Számítógép architektúrák

- **Számítógépek felépítése**
- **Digitális adatábrázolás**
- **Digitális logikai szint**
- **Mikroarchitektúra szint**
- **Gépi utasítás szint**
- **Operációs rendszer szint**
- **Assembly nyelvi szint**
- **Probléma orientált (magas szintű) nyelvi szint**
- **Perifériák**

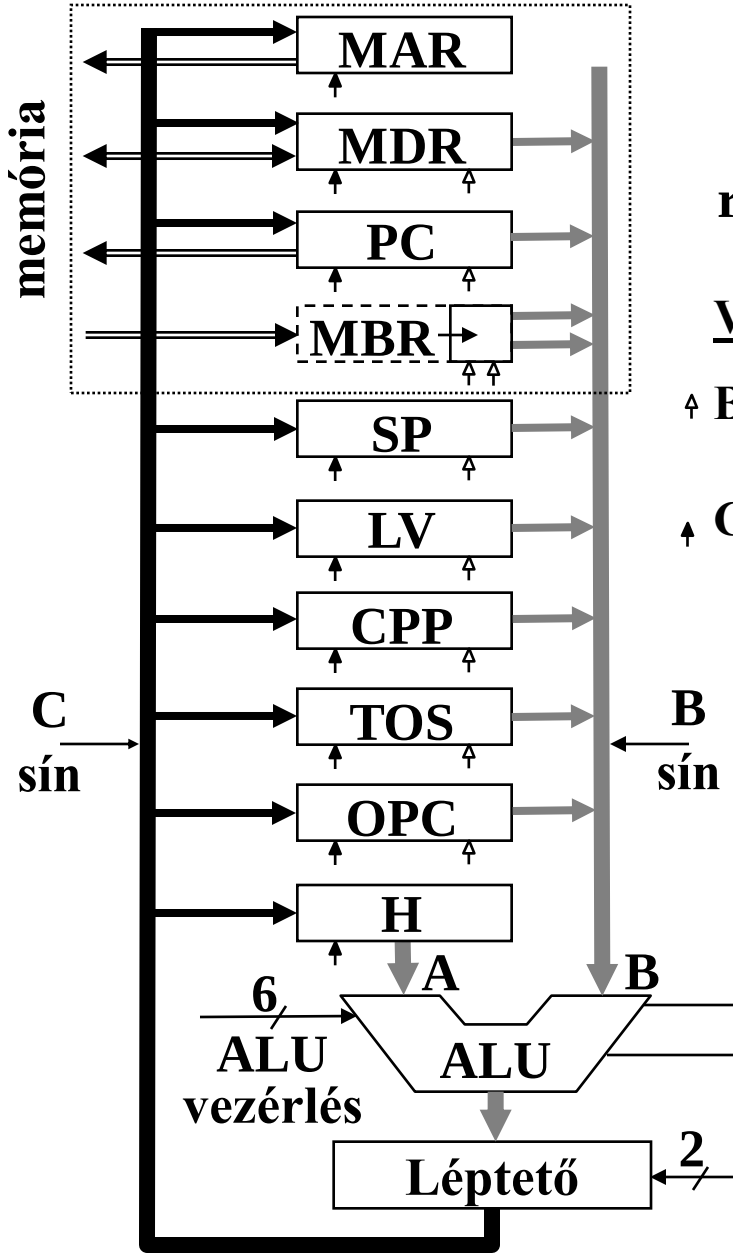
Mikroarchitektúra szint

Feladata az **ISA** (Instruction Set Architecture – gépi utasítás szint) megvalósítása.
Nincs rá általánosan elfogadott, egységes elv.

A **ISA**-szintű utasítások „függvények”, ezeket egy főprogram hívja meg végtelen ciklusban.

A függvények a mikroarchitektúra szintjén valósulnak meg (mikroprogram).

A mikroprogram változói (a regiszterek) definiálják a számítógép állapotát, pl.: **PC** (Program Counter, utasításszámláló).



Memória vezérlő regiszterek

Vezérlő jelek

- **B** sínre írja a regisztert
- **C** sínt a regiszterbe írja

Mic-1

Adatút (Data Path, 4.1. ábra)

32 bites regiszterek, sínek,

ALU, léptető

SLL8 8 bittel balra,

SRA1 *1 bittel jobbra léptet.*

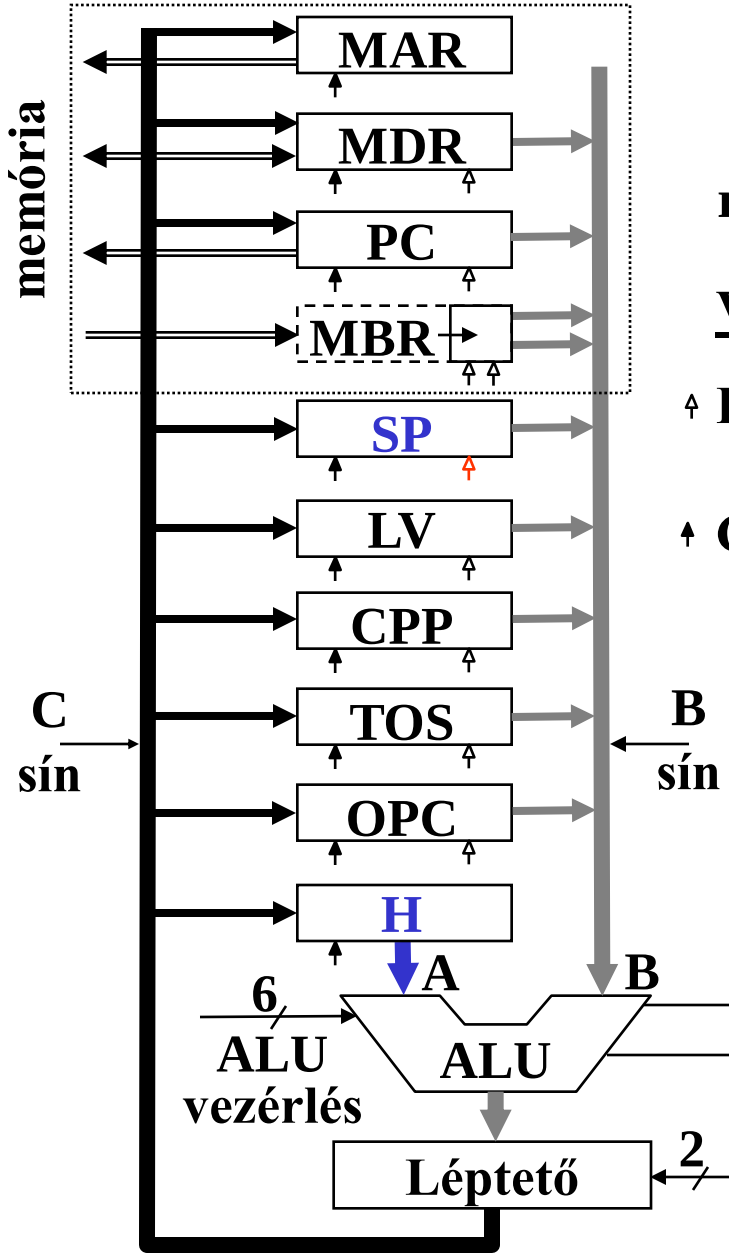
ALU bemenetei:

H (Holding – tartó), **B** sín.

Egy cikluson belül lehetséges egy regiszterből olvasni és az eredményt akár ugyanoda visszaírni **4.3. ábra**.

N **1**, ha az eredmény < **0**, különben **0**,

Z **1**, ha az eredmény = **0**, különben **0**.



Memória vezérlő regiszterek

Vezérlő jelek

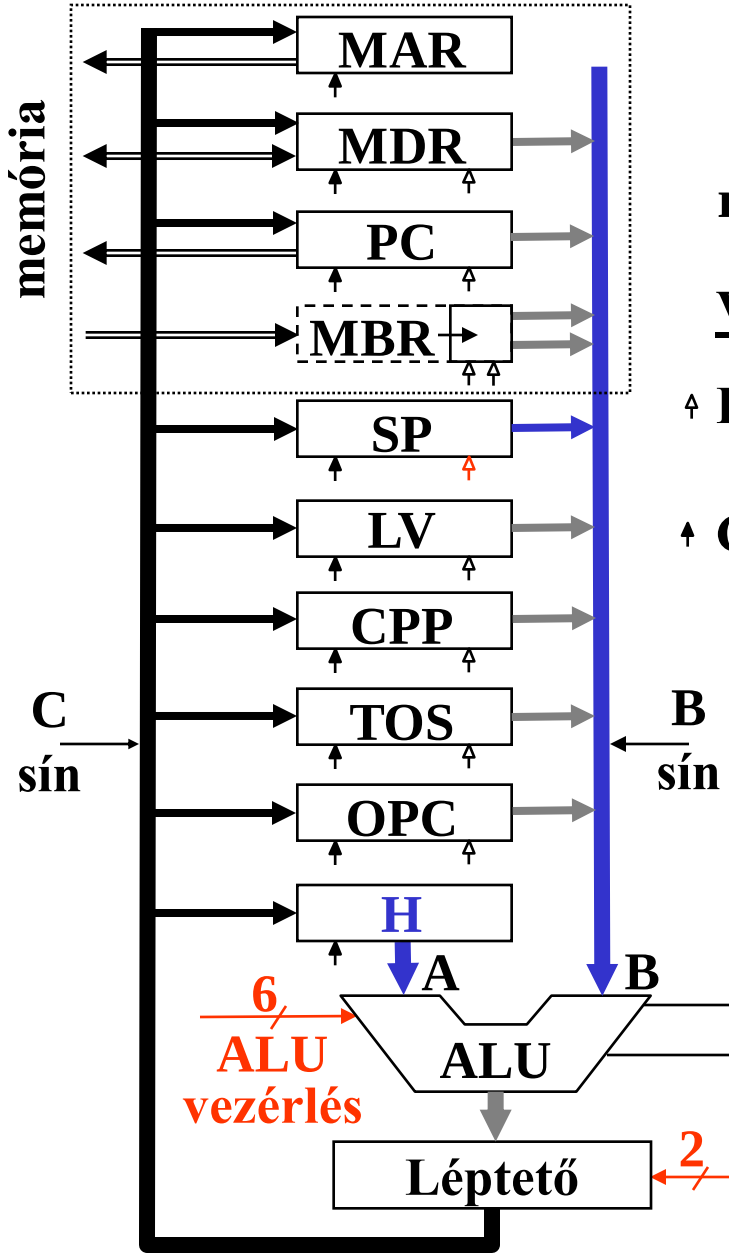
- **B sínre írja a regisztert**
- **C sínt a regiszterbe írja**

- **SP ==> B sín**
- **ALU: B+1**
- **C ==> SP**

N **1**, ha az eredmény < **0**, különben **0**,

Z **1**, ha az eredmény = **0**, különben **0**.

Léptető vezérlés



Memória vezérlő regiszterek

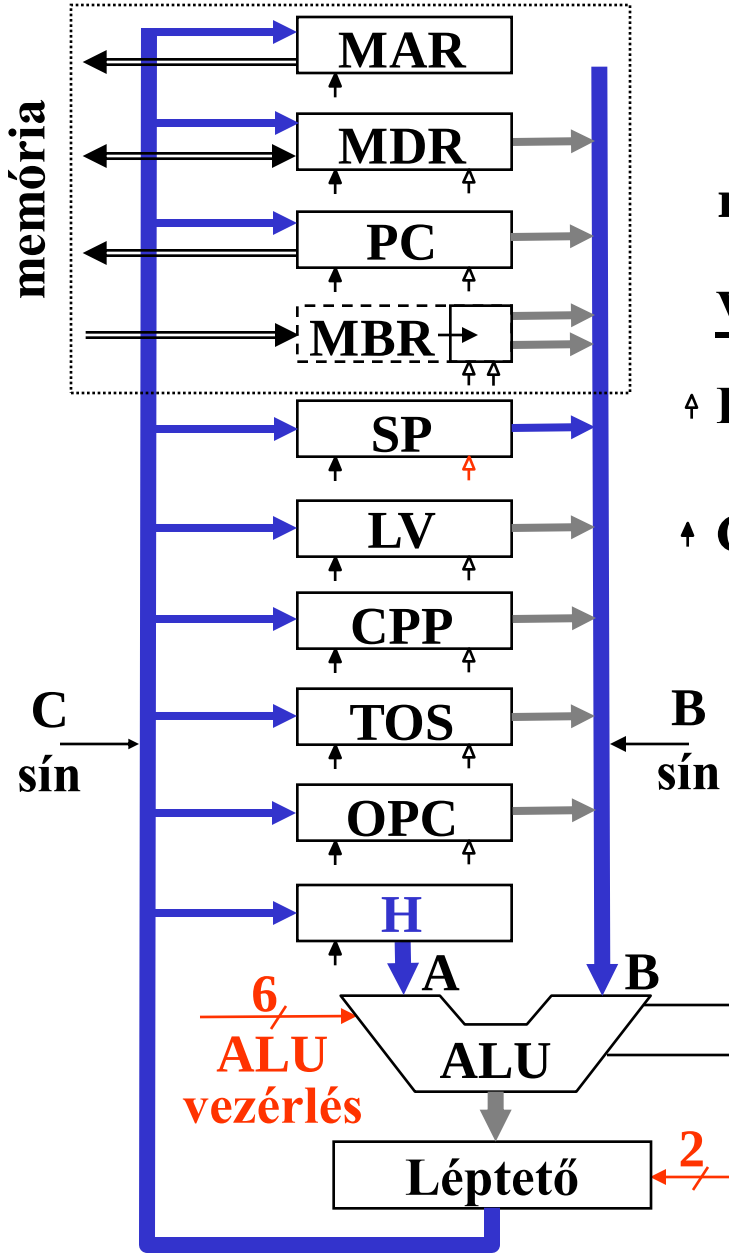
Vezérlő jelek

- **B sínre írja a regisztert**
- **C sínt a regiszterbe írja**

- **SP ==> B sín**
- **ALU: B+1 ==> C**
- **C ==> SP**

N **1**, ha az eredmény < **0**, különben **0**,

Z **1**, ha az eredmény = **0**, különben **0**.



Memória vezérlő regiszterek

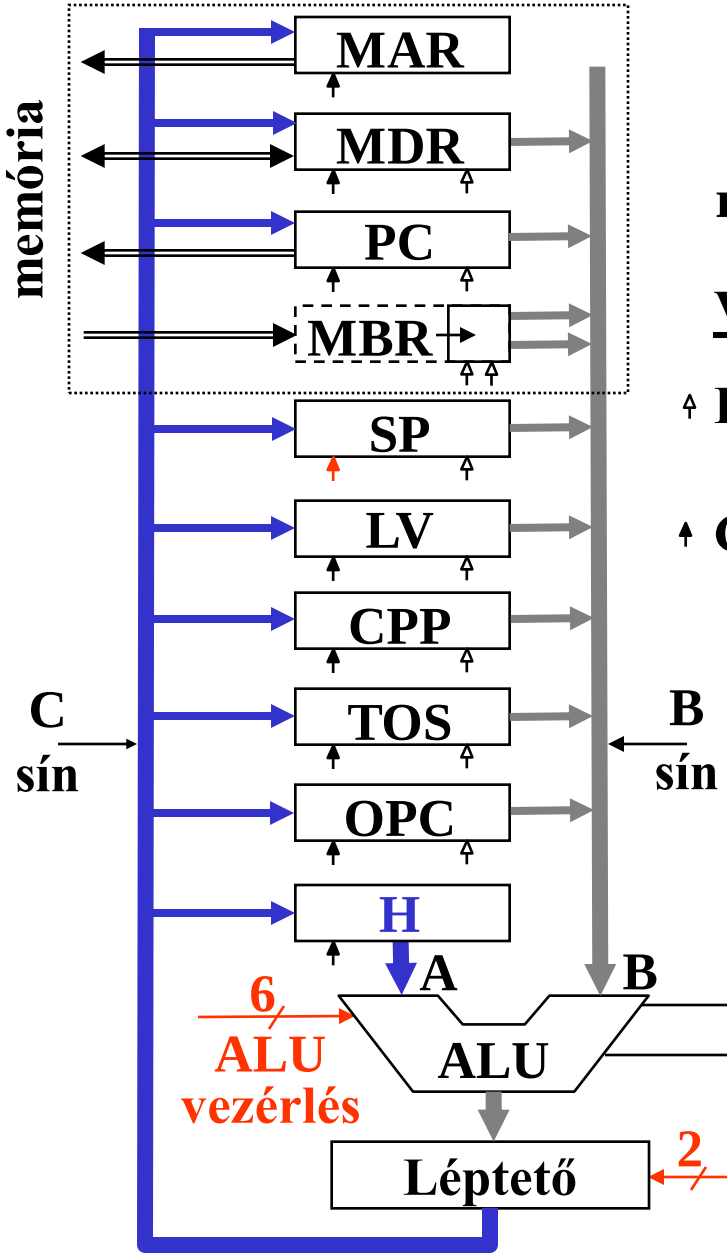
Vezérlő jelek

- ↑ **B sínre írja a regisztert**
- ↑ **C sínt a regiszterbe írja**

- **SP ==> B sín**
- **ALU: B+1 ==> C**
- **C ==> SP**

N **1**, ha az eredmény < **0**, különben **0**,

Z **1**, ha az eredmény = **0**, különben **0**.



Memória vezérlő regiszterek

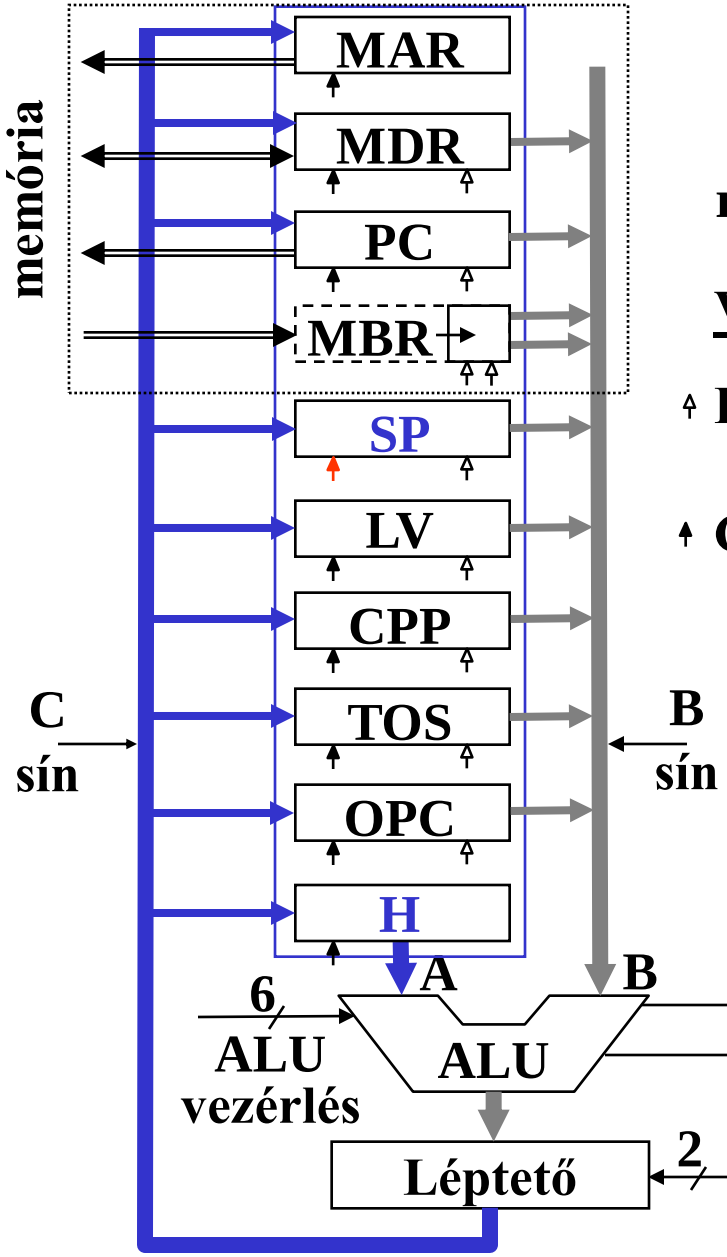
Vezérlő jelek

- ↑ **B sínre írja a regisztert**
- ↑ **C sínt a regiszterbe írja**

- **SP ==> B sín**
- **ALU: B+1 ==> C**
- **C ==> SP**

N **1**, ha az eredmény < **0**, különben **0**,

Z **1**, ha az eredmény = **0**, különben **0**.



Memória vezérlő regiszterek

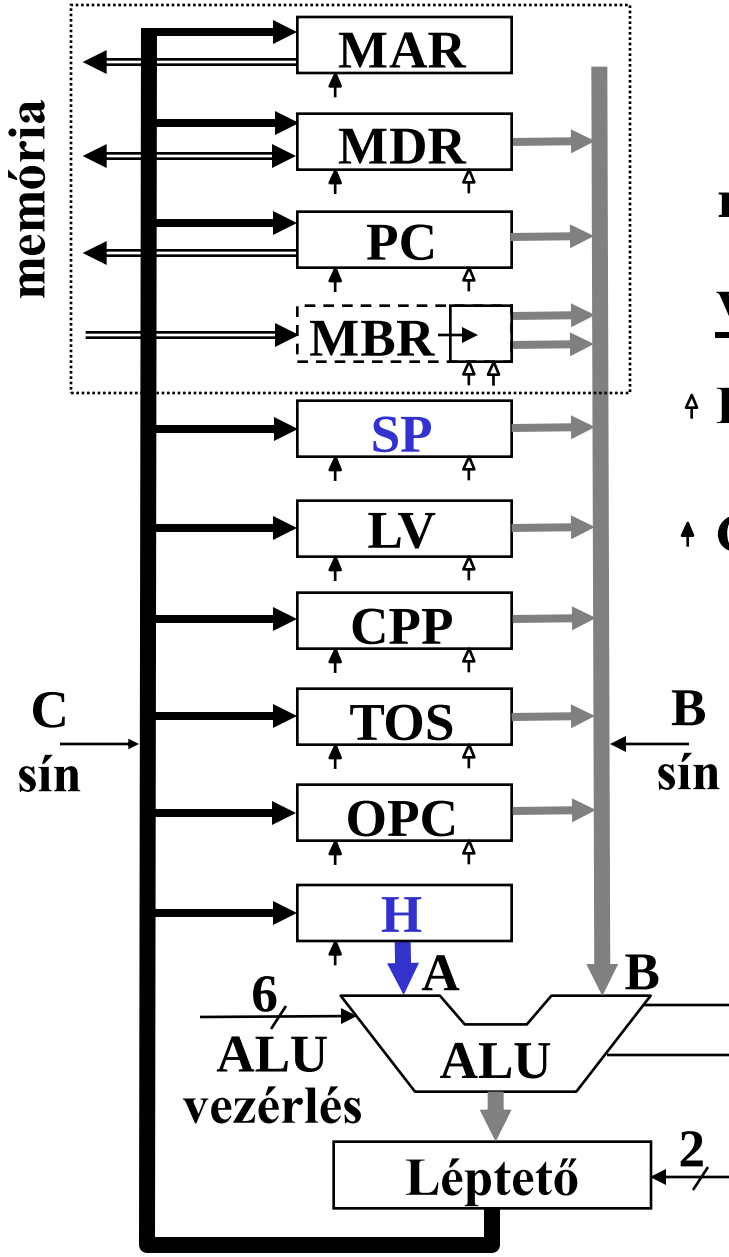
Vezérlő jelek

- **B sínre írja a regisztert**
- **C sínt a regiszterbe írja**

- **SP ==> B sín**
- **ALU: B+1 ==> C**
- **C ==> SP**

N **1**, ha az eredmény < **0**, különben **0**,

Z **1**, ha az eredmény = **0**, különben **0**.



Memória vezérlő regiszterek

Vezérlő jelek

- **B sínre írja a regisztert**
- **C sínt a regiszterbe írja**

- **SP ==> B sín**
- **ALU: B+1**
- **C ==> SP**

N **1**, ha az eredmény < **0**, különben **0**,

Z **1**, ha az eredmény = **0**, különben **0**.

Léptető vezérlés

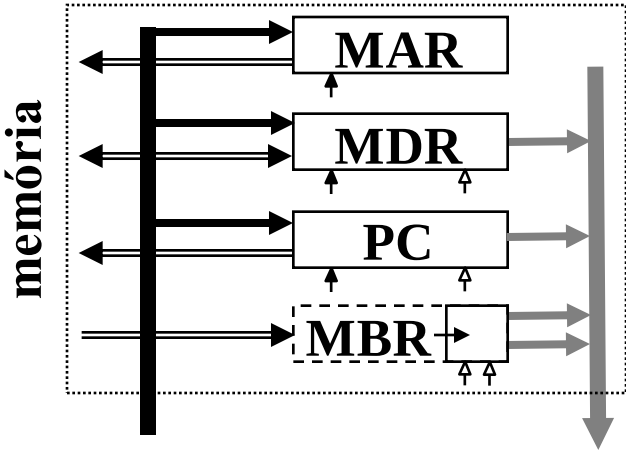
Memóriaműveletek (4.1. ábra)

- **Szócímzés:** 32 bites adat írása, olvasása.

  szó cím = 4 * (bájt cím),
  a túlcsorduló bitek elvesznek

  MAR (Memory Address Register)

  MDR (Memory Data Register)



- **Bájtcímzés**: gépi szintű utasítás bájt olvasás.

