HÁLÓZATI RENDSZEREK ÉS SZOLGÁLTATÁSOK TANSZÉK

MŰEGYETEM 1782

Budapest,
2025.05.07.

# SZÁMÍTÓGÉP ARCHITEKTÚRÁK

Elágazásbecslés

**Horváth Gábor, Belső Zoltán**

BME Hálózati Rendszerek és Szolgáltatások Tanszék

ghorvath@hit.bme.hu, belso@hit.bme.hu

# PROCEDURÁLIS EGYMÁSRAHATÁSOK

- Procedurális egymásrahatás:
  - Ugrások okozzák
  - Megtörik az utasításfolyam szekvenciális viselkedését
- Mi a gond?
  - Pl. feltételes ugrás esetén tudni kell:
    - Az elágazás ***kimenetelét*** (ugrik-e, vagy sem)
    - Ugrás esetén az ***ugrási címet*** (hová ugrik)
  - IF dolga: utasítások betöltése
    - Nincs ideje feltételkiértékelésre **és** címszámításra!
    - Egy tipphez van ideje
      - Ha bejön, annak örül
      - Ha nem:
        - A tévedésből elkezdett utasításokat érvényteleníteni kell
        - Legközelebb tanul a hibájából

# AZ ELÁGAZÁSBECSLÉS FELADATA

- Szekvenciális futást megtörő utasítások:
  - **Feltétel nélküli ugrás:**
    - Direkt: `JUMP -28`
    - Indirekt: `JUMP R1`
    - `-28` vagy `R1`: ugrási cím
  - **Feltételes ugrás:**
    - Direkt: `JUMP -28 IF R2>0`
    - Indirekt: `JUMP R1 IF R2>0`
    - `R2>0`: ugrási feltétel
    - `-28` vagy `R1`: ugrási cím
  - **Szubrutinhívás:**
    - Direkt: `CALL -28`
    - Indirekt: `CALL R1`
    - Visszatérés: `RET`, ugrási cím a veremből
- Az elágazásbecslés feladata:
  - Kimenetel becslése, ha van ugrási feltétel
  - Ugrási cím becslése

# AZ ELÁGAZÁSBECSLÉS JELENTŐSÉGE

- Mit nyerünk, ha bejön a becslés?
  - Fennakadás nélkül zajlik az utasítások betöltése
- Mit vesztünk, ha rossz a becslés?
  - Minél később derül ki a rossz döntés, annál több időt fecsérel a processzor tévedésből betöltött utasításokra
  - Minél hosszabb a pipeline, annál nagyobb a veszteség!
  - Példa (Intel Core i7 Nehalem):
    - 4 utas szuperskalár processzor
    - A belépéstől számítva 17 ciklus múlva számolja ki a tényleges ugrási címet és értékeli ki az ugrási feltételt
    - Tfh. minden negyedik utasítás feltételes ugrás
    - Tfh. az elágazásbecslő pontossága 67%
    - Tfh. más egymásrahatás nincs

# AZ ELÁGAZÁSBECSLÉS JELENTŐSÉGE

- A példa folytatása:
  - 4 utas szuperskalár: órajelciklusonként 4 utasítás az átvitel
    → Ideális eset: átlag **0.25 órajelciklus/utasítás**
  - Utasítások negyede ugrás, minden ugrás 0.33 valószínűséggel 17 ciklusnyi extra késleltetést (kidobott időt) jelent
  - Átvitel: 0.25 + 0.25*0.33*17 = **1.65 ciklus/utasítás**
  - **6.6x** gyorsabb lenne a processzor, ha tökéletes lenne az elágazásbecslője!!!

# AZ ELÁGAZÁSBECSLÉS JELENTŐSSÉGE

- A rossz döntés okozta kiesett ciklusok száma:

| CPU | Kiesett ciklusok száma |
|---|---|
| Intel Pentium I MMX | 4-5 |
| Intel Pentium 4 | átlagosan 45(!) |
| Intel Core2 | 15 |
| Intel Core i7 Skylake | 16.5 |
| Intel Atom | 13 |
| AMD K8 és K10 | 12 |
| AMD Ryzen | 19 |
| Via Nano | 16 |
| ARM Cortex A53 | 7 |
| ARM Cortex A72 | 15 |

