Šta smo do sada naučili?

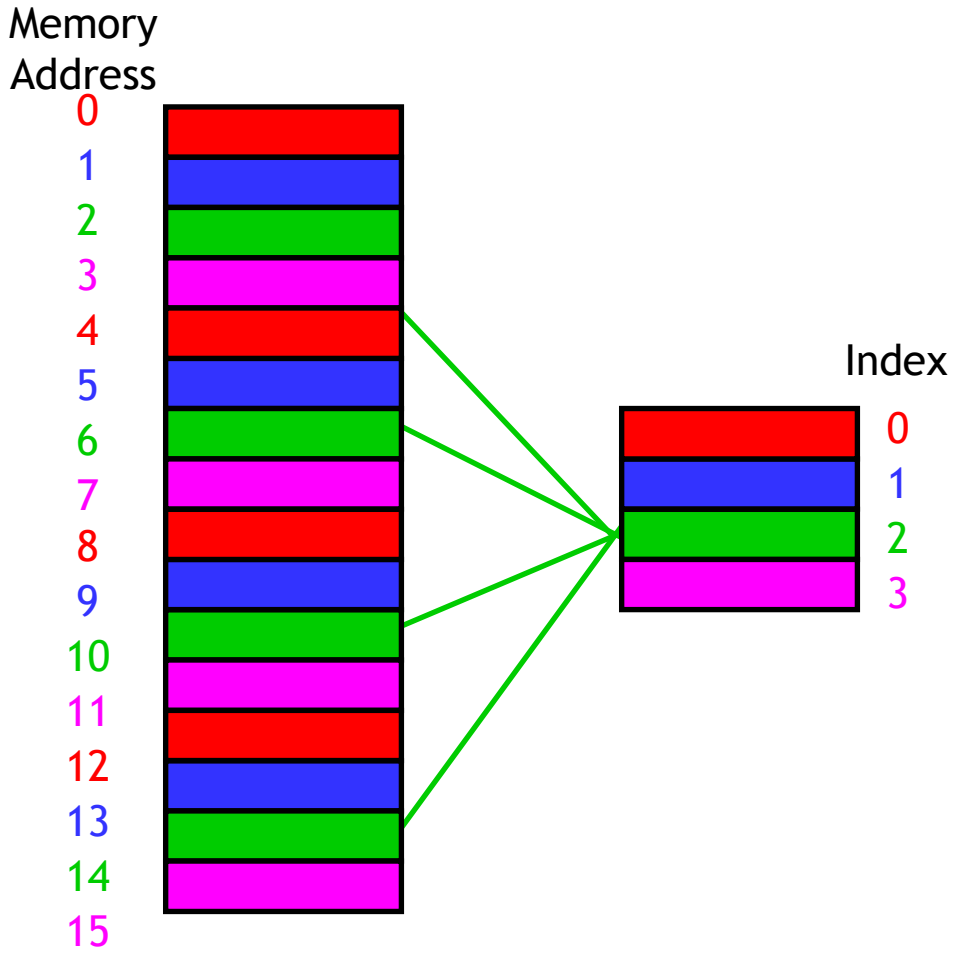
Kako nalazimo podatak u kešu ?

- Pitanje je da li se podatak koji trazimo nalazi u kešu?
- Ako zelimo da citamo

memorijsku adresu *i*,

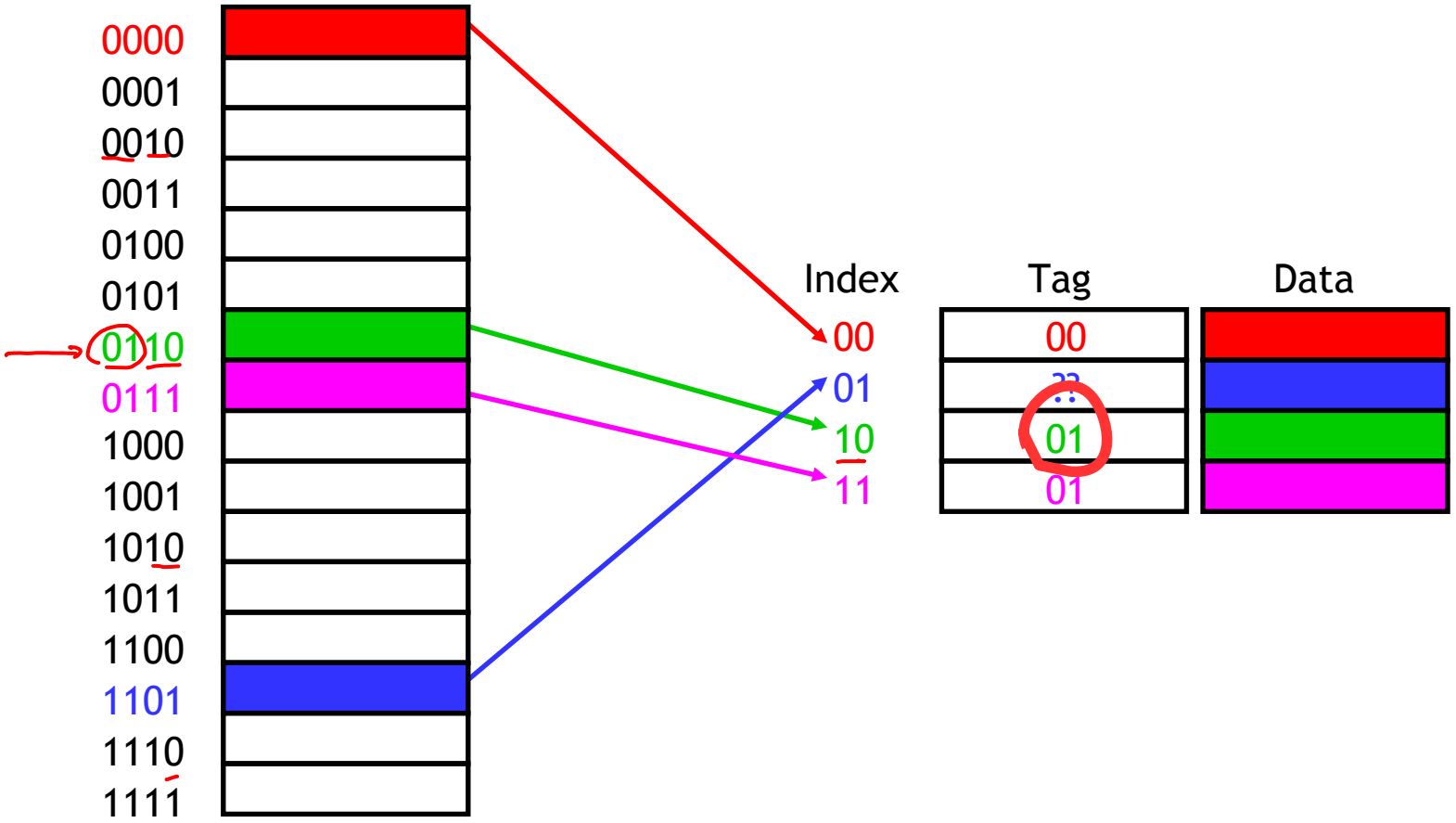
Mozemo da koristimo
mod trick da odredimo koji kes blok sadrzi *i*.

- Ali I podaci sa drugih adresa takodje su mapirani u istom kes bloku. Kako da ih razlikujemo?
- Na primer, kes block 2 moze da sadrzi podatke sa adresa 2, 6, 10 *ili* 14.



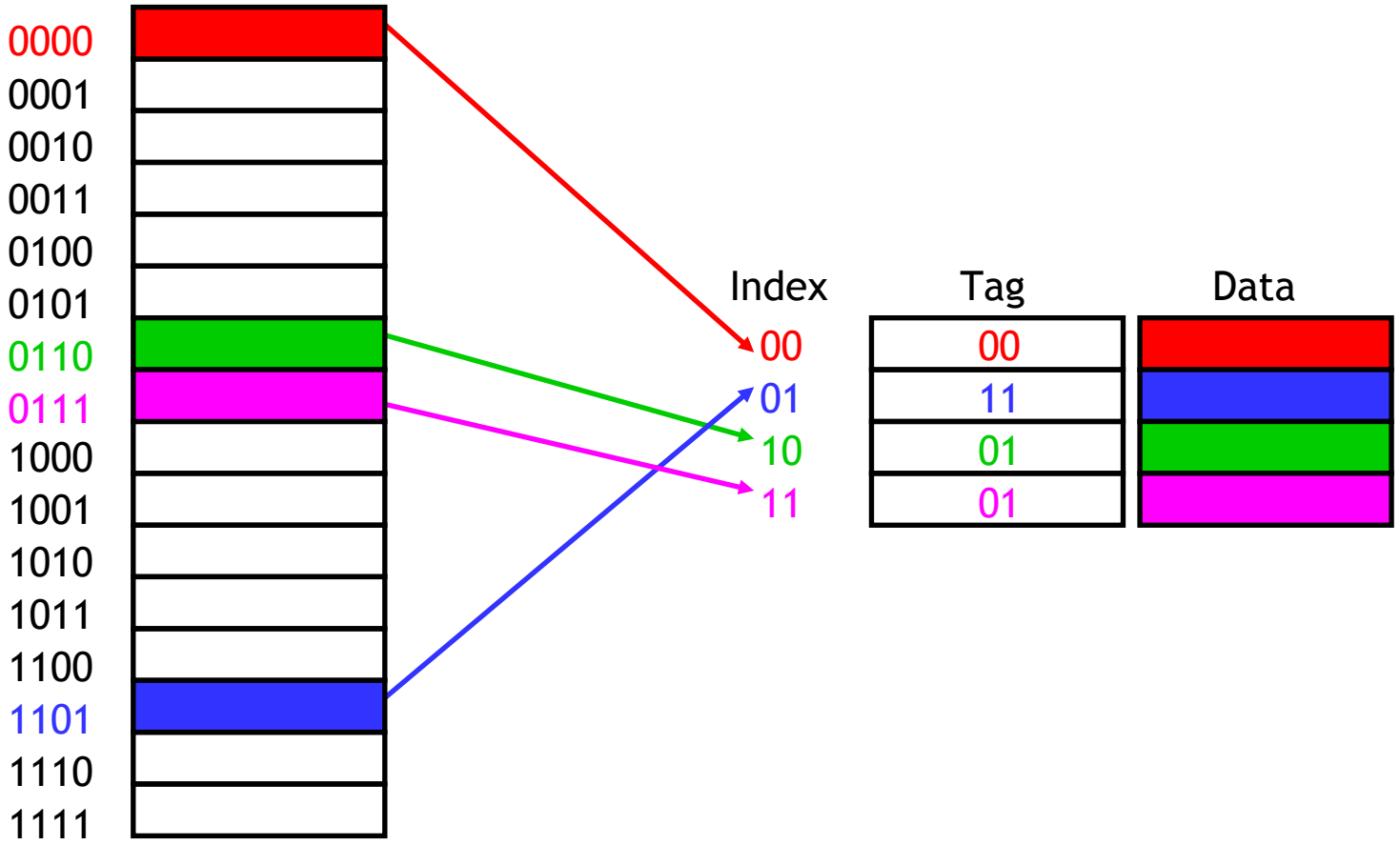
Dodavanje tagova

Resenje je dodavanja tags u memorijski prosto kesa, pomocu kojih pravimo razliku izmedju razlicitih memorijskih lokacija mapiranih u isti kes blok.