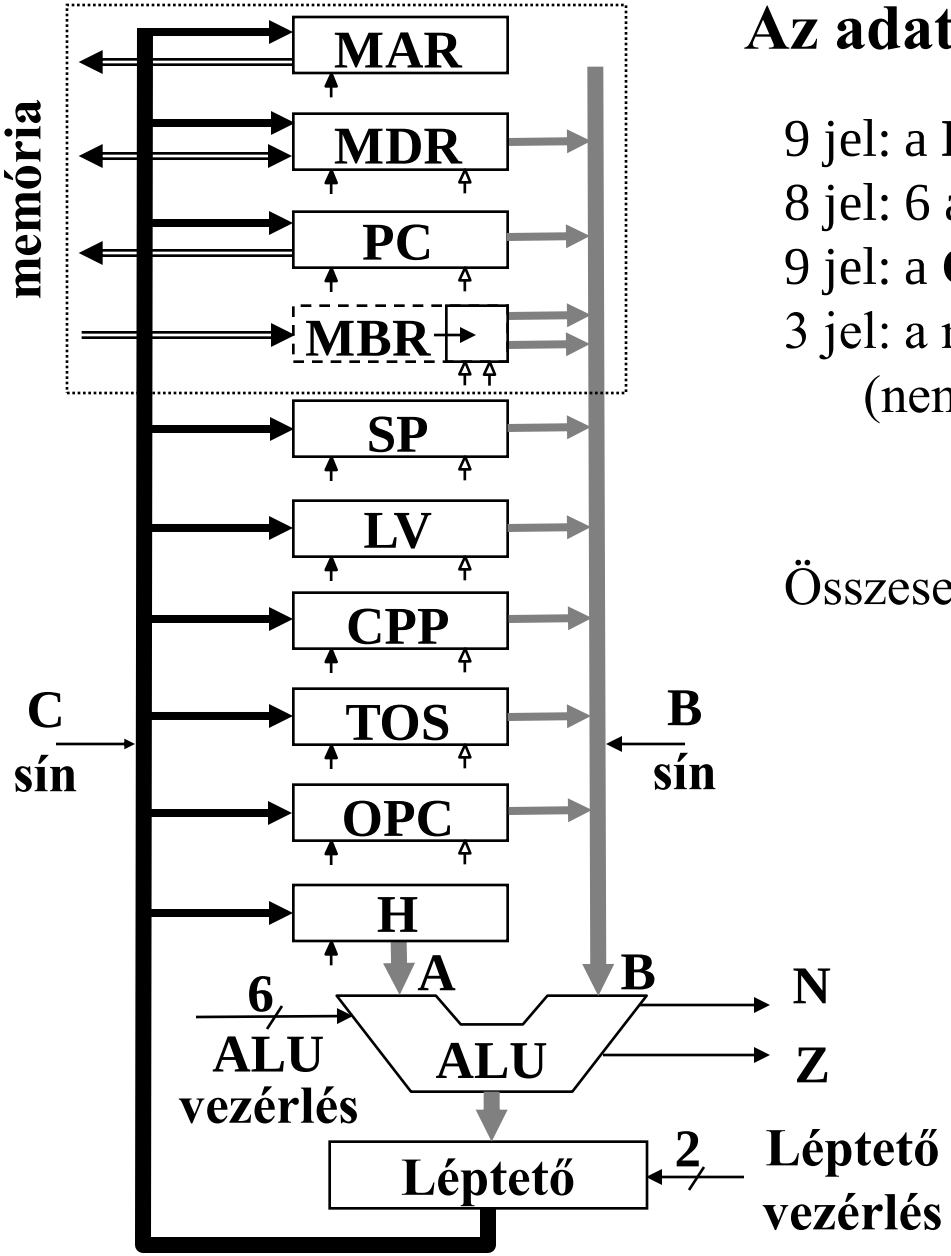
  PC (Program Counter): bájt cím,

  MBR (Memory Byte Register): bájt.

MBR kétfajta értelmezése (két vezérlőjel):

- **MBR: MBR** előjel kiterjesztéssel kerül a **B** sínre,
- **MBRU: MBR** előjel nélküli kiterjesztéssel kerül a **B** sínre.

Az adatút vezérlése (4.1. ábra)



9 jel: a **B** sínre írás a regiszterekből,
8 jel: 6 az **ALU** és 2 a **léptető** vezérlésére,
9 jel: a **C** sínről regiszterekbe írás,
3 jel: a memória eléréshez
(nem ábrázoltuk!)
2 jel: szó íráshoz/olvasáshoz
1 jel: bájt olvasáshoz.
Összesen 29 jel szükséges

A **B** sínre csak egy regiszter írhat egyszerre, ezért 9 helyett elég 4 jel, összesen

24 vezérlő jelre van szükség.

Mikroutasítások

24 bit: az adatút vezérléséhez

9 bit: a következő utasítás címének megadásához,

3 bit: a következő utasítás kiválasztásának módjára.

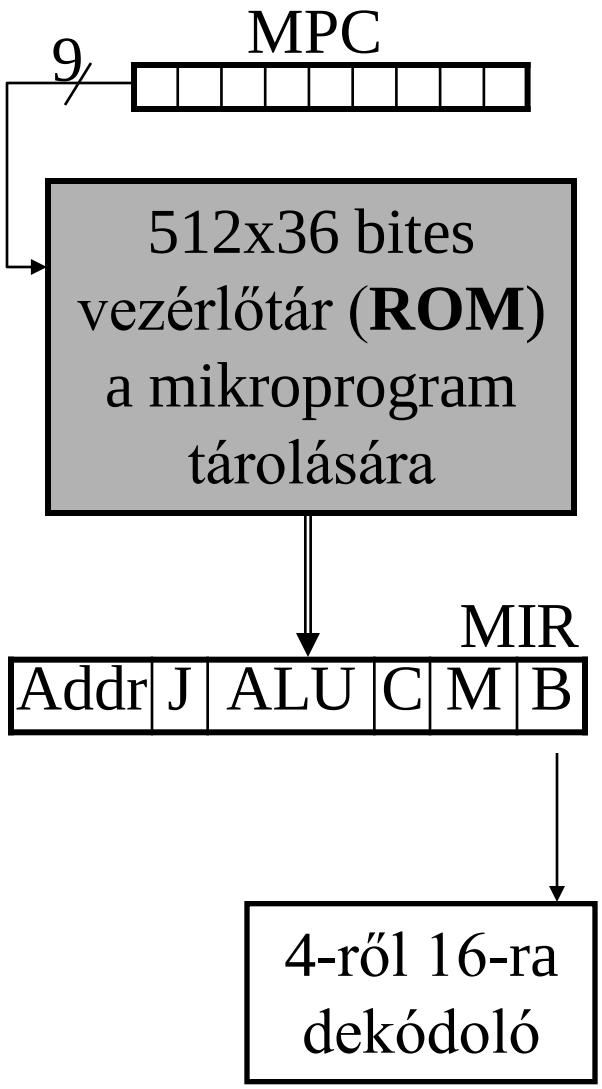
Ez adja a 36 bites mikroutasítást: **4.5. ábra**.

9	3	8	9	3	4
NEXT ADDRESS	JMPC JAMN JAMZ	SLL8 SRA1 F0 F1 ENA ENB INVA INC	H OPC TOS LV SP PC MDR MAR	WRITE READ FETCH	B sín
Addr	JAM	ALU	C	Mem	B

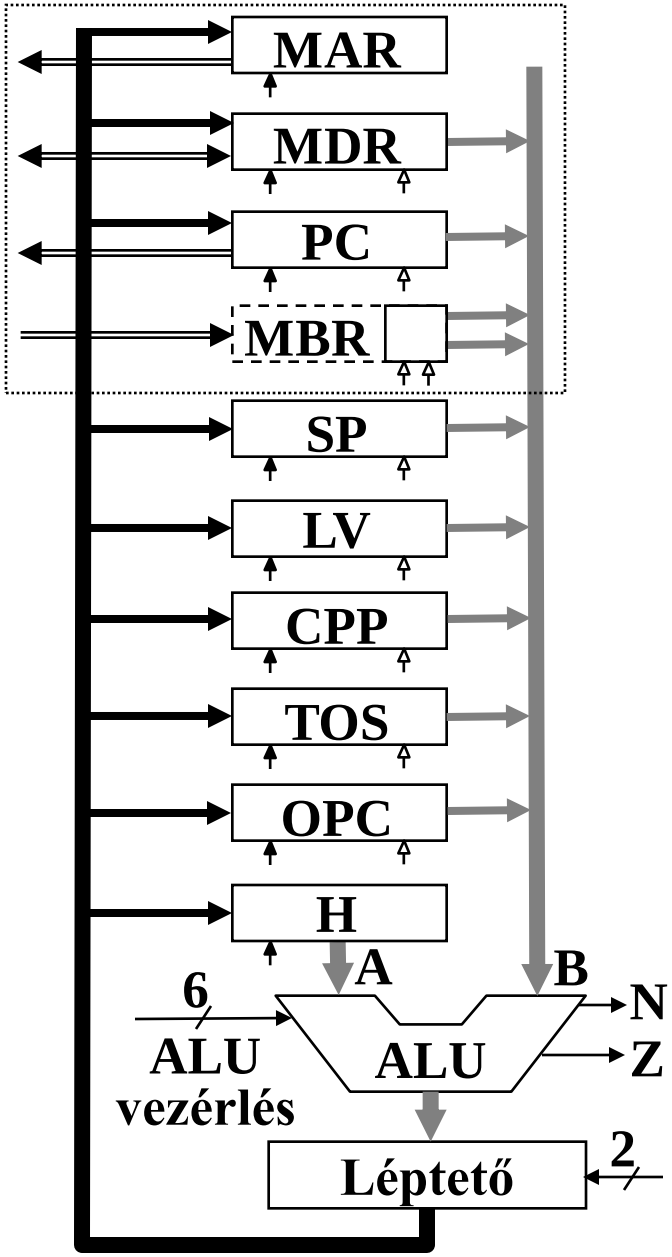
0 = MDR	1 = PC	2 = MBR	3 = MBRU	4 = SP
5 = LV	6 = CPP	7 = TOS	8 = OPC	9-15 semmi

Mic-1: 4.6. ábra.

- 512x36 bites vezérlőtár a mikroprogramnak,
- **MPC** (MicroProgram Counter): mikroprogram-utasításszámláló.
- **MIR** (MicroInstruction Register): mikroutasítás-regiszter.



Az adatút ciklus **(4.6. ábra)** elején **MIR** feltöltődik a vezérlőtár **MPC** által mutatott szavával.



Adatút ciklus (4.6. ábra):

• (**MIR** feltöltődik a vezérlőtár **MPC** által mutatott szavával.)

• Kialakul a **B** sín kívánt tartalma, **ALU** és a **léptető** megtudja, mit kell csinálnia,

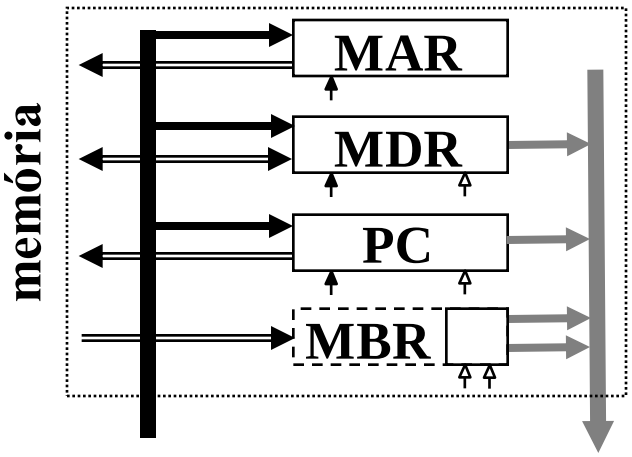
• Az **ALU** és a **léptető** elvégzi a feladatát, a **C** sín, **N** (Negative) és **Z** (Zero) megkapja az új értékét,

• A regiszterek feltöltődnek a **C** sínről. **MBR/MDR** megkapja az értékét, ha az előző ciklus adatot kért a memóriából.

• Kialakul **MPC** új értéke.

• Memória ciklus kezdete.

Memória ciklus

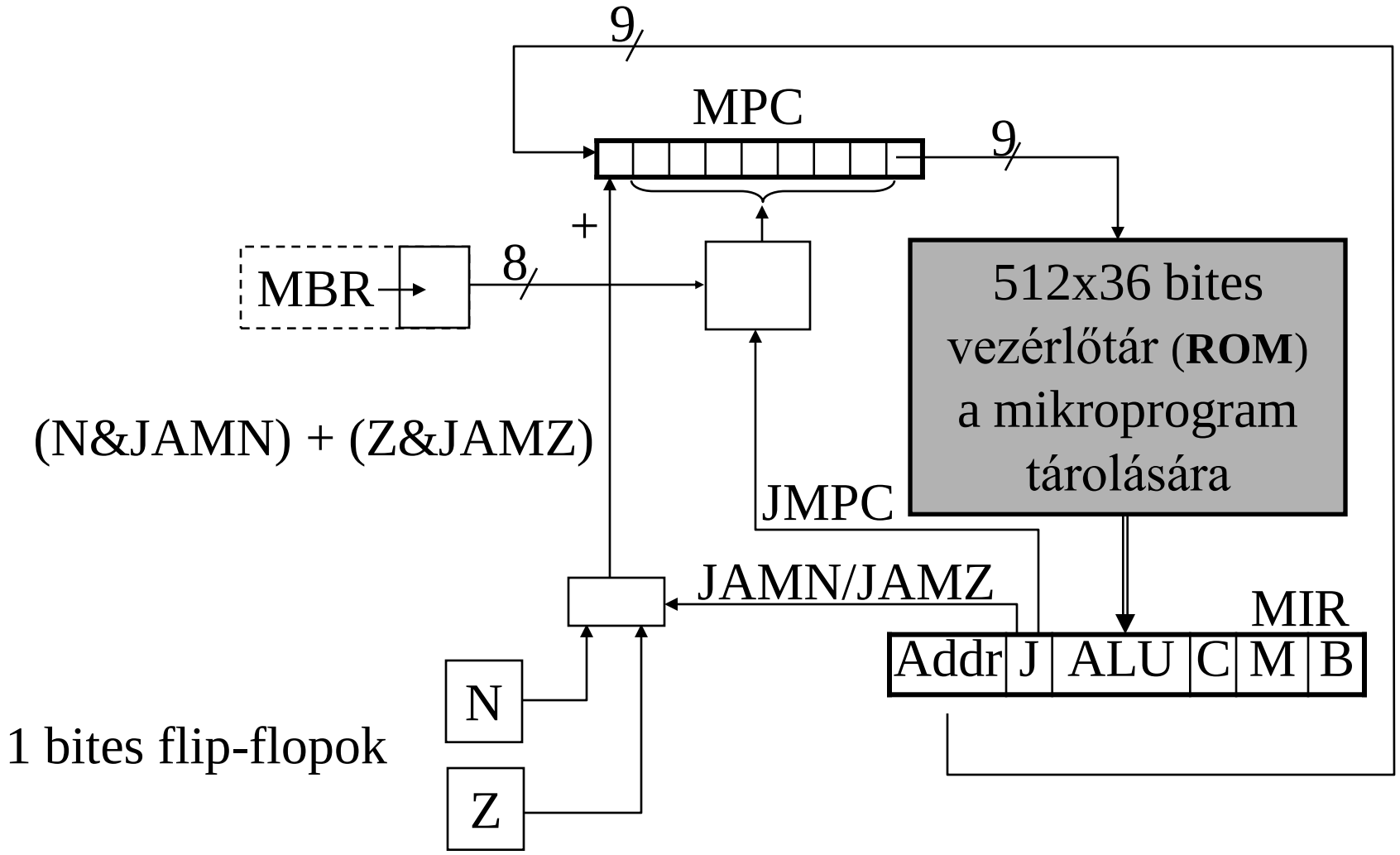


A memória ciklus az adatút végén kezdődik (**MAR** ill. **PC** feltöltése után), ezért ha a memória címet módosította ez a mikroutasítás, akkor a memória cím a módosított **MAR** ill. **PC** regiszter értéke lesz.

Az olvasás eredménye csak két ciklussal később használható az **ALU**-ban, mert **MDR** ill. **MBR** csak a következő adatút ciklus vége felé töltődik fel a memóriából, addig **MDR** ill. **MBR** régi értéke érhető el.

Mic-1: 4.6. ábra.

MPC új tartalmának kialakítása.



MPC új tartalma

- A 9 bites következő cím (**Addr**) az **MPC**-be kerül.
- **JAMN/JAMZ** esetén **MPC** legmagasabb bitjének és az **N/Z** bitnek logikai vagy kapcsolata képződik **MPC** legmagasabb helyértékével (elágazás). Pl.:

Cím	**Addr**	**JAM**	**Adatút vezérlő bitek**	
0x75	**0x092**	**001**	**. . .**	**JAMZ =1**

esetén a mikroprogram a

0x092 címen folytatódik, ha **Z = 0**,

0x192 címen folytatódik, ha **Z = 1**.

Feltételes ugrás – elágazás – a **mikroprogramban**.

MPC új tartalma (folytatás)

- **JMPC** esetén **MPC** 8 alacsonyabb helyértékű bitjének és **MBR** 8 bitjének bitenkénti vagy kapcsolata képződik **MPC**-ben az adatút ciklus vége felé (**MBR** megérkezése után).
  Ilyenkor **Addr** 8 alacsonyabb helyértékű bitje általában **0**
  Feltétlen ugrás az **MBR** –ben tárolt címre – kapcsoló utasítás.

Kezdődhet az újabb mikroutasítás végrehajtása.

Mic-1 működése

• (**MPC**) ⇒ **MIR**	
• regiszter ⇒ **B** sín, **ALU**, léptető megtudja, mit kell csináljon,	**Addr** ⇒ **MPC**
• eredmény ⇒ **C**, **N**, **Z**	
• **C** ⇒ regiszterekbe mem. ⇒ **MDR** és/vagy mem. ⇒ **MBR**	**JAMN**, **JAMZ** (**N**, **Z**) alapján módosul **MPC**
• Memória ciklus indítása (**rd**, **wr**, **fetch**)	**JMPC**(**MBR**) alapján módosul **MPC**.

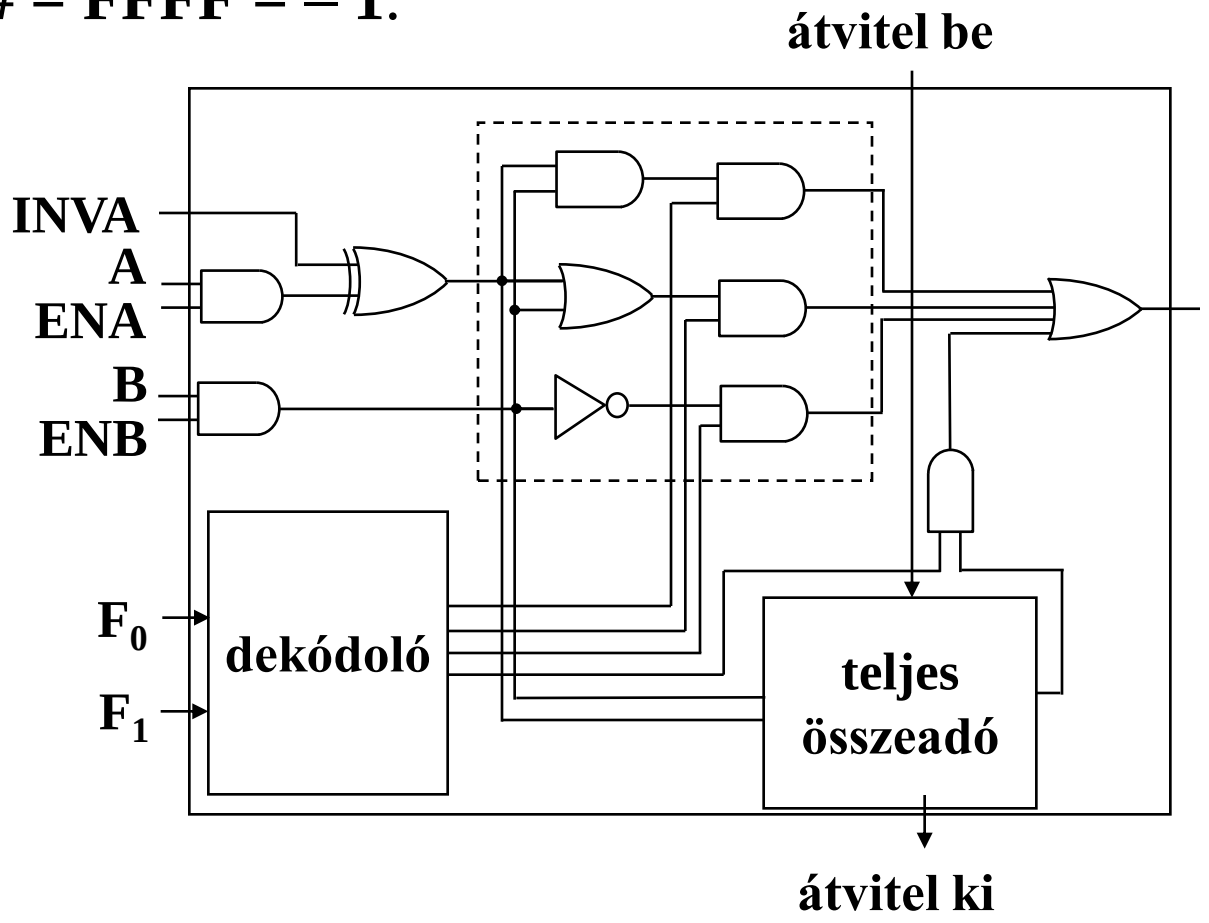
Az **ALU**-nak (**3.19-20. ábra**) 6 vezérlő bemenete van:

- **INVA**:

  Ha **ENA = 1**, akkor ***A#***,

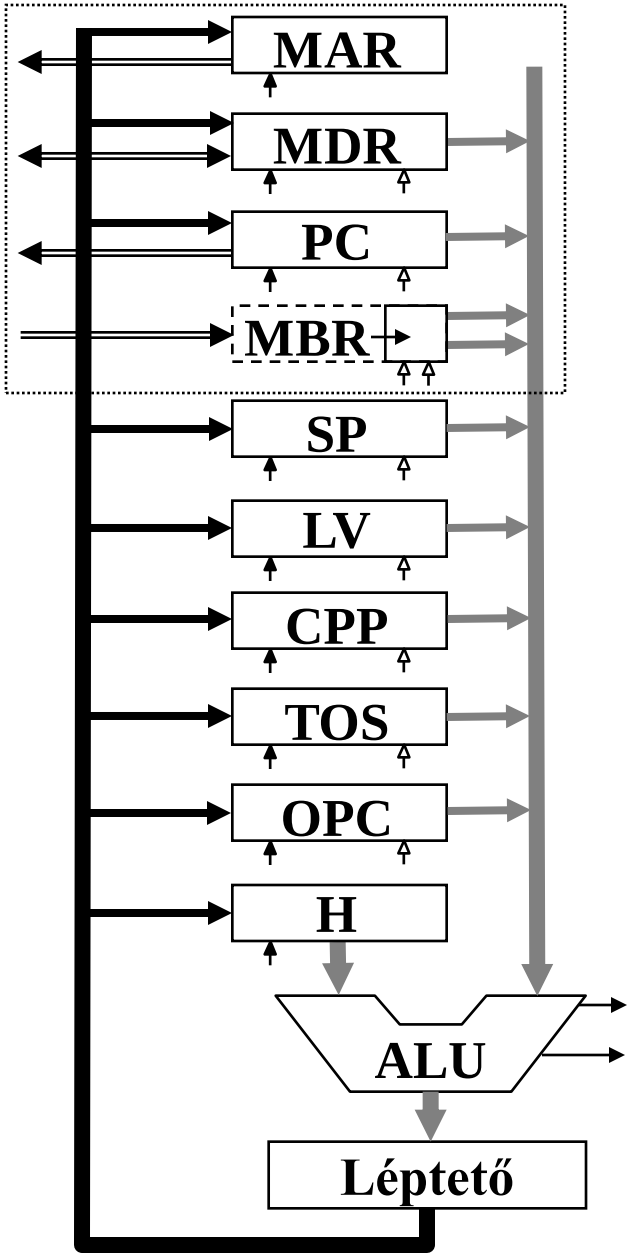
  Ha **ENA = 0**, akkor **0# = FFFF = – 1**.

- **ENA** az ***A*** bemenet engedélyezése (**1**) tiltása (**0**),
- **ENB** a ***B*** bemenet engedélyezése (**1**) tiltása (**0**),
- **F0**, **F1** kiválasztja az ***AND***, ***OR***, ***B#***, **+** művelet valamelyikét,
- **INC**: **+1**.



Néhány példa (**4.2. ábra**)

F_0	F_1	ENA	ENB	INVA	INC	Eredmény
0	1	1	0	0	0	**A**
0	1	0	1	0	0	**B**
0	1	1	0	1	0	**#A**
1	0	*0*	1	0	0	**#B**
1	1	1	1	0	0	**A + B**
1	1	1	1	0	1	**A + B + 1**
1	1	1	0	0	1	**A + 1**
1	1	0	1	0	1	**B + 1**
1	1	1	1	1	1	**B – A**
1	1	0	1	1	0	**B – 1**
1	1	1	0	1	1	**– A**
0	0	1	1	0	0	**A AND B**
0	1	1	1	0	0	**A OR B**
1	1	0	0	0	0	**0**
1	1	0	0	0	1	**1**
1	1	0	0	1	0	**– 1**



MAL (Micro Assembly Language)

SOURCE: a **B** sínre kötött regiszterek bármelyike: **MDR**, **PC**, **MBRU** (előjel nélküli - **U**nsigned **MBR**) **MBR**, **SP**, **LV**, **CPP**, **TOS**, **OPC**.

DEST: a **C** sínre kapcsolt regiszterek bármelyike: **MAR**, **MDR**, **PC**, **SP**, **LV**, **CPP**, **TOS**, **OPC**, **H**.
Több regiszter is kaphatja ugyanazt az értéket.

wr: memóriába írás **MDR**-ből a **MAR** címre.

rd: memóriából olvasás **MDR**-be a **MAR** címről.

fetch: 8 bites utasításkód betöltése **MBR**-be a **PC** címről.

