מבוא למחשבים
Lecture 6

Cache

Dr. Ron Shmueli

חלק נכבד מהשקפים מבוסס על הספר:

- Heuring and Jordan: "Computer System Design and Architecture", **Prentice Hall**, 2004
- Chapter 6

הזיכרון כמערכת היררכית

- מיבנה היררכי - מאפשר ליצור זיכרון מהיר גדול וזול יחסית
- הערכת ביצועים פשוטה - מחזורי המתנה גדלים בסדר גודל לרמה.
- העברת מידע בין רמות בעזרת חומרה

קטן יותר,
מהיר יותר,
יקר יותר

גדול יותר,
איטי יותר,
זול יותר

הזיכרון כמערכת היררכית

- זכרון מטמון – cache זיכרון קטן ומהיר, המכיל עותק חלקי של המידע שסביר להניח שה CPU ייגש אליה.
- **סבירות הגישה לכתובת בזיכרון Locality of Reference**
- **מקומיות בזמן - Temporal Locality** – האינפורמציה שבה נשתמש בעתיד הקרוב סביר שהשתמשנו בה לאחרונה .
- לדוגמא: תוכנית מבלה 90% מהזמן ב- 10% מהקוד – לולאות. משתנים מסוימים מעודכנים פעם אחר פעם. לדוגמא – מיון.
- **מקומיות במקום - Spatial Locality** – אם ניגשנו למילה בזכרון סביר להניח שניגש למילה שכנה בקרוב.
- לדוגמא: קטעי קוד, סביר מאד שנצטרך גם את הפקודה הבאה ואילו שאחריה. נתונים – נתונים קשורים שמורים יחד (מערכים).

לוקאליות – דוגמא

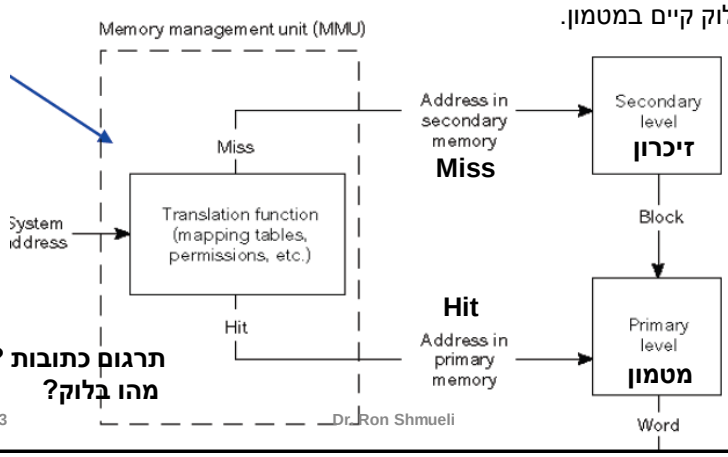
```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

דוגמאות ללוקאליות:

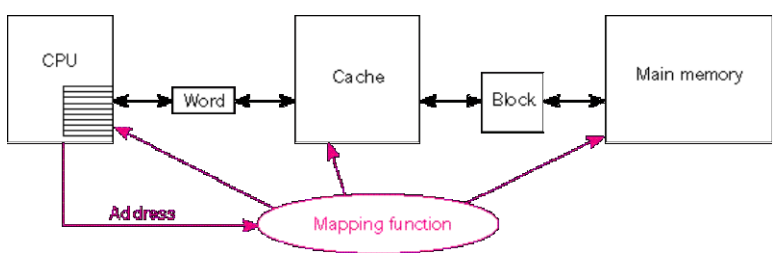
- מידע
  - נפנה לאיברי מערך אחד אחרי השני **לוקאליות מקום**
  - בכל פעם נפנה ל sum **לוקאליות זמן**
- פקודות
  - נפנה לפקודות הכתובות ברשימה – זו אחר זו. **לוקאליות מקום**
  - שוב ושוב נסתובב בלולאה while **לוקאליות זמן**

זיכרון היררכי של שתי רמות

- **גישה לזיכרון מנותבת קודם כל למטמון –**
- **נמצאה המילה המטמון (Hit)** – המילה מובאת ל CPU.
- **לא נמצאה המילה במטמון (Miss)**– הבא בלוק מהזיכרון למטמון והחלף בלוק קיים במטמון.



תרגום כתובות ?
מהו בלוק?



- The cache mapping function is responsible for all cache operations:
  - Placement strategy: where to place an incoming block in the cache
  - Replacement strategy: which block to replace upon a miss
  - Read and write policy: how to handle reads and writes upon cache misses.
- Mapping function must be implemented in hardware. (Why?)

Hits and misses; paging; block placement

Hit: the word was found at the level from which it was requested.