# Ugrási feltétel kimenetelének becslése

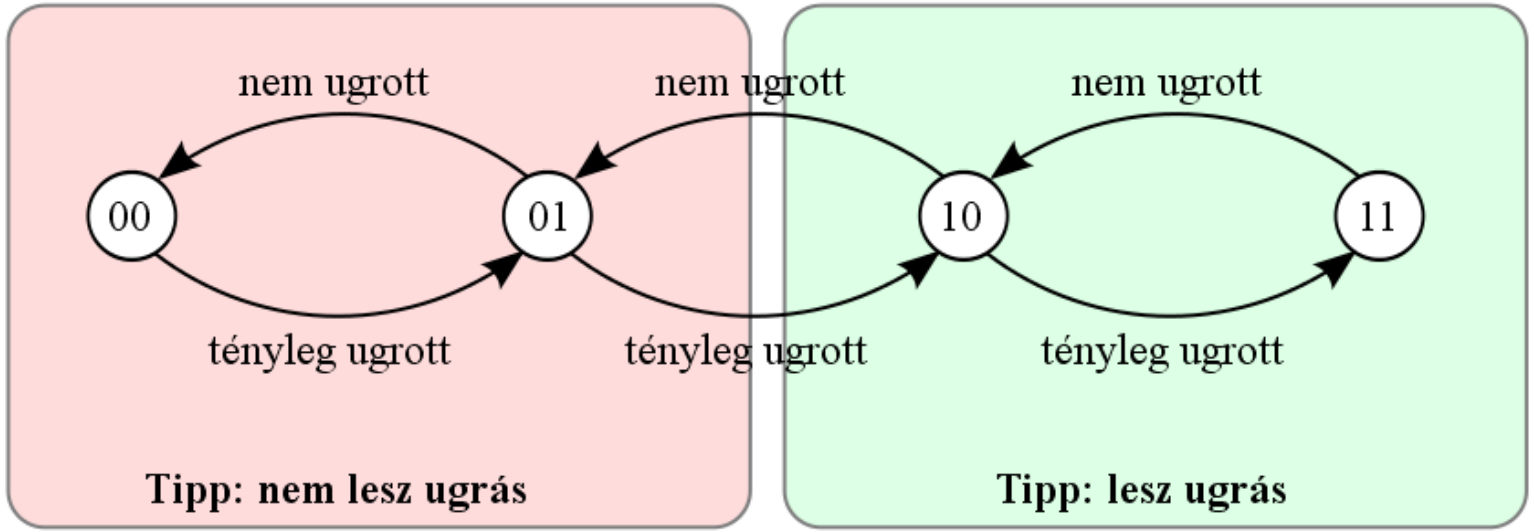
# MI IS A FELADAT?

- Ismert (lehet): az ugró utasítás kimenetelei a múltban
- Feladat: mi lesz a következő kimenetel?
  - **1111111?**
  - **11111101101111011111?**
  - **11001100110011?**
  - **111111111110000000000000?**

- Elvárások:
  - Legyen a becslő *gyors*
  - Legyen a becslő *egyszerű*
  - Legyen nagy a találati aránya

# EGYSZERŰ ÁLLAPOTGÉP ALAPÚ MÓDSZER

- Minden ugró utasításhoz egy állapotgépet rendelünk
- Ez tárolja, mennyire szeret ugrani
- Ha a közepesnél jobban szeret, akkor ugrásra tippelünk
- Táblázat frissítése: a **tényleges** bekövetkezés ismeretében



# EGYSZERŰ ÁLLAPOTGÉP ALAPÚ MÓDSZER

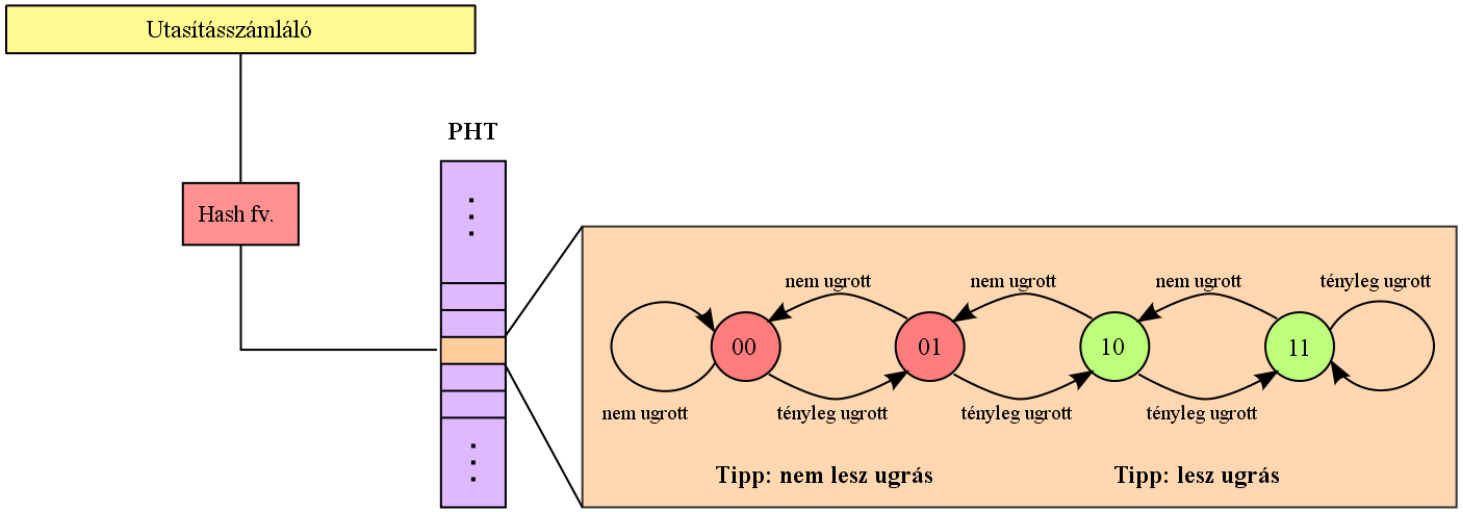
- Hatékonyságvizsgálat:

```
for (i=0; i<m; i++) {
   for (j=0; j<n; j++) {
        ...
  }
}
```

