# **Információfeldolgozási modellek**

k1) Hogyan írható le egy program a vezérlésáramlásos, az adatáramlásos, és az igényvezérelt modellben?  
k2) Hogyan támogatja a feladatban rejlő párhuzamosítási lehetőségek automatikus felderítését és kiaknázását a vezérlésáramlásos, az adatáramlásos, és az igényvezérelt modell?  
k3) Mikor hajtható végre egy utasítás a vezérlésáramlásos, az adatáramlásos, és az igényvezérelt modellben?  
n1) Mutassa be és hasonlítsa össze a 3 tanult információfeldolgozási modellt!

|  |  |  |  |
| --- | --- | --- | --- |
|  | **vezérlésáramlásos** | **adatáramlásos** | **igényvezérelt** |
| **felhasználás** | egyprocesszoros rendszerek, processzorok interfésze, szekvenciális programok | out-of-order támogató processzorok belső felépítése, táblázatkezelés | funkcionális programozási nyelvek |
| **leírás** | folyamatábra | precedenciagráf | redukciós gráf |
| **utasítás végrehajtás ideje** | amikor odaér a vezérlő token | amikor minden operandus rendelkezésre áll | amikor az utasítás eredményére szükség van |
| **párhuzamosítás támogatás** | manuális (Fork/Join primitívek használatával) | automatikus (felderíti azokat az utasításokat, amiknek minden operandusa rendelkezésre áll) | automatikus (felderíti az eredményt adó utasítás operandusait) |

# **A vezérlésáramlásos modell megvalósítása**

k5) Soroljon fel 4 címzési módot!

regiszter: R1 <- R2 + R3  
közvetlen konstans: R1 <- R2 + 12  
direkt: R1 <- R2 + MEM[12]  
regiszter indirekt: R1 <- R2 + MEM[R3]

k6) Mi a különbség a feltétel kódra és a feltétel regiszterre alapozott feltételes ugró utasítások között?

feltételkódnál: az eredmény egy flaget billent be  
feltételregiszternél: az eredmény regiszterben tárolódik

k7) Hogyan lehet az utasításkészleteket bájtsorrend alapján csoportosítani?

little-endian, big-endian, bi-endian

k8) Mit jelent az utasításkészletek ortogonalitása?

minden utasításban minden címzési mód használható

k1) Mi a Neumann architektúra szűk keresztmetszete?  
k2) Mi a Neumann és a Harvard architektúrák közötti alapvető különbség?  
n1) Ismertesse és hasonlítsa össze a Neumann, a Harvard, és a módosított Harvard architektúrát!

CPU

MEM

read, write

Neumann:

CPU

INSTR  
MEM

read, write

DATA  
MEM

read

klasszikus Harvard:

módosított Harvard:

CPU

INSTR  
MEM

read, write

DATA  
MEM

read, write

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Neumann** | **klasszikus Harvard** | **módosított Harvard** |
| **újdonság elődeihez képest** | a programot már nem mechanikus kapcsolókkal állítgatjuk, hanem a memóriában tároljuk | a memória kettéválasztása adat- és utasításmemóriára | az utasításmemória írhatóvá tétele |
| **felhasználás** | PC | mikrokontroller, jelfeldolgozó processzor | mikrokontroller |
| **önmódosító program lehetséges** | igen | nem | igen |

A Neumann architektúra hátránya, hogy a CPU sebességének növekedésével sem a memória, sem a buszok sebességnövekedése nem tud lépést tartani.  
A Neumann architektúra szűk keresztmetszetének nevezzük a limitált adatátviteli sebességet a memória és a CPU között.  
A CPU-nak egyre többször kell bevárnia a lassú memóriát.

A Harvard architektúra előnyei:  
A két memória egyidejűleg képes műveletet végezni.  
A CPU különálló interfésszel rendelkezik a két memória felé, amik különböző technológiájúak is lehetnek.

k3) Mi az előnye a fix és a változó hosszúságú utasításkódolásnak?  
k4) Soroljon fel 5 utasításfajtát/típust!  
n2)Mutassa be az utasítások felépítését, kódolását! Hogy lehet az utasítások operandusainak a számát csökkenteni? Milyen utasításfajtákat/típusokat ismer? Mindegyikre adjon példát is!

utasítások felépítése:

utasítás kódja

operandusok címei

eredmény címe

következő utasítás címe

utasítások kódolása:

változó:  
 helytakarékosabb

fix:  
 egyszerűbb a CPU felépítése

utasítás operandusainak számának csökkentése:

3 operanduscím használata:  
 R1 <- R2 + R3

2 operanduscím használata

R1 <- R1 + R2

1 operanduscím használata: akkumulátor segítségével

ADD R1

utasításfajták:  
 adatmozgatás: R1 <- R2  
 aritmetikai és logikai műveletek: R1 <- R2 + R3  
 vezérlésátadó utasítások: JUMP 34  
 veremkezelő utasítások: PUSH R1  
 I/O műveletek: IN R1 <- 42  
 transzcendens fv-k: R2 <- SIN R1

k9) Mi motiválta a CISC tervezési stratégiát?  
k10) Sorolja fel a CISC számítógépek jellemzőit!  
k11) Sorolja fel a RISC számítógépek jellemzőit!  
k12) Mi az előnye a CISC és a RISC tervezési stratégiának?  
n3) Ismertesse a CISC és a RISC processzor tervezési stratégiákat, sorolja fel ezek jellemzőit, előnyeit, hátrányait!

|  |  |  |
| --- | --- | --- |
|  | **CISC** | **RISC** |
| **megjelenés** | 70-es évek | 80-as évek |
| **lényege** | összetett, változó hosszú utasítások | egyszerű, fix hosszú utasítások |
| **motiváció** | a memória lassú – összetett utasítások miatt ritkábban fordul a MEM-hoz a CPU  a memória drága – összetett utasítások miatt kevesebb MEM-át foglal a program  egyszerűbb fordítóprogram – összetett utasításokkal csökkentik a szemmantikai rést az assembly és a magasabb szintű nyelvek között | az egyszerű, fix hosszú utasítások miatt a processzorok belső felépítése egyszerűbb és átláthatóbb |
| **redundancia** | nagy | kicsi |
| **címzési mód** | sokféle | kevés |
| **utasításvégrehajtási idő** | változatos | egységes |
| **kódméret** | kicsi | nagy |
| **regiszterszám** | kevés | sok |
| **programozás nehézsége** | könnyebb | nehezebb |
| **mikroarchitektúra átláthatósága** | nehezebb | könnyebb |
| **órajel** | kevesebb | több |
| **fogyasztás** | több | kevesebb |

Perifériakezelés  
  
k1) Miért lehet előnyösebb a memóriára leképzett perifériakezelés alkalmazása akkor is, ha vannak külön I/O utasítások?

a memory mapped technika ott is terjed, ahol vannak külön I/O utasítások

ennek oka, a módszer előnyeiben rejlik:

-nincs szükség I/O utasításokra -> egyszerűbb processzor

-a memóriaműveletek rugalmasságával lehet a processzort kezelni

k7) Foglalja össze a perifériakezelésre használatos pont-pont és busz alapú összeköttetések előnyeithátrányait!