Miss: the word was not found at the level from which it was requested. (A miss will result in a request for the block containing the word from the next higher level in the hierarchy.)

Hit ratio (or hit rate) = h = $\frac{\text{number of hits}}{\text{total number of references}} = \frac{Hit\#}{Hit\#+Miss\#}$

Miss ratio: 1 - hit ratio

t_p = primary (cache) memory access time.

t_s = secondary memory access time

Average Access time, $t_a = h \cdot t_p + (1-h) \cdot t_s$.

Speed Up = $\frac{Twithout}{Twith}$

דוגמא

נתון

h = 0.85 ts = 1.2 μs, tp = 0.4 μs,

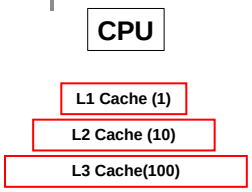
מצא זמן גישה ממוצע ta = h • tp + (1 - h) • ts

מצא Speed Up

Ta = 0.85*0.4 + (1 - 0.85) * 1.2 = 0.52 μ s

SU=1.2/0.52 = 2.37

זיכרון מטמון היררכי



- הוספת רמות לצורך הורדת עלות.
  הרמה הקרובה ל CPU מהירה ביותר (קטנה ביותר)

האם יש הגיון שרמה L1 ו – L2 יהיו בעלות אותו הגודל?

האם יתכן שמידע ברמה L1 לא יהיה ברמה L2 ?

<u>עקרון ההכלה:</u> מידע שנמצא ברמה גבוהה של זיכרון (קרובה יותר ל CPU), ימצא גם בכל הרמות שמתחתיו

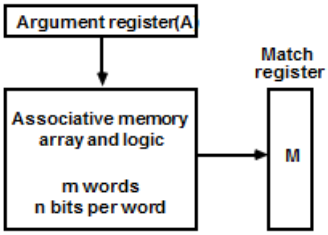
נקודות עיקריות בתכנון זיכרון מטמון

- **<u>מנגנון תרגום כתובת</u>**:
  תרגום כתובת המעבד לכתובת במטמון
- **<u>קביעת גודל בלוק</u>**:
  בלוק קטן – קצב החטאות גדל.
  בלוק גדול – הגדלת זמן העברה
- **<u>אסטרטגיית השמה Block Placem</u>** –
  מיקום הבלוק שהובא מהזיכרון במטמון (האם בכל מקום או במקום ספציפי)
- **<u>זיהוי בלוק Block Identification -</u>**
  כיצד מזהים בלוק כאשר הוא נדרש .
- **<u>מדיניות החלפה Block Replacement</u>** -
  איזה בלוק יוחלף במקרה של החטאה
- **<u>אסטרטגיית כתיבה Write Strategy</u>**
  האם לכתוב למטמון או ישירות לזיכרון

זיהוי והשמה של בלוק

- מיפוי אסוציאטיבי מלא Associative mapped
  - ניתן לשים כל בלוק בכל מקום במטמון
- מיפוי ישיר direct mapped
  - לכל בלוק יש רק מקום אחד בו הוא יכול להיות במטמון
- N-Way Set Associative
  - לכל בלוק יש N מקומות בהן הוא יכול להיות במטמון

זיכרון אסוציאטיבי



- **גישה לזיכרון ע"פ תכולה** וקבלת כתובת **(או ביט שמזוהה עם הכתובת)**

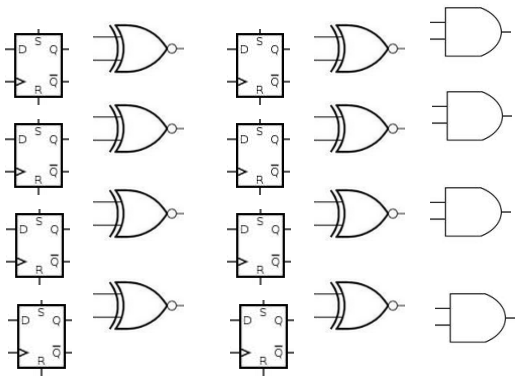
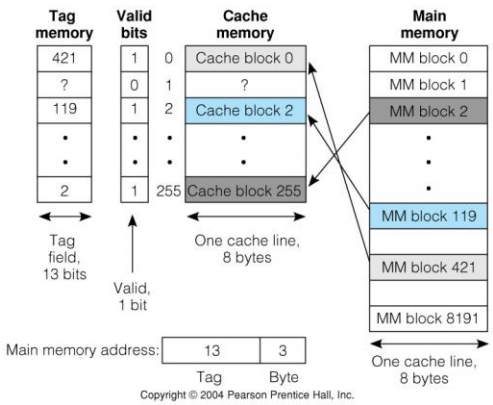


Fig 7.32 Associative mapped caches

Associative mapped cache model: any block from main memory can be put anywhere in the cache.
