

Számítógép Architektúrák

Cache memória

Mészáros András

BME Hálózati Rendszerek és Szolgáltatások Tsz.
meszarosa@hit.bme.hu

1. Feladat

Vegyünk egy 256 byte méretű cache-t 64 byte-os blokkmérettel. A cache kezdetben csupa érvénytelen blokkot tartalmaz. Egy program az alábbi memóriablokkokról olvas (ebben a sorrendben):

- 1, 3, 8, 4, 3, 6, 8, 1

Adja meg a cache hibák számát és a cache végső tartalmát (blokkszámokkal) LRU algoritmus mellett

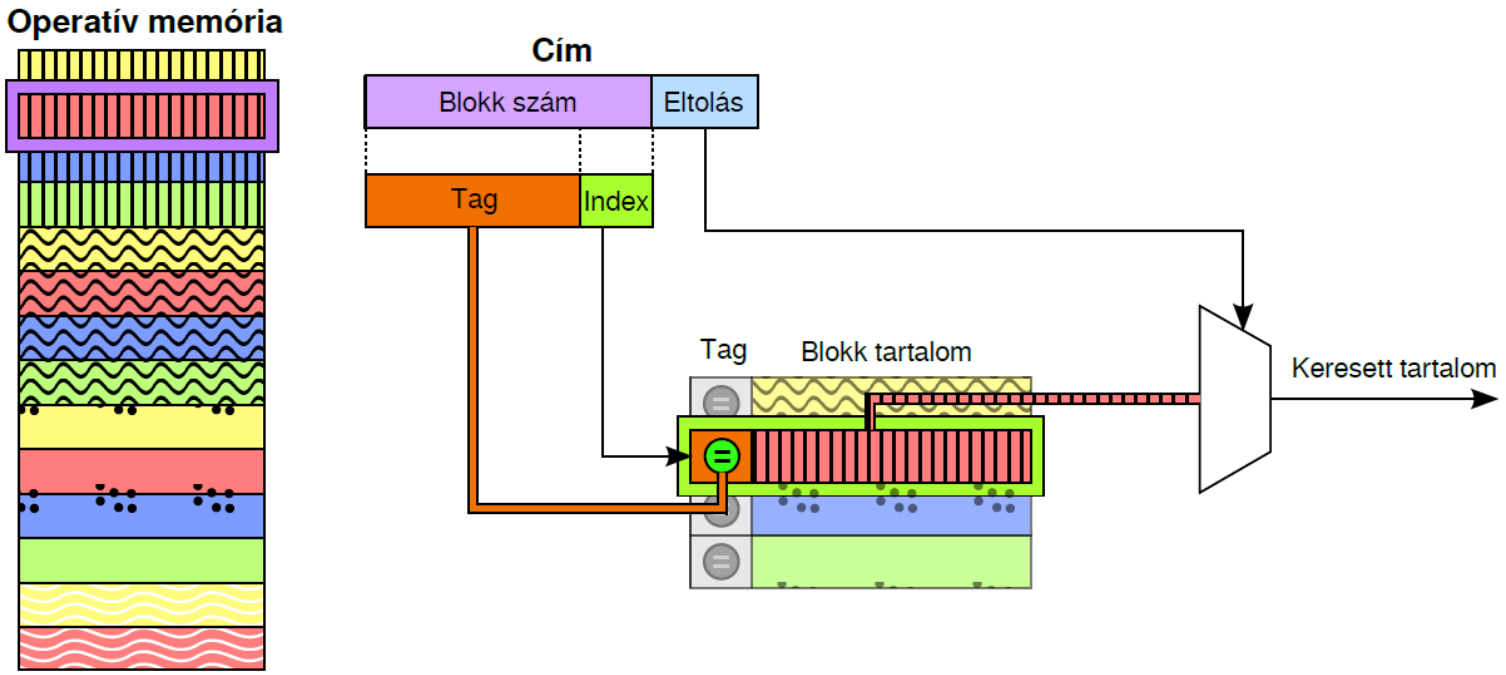
a) direkt leképezés esetén

b) teljesen asszociatív szervezés esetén

c) két utas asszociatív szervezés esetén

