HÁLÓZATI RENDSZEREK ÉS SZOLGÁLTATÁSOK TANSZÉK

MŰEGYETEM 1782

Budapest,
2025.04.01.

SZÁMÍTÓGÉP ARCHITEKTÚRÁK

Pipeline utasításfeldolgozás

Horváth Gábor, Belső Zoltán

BME Hálózati Rendszerek és Szolgáltatások Tanszék
ghorvath@hit.bme.hu, belso@hit.bme.hu

UTASÍTÁSOK FELDOLGOZÁSA

- Most megtanuljuk, hogy lehet hatékonyan.
- Példa processzor: RISC.
- Utasításai:
  - Load/Store
    `R1 ← MEM[R0+42]` vagy `MEM[R0+42] ← R1`
  - Aritmetikai logikai
    `R1 ← R2 * R3` vagy `R1 ← 42 * R3`.
  - Vezérlésátadás:
    - Feltétel nélküli
    - Feltételes
      `JUMP -24 IF R2 == 0`
  - Load/Store kivételével más nem nyúlhat memóriához

UTASÍTÁSOK FELDOLGOZÁSA

- Utasítás végrehajtásának lépései:
  1) Lehívás (Instruction Fetch, **IF**)
     + utasításszámláló növelése (ha nem ugrás)
  2) Dekódolás, regiszterek kiolvasása
     (Instruction decode/register fetch, **ID**)
     - Binárisan kódolt utasítás felkoncolása:
       - Típus leválasztása
         → ALU vezérlő jelek
       - Regiszter operandusok kiolvasása a regiszter tárolóból
  3) Végrehajtás (Execution, **EX**) – ALU dolgozik
     - Load/Store: címet számol (`R1 ← MEM[R0+42]`)
     - Aritmetikai/logikai: eredményt számol (`R1 ← R2*R3`)
     - Feltételes ugrás: feltételt és ugrási címet számol
       (`JUMP [PC]-24 IF R2==0`)

UTASÍTÁSOK FELDOLGOZÁSA

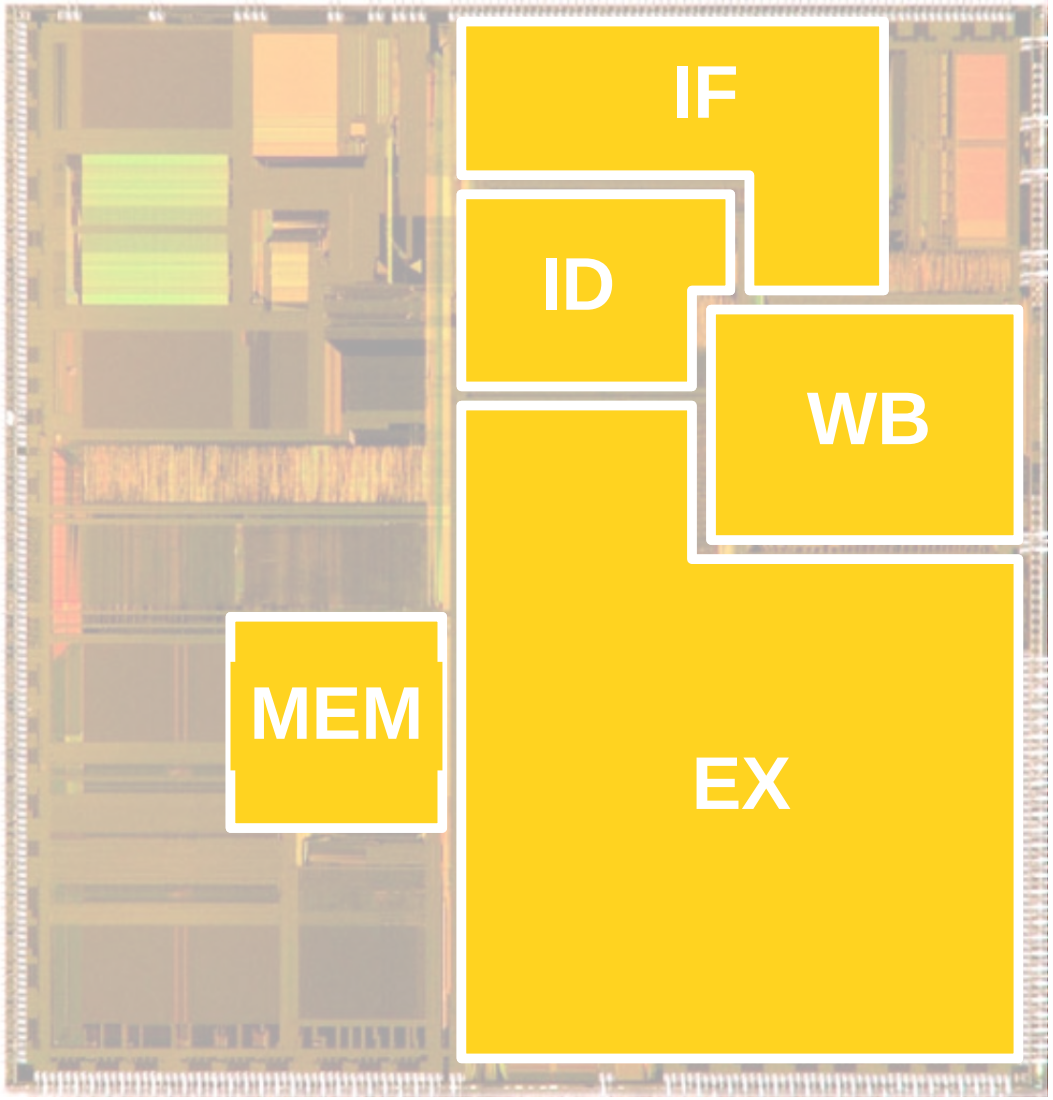
- … folytatás:

4) Memóriaműveletek (Memory access, **MEM**)
   - Load/Store: végrehajtódik
   - Aritmetikai/logikai, ugrások: kihagyható

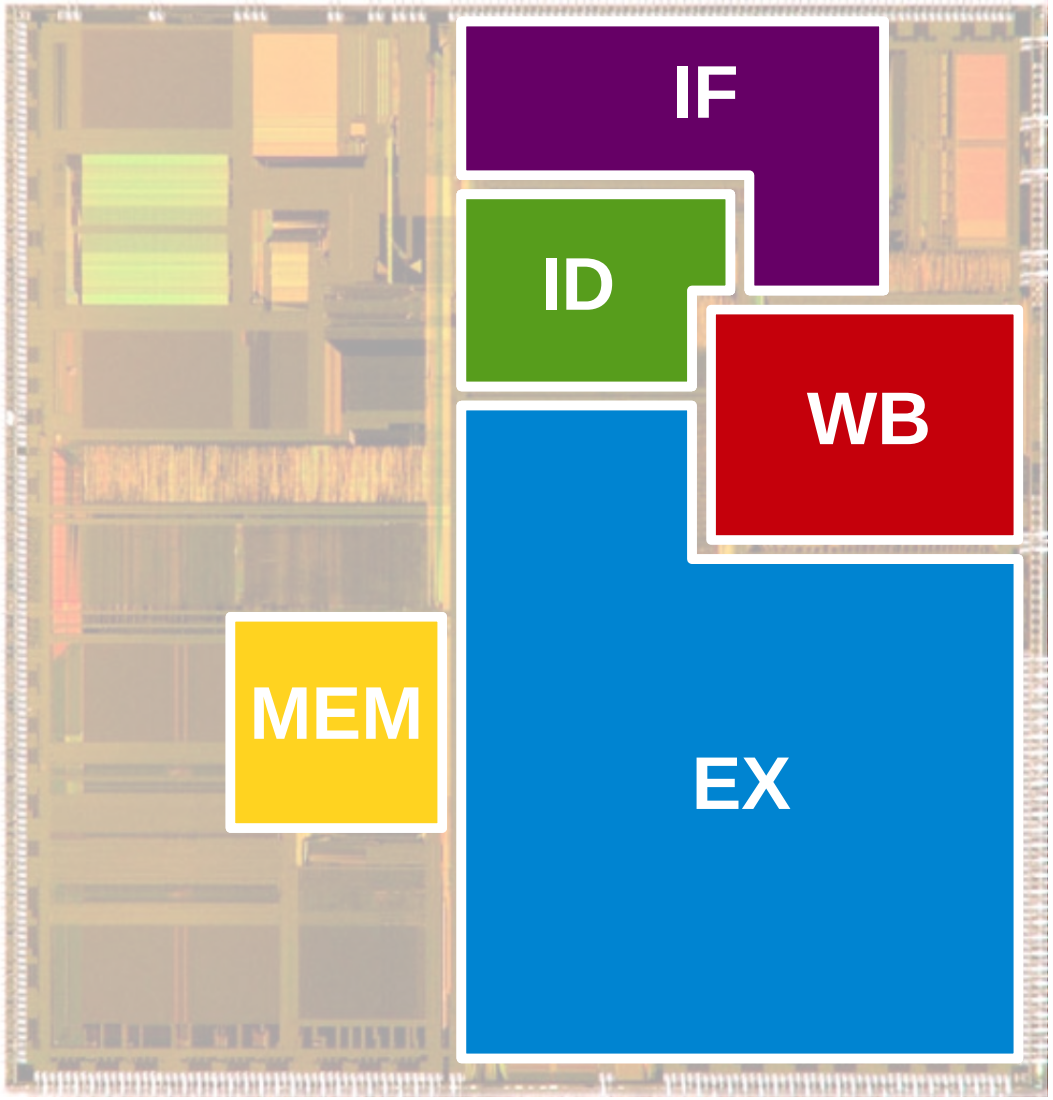
5) Regiszterek frissítése (Write-back, **WB**)
   - Aritmetikai/logikai: eredményt tárol (`R1 ← R2*R3`)
   - Load: eredményt tárol (`R1 ← MEM[R0+42]`)
   - Store, ugrások: kihagyható

- Aritmetikai: IF, ID, EX, WB
- Store: IF, ID, EX, MEM
- Load: IF, ID, EX, MEM, WB
- Ugrások: IF, ID, EX