Assume a 16-bit main memory.*



*16 bits, while unrealistically small, simplifies the examples

Fig 7.33 Associative cache mechanism

Because any block can reside anywhere in the cache, an *associative,* or *content addressable memory* is used. All locations are searched simultaneously.

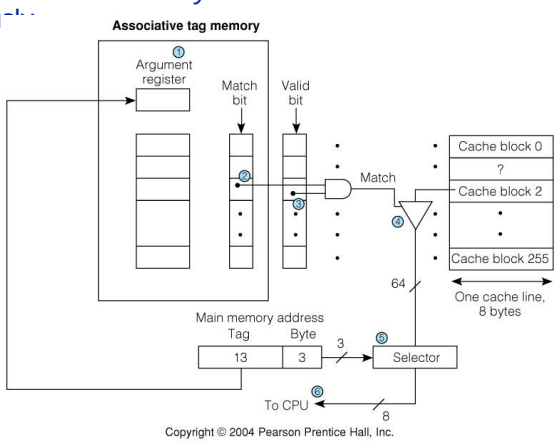
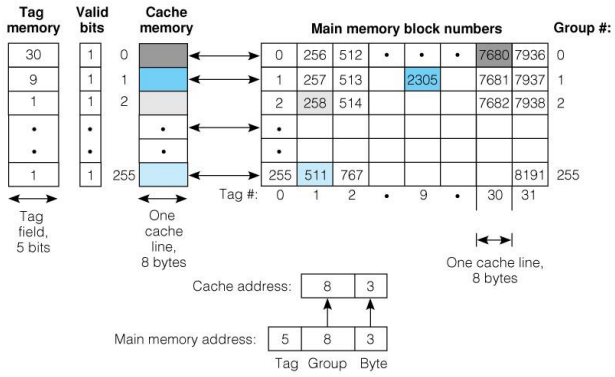


Fig 7.34 The direct mapped cache

Key Idea: all the MM blocks from a given group can go into only one location in the cache, corresponding to the group number.



Now the cache needs only examine the single group that its reference specifies.

Fig 7.35 Direct Mapped Cache Operation

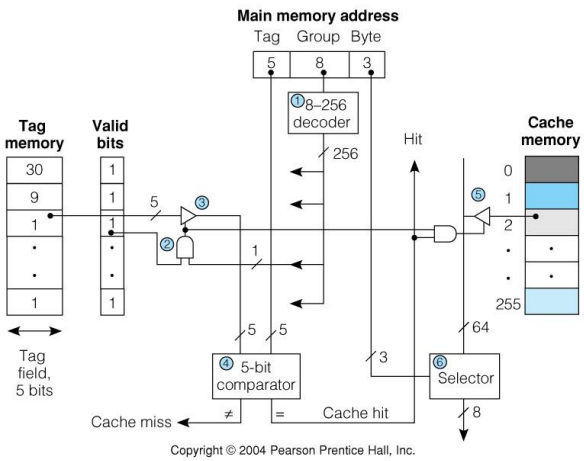
1. Decode the group number of the incoming MM address to select the group

2. If Match AND Valid

3. Then gate out the tag field

4. Compare cache tag with incoming tag

5. If a hit, then gate out the cache line,

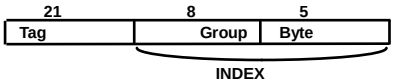


6. and use the word field to select the desired word.

דוגמא קטנה (*)

- גודל מרחב זיכרון מקסימלי לגישה ע"י המעבד 2^{34}
- מרחב זיכרון מטמון 8K (2^{13}) עם גודל בלוק של 32 בתים = 2^5
- טכניקת מיפוי DIRECT Mapping.
- מה גודל השדות TAG , Group , Byte במילת הכתובת ?

- גודל בלוק 2^5 ← שדה ה Byte בגודל 5 bits
- מספר הבלוקים : Block#=Cache size / Block size = $2^{13}/2^5 = 2^8$ ← שדה ה Group בגודל 8 bits
- שדה ה TAG יתרת הביטים במילה של 34 ← Tag=34-8-5=21bits



(הערה: שדה ה byte /word מצביע על אלמנט הזיכרון הבסיסי לגישה)

Direct mapped caches

•The direct mapped cache uses less hardware, but is much more restrictive in block placement.

