# תכנות מתקדם
# מצגת 8
# מטמון

# היררכית הזיכרון

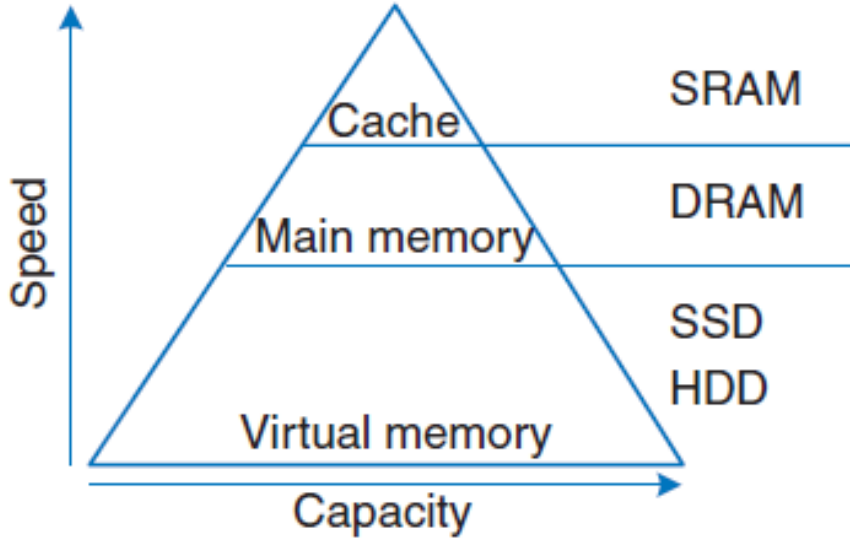
מהירות ביצוע פקודה במעבד (CPU) גדולה בהרבה (פי 10-100) ממהירות גישה לזיכרון הרגיל.

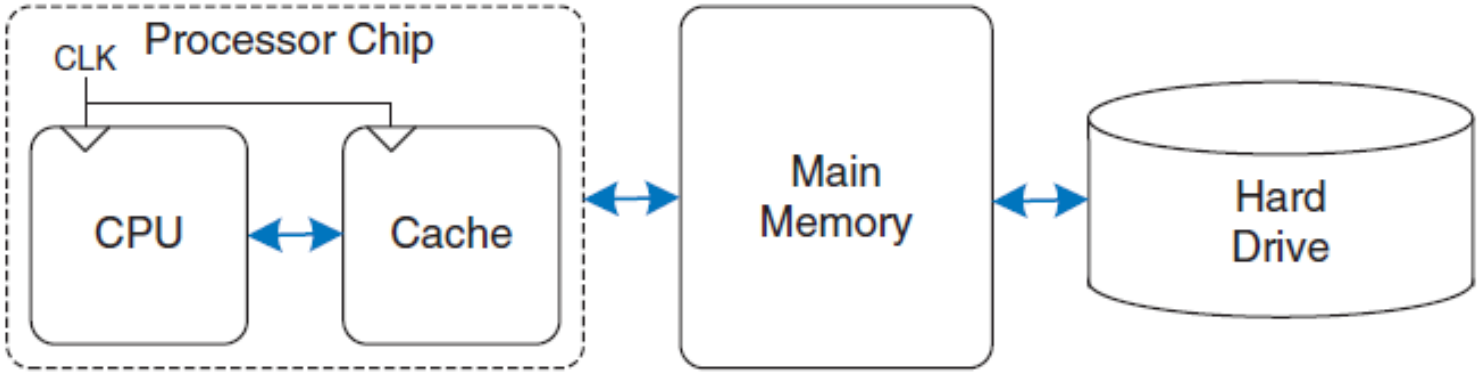
כדי שהמעבד לא יצטרך להמתין, המחשב מכיל זיכרון **קטן מהיר ויקר** הנקרא **מטמון**.

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.

