תכנות מתקדם
מצגת 8
מטמון

היררכית הזיכרון

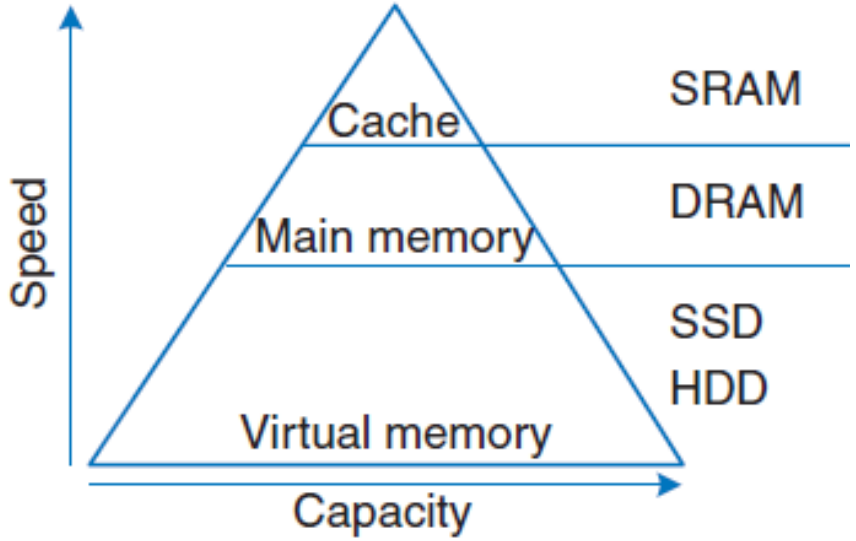
מהירות ביצוע פקודה במעבד (CPU) גדולה בהרבה (פי 10-100) ממהירות גישה לזיכרון הרגיל.

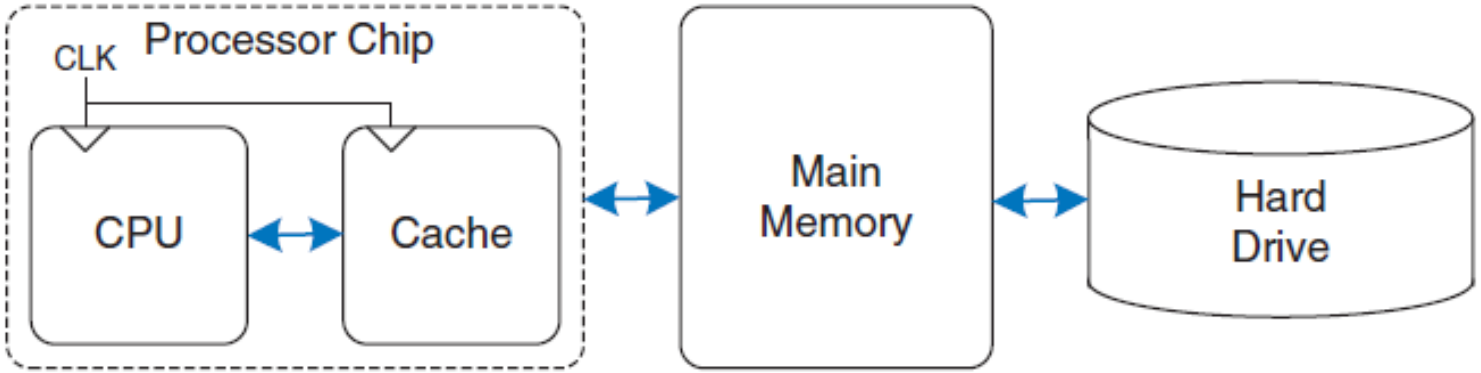
כדי שהמעבד לא יצטרך להמתין, המחשב מכיל זיכרון **קטן מהיר ויקר** הנקרא **מטמון**.

המטמון נמצא בין המעבד לזיכרון הרגיל, המעבד ניגש דרכו לזיכרון.

מאחר שתכניות ניגשות שוב למקומות אליהן ניגשו לאחרונה או למקומות הסמוכים להם (תכונת המקומיות), נכניס למטמון את הגישות האחרונות והסמוכות ואז רוב הגישות יהיו לזיכרון מהיר.