UTASÍTÁSOK FELDOLGOZÁSA

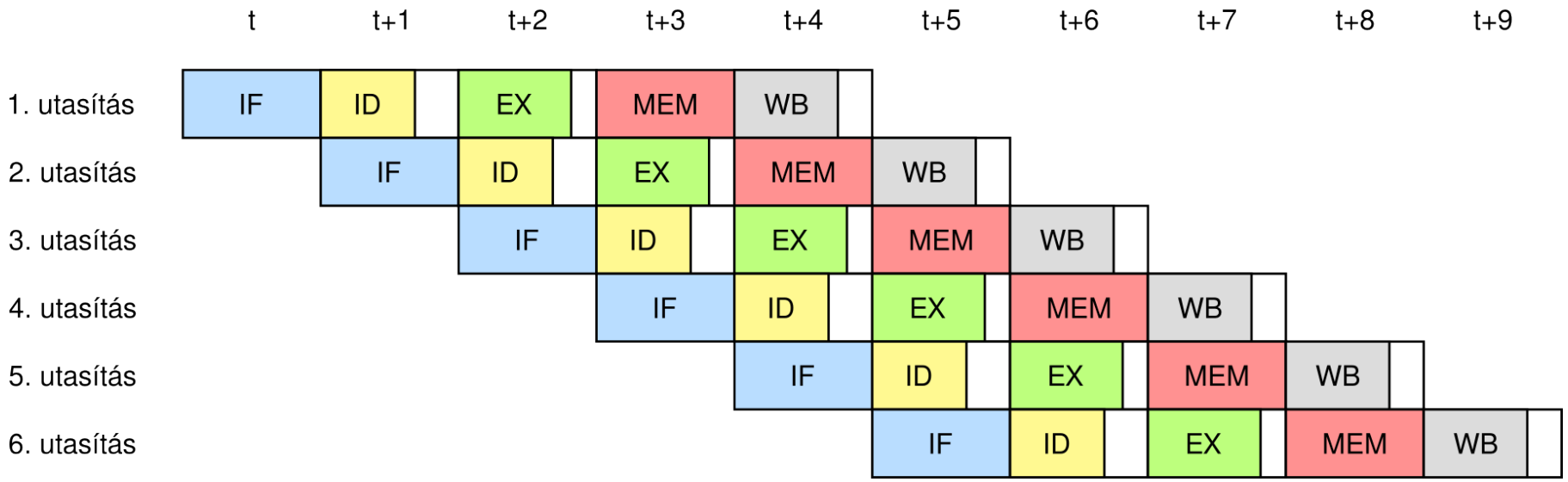


UTASÍTÁSOK FELDOLGOZÁSA



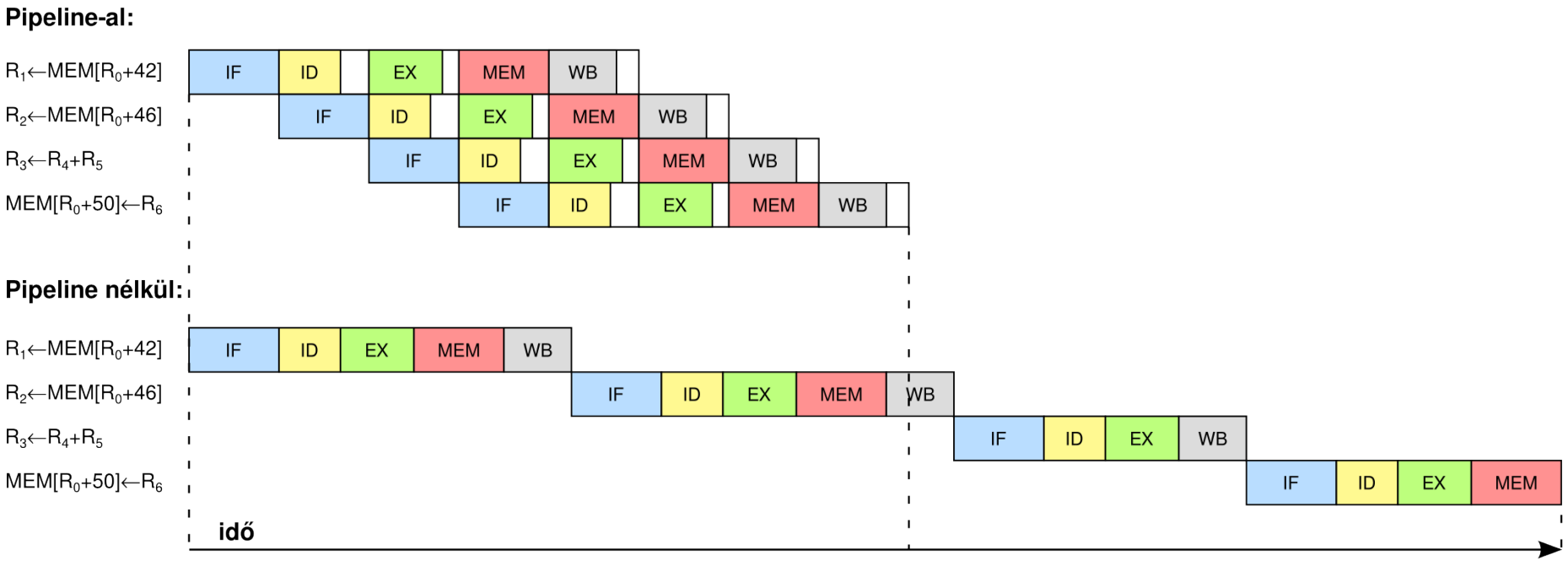
PIPELINE ELV

- Ez a pipeline elv
  - Lásd: chicagói vágóhidak, Ford T-model, stb.
- Átlapolt utasításfeldolgozás:
  - Minden fázisnak ugyanannyi időt kell adni: ***ciklusidő***



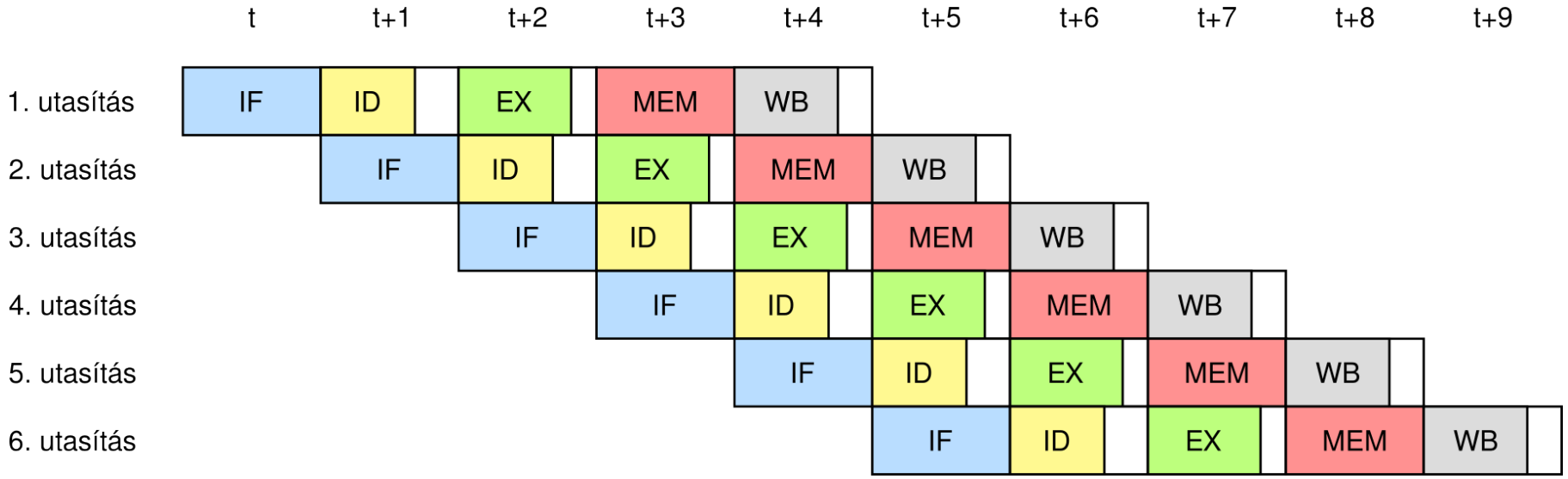
PIPELINE ELV

- Mennyit nyerünk?
  - Egyfelől: átlapoltunk, ez nyereség
  - Másfelől: üresjáratok! (szükségtelen fázisok, túl hosszú fázisok, ...)
- Ha függetlenek az utasítások, akkor sokat!



PIPELINE ELV

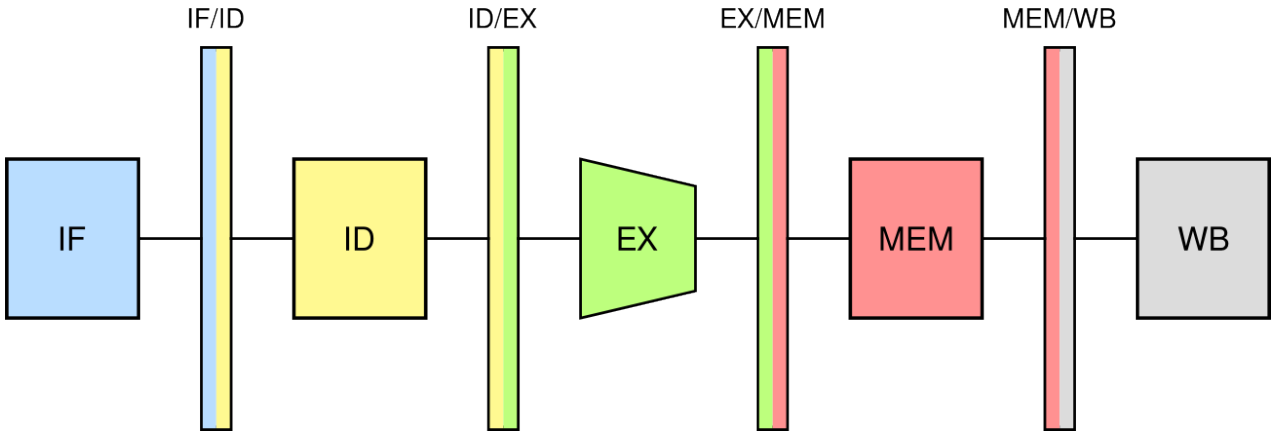
- Mennyiségek:
  - Mélység (deepness): fázisok száma
  - Késleltetés (latency): átfutási idő
  - Átviteli sebesség (throughput): kész utasítás / sec
  - Feltöltési / kiürülési idő



PIPELINE REGISZTEREK

- A fázisok egymásnak adogatják a „munkadarabbal” kapcsolatos dolgokat

  → pipeline regiszterekben tárolódnak



- IF az IF/ID-be teszi a lehívott utasítást,
- ID kiveszi, ALU vezérlőjeleket és operandusokat az ID/EX-be teszi,
- EX kiveszi, eredményt az EX/MEM-be teszi,
- Stb... (később)

EGYMÁSRAHATÁSOK

- Minden szép és jó, de:
- Az utasítások többnyire nem függetlenek!
  - Erőforrásokért versengenek

    → ***Feldolgozási egymásrahatás***
  - Egyik operandusa a másik eredménye

    → ***Adat egymásrahatás***
  - Ugrás: nem tudjuk a folytatást, amíg ki nem értékeljük a feltételt