IJVM (Integer Java Virtual Machine): a **JVM** egész értékű aritmetikát tartalmazó része.

Az IJVM utasítások szerkezete:

- az első mező az **opcode** (Operation Code, műveleti kód),
- az esetleges második mezőben az **operandus** meghatározására szolgáló adat van.

Mikroprogram: betölti, értelmezi és végrehajtja az **IJVM** utasításokat:
betöltés-végrehajtás (fetch-execute) ciklus.

Az IJVM memóriamodellje (4.10. ábra)

A **4 GB** memória, **1 G** szóként is szervezhető.

Konstansok, mutatók

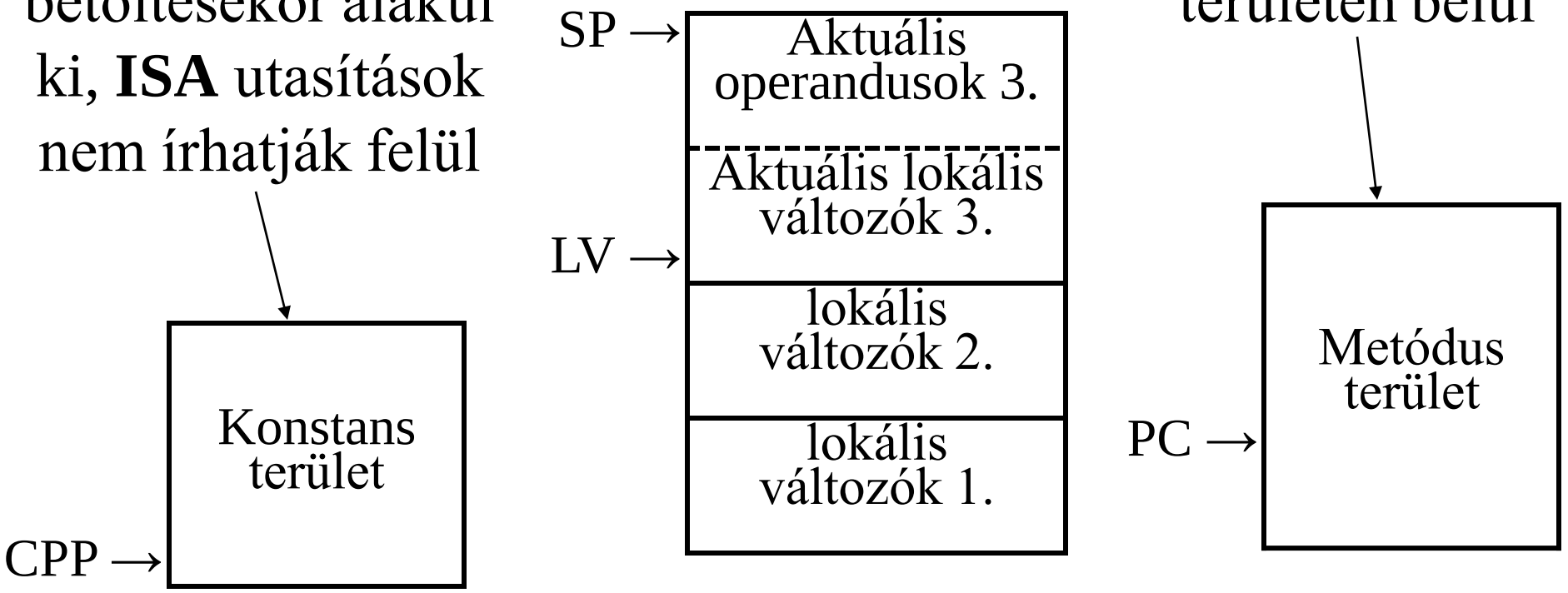
Tartalma a program betöltésekor alakul ki, **ISA** utasítások nem írhatják felül

Verem

lokális változók és operandus verem

Program

PC bájtot címez a metódus területen belül



IJVM néhány utasítása: **4.11. ábra**.

hex	**Mnemonic**	**jelentés**
10	BIPUSH *byte*	Beteszi a *byte*–ot a verembe
A7	GOTO *offset*	Feltétel nélküli ugrás *offset* -re
60	IADD	Kivesz a veremből két szót, az összegüket a verembe teszi
99	IFEQ *offset*	Kivesz a veremből egy szót, ha 0, akkor *offset* -re ugrik
9F	IF_ICMPEQ *offset*	Kivesz a veremből két szót, ha egyenlők, akkor *offset* -re ugrik
15	ILOAD *varnum*	Beteszi *varnum* -ot a verembe
36	ISTORE *varnum*	Kivesz a veremből egy szót, és eltárolja *varnum* -ba
64	ISUB	Kivesz a veremből két szót, a különbségüket a verembe teszi
00	NOP	Nem csinál semmit
5F	SWAP	A verem két felső szavát megcseréli

Java (C) program	IJVM program 4.14. ábra				Bin. kód
	1		ILOAD	j // i = j + k	15 02
	2		ILOAD	k	15 03
i = j + k;	3		IADD		60
if(i = = 3)	4		ISTORE	i	36 01
k = 0;	5		ILOAD	i // if(i = = 3)	15 01
else	6		BIPUSH	3	10 03
j = j – 1;	7		IF_ICMPEQ	L1	9F 00 0D
	8		ILOAD	j // j = j – 1	15 02
	9		BIPUSH	1	10 01
	10		ISUB		64
	11		ISTORE	j	36 02
	12		GOTO	L2	A7 00 0F
	13	L1:	BIPUSH	0 // k = 0	10 00
	14		ISTORE	k	36 03
	15	L2:			

IJVM megvalósítása Mic-1-en (4.11., 17. ábra)

Előkészület a gép indításakor: **PC** a végrehajtandó utasítás címét, **MBR** magát az utasítást tartalmazza.

A főciklus legelső mikroutasítása a **Main1**, ez:

PC=PC+1; fetch; goto(MBR);

PC most a végrehajtandó utasítás utáni bájtra mutat, ez lehet egy újabb utasítás kódja, vagy operandus.

PC új értékének kialakulása után indul a **fetch**-csel kezdeményezett memória ciklus, ez a program következő bájtját olvassa **MBR**-be (a következő mikroutasítás végén lesz **MBR**-ben a bájt).

goto (MBR) elugrik az utasítás feldolgozásához.

Megszakítás rendszer, interrupt utasítások

- Az **I/O** utasítás 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.
  - **Megszakítás**

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.

Csapda és megszakítás

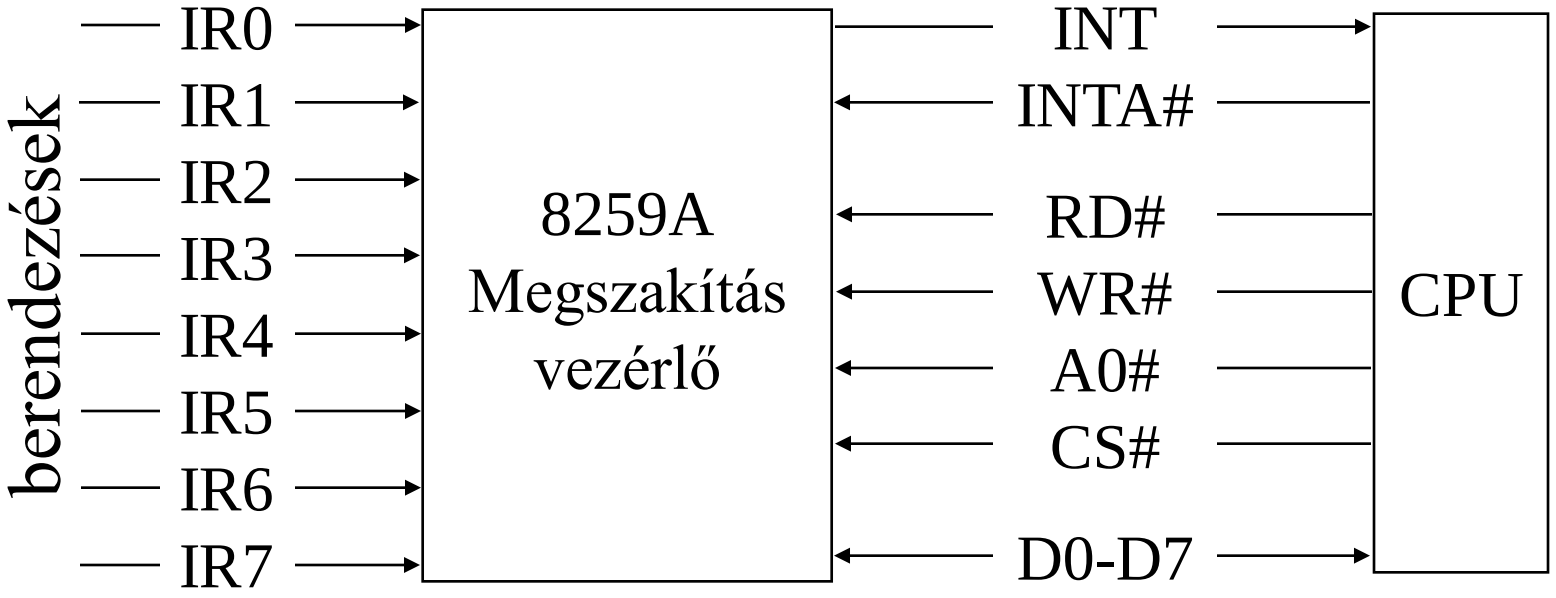
Csapda (trap): A program által előidézett feltétel (pl. túlcsordulás) hatására automatikus eljárás hívás. **Csapda kezelő**. (Eltérülés)

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.

Megszakítás kezelés (3.43. ábra)



IRi $\rightarrow$, **INT** $\rightarrow$, ha **CPU** tudja fogadni, akkor **INTA#** $\rightarrow$, **i** $\rightarrow$ **D0-D7**, a **CPU** megszakításvektor táblázat **i** –edik eleméből tudja a megszakítást kiszolgáló eljárás kezdőcímét, 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 (EF regiszter),
6. a **CPU** az azonosító indexű megszakítás kezelő címét teszi az utasításszámlálóba és gyakran betölti vagy módosítja **PSW**-t.

Szoftver tevékenységek (kiíráskor)**:**

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, sokszor 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.

Ha sok eszköz van a rendszerben, akkor célszerű, ha egy megszakítás kiszolgálása közben másik megszakítás is történhet, ilyenkor a megszakítások hierarchiába vannak rendezve (**5.46. ábra**).

I8086/88

Az **i.** megszakítási okhoz tartozó megszakító rutin **FAR** címe a memória **4*i.** címén található (**0 ≤ i ≤ 255**). A megszakítási ok sorszámát hardver megszakítás esetén a hardver egység installláskor adott száma, szoftver megszakítás esetén az operandus rész szolgáltatja.

Megszakítási ok jelentkezésekor a **STATUS**, **CS** és **IP** a verembe kerül, az **I** és a **T** flag **0** értéket kap (az úgynevezett maszkolható megszakítások tiltása és folyamatos üzemmód beállítása), majd **(CS:IP)** felveszi a megszakítás kezelő kezdőcímét.

Interrupt utasítások

Szoftver megszakítást eredményeznek.

INT i ; 0 <= i <= 255,

; megszakítás az **i.** ok szerint.

Az **INT (=INT 3)** utasítás kódja csak egy byte (a többi 2 byte), így különösen alkalmas nyomkövető (**DEBUG**) programokban történő alkalmazásra.

A **DEBUG** program saját magához irányítja a **3**-as megszakítást. Az ellenőrzendő program megadott pontján (törés pont, **break point**) lévő utasítást (annak 1. bájtját) átmenetileg az **INT 3** utasításra cseréli, és átadhatja a vezérlést az ellenőrzendő programnak. Amikor a program az **INT 3** utasításhoz ér, a megszakítás hatására a **DEBUG** kapja meg a vezérlést. Kiírja a regiszterek tartalmát, és további információt kérhetünk a program állapotáról.
Később visszaírja azt a tartalmat, amit **INT 3** -ra cserélt, elhelyezi az újabb törés pontra az **INT 3** utasítást és visszaadja a vezérlést az ellenőrzendő programnak.

INTO ; megszakítás csak **O=1 (Overflow)**
; esetén a **4.** ok szerint

Visszatérés a megszakító rutinból

IRET ; **IP, CS, STATUS** feltöltése a
; veremből

Szemafor

Legyen az **S** szemafor egy olyan word típusú változó, amely mindegyik program számára elérhető. Jelentse **S=0** azt, hogy az erőforrás szabad, és **S≠0** azt, hogy az erőforrás foglalt. Próbáljuk meg a szemafor kezelését!

; 1. kísérlet

```
ujra:  mov     cx,S
       jcxz    szabad
       . . .   ; foglalt az erőforrás, várakozás
       jmp     ujra
szabad: mov      cx,0FFFFh
       mov     S,cx   ; a program lefoglalta az erőforrást
```

```
 ; 2. kísérlet
ujra: MOV CX,0FFFFH
      XCHG  CX,S ; már foglaltat jelez a
                 ; szemafor!
      JCXZ  szabad  ; ellenőrzés S korábbi tartalma
                 ; szerint.
      . . .      ; foglalt az erőforrás,
                 ; várakozás
      jmp   ujra
szabad: . . .        ; szabad volt az erőforrás,
                 ; de a szemafor már foglalt
```

Az **XCHG** utasítás mikroprogram szinten:

Segéd regiszter <== S; S <== CX; CX <== Segéd regiszter

olvasás – módosítás – visszaírás

```
ujra:  mov     cx,0FFFFh
LOCK   xchg    cx,S       ; S már foglaltat jelez
       jcxz    szabad     ; ellenőrzés S korábbi
                          ; tartalma szerint
       . . .              ; foglalt, várakozás
       jmp     ujra
szabad: . . .             ; használható az erőforrás,
                          ; de a szemafor már foglalt
       . . .
       MOVS,0             ; a szemafor szabadra állítása
```

RISC – CISC

RISC: Reduced Instruction Set Computer
csökkentett utasításkészletű számítógép

CISC: Complex Instruction Set Computer
összetett utasításkészletű számítógép

A 70-es években nagyon sok bonyolult utasítást építettek a gépekbe, mert a **ROM**-oknak a **RAM**-okhoz viszonyított nagy sebessége a mikroprogram gyors lefutását – a bonyolult utasítás viszonylag gyors végrehajtását eredményezte --> **CISC**. Nem volt ritka a 200-300 utasítással rendelkező gép.

A RISC kialakulása

IBM-801 (John Cocke) Seymour Cray ötletei alapján nagy teljesítményű miniszámítógép. Nem került piacra, csak 1982-ben publikálták.

Berkeley 1980 (David Petterson, Carlo Séquin) **RISC I**, később **RISC II**

Stanford 1981 (John Hennessy) **MIPS → SPARC**

Elv: Csak olyan utasítások legyenek, amelyek az adatút egyszeri bejárásával végrehajthatók.

Tipikusan kb. 50 utasításuk van.

Ha egy **CICS** utasítás 4-5 **RISC** utasítással helyettesíthető, és a **RISC** 10-szer gyorsabb, akkor is a **RISC** nyer.

Időközben a **RAM-**ok sebessége csaknem elérte a **ROM-**ok sebességét, ez is a **RISC** mellett szól.

K O M P A T I B I L I T Á S

Az **Intel** túlélte: a **486-**os processzortól kezdődően minden processzora tartalmaz **RISC** magot, amely a legegyszerűbb, és egyben leggyakoribb utasításokat egyetlen adatút ciklus alatt hajtja végre, csak a többit – a ritkábban előfordulókat – interpretálja a **CISC** elvnek megfelelően --> versenyképes maradt.

Korszerű számítógépek (**RISC**) tervezési elvei

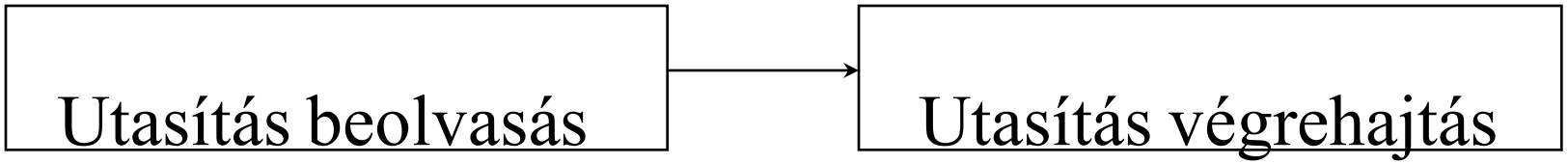
- Minden utasítást közvetlenül a hardver hajtson végre
- Maximalizálni az utasítások kiadásának ütemét
- Az utasítások könnyen dekódolhatók legyenek
- Csak a betöltő és tároló utasítások hivatkozzanak a memóriára

--> Sok (legalább 32) regiszter kell

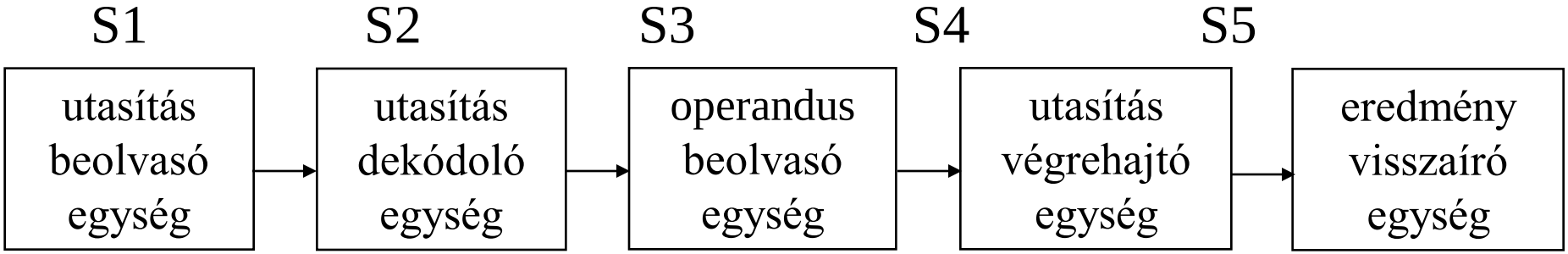
Párhuzamosítás: utasítás vagy processzor szintű.

Utasítás szintű: szállítószalag, csővezeték (pipelining).

Kezdetben:



Minden fázist külön hardver hajt végre (**2.4. ábra**), ezek párhuzamosan működhetnek (szerelő csarnok).

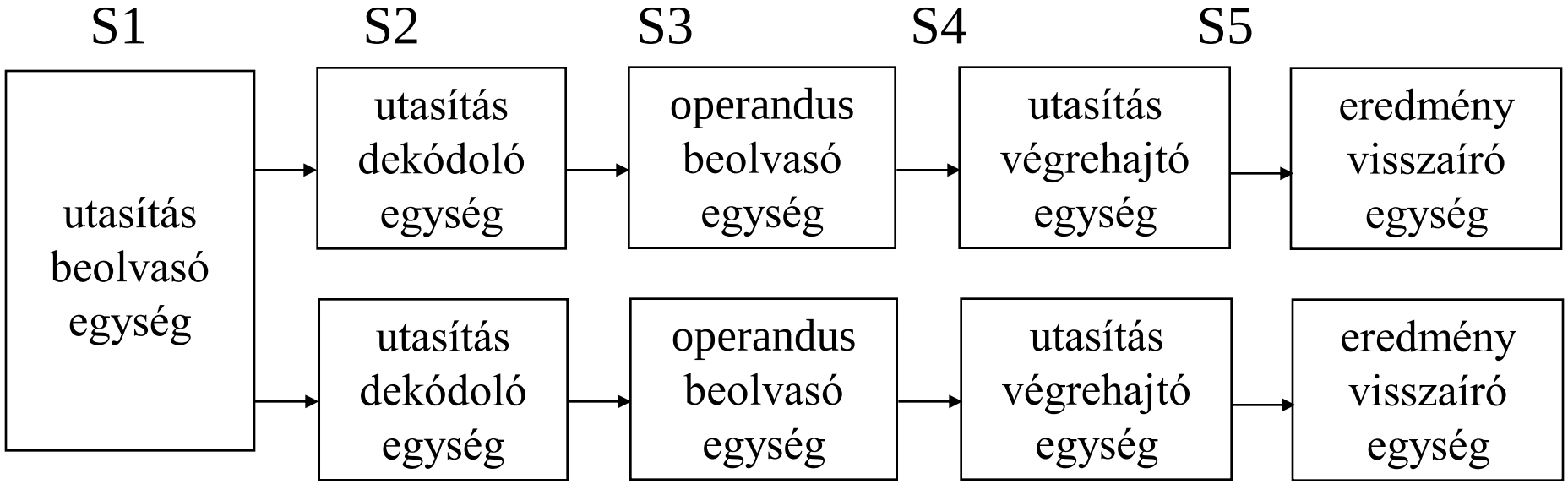


	A végrehajtás alatt lévő utasítás sorszáma									
S1:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
S2:		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
S3:			**1**	**2**	**3**	**4**	**5**	**6**	**7**	**…**
S4:				**1**	**2**	**3**	**4**	**5**	**6**	
S5:					**1**	**2**	**3**	**4**	**5**	
idő	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**…**

2.4. ábra

- **Késleltetés** (latency): mennyi ideig tart egy utasítás.
- **Áteresztőképesség** (processor bandwidth): hány **MIPS** (Million Instruction Per Second) a sebesség.

Több szállítószalagos CPU

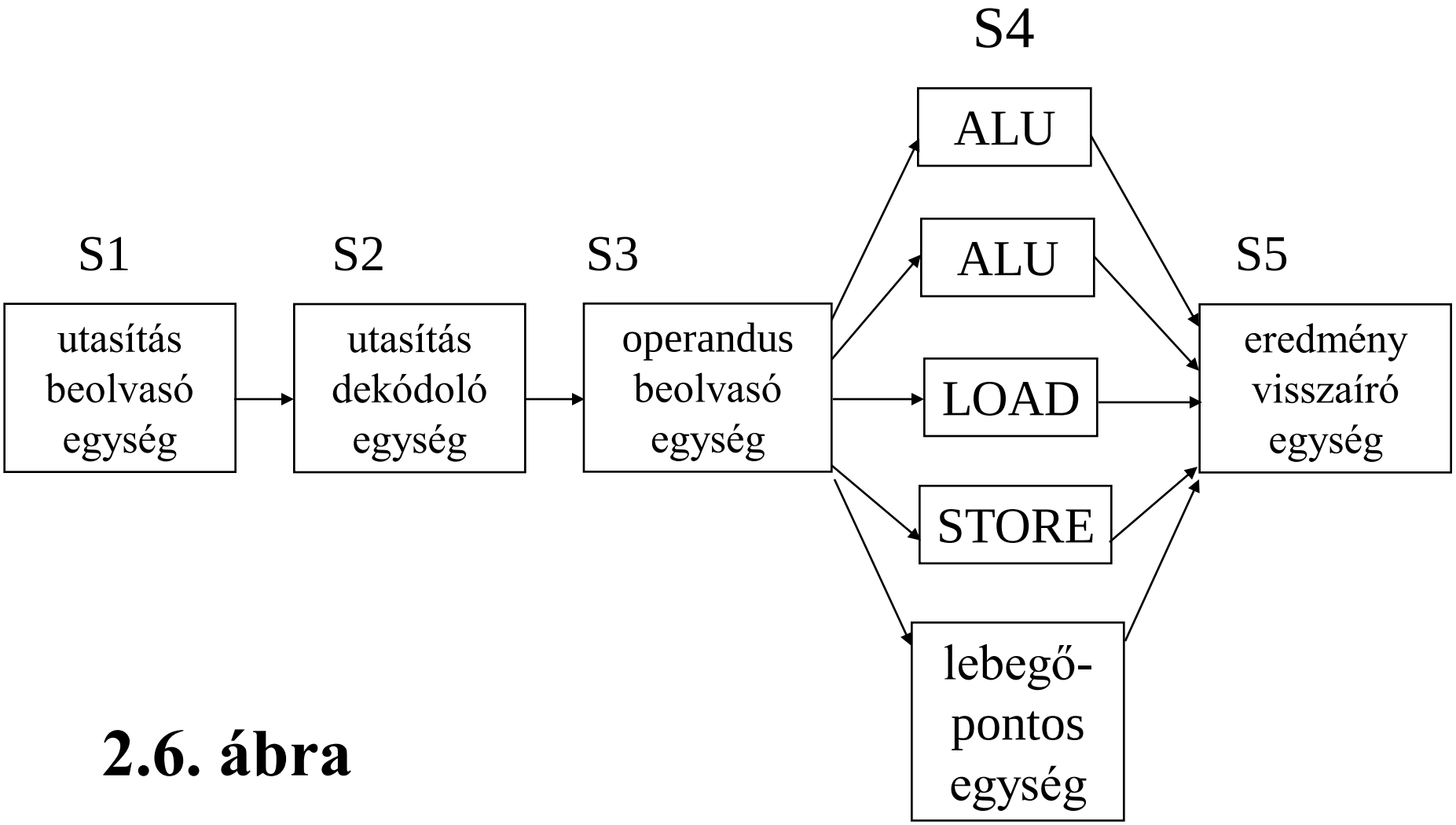


Két szállítószalag (2.5. ábra):

- Két végrehajtó egység, de közös regiszterek,
- A két szállítószalag lehet különböző is (**Pentium**):
  fő – ez többet tud, elsőbbséget élvez – és mellék

Bonyolult szabályok a párhuzamos végrehajthatóságra (fordítók vagy hardver).