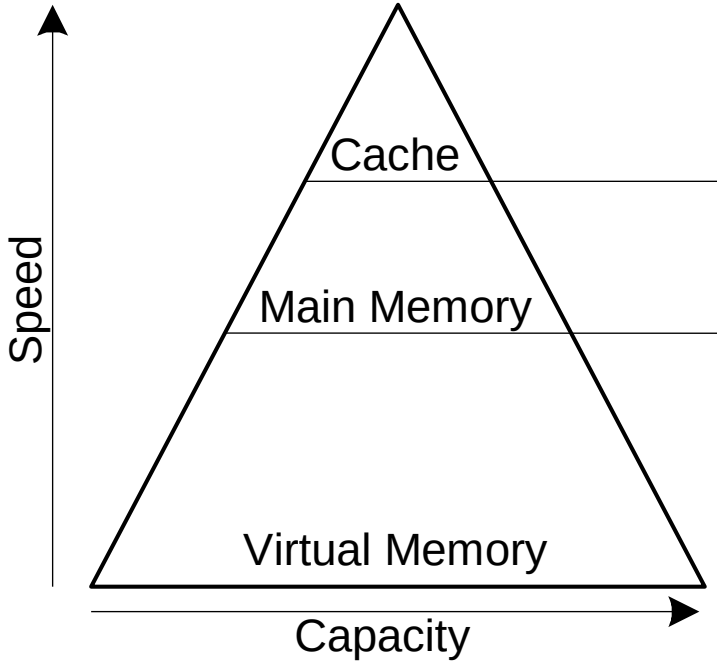
גם הזיכרון הרגיל מהווה מטמון ביחס לדיסק.





היררכית הזיכרון

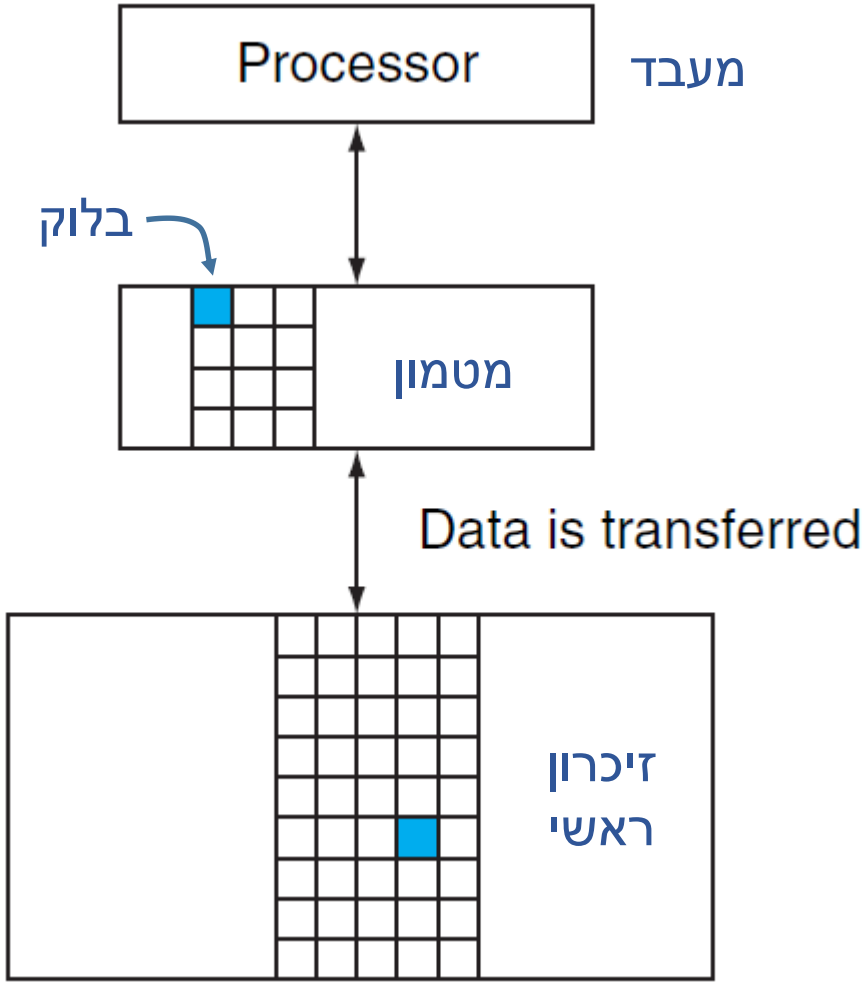
- זיכרון אידאלי: מהיר גדול וזול ...
- אפשר להגיע קרוב לכך אם הזיכרון יהיה בנוי מהיררכיה של זיכרונות.
  - הפקודות והנתונים שהתוכנית משתמשת בהם כעת, יהיו בחלק הקטן והמהיר הנקרא מטמון.
  - יתר הפקודות והנתונים יהיו בחלקים גדולים יותר ואיטיים יותר.



Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$10,000	1	25+
DRAM	$10	10 - 50	10
SSD	$1	100,000	0.5
HDD	$0.1	10,000,000	0.1

מטמון

- מטמון מכיל קטעי זיכרון מהירים הנקראים בלוקים.
- כשמעבד ניגש לנתון בזיכרון מתבצע חיפוש במטמון.
- אם הנתון נמצא במטמון (hit) הגישה תהיה מהירה.
- אם הנתון לא נמצא (miss) הוא מועתק תחילה מהזיכרון למטמון.
  - לפי תכונת המקומיות בזמן, בקרוב ניגש שוב אל הנתון.
- ההעתקה היא בגודל של בלוק (דוגמה: 64 בתים)
  - לפי תכונת המקומיות במקום, בקרוב ניגש לנתונים הסמוכים לנתון.
- לפי תכונת המקומיות, חלק גדול מהגישות יהיו במטמון, אם כן הגישה לזיכרון תהיה מהירה.



יחס הפגיעה במטמון

- **Hit - פגיעה:** הנתון נמצא במטמון
- **Miss - החטאה:** הנתון לא נמצא במטמון
- **Hit Rate - יחס הפגיעה במטמון :** מספר הפגיעות מחולק במספר הגישות לזיכרון
- **Miss Rate - יחס ההחטאה:** 1 – Hit Rate

דוגמה:

תכנית ביצעה 2000 פקודות שניגשות לזיכרון לקריאה או כתיבה של נתונים.
1250 מהגישות נמצאו במטמון.
מהו יחס הפגיעה וההחטאה של הגישות לנתונים?

יחס הפגיעה: 1250/2000 = 0.625
יחס ההחטאה: 1 – Hit Rate = 0.375 = 750/2000

זמן הגישה הממוצע לזיכרון

זמן הגישה לנתון בזיכרון הראשי: t_{Memory}

זמן הגישה לנתון בזיכרון המטמון: t_{cache}

זמן הגישה הממוצע לנתון בזיכרון: $t_{cache} + (MissRate_{cache} * t_{Memory})$

דוגמה, נתון:

t_{Memory} = 100 cycles

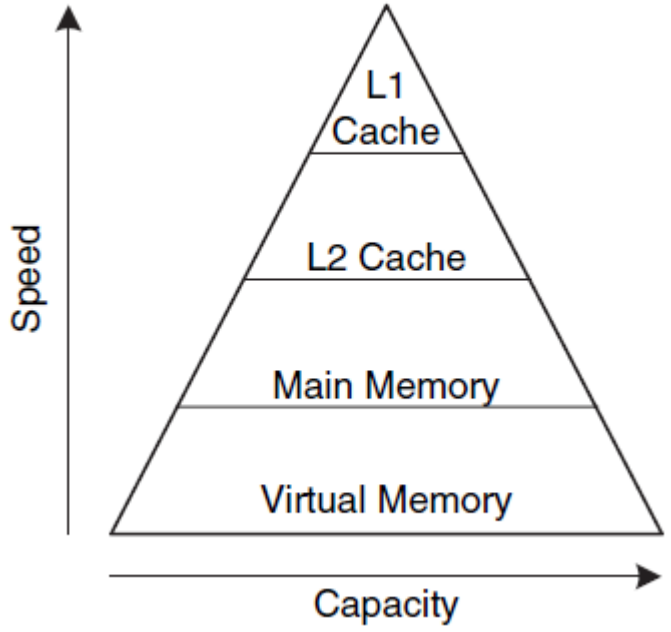
t_{cache} = 2 cycles

Miss Rate = 0.375

מה זמן הגישה הממוצע לנתון בזיכרון?

2 + (0.375 * 100) cycles = 39.5 cycles

מטמון עם כמה רמות



- ככל שהמטמון גדול יותר אחוז הפגיעה גבוה יותר, אבל הוא יותר איטי.
- אפשר להוסיף רמה נוספת של מטמון, גדול יותר איטי יותר (מהיר יותר מהזיכרון הראשי).
- Level 1: small and fast (e.g. 16 KB, 1 cycle)
- Level 2: larger and slower (e.g. 256 KB, 2-6 cycles)