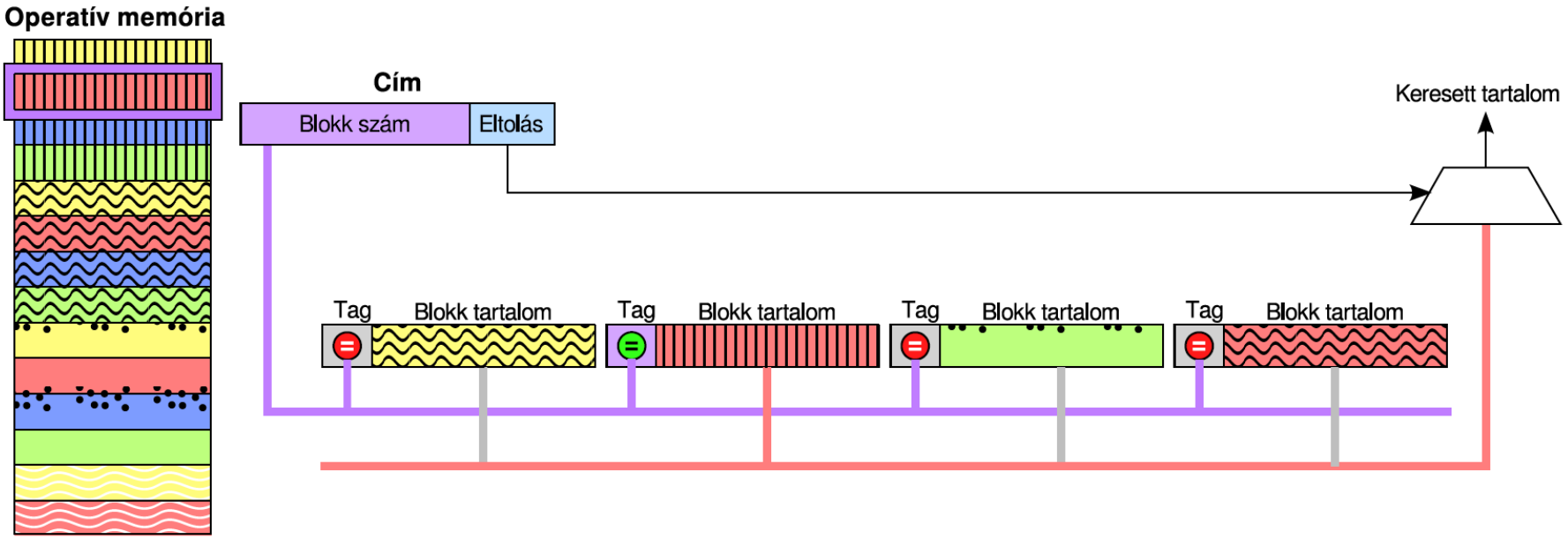
Direkt leképzés

- Minden blokknak egyetlen előre meghatározott helye van
- Olcsó, egyszerű, kis fogyasztású
- Új blokk gyakran kiszorít egy másikat
- Pl. modulo alapú elhelyezés



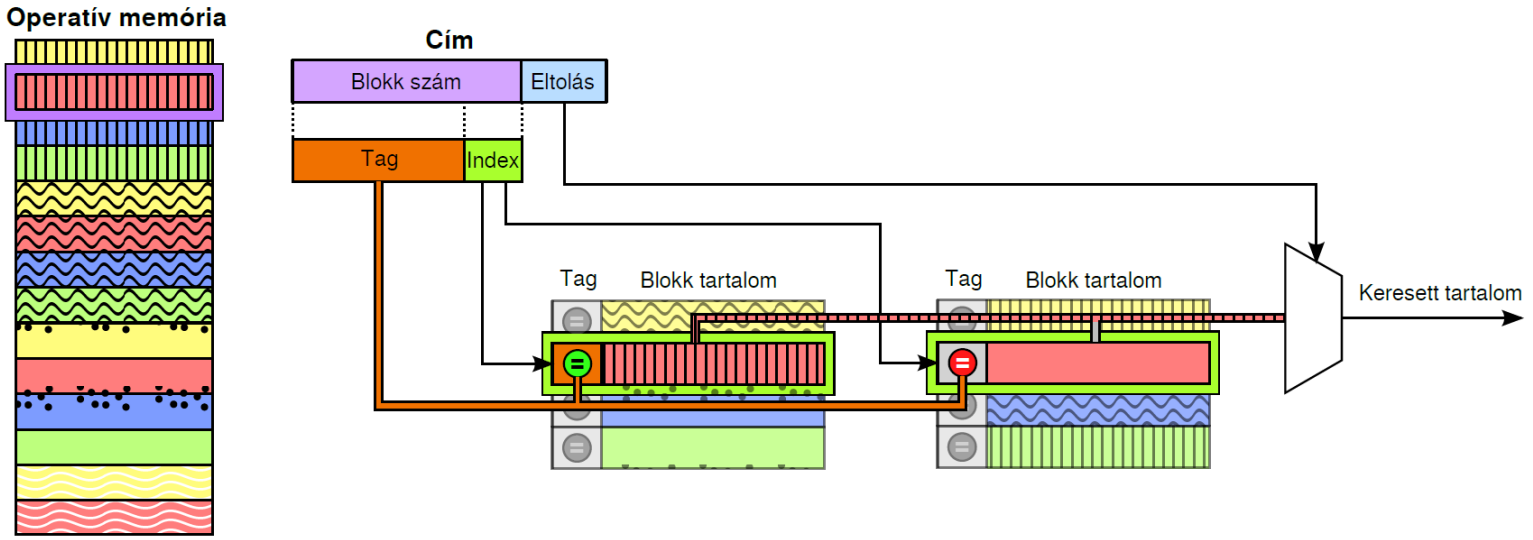
Teljesen asszociatív leképzés

- Bármelyik blokk bárhova kerülhet
- Csak akkor kell kidobni egy szabadon választott blokkot, ha tele a cache
- Sokat fogyaszt (minden blokkal komparálni kell, széles tag)



n-utas asszociatív leképzés

- Arany középút
- Van némi szabadságunk, mit dobjunk ki
- n blokkal kell csak komparálni