- Belső ciklus becslését vizsgáljuk: n*m döntés
- 1 bites számlálóval: 2*m rossz becslés
- 2 bites számlálóval: m rossz becslés

# EGYSZERŰ ÁLLAPOTGÉP ALAPÚ MÓDSZER

- Hol tároljuk az állapotot?
  - Valamilyen cache szerű szervezéssel
    - 32 bit-es tag, 2 bites adat → nem éri meg!
  - Utasítás cache blokkjaiban az utasítások mellett → AMD
  - Külön táblázatban: **PHT** (Pattern History Table)
    - PI: Pentium 1



# KORRELÁCIÓ FIGYELEMBE VÉTELE

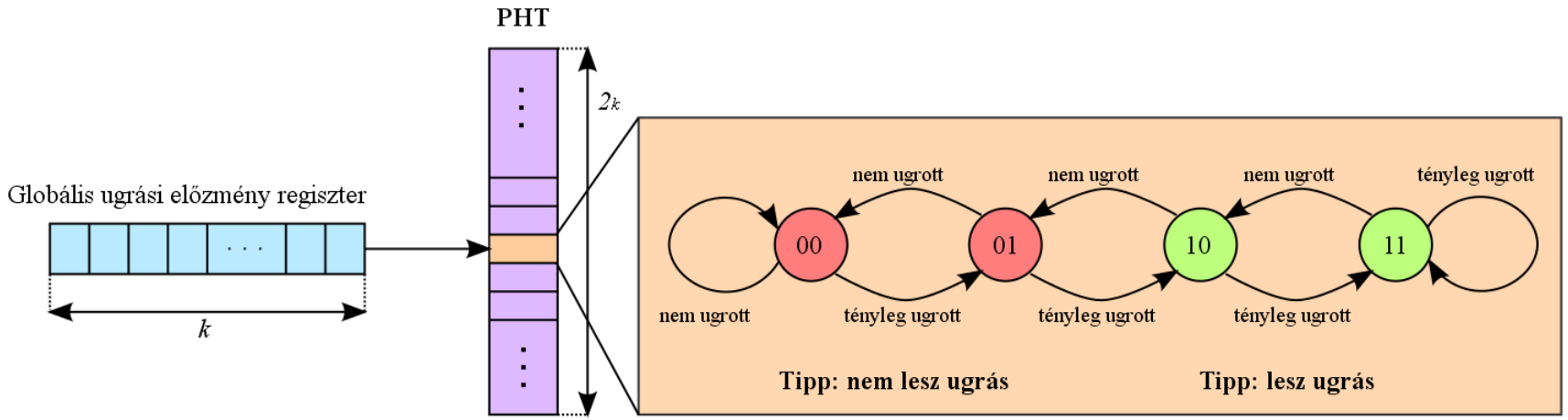
- Az ugró utasítások kimenetele gyakran függ más ugró utasítások kimenetelétől
- Példa:

```
if (a==2)
    a = 0;
if (b==2)
    b = 0;
if (a!=b) {
    ...
}
```

- Ha az első kettő igaz volt, a harmadik nem lesz igaz!
- Jó lenne kihasználni

# KORRELÁCIÓ FIGYELEMBE VÉTELE

- A trükk: tároljuk az egymást követő utasítások kimeneteleit egy shift regiszterben → **globális előzmény regiszter** (Global Branch History Register, GBHR)
  - Ha feltételes ugrás történik, a *tényleges* kimenetele jobbról lép be (0 vagy 1)
  - Egy k bites GBHR az utolsó k ugrás kimenetelét tárolja
  - A PHT-t ezzel indexeljük