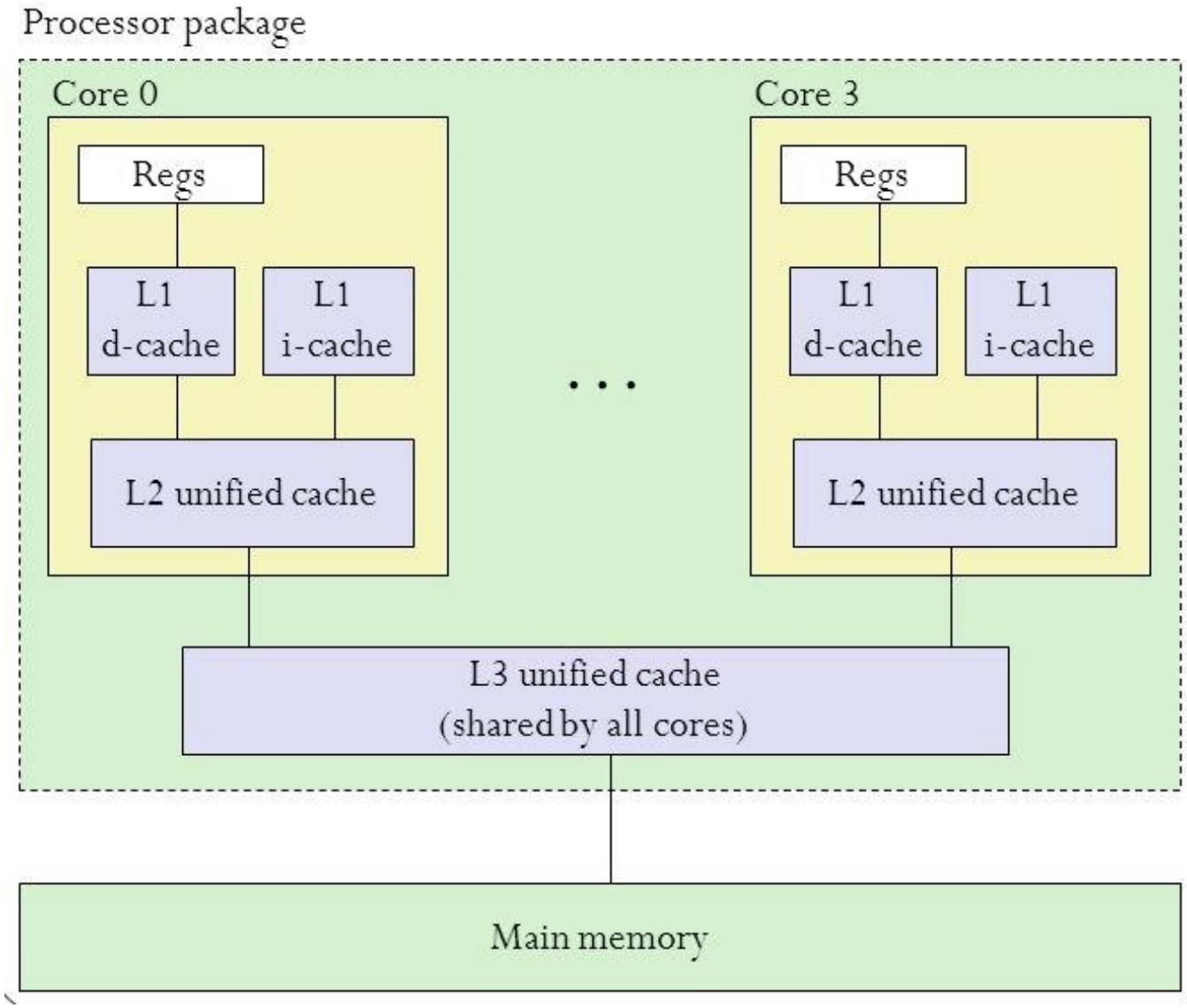
דוגמה: מה זמן הגישה הממוצע לזיכרון כאשר יש שתי רמות של מטמון:

Access Times: L1 = 1 cycle, L2 = 10 cycles, Main Memory = 100 cycles

Miss Rates: L1 = 5%, L2 = 20%

1 cycle + 0.05[10 cycles + 0.2(100 cycles)] = 2.5 cycles

Intel Core i7 Cache Hierarchy



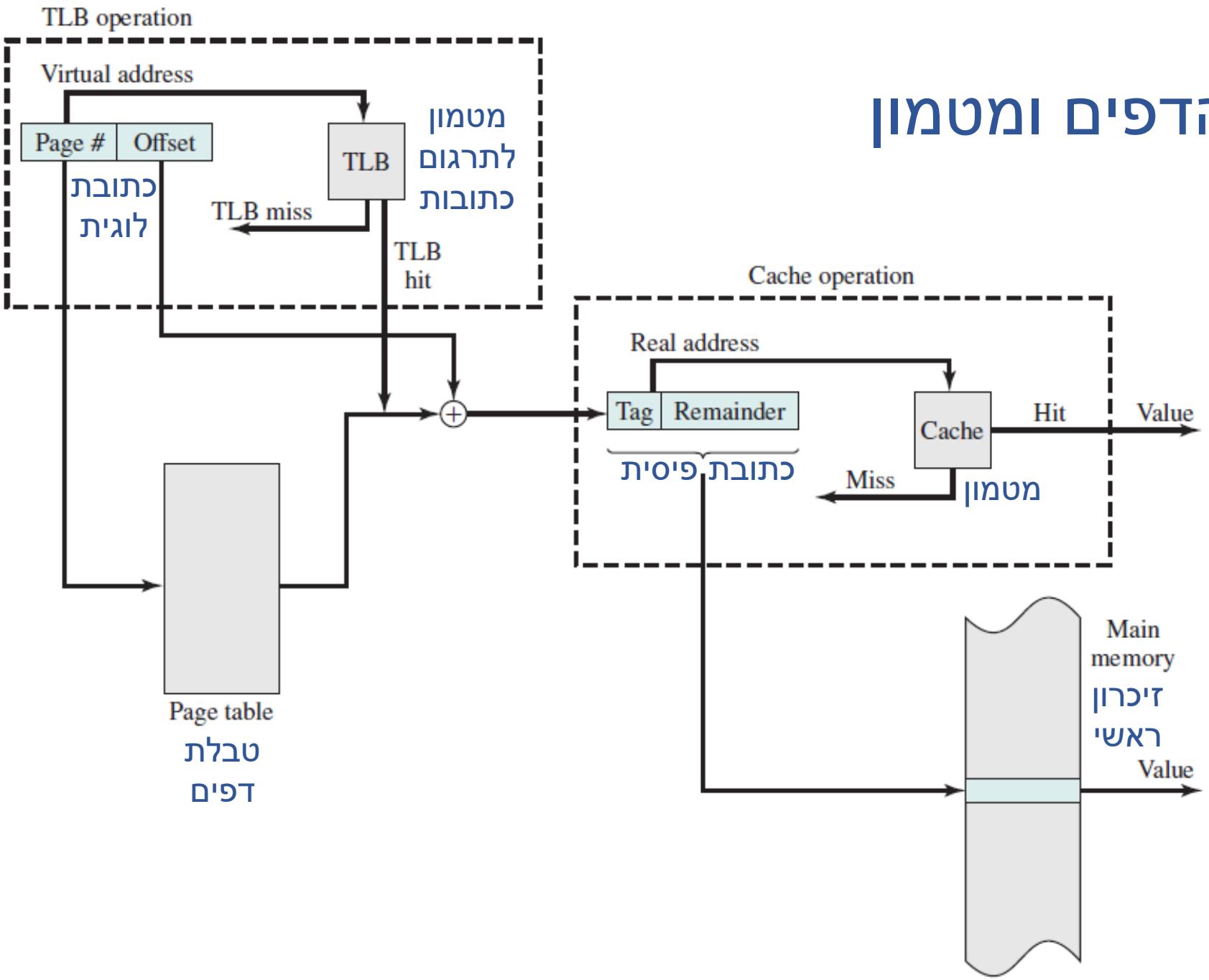
L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Block size: 64 bytes for all caches.

TLB, טבלת הדפים ומטמון



כשמערכת ההפעלה מפנה דף מהזיכרון הראשי ומסמנת בטבלת הדפים שאינו נמצא:

1. היא גם מוחקת את התרגום מה- TLB (אם ישנו).
2. וגם מוציאה את הנתונים מהמטמון (אם ישנם).

צרופים אפשריים של TLB, טבלת הדפים ומטמון

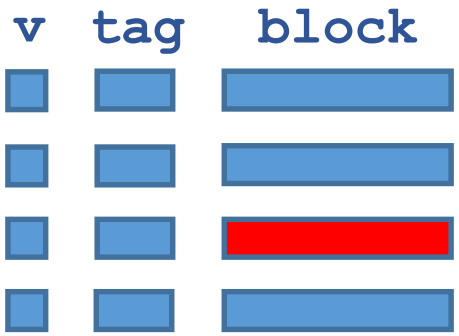
TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

מטמון במיפוי ישיר

זיכרון

- הזיכרון הראשי מחולק (לוגית) לבלוקים שגודלם כגודל הבלוקים של המטמון.
- **לאיזה בלוק במטמון יוכנס בלוק מהזיכרון?**
  - לכל בלוק בזיכרון יש מספר, מספר הבלוק של בית כלשהו בזיכרון הוא החלוקה של כתובת הבית בגודל הבלוק.
  - מספר הבלוק במטמון אליו ימופה הבלוק בזיכרון, הוא שארית החלוקה של מספר הבלוק בזיכרון במספר הבלוקים במטמון.