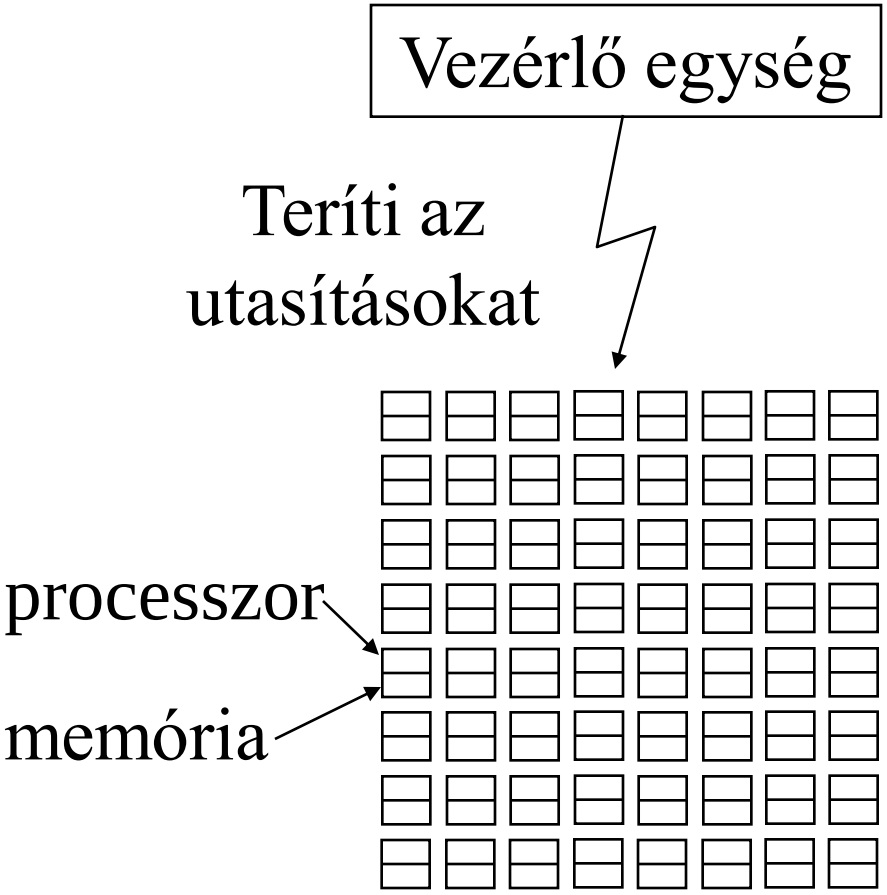
Szuperskaláris processzor 5 funkcionális egységgel:



2.6. ábra

Processzor szintű párhuzamosítás

- **Tömb (array) processzor (2.7. ábra)**



sok azonos processzor (**ILLIAC IV**: (4*)8*8), mindnek saját memóriája. Vezérlő processzor adja ki a feladatot. Mindegyik processzor ugyanazt csinálja, de a saját adatain.

Már nem divatos (drága és nehéz kihasználni).

8*8-as processzor/memória rács

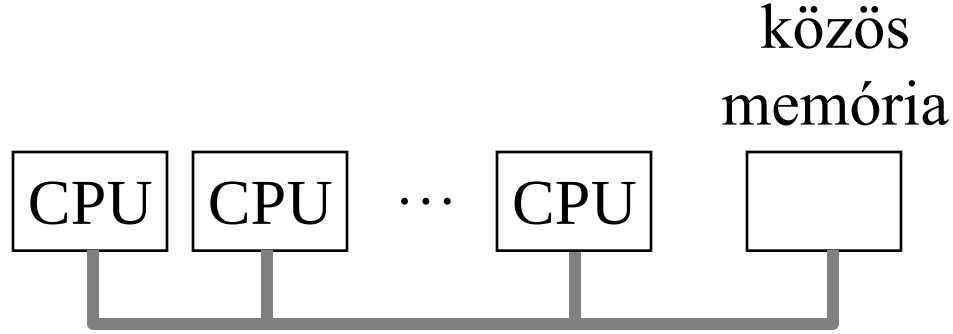
- **Vektor processzorok**

  Vektor regisztereket használnak.

  A vektor regiszter több hagyományos regiszterből áll. Gyors szállítószalag gondoskodik a regiszterek feltöltéséről, szintén gyors szállítószalag továbbítja a regiszterek tartamát az aritmetikai egységbe, pl. a vektor regiszterek összeadásához. Az eredmények szintén vektor regiszterbe kerülnek.

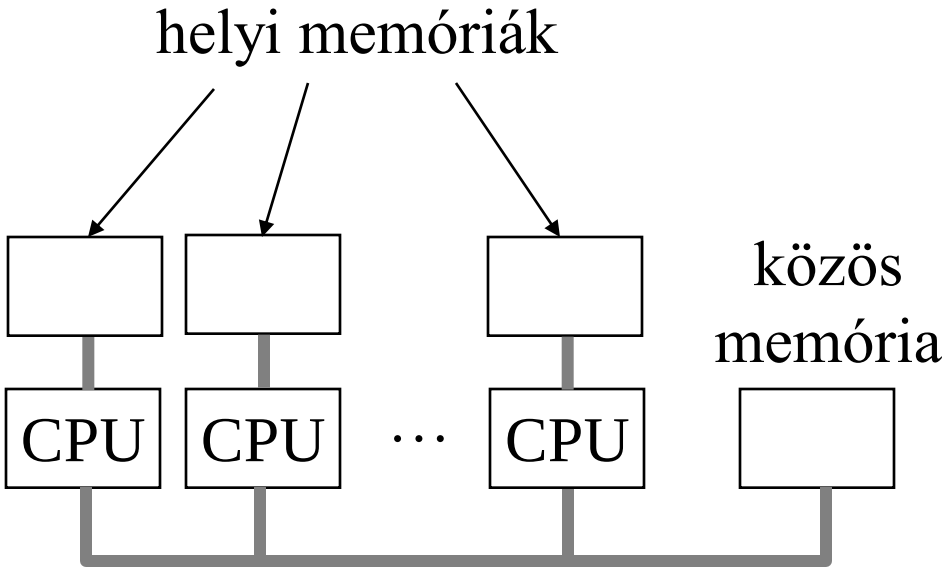
  Jól kombinálhatók hagyományos processzorokkal.

- **Multiprocesszorok**



A közös memória megkönnyíti a feladat megosztását.

- Csak közös memória. Nagyon terheli a memória sínt.



- Lokális memória is van.

Sok (>64) processzoros rendszert nehéz építeni a közös memória miatt.

2.8. ábra

- **Multiszámítógépek:** Nincs közös memória:
  A **CPU**-k üzenetekkel tartják egymással a kapcsolatot.
  Néhány μs üzenet idő.

  2-3 dimenziós hálók, fák, gyűrűk.

  Közel 10 000-es rendszer is van.

A mikroarchitektúra szint tervezése

Mic-1: olcsó, de lassú. **Sebesség növelés:**

- rövidebb óraciklus,
- kevesebb óraciklus az utasítások végrehajtásához,
- az utasítások végrehajtásának átlapolása.

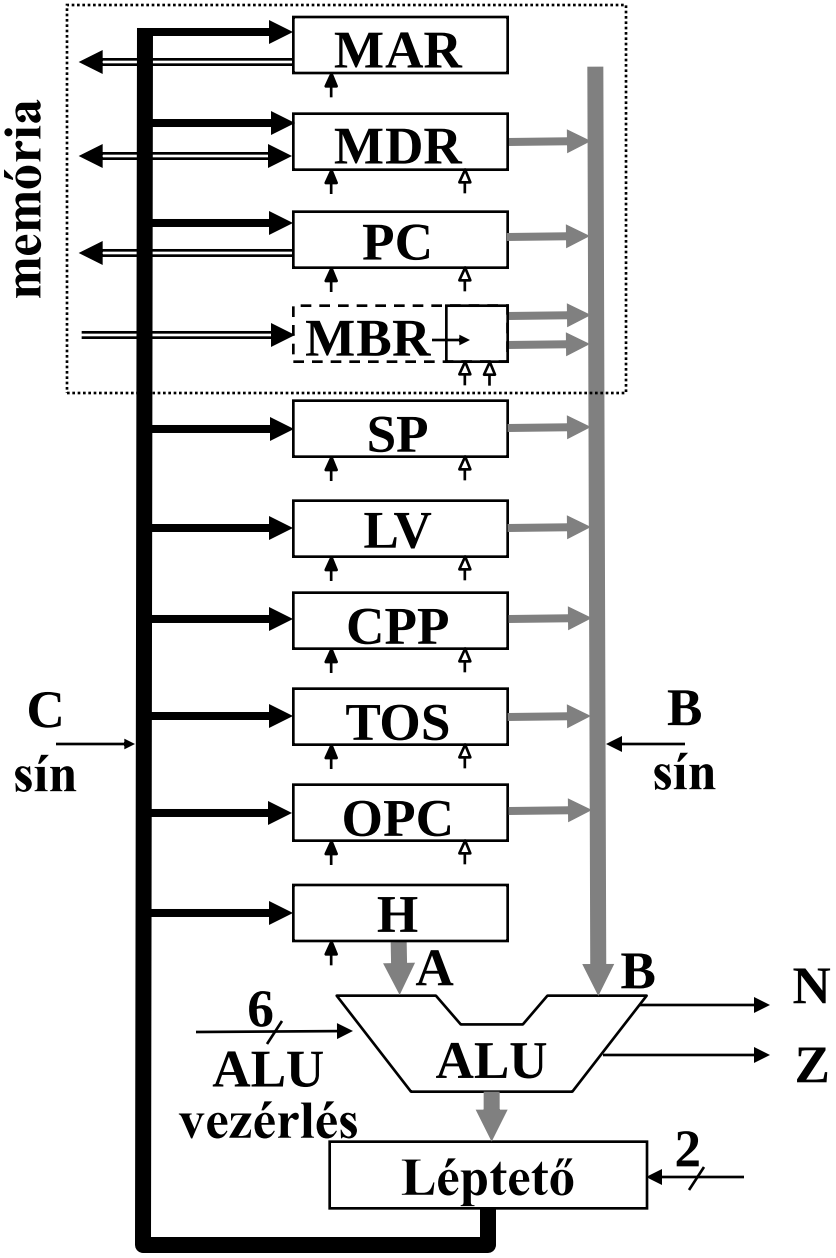
B sín 9 regiszterét 4 bittel címeztük: dekódolóra van szükség, növeli az adatút ciklus idejét! (**4.6. ábra**)

Úthossz (path length, a szükséges ciklusok száma) **rövidítése**: **goto Main1** sokszor megspórolható, jobb microprogram vagy pl. **PC** növelésére külön áramkör (ez legtöbbször ***fetch***-csel együtt történik).

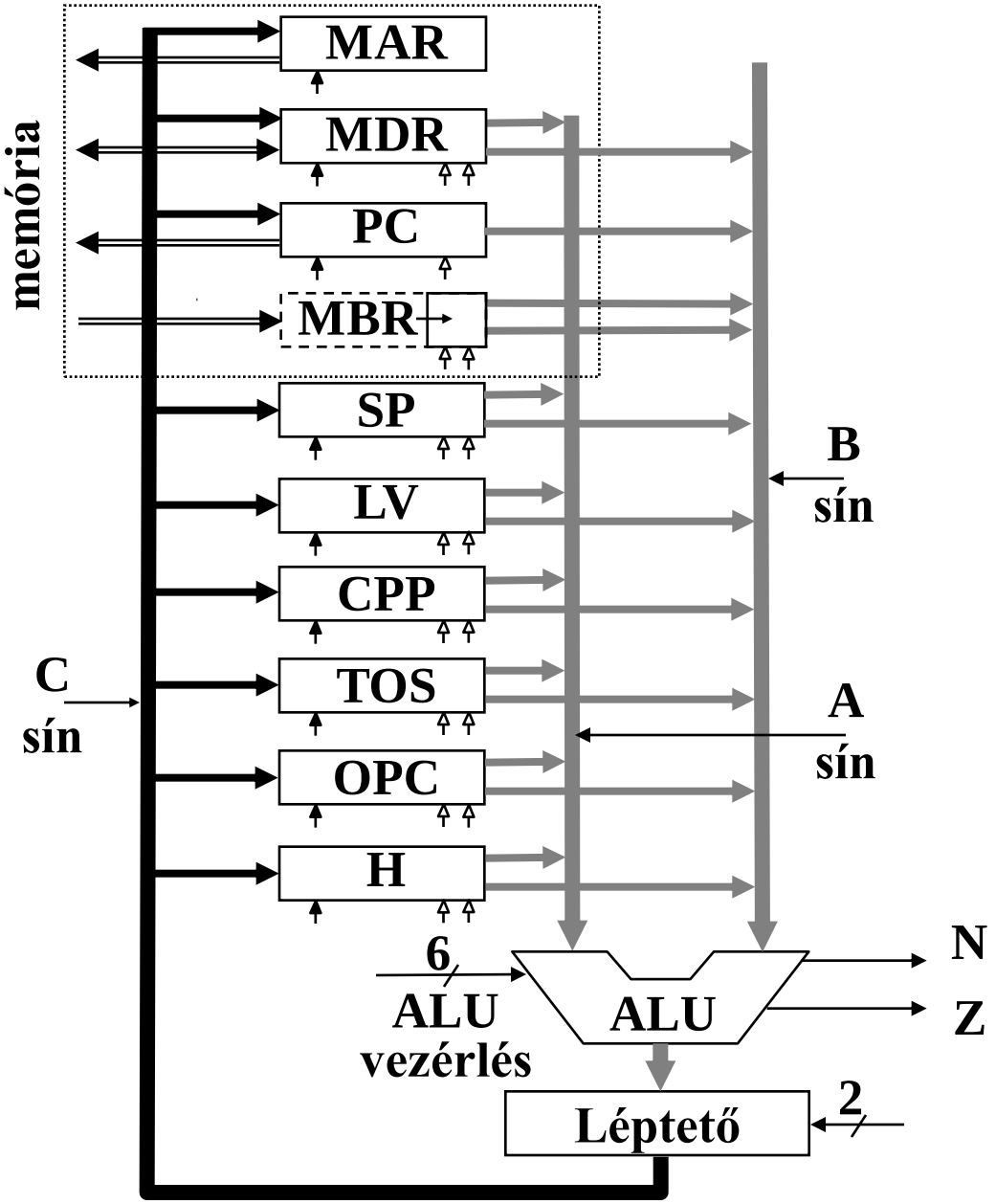
goto Main1 sokszor megspórolható (**4.23-24. ábra**):

0x57 POP A verem legfelső szavát eldobja.

pop1	MAR=SP=SP-1; rd //2. szó címe, olvas
pop2	// vár
pop3	TOS=MDR; goto main1 //TOS=a verem teteje
main1	PC=PC+1;fetch;goto(MBR) //következő ut.
Új változat	
pop1	MAR=SP=SP-1; rd
pop2	PC=PC+1; fetch //következő ut. olvasása
pop3	TOS=MDR; ~~fetch;~~ goto(MBR)



4.1. ábra Mic-1



~4.29. ábra Háromsínes architektúra

Három sínes architektúra

Sok regiszter csatlakozhat az **A** sínhez, nemcsak **H** (**4.1., 4.29. ábra**).

Előnye: a két sínes architektúrával szemben pl. **iload** -ban nem kell **H = LV** (**4.25-26. ábra**).

ILOAD *varnum* // lokális változó a verembe

varnum a lokális változó 8 bites indexe.

A **PC**-vel kapcsolatos teendők:
- **PC** növelése **1**-gyel,
- **fetch**,
- 2 bájtos operandus olvasás a memóriából.

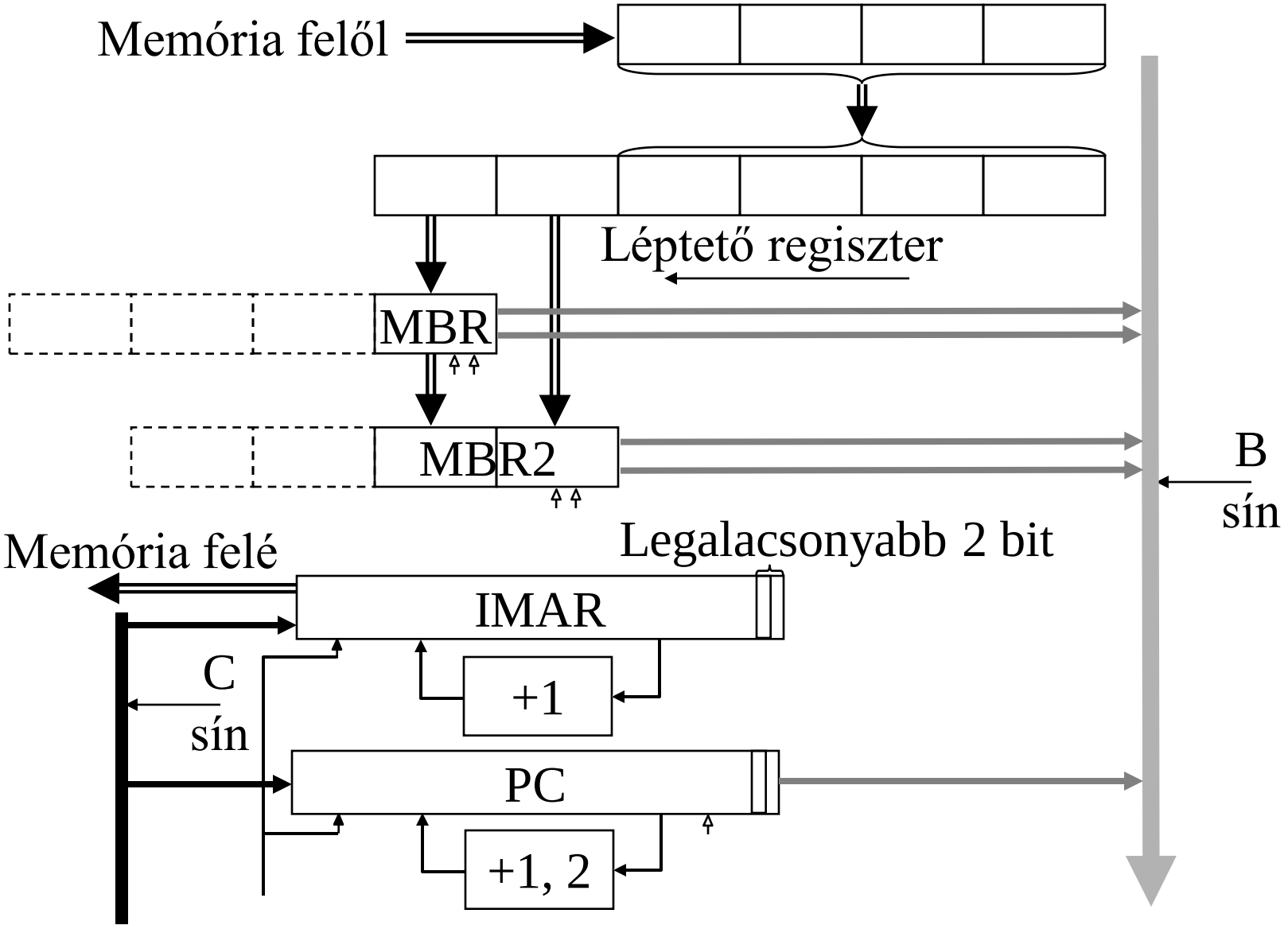
ALU-nál egyszerűbb áramkörrel megvalósíthatók.

Utasításbetöltő egység (IFU – Instruction Fetch Unit)

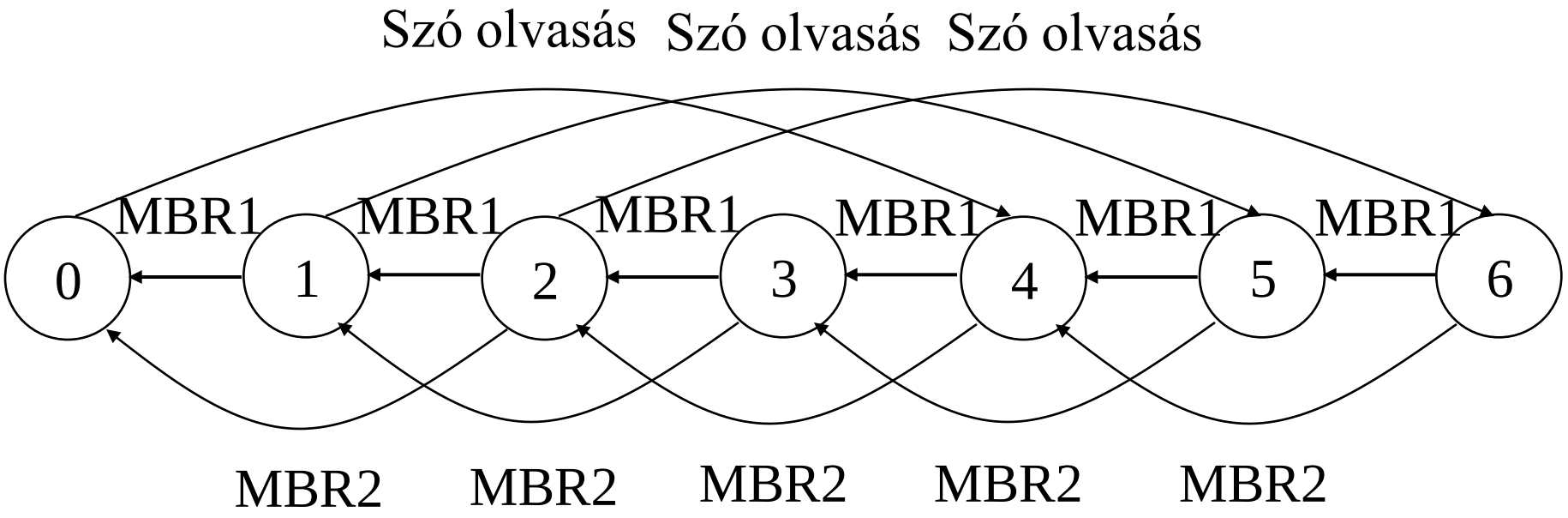
- értelmezhet minden kódot, hogy kell-e operandus,
- de egyszerűbb, ha a kódtól függetlenül előkészíti a következő 8 és 16 bites részt (**4.27. ábra**).

Utasításbetöltő egység (IFU – Instruction Fetch Unit)

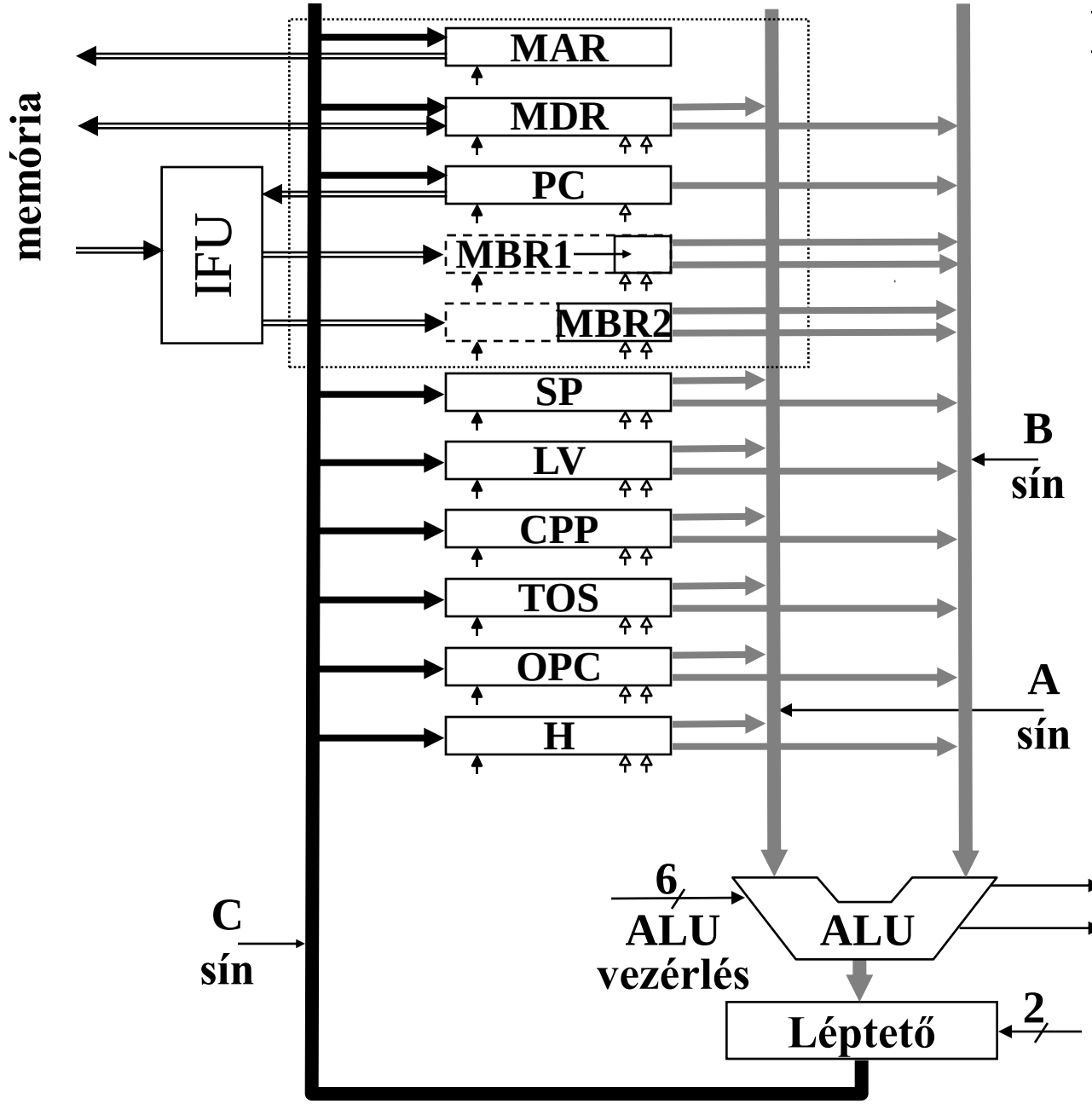
4.27. ábra



Véges állapotú gép (FSM – Finite State Machine, **4.28. ábra**): **0**, **…**, **6**: állapotok, élek: események.