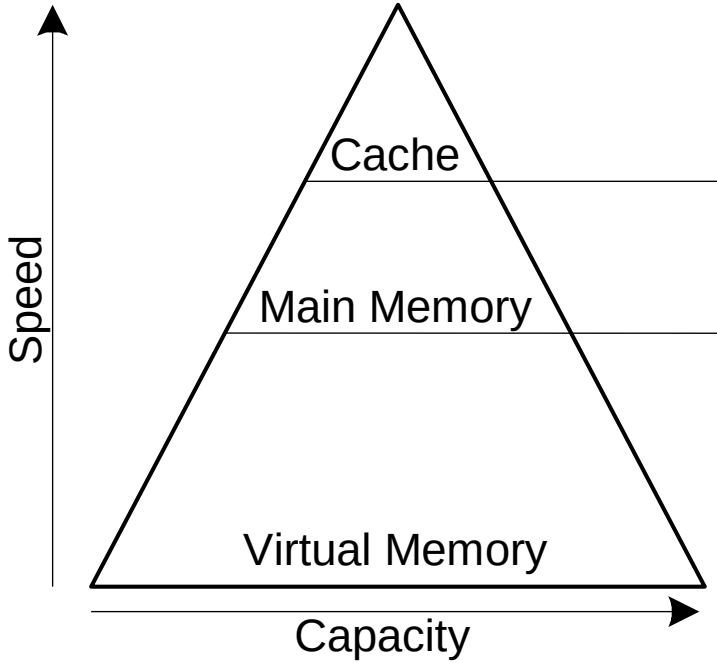
גם הזיכרון הרגיל מהווה מטמון ביחס לדיסק.



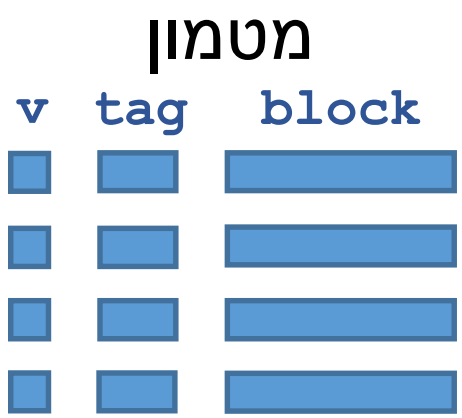
# היררכית הזיכרון

- 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.
  - גודל הבלוק ומספר הבלוקים במטמון הם חזקה של 2.
- 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 נרצה לגשת אל הנתון שוב.
- ההעתקה היא בגודל של בלוק
  - לפי תכונת המקומיות במקום, בקרוב נרצה לגשת לנתונים הסמוכים לנתון.
- אם חלק גדול מהגישות יהיו במטמון, הגישה לזיכרון תהיה מהירה.
- הזיכרון הראשי מחולק לבלוקים שגודלם כגודל הבלוקים של המטמון.

# יחס הפגיעה במטמון

- **Hit**: data found in that level of memory hierarchy
- **Miss**: data not found (must go to next level)
- **Hit Rate**: hits / memory accesses
- **Miss Rate**: misses / memory accesses = 1 – Hit Rate

- Example:
  - A program has 2,000 loads and stores
    - 1,250 of these data values in cache
  - What are the hit and miss rates for the cache?
    - Hit Rate = 1250/2000 = 0.625
    - Miss Rate = 750/2000 = 0.375 = 1 – Hit Rate

# זמן הגישה הממוצע לזיכרון

- Average memory access time:

$$t_{cache} + (MissRate_{cache} * t_{Memory})$$

- Example:

  $t_{cache}$ = 1 cycle

  $t_{Memory}$ = 100 cycles

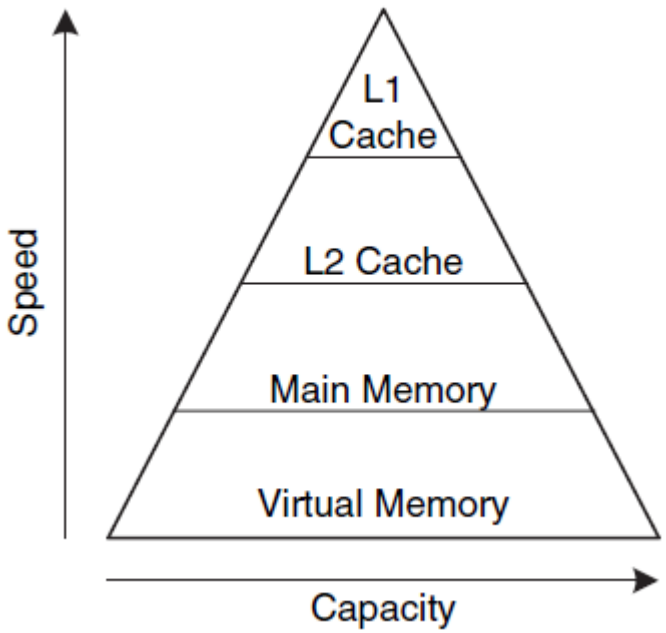
  Miss Rate = 0.375

  - What is the Average memory access time?