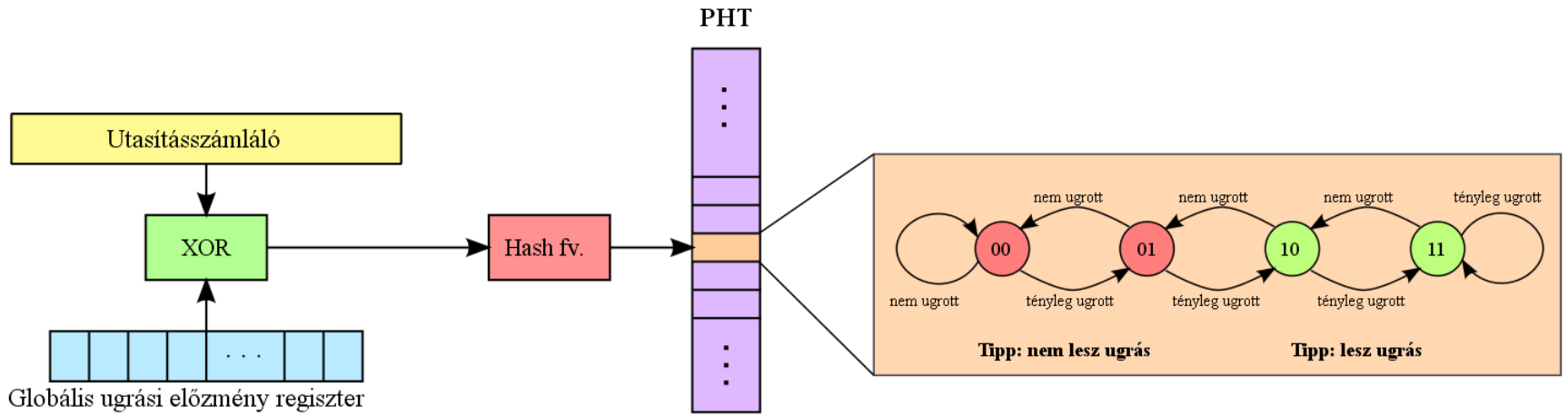


# KOMBINÁLT PREDIKTOROK

- Tanultunk egyszerű állapotgép alapú becslést
  - Utasításra lokális döntést hoznak
- Tanultunk korrelációt figyelembe vevő módszert
  - Kizárólag a globális előzmények alapján dönt
- Miért ne kombináljuk a kettőt?

# A GSHARE PREDIKTOR

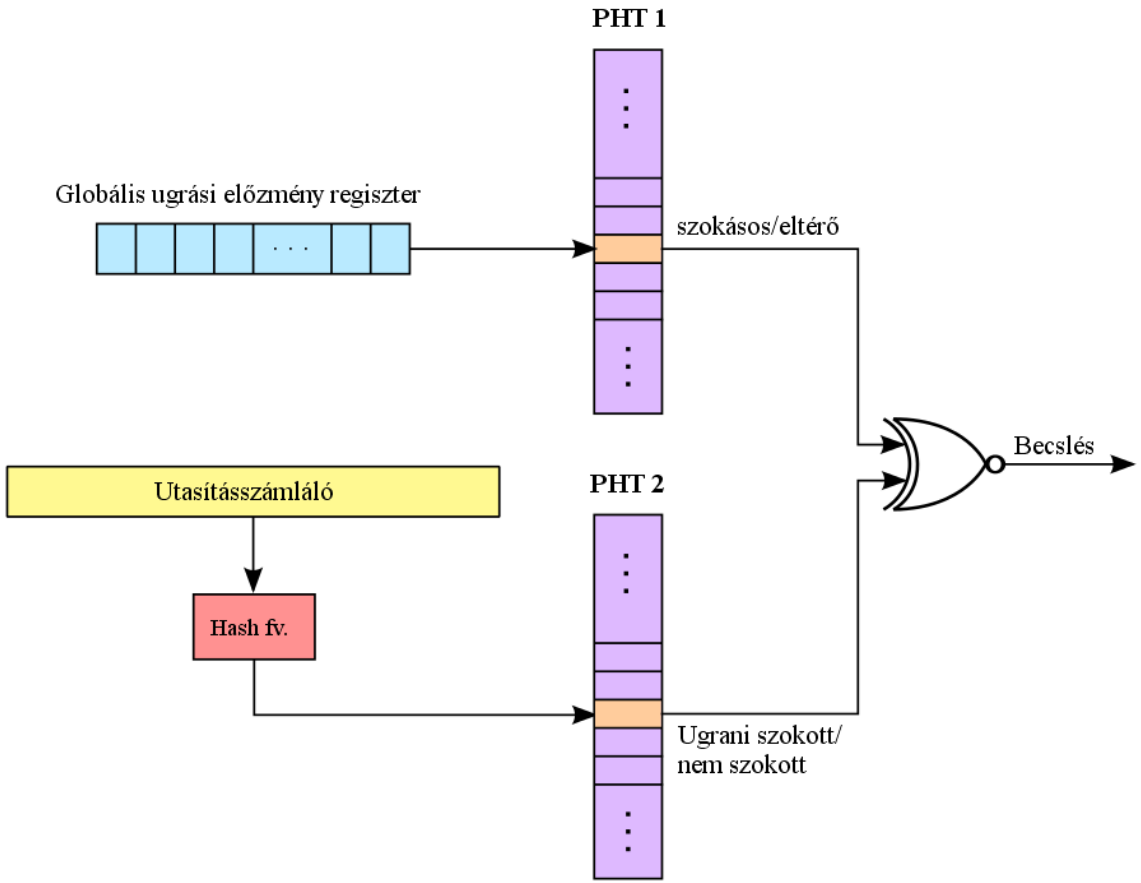
- Kombinálja a lokális és globális információkat
  - Lokális döntés: megnézzük, hogy az adott utasításhoz mit jósol a PHT
  - Globális döntés: megnézzük, hogy az előzmények mellett mit jósol a PHT
  - Kombináljuk a kettőt: PC XOR GBHR




- Nagyon egyszerű, és meglepően pontos!
- SPARC, POWER4, XBox 360, AMD Athlon,
  egy kicsit módosított változata az ARM Cortex A8-ban

