Input, output (I/O) utasítások

A külvilággal történő információ csere **port**-okon (kapukon) keresztül zajlik. A kapu egy memória cím, az információ csere erre a címre történő írással, vagy erről a címről való olvasással történik. Egy-egy cím vagy cím csoport egy-egy perifériához kötődik.

Input/output

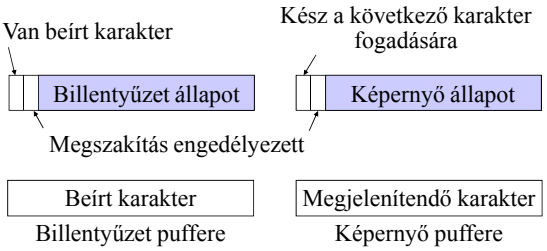
Az **I/O** végrehajtása lassú ↔ a **CPU** gyors, a **CPU** várakozni kényszerül

I/O regiszter (**port**): a **port** és a központi egység közötti információ átadás gyors, a periféria autonóm módon elvégzi a feladatát. Újabb perifériához fordulás esetén a **CPU** várakozni kényszerülhet.

- **Pollozásos technika** (~tevékeny várakozás)**:** a futó program időről időre megkérdezi a periféria állapotát, és csak akkor ad ki újabb **I/O** utasítást, amikor a periféria már fogadni tudja. A hatékonyság az éppen futó programtól függ.

Input/output (5.31-33. ábra)

Az **I/O** vezérlő regiszterei (**5.31. ábra**).
Terminál: külön regiszterek az inputra és outputra.



- **Pollozásos technika** (tevékeny várakozás)**:**

```
public static void output_buffer(int buf[], int count) {
  // count számú bájt kiírása buf-ból az eszközre
int status, i, ready;
 for(i=0; i < count; i++) {
   do {
     status = in(display_status_reg); // az állapot lekérdezése
     ready = (status >> 7) & 0x01;   // a kész bit elkülönítése
   } while(ready != 1);
   out(display_buffer_reg, buf[i]);
 }
}
```

Programozott **B/K** (**5.32. ábra**)

Megszakítás

A (program) megszakítás azt jelenti, hogy az éppen futó program végrehajtása átmenetileg megszakad – a processzor állapota megőrződik, hogy a program egy későbbi időpontban folytatódhassék – és a processzor egy másik program, az úgynevezett **megszakítás kezelő** végrehajtását kezdi meg.

Miután a megszakítás kezelő elvégezte munkáját, gondoskodik a processzor megszakításkori állapotának visszaállításáról, és visszaadja a vezérlést a megszakított programnak.

- **Megszakítás vezérelt I/O**
  Bizonyos előkészületek után az eszköz megkapja a feladatát (pl. az első karakter kiírását), és a program futása folytatódik.
  Ha az eszköz elkészült a feladatával, akkor beállítja a „Megszakítás engedélyezett” bitet. Ez megszakítás kérést eredményez. A megszakítás bekövetkezésekor ez a bit törlődik.
  A megszakítás kezelő újabb feladatot ad az eszköznek (a következő karakter kiírását) mindaddig, amíg a teljes feladat el nem készült, és visszatér a megszakításból.

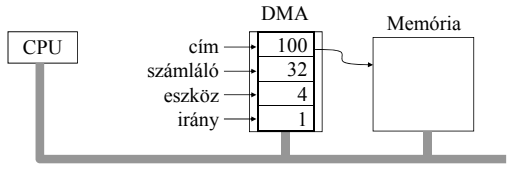
Az **I/O** művelet végrehajtása közben a központi egység más feladatot végezhet.

Pl.: Egy sornyi karakter képernyőre írása a terminálon.

