek és elkezdődik a megfelelő ág végrehajtása. A jó becslési mechanizmus nagyon fontos, különösen a fokfázisú futószalagok esetén.

Egyes processzorok elegendő erőforrással rendelkeznek ahhoz, hogy egyetlen órajel alatt nem csak egy, hanem több utasítást is képesek kezelni minden pipeline fázisban, így többszörözve az áteresztőképességet. Ezek a **szuperskalár** processzorok. Ekkor különösen nagy a hazardok esélye a teljesen párhuzamos végrehajtás miatt. Az ilyen rendszerek ILP mellett többnyire használnak OoOE-t is, így az egy fázisban futó utasítások kiválaszthatóak úgy, hogy azok lehetőleg egymástól függetlenek legyenek. Képesek normál szekvenciális programok futtatására. Képesek lehetnek továbbá mohó végrehajtást megvalósítására is. Ez másik lehetőség a spekulatív végrehajtásra (az elágazásbecslés mellett). Ekkor egy vezérlési hazard

esetén mind a két végrehajtási ág felkerül a futószalagra, de az egyiknek az eredménye eldobásra kerül, amit kiderül melyik ágat kell végrehajtani.

Nagyon hosszú utasításszavú (VLIW) processzor

Az ILP és OoOE megvalósításának egy olyan módja, ahol a forráskód fordítása során az elemi utasításokból egy hosszú, összetett utasítást (ún. köteg) állít elő a speciális fordítóprogram. Az utasítások megfelelő átrendezése révén a kötegek olyan utasításokat tartalmaznak, amelyek a teljesen párhuzamos végrehajtása során a nem fordul elő hazárd. A fordítás természetesen így lassabb lesz (és speciális fordítóprogram is szükséges), de a végrehajtás nagyon hatékony. A hardver lehet sokkal egyszerűbb is mint egy superskalár processzor, mert nincs szükség a futási idejű újrafordításra a hazardok elkerülése érdekében. Ennek egyfajta továbbfejlesztése a EPIC (Explicitly Parallel Instruction Computing).

Vektor processzor

Adatpárhuzamosságot megvalósító SIMD rendszer. Nagyméretű regisztereket tartalmaz, amelyekbe egynél több adat fér el egyszerre és képes egy adott utasítást az összes tárolt értéken egyszerre végrehajtani. (Például egy 128 bites regiszterben elvileg tárolhatunk 2 double vagy 4 float vagy 4 int vagy 8 short vagy 16 char típusú értéket.) A speciális regiszterek használatához speciális utasításokat kell használni.

A történelem során különböző megvalósítások jelentek meg:

- MultiMedia eXtension (MMX)
- 3DNow!
- Streaming SIMD Extension (SSE, SSE2, SSE3, SSE4)
- Advanced Vector eXtension (AVX, AVX2, AVX-512)

Ezekben folyamatosan növekedett a regiszterek mérete és megjelentek újabb és újabb utasítások, de közben törekedtek a visszafelé kompatibilitásra is. Emiatt például az AVX-512 64 bájtos regisztereinek (`ZMM0–ZMM31`) alsó fele használtató 32 bájtos AVX regiszterként (`YMM0–YMM15`), ezek kis helyiértékű 128 bites része alkalmazható SSE regiszterként (`XMM0–XMM15`) továbbá ezeknek alsó fele felfogható 64 bites MMX regiszterként (`MM0–MM7`).

Tömbök (vektorok) feldolgozásánál lehet nagyon hatékony ez a megoldás. A grafikai feldolgozó egységek (GPU) is ilyen logikát használnak működésük során.

Más technológiák

A modern processzorokban további technológiák is elérhetőek a teljesítmény növelése érdekében:

- Dinamikus órajelszabályozás (Intel Turbo Boost, AMD Turbo Core)
- Több szintű (integrált) gyorsítótár
- Hyper-threading (kettő vagy több utas)
- Többmagos és sokmagos processzorok
- Virtualizáció támogatás (VT-x, AMD-V)
- Általános célú GPU, integrált FPGA

Megjegyzés:

A processzorok alapvető működésével és az adott témakörrel kapcsolatos tágabb ismeretség nem hátrány.

Kapcsolódó tantárgyak:

- *Számítógép architektúrák*

Ajánlott irodalom:

- John Paul Shen, Mikko H. Lipasti: *Modern processor design* (Waveland Press, 2013)
  2., 4., és 5. fejezet.
- Sarah L. Harris, David Money Harris: *Digital Design and Computer Architecture* (Morgan Kaufmann, 2016)
  7. fejezet.
- Christopher J. *Hughes: Single-Instruction Multiple-Data Execution (Synthesis Lectures on Computer Architecture)* (Morgan & Claypoolm 2015)
  2. és 3. fejezet.