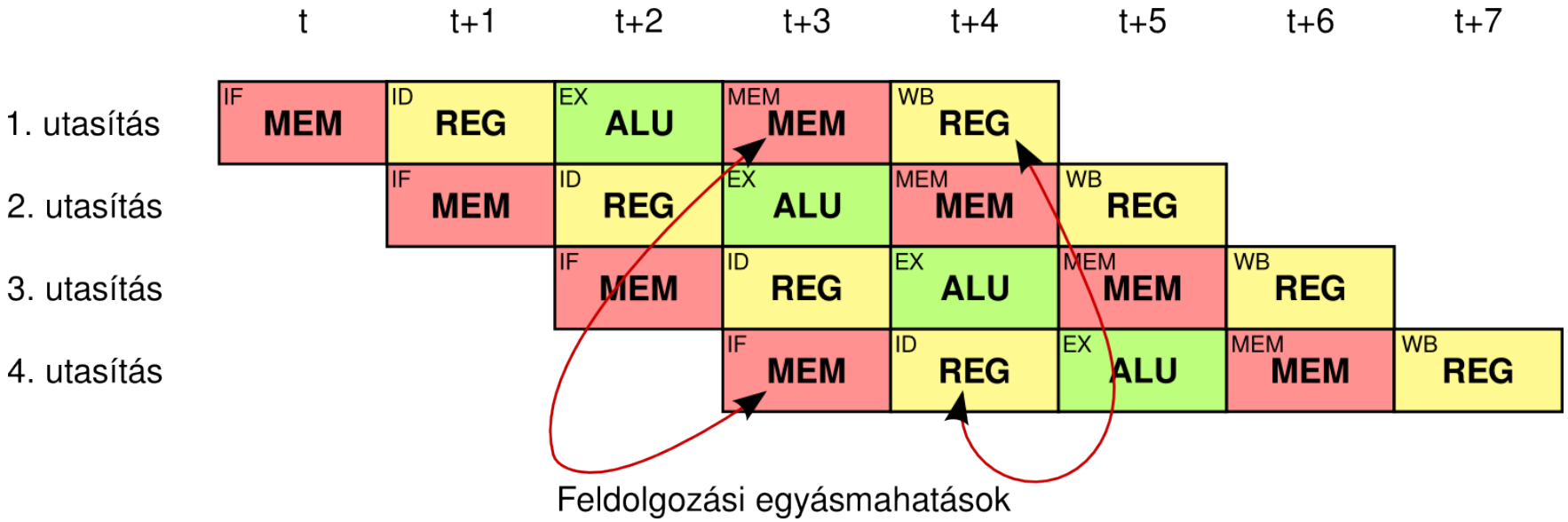
    → ***Procedurális egymásrahatás***

  Kezelés:
  - Trükkel feloldjuk
  - Ha nem megy, megállítjuk a pipeline-t, amíg meg nem oldódik

    → rontja a hatékonyságot

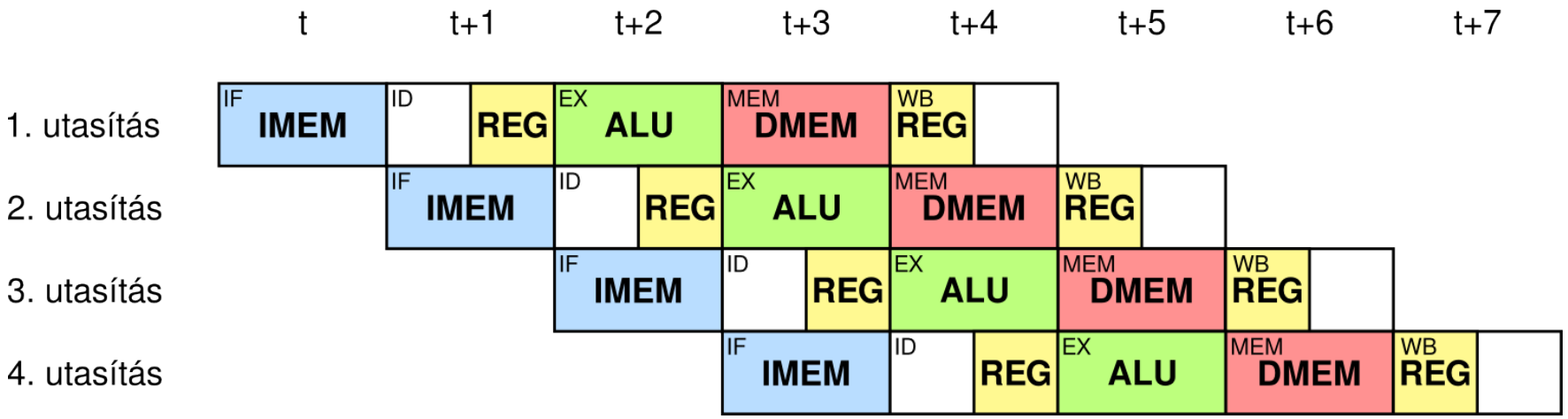
FELDOLGOZÁSI EGYMÁSRAHATÁS

- Egyes erőforrásokra több fázisban is szükség van
  - IF: Memória
  - ID: Regiszter tároló
  - EX: ALU
  - MEM: Memória
  - WB: Regiszter tároló



FELDOLGOZÁSI EGYMÁSRAHATÁS

- Feloldása:
  - Memória esetén:
    - külön utasítás és adat cache
  - Regiszter tároló esetén:
    - egyik az ciklus elején, másik a végén használhatja



ADAT-EGYMÁSRAHATÁS

- Forrása: adatfüggőségek
- Adatfüggőség:

  Több utasítás ugyanazon a címen végez műveletet
  (ugyanaz a regiszter, ugyanaz a memóriacím)

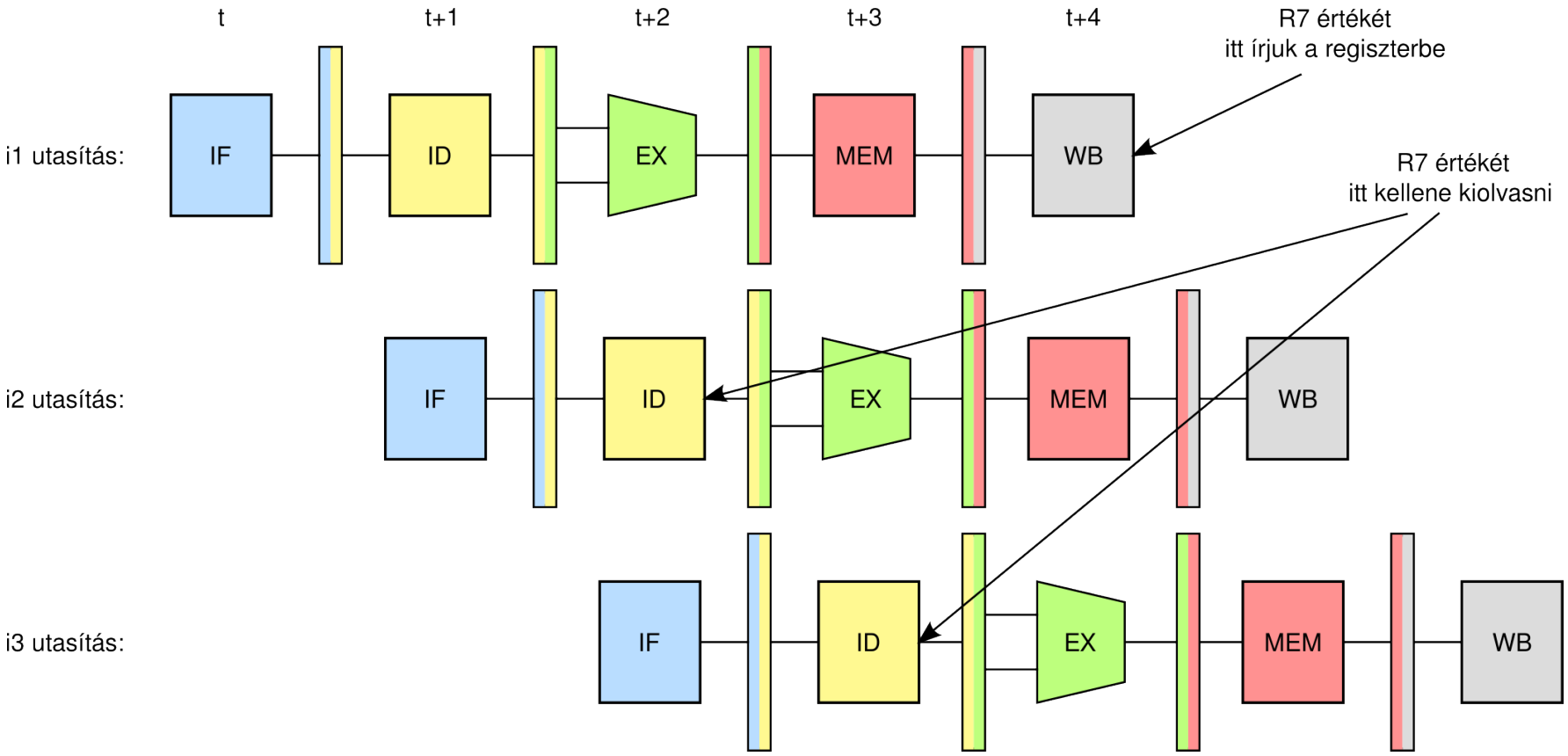
- Pl. az egyik utasítás operandusa az előző eredménye

  → ***RAW függőség*** (Read After Write)

```
i1:  R3 ← MEM[R2]
i2:  R1 ← R2 * R3
i3:  R4 ← R1 + R5
i4:  R5 ← R6 + R7
i5:  R1 ← R8 + R9
```