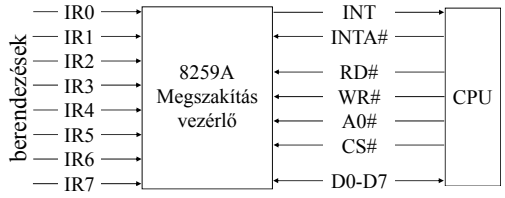
Előkészítés: Egy rendszerprogram összegyűjti a kiírandó karaktereket egy pufferben, beállít egy globális változót, hogy mutasson a puffer elejére, egy másik globális változóban megadja a karakterek számát. Megnézi, hogy a terminál tud-e adatot fogadni (**5.31. ábra**), és ha igen, akkor elindítja az első karakter kiíratását. Ekkor a **CPU** fölszabadul egy másik program futtatására.

A terminál a képernyőre írja a karaktert, és **megszakítást kezdeményez**. A megszakítás kezelő újabb karakter kiírását kezdeményezi . . .

- **DMA** (Direct Memory Access, **5.33. ábra**)
  Egy tömb kiírása/beolvasása során nagyon sokszor következne be megszakítás, és mindannyiszor le kellene fusson a megszakítás kezelő eljárás. Jobb megoldást kínál a **CPU**-nál lényegesen egyszerűbb **DMA**.
  A **DMA** önállóan végzi az eszköz figyelését és az adatok mozgatását. Cikluslopás.



Megszakítás kezelés (3.43. ábra)



IRi →, **INT** →, ha **CPU** tudja fogadni, akkor **INTA#** →, **i** → **D0-D7**, a **CPU** megszakításvektor táblázat **i** –edik eleméből tudja a megszakítást kiszolgáló eljárás kezdőcímét, létrejön a megszakítás …

Nyolcnál több eszköz kiszolgálásához több megszakítás vezérlő kapcsolható össze.

Megszakítás

Hardver tevékenységek (3.42. ábra):

1. Az eszköz vezérlő megszakítás jelet tesz a sínre,
2. ha a **CPU** fogadni tudja a megszakítást, nyugtázza,
3. az eszköz vezérlője az eszköz azonosítószámát (megszakítás-vektor) elküldi a sínen,
4. ezt a **CPU** átmenetileg tárolja,
5. a **CPU** a verembe teszi az utasításszámláló aktuális értékét és a **PSW**-t,
6. a **CPU** az azonosító indexű megszakítás kezelő címét teszi az utasításszámlálóba és gyakran betölti vagy módosítja **PSW**-t.

Szoftver tevékenységek (terminálra íráskor)**:**

7. menti a használni kívánt regisztereket,
8. kiolvassa egy eszközregiszterből a terminál számát,
9. beolvassa az állapotkódot,
10. ha **B/K** hiba történt, itt lehet kezelni,
11. aktualizálja a mutatót és a számlálót, a kimenő pufferbe írja a következő karaktert, ha van,
12. visszajelez az eszköz vezérlőnek, hogy készen van,
13. visszaállítja a mentett regisztereket,
14. visszatér a megszakításból, itt történik a **PSW** eredeti értékének visszaállítása is.

Átlátszóság: Amikor bekövetkezik egy megszakítás, akkor bizonyos utasítások végrehajtódnak, de amikor ennek vége, a **CPU** ugyanolyan állapotba kerül, mint amilyenben a megszakítás bekövetkezése előtt volt.

Csapda és megszakítás

Csapda (trap): A program által előidézett feltétel (pl. túlcsordulás, csapdát eredményező utasítás) hatására automatikus eljárás hívás. **Csapda kezelő**.

Megszakítás (interrupt): Olyan automatikus eljárás hívás, amit általában nem a futó program, hanem valamilyen **B/K** eszköz idéz elő, pl. a program utasítja a lemezegységet, hogy kezdje el az adatátvitelt, és annak végeztével megszakítást küldjön. **Megszakítás kezelő**.

A csapda a **programmal szinkronizált**,
a megszakítás nem.

A megszakító rutin megszakítható-e? Gyors periféria kiszolgálása közben megszakítás kérés, …

„Alap" állapot – „megszakítási" állapot, megszakítási állapotban nem lehet újabb megszakítás.