1. Feladat

Vegyünk egy 256 byte méretű cache-t 64 byte-os blokkmérettel. A cache kezdetben csupa érvénytelen blokkot tartalmaz. Egy program az alábbi memóriablokkokról olvas (ebben a sorrendben):

- 1, 3, 8, 4, 3, 6, 8, 1

Adja meg a cache hibák számát és a cache végső tartalmát (blokkszámokkal) LRU algoritmus mellett

a) direkt leképezés esetén

b) teljesen asszociatív szervezés esetén

c) két utas asszociatív szervezés esetén

S_{cache}= 256B

S_{blokk}=64B

N_{blokk}=

N_{blokk}= 4

Olvasott blokkok:
1, 3, 8, 4, 3, 6, 8, 1

1/a. Feladat

Adja meg a cache hibák számát és a cache végső tartalmát (blokkszámokkal) LRU algoritmus mellett

a) direkt leképezés esetén

Index = x mod N_{blokk} = x mod 4 (osztási maradék 4-gyel)

N_{blokk}= 4

Olvasott blokkok:
1, 3, 8, 4, 3, 6, 8, 1

	Valid	Blokk
0:	0	?
1:	0	?
2:	0	?
3:	0	?

Valid	Blokk
0	?
1	1
0	?
0	?

Valid	Blokk
1	8
1	1
1	6
1	3

1	3	8	4	3	6	8	1
?							
1							
?							
?							
				✓			✓

1/b. Feladat

Adja meg a cache hibák számát és a cache végső tartalmát (blokkszámokkal) LRU algoritmus mellett

b) teljesen asszociatív szervezés esetén

$N_{blokk} = 4$

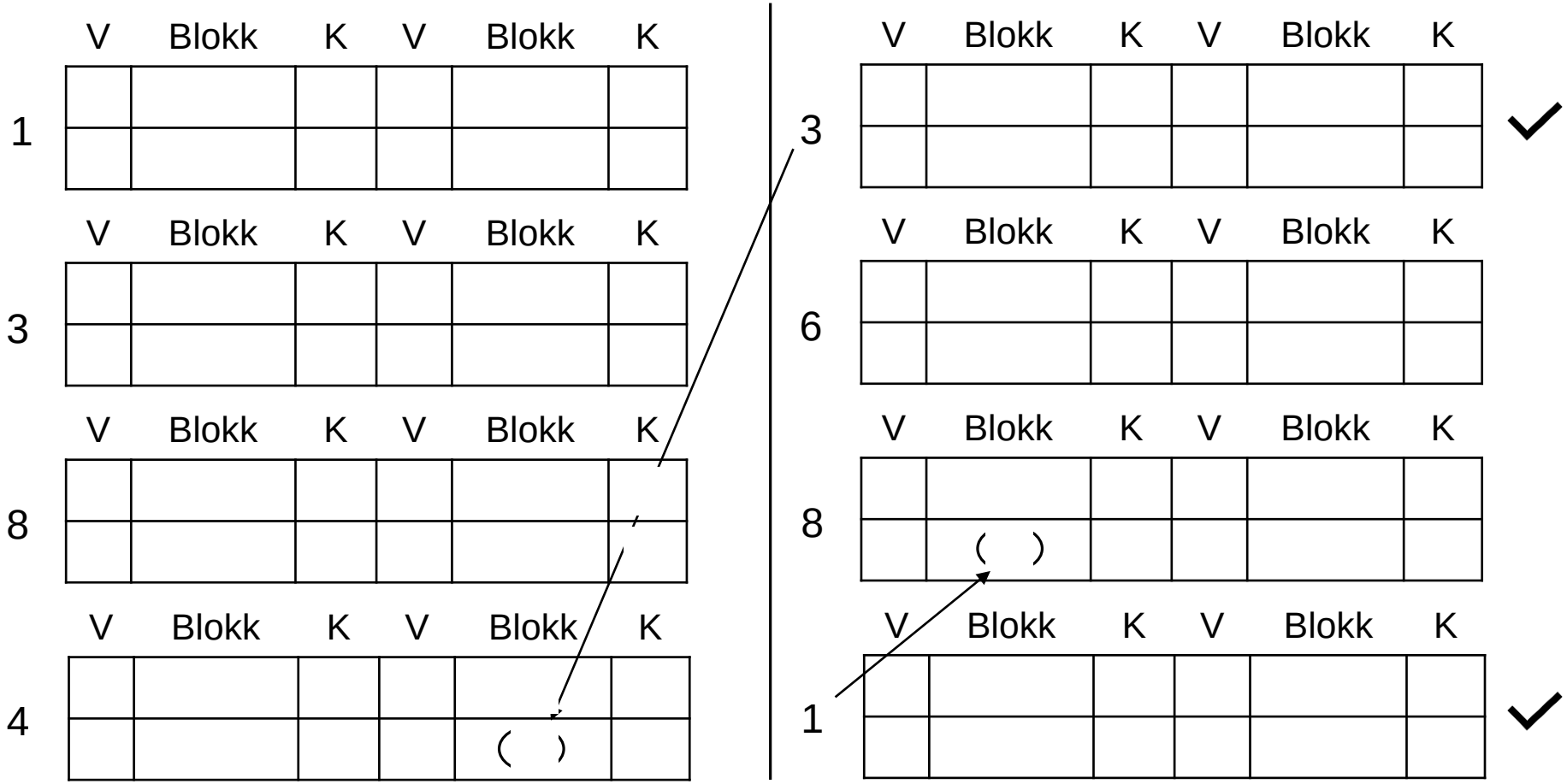
	V	Blokk	K	V	Blokk	K	V	Blokk	K	V	Blokk	K	
1	**1**	**1**	**1**	0	?	4	0	?	3	0	?	2	
3	1	1	**2**	**1**	**3**	**1**	0	?	**4**	0	?	**3**	
8	1	1	**3**	1	3	**2**	**1**	**8**	**1**	0	?	4	
4	1	1	**4**	1	3	**3**	1	8	**2**	**1**	**4**	**1**	
3	1	1	4	1	3	**1**	1	8	**3**	1	4	**2**	✓
6	**1**	**6**	**1**	1	3	**2**	1	8	**4**	1	4	**3**	
8	1	6	**2**	1	3	**3**	1	8	**1**	1	4	**4**	✓
1	1	6	**3**	1	3	**4**	1	8	**2**	**1**	**1**	**1**	