RAW függőségek: i1 ↔ i2, i2 ↔ i3

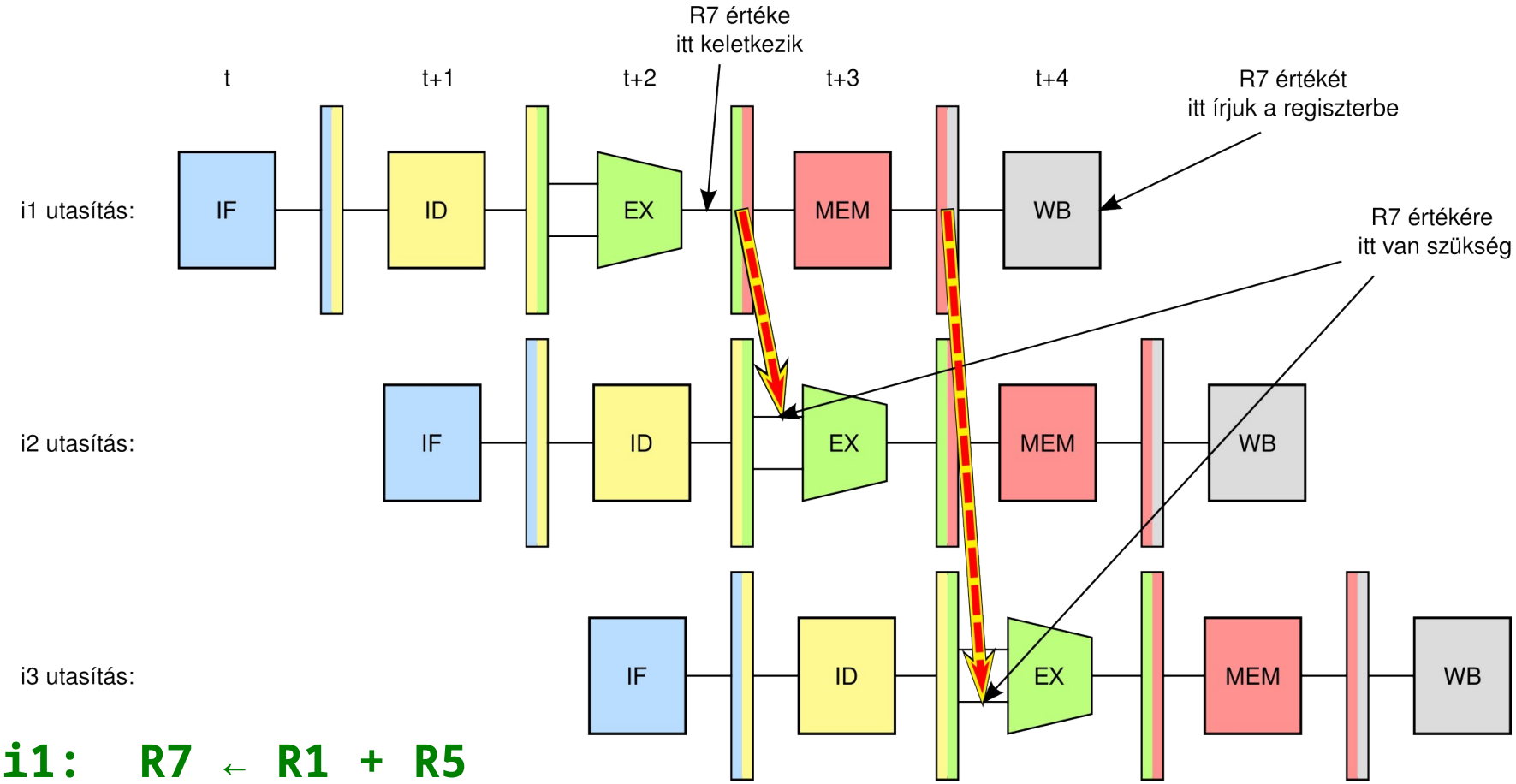
RAW EGYMÁSRAHATÁS




```
i1:  R7 ← R1 + R5
i2:  R8 ← R7 + R2
i3:  R5 ← R8 + R7
```

RAW EGYMÁSRAHATÁS

- Kész van az az eredmény, csak rossz helyen → ***forwarding***



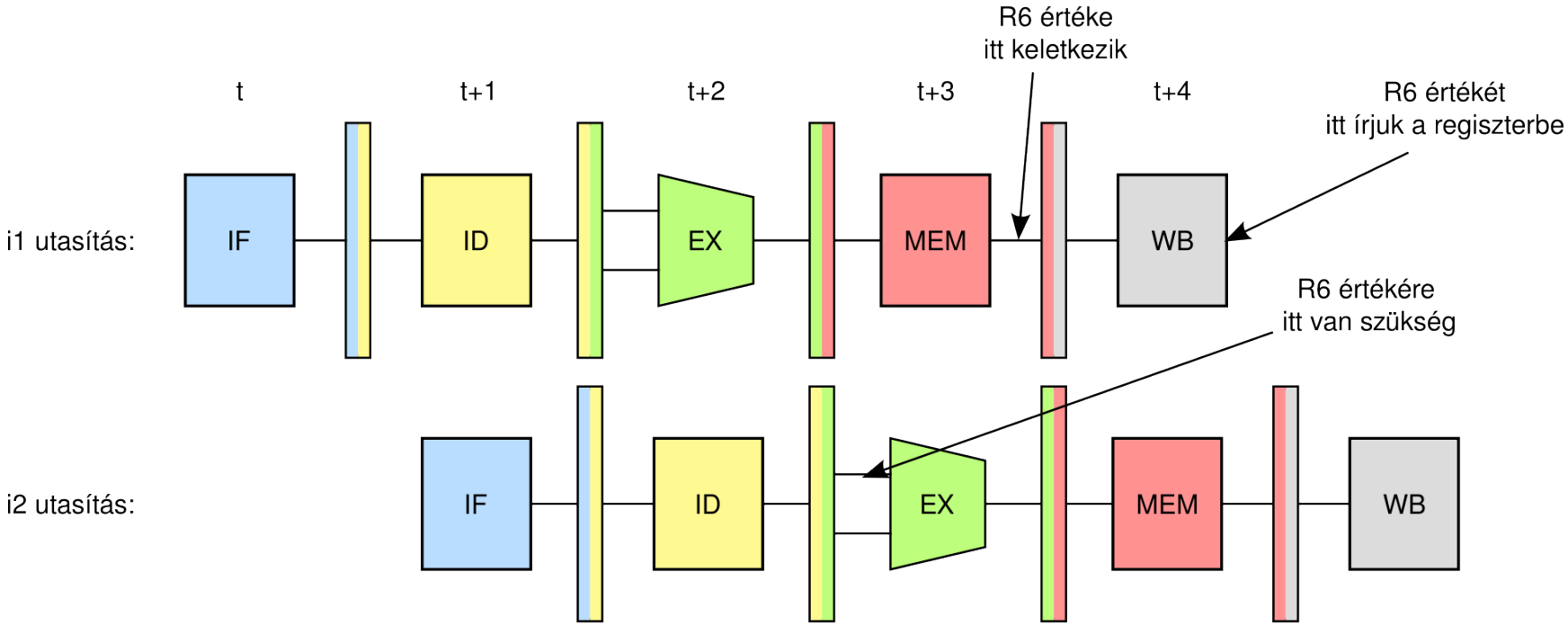

```
i1:   R7 ← R1 + R5
i2:   R8 ← R7 + R2
i3:   R5 ← R8 + R7
```

RAW EGYMÁSRAHATÁS

- Forwarding: van, ami ellen nem véd:

```
i1:  R6 ← MEM[R2]
i2:  R7 ← R6 + R4
```

- R6: értéke a MEM végén áll elő
- i2-nek az EX elején kellene

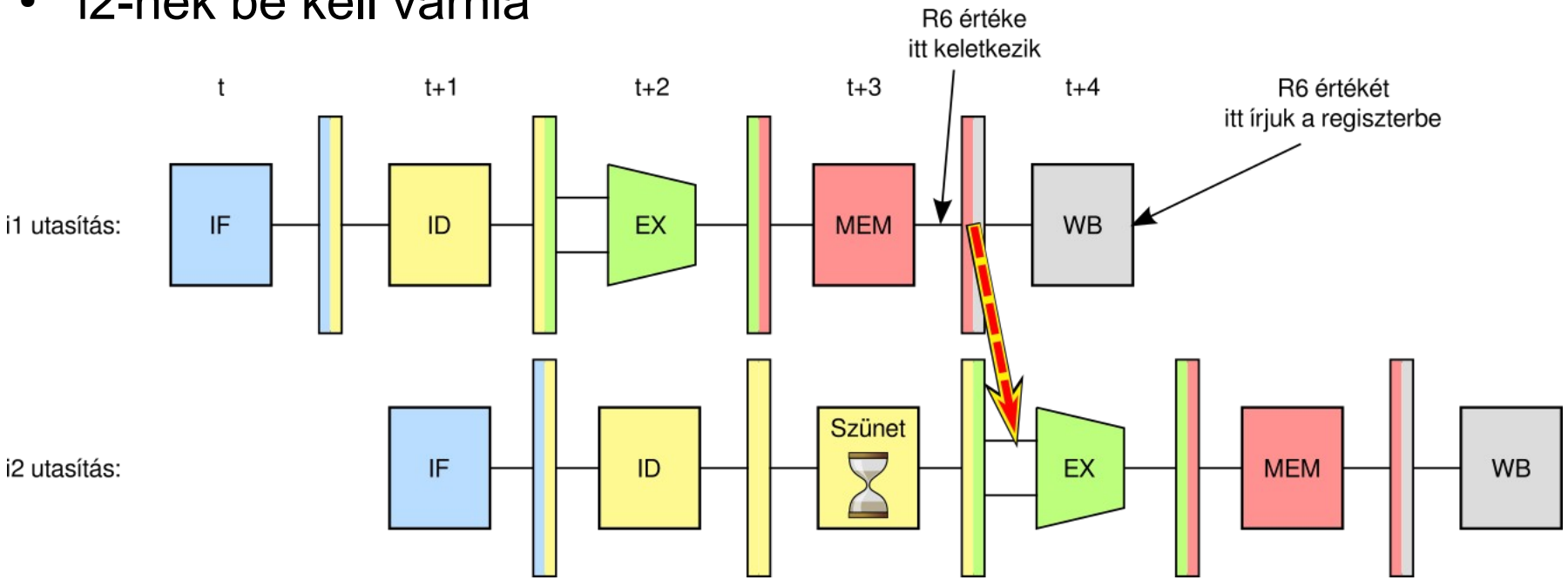


RAW EGYMÁSRAHATÁS

- Forwarding: van, ami ellen nem véd:

```
i1:  R6 ← MEM[R2]
i2:  R7 ← R6 + R4
```

- R6: értéke a MEM végén áll elő
- i2-nek az EX elején kellene
- i2-nek be kell várnia



ADAT-EGYMÁSRAHATÁS

- Adatfüggőségek:
  - **RAW**: későbbi utasítás olvassa a korábban írt regisztert/címet
    - Megoldás: forwarding/várakozás
  - WAR: későbbi utasítás írja a korábban olvasott regisztert/címet
    - Ebben a pipeline-ban nem igényel kezelést
  - WAW: több utasítás ugyanazt a regisztert/címet írja
    - Ebben a pipeline-ban nem igényel kezelést
  - RAR: több utasítás ugyanazt a regisztert/címet olvassa
    - Sosem igényel kezelést