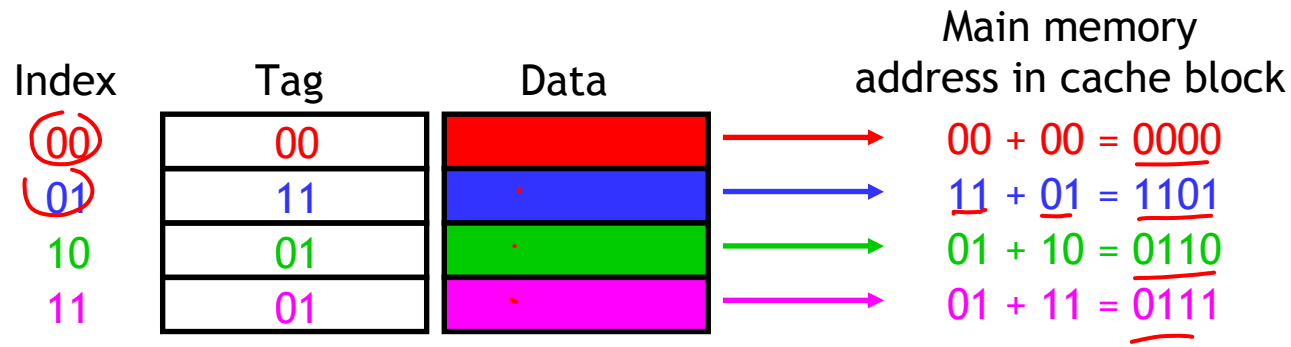
Adding tags

- We need to add tags to the cache, which supply the rest of the address bits to let us distinguish between different memory locations that map to the same cache block.



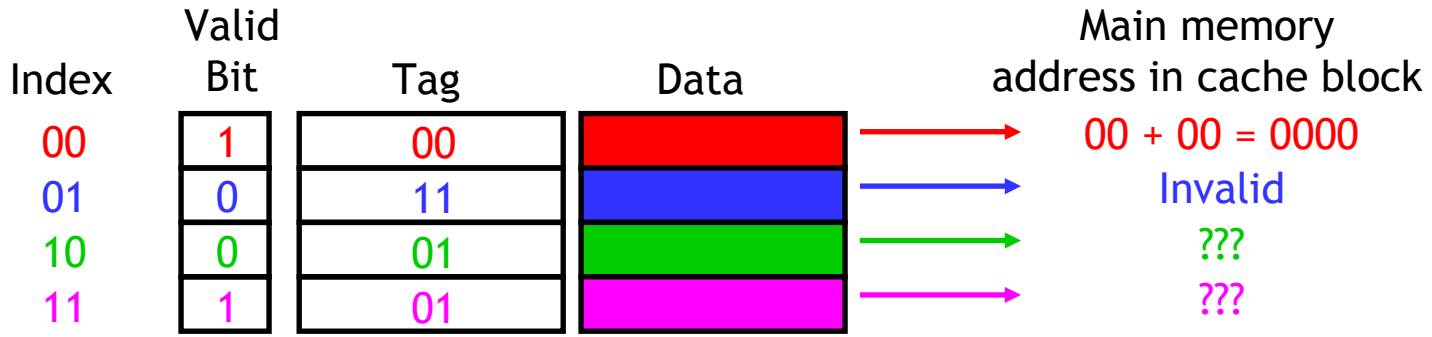
Sta je u kesu?

Sada možemo tačno da kazemo koje adrese glavne memorije su sacuvane u cache, nadovezivanjem cache blok tagove oznake sa indeksima.



valid bit

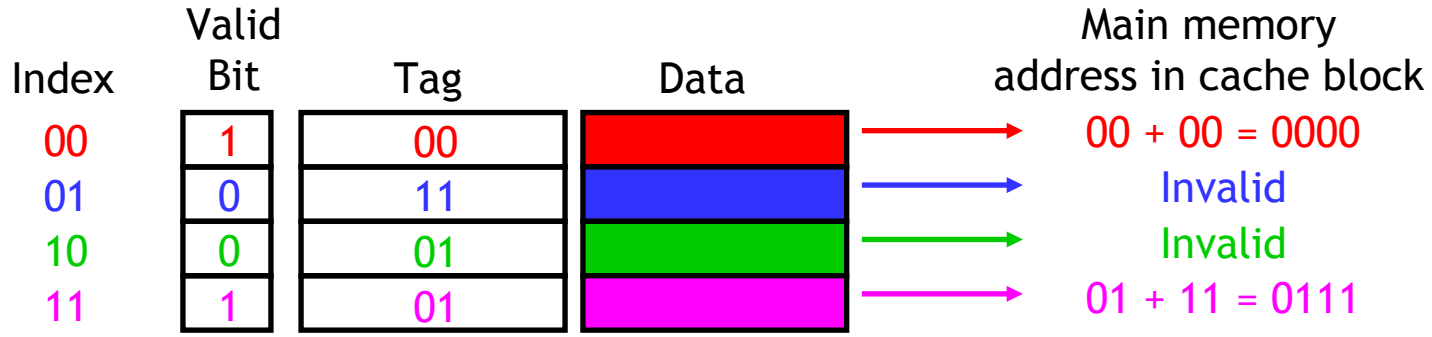
- Na pocetku kes je prazan I ne sadrzi validne podatke.
- Da li se u kesu nalaze validni podaci odredjujemo dodavanjem valid bit za svaki cache blok.
  - Na pocetku inicijalizacije, svi valid bitovi su setovani na 0.
  - Kada se podatak ucita u blok kes memorije, odgovarajuci bit validnosti se setuje na 1.



- Dakle cache sadrzi vise od proste kopije podataka u memoriji, takodje ima I dodatne bitove koji nam pomazu da pronadjemo podatke u kesu I da izvrsimo validaciju podataka.

valid bit

- Na pocetku kes je prazan I ne sadrzi validne podatke.
- Da li se u kesu nalaze validni podaci odredjujemo dodavanjem valid bit za svaki cache blok.
  - Na pocetku inicijalizacije, svi valid bitovi su setovani na 0.
  - Kada se podatak ucita u blok kes memorije, odgovarajuci bit validnosti se setuje na 1.



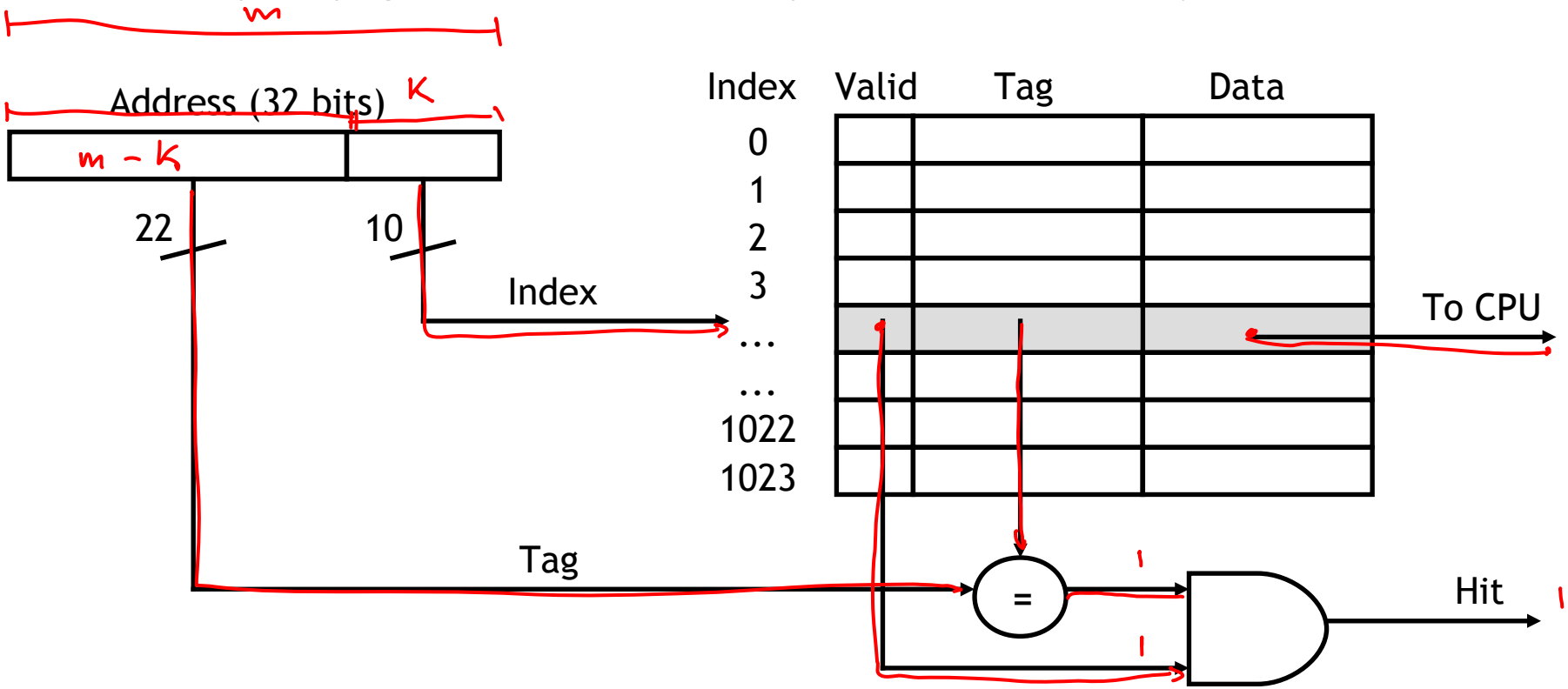
- Dakle cache sadrzi vise od proste kopije podataka u memoriji, takodje ima I dodatne bitove koji nam pomazu da pronadjemo podatke u kesu I da izvrsimo validaciju podataka.

Sta se dogadja kada pogodimo kes?

Kada CPU pokušava da pročita iz memorije, adresa će biti poslana na cache controller.

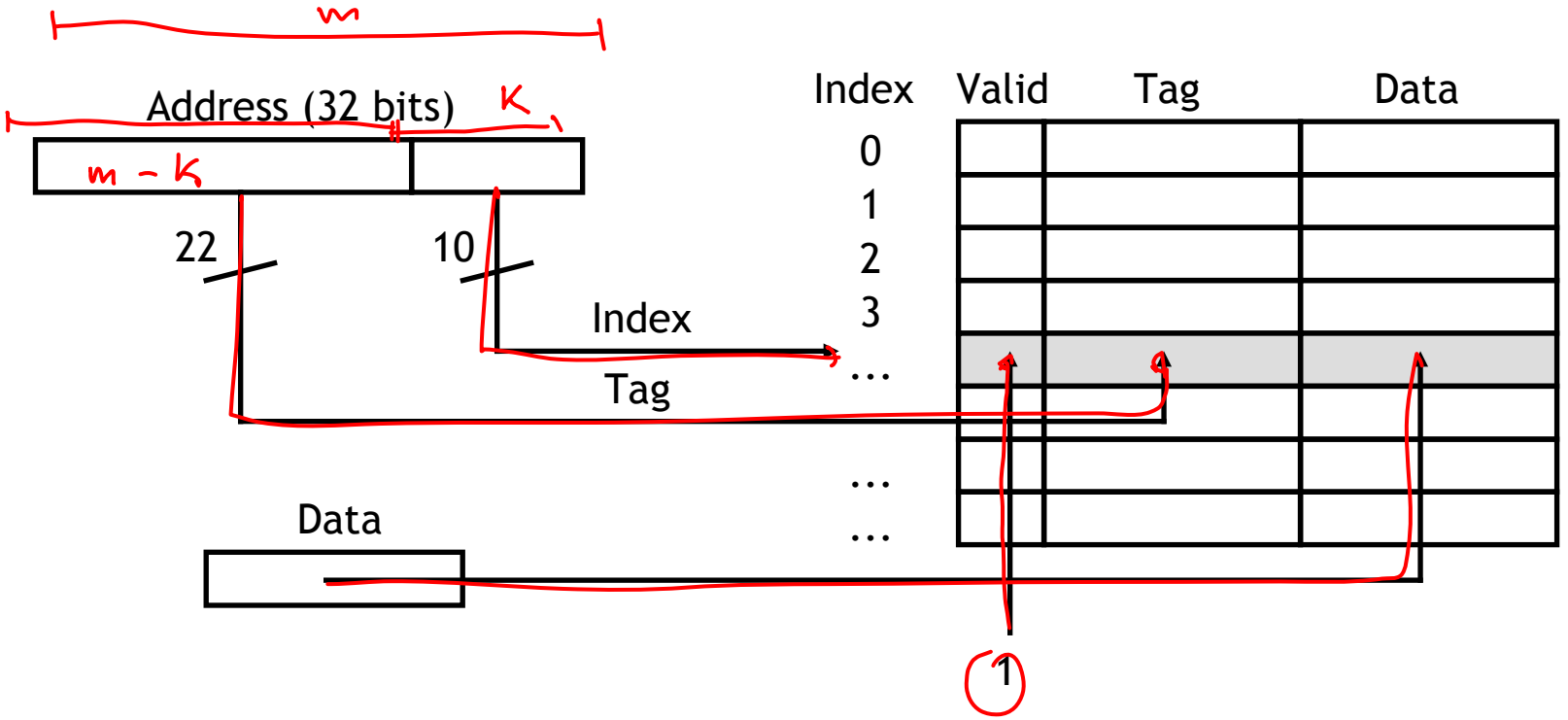
- Nizih *k* bitova adrese indeksiraju blok u kesu.
- Ako je blok validan, onda tag treba da se poklapa sa gornjih (*m* – *k*) bitova *m*-bitne adrese I podatak se salje u CPU.