1/c. Feladat

Adja meg a cache hibák számát és a cache végső tartalmát (blokkszámokkal) LRU algoritmus mellett

$N_{blokk} = 4$

c) Két utas asszociatív szervezés esetén



2. Feladat

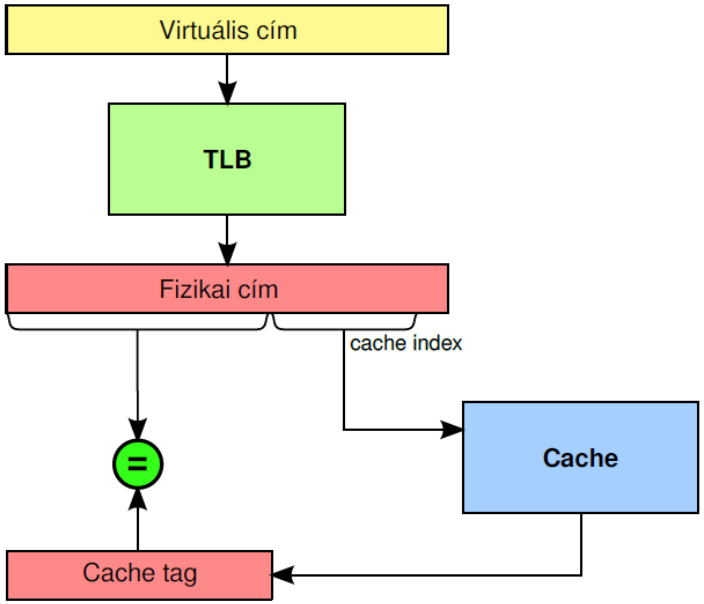
Vegyünk egy 32 kB méretű cache-t 64 byte-os blokkmérettel. A CPU rendelkezzen 32 bites fizikai, 48 bites virtuális címekkel. 4-utas asszociatív szervezés mellett hány darab és hány bites összehasonlítást kell végezni egy keresés során, ha

a) Fizikailag indexelt cache-t használunk, fizikai tag-ekkel

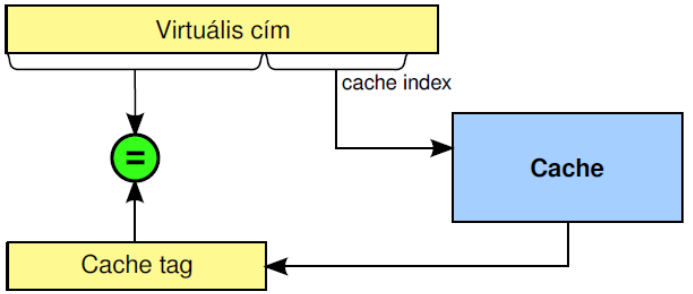
b) Virtuálisan indexelt cache-t használunk, virtuális tag-ekkel

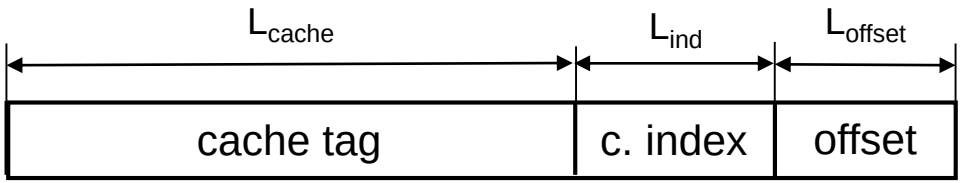
Fizikai/virtuális index/tag

Fizikai index és tag:



Virtuális index és tag:





2. Feladat

Vegyünk egy 32 kB méretű cache-t 64 byte-os blokkmérettel. A CPU rendelkezzen 32 bites fizikai, 48 bites virtuális címekkel. 4-utas asszociatív szervezés mellett hány darab és hány bites összehasonlítást kell végezni egy keresés során, ha

a) Fizikailag indexelt cache-t használunk, fizikai tag-ekkel

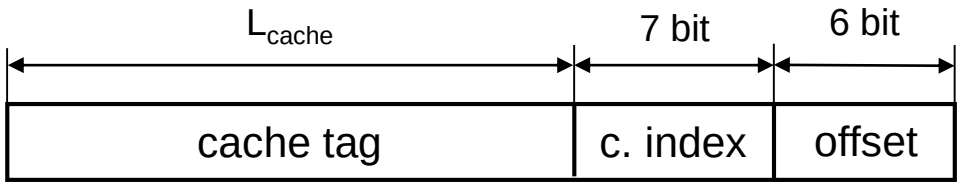
b) Virtuálisan indexelt cache-t használunk, virtuális tag-ekkel

$S_{cache} = 32kB$
$S_{blokk} = 64B$
$L_{fiz} = 32b$
$L_{virt} = 48b$
$L_{offset} = 6b$
$N_{blokk} = 512 = 2^9$
$N_{index} = 2^7$
$L_{ind} = 7b$

$S_{blokk} = 64B = 2^6 \rightarrow L_{offset} = 6b$

$N_{blokk} = S_{cache} / S_{blokk} = 512 = 2^9$

4 utas cache → minden indexhez 4 blokk:
$N_{index} = N_{blokk} / 4 = 2^7 \rightarrow L_{ind} = 7b$



2. Feladat megoldása

Hány darab és hány bites összehasonlítást kell végezni egy keresés során, ha

a) Fizikailag indexelt cache-t használunk, fizikai tag-ekkel

$L_{cache,fiz} = L_{fiz} - L_{offset} - L_{ind} = 32 - 6 - 7 = 19b$

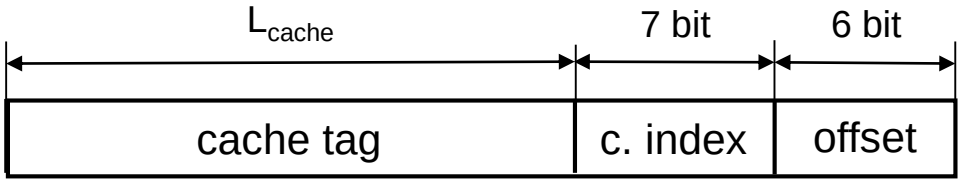
4 darab 19 bites összehasonlítás (4 utas asszociatív)

b) Virtuálisan indexelt cache-t használunk, virtuális tag-ekkel

$L_{cache,virt} = L_{virt} - L_{offset} - L_{ind} = 48 - 6 - 7 = 35b$

4 darab 35 bites összehasonlítás

$S_{cache} = 32kB$
$S_{blokk} = 64B$
$L_{fiz} = 32b$
$L_{virt} = 48b$
$N_{blokk} = 512 = 2^9$
$N_{index} = 2^7$
$L_{offset} = 6b$
$L_{ind} = 7b$



3. Feladat

Vegyünk egy 512 byte méretű cache-t 64 byte-os blokkmérettel. A cache fizikai címeket használ mind az indexeléshez, mind a tag-ekhez. A fizikai címek 16 bit szélesek. Egy program az alábbi memóriacímekről olvas (ebben a sorrendben):

• 13, 136, 490, 541, 670, 74, 581, 980

a) Adja meg a fenti címekhez tartozó ”tag”, ”index” és ”eltolás” mezőket

• teljesen asszociatív szervezés

• direkt leképzés

• két utas asszociatív szervezés

esetén.

b) Adja meg a cache végső tartalmát (blokkszámokkal) mind a 3 esetben, LRU algoritmus mellett! (A cache kezdetben csupa érvénytelen blokkot tartalmaz)

$S_{cache} = 512B$
$S_{blokk} = 64B$
$L_{fiz} = 16b$
$N_{blokk} = 8$
$L_{offset} = 6b$

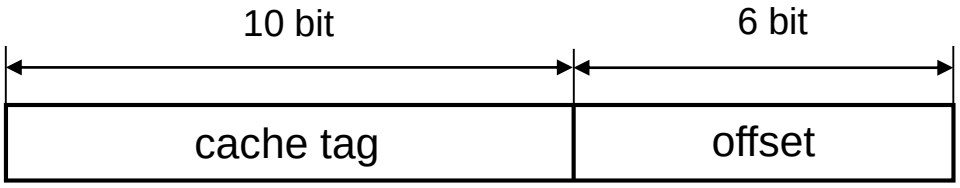
Kérések: 13, 136, 490, 541, 670, 74, 581, 980