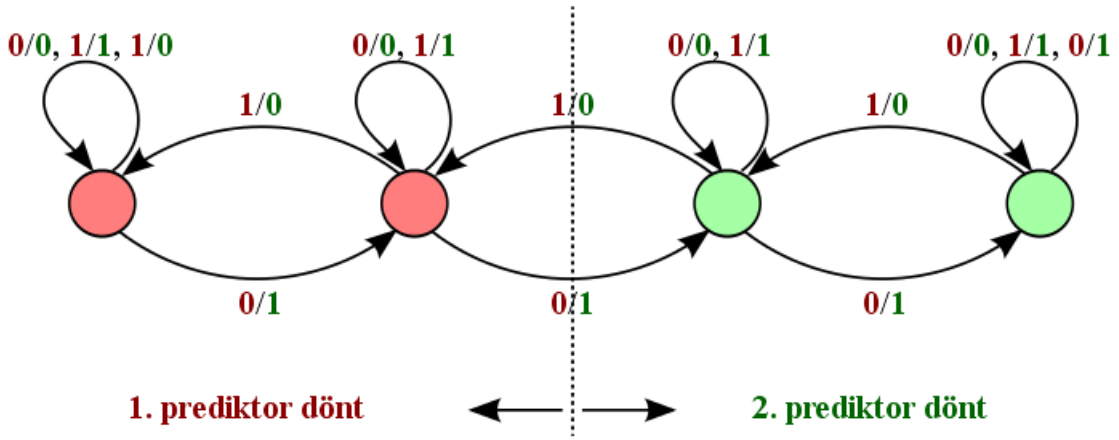
# AZ AGREE PREDIKTOR

- Lokális eljárás: merre „húz” az adott ugró utasítás
- Globális eljárás: az ugrás a tipikus irányba fog-e megvalósulni



# "TOURNAMENT" PREDIKTOROK

- Két prediktor működik egyszerre, egy lokális és egy globális
- Mindig csak az egyik dönt: az, amelyik mostanában pontosabban becsült
- Ezt egy állapotgép követi, hogy mostanában melyik vált be jobban



# Ugrási cím becslése

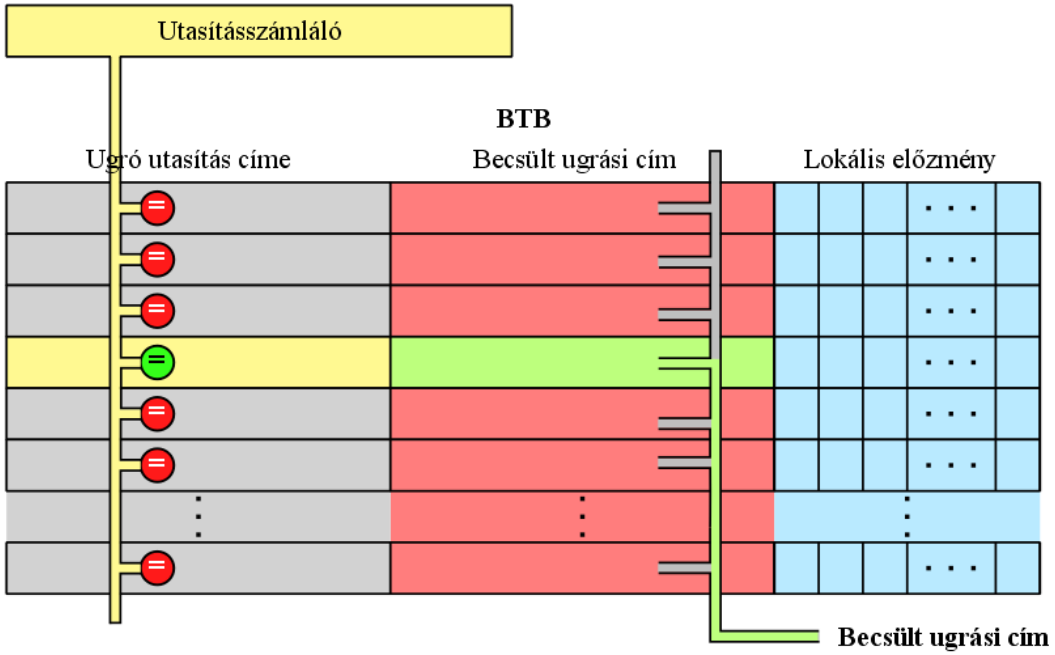
# UGRÁSI CÍM BECSLÉSE

- Honnan kell betölteni a következő utasítást?
  - Ezt az IF-nek sürgősen tudni kell!
  - Feltételes és feltétel nélküli ugrásoknál egyaránt kérdés.
- Kiolvassuk egy speciális "***ugrási cím buffer***"-ból (BTB: Branch Target Buffer)

  → Könnyebb/gyorsabb innen kiolvasni, mint kiszámolni
  - A BTB mezői:
    - tag: az ugró utasítás címe
    - várható ugrási célcím
  - Szervezése:
    - Valamilyen cache szervezés, pl. 4 utas asszociatív
    - Tartalommenedzsment: LRU
  - Téves becsléskor korrigáljuk a régi értéket