Mic-2 (4.29. ábra)



- **Main1** fölösleges,
- **PC** növeléséhez nem kell az **ALU**,
- a 8 és 16 bites operandusokat **IFU** adja.

Mic-2 (4.29. ábra)

Több hardver kell az **A** sín címzése és **IFU** miatt, de kevesebb mikroutasítás kell, pl. **WIDE ILOAD**-hoz az eddigi **9** helyett csak **4** (v.ö. **4.17. ábra**). **WIDE ILOAD *varnum*** //beteszi a 16 bites ***varnum*** indexű lokális változót a verembe:

wide1	goto (MBR1 OR 0x100)
w_iload1	MAR=LV+MBR2U; rd; goto iload2
iload1	MAR=LV+MBR1U; rd // változó olvasása
iload2	MAR=SP=SP+1 // vermelés előkészítése
iload3	TOS=MDR; wr; goto (MBR1)

Mic-2 adatútja és **IFU** kapcsolata:

Ha **PC** értéket kap a **C** sínről, azt **IMAR** is megkapja. Ilyenkor a mikroprogramnak várnia kell a léptető regiszter, **MBR1** és **MBR2** feltöltésére.

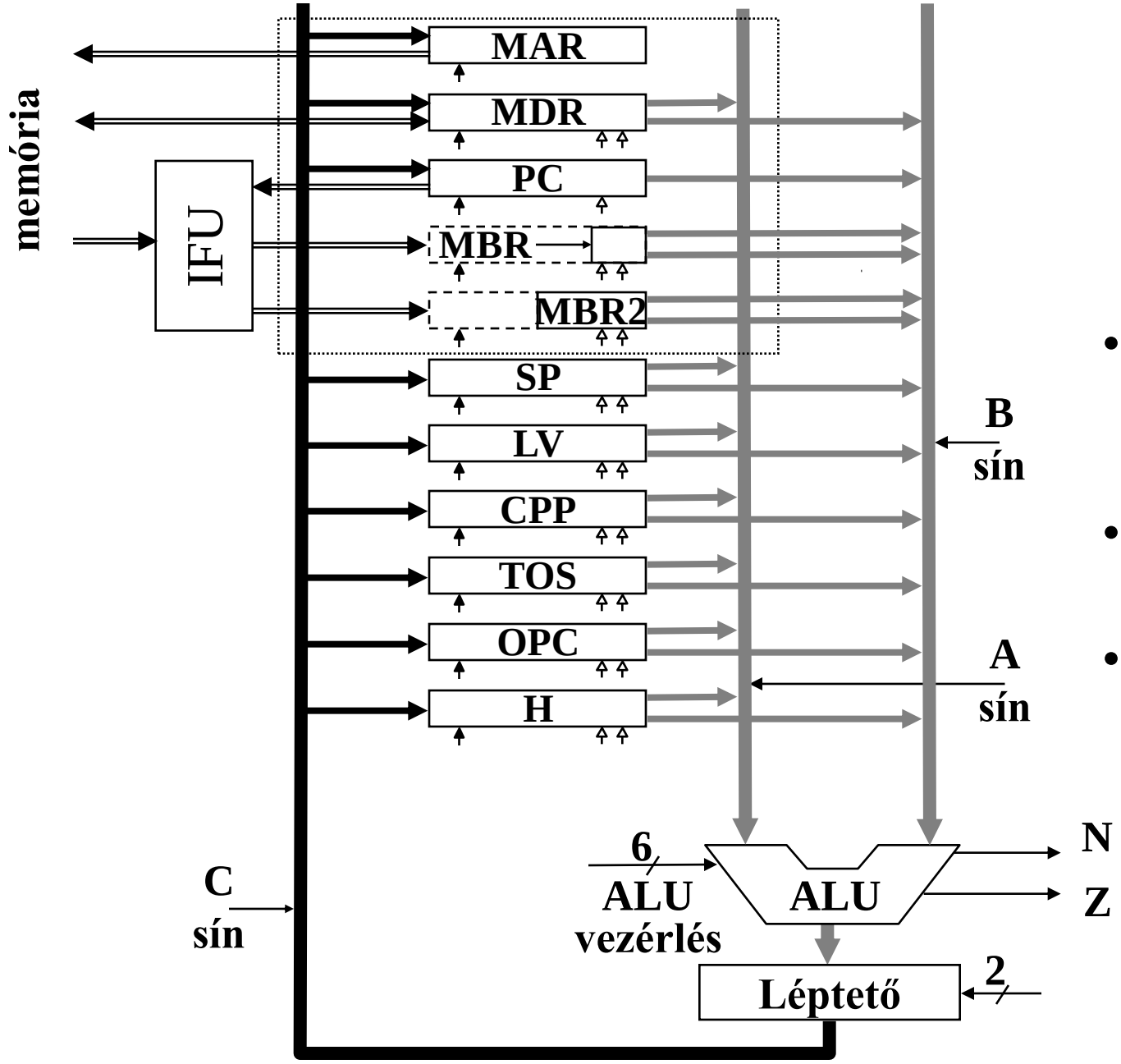
IMAR módosul, amint a léptető regiszterbe írta a következő 4 bájtot, de **PC** csak akkor, ha **MBR1**vagy **MBR2** olvasása történik.

goto1	H=PC-1 // IFU már csinált PC=PC+1-et
goto2	PC=H+MBR2 // itt folytatódik a program
goto3	// IFU még nincs kész, várni kell!
goto4	goto (MBR1) // a folytatás 1. utasítása

Az **IFLT *offset*** utasítás (**Mic-2**)
Kivesz egy szót a veremből és ugrik, ha negatív.

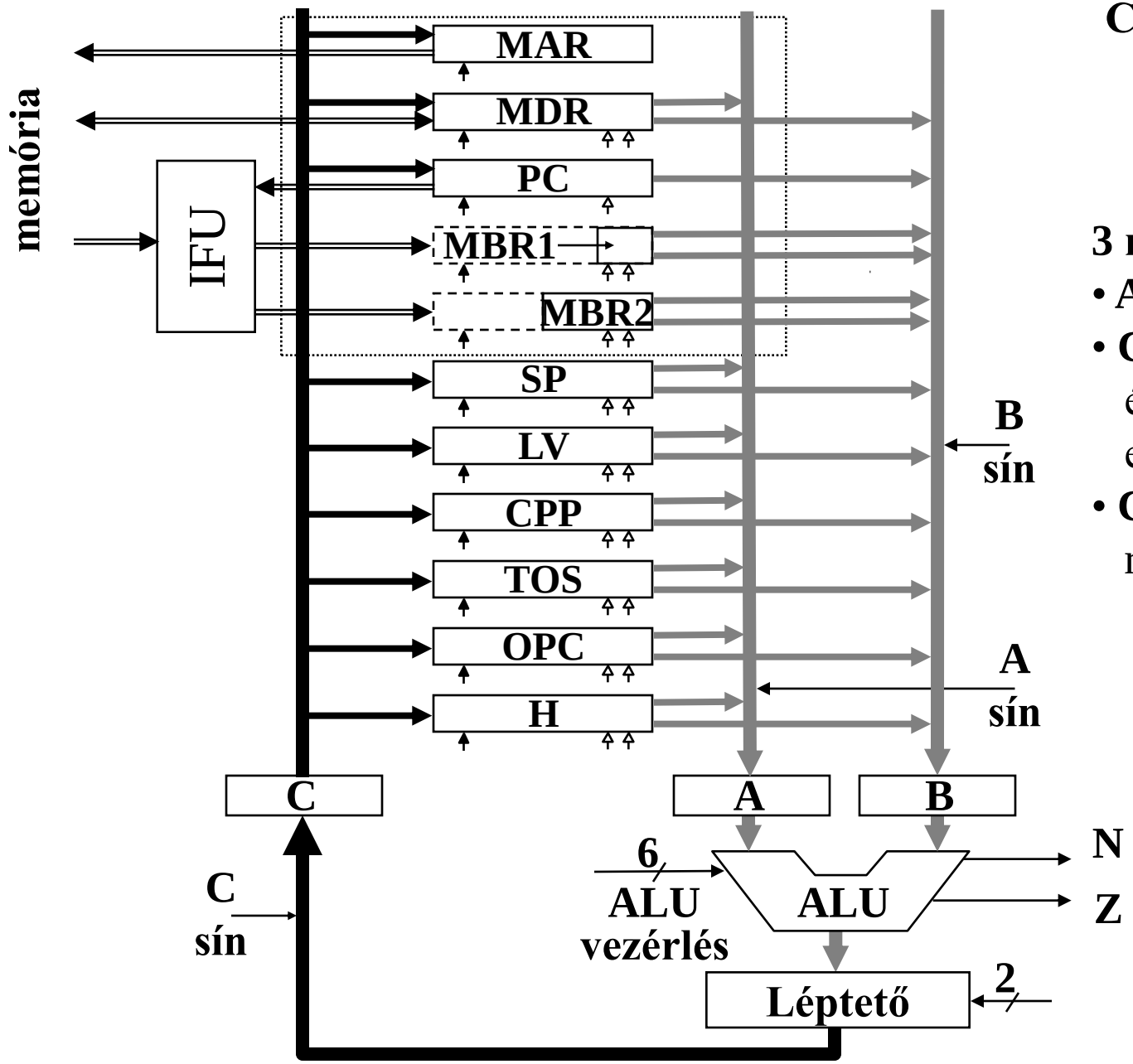
iflt1	MAR=SP=SP-1; rd // 2. szó a veremből
iflt2	OPC=TOS // TOS mentése
iflt3	TOS=MDR // TOS= a verem új teteje
iflt4	N=OPC; if(N) goto T; else goto F //elágazás
T	H=PC-1; goto goto2 // igaz ág
F	H=MBR2 // hamis ág, eldobja ***offset***–et
F2	goto (MBR1) // a folytatás 1. utasítása

A **Mic-2** adatút idejének összetevői (**4.29. ábra**):



- az **A** és **B** sínek feltöltése a regiszterekből,
- az **ALU** és a léptető munkája,
- az eredmények tárolása a **C** sínről.

Csővonalas terv: Mic-3 (4.31. ábra)



A, **B** és **C** tároló.

3 mikrolépés:

- **A**, **B** feltöltése,
- **C** feltöltése az **ALU** és a léptető eredménye alapján,
- **C** tárolása regiszterbe.

A 3 mikrolépés egyidejűleg (párhuzamosan) végrehajtható!

Pl.: a verem két felső szavának cseréje **Mic-3**-on (**4.33. ábra**):

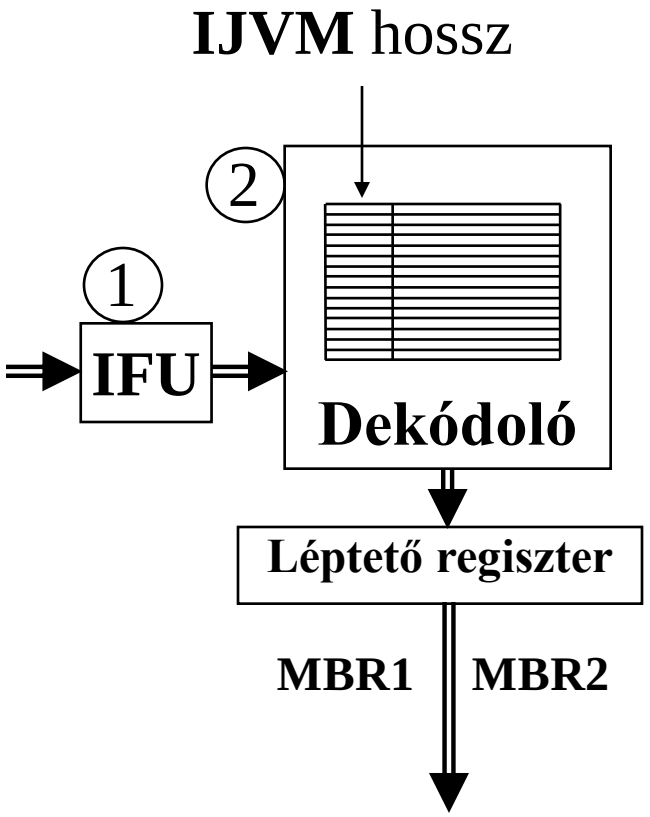
	swap1	swap2	swap3	swap4	swap5	swap6
cy	MAR= SP-1;rd	MAR= SP	H=MDR; wr	MDR= TOS	MAR= SP-1;wr	TOS=H; goto(MBR1)
1	B=SP					
2	C=B-1	B=SP				
3	MAR=C;rd	C=B	*Várni kell!*			
4	MDR=mem	MAR=C	*Várni kell!*			
5			B=MDR			
6			C=B	B=TOS		
7			H=C;wr	C=B	B=SP	
8			mem=MDR	MDR=C	C=B-1	B=H
9					MAR=C;wr	C=B
10					mem=MDR	TOS=C
11						goto(MBR1)

Valódi függőség RAW – Read After Write! Elakadás

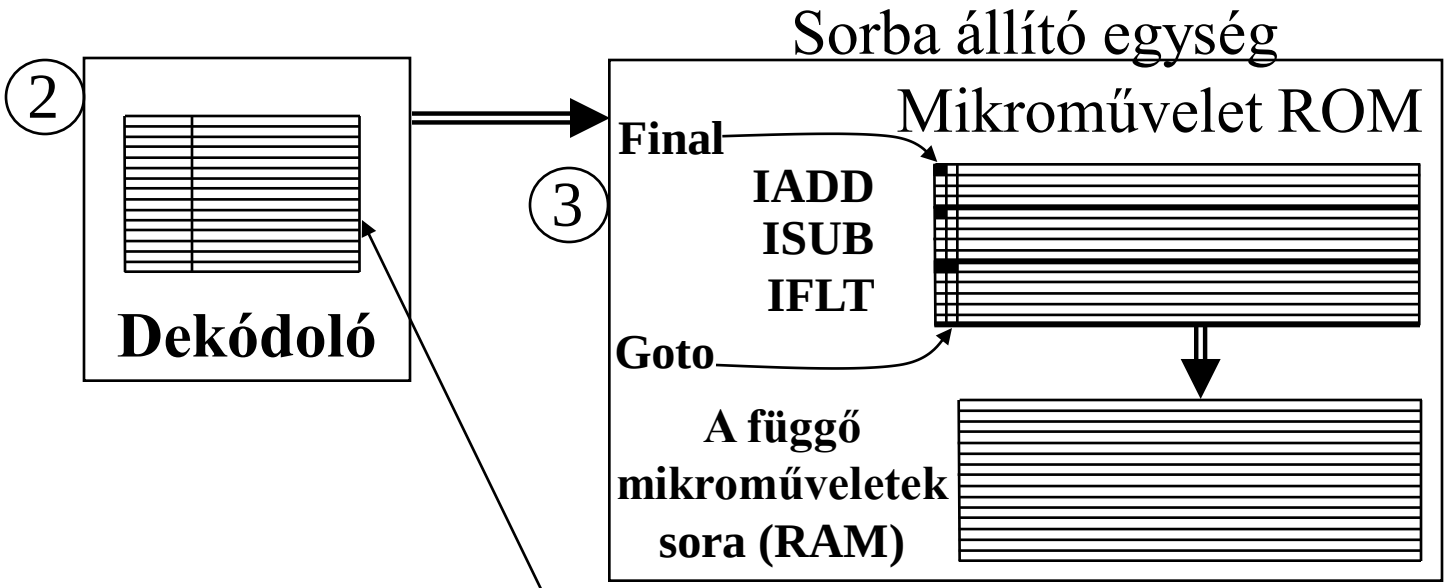
eldugaszolja a csővezetéket!

Hétszakaszú csővezeték: Mic-4 (4.35. ábra)

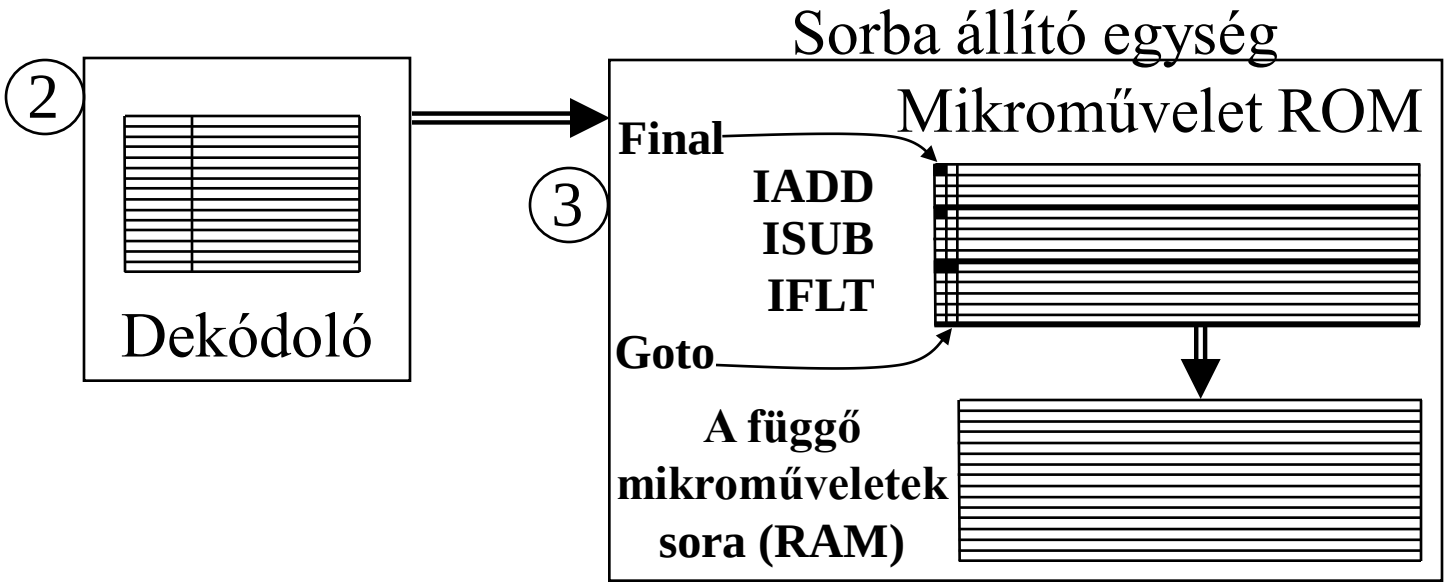
1. Az **IFU** a bejövő bájtfolyamot a dekódolóba küldi.



2. A dekódolóban van egy táblázat, amely minden utasításnak tudja a hosszát. A **WIDE** prefixumot felismeri, pl. **WIDE ILOAD** –ot átalakítja **WIDE_ILOAD** –dá: pl. 9 bites utasítás kód.
El tudja különíteni az utasítás kódokat és az operandusokat. Az operandusokat a léptető regiszterbe teszi, onnan tölti fel **MBR1**-et és **MBR2**-t.



A dekódoló egy másik táblázata megmutatja, hogy a sorba állító egységben lévő **ROM** melyik címén kezdődnek a kódhoz tartozó mikroműveletek.
Nincs **NEXT_ADDRESS** és **JAM** mező. Nincs feltétlen ugrást végző mikroművelet. Az egyes **IJVM** utasításokat megvalósító mikroműveletek egymás után vannak a **ROM**-ban, az utolsónál a **Final** be van állítva.



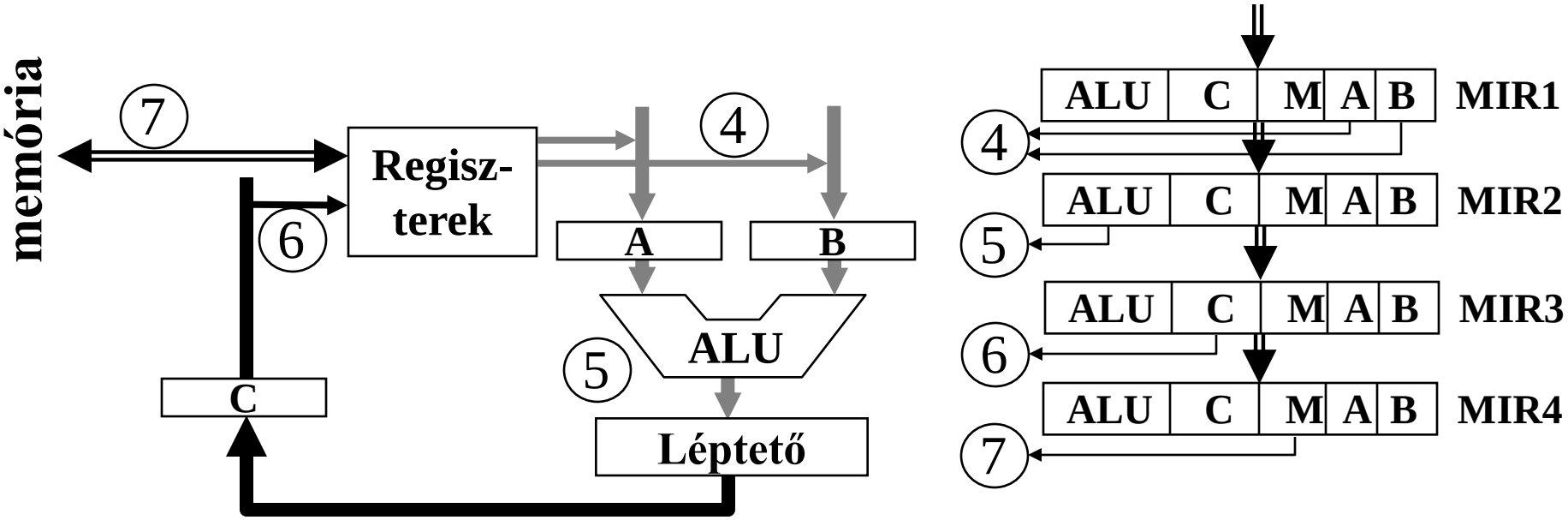
3. A sorba állító egység a **ROM**-ból a **RAM**-ba másolja a mikroműveleteket, amint van hely a **RAM**-ban. A kódhoz tartozó utolsó mikroművelet **Final** bitje jelzi, hogy nincs több átmásolandó mikroművelet.
   Ha a mikroműveletek között nem volt olyan, amelyik **Goto** bitje be volt állítva, akkor nyugtázó jelet küld a dekódolónak, hogy folytathatja a munkáját.

Néhány **IJVM** utasítás (pl. **IFLT**) elágazást kíván. A feltételes mikroutasítások speciális utasítások, ezeket külön mikroműveletként kell megadni. Tartalmazzák a **JAM** biteket és a **Goto** bitet. A **Goto** bit arra szolgál, hogy a sorba állító egység le tudja állítani további utasítások dekódolását. Mindaddig nem lehet tudni, hogy melyik utasítás következik a feltételes utasítás után, amíg a feltétel ki nem értékelődött.

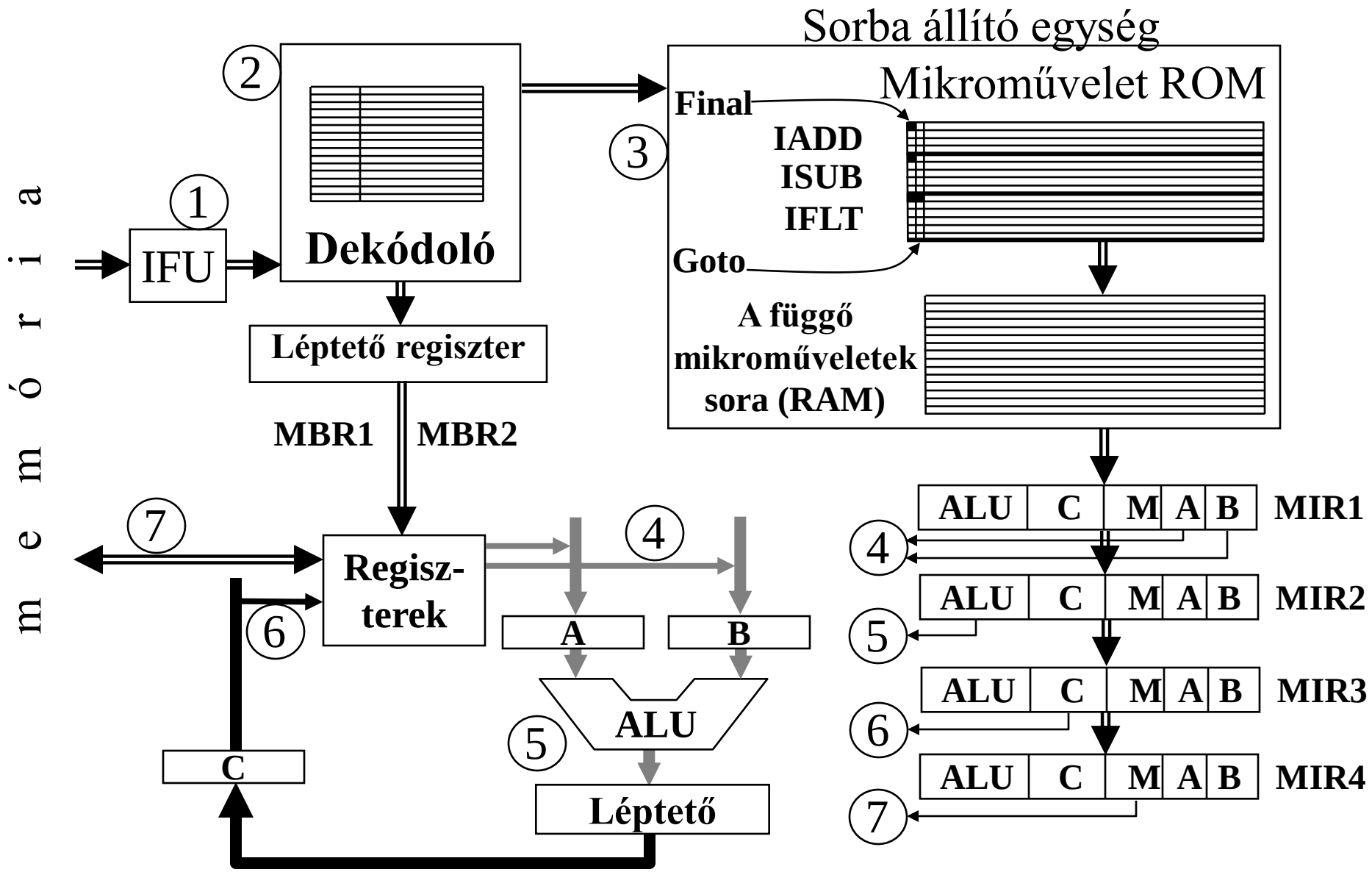
- Ha létrejön az elágazás, akkor a csővezeték nem folytatódhat. „Tiszta lapot” kell csinálni **IFU**-ban, **dekódolóban** és a sorba állító egységben, majd az ***offset***-nek megfelelő címtől folytatódik a betöltés.
- Ha az ugrás feltétele nem teljesül, akkor a **dekódoló** megkapja a nyugtázó jelet, és a következő utasítással folytatódhat a dekódolás.