$N_{blokk} = S_{cache} / S_{blokk} = 8$

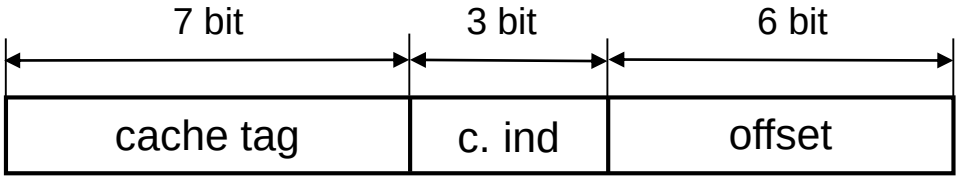
$S_{blokk} = 64B = 2^6 \rightarrow L_{offset} = 6b$

3. Feladat - címtagozódás

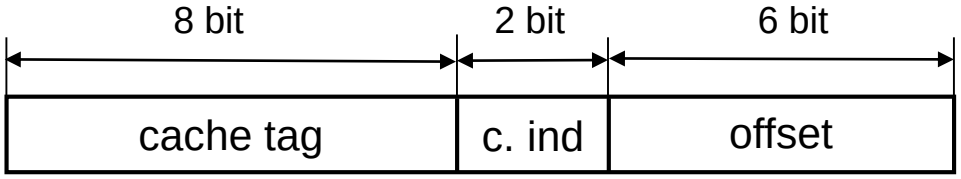
Teljesen asszociatív szervezés:



Direkt leképzés:



2-utas asszociatív szervezés:



$S_{cache} = 512B$
$S_{blokk} = 64B$
$L_{fiz} = 16b$
$N_{blokk} = 8$
$L_{offset} = 6b$

Kérések: 13, 136, 490, 541, 670, 74, 581, 980

3. Feladat– tag/index/offset számítás

Pl. direkt leképzés, 670-es kérés

Eltolás (offset): utolsó 6 bit

- 2^6 = 64-gyel való osztási maradék
- 670 % 64 (= 670 mod 64) = 30

Cache index: blokk szám utolsó 3 bit

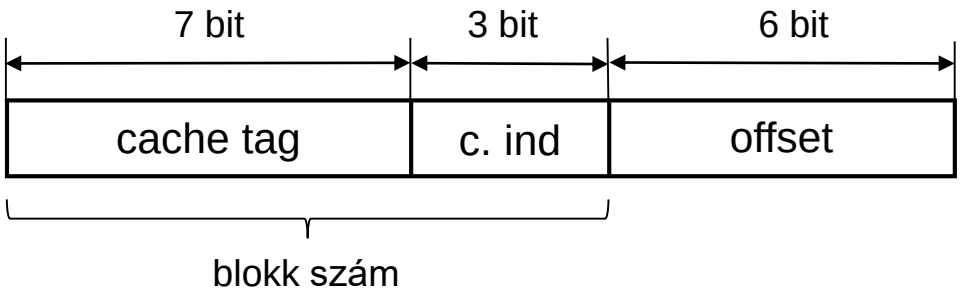
- 2^3 = 8-cal való osztási maradéka
- 10 % 8 = 2

Blokk szám: minden, kivéve utolsó 6 bit

- 2^6 = 64-gyel osztás egészrésze
- [670 / 64] = 10

Cache tag: blokk szám, kivéve 3 bit

- 2^3 = 8-cal osztás egészrésze
- [10 / 8] = 1



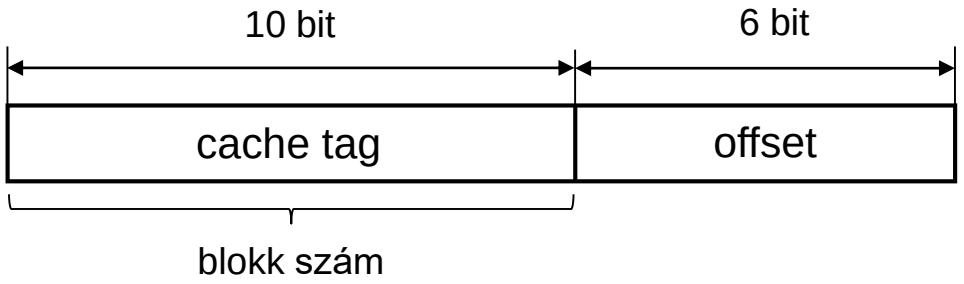
3/a. Feladat – asszociatív

a) Adja meg a fenti címekhez tartozó ”tag”, ”index” és ”eltolás” mezőket teljesen asszociatív szervezés esetén

	13	136	490	541	670	74	581	980
offset								
assz. tag								
dir. tag								
dir. ind.								
2-utas tag								
2-utas ind								