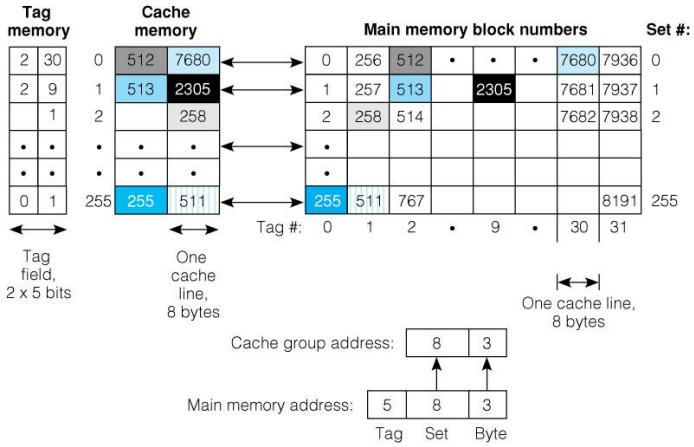
•If two blocks from the same group are frequently referenced, then the cache will “thrash.” That is, repeatedly bring the two competing blocks into and out of the cache. This will cause a performance degradation.

•Block replacement strategy is trivial.

•Compromise - allow several cache blocks in each group– the Block Set Associative Cache. –next–

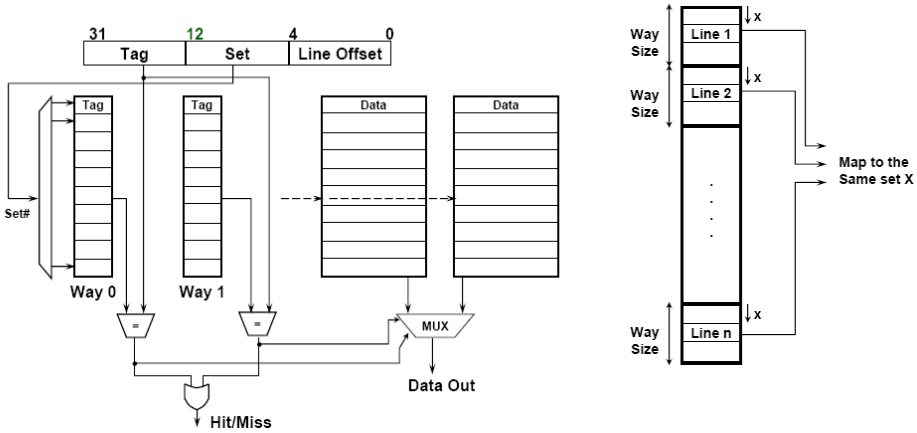
Fig 7.36 2-Way Set Associative Cache

Example shows 256 groups, a set of two per group.
Sometimes referred to as a 2-way set associative cache.



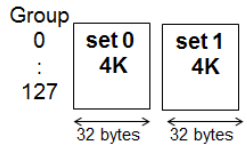
דוגמא – תכנון חומרה ל 2 way (*)

2-Way Cache - Hit Decision



Getting Specific: The Intel Pentium Cache

- The Pentium actually has two separate caches–one for instructions and one for data. Pentium issues 32-bit Main Mem. addresses.
  - Each cache is 2-way set associative
  - Each cache is 8K=2^{13} bytes in size (Data only)
  - 32 = 2^5 bytes per line.
  - Thus there are 64 or 2^6 bytes per set, and therefore $2^{13}/2^6$ or 2^7=128 groups
  - This leaves 32-5-7 = 20 bits for the tag field:



Tag	Set (group)	Word
20	7	5

31 0

Cache Read and Write policies
מדיניות קריאה וכתיבה לזיכרון מטמון

- **Write cache hit policies**
  - **Write-through**–updates both cache and MM upon each write.
    - יתרון: הזיכרון והמטמון מסונכרנים.
    - חסרון: איטי בכתיבה לזכרון
  - **Write back**–updates only cache. Updates MM only upon block removal.
    - "Dirty bit" is set upon first write to indicate block must be written back.
    - יתרון : מהיר גישה למטמון בלבד, יכול לצבור מספר עדכונים לאותה כתובת .
    - חסרון: הזכרון והמטמון לא מסונכרנים

Cache Read and Write policies
מדיניות קריאה וכתיבה לזיכרון מטמון (המשך)

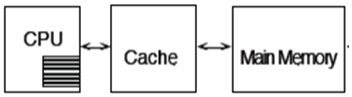
- **Write cache miss policies**
  - **Write allocate -** bring block into cache, then update
  - **Write - no allocate** - write word to MM without bringing block into cache.
- **Read cache miss policies** - **bring block in from MM**
  - **No Re Read** Either forward desired word as it is brought in, or
    - **זמן קריאה = זמן הבאת הבלוק**
  - **Re Read** Wait until entire line is filled, then repeat the cache request.