# AZ UGRÁSI CÍM BUFFER

- Ha már egyszer minden egyes ugró utasításhoz van bejegyzése, mást is tárolhatunk benne:
  - Az ugrási hajlandóságot
  - Az utasítás korábbi kimeneteleit (előzmény)
- Például:



# SZUBRUTINHÍVÁSBÓL VISSZATÉRÉSI CÍM BECSLÉSE

- A **RET** speciális ugró utasítás
  - Ugrási cím a lassú memóriában van (stack)
- Becsléshez hatékony adatszerkezet: **return stack**
  - A CPU-ban található
  - Nagy sebesség, kis kapacitás
  - Szubrutinhíváskor a visszatérési cím
    - az igazi (lassú) stack tetejére kerül
    - ...a return stack tetejére is bekerül!
  - Visszatéréskor:
    - Nem kell megvárni az igazi lassú stack-et!
    - A return stack ciklusidőn belül megmondja a címet
  - Addig hatékony, amíg nem telik meg
    - Ha nincs túl sok egymásba ágyazott függvény hívás

# TIPIKUS UGRÁSI CÍM BUFFER PARAMÉTEREK

| Processzor | BTB bej.-ek száma | BTB szervezése | Return stack |
|---|---|---|---|
| Intel Pentium I MMX | 256 | 4 utas asszoc. | nincs |
| Intel Pentium 4 | 4096 | val. 8 utas asszoc. | nincs |
| Intel Core2 (ciklusokhoz) | 128 | 2 utas asszoc. | 16 |
| Intel Core2 (indir. ugr.) | 8192 | 4 utas asszoc. | |
| Intel Core2 (egyéb ugr.) | 2048 | 4 utas asszoc. | |
| Intel Atom | 128 | 4 utas asszoc. | 8 |
| AMD Steamroller | L1: 512, L2: 10240 | L1: 4 utas, L2: 5 utas | 24 |
| AMD Ryzen | 8/256/4096 | ? | 31 |
| Via Nano | 4096 | 4 utas asszoc. | Nagyon mély |
| ARM Cortex A8/A9 | 512 | 2 utas asszoc. | 8 |

# NEHÉZSÉGEK

- Hogy lehet kibabrálni az ugrási cím becslővel?
- Ugrabugráljunk kiszámíthatatlanul!
  - Pl. heterogén kollekció bejárása
    és virtuális függvények hívása c++-ban
  - Függvénypointereket tartalmazó tömb
- Lehetséges megoldás:
  - A gond az, hogy az ugrási cím buffer csak egyetlen ugrási címet tud tárolni
  - Tároljon többet! A globális előzmények függvényében dönt a becslő, hogy melyik ugrási címet dobja be tippként
  - Pl.: ARM Cortex A15, Intel Core Nehalem, AMD Ryzen, stb.

# *Elágazásbecslés-tudatos programozás*

# ELÁGAZÁSOK SZÁMÁNAK CSÖKKENTÉSE

- 1. példa:

```
for (int i=0; i<N; i++)
    if (data[i] > 500)
        sum += data[i];
```

- A feltétel kiváltható egy logikai kifejezéssel
  - `data[i]>500` ↔ `(data[i]-501) >= 0`
  - `data[i]-501` jobbra shiftelve (aritmetikai shift!):
    - `000...0` vagy `111...1`
  - Maszkoljuk vele az összeadást!
- Elágazásmentesített kód:

```
for (int i=0; i<N; i++) {
    int t = (data[i]-501) >> 31;
    sum += ~t & data[i];
}
```

# ELÁGAZÁSOK SZÁMÁNAK CSÖKKENTÉSE

- 2. példa:

```
for (int i=0; i<N; i++)
    if (min<=data[i] && data[i]<=max)
        sum += data[i];
```

- Első lépés: két feltételből egyet csinálunk
  - `min<=data[i] && data[i]<=max`

    ↔ `(unsigned)(data[i]-min)<=max-min`
  - Már csak egy feltételünk van:

```
for (int i=0; i<N; i++)
    if ((unsigned)(data[i]-min) <= max-min)
        sum += data[i];
```

- Második lépés: az előbbi trükk

```
for (int i=0; i<N; i++) {
    int t = (max-min-(unsigned)(data[i]-min)) >> 31;
    sum += ~t & data[i];
}
```

# ELÁGAZÁSOK SZÁMÁNAK CSÖKKENTÉSE

- 3. példa:

```
for (int i=0; i<N; i++)
   if (!((data[i]>='a' && data[i]<='z') || (data[i]>='A' && data[i]<='Z')))
        data[i] = ' ';
```

- Trükk: segédtömb (**look-up table**, LUT)
  - Minden betűre a cserélendő karakter
- Segédtömb előkészítés + elágazásmentes kód:

```
for (int j=0; j<256; j++)
    if (!((j>='a' && j<='z') || (j>='A' && j<='Z')))
        LUT[j] = ' ';
    else
        LUT[j] = j;
for (int i=0; i<N; i++)
    data[i] = LUT[data[i]];
```