- Példa:

```
i1:  R3 ← MEM[R2]
i2:  R1 ← R2 * R3
i3:  R4 ← R1 + R5
i4:  R5 ← R6 + R7
i5:  R1 ← R8 + R9
```

- WAW: i2 ↔ i5
- WAR: i3 ↔ i4

PROCEDURÁLIS EGYMÁSRAHATÁS

- Feltételes ugró utasításokkal baj van
  - IF betölti őket
  - Ciklusfeltétel & ugrási cím csak az EX-ben derül ki
  - Addig honnan vegyük a további utasításokat?
- Megoldások:
  - Álljunk meg és várjuk meg, míg az EX-be kerül → Intel 80386
  - Tippeljük meg a kimenetelt és az ugrási címet

    → ***elágazásbecslés***

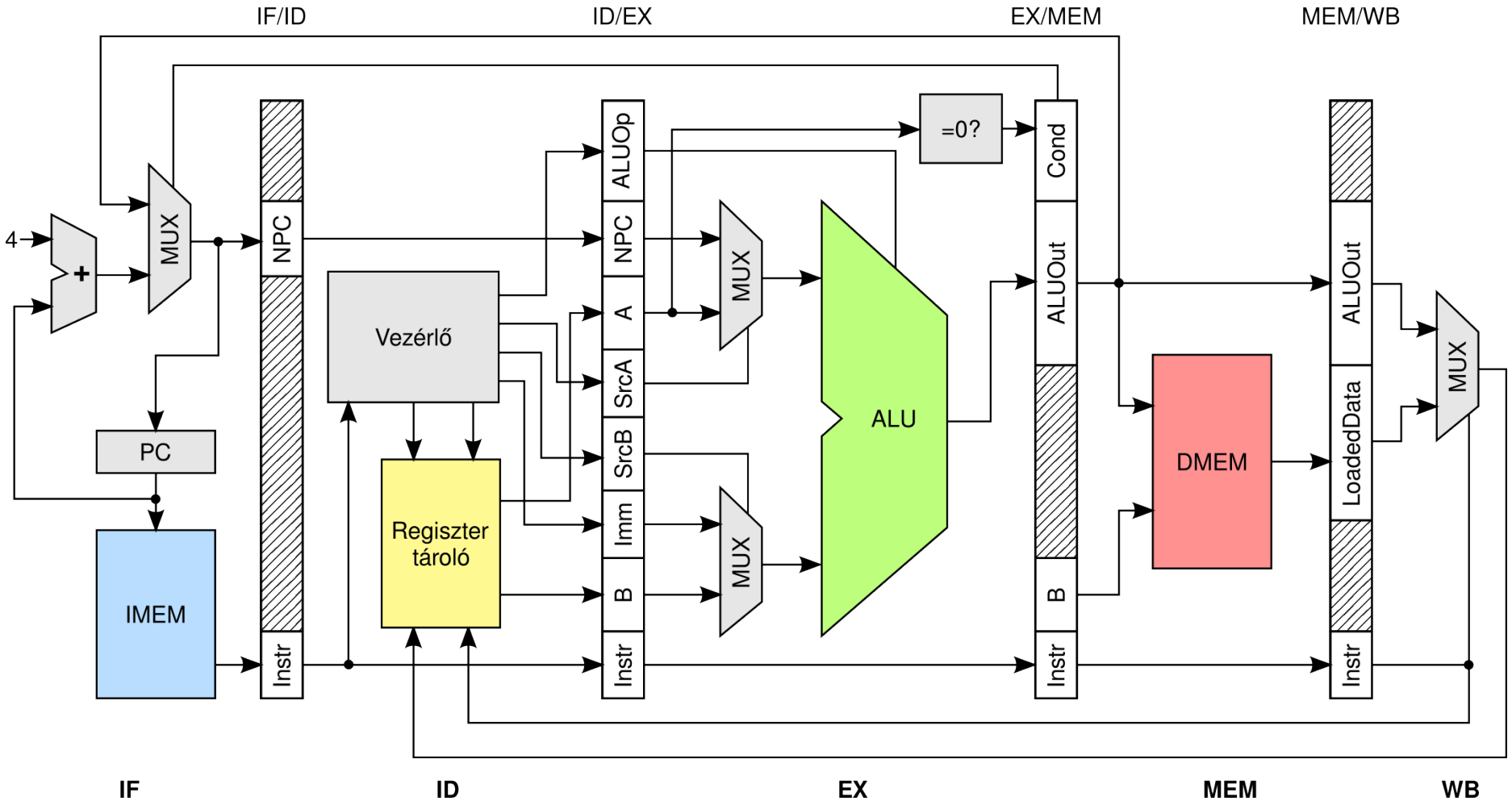
PROCEDURÁLIS EGYMÁSRAHATÁS

- **Statikus elágazásbecslés** → nem adaptív
  - Ugrás meghiúsulására voksol
    - Ingyen van.
      Ha rossz a tipp, érvénytelenítjük a tévesen behozottakat
  - Ugrás bekövetkezésére voksol
    - De akkor honnan veszi a címet?
    - Akkor jó, ha a cím hamarabb előáll, mint a feltétel
  - Visszafele ugrás: megvalósulást, előre: meghiúsulást tippel
    - Motiváció: ciklusok visszafele ugranak
- **Dinamikus elágazásbecslés**
  - Rátanul minden ugrás viselkedésére
  - Drasztikusan jobb, mint a statikus
  - Nem utópia! Sőt, nem is túl bonyolult!

RÉSZÖSSZEFOGLALÓ

- Mit tanultunk eddig?
  - Pipeline elvet
  - Fázisok pipeline regisztereken keresztül kommunikálnak
  - Hogy hogyan kell kezelni az egymásrahatásokat
- Csináljunk saját pipeline-os processzort!

PIPELINE IMPLEMENTÁCIÓ



AZ ALU

- ALU: központi szerep
- Számol:

```
R1 ← R1 + 42
R1 ← R2 + R3
PC ← PC + 42          (JUMP +42)
R1 ← MEM [R2 + 42]
```

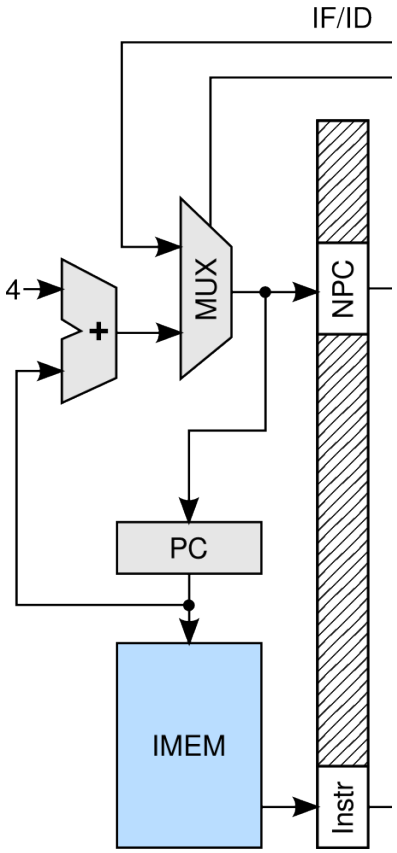
  - Első operandus: **regiszter** vagy **PC**
  - Második operandus: **regiszter** vagy **konstans** (immediate)
- Feltételt értékel ki:

```
JUMP -28 IF R1==0
```

- Az ID feladata:
  - Operandusok előkészítése
  - Operanduskiválasztó jel előállítása
  - Műveleti kód (+,-,*,/)

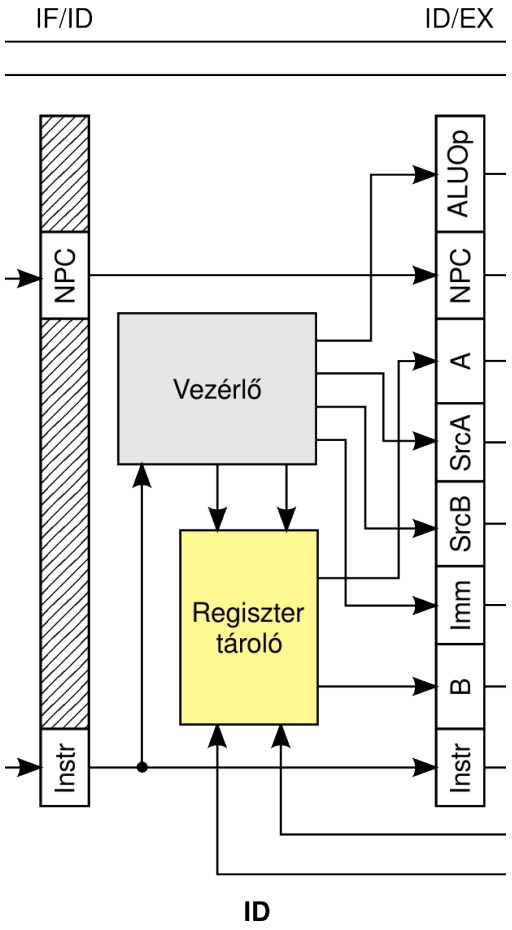
IF FÁZIS

- Utasításszámláló update:
  - Ha ugrás (**`EX/MEM.Instr.Opcode==branch`**), és ugrani kell (**`EX/MEM.Cond==TRUE`**):
    - **`PC ← EX.MEM.ALUOut`**
  - Egyébként:
    - **`PC ← PC+4`**
- Utasításszó és (új) programszámláló továbbadása:
  - **`IF/ID.NPC ← PC`**
  - **`IF/ID.Instr ← IMEM[PC]`**