    - **זמן קריאה = זמן הבאת הבלוק + זמן קריאה רגילה**

Block replacement strategies

- Not needed with direct mapped cache
- FIFO - הבלוק שנמצא הכי הרבה זמן מוחלף (חוסר התאמה בן שימוש לזמן שהייה)
- Least Recently Used (LRU)
  - מונה מיוחס לכל בלוק בכל יחידת זמן המונים גדלים – גישה לבלוק מאפסת את המונה. יוחלף הבלוק עם המונה הגבוה ביותר
- Random replacement - replace block at random.
  - Even random replacement is a fairly effective strategy.

דוגמא

- נתון מטמון L עם המאפיינים הבאים:
  - מדיניות כתיבה : Write Back
  - זמן קריאה מהמטמון 10ns
  - miss rate = 10%
  - במצב הנתון 60% מה dirty bits במצב on
  - מדיניות כתיבה NoReRead
  - זמן קריאה או כתיבה לזיכרון הראשי 100ns
- מה הזמן הממוצע לקריאת מילה מהזיכרון ta ???



$t_a = h \cdot t_p + (1-h) \cdot t_s$.

t_a = 0.9 * 10ns + 0.4* 0.1* 100ns+ 0.6*0.1*200ns

hit rate; dirty 40% off; miss rate; dirty 60% on (read+write)

דוגמא ממבחן

שאלה 2 (30 נקודות)

לתוכנית מחשב הוקצה תחום זיכרון הכולל 8 בלוקים ראשונים המאוחסנים בזיכרון המרכזי.
נתונה התוכנית הבאה:

```
For i=0 upto 4
  For k=0  upto 511
    Read M[128*i+ k ]
  End
End
```

כל בלוק מכיל 128 מילים בנות 32 סיביות כ"א. כמו כן קיים זיכרון מטמון אסוציאטיבי המורכב מ-3 בלוקים. זמן קריאת מילה מהמטמון למעבד הוא 0.3T. זמן העברת בלוק מהזיכרון הראשי לזיכרון המטמון הוא 3T, זמן קריאה של מילה המצויה בזיכרון המרכזי אל המעבד (ללא מטמון) הוא 5T.
הנח כי זכרון המטמון ריק לפני תחילת הריצה.

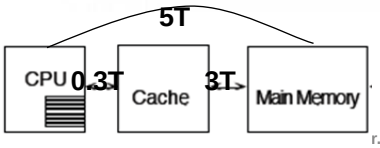
2.1 (15 נקודות) חשב (במדוייק, ללא קרובים) את יחס הפגיעה (hit ratio) בהינתן זיכרון המטמון והתוכנית הרשומים מעלה לפי מנגנוני ההחלפה FIFO ו- LRU.

2.2 (3 נקודות) חשב את הזמן הממוצע לקריאה עבור התוכנית וזיכרון המטמון הנתונים לפי מנגנוני ההחלפה FIFO ו- LRU .

2.3 (2 נקודות) חשב את מקדם האצה (speed up) עם וללא זיכרון מטמון לפי מנגנוני ההחלפה FIFO ו- LRU .

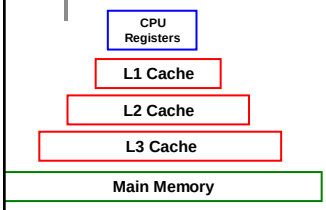
2.4 (5 נקודות) הסבר וחשב כיצד שינוי של מנגנון ההחלפה ל-LIFO היה משנה את יחס הפגיעה (hit ratio) .

2.5 (5 נקודות) מה יהיה יחס הפגיעה (hit ratio) שיתקבל בהנחה שקיים זיכרון מטמון אסוציאטיבי המורכב מ- 2 בלוקים. (FIFO ≠ LRU)



להניח -לפקודות ונתונים מטמון נפרד
מדיניות קריאה NoReRead
מתייחסים רק לנתונים

דוגמא 1 +הסבר על היררכית זיכרון (1)



- במחשב C123 הותקנה היררכית זיכרון המכילה 3 רמות של cache, המשמרות את עקרון ההכלה, כמתואר בשרטוט הבא:

ה- CPU יזם קריאה מכתובת m. בקשה זאת גררה שרשרת ארועים בכל המטמונים.

הנח שלכל המטמונים אותו גודל שורה וכולם מאורגנים ב- K-way. כמו כן הנח שכל המטמונים היו מלאים בעת בקשת הקריאה. מה היה המצב בהיררכיית המטמונים שהוביל לצורך בהחלפת שורה ב- L3?

תשובה:

אם יש צורך להחליף שורה ב- L3 מכאן שהיו 3 החטאות בכל שלוש הרמות של המטמון.