Az adatutat 4 független **MIR** vezérli. Minden óraciklus kezdetekor **MIRi** föltöltődik a fölötte lévőből, **MIR1** pedig a **RAM**-ból.

4. **MIR1** az **A, B** regiszterek feltöltését,
5. **MIR2** az **ALU** és a léptető működését,
6. **MIR3** az eredmény tárolását,
7. **MIR4** pedig a memória műveleteket vezérli.



Hétszakaszú csővezeték: Mic-4 (4.35. ábra)



IFLT *offset* programozása **Mic-4**-en:

	iflt1	iflt2	iflt3	iflt4 (Final=1, Goto=1)	
cy	**MAR=SP= SP-1; rd**	**OPC= TOS**	**TOS=MDR**	**N=OPC; if(N) *GOTO offset***	
1	**B=SP**				
2	**C=B-1**	**B=TOS**			
3	**MAR=SP=C; rd**	**C=B**	***Várni kell!***		
4	**MDR=mem**	**OPC=C**	***Várni kell!***		
5			**B=MDR**		
6			**C=B**	**B=OPC**	
7			**TOS=C**	**C=B**	
8				**N PC=PC-1+MBR2; „tiszta lap”, majd a PC által mutatott címtől utasítás betöltés, …**	**#N MBR2 –t eldobni, folytatódhat a dekódolás**

A 8. ciklus feladata túl bonyolult! **MBR2-1** előre kiszámítható.

IFLT *offset* programozása **Mic-4**-en:

	iflt1	iflt2	iflt3	iflt4	iflt5 (Final=1, Goto=1)	
cy	**MAR=SP= SP-1; rd**	**OPC= TOS**	**H=MBR2-1**	**TOS=MDR**	**N=OPC; if(N) *GOTO offset***	
1	**B=SP**					
2	**C=B-1**	**B=TOS**				
3	**MAR=SP=C; rd**	**C=B**	**B=MBR2**			
4	**MDR=mem**	**OPC=C**	**C=B-1**	***Várni kell!***		
5			**H=C**	**B=MDR**		
6				**C=B**	**B=OPC**	
7				**TOS=C**	**C=B**	
8					**N PC=PC+H; „tiszta lap”, majd a PC által mutatott címtől utasítás betöltés, …**	**#N folytatódhat a dekódolás**

Az **IJVM** feltétlen ugrását a dekódoló is feldolgozhatja.

Elágazás jövendölés (4.40. ábra)

Legkorábban a dekódoló veheti észre, hogy ugró utasítást kell végrehajtani, de addigra a következő utasítás már a csővezetékben van!

Program	Címke	Gépi utasítás	Megjegyzés
`if(i==0)`		`CMP i,0`	összehasonlítás
		`BNE else`	feltételes ugrás
`k=1;`	`then:`	`MOV k,1`	`k=1`
`else`		`BR next`	feltétlen ugrás
`k=2;`	`else:`	`MOV k,2`	`k=2`
	`next:`		

A **`BR next`** utasítással is probléma van!

Elágazás jövendölés (4.40. ábra)

Eltolás rés (delay slot): Az ugró utasítás utáni pozíció. Az ugró utasítás végrehajtásakor ez az utasítás már a csővezetékben van!

Megoldási lehetőségek:

- **Pentium 4**: bonyolult hardver gondoskodik a csővezeték helyreállításáról
- **UltraSPARC III**: az eltolás résben lévő utasítás végrehajtásra kerül(!). A felhasználóra (fordítóra) bízza a probléma megoldását, a legrosszabb esetben **`NOP`** utasítást kell tenni az ugró utasítás után.

Feltételes elágazás

Sok gép megjövendöli, hogy egy ugrást végre kell hajtani vagy sem.

Egy triviális jóslás:

- a visszafelé irányulót végre kell hajtani (ilyen van a ciklusok végén),
- az előre irányulót nem (jobb, mint a semmi).

Feltételes elágazás esetén a gép tovább futhat a jövendölt ágon,

- amíg nem ír regiszterbe,
- csak „firkáló” regiszterekbe írhat.

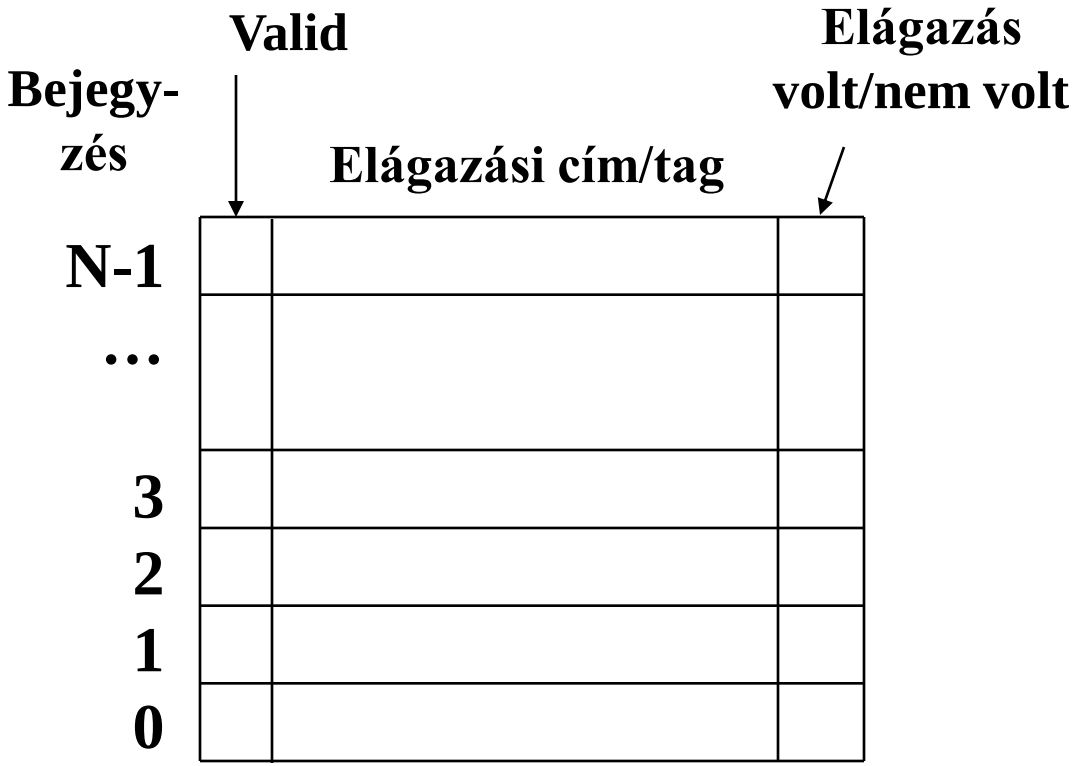
Ha a jóslat bejött, akkor minden rendben, ha nem, akkor sincs baj.

Több feltételes elágazás egymás után!

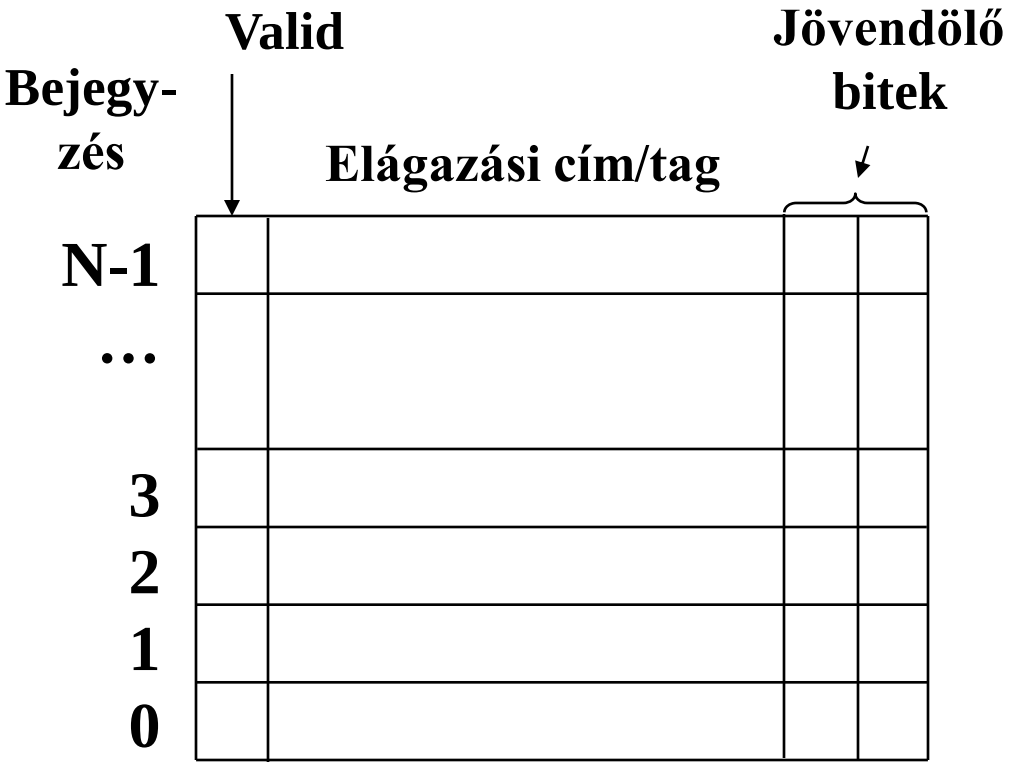
Dinamikus elágazás jövendölés

Elágazás előzmények tábla (**4.41. ábra**), hasonló jellegű, mint a gyorsító tár. Lehet több utas is!

- Egy jövendölő bit: mi volt legutóbb,

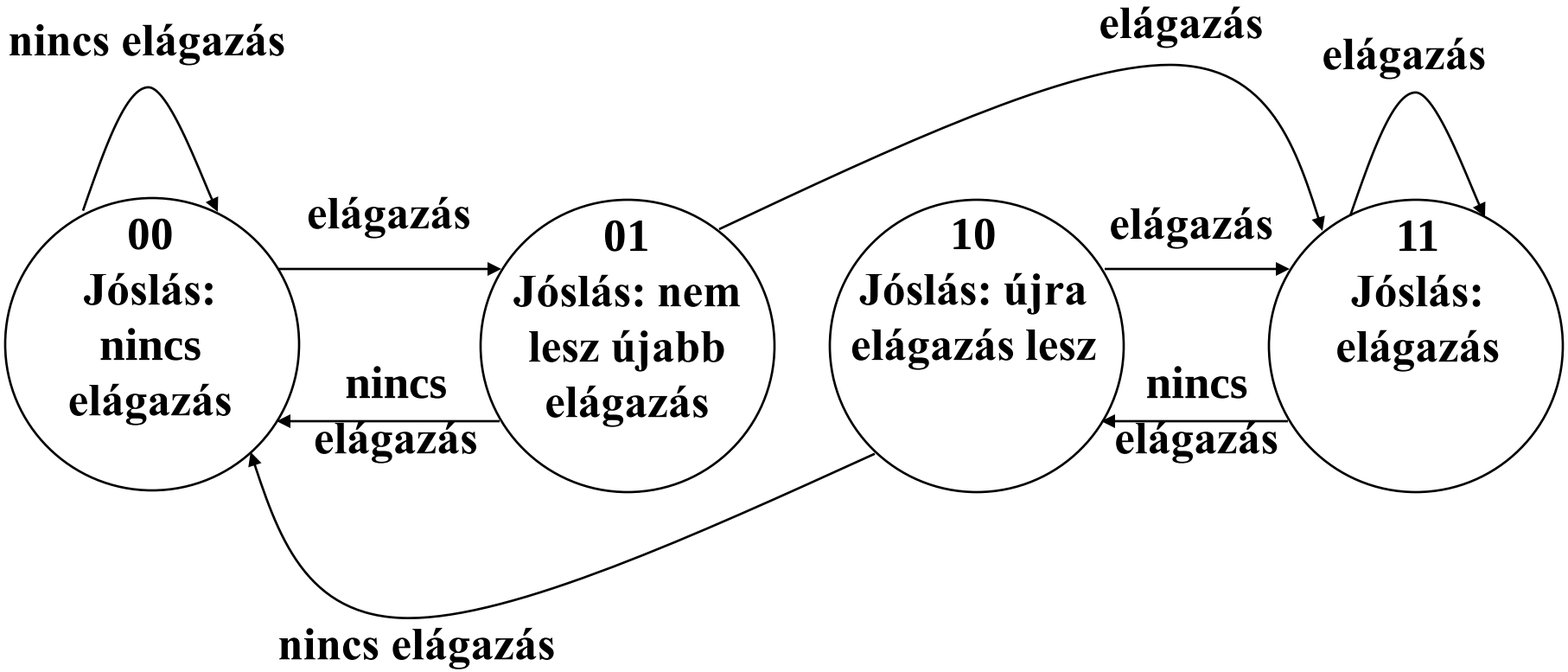


- Két jövendölő bit: mi várható és mi volt legutóbb.



Ha egy belső ciklus újra indul, akkor az várható, hogy a ciklus végén vissza kell ugrani, pedig legutóbb nem kellett.

A várható bitet csak akkor írja át, ha egymás után kétszer téves volt a jóslat (**4.42. ábra**).



- A táblázat a legutóbbi célcímet is tartalmazhatja.

Ha az a jövendölés, hogy lesz elágazás, akkor arra számít, hogy a legutóbb tárolt célcímre kell ugrani (ezt persze ellenőrizni kell).

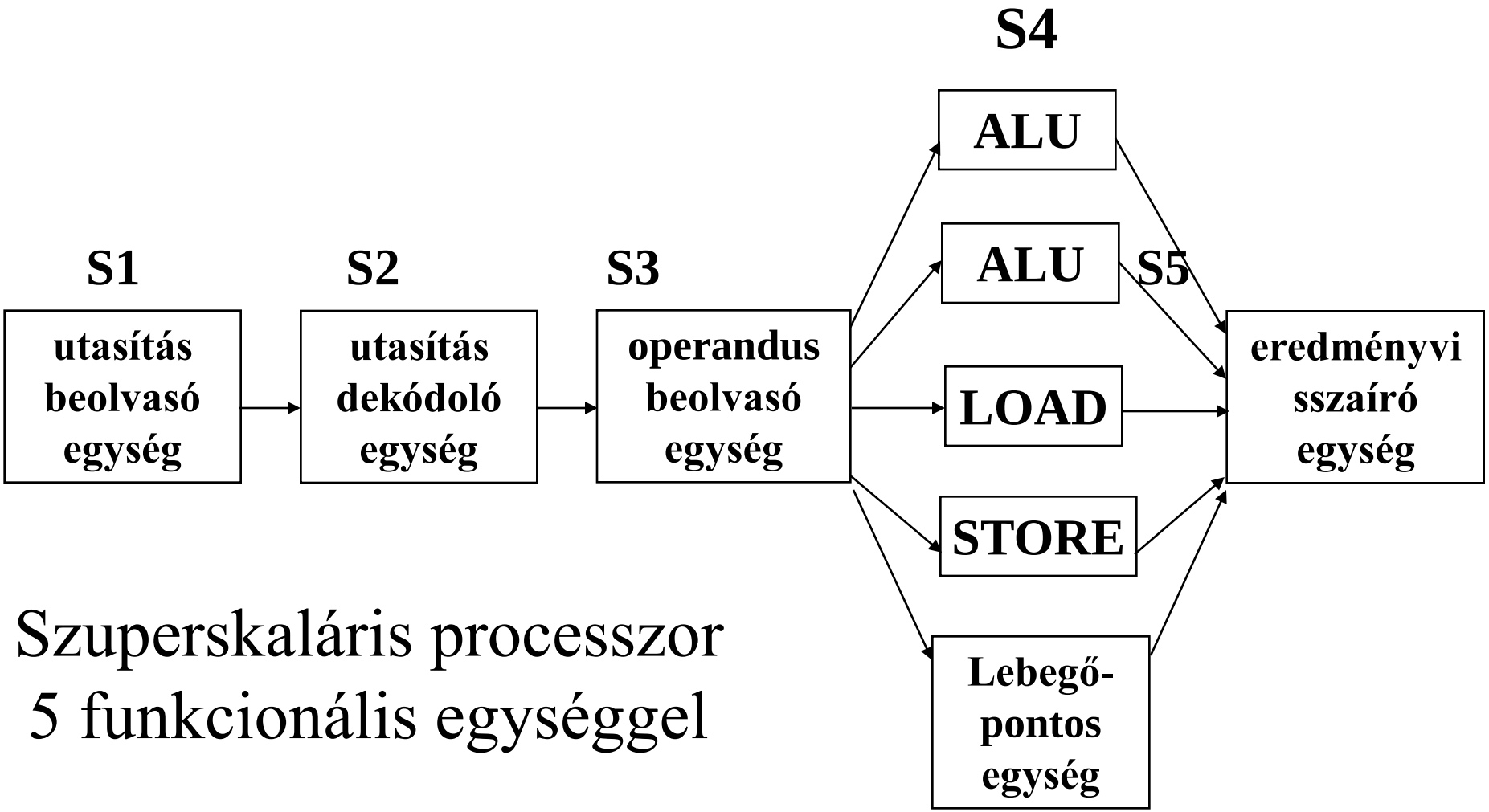
- Figyeljük, hogy az utolsó *k* feltételes elágazást végre kellett-e hajtani. Ez egy *k* bites számot eredményez, ezt az elágazási előzmények blokkos regiszterében tároljuk. Ha a *k* bites szám megegyezik a táblázat valamely bejegyzésének a kulcsával (találat), akkor az ott talált jövendölést használja.

Statikus elágazás jövendölés

A feltételes utasításoknak néha olyan változata is van (pl. **UltraSPARC III**), mely tartalmaz bitet a jóslásra. A fordító ezt a bitet valahogy beállítja.

Olyankor is statikus elágazás jövendölés történik, ha a processzor arra számít, hogy a visszafelé ugrások bekövetkeznek, az előre ugrások nem.

Szuperskaláris architektúrák (2. 6. ábra)



Szuperskaláris processzor
5 funkcionális egységgel

Szuperskaláris architektúra esetén a dekódoló egység az utasításokat mikroutasításokra darabolhatja. Legegyszerűbb, ha a mikroutasítások végrehajtási sorrendje megegyezik a betöltés sorrendjével, de ez nem mindig optimális.

Függőségek

Ha egy utasítás írni/olvasni akar egy regisztert, akkor meg kell várja azon korábbi utasítások befejezését, amelyek ezt a regisztert írni/olvasni akarták!

Függőségek

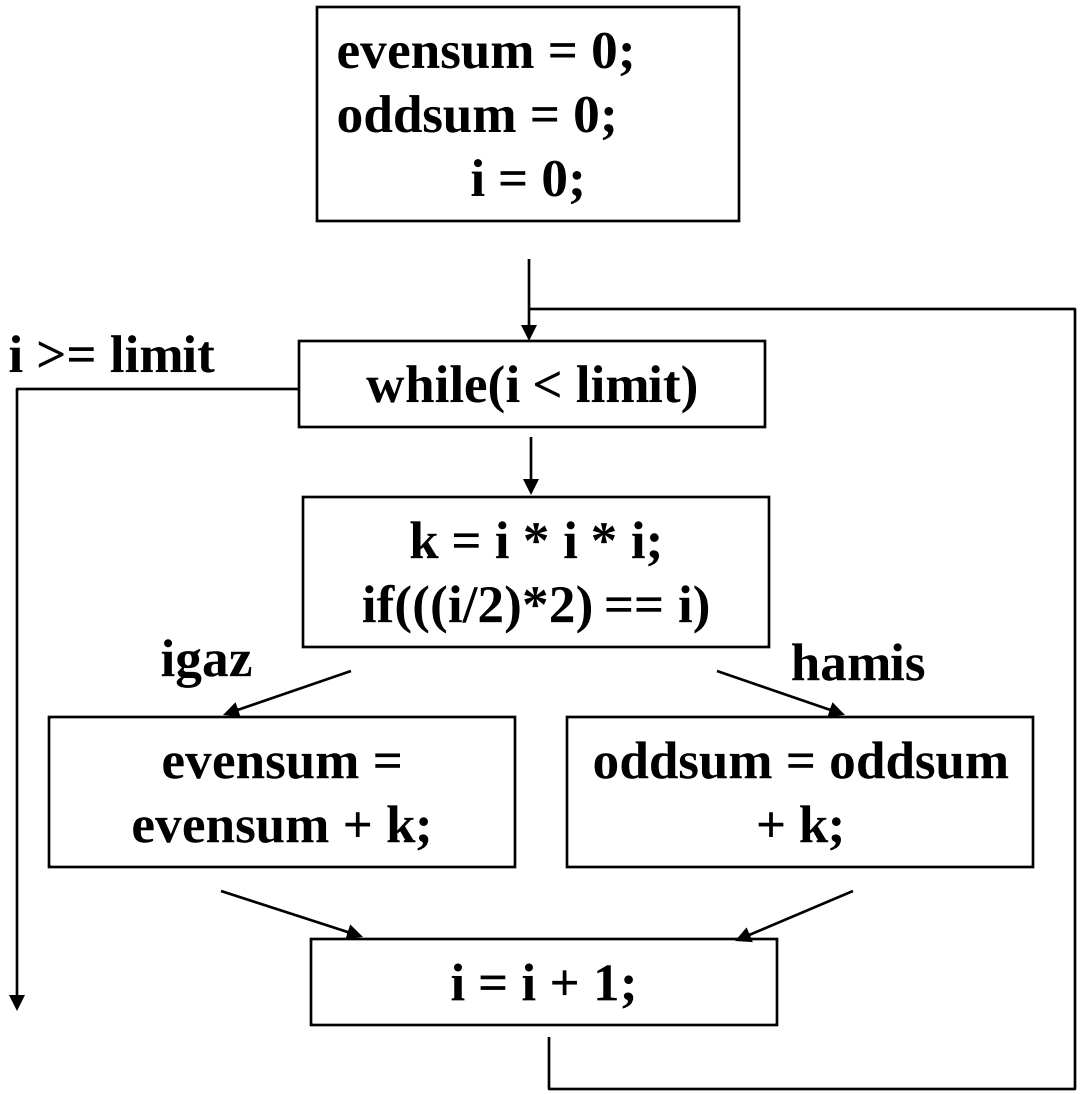
Egy utasítás nem hajtható végre az alábbi esetekben:

- **RAW** (valódi) függőség (Read After Write):
  Onnan akarunk olvasni (operandus),
  ahova még nem fejeződött be egy korábbi írás.
- **WAR** függőség (Write After Read):
  Olyan regiszterbe szeretnénk írni az eredményt,
  ahonnan még nem fejeződött be egy korábbi olvasás.
- **WAW** függőség (Write After Write):
  Olyan regiszterbe szeretnénk írni az eredményt,
  ahova még nem fejeződött be egy korábbi írás.
  Ne boruljon föl az írások sorrendje!

A modern **CPU**-k gyakran titkos regiszterek tucatjait használják regiszter átnevezésre, sokszor kiküszöbölhető vele a **WAR** és **WAW** függőség.

Feltételezett végrehajtás (4.45. ábra)

```
evensum = 0;
oddsum = 0;
i = 0;
while(i < limit) {
  k = i * i * i;
  if(((i/2)*2) == i)
    evensum = evensum + k;
  else
    oddsum = oddsum + k;
  i = i + 1;
}
```



Feltételezett végrehajtás (4.45. ábra)
Speculative Execution

Alap blokk (**basic block**): lineáris kód sorozat. Sokszor rövid, nincs elegendő párhuzamosság, hogy hatékonyan kihasználjuk.

Emelés: egy utasítás előre hozatala egy elágazáson keresztül (lassú műveletek esetén nyerhetünk vele). Pl. evensum és oddsum regiszterbe tölthető az elágazás előtt. Az egyik **LOAD** – természetesen – fölösleges.

Ha valamit nem biztos, hogy meg kell csinálni, de nincs más dolga a gépnek, akkor megteheti, de csak „firkáló” regiszterekbe írhat. Ha később kiderül, hogy kell, akkor átírja az eredményeket a valódi regiszterekbe, ha nem kell, elfelejti.

Feltételezett végrehajtás (Speculative Execution)

Mellékhatások:

- fölösleges gyorsító sor csere, **`SPECULATIVE_LOAD`**
- csapda (pl. ***`x=0`*** esetén ***`if(x>0) z=y/x;`***), **mérgezés bit**.

Pentium 4 (2000. november)

Felülről kompatibilis az **I8088**, **…**, **Pentium III**-mal.