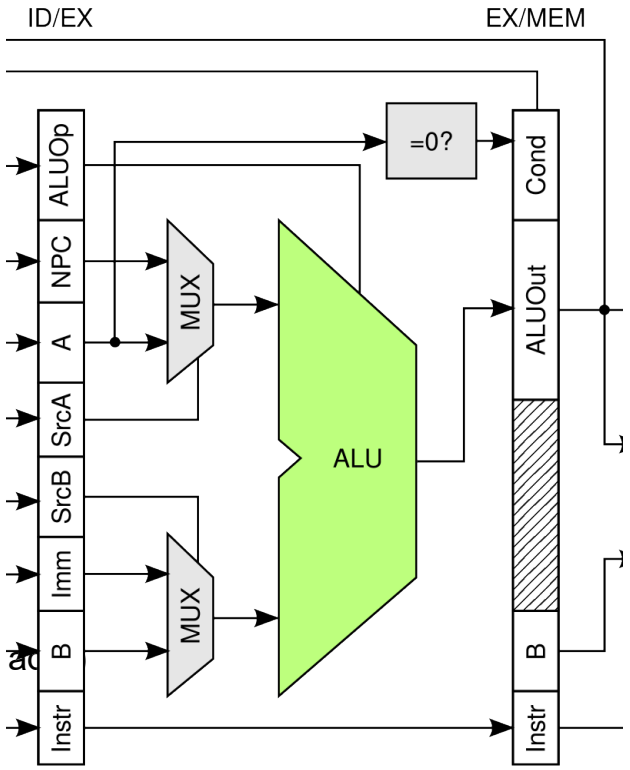
ID FÁZIS

- Operandusok előállítása:
  - Első operandus:
    - `ID/EX.NPC ← IF/ID.NPC`
    - `ID/EX.A ← Reg [IF/ID.Instr.ra]`
  - Első operandus kiválasztójel:
    - Ha ugró utasítás (`IF/ID.Instr.Opcode == branch`):
      - `ID/EX.SrcA ← npc`
    - Egyébként:
      - `ID/EX.SrcA ← regA`
  - Második operandus:
    - `ID/EX.Imm ← IF/ID.Instr.imm`
    - `ID/EX.B ← Reg [IF/ID.Instr.rb]`
  - Második operandus kiválasztójel:
    - Ha konstans (`IF/ID.Instr.HasImm`):
      - `ID/EX.SrcB ← imm`
    - Egyébként:
      - `ID/EX.SrcB ← regB`
- Műveleti kód előállítása:
  - Ha aritmetikai az utasítás (`IF/ID.Instr.Opcode == arithm`):
    - `ID/EX.ALUOp ← IF/ID.Instr.Func`
  - Egyébként (utasításszámláló növelés, címszámítás):
    - `ID/EX.ALUOp ← „+”`
- Utasításszó továbbadása:
  - `ID/EX.Instr ← IF/ID.Instr`



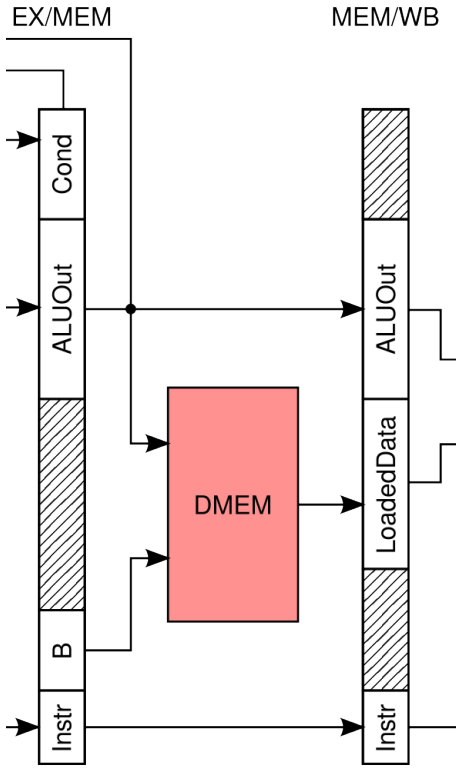
EX FÁZIS

- ALU bemenetei:
  - Ha `ID/EX.SrcA == npc`
    - `ALU.A ← ID/EX.NPC`
  - Ha `ID/EX.SrcA == regA`
    - `ALU.A ← ID/EX.A`
  - Ha `ID/EX.SrcB == imm`
    - `ALU.B ← ID/EX.Imm`
  - Ha `ID/EX.SrcB == regB`
    - `ALU.B ← ID/EX.B`
  - Művelettípus:
    - `ALU.Op ← ID/EX.ALUOp`
- Eredmény tárolása:
  - `EX/MEM.ALUOut ← ALU.Out`
- Összehasonlító egység:
  - `EX/MEM.Cond ← ID/EX.A == 0`
- Store utasítás esetén B továbbítása
  - `EX/MEM.B ← ID/EX.B` (Ez lesz majd a beírandó ad
- Utasításszó továbbadása:
  - `EX/MEM.Instr ← ID/EX.Instr`



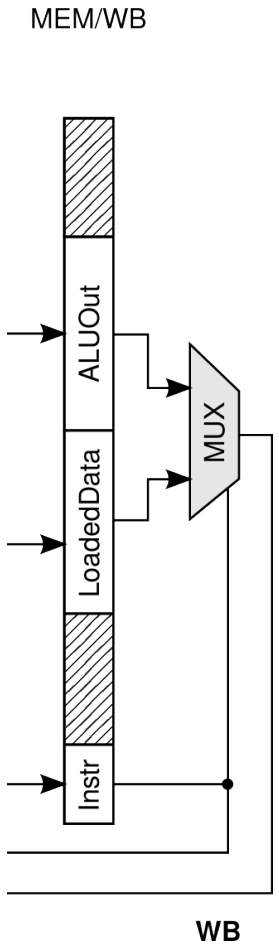
MEM FÁZIS

- Memóriacím:
  - **`ALUOut`**
- Store esetén (**`ID/EX.Instr.Opcode == Store`**)
  - **`MEM[EX/MEM.ALUOut] ← EX/MEM.B`**
    (Beírandó adat: B)
- Load esetén (**`ID/EX.Instr.Opcode == Load`**)
  - **`MEM/WB.LoadedData ← MEM[EX/MEM.ALUOut]`**
- Aritmetikai utasítás esetén:
  - **`MEM/WB.ALUOut ← EX/MEM.ALUOut`**
    (Eredmény továbbadása)
- Utasításszó továbbadása:
  - **`MEM/WB.Instr ← EX/MEM.Instr`**

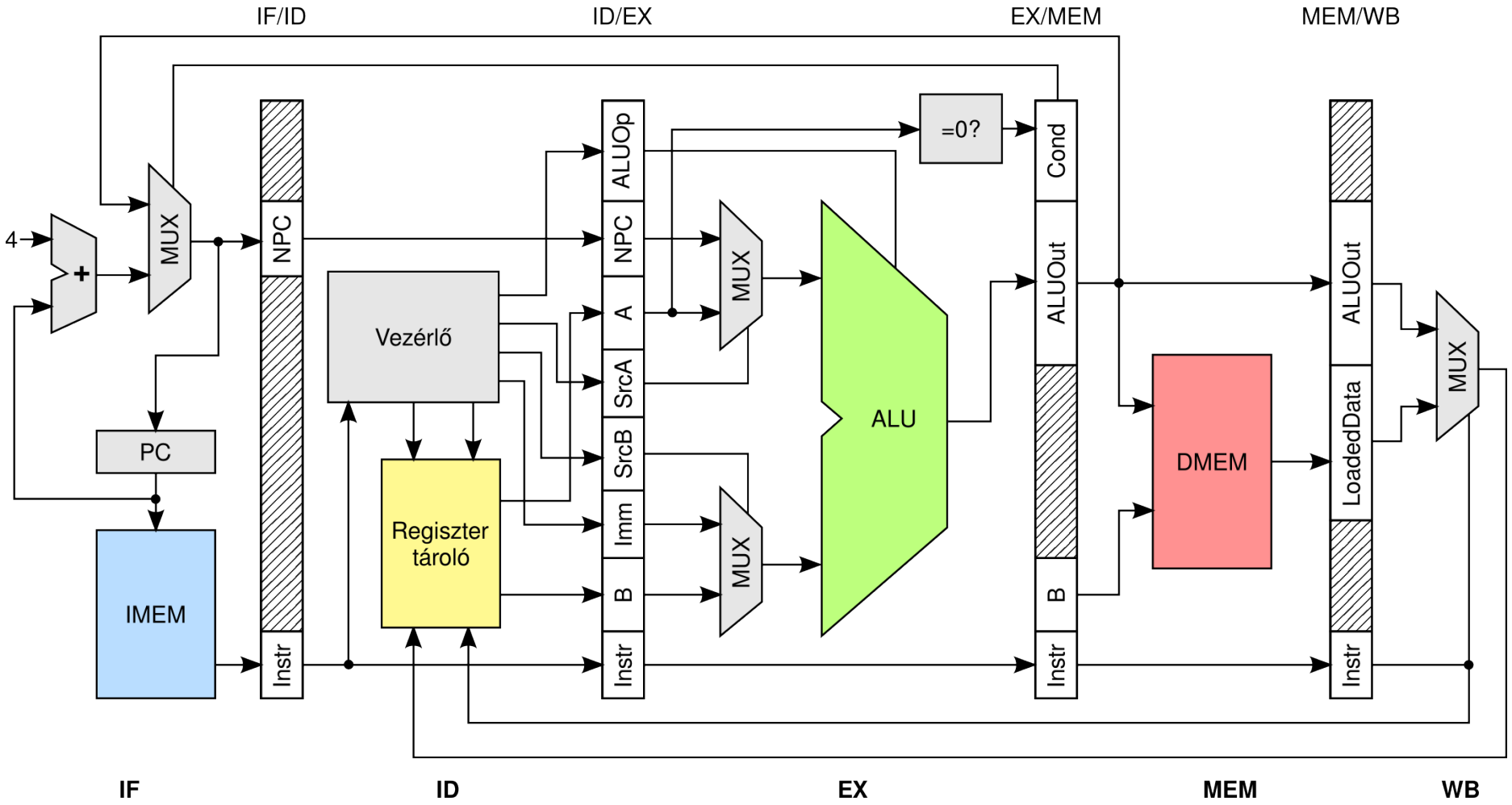


WB FÁZIS

- Regisztertároló (`Reg[]`) frissítése
  - Aritmetikai utasítás esetén (`MEM/WB.Instr.Opcode == arithm`)
    - `Reg[MEM/WB.Instr.rd] ← MEM/WB.ALUOut`
  - Load utasítás esetén (`MEM/WB.Instr.Opcode == Load`)
    - `Reg[MEM/WB.Instr.rd] ← MEM/WB.LoadedData`