אילו מהאירועים הבאים אפשריים, אילו לא ולמה?

כל האירועים מתוארים לפי סדר: קודם ב- L1 אח"כ L2 ובסוף ב- L3. M=Miss H=Hit ומשבצת ריקה מסמלת שלא קרה דבר (כלומר לא בוצעה פניה למטמון הזה).

L1	L2	L3	אפשרי?	נימוק
H	H	H	לא	אחרי hit ברמה מסויימת לא פונים לרמה שמתחתיה.
H	M		לא	אחרי hit ברמה מסויימת לא פונים לרמה שמתחתיה. בנוסף יש כאן סתירה לעקרון ההכלה.
M	H		כן	המידע נמצא ברמה השנייה
	M	H	לא	הפנייה לרמות עפ"י סדר. חייבים לעבור דרך L1.
H			כן	המידע נמצא ברמה הראשונה
M	M	M	כן	המידע נמצא רק בזכרון הראשי
M	H	M	לא	אחרי hit ברמה מסויימת לא פונים לרמה שמתחתיה. בנוסף יש כאן סתירה לעקרון ההכלה.
M	H	H	לא	אחרי hit לא פונים לרמות שמתחת.
M	M	H	כן	המידע נמצא ברמה השלישית
M	M		לא	חסרה פניה לרמה L3 אחרי ההחטאות ברמות L1 ו- L2
M		H	לא	הפנייה לרמות עפ"י סדר. חייבים לעבור דרך L2.

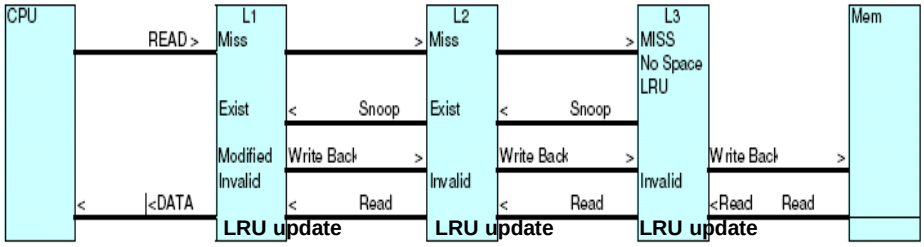
המשך דוגמא 1 - עיקרון הכלה

עקרון ההכלה: מידע שנמצא ברמה גבוהה של זיכרון (קרובה יותר ל CPU), ימצא גם בכל הרמות שמתחתיו

- **Snoop - במקרה שרמת מטמון מסוימת רוצה למחוק נתון מסוים היא "מסתכלת" על הדרגה שמעל ובודקת האם הנתון נמצא גם שם, ואם כן יש למחוק אותו קודם כדי לשמור על עקרון ההכלה.**

המשך דוגמא: תיאור אפשרי של גישה לזיכרון היררכי

תאר (באופן איכותי) את שהתרחש מרגע גילוי הצורך בהחלפת השורה ב-L3 ועד התייצבות תוכן היררכיית הזיכרון ומסירת תוכן כתובת m למעבד (קרי – עד למאוחר שבין שני אירועים אלו). ניתן להניח שכל המטמונים עובדים לפי מדיניות write-back, מדיניות LRU ושומרים על עקרון ההכלה.



דוגמא 2

- נתון מטמון בעל המאפיינים הבאים:
  - גודל (data area) – 64 בתים.
  - מספר בלוקים – 16
  - אורך כתובת לגישה – 30 סיביות.
  - ארגון 2 way set associative.
  - לכל בלוק מוצמדות גם סיביות valid ו- dirty.
  - מדיניות החלפת בלוקים – LRU
- א. כמה סה"כ סיביות זיכרון מכיל המטמון (כולל כל השדות) ?

דוגמא 2 (2)

גודל ה- DATA הוא 64 בתים, ומספר הבלוקים הוא 16 מכאן שכל בלוק הוא 4 בתים. לכן השדה ה Byte (displacement)המצביע על מילה בבלוק בגודל של 2 סיביות.

המטמון מאורגן בשיטת way set associative, 2 ו- סה"כ יש 16 בלוקים, מכאן שבכל way יש 8 שורות ולכן שדה ה- Group יהיה בגודל 3 סיביות.

נתון כי אורך כתובת לגישה למטמון היא 30 ביט, ולכן גודל שדה ה- tag יהיה: 25 =2 – 3 – 30 סיביות.