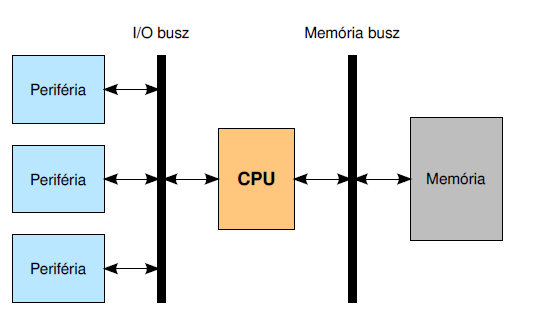
|  |  |
| --- | --- |
| **pont-pont** | **busz** |
| gyorsabb | lassabb |
| sok perifériánál sincs torlódás | sok perifériánál nagy az esély a torlódásnak |
| drágább | olcsóbb |

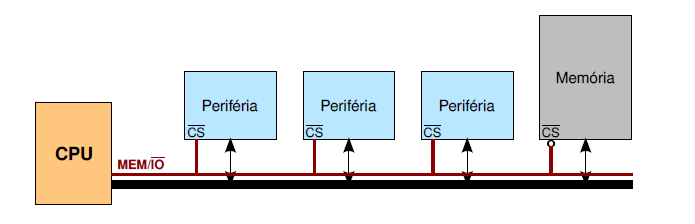
k9) Mi a különbség a szinkron és az aszinkron buszok között? Melyik megoldással lehet nagyobb

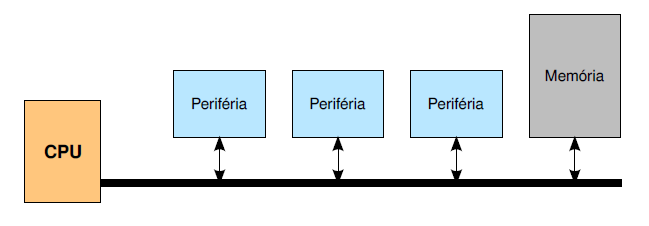
adatátviteli sebességet elérni? Melyik megoldással lehet nagyobb fizikai távolságot áthidalni?

|  |  |  |
| --- | --- | --- |
|  | **szinkron** | **aszinkron** |
| **lényege** | órajel ütemében történnek a dolgok | nincs sebességkorlát, viszont több vezérlőjelre van szükség |
| **sebesség** | gyorsabb | lassabb, a járulékos kommunikáció miatt |
| **fizikai távolságok** | kedvezőtlenebb, mert az órajelnek a busz teljes hosszát be kell járnia | kedvezőbb |

n1) Mi a különbség a memóriára leképzett és a szeparált I/O utasításokkal megvalósított perifériakezelés között? Mit jelent, és hogy működik a multiplexált memória- és I/O busz?

**szeparált I/O utasításokkal megvalósított perifériakezelés**  
szeparált busszal:  
a CPU külön buszon kommunikál a memóriával és a perifériákkal  
******

**szeparált I/O utasításokkal megvalósított perifériakezelés**  
multiplexált busszal  
a CPU azonos buszon kommunikál a memóriával és a perifériákkal, de azokat egy vezérlőjellel megkülönbözteti (MEM//IO) ****

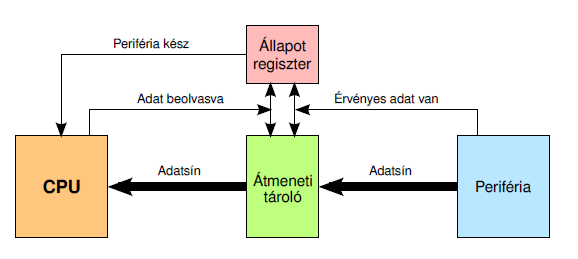
memory mapped perifériakezelés  


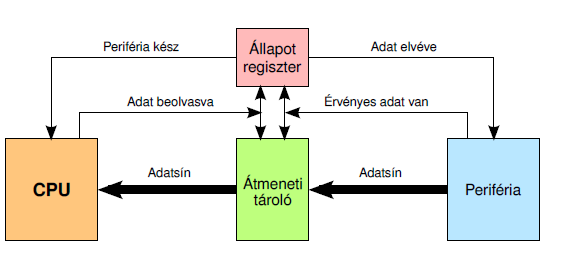
A memory mapped perifériakezelésnél azonos,  
amíg a szeparált I/O utasításokkal megvalósított perifériakezelésnél különböző utasításokkal lehet kezelni a memóriát és a periférifériákat.

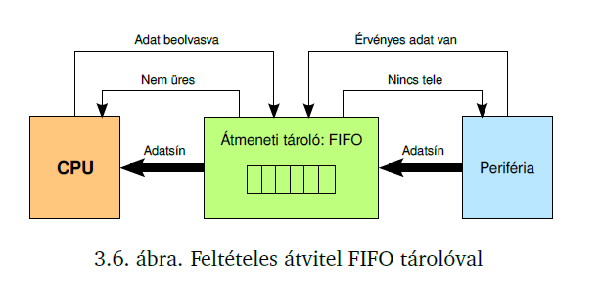
k2) A processzor és a periféria közötti feltétel nélküli adatátvitel során milyen hibák adódhatnak a

forgalomszabályozás hiánya miatt?  
k3) Mikor célszerű FIFO sor alapú átmeneti tárolót alkalmazni a processzor és a periféria közötti feltételes adatátvitel során?  
n2) Mutassa be és hasonlítsa össze a perifériakezelés során használatos forgalomszabályozási

megoldásokat!

**egyoldali** feltételes adatátvitel ******

**kétoldali** feltételes adatátvitel ******

**FIFO-val történő** feltételes adatátvitel  
******

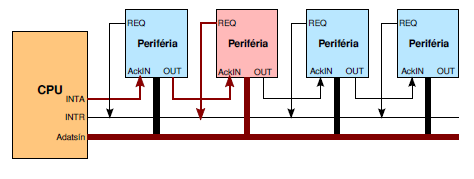
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | feltétel nélküli adatátvitel | feltételes adatátvitel | | |
| egyoldali | kétoldali | FIFO-val történő |
| lényege | nincs forgalomszabályozás | csal a CPU figyeli az állapotregisztert | a CPU és a PER is figyeli az állapotregisztert | CPU és periféria is figyeli a FIFO-t |
| hátrányok | adategymásrafutás, adathiány | adategymásrafutás | sokszor várakozásra kényszerül | a kétoldalinál kevesebbet, de a többihez képest többet kényszerül várakozásra |
| használata | kapcsoló, LED | hálózati kártya |  | ha az adatforgalom vagy a felek rendelkezésre állása ingadozó |

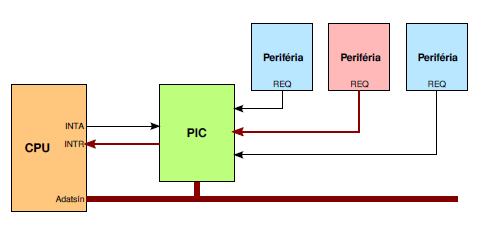
k4) Milyen előnyökkel jár, ha a periféria jelzéseit polling helyett interrupt segítségével kezeljük?  
n3) Mutassa be és hasonlítsa össze a perifériák jelzéseinek polling és interrupt alapú kezelését! Milyen megoldásokat ismer az interruptot kiváltó periféria azonosítására?

|  |  |  |
| --- | --- | --- |
|  | polling | interrupt |
| lényege | jelzés periodikus ellenőrzése futó programban | jelzés hatására futó program megszakítása és a jelzés lekezelése |
| előny | bármennyi eszköz kezelhető ilyen módon | kevesebb órajelciklust használ |
| hátrány | több órajelciklust használ | kevés eszköz kezelhető ilyen módon |

interruptot kiváltó periféria azonosítása:

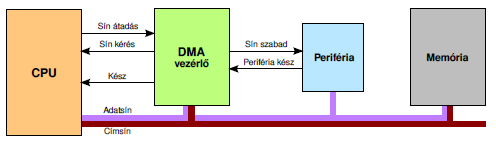
**polling:** minden perifériát körbekérdez, hogy ki küldte az INTet

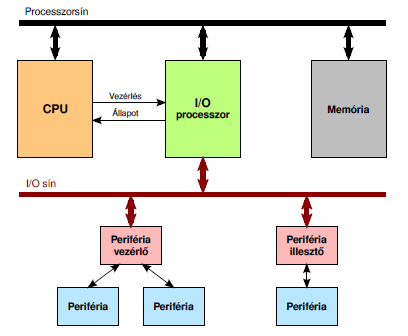
**daisy chaining:** az adatsínen talált azonosítóból tudja, hogy ki küldte az INTet  
****

**interrupt vezérlő (PIC) használata:** az INT kérések a PIC-be futnak be, ami kezeli őket

k5) Mikor előnyösebb a DMA alapú adatátvitel, mint a processzor felügyeletével végzett adatátvitel?  
k6) Mi a különbség a DMA vezérlő és az I/O processzor között?  
n4) Ismertesse a perifériakezelés során a processzor tehermentesítésére tanult megoldásokat!