29.000, …, 42 ---> 55 M tranzisztor, 1,5 ---> 3,2 GHz, 63-82W, 478 láb (**3. 44. ábra**), 32 bites gép, 64 bites adat sín.

NetBurst architektúra. 2 fixpontos **ALU** ---> többszálúság (hyperthreding): 5% többlet a lapkán ~ két **CPU**.

Mindkét **ALU** kétszeres órajel sebességgel fut

Többszálúság (hyperthreding, 8.7. ábra)

Többszörözött regiszter készlet esetén valósítható meg némi szervező hardver hozzáadásával.

(a)	**A1**	**A2**			**A3**	**A4**	**A5**			**A6**	**A7**	**A8**
(b)	**B1**			**B2**			**B3**	**B4**	**B5**	**B6**	**B7**	**B8**
(c)	**C1**	**C2**	**C3**	**C4**			**C5**	**C6**			**C7**	**C8**
	A1	**B1**	**C1**	**A2**	**B2**	**C2**	**A3**	**B3**	**C3**	**A4**	**B4**	**C4**

Óraciklus →

Az (a), (b) és (c) processzus külön futtatva az üres négyzeteknél várakozni kényszerül a memóriához fordulások miatt.

Pentium 4

Gépi utasítások → **RISC** szerű mikroutasítások, több mikroutasítás futhat egyszerre: szuperskaláris gép, megengedi a sorrenden kívüli végrehajtást is.

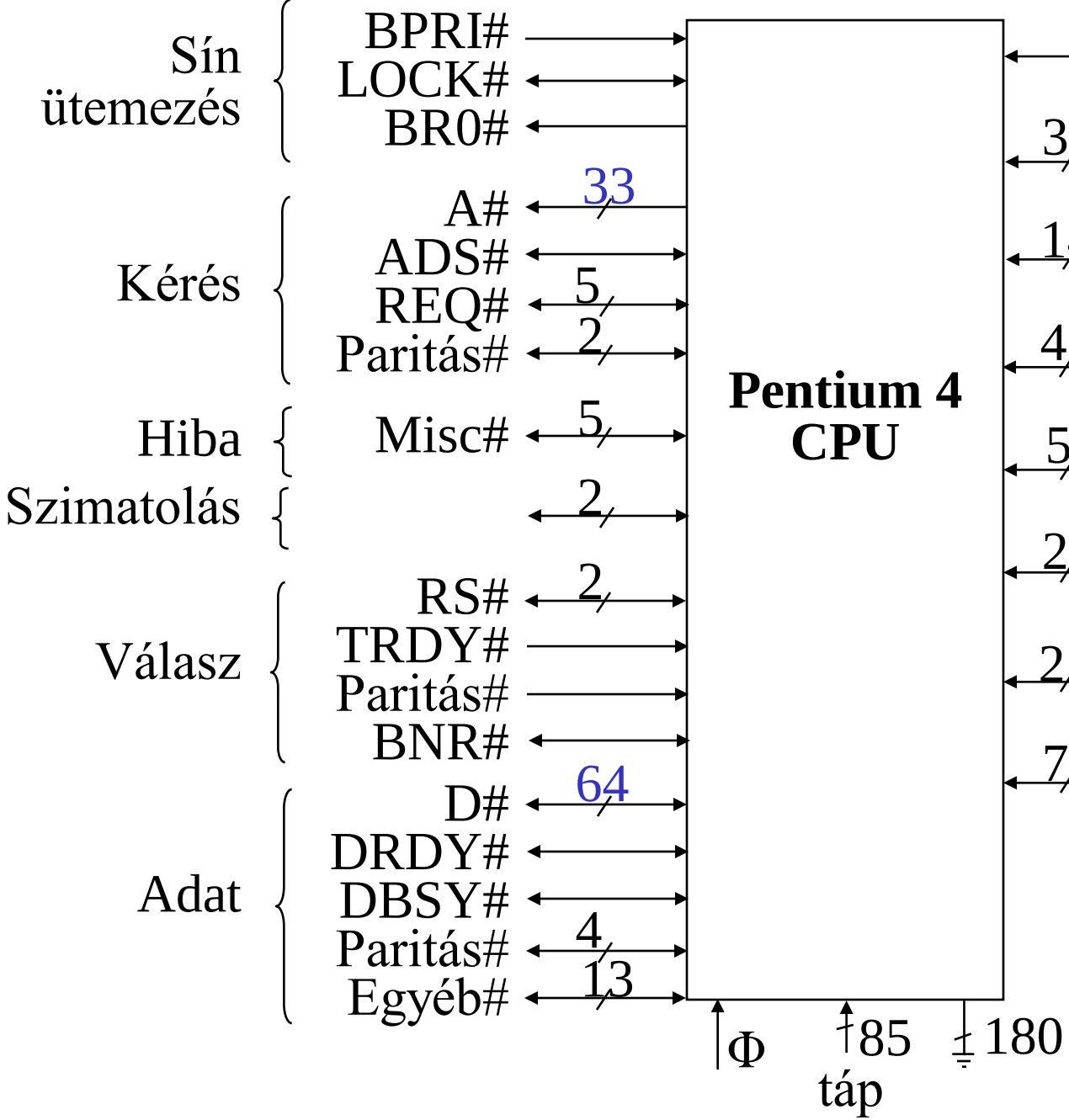
2-3 szintű belső gyorsító tár.

L1: **8 KB** utasítás + nyomkövető akár 12000 dekódolt mikroutasítás tárolására + **16 KB** adat.

L2: **256 KB – 1 MB**, 8 utas halmaz kezelésű, 128 bájtos gyorsító sor.

Az Extrem Edition-ban **2 MB** (közös) **L3** is van.

Multiprocesszoros rendszerekhez szimatolás - snoop.



Pentium 4 logikai lábkiosztása (3.45. ábra)

Pentium 4 memória sín

A memóriaigények, tranzakciók 6 állapota: 6 fázisú csővezeték (**3.45. ábra** bal oldal) fázisonként külön vezérlő vonalakkal (amint a mester megkap valamit, elengedi a vonalakat):

0. **Sín ütemezés (kiosztás, bus arbitration)**: eldől, hogy melyik sínmester következik,
1. **Kérés**: cím a sínre, kérés indítása,
2. **Hibajelzés**: a szolga hibát jelez(het),
3. **Szimatolás**: a másik **CPU** gyorsító tárában,
4. **Válasz**: kész lesz-e az adat a következő ciklusban,
5. **Adat**: megvan az adat.

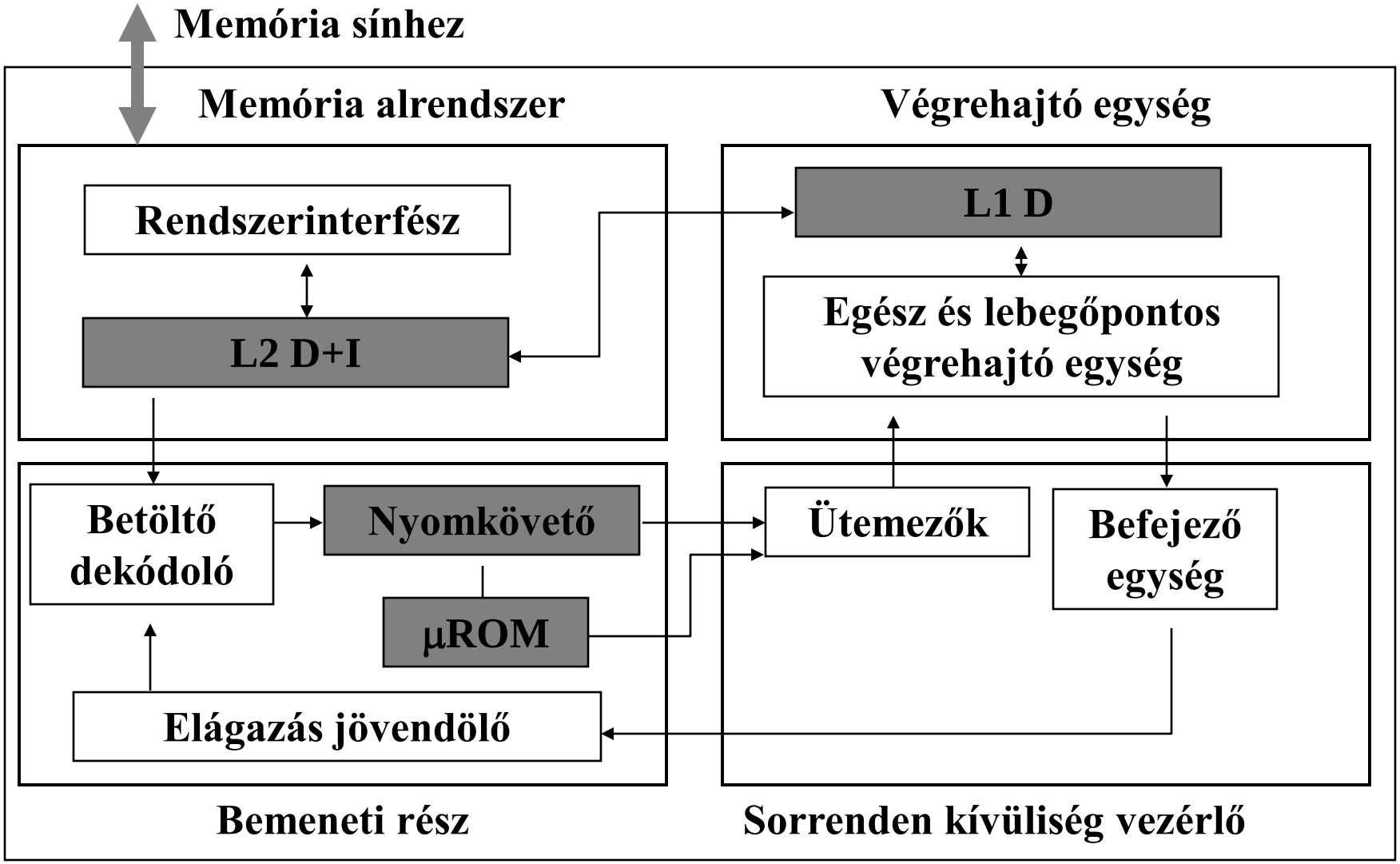
Pentium 4 memória sín csővezetéke (3.46. ábra)

Φ: tranzakció	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
1	K	H	S	V	A							
2		K	H	S	V	A						
3			K	H	S	V	A					
4				K	H	S	V		A			
5					K	H	S	V		A		
6						K	H	S		V	A	
7							K	H	S		V	A

Ütemezés (nem ábrázoltuk), csak akkor kell, ha másé a sín.

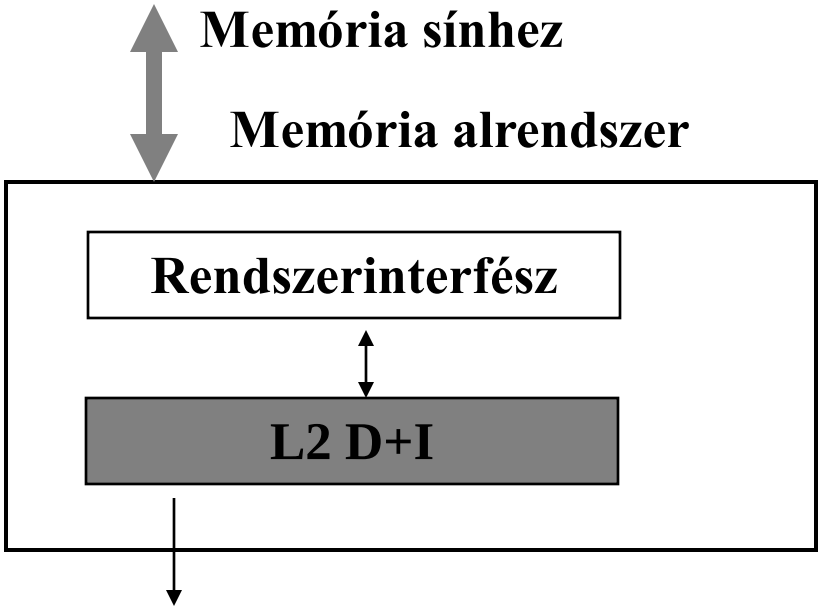
K: kérés, **H**: hiba, **S**: szimatolás (átkérés), **V**: válasz, **A**: adat

A Pentium 4 mikroarchitektúrája



4.46. ábra. A Pentium 4 blokkdiagramja

4.46. ábra. A Pentium 4 memória alrendszere



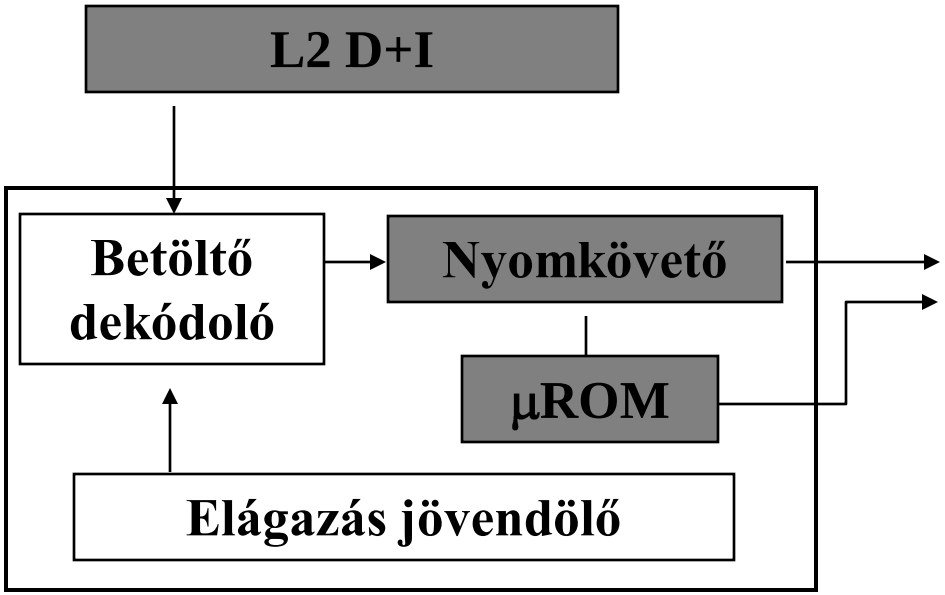
L2 256 KB az első,
512 KB a második,
1 MB a harmadik
generációs
Pentium 4-ben.

L2 8 utas halmaz kezelésű, késleltetve visszaíró
128 bájtos gyorsító sor, minden második ciklusban kezdődhet egy 64 bájtos feltöltés a memóriából.

Előre betöltő: megpróbálja **L2**-be tölteni azt a gyorsító sort, amelyre majd szükség lesz (nincs az ábrán).

4.46. ábra. A Pentium 4 bemeneti rész

L2-ből betölti és dekódolja a programnak megfelelő sorrendben az utasításokat. Az utasításokat **RISC** szerű mikroműveletek sorozatára bontja. Ha több, mint 4 mikroművelet szükséges, akkor **μROM**-ra történik utalás. A dekó-dolt mikroműveletek a Nyomkövetőbe kerülnek (nem kell újra dekódol-ni).



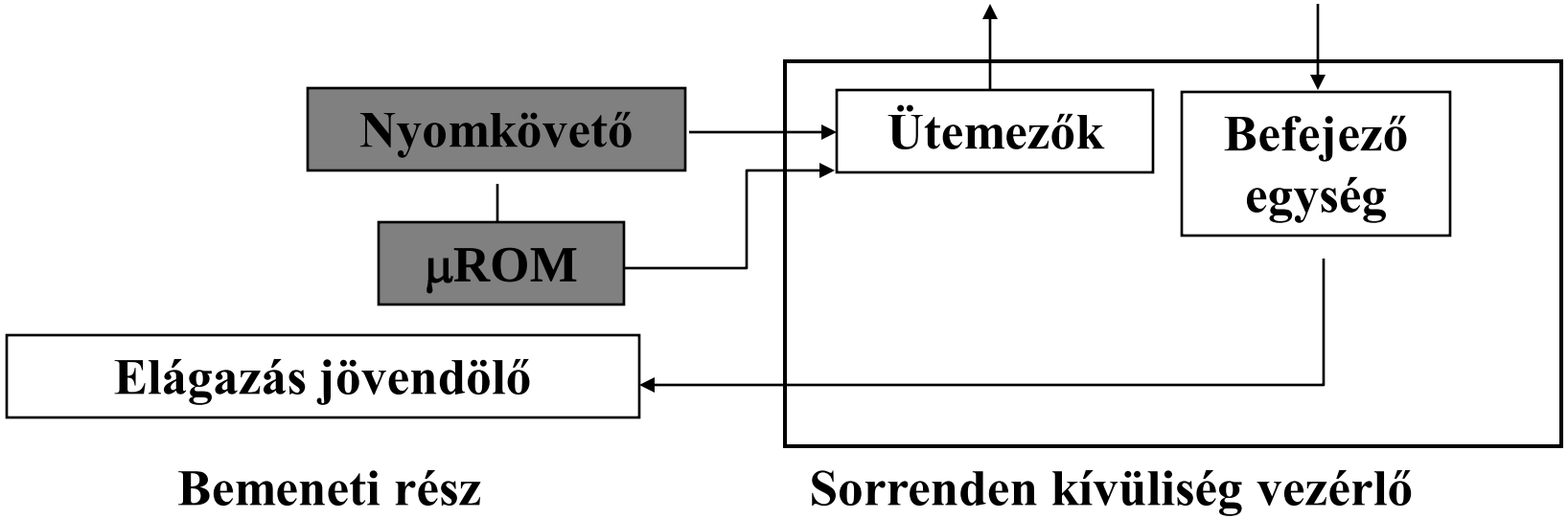
Bemeneti rész

A bemeneti rész az utasításokat **L2**-ből kapja, egyszerre 64 bitet. Ezeket dekódolja, a nyomkövető gyorsító tárban tárolja (akár 12 K mikroműveletet). 6 mikroműveletet csoportosít minden nyomkövető sorban.

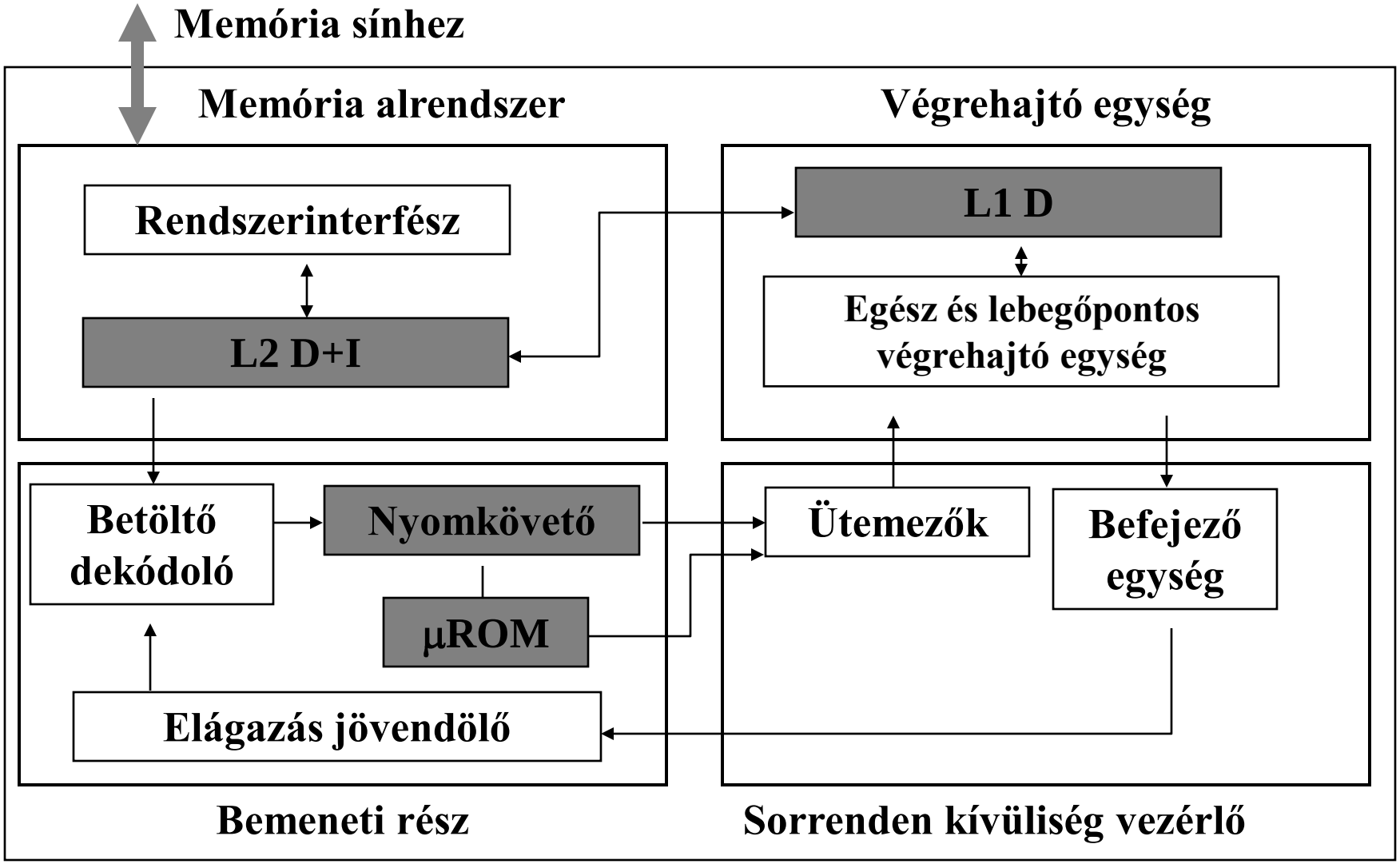
Feltételes elágazásnál az utolsó **4 K** elágazást tartalmazó **L1 BTB**-ből (Branch Target Buffer – elágazási cél puffer) kikeresi a jövendölt címet, és onnan folytatja a dekódolást. Ha az elágazás nem szerepel **L1 BTB**-ben, akkor statikus jövendölés történik: visszafelé ugrást végre kell hajtani, előre ugrást nem.

4.46. ábra. Sorrenden kívüliség vezérlő

Az utasítások a programnak megfelelő sorrendben kerülnek az ütemezőbe, eltérő sorrendben kezdődhet a végrehajtásuk (esetleg regiszter átnevezéssel), de a pontos megszakítás követelménye miatt az előírt sorrendben fejeződnek be.



A Pentium 4 mikroarchitektúrája



4.46. ábra. A Pentium 4 blokkdiagramja

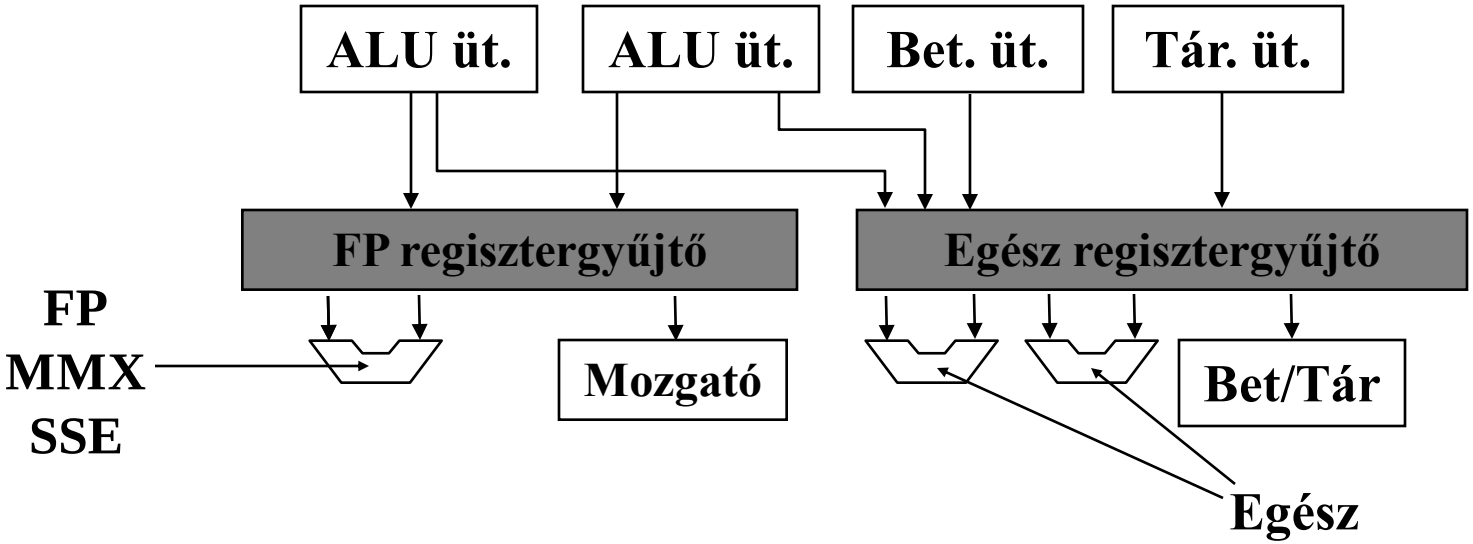
Ha egy mikroműveletnek minden inputja rendelkezésre áll, akkor az esetleges **WAR** vagy **WAW** függőséget a 120 firkáló regiszter segítségével kiküszöböli. **RAW** függőség esetén a mikroműveletet várakoztatja, és a rákövetkező mikroműveleteket kezdi feldolgozni.
Egyszerre akár 126 utasítás feldolgozása is folyamatban lehet, köztük 48 betöltés és 24 tárolás.

Az utasítások a programnak megfelelő sorrendben kerülnek az ütemezőbe, eltérő sorrendben kezdődhet a végrehajtásuk, de az előírt sorrendben fejeződnek be.