    1 + (0.375 * 100) cycles = 38.5 cycles

# מטמון עם כמה רמות

- Larger caches have lower miss rates.
- Expand memory to multiple levels of caches
- Level 1: small and fast (e.g. 16 KB, 1 cycle)
- Level 2: larger and slower (e.g. 256 KB, 2-6 cycles)




Example:

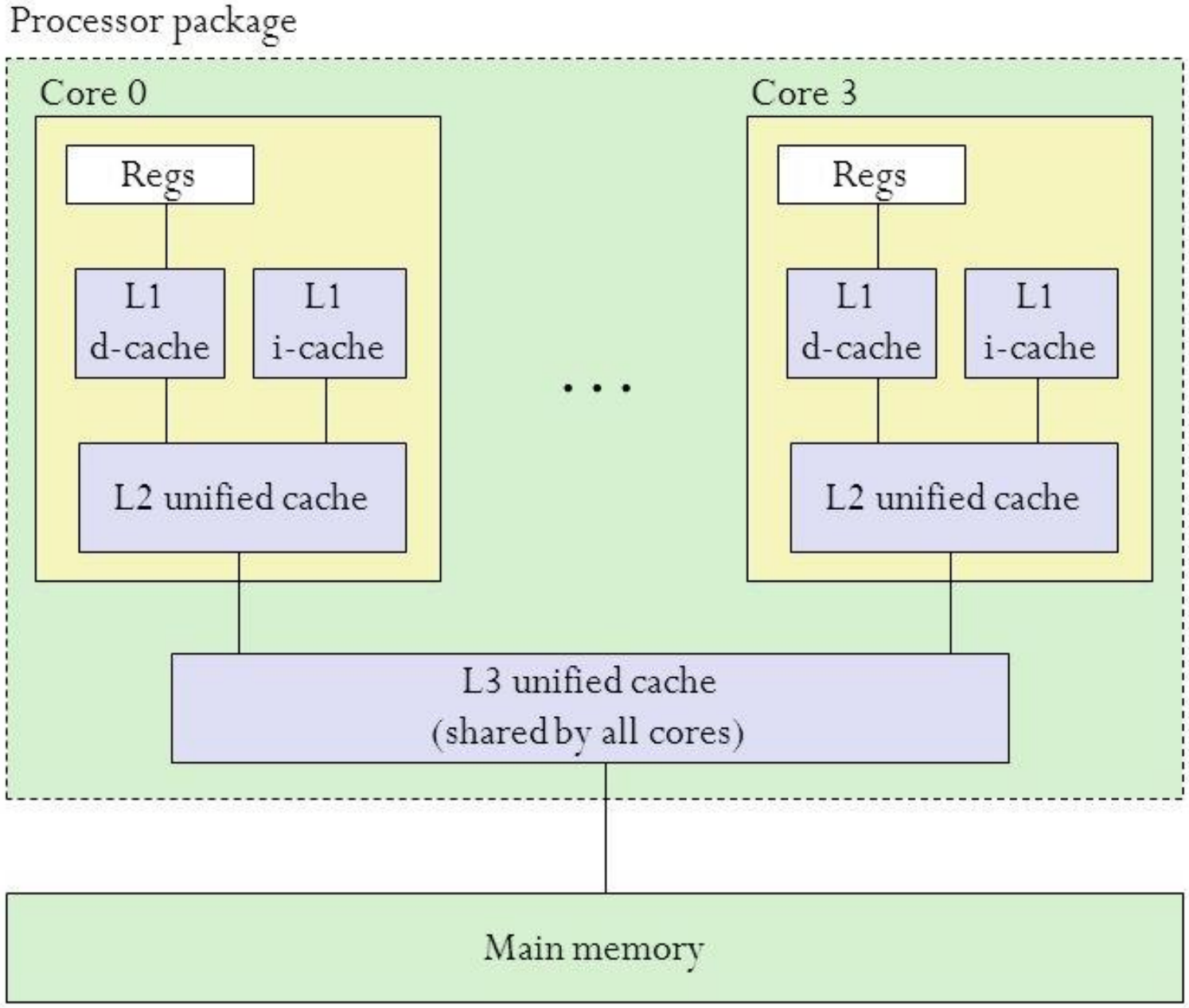
What is the average memory access time with 3 levels of hierarchy