Direct Mapped

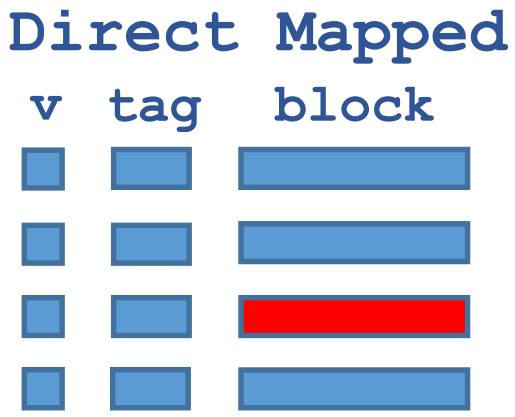


מטמון

דוגמה: נניח שגודל הבלוק הוא 16 בתים והמטמון מכיל 64 בלוקים, לאיזה בלוק במטמון ימופה בית שכתובתו 1210 ?

הבית שכתובתו 1210 נמצא בזיכרון בבלוק שמספרו 1210 / 16 = 75, הוא ימופה לבלוק שמספרו במטמון הוא: 75 % 64 = 11

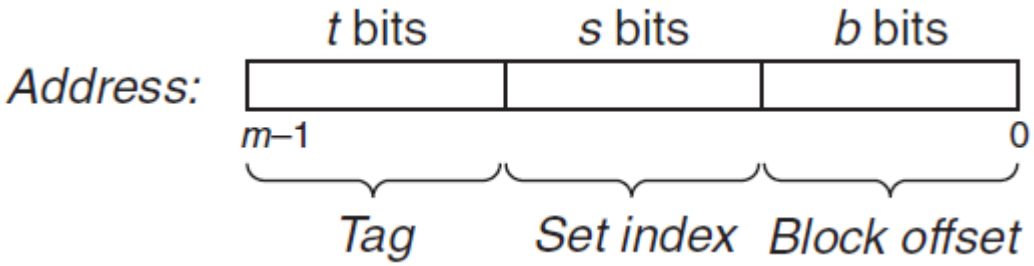
מטמון במיפוי ישיר



גודל הבלוק ומספר הבלוקים במטמון הם חזקה של 2.
איך נדע לפי הביטים של כתובת הנתון לאיזה בלוק יוכנס ?
איך נדע לפי הביטים אם הנתון נמצא במטמון ?

- נניח שגודל הבלוק הוא 2^b בתים.
- נניח שמספר הבלוקים במטמון הוא 2^s.
- אזי אם כתובות של בית בזיכרון היא בגודל m ביטים:

offset bits: b הביטים הימניים הם המקום של הבית בתוך הבלוק.
index bits: s הביטים האמצעיים הם מספר הבלוק במטמון שיכיל את הנתון.
tag bits: m – (s + b) הביטים הנותרים משמאל יוצמדו לכל בלוק שיוכנס למטמון
ובאמצעותם נוכל לדעת לאיזה בלוק בזיכרון הוא שייך.