- Az előkészítés ideje fix
  - Ha N nagy, eltörpül

# ELÁGAZÁSOK SZÁMÁNAK CSÖKKENTÉSE

- Mérési eredmények:

| | i7-2600 | Pentium 4 | Rasp. Pi | RK3188 |
|---|---|---|---|---|
| 1. példa, eredeti | 7,583 ms | 14,122 ms | 59,202 ms | 11,296 ms |
| 1. példa, elágazásmentes | 1,297 ms | 4,251 ms | 58,410 ms | 8,628 ms |
| 2. példa, eredeti | 8,211 ms | 19,6 ms | 73,267 ms | 21,496 ms |
| 2. példa, 1 elágazással | 7,942 ms | 14,295 ms | 61,578 ms | 13,347 ms |
| 2. példa, elágazásmentes | 1,203 ms | 4,252 ms | 58,328 ms | 8,268 ms |
| 3. példa, eredeti | 6,533 ms | 10,377 ms | 48,532 ms | 21,397 ms |
| 3. példa, segédtömbbel | 1,151 ms | 3,641 ms | 37,896 ms | 18,240 ms |

# AZ ELÁGAZÁSOK KISZÁMÍTHATÓBBÁ TÉTELE

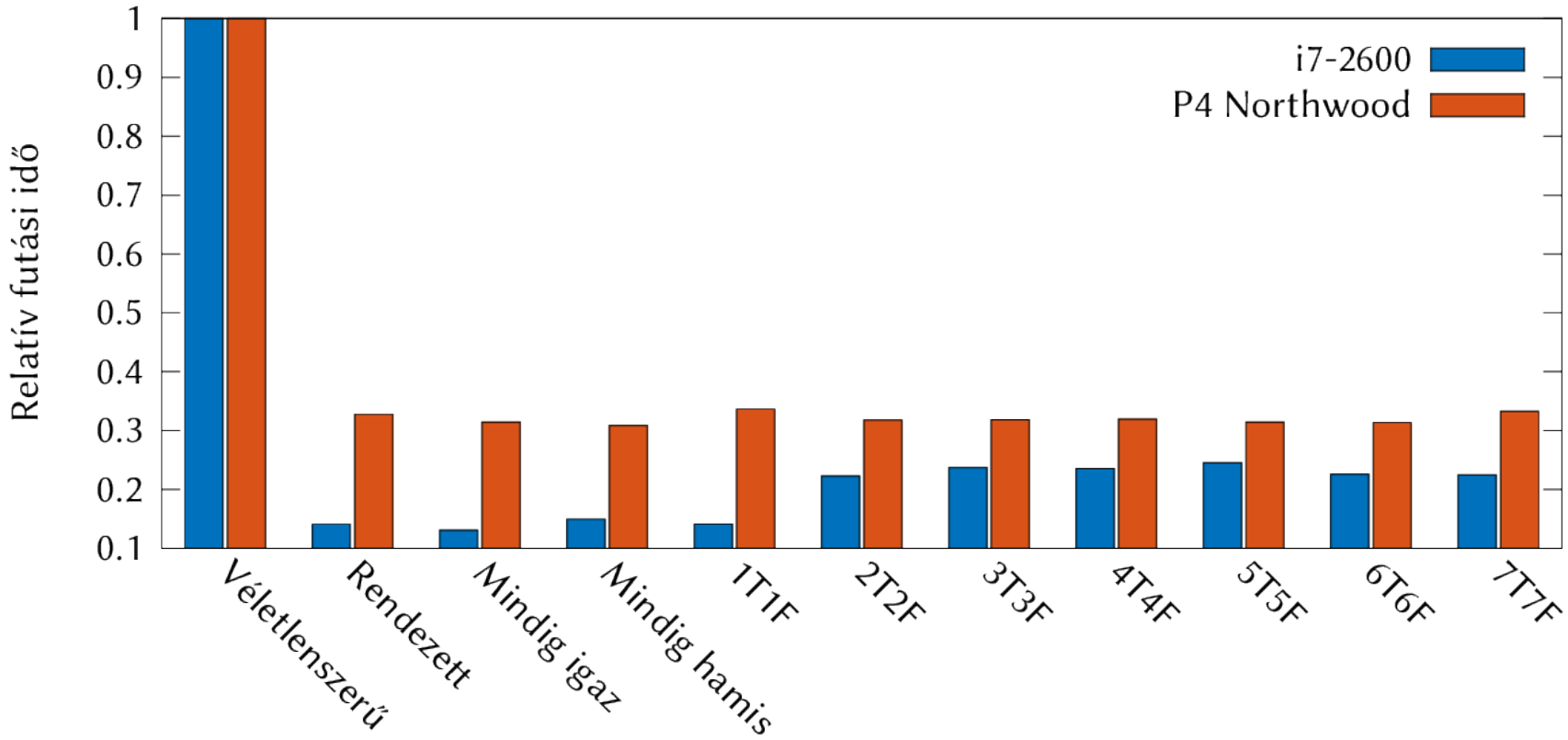
- Újra az 1. példa:

```
for (int i=0; i<N; i++)
    if (data[i] > 500)
        sum += data[i];
```