Nagy mennyiségű adat mozgatásával járó perifériaműveleteknél a DMA vagy I/O processzor tehermentesíti a CPU-t a memória és a perifériák közötti közvetlen adatátvitel által.

**DMA:**   
  
A CPU számára a DMA könnyebbséget jelent azáltal, hogy az adatátvitellel nem adategységenként, hanem adatátviteli blokkonként kell foglalkoznia.

**I/O processzor:**  


a DMA koncepció továbbfejlesztése azáltal, hogy saját utasításkészlettel rendelkezik  
következmények:  
-eltakarja a perifériák változatos igényeit a CPU elől, így a CPU-nak nem kell ismernie őket  
-a perifériáknak nem kell közvetlenül a processzorsínhez kapcsolódniuk, elég, ha csak az I/O processzorral vannak összeköttetésben. Például I/O sín segítségével:  
Itt a perifériavezérlő/illesztő feladata, hogy közös kommunikációs protokollra fordítsa le a különféle perifériák nyelvét.  
periféria vezérlő: több perifériát tud kezelni  
periféria illesztő: egy perifériát tud kezelni

k8) Mi a különbség a centralizált és az elosztott arbitráció között? Melyiknek jobb a hibatűrő képessége?  
n5) Ismertesse az I/O sínen használatos arbitrációs eljárásokat!

A masterek versenyeznek a buszért, az, hogy ki kapja azt meg az arbitráció dönti el.

|  |  |  |  |
| --- | --- | --- | --- |
| **centralizált arbitráció** | | **elosztott arbitráció** | |
| arbiter segítségével történik | | nincs arbiter ezért kevésbé érzékeny a meghibásodásokra | |
| daisy chaining | párhuzamos | önkiválasztó | ütközésdetektáláson alapuló |
| rugalmatlan priorizálás | rugalmas priorizálás | mindenki látja mindenki buszfoglalási kéréseit és mindenki tisztában van mindenki prioritásával | nincs arbitráció, a masterek hallgatóznak a buszon, hogy a saját „adásuk” van-e rajta, ha igen, használja azt, ha nem, kis idő múlva próbálkozik újra |
|  |  |

k10) A perifériakezelés szempontjából mik a több buszos rendszerek előnyei az egybuszoshoz képest? Mi a híd szerepe a több buszos rendszerekben?  
k11) A perifériakezelés szempontjából mik a híd alapú rendszerek előnyei az egy-, és több-buszos

rendszerekhez képest? Mi az északi és déli híd szerepe a híd alapú rendszerekben?  
n6) Mutassa be és hasonlítsa össze a perifériakezelésre használatos egy-, több-buszos, ill. híd alapú rendszereket!

|  |  |  |
| --- | --- | --- |
| **egybuszos** | **több buszos** | **híd alapú** |
|  |  |  |
| a perifériák és a memóriák a processzor órajelével megegyező buszon vannak | I/o sín: állandó, szabványos sebességű sín  híd: átjárás a sínek között | északi híd: memória és grafikus megjelenítő buszaiért felelős  déli híd: I/O buszokért felelős |
| egyszerű megvalósítani | orvosolta az egybuszos rendszerek hátrányait | orvosolta a több buszos rendszerek hátráyait |
| magasabb CPU órajel -> magasabb busz órajel -> drágább perifériák, korlátozott fizikai távolság | a memóriák és a perifériák egyre változatosabbá vállnak |  |

sz1) Egy periféria x maximális sebességű interfésszel rendelkezik, melyen egyoldali feltételes adatátvitellel időnként y méretű blokkokat visz át. Az adat rendelkezésre állását a "Periféria kész" jelzés beállításával jelzi a processzor felé. Mekkora legyen a processzor polling periódusa, hogy ne legyen adatvesztés? Ha egy lekérdezés z órajelciklust vesz igénybe, és a processzor w órajelfrekvenciával működik, az órajelek hány százalékát viszi el a periféria jelzésének ciklikus lekérdezgetése? Mennyivel hatékonyabb a jelzés interrupt alapú feldolgozása, ha a periféria az idő q %-ában küld ténylegesen adatot?

a CPU polling periodusa =

a polling az órajel -át veszi el  
az interrupt az órajel -át veszi el  
ez -al hatékonyabb megoldás

# PCI/PCIExpress gyakorlat

k2) Mi az a PCI function?  
logikai periféria a PCI eszközön

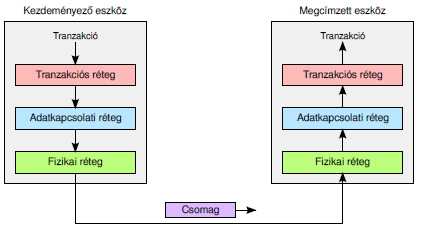
k3) Mi a Host/PCI és a PCI/PCI híd szerepe?  
Host/PCI:  
 CPU,MEM kérések ------ PCI busz összekapcsolása

PCI/PCI:  
 PCI busz ------ PCI busz összekapcsolása

k1) Milyen tranzakciós modelleket támogat a PCI busz?  
k4) Milyen forgalomszabályozási megoldást használ a PCI, és milyet a PCI Express?  
k5) Milyen megoldást használ a PCI arbitrációra?  
k6) Milyen módokon kérhet megszakítást egy PCI periféria?  
n1) Ismertesse, hogy a PCI busz hogyan valósítja meg a forgalomszabályozást, az arbitrációt és a megszakításkezelést! Milyen módon lehet a processzort tehermentesíteni az adatátvitel során?  
n2) Mutassa be, hogy hogyan történik az adatátvitel a PCI Express alapú rendszerekben a processzor és a perifériák között! Hogyan valósítja meg a PCI Express a forgalomszabályozást?

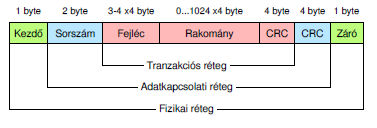
|  |  |  |
| --- | --- | --- |
|  | **PCI** | **PCI Express** |
| **forgalomszabályozás** | IRDY#: az Initiator jelzi, hogy kész a következő adat átvitelére TRDY#: a Target jelzi, hogy kész a következő adat átvitelére | -a beérkező csomagok feldolgozásukig egy bufferben tárolódnak -ha a következő csomag nem fér be a bufferbe akkor vár a küldéssel, egyébként a bufferbe teszi azt |
| **arbitráció** | centralizál párhuzamos arbitráció:    rejtett arbitráció: az arbitráció tranzakció közben is történhet | nincs |
| **megszakításkezelés** | -üzenettel jelzett interrupt (MSI)  -4 interruptvezeték egyikével | -üzenettel jelzett interrupt (MSI)  -emulálja a PCI-nál használt 4 interruptvezeték hatását |
| **támogatott tranzakciós modellek** | Programozott I/O, DMA, Peer-to-peer | |
| **CPU tehermentesítése** | DMA és Peer-to-peer tranzakciós modellekkel | |

A PCIe-nél az adatátvitel sorosan, bitenként történik, mert a PCI párhuzamos adatátvitelénél a vezetékek tulajdonságai miatt nem lehet magas órajelet elérni.

