1. **A Neumann architektúrájú processzorok minimális regiszterkészletének funkcionális leírása**

* **AC** (akkumulátor regiszter): általános célú regiszter, műveletekhez szükséges operandusok egyikét, valamit azok eredményét tárolja
* **AR** (adat regiszter): adatátvitelt biztosít a memória és a CPU belső alkotóelemei közt (olvasásnál ide kerül be először a memóriából, írásnál ide kerül bele, majd innen íródik ki)
* **PC** (program számláló): a következő utasítás címét tárolja
* **CR** (cím regiszter): a memória fele közvetített (utasítás vagy operandus) címét tárolja (ide íródik be a cím, amelyen az következő utasítás, vagy éppenséggel az adat található, ezzel kerül megcímzésre a tároló)
* **UR** (utasítás regiszter): dekódolja az utasítást, felosztja utasítás kódra (ez megy a PU fele) illetve operandus címre (ha van, pl. melyik címen van a szükséges adat)
* **ALU** (aritmetikai-logikai egység): egy vagy két operandusú logikai és aritmetikai műveleteket hajd végre
* **PU** (processing unit): többállapotú automata, az egész CPU-t összehangolja, az általa létrehozott kontrolljelek vezérlik az összes részegység működését

1. **A Neumann architektúrájú processzorok vezérlő egységének funkcionális leírása**

* a központi egység fő feladata végrehajtani a memóriában tárolt utasításokat
* egy utasítás-végrehajtás ciklus két fázisból tevődik össze:
  + kiolvasás és dekódolás (fetch and decode)
  + az utasítás végrehajtása (execute)
* az utasítás végrehajtása tartalmazza az utasítás dekódolását, az operandusok betöltséét, a múvelet végrehajtását, és ha szükséges, az eredmény visszaírását a memóriába
* a vezérlő egységnek kell biztosítani az utasításokhoz szükséges vezérlőjelek sorozatát
* a vezérlőjelek engedélyezik, vagy lezárják az adat útvonalát, viszont ez több útvonalon is átvezethető
* azt, hogy melyik útvonalon halad az adat, az utasítás határozza meg

1. **A processzorok belső sínjeinek – az adatút – kialakításának lehetőségei**
2. **Függvényhívási utasítás végrehajtásához szükséges CPU alegységes és azok feladatai**

* függvényhívás esetén, a függvényhívást követő utasítás címét (**PC+1**) a vezérlőegység automatikusan elmenti a verembe (**SP**), majd a programszámlálóba (PC) betöltődik a függvény kezdőcíme
* a visszatérés esetén, a vezérlőegység kiolvassa a verem tetejéről az elmentett utasítás címét, majd betölti a programszámlálóba
* ezen kívül, ha szükséges, olyan regiszterek (pl. **FS**)állapotai is elmentődnek, melyek állapotai szükségesek a korábbi állapot visszaállításához; ezek értékei a függvényből való visszatérés esetén ismét betöltődnek

1. **Az utasítás-formátum hatása a processzorok utasítás-készlet architektúra szintjének tervezésére**

* az **ENIAC**-al szemben, a Neumann architektúrára épülő gépek más utasítás-formátummal rendelkeznek
* az ENIAC formátuma:

Utasítás kód-**x** **|** 1. operandus címe-**y** **|** 2. operandus címe-**y |** eredmény címe-**y |** következő utasítás címe-**y**

* ebben az esetben a memória lokáció mérete **x+4y**, az jelentvén, hogy csupán utasítást tud kezelni, illetve memória likációt megcímezni
* ezzel szemben a Neumann architektúrához tartozó utasítások másként vannak felépítve:

Utasítás kód – **n |** Operandus cím – **n**

* a memória lokáció mérete **n+m**, a processzor pedig utasítást tud kezelni, memória lokáció megcímezni
* észrevehető, hogy a 2. esetben sokkal több utasítás és nagyobb méretű memória használható ugyanakkora utasításra vetítve