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



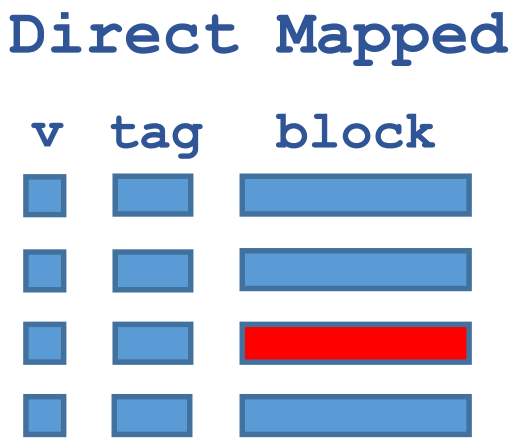

L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

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.

# מטמון במיפוי ישיר

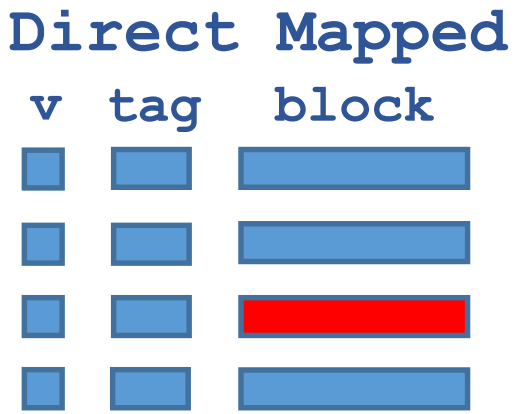




- הזיכרון הראשי מחולק לבלוקים שגודלם כגודל הבלוקים של המטמון.
- לאיזה בלוק במטמון יוכנס בלוק מהזיכרון?
  - לכל בלוק בזיכרון יש מספר, מספר הבלוק של בית כלשהו בזיכרון הוא החלוקה של כתובת הבית בגודל הבלוק.
  - מספר הבלוק במטמון אליו ימופה הבלוק בזיכרון, הוא שארית החלוקה של מספר הבלוק בזיכרון במספר הבלוקים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

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


איך נדע לפי הביטים של כתובת הנתון לאיזה בלוק יוכנס ?

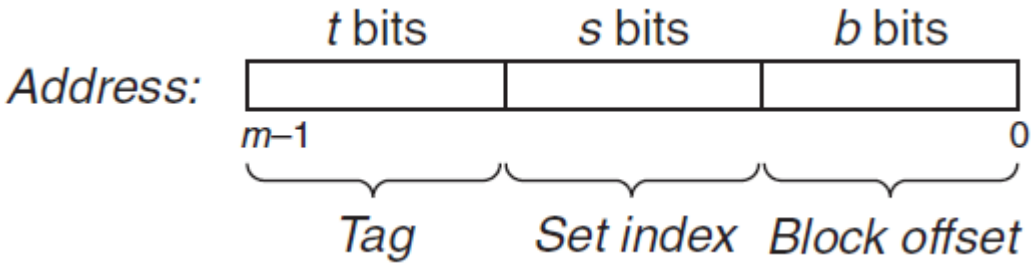
איך נדע לפי הביטים אם הנתון נמצא במטמון ?

- 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.