- Prikazan je dijagram 32-bit memorijske adrese I 2^{10}-byte cache.



Upisivanje bloka u kes kada se promasi?

- Kada se podatak procita iz glavne memorije stavljanje podatka I na kes je jednostavno.
  - Nizih *k* bitova adrese indeksiraju cache block.
  - Gornjih (*m* – *k*) bitova adrese cuvaju se u tag polju kesa.
  - Podatak iz glavne memorije se cuva u polju podataka kesa.
  - Valid bit je setovan na 1.



Kolika je velicina kesa?

Za bajt-adresabilni racunar sa 16-bitnom adresom I kesom sa sledećim karakteristikama:

- direct-mapped
- Svaki blok sadrzi jedan bajt
- Kes indeks su cetiri bitova najmanje tezine

Dva pitanja:

- Od koliko blokova se sastoji kes?
-
- 2^4=16 blokova
- Koliko bitova je potrebno za izgranju kesa?

Velicina taga = 12 bitova (16 bit adresa - 4 bit index)

(12 tag bit + 1 valid bit + 8 data bits) x 16 blocks = 21 bits x 16 = 336 bits

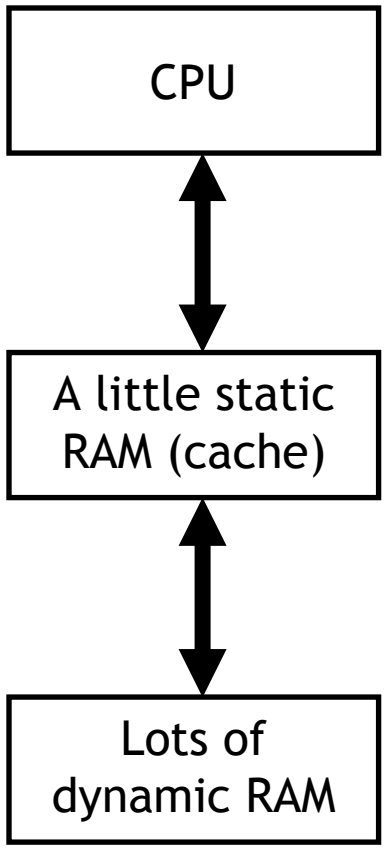
Performanse memorijskog sistema

Da bi se ispitala performansa memorijskog sistema, treba povesti racuna 0 nekoliko važnih faktora.

- Koliko dugo traje prenos podataka iz kesa u CPU?
- Koliko dugo traje kopiranje podataka iz memorije u kes?
- Koliko cesto pristupamo glavnoj memoriji?

Postoje i imena za svaki od navedenih faktora.

- hit time - koliko dugo traje prenos podataka iz kesa u CPU. Ovo je obično brzo, reda veličine 1-3 ciklusa takta.
- miss penalty -koliko dugo traje kopiranje podataka iz glavne memorije u kes. To često zahteva nekoliko desetina ciklusa takta.
- miss rate je procenat promašaja.



Srednje vreme pristupa memoriji

- Srednje vreme pristupa memoriji (average memory access time AMAT), se racuna kao.

AMAT = Hit time + (Miss rate x Miss penalty)

Ovo je samo prosečno vreme za pogodak na kesu i vreme za promasaj na kesu.

Kako možemo poboljšati prosečno vreme pristupa memori?

- Jasno,nizi AMAT je bolji.
- Miss penalties je obicno mnogo veci od hit times, tako da je bolji nacin za nizi AMAT smanjenje miss penalty ili miss rate.

Međutim, AMAT treba koristiti samo kao opstu smernicu.

Zapamtimo da je execution time najbolji parametar za merenje perfomanse.

Primer:

Pretpostavimo da je 33% instrukcija u programu pristup podacima.

Cache hit (pogodak na kesu) je 97% i hit time je jedan ciklus, ali miss penalty je 20 ciklusa

AMAT = Hit time + (Miss rate x Miss penalty)
= 1 cycle + (3% x 20 cycles)
= 1.6 cycles

- Za savrsen kes, bez promasaja, AMAT bi trebalo da je jedan ciklus.
- Za samo 3% promasaja AMAT se povecava na 1.6 puta

- Kako da se smanji miss rate, procenat promasaja?

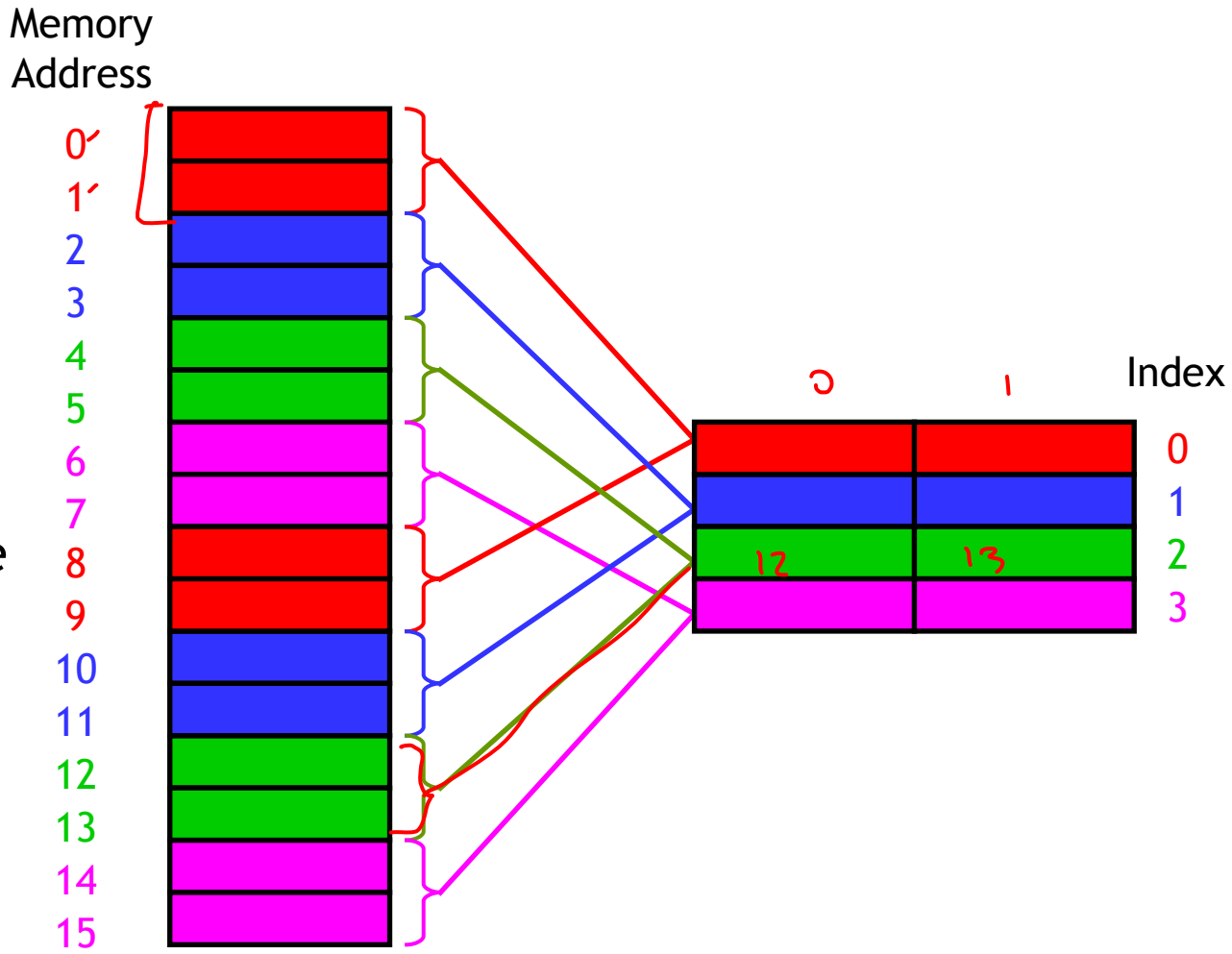
Spatial locality

- Jedno bajtni kes blok ne moze da iskoristi prednosti prostorne lokalnosti(spatial locality), koja pretpostavlja da pristup jednoj adresi prati pristup sledecim bliskim adresama.
-

Spatial locality

- Ono sto moze da se uradi je povecanje velicine kes bloka na vise od jednog bajta.

- Sada imamo dva Bloka tako da mozemo da upisemo u kes dva bajta u isto vreme.
- Ako citamo sa adrese 12, podaci na adresama 12 *i* 13 bice kopirane u blok 2 kesa.



Adrese blokova

- Ako je cache blok velicine is 2^n bytes, konceptualno mozemo da podelimo I glavnu memoriju u 2^n-byte komada takodje.
- Da bi smo odredili adresu bloka podatka sa

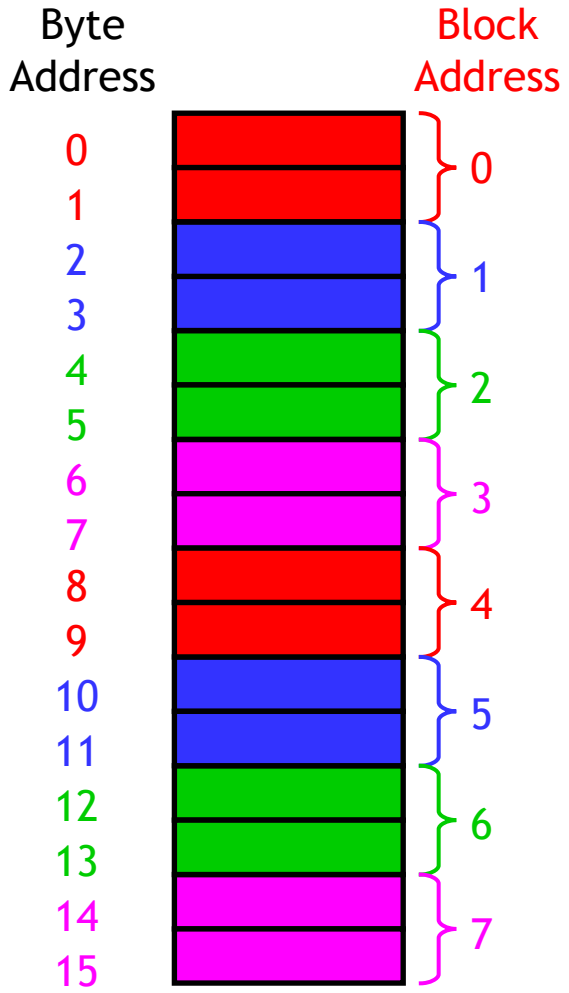
adrese *i*, izvrsimo celobrojno deljenje