Egy tranzakció eljuttatása egyik eszközből a másikba csomagonként történik:

tranzakciós réteg: becsomagolás  
adatkapcsolati réteg: hibamentességért felelős

fizikai réteg: a csomag átvitele a soros érpáron



# Memóriakezelés

k2) Mik a virtuális tárkezelésben a nagy lapméret előnyei és hátrányai?

előny:  
 kevesebb TLB hiba  
 diszk alapú háttértárnál a nagy lapok betöltése szinte ugyanannyi idő, mint a kis lapoké

hátrány:

nagyobb az esélye, több felesleges adat lesz a gyors elérésű memóriában

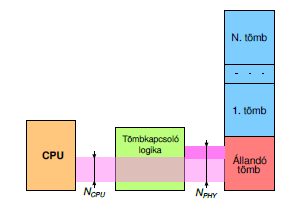
k9) Hogyan működik a szoftver menedzselt TLB?  
A laptábla bejárása és karbantartása nem a hardver, hanem az OS feladata.

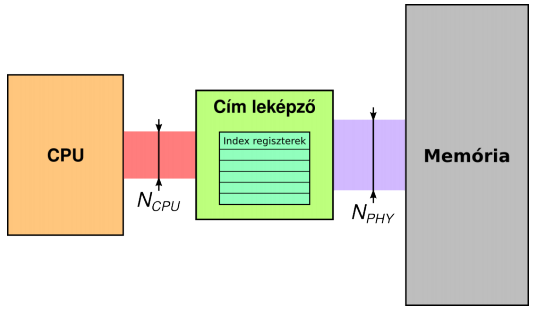
k7) Mi a cím leképző periféria szerepe az indexelt leképzésben?  
k8) Miért van szükség állandó memóriaterületre a tömbkapcsolásban és az indexelt leképzésben?

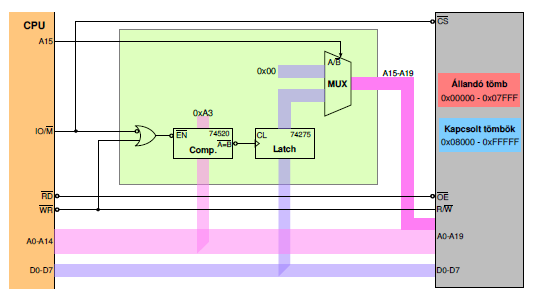
n1) Ismertesse a tömbkapcsolás blokksémáját, működését és adjon meg egy lehetséges implementációt!

n2) Ismertesse a tömbkapcsolást és az indexelt leképzést, majd hasonlítsa össze a kettőt!

probléma: túl kicsi a CPU címtartománya  
megoldás: tárbővítés (tömbkapcsolással vagy indexelt leképezéssel)

**a tömbkapcsolás blokksémája:  
**

**az indexelt leképezés blokksémája:  
**

**a tömbkapcsolás egy lehetséges implementációja:**

|  |  |  |
| --- | --- | --- |
|  | **tömbkapcsolás** | **indexelt leképezés** |
| **működés** | a memóriát címezhető méretű tömbökre osztjuk, ezeket ki-be kapcsolgatjuk | a memóriát egy címezhető méretű ablakkal indexeljük, eltolás által eszköze a címleképző periféria: a CPU által kiadott címek leképzése a fizikai memóriára |
| **komplexitás** | egyszerűbb | összetettebb |
| **sebesség** | gyors | gyors, de a tömbkapcsolásnál lassabb |
| **tömb/ablakméret** | azonos | azonos |
| **transzparens?** | nem | nem |
| **ablakátlapolás** | nincs | van |

Továbbá tárbővítésnél szükség van egy **állandó memóriaterületre**, azoknak az információknak a tárolására, amiknek állandóan elérhetőnek kell lenni.  
Ilyen például az interrupt vektortábla, mert interrupt bármikor bekövetkezhet.  
Továbbá praktikus módon ide elhelyezhetjük a stacket, vagy az operációs rendszer gyakran használt szubrutinjait.

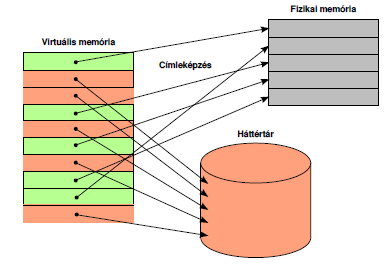
k1) Mi a TLB lefedettség? Miért elnyős a nagy TLB lefedettség?  
k5) Milyen információt kell tartalmaznia a laptábláknak a címfordítás támogatásához?  
n3) Ismertesse a virtuális memóriakezelés elvét, a címfordítás folyamatát és a TLB szerepét!

alapprobléma: túl nagy a CPU címtartománya  
megoldás: virtuális tárkezelés  
elv: a programok számára a CPU teljes címtartománya virtuálisan a rendelkezésükre áll, az pedig, hogy a virtuális memóriát hogyan kell leképezni a fizikai memóriára a CPU és az OS dolga.

címfordítás: a virtuális címek leképzése fizika címekre, eszköze az MMU

1. a virtuális címtartományt egyforma méretű lapokra partícionáljuk  
2. a fizikai memóriát ugyanekkora keretekre osztjuk  
lap/keretméret: 2Lalső L bit: eltolás  
felső bitek:  
 virtuális címnél: lapsorszám  
 fizikai címnél: keretsorszám

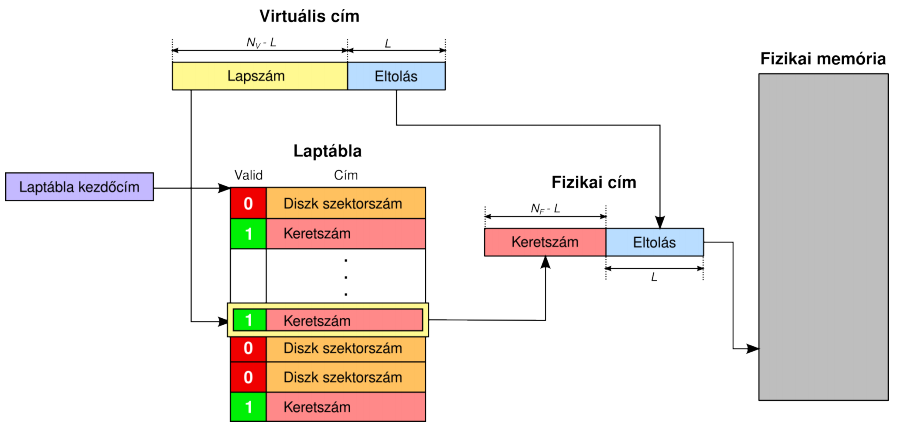
a lap ↔ keret összerendelést a laptábla tárolja  
tartalma:  
 lapsorszám  
 keretsorszám  
 védelmi információk  
 vezérlőbitek (valid, dirty, accessed)

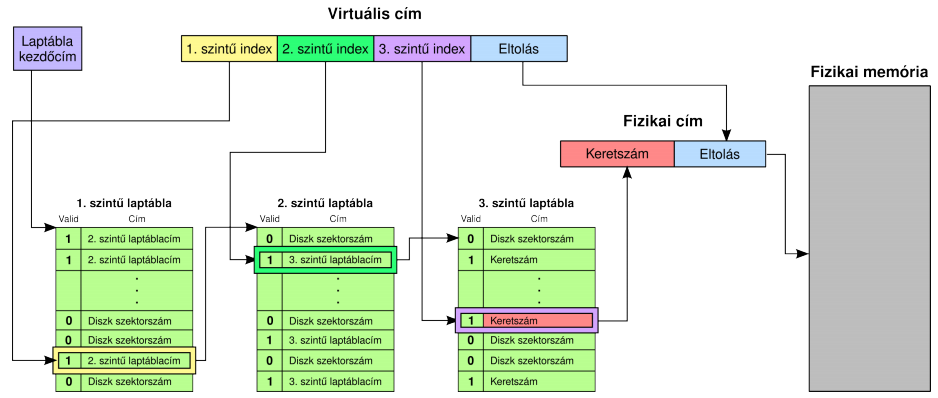
******

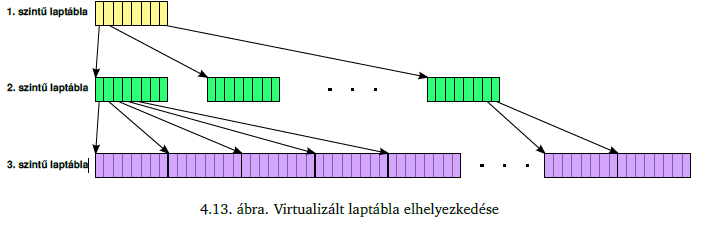
Címfordításnál a programban 1 memóriaművelet a valóságban 2 memóriaművelet:

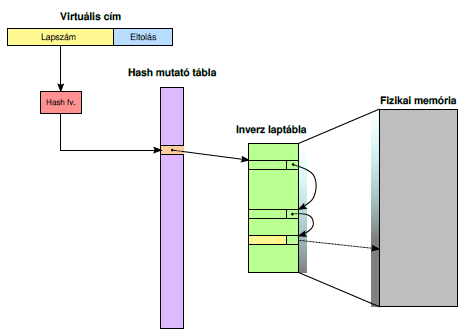
-laptábla segítségével a fizikai cím kiszámítása  
 -kívánt memóriaművelet végrehajtása  
megoldás: a leggyakrabban használt laptáblákat egy gyors elérésű cache memóriában tároljuk, ez a TLB  
a CPU először itt keresi a laptáblát, ha talál akkor jöhet a kívánt memóriaművelet  
tehát minnél nagyobb a TLB lefedettség annál ritkábban kell a memóriában lévő laptáblához fordulni

k4) Miért jobb az inverz laptábla, mint a klasszikus hierarchikus laptábla?  
k6) Mi a hash mutató tábla szerepe az inverz laptáblák esetében?  
n4) Ismertesse az egyszintű és a hierarchikus laptáblák működését, majd hasonlítsa össze a kettőt!  
n5) Ismertesse a klasszikus hierarchikus laptáblák és a virtualizált laptáblák működését, majd hasonlítsa össze a kettőt!  
n6) Ismertesse a klasszikus hierarchikus laptáblák és az inverz laptáblák működését, majd hasonlítsa össze a kettőt!

**egyszintű laptábla:**  


**hierarchikus laptábla:**  


**virtualizált laptábla:  
**

**inverz laptábla:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **egyszintű laptábla** | **hierarchikus laptábla** | **virtualizált laptábla** | **inverz laptábla** |
| **lényege** | a laptáblabejegyzések a fizikai memóriában folytonos tömbként vannak tárolva | a laptáblát is lapokra bontjuk a laptáblalapok elhelyezkedését egy másik laptáblában tároljuk, azokét egy harmadikban | a laptáblát a virtuális memóriában van tárolva | i. bejegyzés = i. keretben lévő lap száma elemszám = lapok száma |
| i. bejegyzés = i. laphoz tartozó keret elemszám = keretek száma | i. bejegyzés = i. laphoz tartozó keret elemszám = keretek száma | i. bejegyzés = i. laphoz tartozó keret elemszám = keretek száma |
| **memóriaműveletek száma** | 1 memóriaművelet | a laptábla mélységével megegyező mennyiségű memóriaművelet | 1. virtuális cím kiszámolása 2. ha TLB találat van: 1 memóriaművelet | 1. virtuális cím kiszámolása 2. ha TLB találat van: 1 memóriaművelet |
| 3. ha TLB hiba van: hierarchikus módon számolunk | 3. ba TLB hiba van: a virtuális cím lapszámából tartalom szerint keres, amit a hash mutató tábla tesz lehetővé |
| **lapok helye** | teljes laptábla a fizikai memóriában | ami fontos az van a fizikai memóriában, a többi a háttértáron | teljes laptábla a virtuális memóriában lapjai a TLB-re vagy hierarchikus módon vannak leképezve | a laptábla kicsi, elfér a fizikai memóriában, a hash mutató táblát is a fizikai memóriában tároljuk |

# Cache memória

k1) Ismertesse a tanult lokalitási elveket!

időbeli: ha egy adathoz hozzányúltunk, valószínűleg újra hozzá fogunk  
térbeli: ha egy adathoz hozzányúltunk, valószínűleg a környezetéhez is hozzá fogunk  
algoritmikus: sokszor kiszámítható, hogy milyen algoritmusokat használ a program, így az is, hogy milyen adathoz fogunk hozzányúlni

k2) Mi a lokalitási elvek szerepe a tárhierarchia hatékony működésében?  
Amelyik adat használva lesz a lokalitási elvek szerint azt az adatot minnél közelebb kell vinni a CPU-hoz.

k3) Ismertesse a cache megvalósítások csoportosítását címzési mód és menedzsment szerint!

címzési mód szerint:  
 transzparens: a chache és az operatív memória címzés szempontjából nem különül el  
 nem transzparens: a chache és az operatív memória címzés szempontjából nem elkülönül  
menedzsment szerint:   
 implicit: a chache tartalmát maga a chache vezérli  
 explicit: a chache tartalmát a program vagy az OS vezérli

k4) Mi a különbség a transzparens cache és a scratch-pad memória között?

transzparens cache transzparens, implicit  
scratch-pad memória nem transzparens, explicit

k5) Transzparens szervezésű, implicit menedzsmentet alkalmazó cache esetén milyen biteket/információkat kell tárolni minden egyes blokkra a hasznos adatokon kívül?

cache tag, valid bit, dirty bit

k10) Soroljon fel 4 cache blokk csere stratégiát és ismertesse a működésüket!  
• Random:  
 véletlenszerű választás

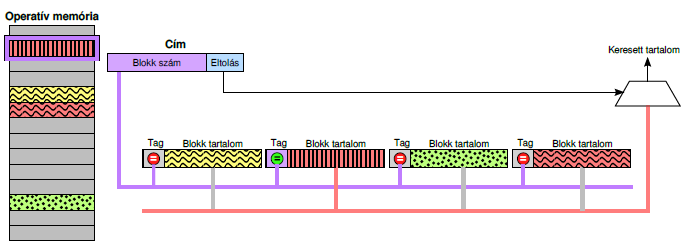
• LRU (least recently used): legrégebben használt

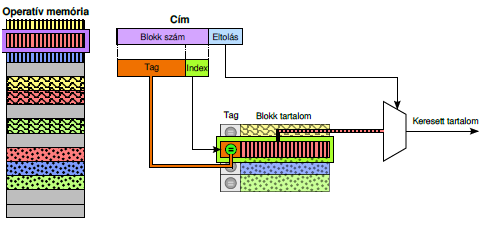
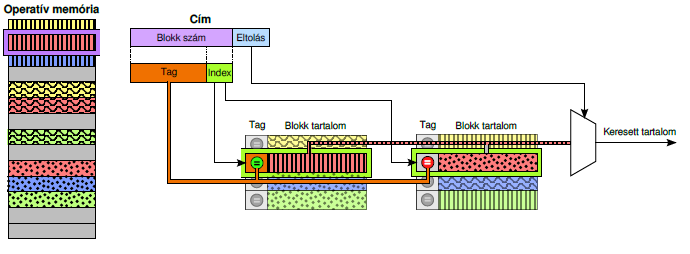
• NMRU (not most recently used): nem a legutóbb használt:  
 a legutóbb hivatkozott kivételével véletlenszerű választás

• LFU (least frequently used): legritkábban használt

k11) Ismertesse a write-through és write-back írási stratégiákat, és sorolja fel előnyös és hátrányos tulajdonságaikat!  
write-through: ha egy cache-ben lévő blokkra írnak, az rögtön bekerül az operatív memóriába is  
write-back: csak akkor kerül egy cache-ben lévő blokk az operatív memóriába, ha az kiürül a cacheből  
a write-back gyorsabb, de kevésbé koherens

k6) Mik a direkt leképzésen alapuló cache szervezés előnyei a teljesen asszociatív leképzéshez képest?  
n1) Ismertesse a teljesen asszociatív és a direkt leképzéses cache szervezés működését, előnyeit, hátrányait!  
n2) Ismertesse a direkt leképzéses és az n-utas asszociatív cache szervezés működését, előnyeit, hátrányait!

**teljesen asszociativ   
**

**direkt leképezés  
  
n-utas asszociatív   
**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **teljesen asszociativ** | **direkt leképezés** | **n-utas asszociatív** |
| versenyhelyzet | nincs, az operatív memória bármely blokkját a chache bármely blokkjába tehetjük chache tag: a blokk operatív memória beli sorszáma | minden operatív memóriabeli blokk csak egyetlen helyre kerülhet a chache-ben, amit a blokkszám dönt el, ezért versenyhelyzet alakulhat ki | a teljesen aszociatív és a direkt leképzést ötvözi azáltal, hogy az index több (n db) blokkra mutat, ahol megtalálhatjuk az adatot ezáltal a direkt leképzéshez képest kevesebb a versenyhelyzet |
| keresés | a hivatkozott cím blokkszámát az összes chache bejegyzés tagjával összehasonlítjuk | (1) indexelés: a megfelelő cache blokkot (2) komparálás, 1 db komparátorral | (1)indexelés: a megfelelő cache blokk halmazt (n blokkot) (2)komparálás, n db komparátorral |
| fogyasztás | nagy,  minden taghoz tartozik egy széles komparátor | kicsi, csak egy rövid komparátor működik | közepes,  n komparátor működik |
| sebesség | gyors, a keresés egy lépésben történik | lassabb a teljesen asszociativnál | lassabb a teljesen asszociativnál |

k7) Milyen előnyei vannak a tisztán virtuális címekre alapozott cache szervezésnek a tisztán fizikai címekre alapozott szervezéshez képest?  
n3) Írja le transzparens cache szervezés esetén a virtuális és fizikai címek alkalmazási lehetőségeit és sorolja fel ezek előnyeit, hátrányait!

|  |  |  |
| --- | --- | --- |
| **fizikailag indexelt cache fizikai taggel** | **virtuálisan indexelt cache fizikai taggel** | **virtuálisan indexelt cache virtuális taggel** |
|  |  |  |
| **1)címfordítás**  **2)cache indexelés**  **3)komparálás** | 1)egyidejűleg:  ● címfordítás  ● cache indexelés  2) komparálás | 1)cache indexelés 2) komparálás címfordítás csak cache hiba esetén kell |
| 3 lépés | 2 lépés | 2 lépés, egyszerűbb |
| -rövidebb a tag -ha egy keretet több lappal lehet elérni akkor sem lesz benne a chache-ben | -rövidebb a tag -ha egy keretet több lappal lehet elérni akkor sem lesz benne a chache-ben | -szélesebb a tag -ha egy keretet több lappal lehet elérni, többször lesz bent a cacheben |

k8) Mi a cache szemetelés jelensége, mi a cache szemét?

k9) Mit jelent a cache blokkok idő előtti betöltése, és milyen előnyökkel jár?

n4) Ismertesse a cache blokkok betöltésével kapcsolatos tartalom menedzsment feladatokat és azok tanult megoldásait (különös tekintettel a cache szemetelés elkerülésére és az idő előtti betöltés hatékony megvalósítására)!

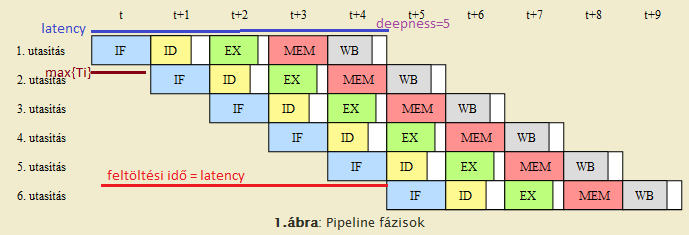
**mikor tegyünk be egy memória blokkot a cache-be?**•soha: cache szemetelés ellen véd  
• első hivatkozáskor: elsőre sokáig tart (memória → cache átvitel), de később gyors lesz  
• Idő előtti betöltés (prefetch): spekulációt igényel

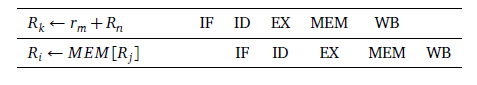
**cache szemetelés megelőzése:**  
cache szemét: az a blokk, ami a cacheebe kerülése és a cacheből való kirakása között egyszer sem lett meghivatkozva  
cache szemetelés pl. médialejátszás: kijátszik egy képet, soha többet nem kell  
elkerülhetjük a cache szemetelést:  
 speciális hardver utasításokkal  
 időben nem lokális viselkedés kiszűrésével

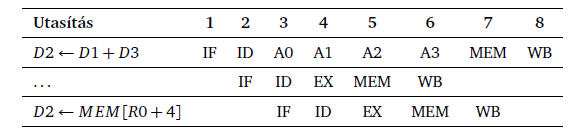
**Idő előtti betöltés (prefetch):**def: az adatok már akkor a cacheben vannak, mielőtt a program használná azokat  
előny: így nem kell megvárni a lassú memória → cache átvitelt  
explicit menedzsment: az alkalmazás javasolhatja az idő előtti betöltést  
implicit menedzsment: a CPU prefetch egysége heurisztikus algoritmusokkal végzi az idő előtti betöltést   
 •ha hivatkozás történik az i. blokkra (akár bent van a cache-ben, akár nincs), akkor betöltjük a cachebe az i+1. blokkot is   
 •ha hivatkozás történik az i. blokkra és az nincs bent a cache-ben, akkor az i. mellett betöltjük a cache-be az i+1. blokkot is

# Pipeline utasításfeldolgozás

k2) Mit jelent a pipeline késleltetése, átviteli sebessége, és feltöltési ideje?

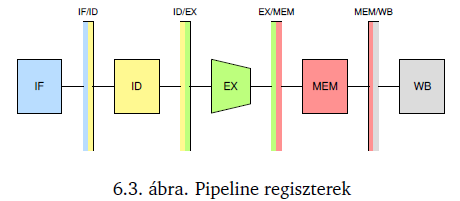
késleltetés (latency)  
az utasítások végrehajtásának elkezdése és befejezése között eltelt idő  
= deepness \* max{Ti}.  
  
átviteli sebessége (throughput):   
időegységenként befejezett utasítások száma  
= 1/ciklusidő ideális esetben.   
  
feltöltődési idő:   
az első utasítás betöltésétől a pipeline megteléséig eltelt idő  
=latency

k11) Adjon példát arra az esetre, amikor a pipeline feldolgozás során az azonnali kivételkezelés miatt a kivételek kezelési sorrendje nem felel meg az utasítássorrendnek!  
  
Ha az aritmetikai utasítás az EX fázisban vált ki kivételt (pl. túlcsordulást), a Load pedig az IF fázisban (pl. laphibát), és a kivételeket a kiváltásuk pillanatában kezelnék, akkor felborulna a sorrend (és sérülne a pontos kivételkezelés elve), hiszen a későbbi utasítás kivételét hamarabb kezelnénk, mint a korábbiét.

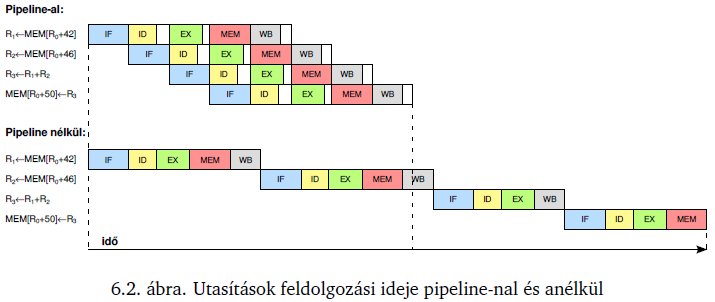
k12) Adjon példát arra az esetre, amikor a pipeline feldolgozás során az eltérő késleltetésű aritmetikai műveletek miatt a WAW függőség problémát okoz!  
  
Ha az első utasítás két lebegőpontos szám összeadása, a második pedig egy load művelet, és mind a 2 ugyanabba a regiszterbe ír. A második művelet eredményét várjuk a regiszterben, de mivel az első tovább tart, ezért végül az lesz benne.

k1) Milyen fázisokra osztható egy RISC utasítás feldolgozása? Mindegyik fázis feladatát fogalmazza meg egy mondatban!  
k3) Mi a pipeline regiszterek szerepe a pipeline feldolgozásban?  
n1) Ismertesse a pipeline utasításfeldolgozás elvét, az 5 fokozatú pipeline fázisainak feladatait, és a

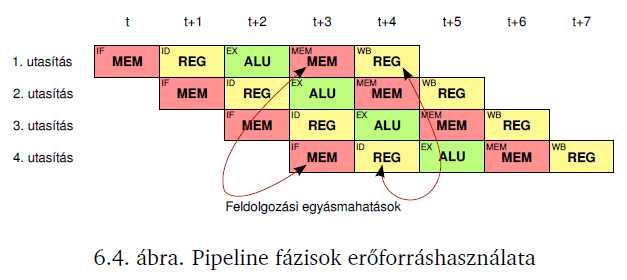
pipeline regiszterek szerepét! Egy egyszerű példával mutassa meg, hogy a pipeline segítségével tényleg gyorsabban végrehajtható egy utasítássorozat, mint anélkül!

**elv:**  
 Amikor egy utasítás feldolgozása a következő fázisba lép, az előző fázis erőforrásaival a következő utasításon is el lehet kezdeni a munkát.

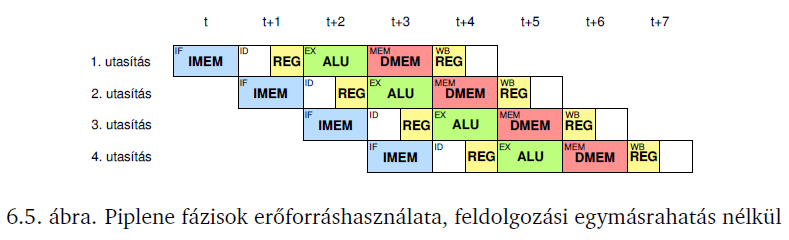
**fázisok feladatai, regiszterek** (def: biztosítják az információáramlást a fázisok között) **tartalma:**  
**IF:** (instruction fetch, IF)  
 utasítás lehívása a memóriából   
 utasításszámláló növelése  
IF/ID:  
 lehívott utasítás  
 utasításszámláló megnövelt értéke  
**ID:** (instruction decode/register fetch, ID)  
 vezérlőjelek dekódolása a fázisok számára  
 operandusok kiolvasása regiszterekből   
ID/EX:  
 a fázisok vezérlőjelei  
 operandusok  
**EX:** (execution, EX)  
 ID/EX-ben előírt művelet elvégzése az ID/EX-ben előírt operandusokkal  
 eszköze: ALU  
EX/MEM:  
 eredmény  
**MEM:** (memory access, MEM)  
 Load/Store utasítások esetén:  
 memóriaműveletek végrehajtása  
 egyéb utasítások esetén:  
 ez a fázis kihagyható  
MEM/WB:  
 az EX/MEM tartalma  
**WB:** (write-back cycle, WB)  
 regiszterfrissítés (ha nem kell regiszterbe írni, elhagyható)

**Az utasításfeldolgozás gyorsabb pipeline-nal, mint anélkül:**

k4) Az egymást követő utasítások között milyen adat-függőségeket ismer?  
k5) Mi a feldolgozási egymásrahatás, és hogy oldható fel a tanult 5 fokozatú pipeline-ban?  
k6) Mi a forwarding technika szerepe a pipeline utasításfeldolgozásban?  
k7) Soroljon fel 3 megoldást statikus elágazásbecslésre!  
k8) Mi a különbség a statikus és a dinamikus elágazásbecslés között?  
n2) Milyen egymásrahatások fordulhatnak elő az utasítások pipeline feldolgozása során? Javasoljon megoldást mindegyikre!

Feldolgozási egymásrahatás  
a processzor egyes erőforrásaira a pipeline több fázisának is szüksége van.  
megoldás:  
 szünet tartása, amíg erőforrás felszabadul.

típusai:  
**a) verseny a memóriáért:**megoldás:  
 az utasítás- és az adatmemória szétválasztása

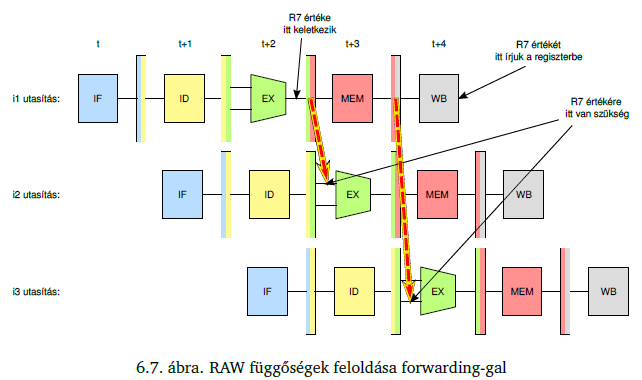
**b) verseny a regiszterekért:**megoldás:gyorsabb regiszter tároló *(az ID ill. a WB fázisok közül az egyik a ciklusidő első felében, a másik pedig a második felében használ)*

Adat-egymásrahatás  
megoldás:  
 forwarding: *(ekkor az ALU nem a memóriából kapja az adatot, hanem egy pipeline regiszterből, amiben már szerepel a helyes adat, de még nem került el a WB fázisig) + ha nem elég a forwarding, szünet beiktatása*

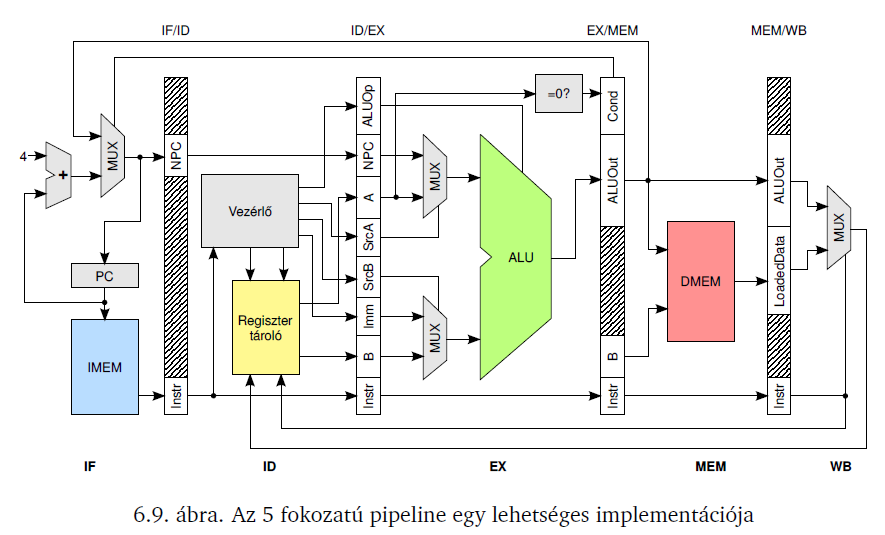
típusai:  
**WAW**: Write-After-Write : 2 utasítás ugyanabba a regiszterbe írja az eredményét

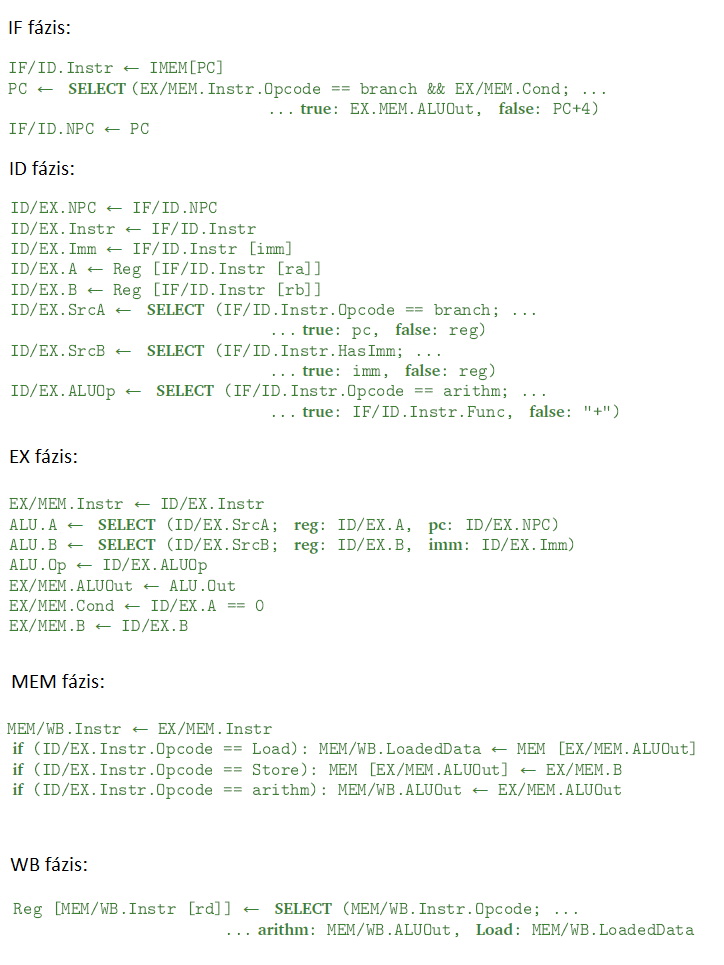
**WAR**: Write-After-Read : Az egyik utasítás olyan regiszterbe írja az eredményét, ami egy korábbi utasítás bemenő operandusa

**RAW**: Read-After-Write : Egy utasítás egyik vagy mindegyik operandusa az előtte lévő utasítás eredménye

Példa a RAW függőség megoldására:  
i1: R7 ← R1 + R5  
i2: R8 ← R7 - R2  
i3: R5 ← R8 + R7  
  
Az R7-re szükség van az i2 utasítás EX fázisánál. Ez még csak az i1 utasítás EX/MEM regiszterében van benne, így forwardingot alkalmazva nem kell szünetet tartani az i1 utasítás WB fázis végéig.  
Az R7re és az R8-ra szükség van az i3 utasítás EX fázisánál. Ezek még csak az i1 utasítás MEM/WB és az i2 utasítás EX/MEM regiszterében találhatóak meg, így forwardingot alkalmazva nem kell szünetet tartani az i1 és i2 utasítás végéig.   
  
Procedurális egymásrahatás  
feltételes ugrásoknál: az ugrásfeltétel kiértékelése az EX fázisban. IF honnan tudja a következő utasításokat, amíg az EX fázis végre nem hajtódott?  
megoldások:  
 1) sehonnan -> vár  
 2) elágazásbecsléssel:  
  
2) elágazásbecslés:  
A statikus elágazásbecslés és a dinamikus elágazásbecslés közötti különbség, hogy amíg a statikus a processzor előéletének figyelembevétele nélkül becsli meg az elágazás kimenetét, addig a dinamikus ilyen tényezőket is figyelembe vesz.  
  
a) statikus elágazásbecslés taktikák:  
 …mindig bekövetkezik az ugrás  
 …soha nem következik be az ugrás  
 …ha az ugrási cím visszafelé mutat:  
 bekövetkezik  
 egyébként ha az ugrásicím előre mutat:  
 nem következik be

b) Dinamikus elágazásbecslés:  
az elágazásbecslő logika folyamatosan követi, monitorozza a program feltételes ugró utasításait (természetesen ez mindenféle adatok folyamatos könyvelésével jár) és a múltbéli viselkedés valamint a környező más feltételes ugró utasítások együttes viselkedése alapján becsüli meg az ugrás kimenetelét és címét

n3) Adja meg a tanult 5 fokozatú pipeline egy lehetséges implementációját! Rajzolja fel a blokksémát, adja meg a pipeline regiszterek mezőit, és írja le a viselkedést pszeudokóddal! Az egymásrahatások kezelésétől eltekinthet!  




k9) Mit jelent a pontos kivételkezelés?  
k10) Milyen kivételek fordulhatnak elő a tanult 5 fokozatú pipeline különböző fázisaiban?  
n4) Osztályozza a pipeline utasításfeldolgozás során előforduló kivételeket! Milyen kivételek fordulhatnak elő a tanult 5 fokozatú pipeline különböző fázisaiban? Mi a pontos kivételkezelés, és miért nehéz ezt megvalósítani?

**Osztályzás:**

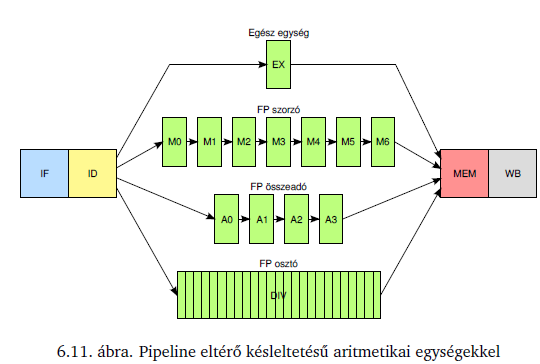
Szinkron vagy aszinkron  
Maszkolható vagy nem maszkolható  
Utasítások között vagy közben kezelendő események   
Folytatható vagy termináló

Jellegzetes kivételek:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Kivétel** | **Szinkron/Aszinkron** | **Maszkolható** | **Kezelendő** | **Folytatható** |
| periféria kérés | aszinkron | igen | utasítások között | igen |
| integer túlcsordulás | szinkron | igen | utasítások közben | igen |
| laphiba | szinkron | nem | utasítások közben | igen |
| védelmi hiba | szinkron | nem | utasítások közben | igen |
| érvénytelen utasítás | szinkron | nem | utasítások közben | igen/nem |
| hardver meghibásodás | aszinkron | nem | utasítások közben | nem |

**Fázisok kivételei:**  
IF: laphiba, védelmi hiba  
ID: érvénytelen utasítás  
EX: aritmetikai hiba (pl. integer túlcsordulás)  
MEM: laphiba, védelmi hiba  
WB: itt nem történhet kivétel  
  
**Pontos kivételkezelés:**A kivételt kezelő szoftver vagy hardver olyan állapotot lásson, hogy   
a kivétel(=i.) előtti utasítások(<i.)  
 teljesen lefutottak,  
az azután következők (≥i.)  
 pedig egyáltalán nem.

Ezt a viselkedést pontos kivételkezelésnek nevezik.  
Ezt nehéz megvalósítani, mivel több utasítás végrehajtása zajlik egyidejűleg.

n5) Adjon példát olyan utasítás pipeline-ra, melyben különböző késleltetésű műveleti egységek vannak! Rajzolja fel a blokksémát! Milyen problémákat hoz be a műveleti egységek különböző késleltetése, és hogyan lehet ezeket megoldani?  
A lebegőpontos műveletek (floating point, FP) több időt vesznek igénybe, mint az egész műveletek -> az EX fázist is pipeline-nal valósítjuk meg.  
  
**Problémák és kezelésük:**  
Feldolgozási egymásrahatás  
addig nem engedjük a következő fázisba az utasítást, amíg azt a fázist egy másik utasítás használja.  
  
RAW egymásrahatás  
forwardinggal javíthatunk a gyorsaságon egy keveset  
  
A WAW függőség  
szünet beiktatásával kiküszöbölhető