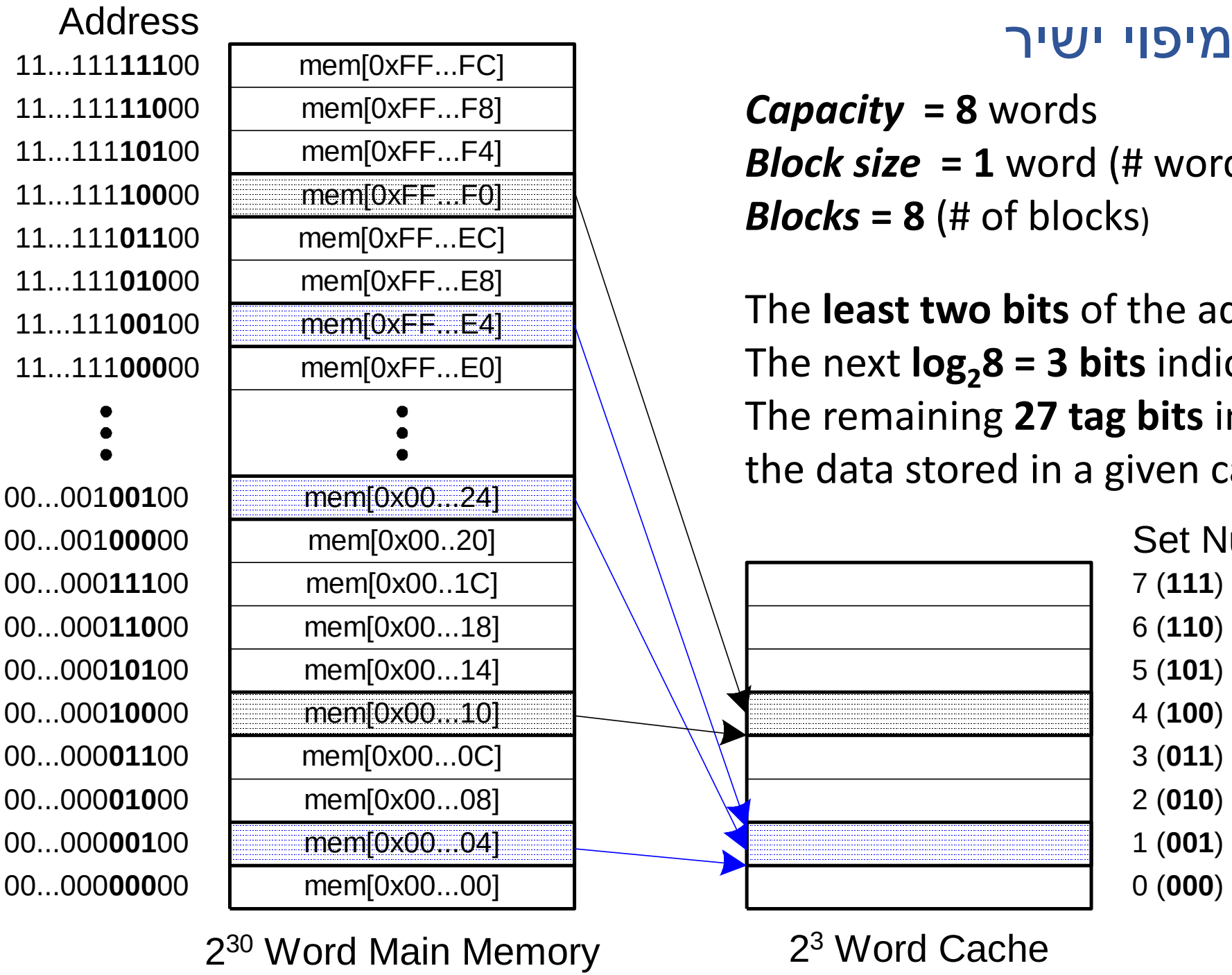
tag bits: m – (s + b) הביטים הנותרים משמאל יוצמדו לכל בלוק שיוכנס למטמון ובאמצעותם נוכל לדעת לאיזה בלוק בזיכרון הוא שייך.