- Elemek rendezetlenek → kiszámíthatatlan kimenetel
- Rendezzük az elemeket!
- Mérések:
  - Rendezetlen eset
  - Rendezett tömb
  - Olyan tömb, amiben minden elemre teljesül
  - Olyan tömb, amiben egyetlen elemre sem teljesül
  - Adott teljesülési minta mellett

# AZ ELÁGAZÁSOK KISZÁMÍTHATÓBBÁ TÉTELE

- Mérési eredmények:



# AZ UGRÁSI CÍM BECSÜLHETŐSÉGÉNEK JAVÍTÁSA

- Példa: heterogén kollekció 16 leszármazottal

```
class A {
public:
    virtual int value ()=0;
    virtual int type () const =0;
    virtual ~A() {}
};
```

```
class B1 : public A {
  int v;
public:
  B1 () : v(rand()) {}
  int value () { return ++v; }
  int type () const { return 1; }
  ~B1 () {}
};
```

```
class B16 : public A {
  int v;
public:
  B16 () : v(rand()) {}
  int value () { return ++v; }
  int type () const { return 16; }
  ~B16 () {}
};
```

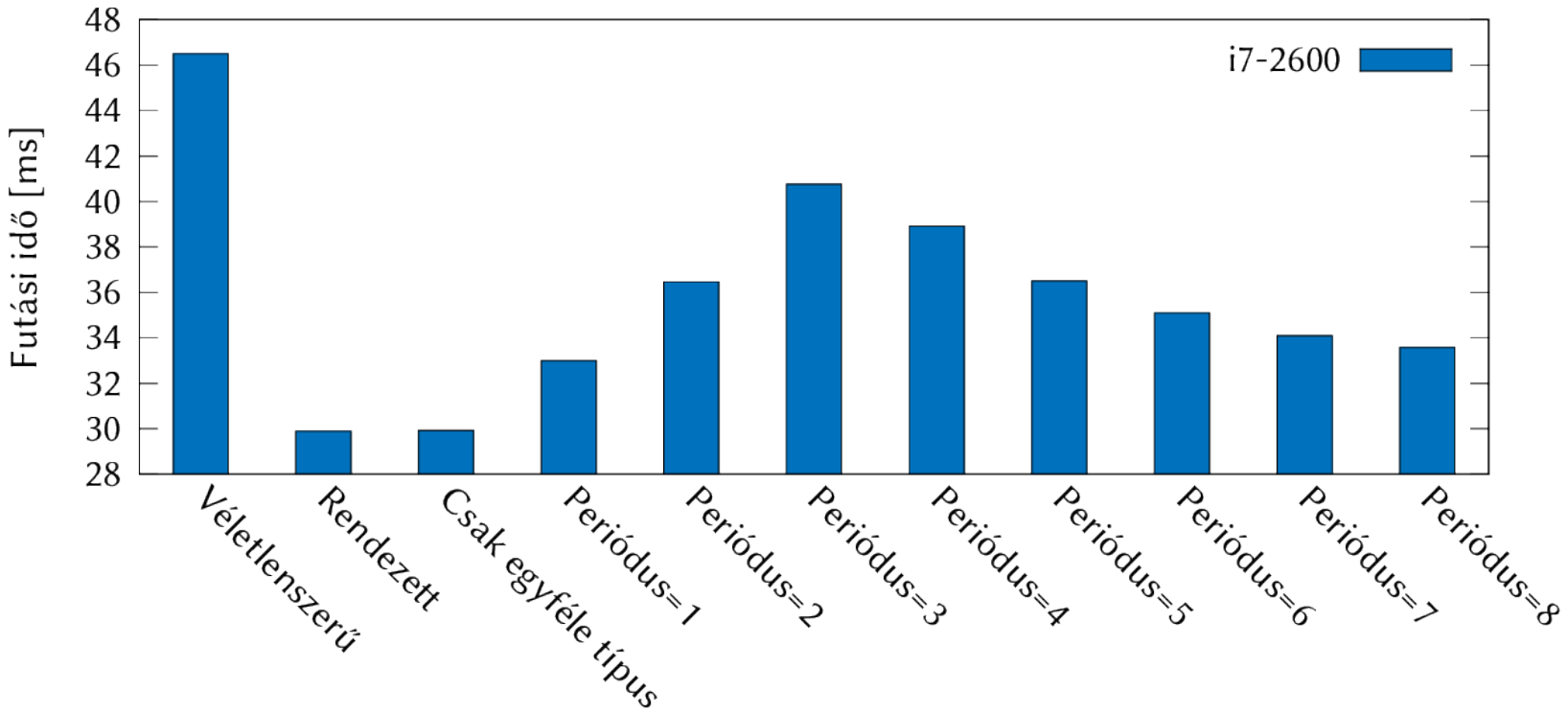
- Bejárás:

```
sum = 0;
for (i=0; i<siz; i++)
    sum += data[i]->value();
```

- Gond:
  - A virtuális függvényhívás **indirekt** ugrás
  - Az ugrási cím kiszámíthatatlan!
- Megoldás:
  - Rendezzük típus szerint a tömböt!

# AZ UGRÁSI CÍM BECSÜLHETŐSÉGÉNEK JAVÍTÁSA

- Mérési eredmények:



HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK

MŰEGYETEM 1782