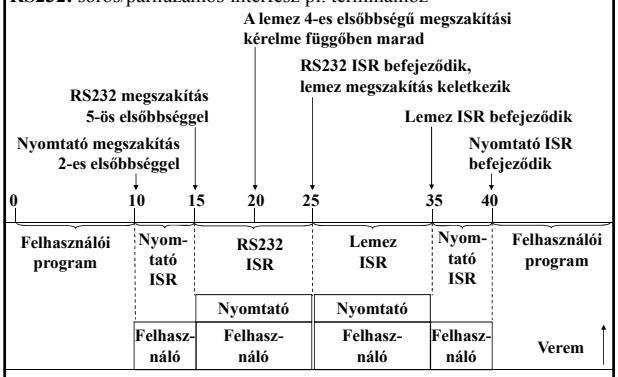
Hierarchia: megszakítási állapotban csak magasabb szintű ok eredményezhet megszakítást.

Bizonyos utasítások csak a központi egység bizonyos kitüntetett állapotában hajthatók végre, alap állapotban nem → csapda, szoftver megszakítás.

Megoldható az operációs rendszer védelme, a tár védelem stb.

A megoldás nem tökéletes: **vírus**.

ISR: Interrupt Service Routine **5.46. ábra**
RS232: soros/párhuzamos interfész pl. terminálhoz



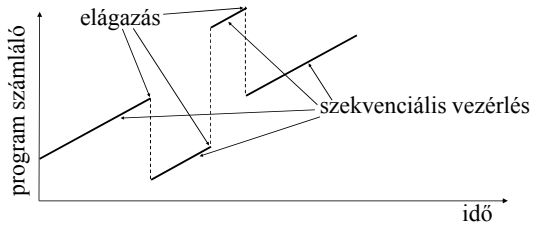
Sok esetben a programnak nincs mit tennie, amíg az I/O be nem fejeződik, pl. a klaviatúráról vár adatot. A várakozás helyett jobb megoldás, ha az operációs rendszer átmenetileg fölfüggeszti a program működését, és elindítja egy másik program futását.

Ez vezetett a **multiprogramozás** kialakulásához.

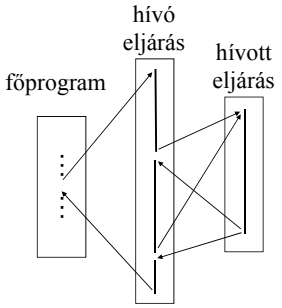
Vezérlési folyamat

Szekvenciális vezérlés: Az utasítások abban a sorrendben kerülnek végrehajtásra, ahogy a memóriában elhelyezkednek.

Elágazás: **5.39. ábra**.



Eljárás (5.44. ábra): Az eljáráshívás úgy tekinthető, mint egy magasabb szinten definiált utasítás végrehajtása: sokszor elég, ha azt tudjuk, mit csinál az eljárás, nem lényeges, hogy hogyan.

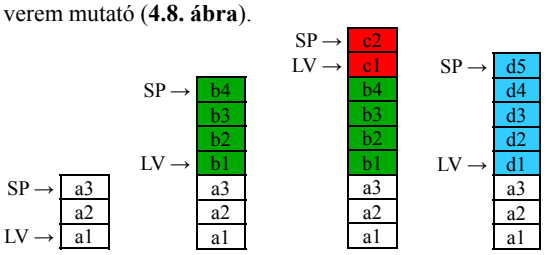


Rekurzív eljárás: önmagát közvetlenül vagy közvetve hívó eljárás.

Eljárás: paraméterek, munka terület.

A hívó és hívott eljárás paraméterei, változói nem lehetnek azonos területen: **lokális változók**.

Verem (stack): **LV** (Local Variable), **SP** (Stack Pointer) verem mutató (**4.8. ábra**).



Rekurzív és re-entrant eljárások

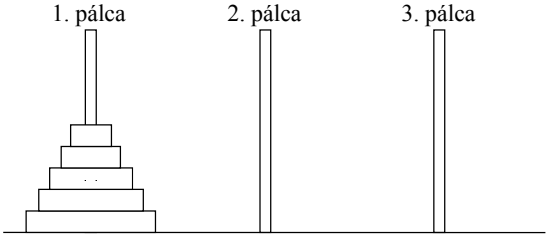
Egy eljárás **rekurzív**, ha önmagát hívja közvetlenül, vagy más eljárásokon keresztül.

Egy eljárás **re-entrant**, ha többszöri belépést tesz lehetővé, ami azt jelenti, hogy az eljárás még nem fejeződött be, amikor újra felhívható. A rekurzív eljárással szemben a különbség az, hogy a rekurzív eljárásban „programozott", hogy mikor történik az eljárás újra hívása, re-entrant eljárás esetén az esetleges újra hívás ideje a véletlentől függ. Ez utóbbi esetben az biztosítja, hogy a munkaterületek ne keveredjenek össze, hogy újabb belépés csak másik processzusból képzelhető el, és minden processzus saját vermet használ.

Rekurzív és re-entrant eljárások

Ha egy eljárásunk készítésekor betartjuk, hogy az eljárás a paramétereit a vermen keresztül kapja, kilépéskor visszaállítja a belépéskori regiszter tartalmakat – az esetleg eredményt tartalmazó regiszterek kivételével –, továbbá a fenti módon kialakított munkaterületet használ, akkor az eljárásunk rekurzív is lehet, és a többszöri belépést is lehetővé teszi (re-entrant).

Hanoi tornyai (5.40-42. ábra)



Feladat: tegyük át az összes korongot az 1. pálcáról a 2-ra úgy, hogy egyszerre csak egy korongot mozgathatunk, és kisebb korongra nem tehetünk nagyobbat

Hanoi tornyai (5.40-42. ábra)

Rekurzív eljárás, amely **`n`** korongot mozgat **`i`**-ről **`j`**-re:

```
public void towers (int n, int i, int j) {
  int k;

  if(n==1)
    System.out.println(
        "Korong mozgatása:"+i+"-ről"+j"-re");
  else {
    k=6-i-j;
    towers(n-1, i, k);
    towers(1, i, j);
    towers(n-1, k, j);
  }
}
```


~ 5.43. ábra A verem tartalma az eljárásban:

towers(n,i,j) ; c konvenció V: visszatérési cím, F: régi FP

Hívások:		j i n k	j i n k	j i n k
towers(3,1,3)		**3, 1, 3, V, F, k**		
towers(2,1,2)		3, 1, 3, V, F, **2, 2, 1, 2, V, F, k**		
towers(1,1,3)	**1 → 3**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, **3, 3, 1, 1, V, F, k**		
visszatérés után		3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3		
towers(1,1,2)	**1 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, **1, 1, V, F, k**		
towers(1,3,2)	**3 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 3, **1, V, F, k**		
towers(1,1,3)	**1 → 3**	**3, 1, 3, V, F, 2, 3, 1, 1, V, F, k**		
towers(2,2,3)		**3, 1, 3, V, F, 2, 3, 2, 2, V, F, k**		
towers(1,2,1)	**2 → 1**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, **1, 1, 2, 1, V, F, k**		
towers(1,2,3)	**2 → 3**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, **1, 3, 2, 1, V, F, k**		
towers(1,1,3)	**1 → 3**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, **1, 3, 1, 1, V, F, k**		

~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

Hívások:	j i n	k | j i n	k | j i n	k
towers(3,1,3)	3, 1, 3, V, F, k			

~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

Hívások:	j i n	k | j i n	k | j i n	k
towers(3,1,3)	3, 1, 3, V, F, k			
towers(2,1,2)	3, 1, 3, V, F, 2,	2, 1, 2, V, F, k		

~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

Hívások:	j i n	k | j i n	k | j i n	k
towers(3,1,3)	3, 1, 3, V, F, k			
towers(2,1,2)	3, 1, 3, V, F, 2, 2, 1, 2, V, F, k			
towers(1,1,3) 1 → 3	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3,	3, 1, 1, V, F, k		

~ 5.43. ábra A verem tartalma az eljárásban:

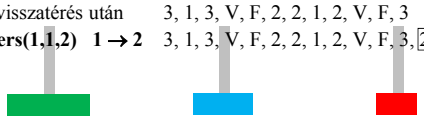
V: visszatérési cím, F: régi FP

Hívások:	j i n	k | j i n	k | j i n	k
towers(3,1,3)	3, 1, 3, V, F, k			
towers(2,1,2)	3, 1, 3, V, F, 2, 2, 1, 2, V, F, k			
towers(1,1,3) 1 → 3	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 3, 1, 1, V, F, k			
visszatérés után	3, 1, 3, V, F, 2,	2, 1, 2, V, F, 3		

~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

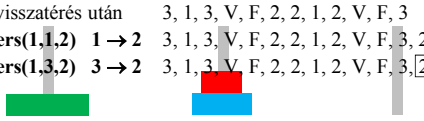
Hívások:	j i n	k | j i n	k | j i n	k
towers(3,1,3)	3, 1, 3, V, F, k			
towers(2,1,2)	3, 1, 3, V, F, 2, 2, 1, 2, V, F, k			
towers(1,1,3) 1 → 3	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 3, 1, 1, V, F, k			
visszatérés után	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3			
towers(1,1,2) 1 → 2	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3,	2, 1, 1, V, F, k		



~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

Hívások:	j i n	k | j i n	k | j i n	k
towers(3,1,3)	3, 1, 3, V, F, k			
towers(2,1,2)	3, 1, 3, V, F, 2, 2, 1, 2, V, F, k			
towers(1,1,3) 1 → 3	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 3, 1, 1, V, F, k			
visszatérés után	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3			
towers(1,1,2) 1 → 2	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 1, 1, V, F, k			
towers(1,3,2) 3 → 2	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3,	2, 3, 1, V, F, k		



~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

Hívások:	j i n k	j i n k	j i n k
towers(3,1,3)	3, 1, 3, V, F, k		
towers(2,1,2)	3, 1, 3, V, F, 2, 2, 1, 2, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 3, 1, 1, V, F, k		
visszatérés után	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3		
towers(1,1,2) **1 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 1, 1, V, F, k		
towers(1,3,2) **3 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 3, 1, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 3, 1, 1, V, F, k		

~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

Hívások:	j i n k	j i n k	j i n k
towers(3,1,3)	3, 1, 3, V, F, k		
towers(2,1,2)	3, 1, 3, V, F, 2, 2, 1, 2, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 3, 1, 1, V, F, k		
visszatérés után	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3		
towers(1,1,2) **1 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 1, 1, V, F, k		
towers(1,3,2) **3 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 3, 1, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 3, 1, 1, V, F, k		
towers(2,2,3)	3, 1, 3, V, F, 2, 3, 2, 2, V, F, k		

~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

Hívások:	j i n k	j i n k	j i n k
towers(3,1,3)	3, 1, 3, V, F, k		
towers(2,1,2)	3, 1, 3, V, F, 2, 2, 1, 2, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 3, 1, 1, V, F, k		
visszatérés után	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3		
towers(1,1,2) **1 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 1, 1, V, F, k		
towers(1,3,2) **3 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 3, 1, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 3, 1, 1, V, F, k		
towers(2,2,3)	3, 1, 3, V, F, 2, 3, 2, 2, V, F, k		
towers(1,2,1) **2 → 1**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, 1, 1, 2, 1, V, F, k		

~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

Hívások:	j i n k	j i n k	j i n k
towers(3,1,3)	3, 1, 3, V, F, k		
towers(2,1,2)	3, 1, 3, V, F, 2, 2, 1, 2, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 3, 1, 1, V, F, k		
visszatérés után	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3		
towers(1,1,2) **1 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 1, 1, V, F, k		
towers(1,3,2) **3 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 3, 1, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 3, 1, 1, V, F, k		
towers(2,2,3)	3, 1, 3, V, F, 2, 3, 2, 2, V, F, k		
towers(1,2,1) **2 → 1**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, 1, 1, 2, 1, V, F, k		
towers(1,2,3) **2 → 3**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, 1, 3, 2, 1, V, F, k		

~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

Hívások:	j i n k	j i n k	j i n k
towers(3,1,3)	3, 1, 3, V, F, k		
towers(2,1,2)	3, 1, 3, V, F, 2, 2, 1, 2, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 3, 1, 1, V, F, k		
visszatérés után	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3		
towers(1,1,2) **1 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 1, 1, V, F, k		
towers(1,3,2) **3 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 3, 1, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 3, 1, 1, V, F, k		
towers(2,2,3)	3, 1, 3, V, F, 2, 3, 2, 2, V, F, k		
towers(1,2,1) **2 → 1**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, 1, 1, 2, 1, V, F, k		
towers(1,2,3) **2 → 3**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, 1, 3, 2, 1, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, 1, 3, 1, 1, V, F, k		

~ 5.43. ábra A verem tartalma az eljárásban:

V: visszatérési cím, F: régi FP

Hívások:	j i n k	j i n k	j i n k
towers(3,1,3)	3, 1, 3, V, F, k		
towers(2,1,2)	3, 1, 3, V, F, 2, 2, 1, 2, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 3, 1, 1, V, F, k		
visszatérés után	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3		
towers(1,1,2) **1 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 1, 1, V, F, k		
towers(1,3,2) **3 → 2**	3, 1, 3, V, F, 2, 2, 1, 2, V, F, 3, 2, 3, 1, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 3, 1, 1, V, F, k		
towers(2,2,3)	3, 1, 3, V, F, 2, 3, 2, 2, V, F, k		
towers(1,2,1) **2 → 1**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, 1, 1, 2, 1, V, F, k		
towers(1,2,3) **2 → 3**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, 1, 3, 2, 1, V, F, k		
towers(1,1,3) **1 → 3**	3, 1, 3, V, F, 2, 3, 2, 2, V, F, 1, 3, 1, 1, V, F, k		
visszatérések után			

Rekurzív eljárások megvalósításához veremre van szükség. Minden hívás esetén az eljárás paramétereit a verembe kell tenni, és ott kell elhelyezni a lokális változókat is!

Eljárás prológus: a régi verem keret mutató (**FP**) elmentése, új verem keret mutató megadása, és a verem mutató (**SP**) növelése, hogy legyen hely a veremben a lokális változók számára.

Eljárás epilógus: visszatéréskor a verem kitakarítása.

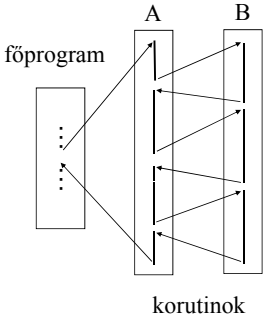
A Hanoi tornyai probléma megoldása

5.47. ábra: **Pentium 4** program.

5.48. ábra: **UltraSPARC III** program.
Eltolás rés!

Vezérlési folyamat

- **Szekvenciális vezérlés (5.39. ábra)**
- **Elágazás.**
- **Eljárás: 5.44. ábra.**
- **Megszakítások.**
- **Csapdák.**
- **Korutinok: 5.45. ábra.** Párhuzamos feldolgozás szimulálására alkalmas egy **CPU**-s gépen.



goto helyett jobb a ciklus vagy az eljárás alkalmazása!

A Pentium 4 utasításai

- Egész utasítások legnagyobb része: **5.34. ábra**.
- Egyéb utasítások (pl. lebegőpontosak).

Az UltraSPARC III utasításai

Összes egész utasítás: **5.35. ábra**.
A utasításnévben **CC**: beállítja a feltételkódot.
ADD, **ADDC**, **ADDCC**, **ADDCCC** utasítások.
Szimulált utasítások (**5.36. ábra**), pl.:

```
MOV SRC,DST  ≡  OR SRC,G0,DST
```

A 8051 utasításai (5.37. ábra)

Bit utasítások, pl. a 43. bit 1-re állítása:

```
SETB 43
```


A feltételes ugró utasítások eldugaszolják a csővezetéket

- Feltételes végrehajtás
- Predikáció

Feltételes végrehajtás (5.51-52. ábra):

C pr. rész	Általános assembly	Feltételes végrehajtás
if(R1 == 0) **R2 = R3;**	**CMP R1, 0** **BNE L1** **MOV R2, R3** **L1: …**	**CMOVZ R2, R3, R1**

CMOVZ R2, R3, R1 csak akkor hajtja végre **R2 = R3** -t, ha **R1= 0**.