Pontos megszakítás: a megszakítás előtti összes utasítás befejeződött, az utána következőkből egy sem kezdődött el.

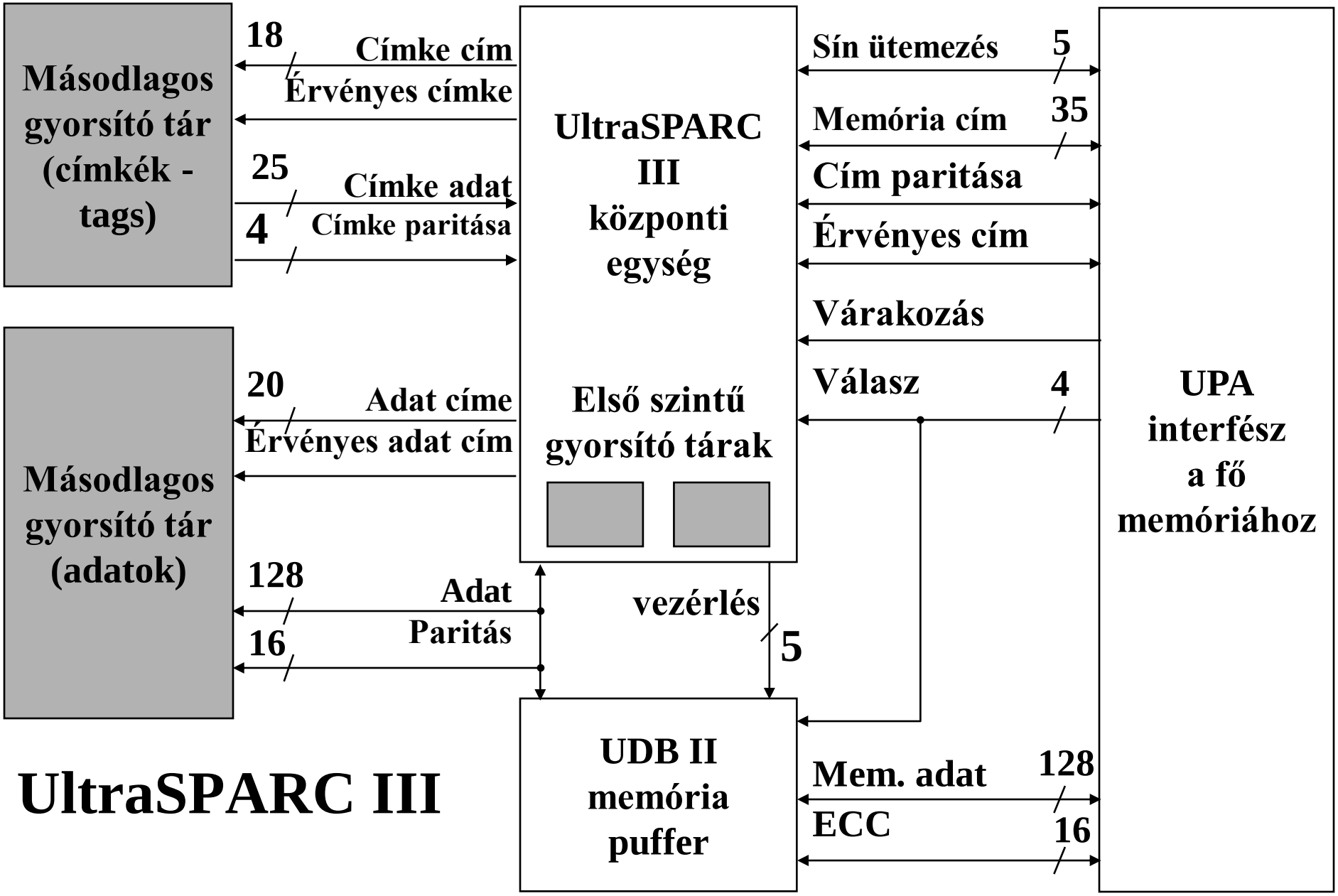
Az egyik egész aritmetikájú **ALU** az összes logikai, aritmetikai, és elágazó, a másik csak az összeadó, kivonó, léptető és forgató utasítás végrehajtására képes.

Mindkét regisztergyűjtő 128 regisztert tartalmaz, időben változik, hogy melyikben van **EAX**, …



UltraSPARC III (2000)

- 64 bites **RISC** gép, felűről kompatibilis a 32 bites **SPARC V8** architektúrával és az **UltraSPARC I, II-**vel. Új a **VIS 2.0** utasításkészlet (3D grafikus alkalmazásokhoz, tömörítéshez, hálózat kezeléshez, jelfeldolgozáshoz, stb).
- Több processzoros alkalmazásokhoz készült. Az összekapcsoláshoz szükséges elemeket is tartalmazza.
- 2000-ben 0.6, 2001-ben 0.9, 2002-ben 1.2 GHz, órajel ciklusonként 4 utasítást tud elvégezni.



UltraSPARC III

UltraSPARC III

CPU 29 millió tranzisztor, 4 **CPU** közös memóriával használható. 1368 láb (**3. 47. ábra**). 64 (jelenleg csak 44) bites cím és 128 bites adat lehetséges.

Belső gyorsító tár (**32 KB** utasítás + **64 KB** adat).

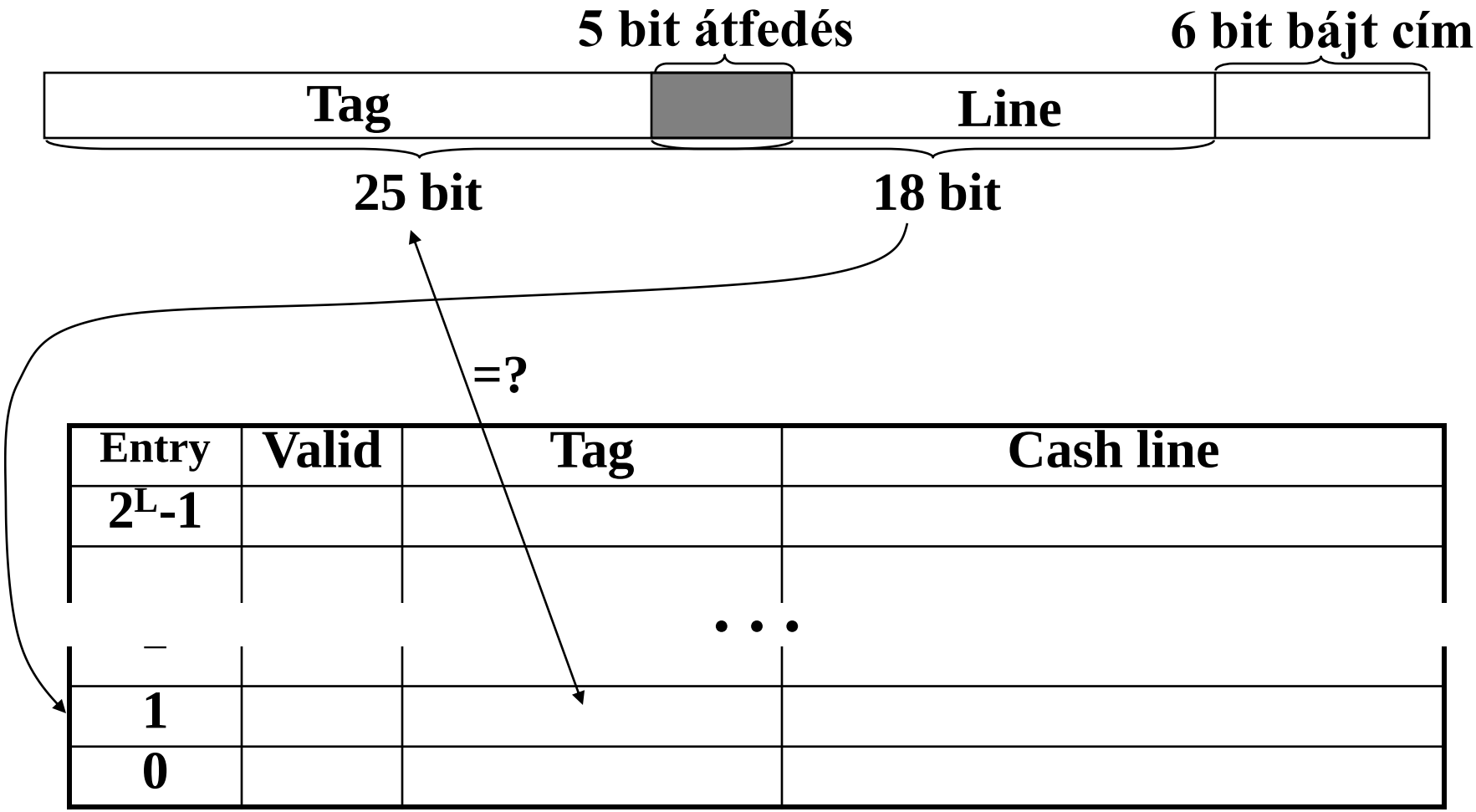
2 KB előre betöltő és tároló gyorsítótár **L2** eléréséhez.

A gyorsító sor (cache line) mérete **64 (32) B**.

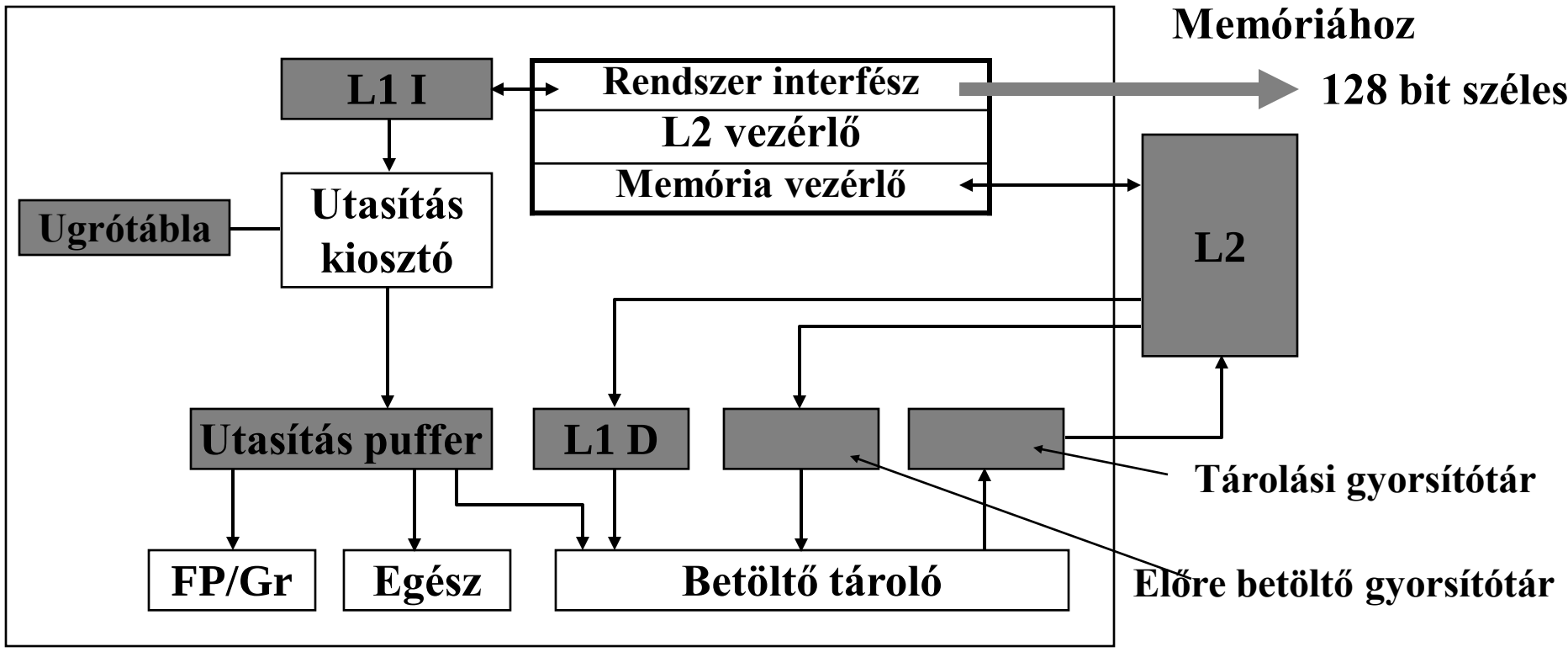
Külső **1 - 8 MB** (**UltraSPARC II**-nek **0.5-16 MB**).

8 K - 256 K db **64 B**-os gyorsító sor (cache line) lehet. A címzéséhez **13 – 18** bit szükséges. A **CPU** mindig **18** bites **Line** címet (**Címkeazonosító**t) ad át.

A cím 64 bit-es, de egyelőre 44 bit-re van korlátozva

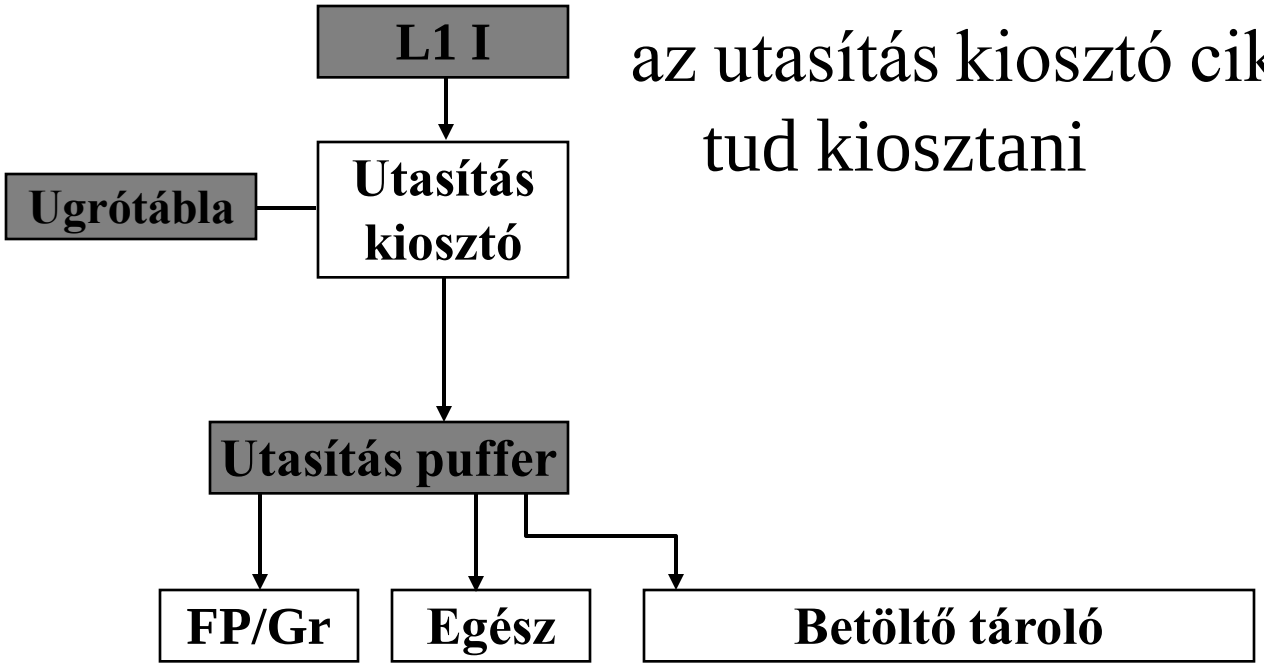


Az UltraSPARC III CPU mikroarchitektúrája



4.48. ábra. Az UltraSPARC III CPU blokkdiagramja

L1 I 32 KB 4 utas halmazkezelésű,
az utasítás kiosztó ciklusonként 4 utasítást tud kiosztani



Két egész aritmetikájú ALU + regiszterek + firkáló regiszterek,

Lebegőpontos ALU-k: összeadó/kivonó, szorzó/osztó + 32 regiszter + grafikai utasítások.

UltraSPARC III CPU mikroarchitektúrája

A **SPARC** sorozat **RISC** elgondoláson alapul. A legtöbb utasításnak két forrás és egy cél regisztere van.

Előre betöltés speciális utasításokkal, és a visszafelé kompatibilitás miatt hardveresen is.

2 bites elágazás jövendölő + statikus elágazás jövendölés.

I-8051 (1980)

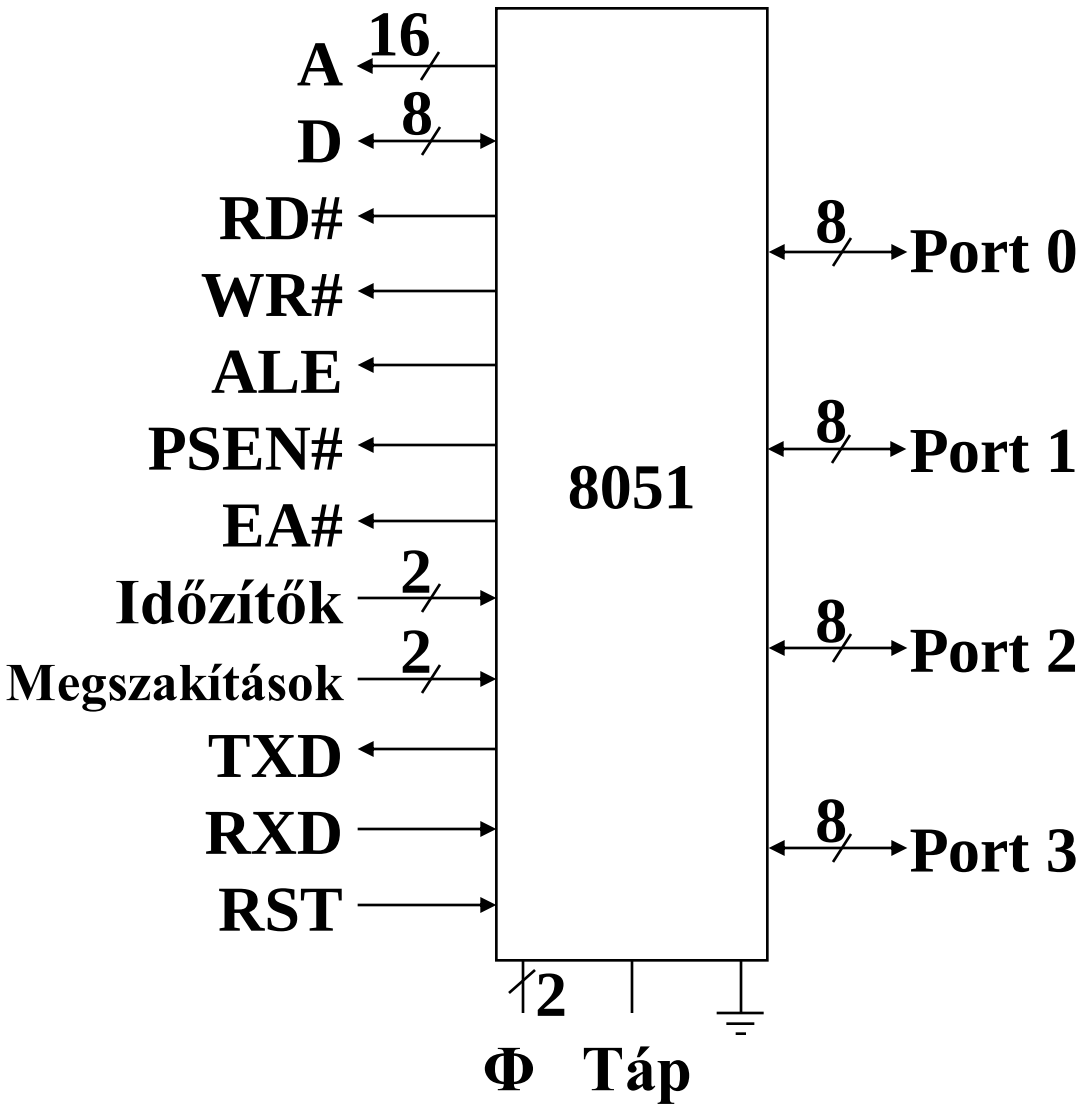
Cél: beépített rendszerekben való alkalmazás.

Fő szempont: olcsóság (ma már 10-15 ¢), sokoldalú alkalmazhatóság.

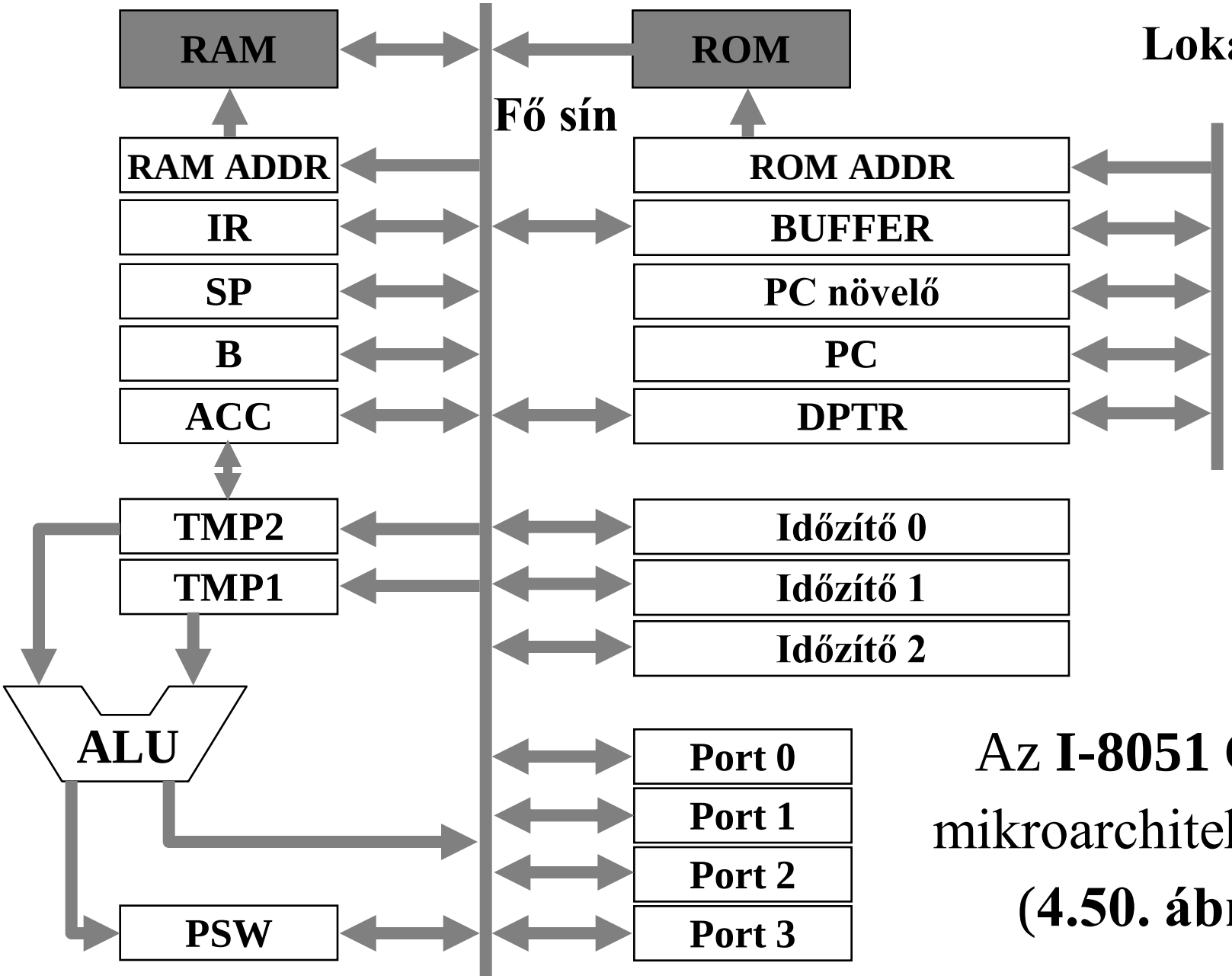
A memóriával, be- és kivitellel együtt egyetlen lapkára integrált számítógép. 40 multiplexelt lábú standard tokban kerül forgalomba. 60 000 tranzisztor. 4 KB ROM, 128 B RAM, max. 64 KB külső memória.
16 címvezeték. 8 bites adat sín. 32 K/B vonal
4 db 8 bites csoportba rendezve, ezek mindegyike hozzáköthető nyomógombhoz, kapcsolóhoz, LED-hez, …
Időzítők.

Pl. Rádiós óra: nyomógombok, kapcsolók, kijelző.

Az **I-8051** logikai lábkiosztása (**3.50. ábra**)



Address
Data
RD# olvas a memóriából
WR# ír a memóriába
Address Latch Enable:
külső memória esetén:
a sínen érvényes a cím
Program Store ENable:
olvasás a programot
tároló memóriából
External Access (az értéke állandó): (1) a 0-4095 címek a belső,
(0) a külső memóriára vonatkoznak



Az **I-8051 CPU** mikroarchitektúrája **(4.50. ábra)**

Az **I-8051 CPU** mikroarchitektúrája (**4.50. ábra**)

A legtöbb utasítás egy óraciklust igényel. A ciklus hat állapota:

1. Az utasítás a **ROM**-ból a **fősín**re és **IR**-be kerül.
2. Dekódolás, **PC** növelése.
3. Operandusok előkészítése.
4. Egyik operandus a **fősín**re, onnan általában **TMP1**-be, a másik **ACC**-ből **TMP2**-be kerül.
5. Az **ALU** végrehajtja a műveletet.
6. Az **ALU** kimenete a **fősín**re kerül, **ROM ADDR** felkészül a következő utasítás olvasására.

Összehasonlítás

Pentium 4 **CISC** gép
egy **CISC** utasítás --> több **RISC** mikroutasítás

UltraSPARC III **RISC** gép

I-8051 inkább **RISC**, mint **CISC** gép

picoJava II verem gép, sok memória hivatkozás
több **CISC** utasítás is --> egyetlen **RISC** mikroutasítás