# מטמון במיפוי ישיר

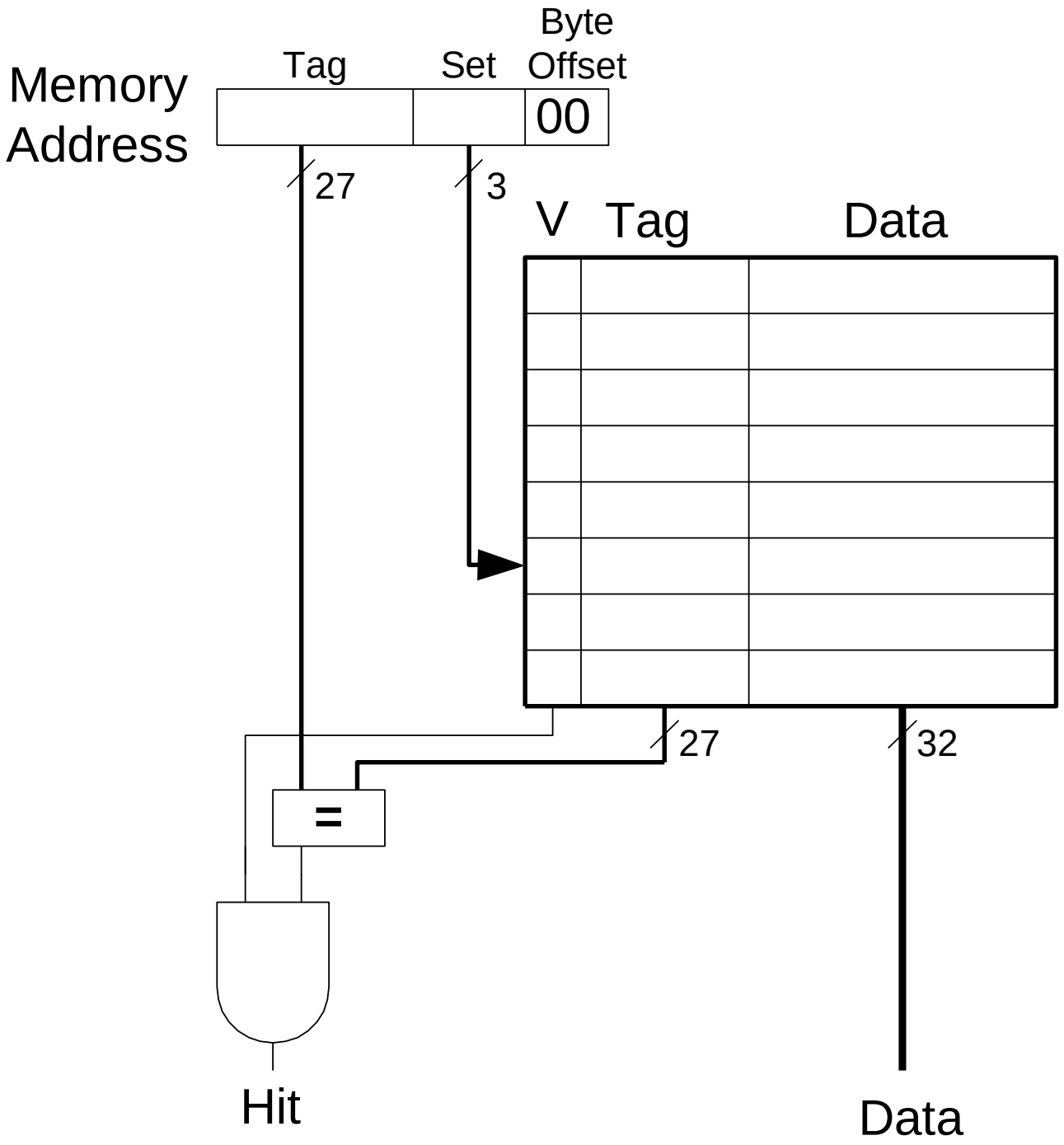
***Capacity* = 8** words
***Block size* = 1** word (# words brought at once)
***Blocks* = 8** (# of blocks)

The **least two bits** of the address are 00
The next **$\log_2 8$ = 3 bits** indicate the block
The remaining **27 tag bits** indicate the address of the data stored in a given cache block




The data at addresses 0x00000004, 0x00000024, . . ., 0xFFFFFFE4 map to set 1

# חומרה למטמון במיפוי ישיר



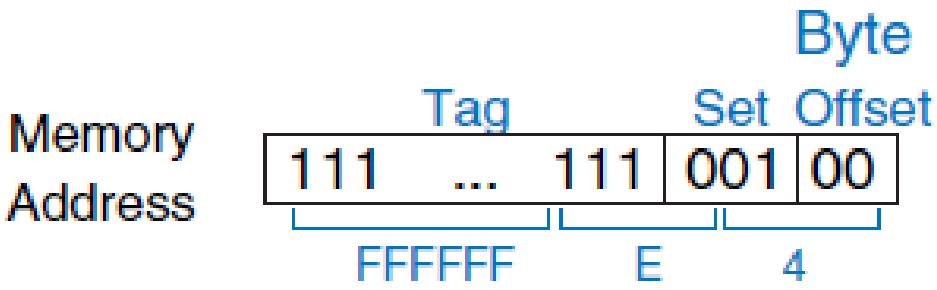

8-entry x (1+27+32)-bit SRAM

The cache uses a **valid bit** for each set To indicate whether the set holds meaningful data

# דוגמה: מטמון במיפוי ישיר

- To what **cache block** (of previous cache) does address 0xFFFFFFE4 map ?