13 = 0 * 64 + 13
136 = 2 * 64 + 8
490 = 7 * 64 + 42
541 = 8 * 64 + 19
670 = 10 * 64 + 30
74 = 1 * 64 + 10
581 = 9 * 64 + 30
980 = 15 * 64 + 20

offset = x % 64 (x mod 64)
assz. tag = [x / 64]
blokk szám = assz. tag



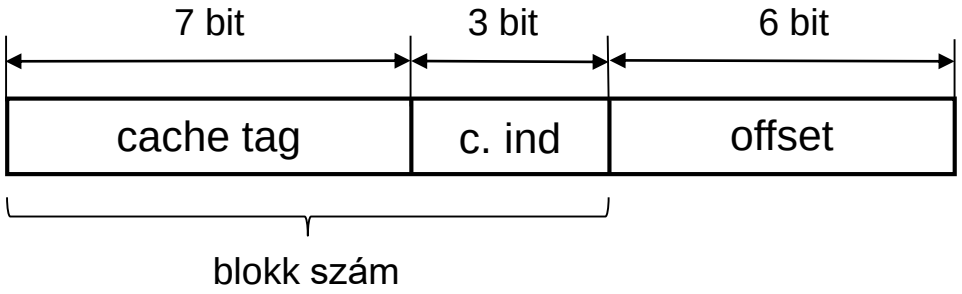
3/a. Feladat – direkt

a) Adja meg a fenti címekhez tartozó ”tag”, ”index” és ”eltolás” mezőket direkt leképzés esetén

	13	**136**	**490**	**541**	**670**	**74**	**581**	**980**
offset	13	8	42	19	30	10	5	20
assz. tag	0	2	7	8	10	1	9	15
dir. tag								
dir. ind.								
2-utas tag								
2-utas ind								

13 = 0 * 64 + 13
136 = 2 * 64 + 8
490 = 7 * 64 + 42
541 = 8 * 64 + 19
670 = 10 * 64 + 30
74 = 1 * 64 + 10
581 = 9 * 64 + 30
980 = 15 * 64 + 20

direkt tag = [blokk szám / 8]
direkt ind.= blokk szám % 8



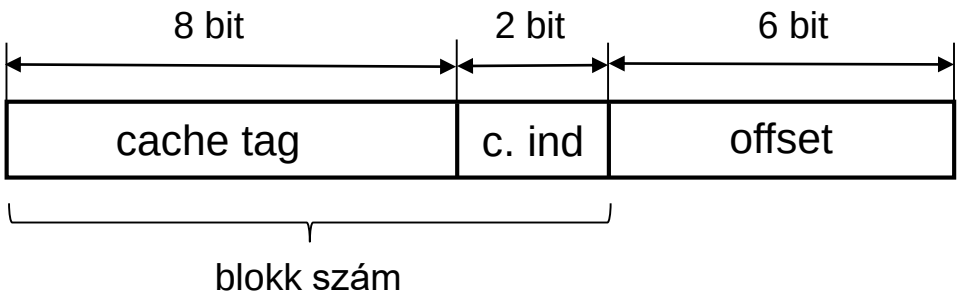
3/a. Feladat – 2 utas

a) Adja meg a fenti címekhez tartozó ”tag”, ”index” és ”eltolás” mezőket 2-utas asszociatív szervezés esetén

	13	**136**	**490**	**541**	**670**	**74**	**581**	**980**
offset	13	8	42	19	30	10	5	20
assz. tag	0	2	7	8	10	1	9	15
dir. tag	0	0	0	1	1	0	1	1
dir. ind.	0	2	7	0	2	1	1	7
2-utas tag								
2-utas ind								

13 = 0 * 64 + 13
136 = 2 * 64 + 8
490 = 7 * 64 + 42
541 = 8 * 64 + 19
670 = 10 * 64 + 30
74 = 1 * 64 + 10
581 = 9 * 64 + 5
980 = 15 * 64 + 20

2-utas tag = [blokk szám / 4]
2-utas ind.= blokk szám % 4



3/b. Feladat – asszociatív

b) Adja meg a cache végső tartalmát (blokkszámokkal) mind a 3 esetben, LRU algoritmus mellett!

	13	**136**	**490**	**541**	**670**	**74**	**581**	**980**
assz. tag	0	2	7	8	10	1	9	15

Teljesen asszociatív:

8 blokk, pont 8 kérés → minden cache blokkba épp kerül valami

V	Tag	Tartalom
1	0	0. blokk

V	Tag	Tartalom
1	2	2. blokk

…

V	Tag	Tartalom
1	15	15. blokk

3/b. Feladat – direkt

b) Adja meg a cache végső tartalmát (blokkszámokkal) mind a 3 esetben, LRU algoritmus mellett!

	13	**136**	**490**	**541**	**670**	**74**	**581**	**980**
assz. tag	0	2	7	8	10	1	9	15
dir. tag	0	0	0	1	1	0	1	1
dir. ind.	0	2	7	0	2	1	1	7

	V	Tag	Tartalom
0:			
1:			
2:			
3:			
4:			
5:			
6:			
7:			

3/b. Feladat – 2 utas

b) Adja meg a cache végső tartalmát (blokkszámokkal) mind a 3 esetben, LRU algoritmus mellett!