$$i \;/\; 2^n$$

- Nas primer ima dvo bajtni cache blok, tako da

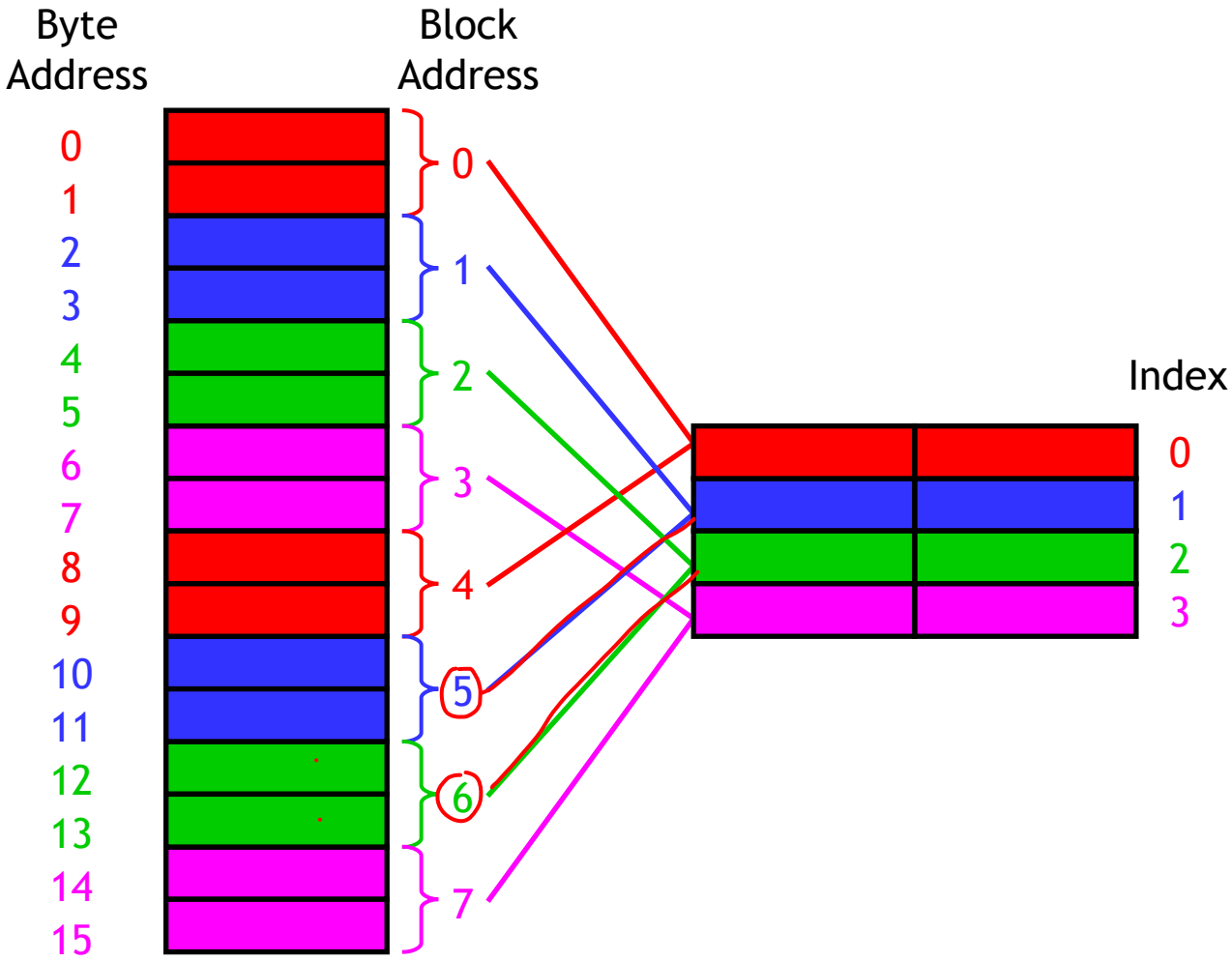
16-byte glavnu memoriju mozemo da posmatramo kao “8-block” glavne memorije.

- Na primer memorijske adrese 12 i 13 odgovaraju blok addesi 6, jer je 12 / 2 = 6 i 13 / 2 = 6.



Mapiranje kesa

- Jednom kada znamo blok adrese, mozemo da ih mapiramo u kes blokove.
- Na primer, blok memorije 6 pripada bloku u kesu 2, jer je 6 mod 4 = 2.
- Ovo odgovara smestanju podataka iz memorijske adrese 12 i 13 u cache blok 2.



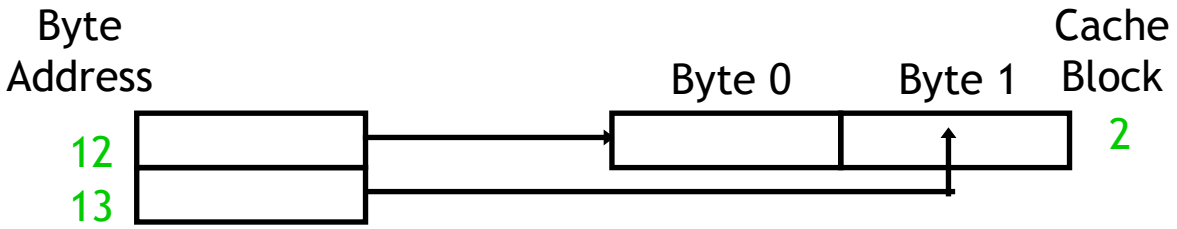
Smestanje podataka unutar bloka

Kada smo pristupili jednom bajtu podataka u memoriji kopiramo čitav blok u cache, sa nadom da ce da se iskoristi prostorna lokalnost.

- U nasem primeru, ako program cita sa bajt adrese 12 bice upisan kompletan memorijski blok 6 (obe adrese 12i 13) u cache blok 2.

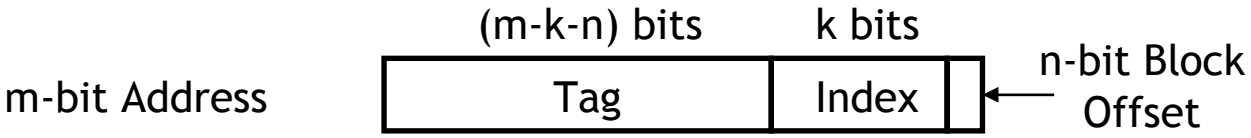
Isto se desava I ako citamo sa adrese 13, takodje ce biti upisan kompletan memorijski blok 6 (obe adrese 12i 13) u cache blok 2,

- Da bi upis bio jednostavniji bajt *i* memorijskog bloka je uvek upisan u byte *i* odgovarajuceg cache bloka.

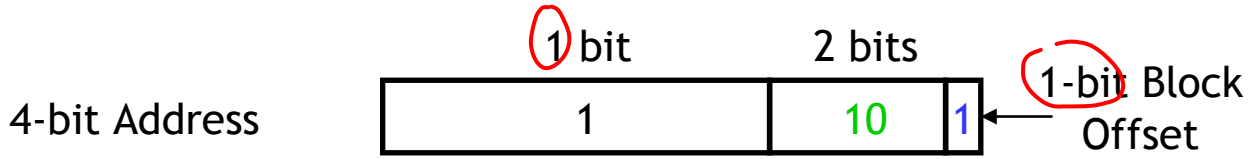


Lociranje podatka u kesu

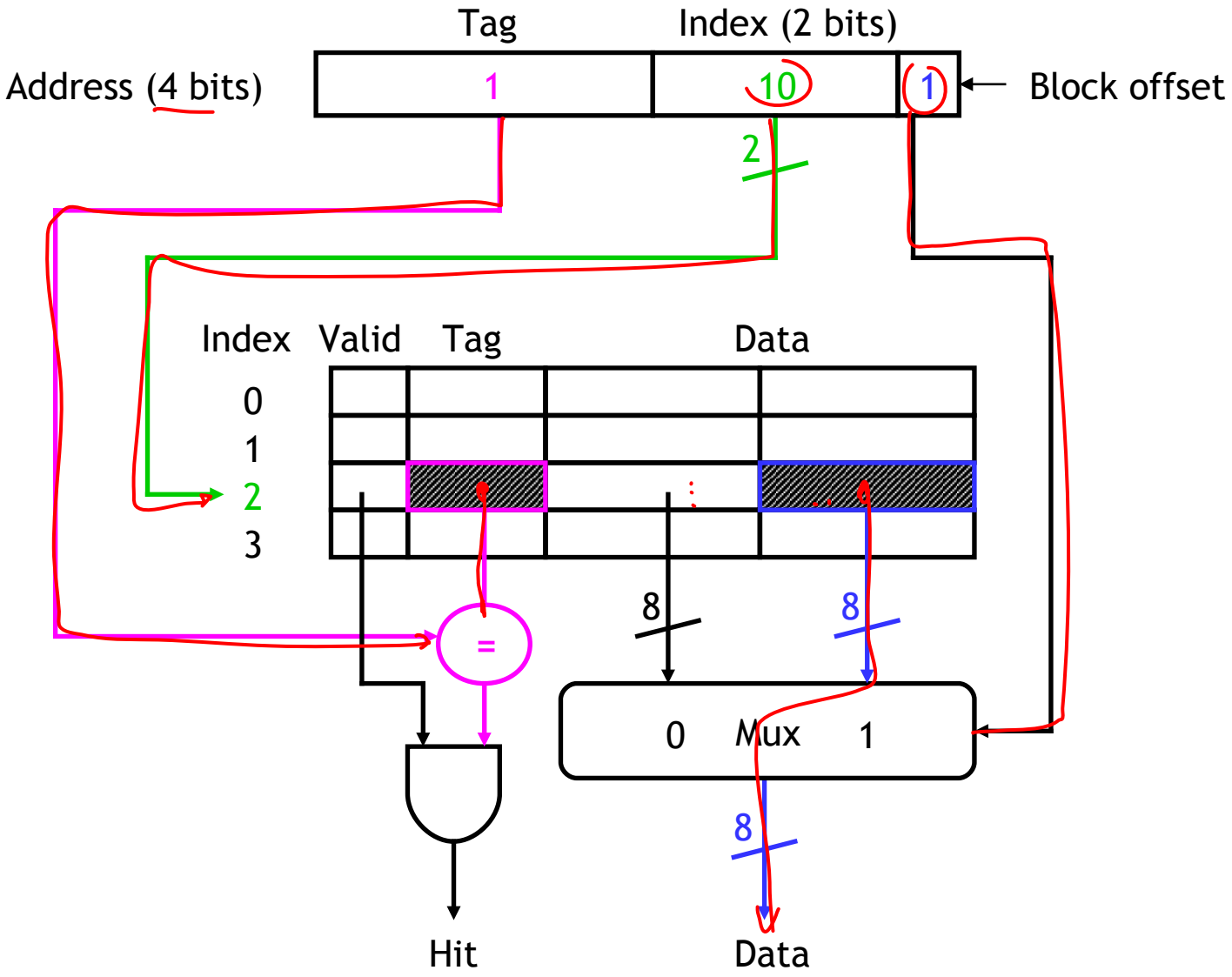
- Neka imamo kes sa 2^k blokova, svaki sadrzi 2^n bajtova.
- *k* bitova adrese selektuje jedan od 2^k kes bloka.
  - nizih *n* bitova je sada block offset koji definise koji od 2^n bajtova u kes bloku cuva podatke.