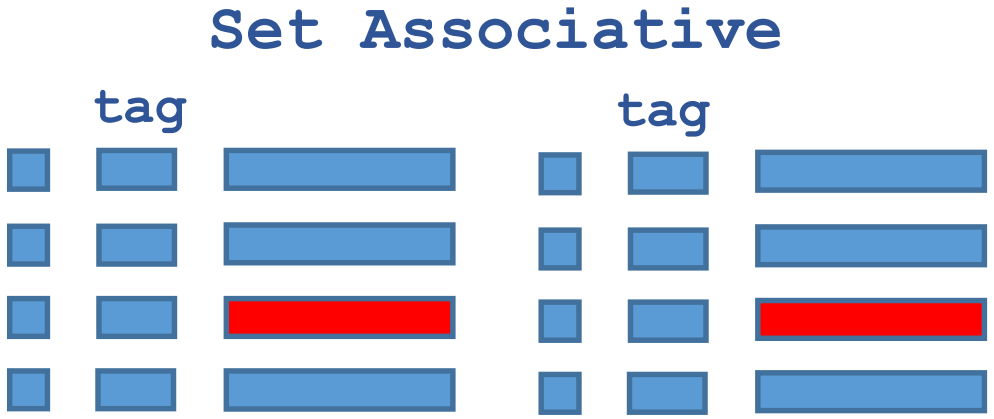
It maps to set 1 and its tag is all 1's




- Find the **number of block and tag bits** for a direct mapped cache with 1024 ($2^{10}$) blocks and a one-word block size. (The address size is 32 bits)

A cache with 1024 blocks requires $\log_2 1024 = 10$ block bits
The two least significant bits of the address are the byte offset
The remaining 32 – 10 – 2 = 20 bits form the tag.

# 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).

# מטמון

במעבד Intel Core i7 הנתונים של מטמון L1 הם:

Capacity: 32 KB, 8-way (lines), Block size: 64 bytes

איך תחולק הכתובת ?

מספר הסטים: C / (E * B) = 32,768 / (8 * 64) = 64 = S

offset bits: 6 ביטים, גודל כל בלוק הוא 64 בתים.

index bits: 6 ביטים, ישנם 64 סטים.

tag bits: 52 הביטים הנותרים.

## Intel Core i7 cache hierarchy

| Cache type | Access time (cycles) | Cache size ($C$) | Assoc. ($E$) | Block size ($B$) | Sets ($S$) |
|---|---|---|---|---|---|
| L1 i-cache | 4 | 32 KB | 8 | 64 B | 64 |
| L1 d-cache | 4 | 32 KB | 8 | 64 B | 64 |
| L2 unified cache | 10 | 256 KB | 8 | 64 B | 512 |
| L3 unified cache | 40–75 | 8 MB | 16 | 64 B | 8,192 |

# Conflict Misses

נניח שמטמון מכיל שני סטים כל אחד בגודל 16 בתים.

המטמון הוא Direct-Mapped, כל סט מכיל בלוק אחד.

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 סט.

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

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

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

ומה יהיה אז אחוז הפגיעה ? 75%

# Conflict Misses

```
typedef int arr[2][2];
void
transpose(arr dst, arr src)
{
  int i, j;
  for (i = 0; i < 2; i++)
    for (j = 0; j < 2; j++)
      dst[j][i] =
      src[i][j];
}
```

נניח שמטמון מכיל שני סטים, כל אחד בגודל 8 בתים.

המטמון הוא Direct-Mapped, כל סט מכיל בלוק אחד.

המערך src תופס 4 * 4 = 16 בתים ומתחיל בכתובת 0.

המערך dst מתחיל אחריו ותופס את 16 הבתים הבאים.

עבור כל גישה למערכים לציין אם הם פגיעה או פספוס. (ומאיזה סוג)

מה תהיה התוצאה אם נכפיל את מספר הסטים במטמון.

# 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 הפספוס (**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**).