	13	**136**	**490**	**541**	**670**	**74**	**581**	**980**
assz. tag	0	2	7	8	10	1	9	15
2-utas tag	0	0	1	2	2	0	2	3
2-utas ind	0	2	3	0	2	1	1	3

	V	Tag	Tartalom
0:			
1:			
2:			
3:			

V	Tag	Tartalom

4. Feladat

Vegyünk egy 1 kB méretű cache-t 64 byte-os blokkmérettel. A cache kezdetben csupa érvénytelen blokkot tartalmaz.

Lefuttatjuk az alábbi programot:

S_{cache} = 1kB
S_{blokk} = 64B
S_{int} = 2B

```
short int t[32][32];
int sum = 0;
for (int i=0; i<32; i++)
    for (int j=0; j<32; j++)
        sum += t[i][j];
```

Feltételezések: a short int 2 byte-os, a t tömb blokkhatáron kezdődik, a két dimenziós tömb a memóriában sor-folytonosan helyezkedik el, a cache direkt leképzést használ. Az i,j változó regiszterekben van tárolva, a cache-t nem terhelik.

a) Hány cache hibát vált ki a fenti algoritmus? Számolja ki a cache hiba-arányt!

b) Hány cache hibát vált ki a fenti algoritmus, ha megcseréljük a két ciklust? Számolja ki a cache hiba-arányt!

c) Mekkora cache-re lenne szükség, hogy a megcserélt for ciklusokkal is ugyanolyan hibaarányt kapjunk, mint az eredeti algoritmussal?

4/a. Feladat

a) Hány cache hibát vált ki a fenti algoritmus? Számolja ki a cache hiba-arányt!

S_{cache} = 1kB
S_{blokk} = 64B
S_{int} = 2B

```
short int t[32][32];
int sum = 0;
for (int i=0; i<32; i++)
    for (int j=0; j<32; j++)
        sum += t[i][j];
```

Pont egy t[i][0:31] sor fér bele egy cache blokkba (S_{blokk} / S_{int} = 32).

N_{blokk} = S_{blokk} / S_{cache} = 16 blokk fér a cache-be

```
sum += t[0][0]; // t[0][0:31] behozása
sum += t[0][1];
…
sum += t[1][0]; // t[1][0:31] behozása
sum += t[1][1];
…
sum += t[31][0]; // t[31][0:31] behozása
…
```

4/a. Feladat

a) Hány cache hibát vált ki a fenti algoritmus? Számolja ki a cache hiba-arányt!

```
sum += t[0][0]; // t[0][0:31] behozása
sum += t[0][1];
…
sum += t[1][0]; // t[1][0:31] behozása
sum += t[1][1];
…
sum += t[31][0]; // t[31][0:31] behozása
…
```

A `t[i][0]` hozzáadása mindig cache hibával jár a többi művelet nem.

→ 32 sor = 32 cache hiba

→ A cache hiba-arány 1/32 = 3,125%

4/b. Feladat

b) Hány cache hibát vált ki a fenti algoritmus, ha megcseréljük a két ciklust? Számolja ki a cache hiba-arányt!

```
    short int t[32][32];
    int sum = 0;
    for (int j=0; j<32; j++)
        for (int i=0; i<32; i++)
            sum += t[i][j];

sum += t[0][0]; // t[0][0:31] behoz
sum += t[1][0]; // t[1][0:31] behoz
…
sum += t[15][0]; // t[15][0:31] behoz – cache megtelt
sum += t[16][0]; // t[16][0:31] be - t[0][0:31] ki
…
sum += t[31][0]; // t[31][0:31] be - t[15][0:31] ki
sum += t[0][1]; //t[0][0:31] már nincs benn (lásd ábra)
```

$N_{blokk} = 16$

0:	t[16][0:31]
1:	t[17][0:31]
2:	t[18][0:31]
3:	t[19][0:31]
4:	t[20][0:31]
5:	t[21][0:31]
6:	t[22][0:31]
7:	t[23][0:31]
8:	t[24][0:31]
9:	t[25][0:31]
10:	t[26][0:31]
11:	t[27][0:31]
12:	t[28][0:31]
13:	t[29][0:31]
14:	t[30][0:31]
15:	t[31][0:31]

4/b. Feladat

b) Hány cache hibát vált ki a fenti algoritmus, ha megcseréljük a két ciklust? Számolja ki a cache hiba-arányt!

```
short int t[32][32];
int sum = 0;
for (int j=0; j<32; j++)
    for (int i=0; i<32; i++)
        sum += t[i][j];
```

Minden egyes összeadás cache hibát okoz

→ 32 * 32 = 1024 cache hiba

→ 100% hibaarány

4/c. Feladat

c) Mekkora cache-re lenne szükség, hogy a megcserélt for ciklusokkal is ugyanolyan hibaarányt kapjunk, mint az eredeti algoritmussal?

S_{cache} = 1kB
S_{blokk} = 64B
S_{int} = 2B

Ugyanolyan hibaarány, ha csak egyszer kell behozni minden cache blokkot (ez a minimum)

→ A teljes tömb befér a cache-be

A tömb mérete:

32 * 32 * S_{int} = 2kB, ekkora cache-re lenne szükség