עבור way set associative 2 מספיק ביט יחיד לכל set ע"מ לנהל מדיניות של LRU. ובסה"כ:

CacheSize = DataSize + (tag + valid + dirty) * #ofBlocks + LRUbits =

= 64*8 + (25 + 1 + 1) * 16 +8 = 952bit

V(1)	M(1)	Tag(25)	Data(32)

LRU(1)

V(1)	M(1)	Tag(25)	Data(32)	#index
				0
				1
				2
				3
				4
				5
				6
				7

דוגמא 2 (3)

מהי מדיניות הכתיבה ומדוע ניתן להניח כך?

מדיניות הכתיבה היא write back, הנחנו כך כאשר החלטנו ליישם ביט dirty.

למטמון בעל מדיניות כתיבה אחרת אין צורך בביט זה, שתפקידו הוא לסמן האם יש לעדכן את השורה בזיכרון הראשי כאשר יהיה צורך לפנותה מהמטמון.

דוגמא 2 (4)

למטמון הנתון נשלחה סדרת הכתובות הבאה (גישה לבתים):

Addresses: 5 7 1 4 36 8 100 6 4 12 36 12 68 5 7

בהנחה שמדיניות הפינוי היא LRU והמטמון מלא ב- data של בלוקים שאינם מופיעים בסדרה דלעיל מצא:

כמה החטאות יווצרו?

מה מכיל המטמון בתום הסדרה (קרי – אילו בלוקים בכל set)?

מיבנה כתובת

TAG(25)	INDEX(3)	Word(2)

כדי לפתור את השאלה נצטרך לחשב לאיזה index במטמון נכניס את הבלוק המבוקש מהזכרון הראשי. את שדה ה- index ניתן לחשב לפי
index = floor(ADDRESS / line length) mod (lines per way)

הסבר: חלוקת הכתובת ב- 4 (אורך השורה) תוריד את שדה ה- disp, וביצוע mod8 יתן את שלושת סיביות ה- LSB של מספר הבלוק שהן בעצם ה- set.

דוגמא 2 (5)

	add	tag	set	Hit/ miss	Way	נימוק
1	5	0	1	m	0	עפ"י הנתון השורה לא נמצאת
2	7	0	1	H	0	בפקודה הקודמת הבאנו שורה שמכילה גם את כתובת 7 (וגם את 4 ו- 6)
3	1	0	0	M	0	עפ"י הנתון השורה לא נמצאת
4	4	0	1	H	0	הנתון הובא ב- 1
5	36	1	1	M	1	הנתון לא נמצא. היות וכבר הובא נתון בעל set=1(ב- 1), way = 1.
6	8	0	2	m	0	עפ"י הנתון השורה לא נמצאת
7	100	3	1	M	0	בפעם האחרונה בה קבלנו set = 1 השורה הובאה ל- way1 ולכן עפ"י LRU צריך לפנות את way0.
8	6	0	1	M	1	אמנם השורה המכילה את כתובת 6 הובאה כבר ב- (2) אבל ב- (7) החלפנו אותה בשורה חדשה ולכן החטאה. בפעם האחרונה בה קבלנו set = 1 השורה הובאה ל- way0 ולכן עפ"י LRU צריך לפנות את way1.
9	4	0	1	H	0	ב- (8) שוב הבאנו את השורה של 4-7
10	12	0	3	M	0	עפ"י הנתון השורה לא נמצאת
11	36	1	1	M	0	בפעם האחרונה בה קבלנו set = 1 השורה הובאה ל- way1 ולכן עפ"י LRU צריך לפנות את way0.
12	12	0	3	H	0	הנתון כבר הובא ב- (10).
13	68	2	1	M	1	בפעם האחרונה בה קבלנו set = 1 השורה הובאה ל- way0 ולכן עפ"י LRU צריך לפנות את way1.
14	5	0	1	M	0	אמנם השורה המכילה את כתובת 5 הובאה שוב ב- (8) אבל ב- (13) החלפנו אותה בשורה חדשה ולכן החטאה. בפעם האחרונה בה קבלנו set = 1 השורה הובאה ל- way0 ולכן עפ"י LRU צריך לפנות את way1.
15	7	0	1	H	1	ב- (14) שוב הבאנו את השורה של 4-7

דוגמא 2 (6)

סה"כ 4 פגיעות, 10 החטאות

				LRU					#set
				0	0	1	2	3	0
68	69	70	71	0	4	5	6	7	1
				0	8	9	10	11	2
				0	12	13	14	15	3
									4
									5
									6
									7

שיפור ה-hit rate של המטמון ע"י קומפיילר

- דוגמא - לאפשרות של קומפיילר לשפר hit rate ניתן לראות בלולאות מקוננות אשר סדר הגישות בהן לזיכרון הוא לא סדרתי.
- אם המטמון אינו גדול דיו לאחסן את כל הנתונים עלול להיות דפדוף רב של שורות.