1. **Az operatív tárak szervezése. Lapszervezésű virtuális tár.**

* A számítógépek memóriája, felhasználás szerint 3 típusú lehet:
  + Belső memória (regiszterek): kicsi, gyors, ideiglenes tárolásra szolgál
  + Alap/központi memória: működés közbeni tárolásra szolgál, közvetlen kapcsolaton keresztül fér hozzá a CPU (sín, vezérlő jelek)
  + Háttér memória: nagy, lassú, hosszú távú tárolásra alkalmazzák
* Elérési mód szerint:
  + RAM: minden lokáció elérési ideje ugyanaz
  + Soros elérésű memória: az elérési idő változik a cím függvényében
* Fizikai felépítés alapján:
  + RAM:
    - Dinamikus – kondenzátorokból épül fel, periódikusan frissíteni kell
    - Statiku – D bistabilokból épül fel
  + ROM
  + EPROM – optikai úton törölhető, újrapprogramozható
  + EEPROM – elektromosan törölhető, újraprogramozható
  + FLASH-ROM – gyors, nagyszámú törlés/újraírást támogató ROM memória
* a központi memória szavakra szervezett, az adatok írása/olvasása egyidőben történik a szavat alkotó összes bit esetén
* a virtuális memória lényegében a címzéstartományt jelenti, ez nem más mind egy másodlagos memória (háttértár)
* a virtuális memórria egyenlő lapokra oszlik
* létezik egy laptáblázat, mely információt tartalmaz minden oldalról: az odlal címét a memóriában, be van-e töltve, és állapotbiteket
* egy virtuális cím egy lapcímből és egy eltolásból áll, a processzor a Memory Management Unit-on keresztül kikeresi a lapszámnak megfelelő információt, ha az alapmemóriában megtalálható, kiolvassa a kezdőcímét és hozzáadva az eltolást, kialakítja a fizikai címet
* ha nincs bent a lap, beolvasódik a háttértárról
* ha nincs elég hely a lap betöltésére, a legrégebben használt lap, kicserélődik a kérttel

1. **Vektorizált megszakításrendszert alkalmazó I/O adatátvitelek végrehajtásának lépései**

* vektorizált megszakításokat a perifériák által küldött adatmennyiség lecsökkentésének érdekében,a címküldés helyettesíthető megszakítás vektor küldésével
* ennek alapján a CPU kiolvassa a megszakítás vektor táblázatból a lekezelő rutin címét
* míg a CPU arra vár, hogy lassú perifériákkal adatcserét végezzen, más perifériák adatokat szolgálhatnak, ennek kivitelezésére egy lehetséges megoldás a megszakításrendszer
* ha az adat elvihető, a perifériás berendezés megszakításkéréssel jelentkezik a CPU felé
* ezek a megszakításkérések úgy tekinthetőek mint függvényhívás
* a CPU meghatározza a megszakításkezelő címét, majd függvényhívásként átugrik arra, lementi a PC+1-et, illetve a FR-t
* mikor az adatok elmentődtek, a függvényből visszatér a CPU, majd folytassa addigi munkáját

1. **Közvetelk memória-hozzáférést (DMA) alkalmazó I/O adatátvitelek végrehajtásának lépései**

* célja a háttértárolón lévő adat memóriába helyezése, minimális CPU időt felhasználva
* ez néhány egyszerű lépséből valósítható meg:
  + a CPU felprogramozza a DMA kontrollerét C darab byte átvitelére X címre
  + a DMA egy kérést intéz a háttértár kontrolleréhez, az adat átmásolására a memóriába
  + a kontroller inicializálja a DMA átvitelt
  + a kontroller elküld minden byte-ot a DMA kontrollerének, a DMA növeli a cél címét(++X), csökkenti a byte-ok számát(--C), ismétli míg C!=0
  + ha nincs több átvitelre váró adat, a DMA egy megszakítást küld a CPU felé, jelezvén, hogy az átvitel megtörtént

1. **Az utasítások végrehajtásának párhuzamosítása csővezeték struktúrák alkalmazásával**

* a lapozás és szegmentálás megnöveli az utasítások végrehajtásának idejét, mivel több mikorlépésre van szükség, valamit több memória hozzáférési ciklusra
* alapesetben is megkülönböztetünk 3 lépést: **decode 🡪 fetch 🡪 execute**
* ha gyorsítani akarjuk ezt a folyamatot, akkor tervezhetünk külöegységeket melyek 1-1 fázis tudnak elvégezni. Ezeket pedig pipeline struktúrába helyezzük, ekkor így alakul:

**fetch 🡪 decode 🡪 exec**

**fetch 🡪 decode 🡪 exec**

**fetch 🡪 decode 🡪 exec**

* ilyen módon elérhetünk egy gyenge párhuzmaosítást, illetve növeljük a sebességet is