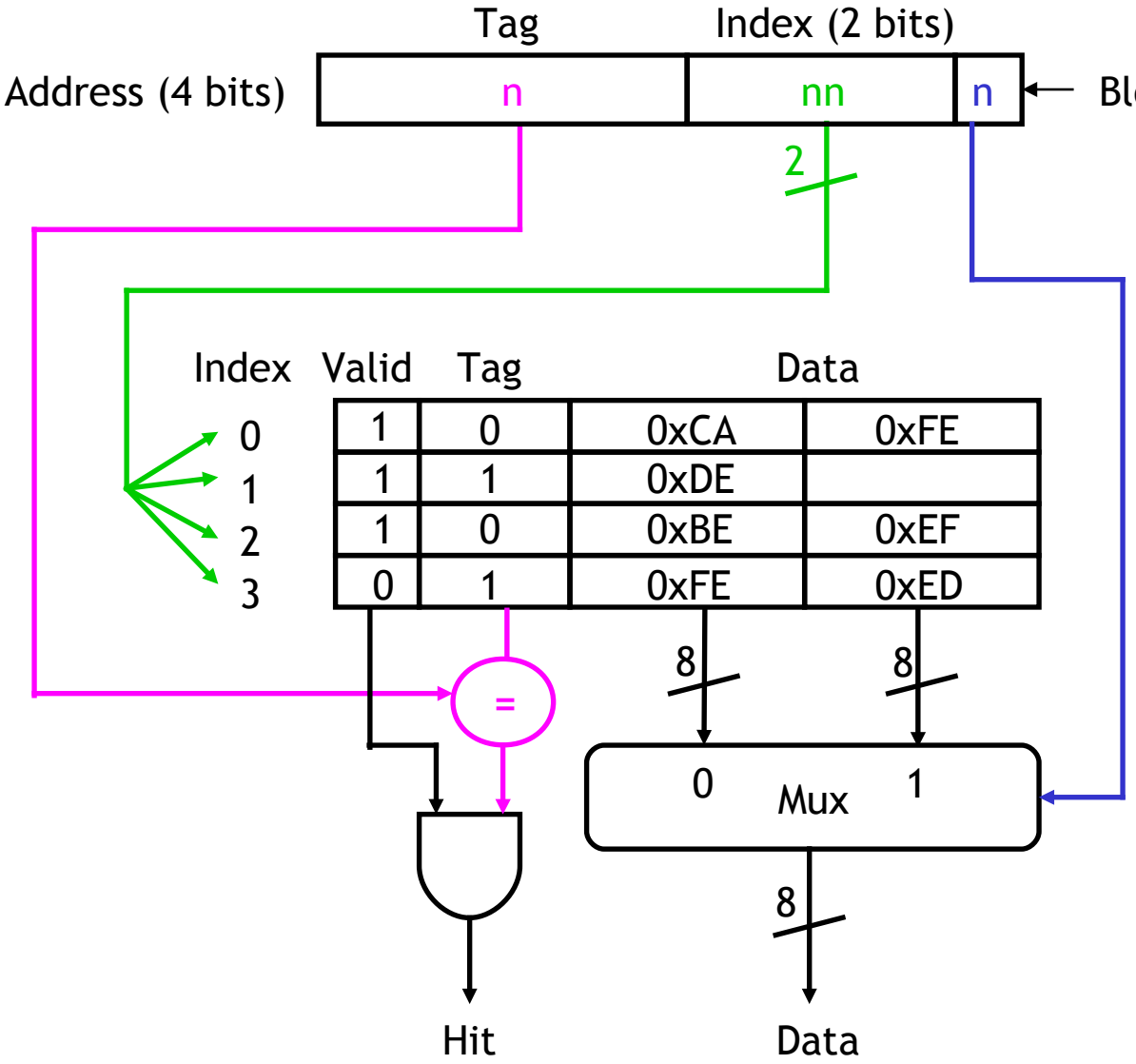
- Nas primer koristi 2^2-bloka kesa sa 2^1 bajta po bloku. Tako, memorijska adresa 13 (1101) treba da bude smestana u bajt 1 kes bloka 2.



Primer


Tag
Index (2 bits)
Address (4 bits)
1
10
1
Block offset
2
Index
Valid
Tag
Data
0
1
2
3
=
8
8
0
Mux
1
8
Hit
Data

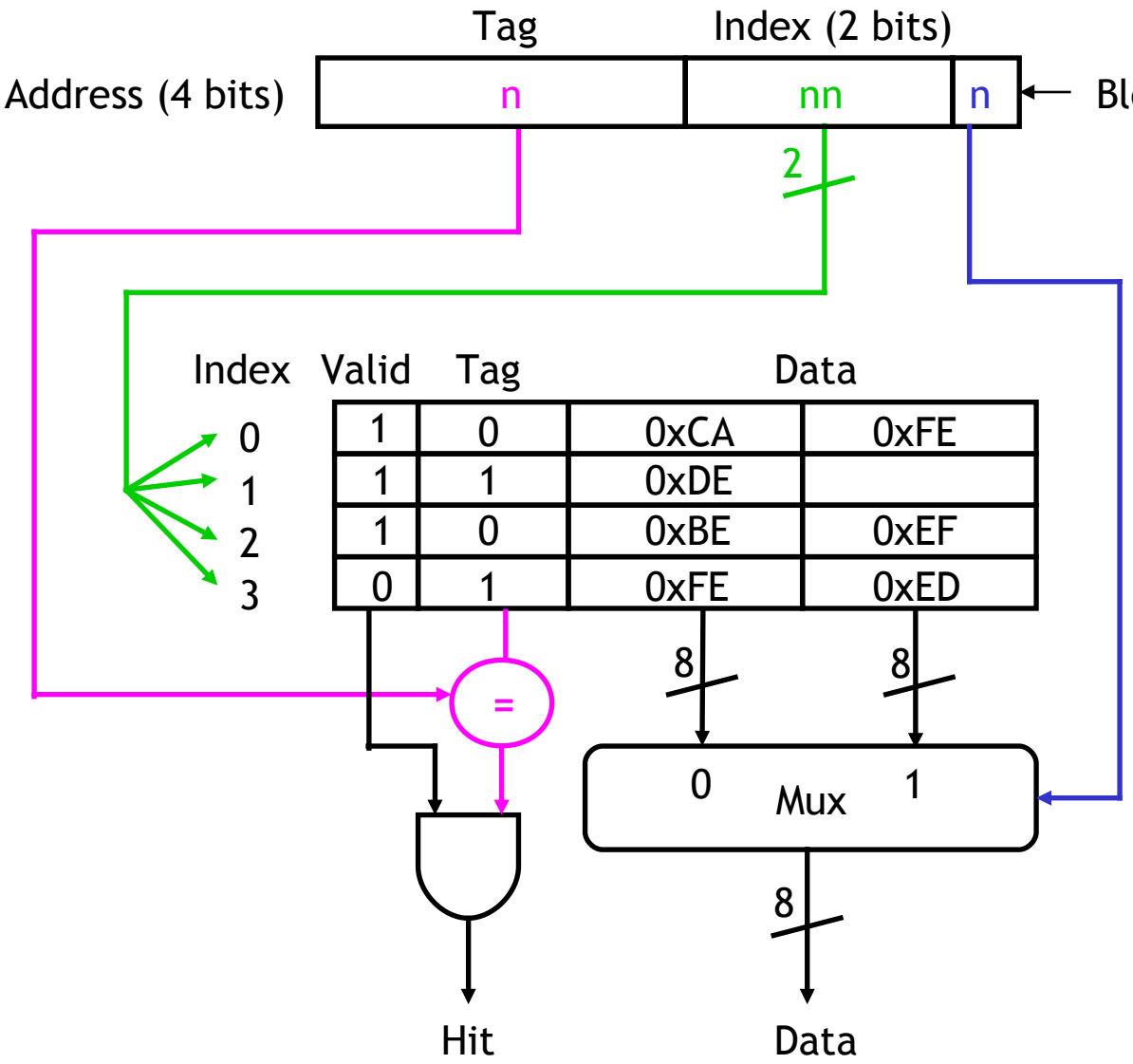
Primer za vezbu



Za navedene adrese dole, koji bajt je procitan iz kesaili se dogodio promasaj?

- 1010
- 1110
- 0001
- 1101

Primer za vezbu



Za navedene adrese dole,

koji bajt je procitan iz kesa ili se dogodio promasaj?

1010

1110

0001

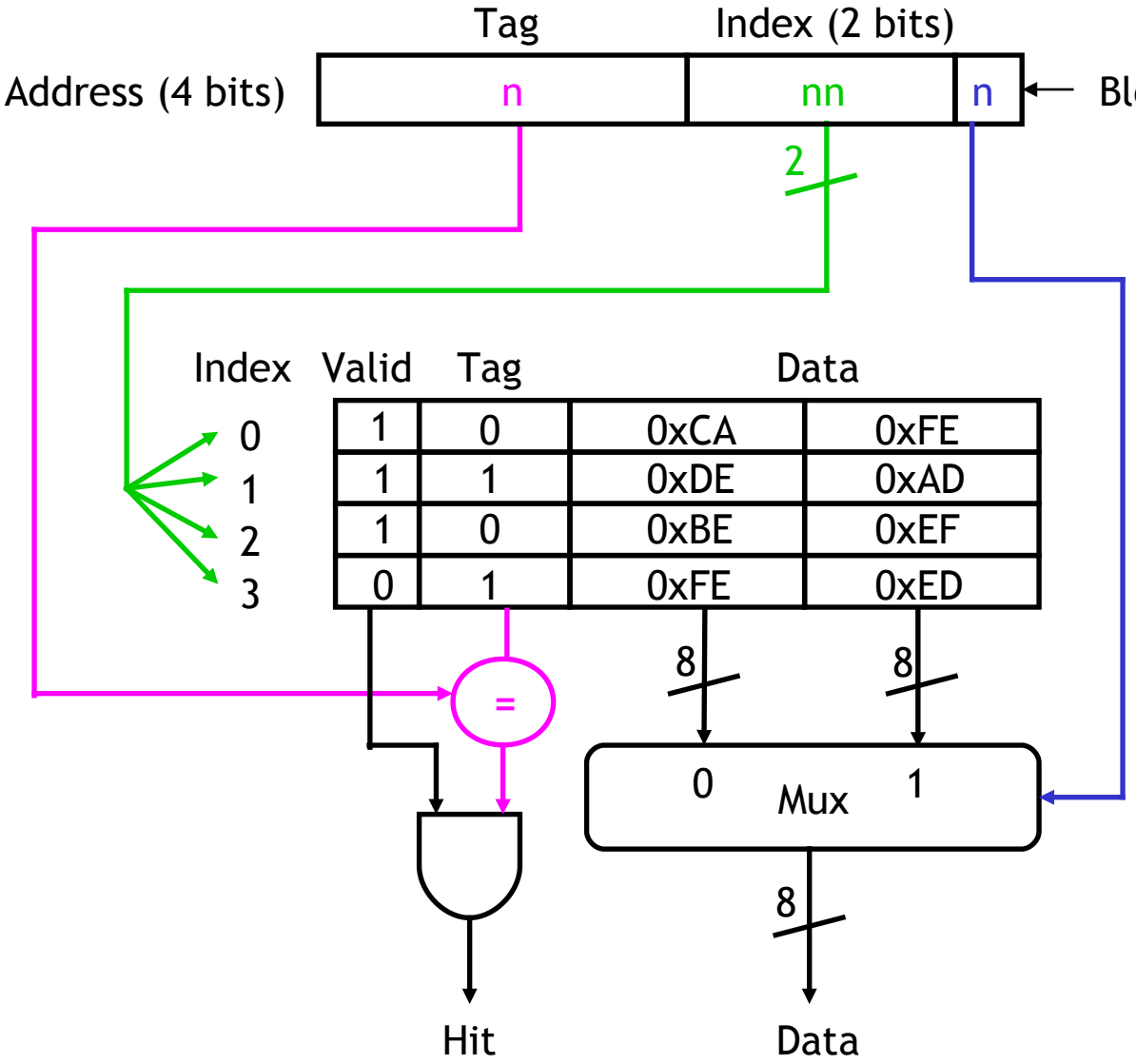
1101

tag 1

index 01

offset 0

Primer za vezbu

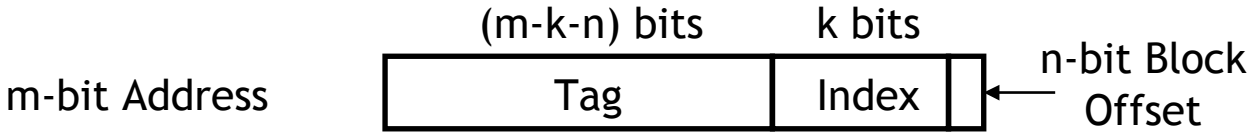


For the addresses below, what byte is read from the cache (or is there a miss)?