- לעיתים החלפת סדר הלולאות עשוי לפתור את הבעיה.

- Before

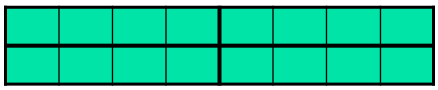
```
for (j=0; j<100; j++)
        for (i=0; i<5000; i++)
                x[i][j]=2*x[i][j];
```

- After

```
for (i=0; i<5000; i++)
        for (j=0; j<100; j++)
                x[i][j]=2*x[i][j];
```


דוגמא

- נתון מערך של 2 שורות של 2 בלוקים. כל בלוק 4 בתים.
- התכנית עוברת על המערך פעמיים, בית – בית, לאורך הטורים.
- המטמון הינו מטמון direct mapped של שני בלוקים.

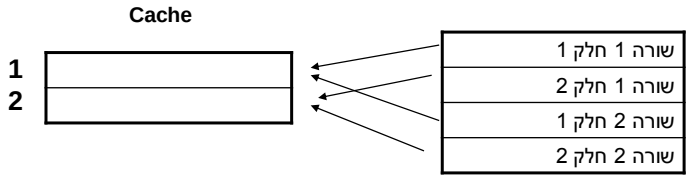


- **כמה החטאות ייווצרו במהלך התכנית ומאילו סוגים?**
- **הנח:**

1. **טרם הרצת התכנית אין במטמון שום מידע שקשור אליה.**
2. **המערך מיושר בזיכרון לפי השורות.**
3. **עדכון כל תא לא תלוי בשאר התאים.**

דוגמא (2)

- ראשית, נבדוק לאן ממופים הבלוקים שבמערך:



1	2
1	2

•אין הכרח שהבלוק הראשון ימופה לתא הראשון במטמון, אולם היות והמיפוי מחזורי, מובטח כי כל הבלוקים במערך שמסומנים כ- 1 ימופו לאותה שורה במטמון.

דוגמא (3)

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8

עמודה	שורה	ממופה ל- set#	ב- 2 way set associative	סוג החטאה
1	1	1	Way 0 (M)	Compulsory – פעם ראשונה בבלוק
1	2	1	Way 1 (M)	Compulsory – פעם ראשונה בבלוק
2	1	1	Way 0 (H)	Conflict
2	2	1	Way 1 (H)	Conflict
3-4		1		כנ"ל
5	1	2	Way 0 (M)	Compulsory – פעם ראשונה בבלוק
5	2	2	Way 1 (M)	Compulsory – פעם ראשונה בבלוק
6-8		2		Conflict – בדיוק כמו בעמודות 2-4
1	1	1	Way 0 (M)	Capacity – גם ב- fully associative היינו מקבלים החטאה
1	2	1	Way 1 (M)	Capacity – גם ב- fully associative היינו מקבלים החטאה
2-4		1		Conflict
5	1	2	Way 0 (M)	Capacity – גם ב- fully associative היינו מקבלים החטאה
…	…	…	…	…

סה"כ 32 החטאות מתוכן 4 מסוג Compulsory, 4 מסוג Capacity ו- 24 מסוג Conflict

דוגמא (3)

מוצע לשפר את ביצועי התכנית באמצעות מהדר (קומפיילר).

אילו שינויים ניתן לבצע בתכנית (תוך ידיעת נתוני המעבד)?

1. עדכון המערך ברזולוציה של בלוקים.
2. עדכון המערך בשורות.
3. ביצוע שני העדכונים לתא, אחד אחרי השני.

נניח שהמהדר משנה את אופן ריצת התכנית כך שהמערך מעודכן שורה אחר שורה. כמה החטאות ייוצרו ומאילו סוגים?

דוגמא (4)

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8

שורה	עמודה	ממופה ל- set#	ב- fully associative	סוג החטאה
1	1	1	Way 0	Compulsory – פעם ראשונה בבלוק
1	2-4	1		HIT
1	5	2	Way 1	Compulsory – פעם ראשונה בבלוק
1	6-8	2		HIT
2	1	1	Way 0	Compulsory – פעם ראשונה בבלוק
2	2-4	1		HIT
2	5	2	Way 1	Compulsory – פעם ראשונה בבלוק
2	6-8	2		HIT
1	1	1	Way 0	Capacity – גם ב- fully associative היינו מקבלים החטאה
1	2-4	1		HIT
1	5	2	Way 1	Capacity – גם ב- fully associative היינו מקבלים החטאה
1	6-8	2		HIT
...	...	...	...	...

סה"כ 8 החטאות מתוכן 4 מסוג Compulsory, 4 מסוג Capacity