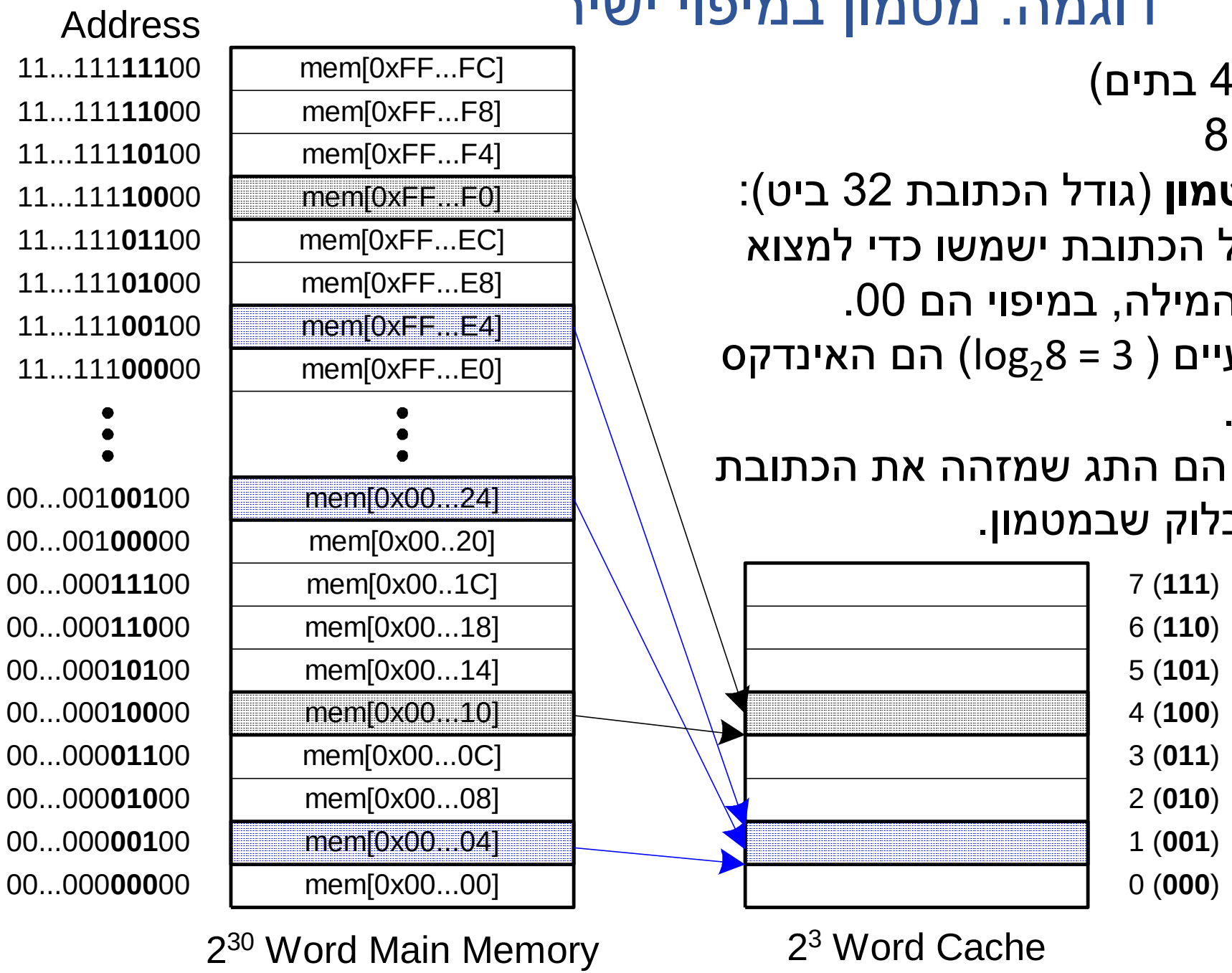
דוגמה: מטמון במיפוי ישיר

גודל הבלוק: מילה אחת (4 בתים)

מספר הבלוקים במטמון: 8

מיפוי כתובת בזיכרון למטמון (גודל הכתובת 32 ביט):

- שני הביטים הימניים של הכתובת ישמשו כדי למצוא את מרחק הבית בתוך המילה, במיפוי הם 00.
- שלושת הביטים האמצעיים ($\log_2 8 = 3$) הם האינדקס - מספר הבלוק במטמון.
- 27 הביטים השמאליים הם התג שמזהה את הכתובת של המילה שנמצאת בבלוק שבמטמון.



המילים בכתובות:

`0x00000004`

`0x00000024`

. . .

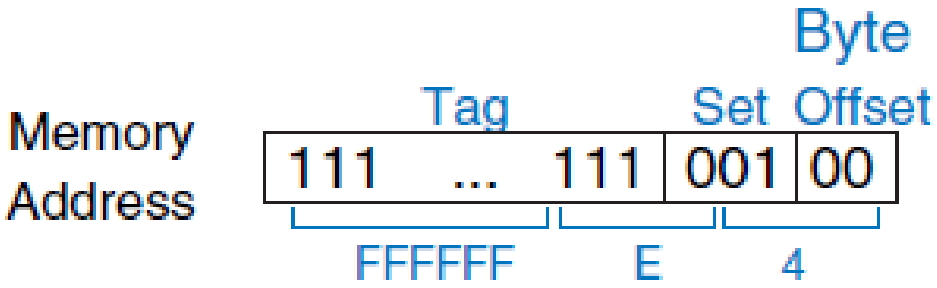
`0xFFFFFFE4`

ימופו לבלוק 1

דוגמה: מטמון במיפוי ישיר

- שאלה: לאיזה בלוק במטמון (הקודם) תמופה כתובת **`0xFFFFFFE4`**?

נכתוב את הכתובת ההקסה-דצימלית כמספר בינרי ונראה ששלושת הביטים של האינדקס הם 001, כלומר המילה תמופה לבלוק 1 במטמון, התג יכיל אחדות.



- שאלה:

נניח שישנם 1024 (2^{10}) בלוקים במטמון, גודל הבלוק הוא מילה וגודל הכתובת 32 ביט.

כמה ביטים בכתובת ישמשו לאינדקס (בחירת הבלוק במטמון) וכמה ישמשו לתג (לזיהוי הכתובת) ?

שני הביטים הימניים של הכתובת ישמשו כדי למצוא את מרחק הבית בתוך המילה.

במטמון שבו 1024 בלוקים יש צורך ב- 10 ביטים של אינדקס כדי לבחור בלוק.

שאר 20 = 2 – 10 – 32 ישמשו לתג.