Feltételes végrehajtás (5.51-52. ábra):

Hasonlóan:

C pr. rész	Általános assembly	Feltételes végrehajtás

if(R1 == 0) {
  R2 = R3;
  R4 = R5;
} else {
  R6 = R7;
  R8 = R9;
}
	CMP R1, 0
	BNE L1
	MOV R2, R3
	MOV R4, R5
	BR L2
	L1: MOV R6, R7
	MOV R8, R9
	L2: …
		CMOVZ R2, R3, R1
		CMOVZ R4, R5, R1
		CMOVN R6, R7, R1
		CMOVN R8, R9, R1

CMOVN R6, R7, R1 csak akkor hajtja végre **R6 = R7** -et, ha **R1≠ 0**.

Máté: Architektúrák 12. előadás 43

Predikáció, IA – 64 (5. 53. ábra)

64 predikátum regiszter: 1 bites regiszterek, többnyire párban. Az **IA – 64** minden utasítása predikátumos. **CMPEQ R1, R2, P4** beállítja **P4**-et és törli **P5**-öt, ha **R1 = R2**, különben **P5**-öt állítja be és **P4**-et törli.

if(R1 == R2)
  R3 = R4 + R5;
else
  R6 = R4 – R5;
	CMP R1, R2
	BNE L1
	MOV R3, R4
	ADD R3, R5
	BR L2
	L1: MOV R6, R4
	SUB R6, R5
	L2: …
		CMPEQ R1, R2, P4
		<P4>ADD R3, R4, R5
		<P5>SUB R6, R4, R5

Máté: Architektúrák 12. előadás 44

Feladatok

Milyen vezérlő regiszterei vannak egy terminálnak?

Mire szolgál a terminál billentyűzet puffer regisztere?

Hogyan történik a programozott **B/K**?

Mit nevezünk pollozásos technikának?

Mire használható a **DMA**?

Hogy működik a **DMA**?

Milyen regiszterei vannak a **DMA**-nak?

Máté: Architektúrák 6. előadás 45

Feladatok

Mit nevezünk program megszakításnak?

Mi a megszakítás kezelő?

Hogyan történhet a nyomtatás szervezése megszakítás segítségével?

Megszakítható-e a megszakítás kezelő?

Mi a csapda?

Mi a különbség a csapda és a megszakítás között?

Hogy működik a 8259A megszakítás vezérlő lapka?

Milyen hardver és milyen szoftver tevékenységek tartoznak a megszakításhoz?

Mit jelent az átlátszóság megszakítás esetén?

Máté: Architektúrák 6. előadás 46

Feladatok

Mi a szekvenciális vezérlés?

Mi az eljárás?

Mi a lokális adat terület?

Hogy alakíthatunk ki lokális adat területet?

Mi a rekurzív eljárás?

Mi a re-entrant eljárás?

Mondjon példát rekurzív eljárással megoldható problémára!

Hogy oldható meg a Hanoi tornyai probléma?

Máté: Architektúrák 11. előadás 47

Feladatok

Mi az eljárás prológus?

Mi az eljárás epilógus?

Mit nevezünk korutinnak (társrutin, coroutine)?

Mi az eltolás rés?

Mi a különbség az **UltraSPARC III ADD**, **ADDC**, **ADDCC** és **ADDCCC** utasításai között?

Mit értünk feltételes végrehajtáson?

Mi a feltételes végrehajtás előnye?

Mit értünk predikáción?

Hogy küszöböli ki a feltételes végrehajtás és a predikáció a csővezeték elakadását? Jelent-e ez késleltetést a program futásában?

Máté: Architektúrák 12. előadás 48

Az előadáshoz kapcsolódó

Fontosabb tételek

Programozott és megszakítás vezérelt I/O. DMA

Vezérlési folyamat. Szekvenciális vezérlés, elágazás, ciklus szervezés, eljárás, rekurzív eljárás, megszakítás, csapda. Korutinok

Feltételes végrehajtás, predikáció