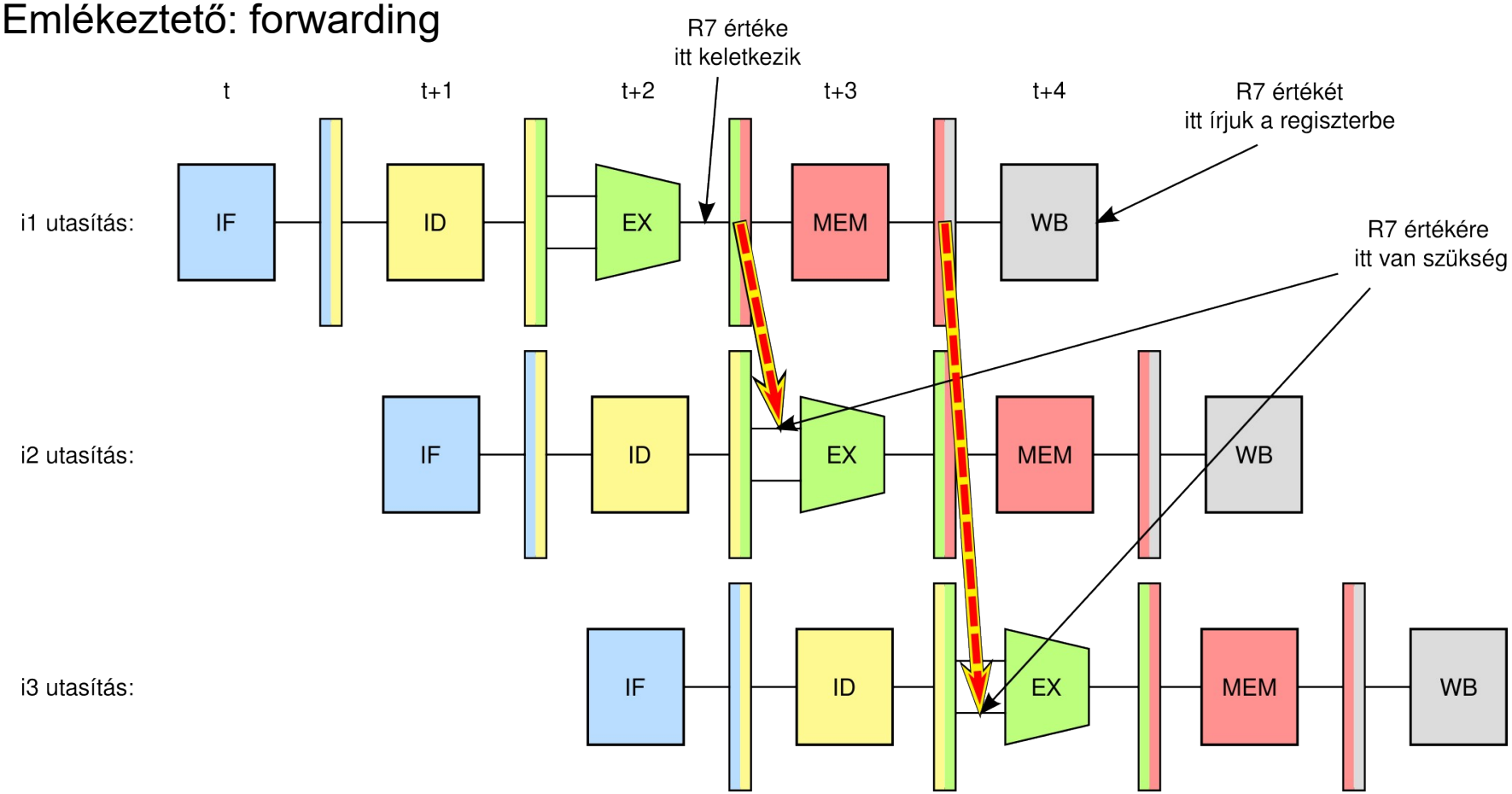


PIPELINE IMPLEMENTÁCIÓ



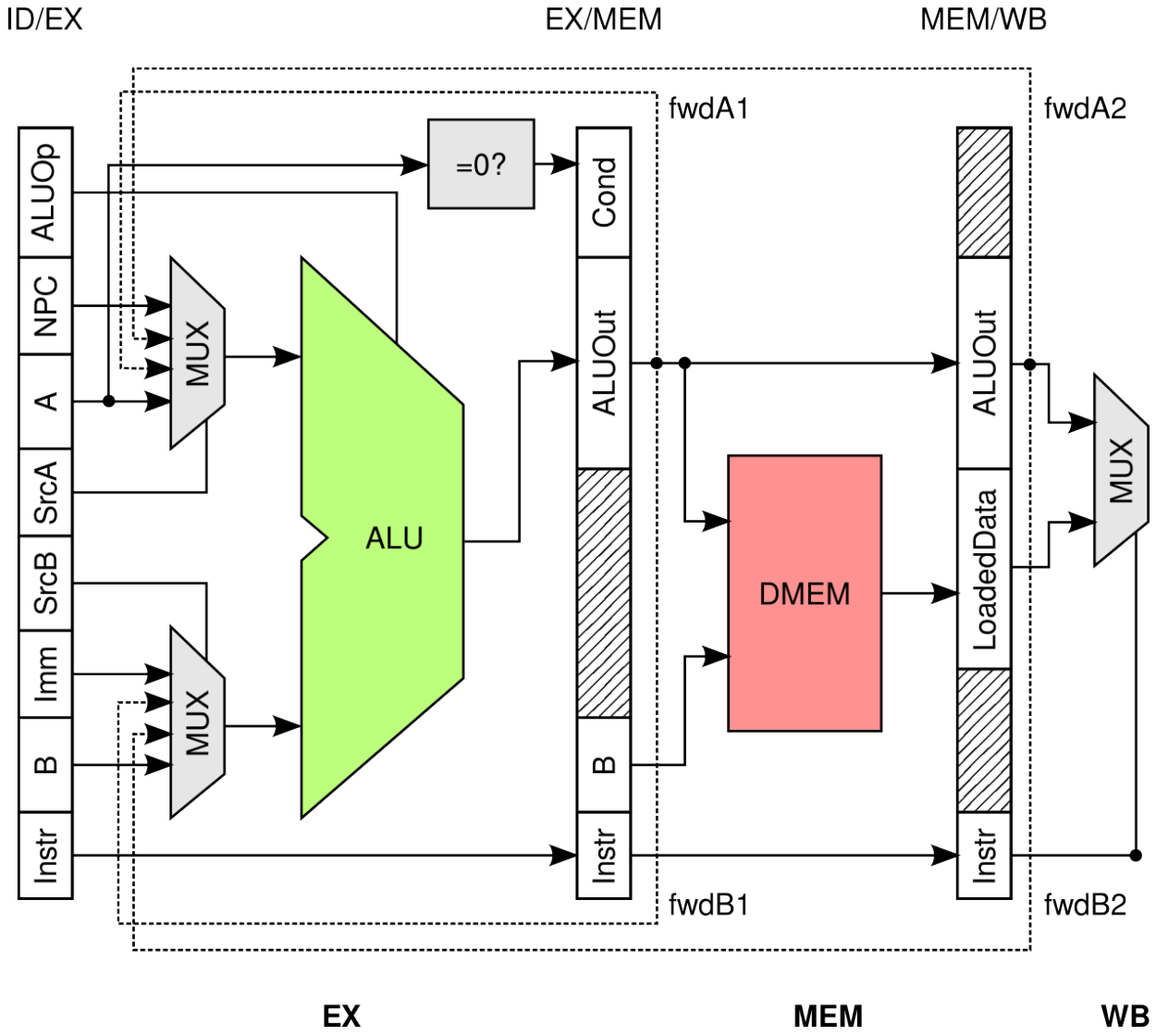
RAW EGYMÁSRAHATÁS

- Emlékeztető: forwarding




```
i1:  R7 ← R1 + R5
i2:  R8 ← R7 + R2
i3:  R5 ← R8 + R7
```

RAW EGYMÁSRAHATÁS



RAW EGYMÁSRAHATÁS

- Honnan tudja az ID, hogy mik lesznek az ALU operandusai?
  - Két operandus
  - Mindkettő kétféle lehet (NPC ↔ regiszter ill. immediate ↔ regiszter)
  - Regiszterek három helyről származhatnak:
    - ID-től
    - ALU kimenetről
    - Egyel korábbi ALU kimenetről
- Kiválasztó logika (ID-ben!):
  - pl. 1-es operandusra, **`ID/EX.SrcA`** lehet 0, 1, 2, 3:
    - **`npc: IF/ID.Instr.Opcode == branch`**
    - **`fwdA1:`** ha **`IF/ID.Instr.Opcode==arithm && IF/ID.Instr.ra == ID/EX.Instr.rd`**
    - **`fwdA2:`** ha **`IF/ID.Instr.Opcode==arithm && IF/ID.Instr.ra == EX/MEM.Instr.rd`**
    - **`regA:`** ha **`IF/ID.Instr.Opcode==arithm`**, egyébként
  - Hasonlóan a 2-es operandusra

EGYÉB EGYMÁSRAHATÁSOK

- Ha szünetet igényel a RAW egymásrahatás:
  - ID az előbbi módon detektálja
  - Szól az IF-nek, hogy most ne hívjon le új utasítást
- Procedurális egymásrahatás:
  - Megáll & vár taktika: 2 szünet beiktatása, mint előbb
  - „Csak nem következik be” taktika:
    - Folytatja a következő utasítások lehívását
    - Ha mégis ugrás történt, a két félig kész utasítást hatástalanítani kell:
      - Utasítások egy „Valid” flag-et cipelnek magukkal
      - Ha kiderül, hogy félbe kell hagyni őket, Valid=0
      - Valid=0-ás utasítást a MEM és a WB fázis semmibe vesz

RÉSZÖSSZEFOGLALÓ

- Mit tanultunk eddig?
  - Pipeline elvet
  - Fázisok pipeline regisztereken keresztül kommunikálnak
  - Hogy hogyan kell kezelni az egymásrahatásokat
  - Csináltunk saját pipeline-os processzort
  - A processzorunk már az egymásrahatásokat is kezeli
- Addig jó, amíg valami rendkívüli nem történik
- Mi történhet?
  - Kivétel!
    - Periféria kérés,
    - Laphiba,
    - Védelmi hiba,
    - Érvénytelen utasítás,
    - Stb.

PONTOS KIVÉTELKEZELÉS

- Kívánatos viselkedés:
  - Ha az i. utasítás közben keletkezik kivétel
  - Tűnjön úgy, mintha <i utasítások befejeződtek volna
  - És a >i utasítások egyáltalán el sem kezdődtek volna

  → ***pontos kivételkezelés***
- Pipeline feldolgozás esetén nem triviális
- A kivételkor jó lenne megakadályozni, hogy a későbbi utasításoknak hatása legyen
- … de lehet, hogy már meg is történt a baj

PONTOS KIVÉTELKEZELÉS

- Melyik fázisban mi történhet:
  - IF fázis: Laphiba, védelmi hiba
  - ID fázis: Érvénytelen utasítás
  - EX fázis: Aritmetikai hiba (pl. integer túlcsordulás)
  - MEM fázis: Laphiba, védelmi hiba
  - WB fázis: Itt nem történhet kivétel

	1	2	3	4	5	6
Rk ← Rm + Rn	IF	ID	EX	MEM	WB	
Ri ← MEM [Rj]		IF	ID	EX	MEM	WB

- 1. utasítás: EX fázisban túlcsordulás
- 2. utasítás: IF fázisban laphiba

  → Felborul a sorrend! Egy későbbi utasítás hibája jön előbb!!

PONTOS KIVÉTELKEZELÉS

- Egy lehetséges megoldás:
  - A kivételt nem kezeljük azonnal
  - Csak feljegyezzük, hogy volt (utasítás görgeti maga előtt)
  - Tényleges kezelés: WB fázisban, vagy utána

    → sorrend garantáltan helyes marad!