Conflict Misses

נניח שמטמון במיפוי ישיר מכיל שני בלוקים כל אחד בגודל 16 בתים.

המערך x תופס 32 = 4 * 8 בתים ומתחיל בכתובת 0.

המערך y מתחיל אחריו ותופס את 32 הבתים הבאים.

```
float dotprod(float x[8], float y[8])
{
    float sum = 0.0;
    int i;
    for (i = 0; i < 8; i++)
    sum += x[i] * y[i];
    return sum;
}
```

x[i] ו- y[i] ימופו לאותו בלוק.

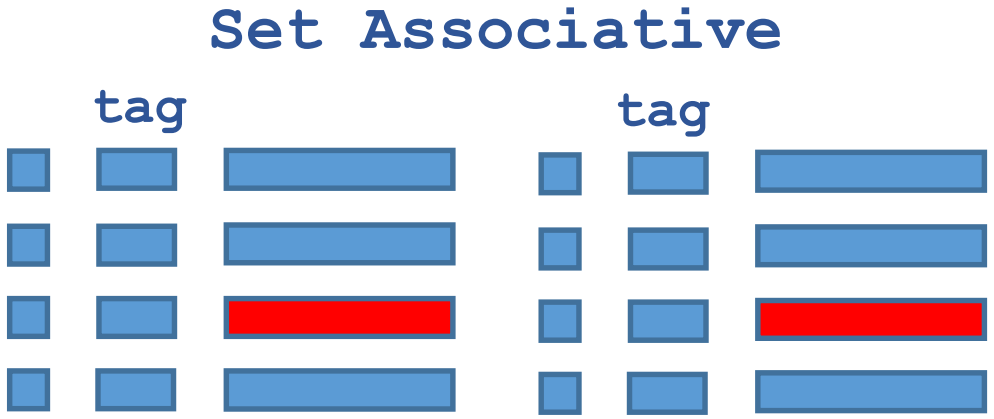
הפניה ל- x[0] תגרום להבאת הבלוק המכיל את x[0] עד x[3] לבלוק 0.

הפניה ל- y[0] תגרום להבאת הבלוק המכיל את y[0] עד y[3] לבלוק 0 במקום הבלוק הקודם.

אם נגדיר את המערך הראשון x[12] הבעיה תיפתר.

ומה יהיה אז אחוז הפגיעה ? 75%

מטמון במיפוי לקבוצה



- אם ניגש לסירוגין לשני בלוקים בזיכרון שממופים לאותו בלוק במטמון, הם יפנו זה את זה.
- כדי למנוע זאת, יש במטמון קבוצה (set) של בלוקים במקום כל בלוק.

- מספר **הקבוצה** במטמון אליו ימופה הבלוק בזיכרון, הוא שארית החלוקה של מספר הבלוק בזיכרון במספר **הקבוצות** במטמון.
- אם מספר הקבוצות במטמון הוא 2^s , s הביטים האמצעיים הם מספר הקבוצה.
- כל בלוק בקבוצה יוכל להכיל את הנתון, בהכנסה לקבוצה נצטרך להחליט את מי לפנות (LRU).
- בחיפוש במטמון נצטרך להשוות את התג לכל התגים שבקבוצה (במקביל).

Cache Behavior

```
int x[2][128];
int i;
int sum = 0;

for (i = 0; i < 128; i++) {
    sum += x[0][i] * x[1][i];
}
```

המערך **x** תופס:

2 * 128 * 4 = 1024 בתים

ומתחיל בכתובת **0**.

מהו אחוז ההחטאה (**Miss**) במקרים הבאים:

1. המטמון הוא **Direct Mapped**
   גודל המטמון 512 בתים.
   מכיל 32 בלוקים בגודל 16 בתים.
2. המטמון הוא **Direct Mapped**
   גודל המטמון 1024 בתים.
   מכיל 64 בלוקים בגודל 16 בתים
3. המטמון הוא **Two-Way Set Associative**
   גודל המטמון 512 בתים.
   מכיל 16 קבוצות, בכל קבוצה 2 בלוקים בגודל 16 בתים.
   LRU replacement policy

במקרה 3:

האם הגדלת המטמון תגדיל את הפגיעה ?

האם הגדלת הבלוק תגדיל את הפגיעה ?

(100%, 25%, 25% , לא, כן)

Cache Misses

- Cold (compulsory) miss

הגישה הראשונה לבלוק שנמצא בזיכרון.

- Conflict miss

המטמון גדול אבל יש כמה נתונים שהתכנית פונה לסירוגין וממופים לאותו בלוק במטמון.

- Capacity miss

המטמון קטן מהפקודות והנתונים שהתכנית כעת משתמשת בהם (**working set**).