- 1010 (0xDE)
- 1110 (miss, invalid)
- 0001 (0xFE)
- 1101 (miss, los tag)

Koriscenje aritmetike

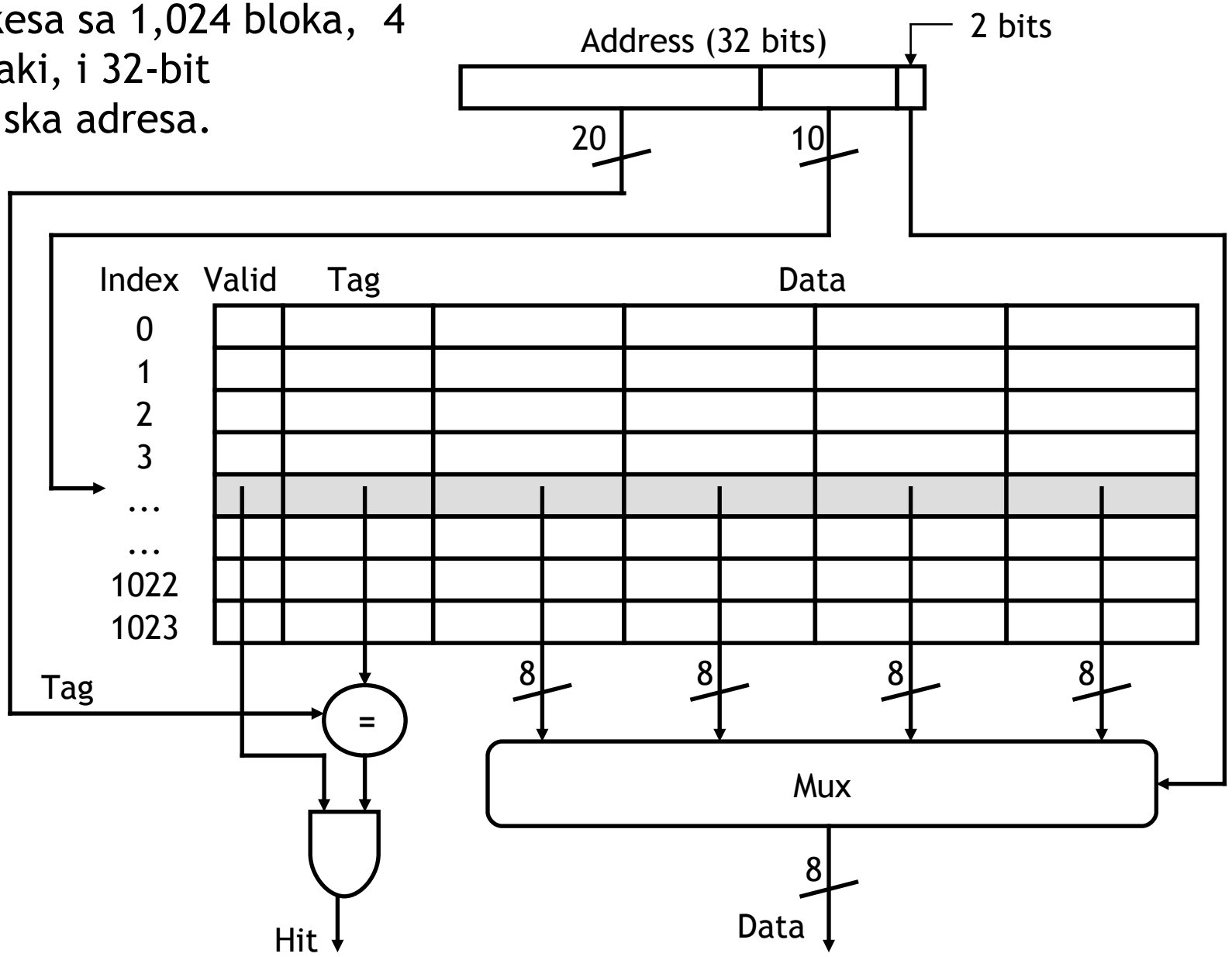
- Ekvivalentan nacin da se nadje tacna lokacija u kesu je koricsenje aritmetike.



- Mozemo da nadjemo index u dva koraka.
  - Celobrojno deljenje adrese sa 2^n za nalazenje bloka adrese.
  - Onda mod block adrese sa 2^k za nalazenje indeksa.
- block offset je samo memorijska adresa po mod 2^n.
- Na primer, mozemo da nadjemo adresu 13 u 4-bloku, 2-bajta po bloku kesa.
  - Blok adresa je 13 / 2 = 6, tako da je index 6 mod 4 = 2.
  - Block offset je 13 mod 2 = 1.

Primer dijagrama veceg kesa

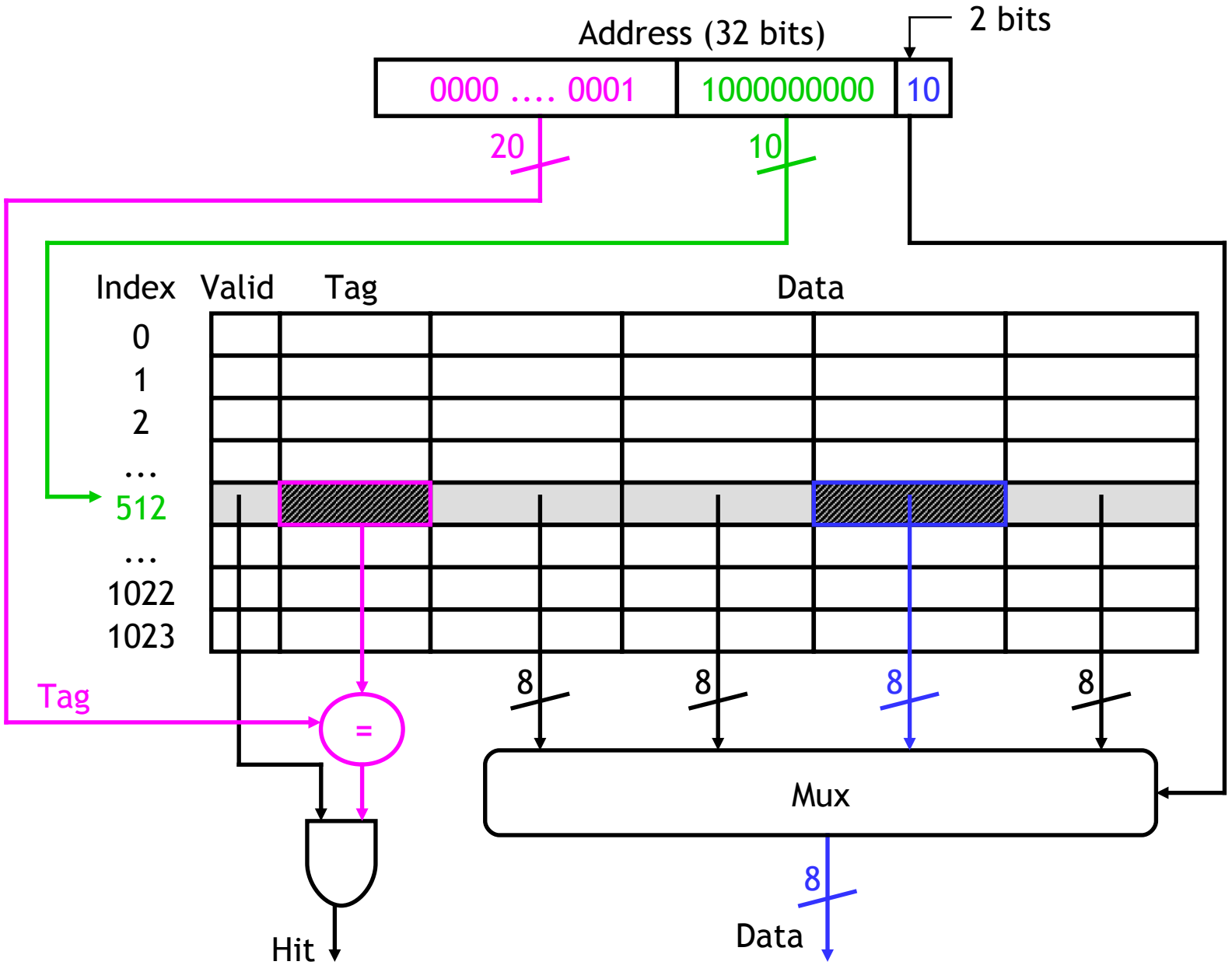
- Primer kesa sa 1,024 bloka, 4 bajta svaki, i 32-bit memorijska adresa.



Primer mapiranja veceg kesa

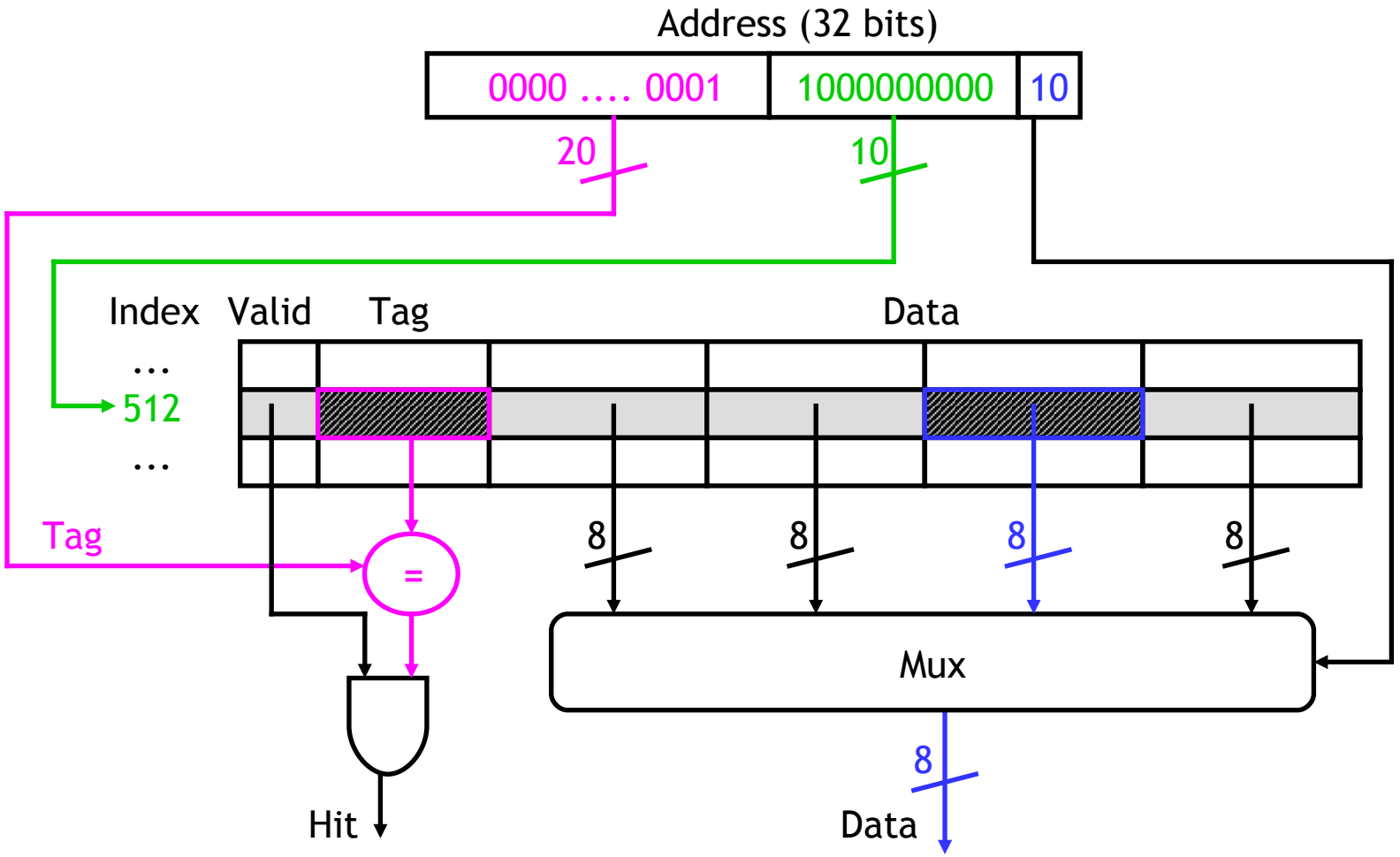
- Gde bi bajt sa memorijske adrese 6146 bio uskladisten u direktom preslikavanjum 2^{10}-blok kesa sa 2^{2}-byte blokom?
- Možemo odrediti to u binarnom obliku.
  - 6146 binarno je 00...01 1000 0000 00 10.
  - Nizih 2 bita, 10, znaci da se radi o drugom bajtu u bloku.
  - Sledecih 10 bitova, 1000000000, predstavlja indeks bloka (512).
- Ekvivalentno, ako koristimo aritmetiku.
  - Blok offset 6146 mod 4, iznosi 2.
  - Adresa bloka je 6146/4 = 1536, dakle indeks 1536 mod 1024, or 512.

Diagram mapiranja veceg kesa



Sta se desava sa ostatkom u kes bloku?

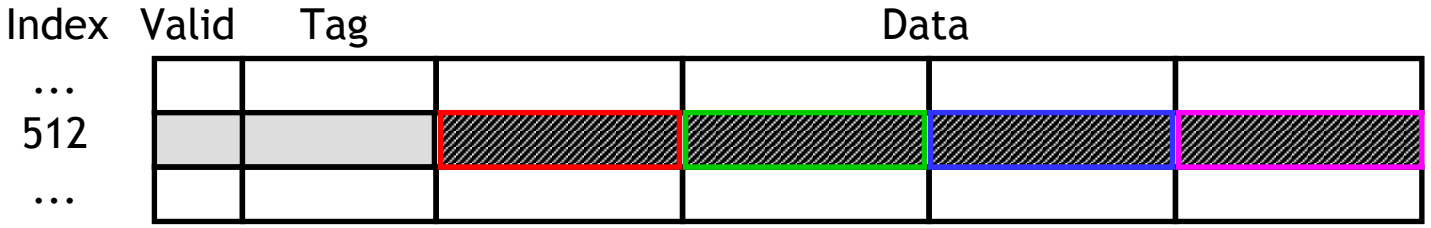
- Ostala tri bajta istog kes bloka dolaze iz istog memorijskog bloka, čije adrese moraju da imaju isti indeks (1000000000) I isti tag (00...01).



Ostatak u istom kes bloku

- Opet, blok *i* memorijskog bloka je sacuvan u u byte *i* odgovarajuceg kes bloka.
  - U nasem primeru, memorisjki blok 1536 sastoji se od bajtova adrese 6144 do 6147. Dakle bajtovi 0-3 kes bloka sadrze podatke sa adrese 6144, 6145, 6146 i 6147 respektivno.
  - Najniza 2 bita memorijske adrese definisu blok offset.

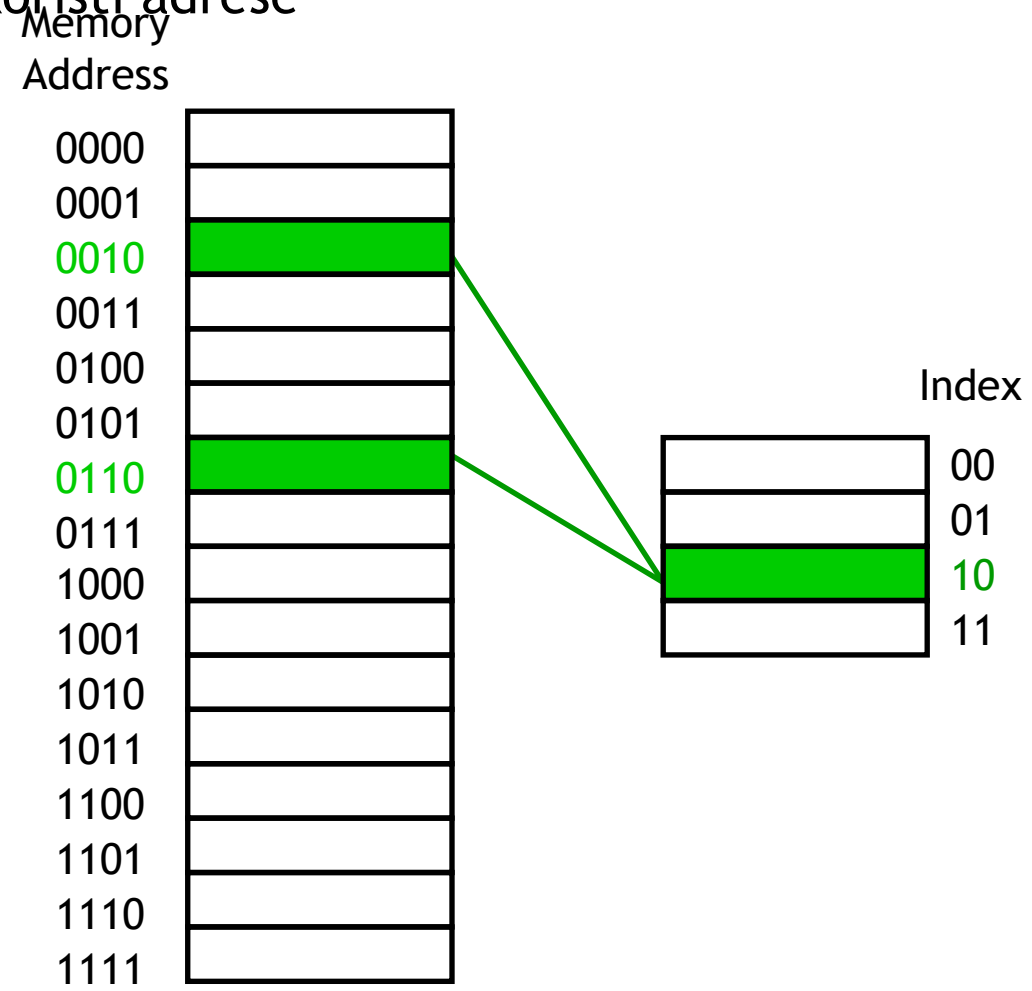
Block offset	Memory address	Decimal
00	00..01 1000000000 00	6144
01	00..01 1000000000 01	6145
10	00..01 1000000000 10	6146
11	00..01 1000000000 11	6147



Nedostaci direktnog mapiranja

- Direktno mapiranje je jednostavno: indeksi I offset se racunaju jednostavno, jer svaka memorijska adresa pripada tačno jednom bloku.
- Ali,sta se desava ako program koristi adrese

  2, 6, 2, 6, 2, …?



Nedostaci direktnog mapiranja

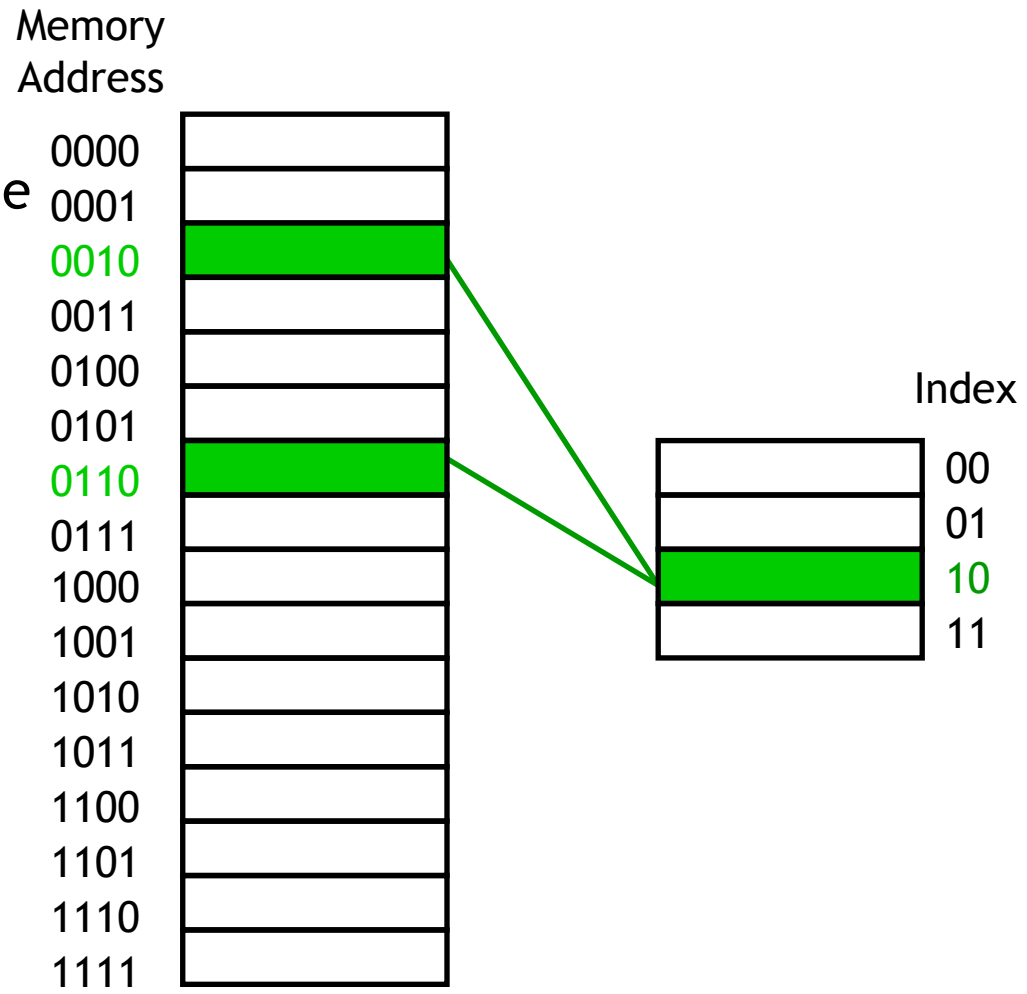
Direktno mapiranje je jednostavno: indeksi I offset se racunaju jednostavno, jer svaka memorijska adresa pripada u tačno jednom bloku.

- Ipak, ovo mapiranje nije fleksibilno. Ako program koristi adrese 2, 6, 2, 6, 2, ..., onda svaki pristup kesu rezultuje promasaj I ucitavanje u kes blok 2.
- Ovaj kes ima četiri bloka,

ali direknto mapiranje ne dopusta nam da koristitmo sve blokove.

- Ovo rezultira sa nezeljeno

mnogo promasaja

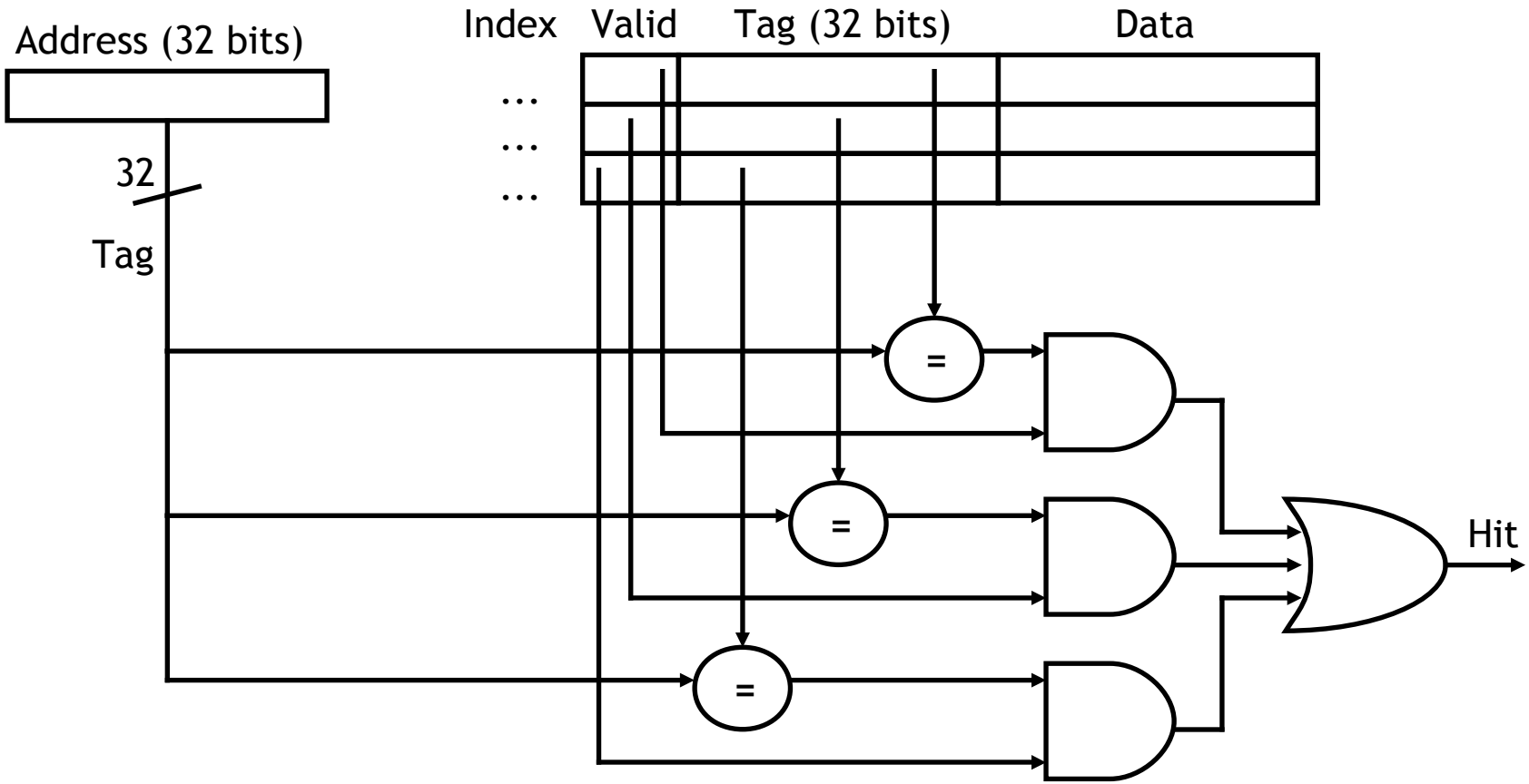


Poptupno asocijativni kes

- Potpuno asocijativni kes (fully associative cache) dozvoljava da se podaci smestaju u bilo koji blok kesa, umesto da se svaki memorijski blok smesta u jednan određeni blok kesa.
  - Kada se podaci preuzmu iz memorije, može da se sacuva u bilo koji neiskorišceni blok u kesu.
  - Ovaj nacin mapiranja resava problem konflikta da se dve ili vise memorijskih adresa mapiraju u isti blok.
- U prethodnom primeru, mozemo da stavimo memorijsku adresu 2 u kes blok 2, a adresu 6 u blok 3. U tom slucaju ponavljanje pozivanja adrese 2 I 6 bice pogodak a ne promasaj.
- Ako su svi blokovi zauzeti onda je najbolje zameniti poslednji korisceni blok(least recently used) , pod pretpostavkom da nece biti ponovo koriscen skorije vreme.

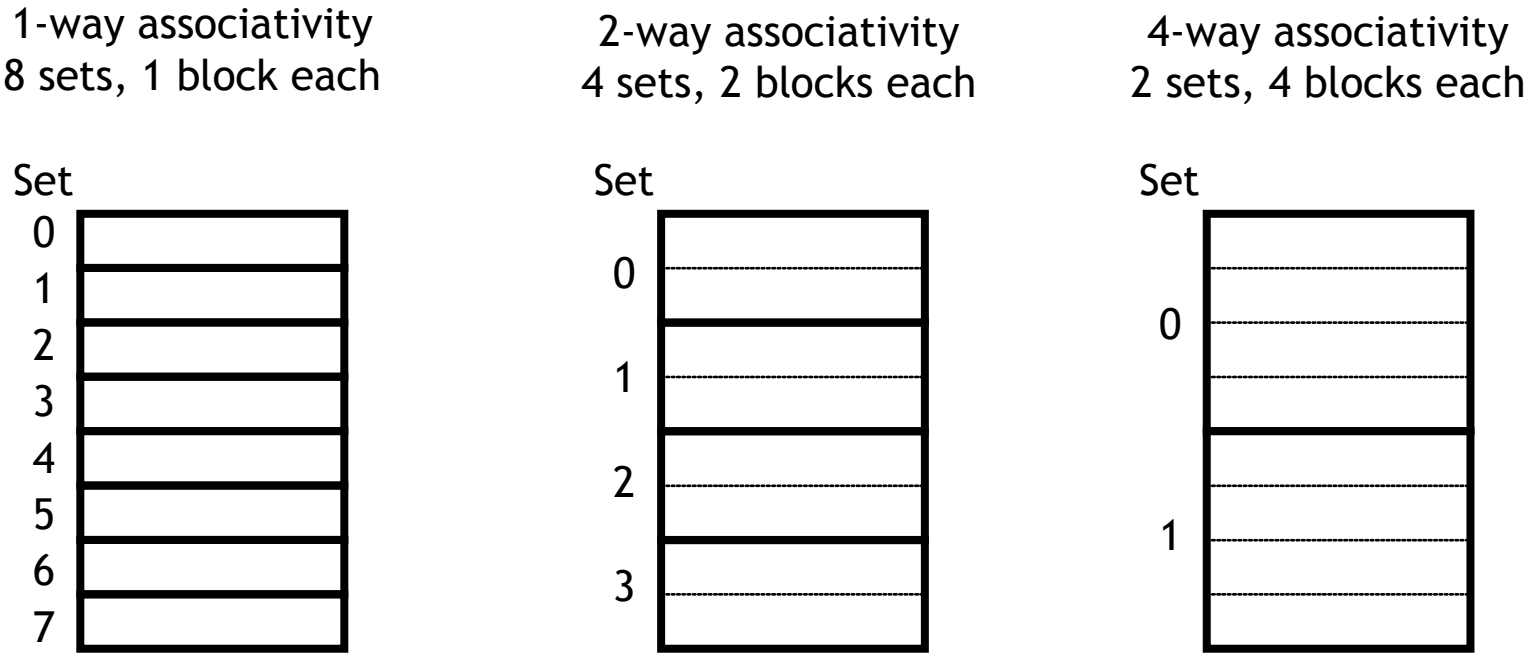
Cena potpune asocijativnosti

- Međutim, potpuno asocijativni cache je skupo implementirati.
  - Zbog toga sto ne postoji indeks polje u adresi, kompletna adresa se koristi kao tag povecavajuci velicinu kesa.
  - Podaci mogu biti bilo gde u kesu, pa moramo proveriti oznaku svakog keš bloka. To je puno komparatora



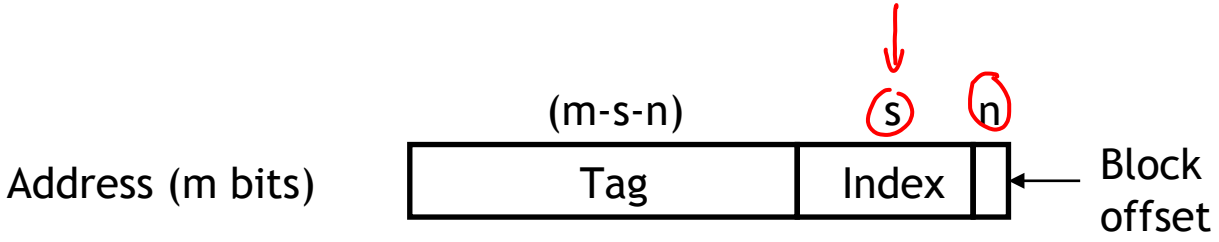
Set associativity

- Nesto izmedju je set-associative cache.
  - Kes je podeljen u grupe blokova nazvane setovi.
  - Svaka memorijska adresa mapira tacno jedan set u kesu, ali podatak moze da se smesti u bilo koji blok tog seta.
- Nekoliko mogucih organizacija 8-blokovsog kesa.



Pristup set asocijativnom bloku

- Ako kes ima 2^s setova i svaki blok ima 2^n bajtova, memorijska adresa moze da se podeli na sledeci nacin.



-
-

Block Offset = Memory Address mod 2^n

Block Address = Memory Address / 2^n

Set Index = Block Address mod 2^s

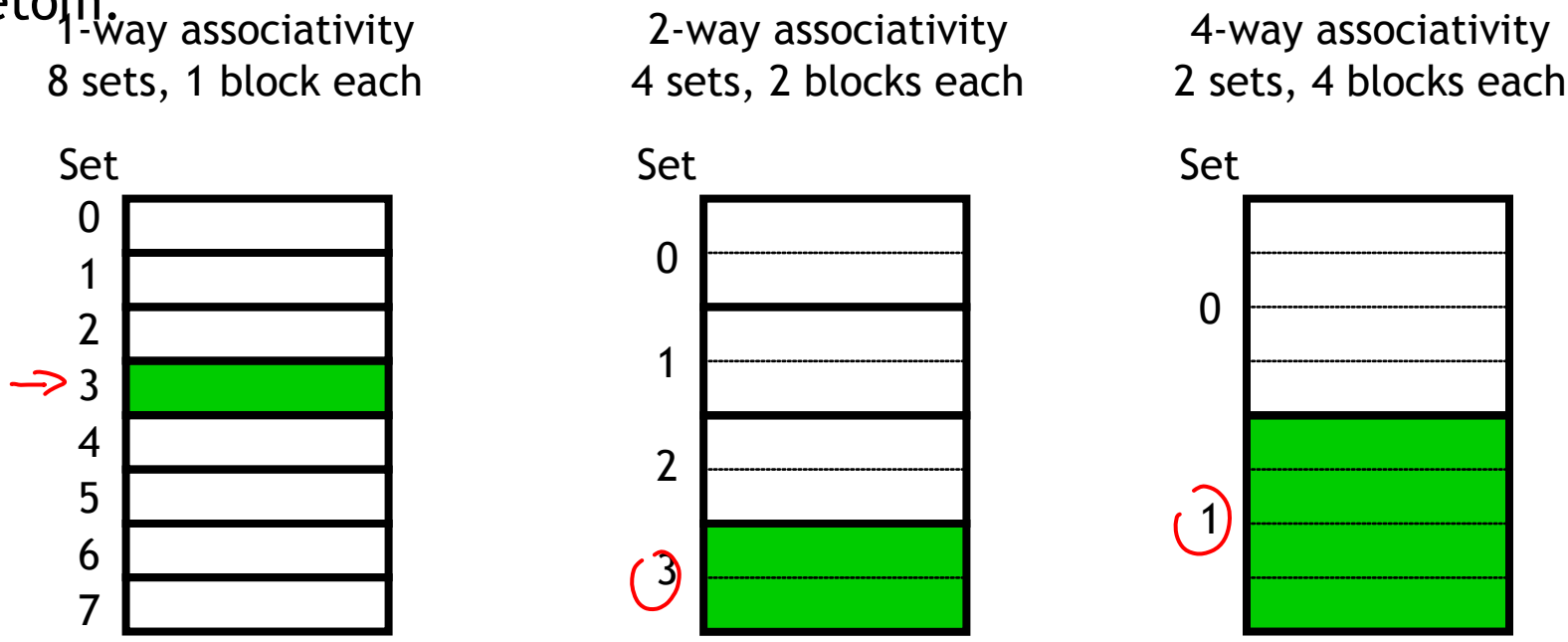
Primer smestanja podataka u set-associative caches

- Smestanje podatka sa memorijske adrese 6195 , pod pretpostvakom da imamo 8-block kes sa 16 bajtova po bloku
- 6195 binarno je 00...0110000 011 0011.
- Svaki blok ima 16 bajtova, tako da nizih 4 bita su block offset.
- Za 1-way kes, sledeca tri bita (011) predstavljaju indeks seta.

• Za 2-way kes, sledeca dva bita (11) predstavljaju indeks seta.

Za 4-way kes, sledeci bit (1) predstavljaju indeks seta.

- Podatak moze da se smesti u bilo koji blo , prikazan zeleno, sa koreknim setom.



Zamena bloka

- Bilo koji prazan blok može se koristiti za smestanje podataka.
- Ako nema praznih blokovi, cache kontroler će pokušati da zameni poslednji korišceni blok.
- Skupoje pratiti stvarno poslednji korisceni blok, tako da se koriste neke aproksimacije .