	1	2	3	4	5	6
Rk ← Rm + Rn	IF	ID	EX	MEM	WB	
Ri ← MEM [Rj]		IF	ID	EX	MEM	WB

ELTÉRŐ IDEJŰ ARITMETIKAI MŰVELETEK

- Áttekintettük, mi a helyzet a kivételekkel
- Újabb csavar: mi van, ha az EX fázis nem mindenkinek egyforma hosszú?

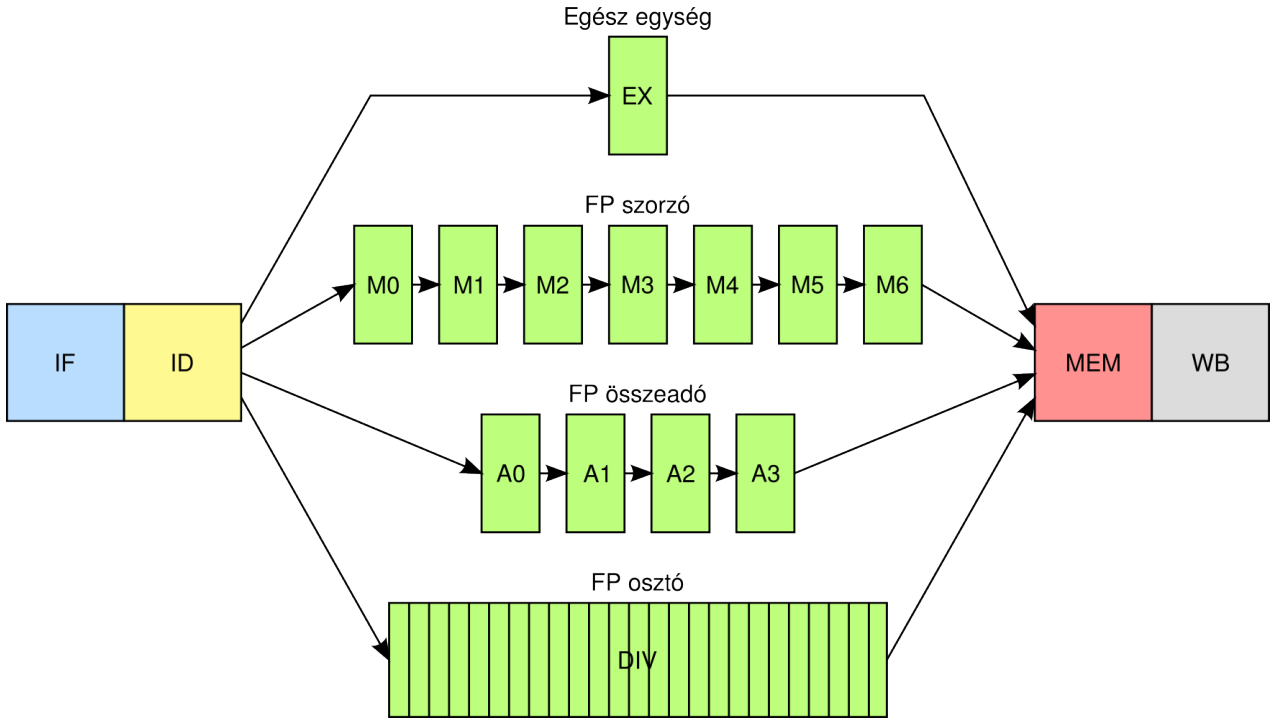
ELTÉRŐ IDEJŰ ARITMETIKAI MŰVELETEK

- Aritmetikai műveletek ***késleltetése***:
  - Lebegőpontos műveletek tovább tartanak
  - Szorzás tovább tart, mint az összeadás
  - Osztás még tovább tart
- A műveleti egységek lehetnek maguk is pipeline szervezésűek
  - Pl. az összeadás 3 ciklus késleltetésű,
  - ...de minden ciklusban új összeadást tud kezdeni

  → ***Iterációs idő*** = 1

ELTÉRŐ IDEJŰ ARITMETIKAI MŰVELETEK

Egység	Késleltetés	Iterációs idő
Integer ALU	1	1
FP összeadó	4	1
FP szorzó	7	1
FP osztó	25	25



ELTÉRŐ IDEJŰ ARITMETIKAI MŰVELETEK

- Érdekesség:
  - Végrehajtás kezdete: ***in-order***, vége: ***out-of-order*** is lehet!

D4 ← D1 * D5	IF	ID	M0	M1	M2	M3	M4	M5	M6	MEM	WB
D2 ← D1 + D3		IF	ID	A0	A1	A2	A3	MEM	WB		
D0 ← MEM [R0+4]			IF	ID	EX	MEM	WB				
MEM [R0+8] ← D5				IF	ID	EX	MEM	WB			

- Nem volt gond, utasítások függetlenek
- Nem lesz mindig így
- ID egység új feladatai:
  - a program szemantikájának megőrzése
  - újfajta egymásrahatások kezelése

ÚJFAJTA FELDOLGOZÁSI EGYMÁSRAHATÁSOK

Utasítás	1	2	3	4	5	6	7	8	9	10	11
D4 ← D1 * D5	IF	ID	M0	M1	M2	M3	M4	M5	M6	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
D2 ← D1 + D3				IF	ID	A0	A1	A2	A3	MEM	WB
...					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
D0 ← MEM [R0+4]							IF	ID	EX	MEM	WB

- Gond:
  - 10.: egyszerre hárman MEM-ben
  - 11.: egyszerre hárman WB-ben
- Megoldás:
  - ID tudja, mi mennyi ideig tart → előre látja ezt a helyzetet
  - Ha valaki feldolgozási egymásrahatást okozna → parkoltatja

TÖBB RAW EGYMÁSRAHATÁS

- Adatfüggőségek még nagyobb gondot okoznak
- Oka: hosszabb pipeline:
  - Későn áll elő az eredmény
    → ha valaki a korábbi eredményétől függ, többet kell várnia
  - Több utasítás van feldolgozás alatt
    → nagyobb a sansz a RAW függőségre

Utasítás	1	2	3	4	5	6	7	8	9	10	11	12	14	14	15	16	17
D4 ← MEM [R0+4]	IF	ID	EX	ME	WB												
D0 ← D4 * D6		IF	ID	**SZ**	M0	M1	M2	M3	M4	M5	M6	ME	WB				
D2 ← D0 + D8			IF	**SZ**	ID	**SZ**	**SZ**	**SZ**	**SZ**	**SZ**	**SZ**	A0	A1	A2	A3	ME	WB
MEM [R0+4] ← D2					IF	**SZ**	**SZ**	**SZ**	**SZ**	**SZ**	**SZ**	ID	EX	**SZ**	**SZ**	**SZ**	ME

WAW EGYMÁSRAHATÁSOK

- WAW függőség:
  - Két utasítás ugyanabba a regiszterbe menti az eredményét
- Ha megengedjük, hogy az utasítások sorrenden kívül fejeződjenek be → baj lehet!
  - Mert lehet, hogy a végén a korábbi eredménye kerül bele!
  - Az ID fázisnak detektálnia, és szünetekkel kezelnie kell

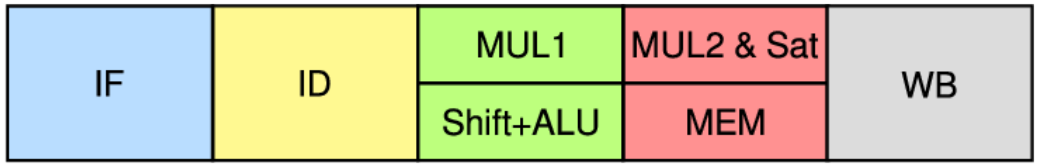
Utasítás	1	2	3	4	5	6	7	8
D2 ← D1 + D3	IF	ID	A0	A1	A2	A3	MEM	WB
...		IF	ID	EX	MEM	WB		
D2 ← MEM [R0+4]			IF	ID	EX	MEM	WB	

- Láthatóan rossz.
  - Megoldás: ID 2 szünetet iktat be a 3. utasításnál

Alternatív pipeline struktúrák

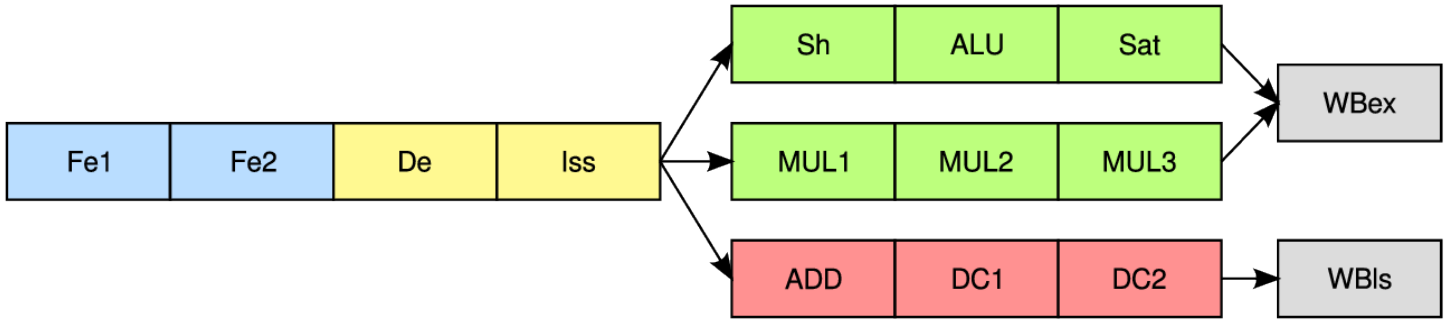
AZ ARM926EJ-S PROCESSZOR PIPELINE-JA

- 5 fokozatú, hasonlít a tanultakhoz
- Eltérés:
  - Szorzás két fázist igényel → második lépése: MEM
  - Van telítődő aritmetika → telítődés a MEM-ben



AZ ARM1176JZF-S PROCESSZOR PIPELINE-JA

- A Raspberry Pi processzora
- Mélység: 8
  - IF: 2 fázis (Fe1: lehívás, Fe2: elágazásbecslés)
  - ID: 2 fázis (De: dekódolás, Iss: regisztertároló kiolvasása)
- 3 műveleti egység, késleltetés: 3, iterációs idő: 1
  - ALU (Sh: shift, ALU: számolás, Sat: telítődés)
  - Szorzás (MUL1, MUL2, MUL3)
  - Memóriaműveletek (ADD: címszámolás, DC1, DC2: adatcache elérés)
  - Az ALU és a szorzó nem megy egyszerre, a többi igen



HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK