זיכרון מטמון - Cache

מה זו היררכיית זיכרון?
בגישה של היררכיית זיכרון, הזיכרון מחולק למספר רמות – הרמה התחתונה היא הרמה הגדולה והאיטית יחסית והרמה העליונה(cache) היא הרמה הקטנה והמהירה ביותר.
כל המידע ברמת זיכרון נתונה, נמצא בכל אחת מהרמות שמתחתיה.
מעבר של נתונים מרמה אחת לשנייה נעשה ביחידות של בלוקים (ע"מ לנצל את תכונת המקומיות במרחב, ובכך לחסוך בכמות הגישות לזיכרונות איטיים).

למה צריך זיכרון מטמון?
במצב האידיאלי היינו רוצים מעבד עם זכרון יחיד (לפשטות), הגדול ביותר, המהיר ביותר והזול ביותר.
אלא שאילוצים טכנולוגיים גורמים לכך שככל שהזכרון גדול יותר, כך הוא איטי יותר וככל שהזכרון קטן יותר כך הוא מהיר יותר ויקר יותר..
והמטרה היא להשיג מערכת זיכרון מהירה **כמעט** כמו המהירה ביותר, וזולה כמעט כמו הזולה ביותר.

```
1)  for (I=0;I<1000;I++)
2)       X+=arr[I];
```

כלל אצבע – המערכת מבלה 80% מהזמן בביצוע של 20% מהקוד (במקרה זה שורות 1,2)
מקומיות בזמן – תוכניות משתמשות פעם נוספת בנתונים ופקודות שבהם הן השתמשו לאחרונה (בתוכנית שבדוגמא המשתנה I בשימוש פעמים רבות, והפקודות עצמן מובאות שוב ושוב). זיכרון מטמון קטן ומהיר שיכיל את הפקודות המרכיבות את הקוד שבלולאה יהיה ניצול טוב של תכונת המקומיות בזמן.
מקומיות במרחב - ל-2 משתנים שכתובותיהן נמצאות אחת ליד השנייה, תהייה התייחסות קרובה בזמן (כל כתובות המערך arr סמוכות אחת לשנייה). מערכת הזיכרון עובדת ביחידות של בלוקים של Bytes, ובכך מנצלת את תכונת המקומיות במרחב.

נוסחא להערכת ביצועי זיכרון היררכי:

Average Memory Access Time =
hit time + miss rate x miss penalty

hit time – זמן קריאת/כתיבת נתון לבלוק בזיכרון המטמון, כאשר הבלוק נמצא.
miss rate – החלק של הגישות לזיכרון שבהן הבלוק שאליו אנו רוצים לקרוא/לכתוב אינו נמצא ב-cache.
miss penalty – הזמן הלוקח להביא בלוק מהזיכרון הראשי ל-cache.

כדי להבין את אופן פעולת היררכיית זיכרון, נענה לגביה על 4 שאלות:
1) להיכן ב- Cache יכול להתמפות בלוק מהזיכרון?
2) כיצד מוצאים בלוק ב-Cache?
3) איזה בלוק מחליפים במקרה שהבלוק לא נמצא ב-Cache (miss), וצריך להביאו מהזיכרון?.
4) מה קורה בכתיבה?

1) להיכן ב- Cache יכול להתמפות בלוק מהזיכרון?

קיימות 3 שיטות למיפוי בלוק מהזיכרון הראשי ל- Cache

1. מיפוי ישיר (direct mapped) – בלוק מהזיכרון ממופה לבלוק אחד ויחיד בזיכרון המטמון.
   פונקציית המיפוי הינה:

   (מספר הבלוק בזיכרון) mod (מספר הבלוקים ב-cache)

   חסרון עיקרי – אם במקרה משתמשים תכופות ב-2 כתובות שמתמפות לאתו בלוק ב-cache, יווצר מצב שבו נביא בלוק מהזיכרון, ומייד אח"כ נביא בלוק אחר שידרוס אותו וחוזר חלילה, לדוגמא בתוכנית:

```
for (I=0;I<10;I++)
{
        x1+=arr1[I];
        x2+=arr2[I];
}
```

   יתרון עיקרי – פשטות במימוש.
2. Fully associative – בשיטה זו ניתן למפות כל בלוק בזיכרון לכל בלוק ב-cache.
   חסרון עיקרי – סיבוכיות במימוש
   יתרון עיקרי – גמישות רבה בבחירת הבלוק להחלפה.
3. n way set associative – בלוק מהזיכרון ימופה לקבוצת (set) בלוקים ב-cache, בתוך הקבוצה בגודל n בלוקים, ניתן למפות לכל אחד מ-n הבלוקים.
   פונקציית המיפוי של בלוק לקבוצה הינה:

   (מספר הבלוק בזיכרון) mod (מספר הקבוצות ב-cache)

   פיתרון זה מהווה פשרה בין סיבוכיות מימוש לגמישות בבחירת הבלוק להחלפה.

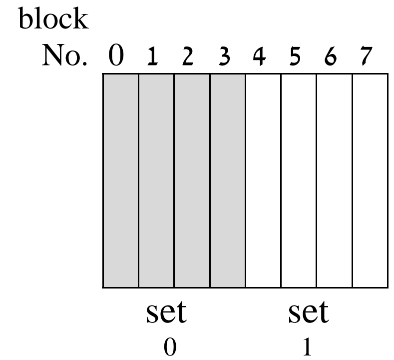
הערה:
ניתן להתייחס למיפוי ישיר כ-1 way set assosiative,
ול- Fully associative כ-M way set associative,
כאשר M הינו מספר הבלוקים ב-cache.

דוגמא למיפוי בלוק מס' 12 בזיכרון הראשי ל-cahce:

4-way set associative

בלוק 12 יכול להתמפות לכל אחד מ-4 הבלוקים שב-set 0.

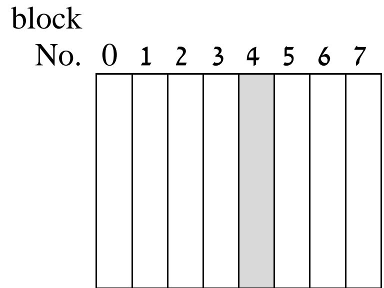
(12 mod 2 = 0)



Direct Mapped:

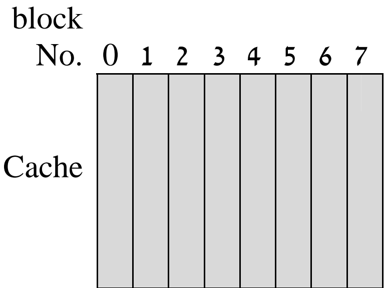
בלוק 12 יכול להתמפות רק לבלוק מס' 4 ב-cache

(12 mod 8 = 4)

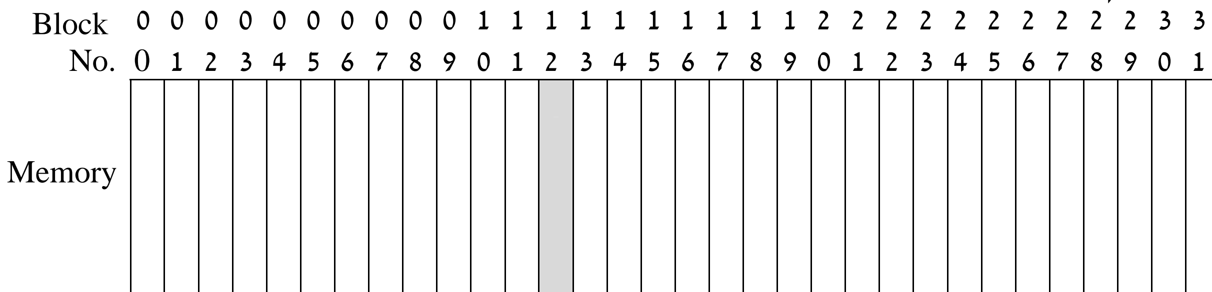


Fully associative:

בלוק 12 יכול להתמפות לכל אחד מהבלוקים ב-cache



הזיכרון הראשי:



2) כיצד מוצאים בלוק ב-Cache?

נניח:

גודל הזיכרון: 2^{32}=4GBytes , לכן כתובת של Byte הינה בת 32 ביטים.

גודל בלוק: 2^3=8Bytes

גודל ה-cache – 8Kbyte = $1024=2^{10}$ בלוקים.

שיטת מיפוי n way set associative

מס' הקבוצות(sets) ב-cache = n/מס' הבלוקים ב-cache

מבנה הכתובת:

Byte Address		
Block Frame (= Block address)		**Offset**
Tag	**Index**	
מה שנשאר	$\log_2$(מספר הקב' ב-cache)	Log_2 (block size) =3 bits

Offset – בוחר את ה-Byte המתאים מתוך הבלוק.

Index - בוחר את הקבוצה (set) המתאימה.

Tag – נשמר יחד עם הבלוק ב-cache.

כאשר מאחסנים בלוק ב- cache שומרים יחד איתו גם valid bit שערכו 1 כאשר יש מידע אמיתי בבלוק, ואת שדה ה-Tag.

אין צורך,כמובן, לשמור את שדה ה-offset, כי כל הבלוק נשמר כיחידה אחת.

אין צורך לשמור את שדה ה-Index, כי ממילא הימצאות של בלוק בקבוצה x ב-cache, מחייבת ששדה האינדקס של אותו בלוק יהיה x.

כדי למצוא Byte ב-cache, נלך אל ה- set המתאים בעזרת ה-Index, נעבור על כל n הבלוקים שב-set (במקביל) ונשווה את ה-Tag של הכתובת הרצוייה ל- Tag ששמור יחד עם הבלוק ב-cache. אם יש התאמה של ה-Tag ו-valid bit=1 אז נמצא ה- Byte וזהו cache hit, אחרת(אם אין התאמה של ה-Tag או ש-valid bit=0) אז ה-Byte הרצוי אינו ב-cache, ויש להביאו מהזיכרון וזהו cache miss.

3) איזה בלוק מחליפים במקרה שהבלוק לא נמצא ב-Cache (miss), וצריך להביאו מהזיכרון?

כאשר המעבד מבקש לקרוא נתון מבלוק שלא נמצא ב-cache, הבלוק מובא מהזכרון ובהנחה שה-cache מלא, יש לבחור איזה בלוק ב-cache ל"זרוק" על מנת לפנות מקום לבלוק החדש.
במקרה של מיפוי ישיר יש רק בלוק אחד ויחיד ואין אפשרות בחירה (מה שמפשט את המימוש),
במקרה של n way set assosiative, יש n בלוקים שניתן להחליף.

אסטרטגיות החלפה עיקריות:

1) בחירה אקראית מתוך n האפשרויות (יתרון - פשטות מימוש).
2) LRU (least recently used) – "זריקת" הבלוק שלא נעשה בו שימוש הכי הרבה זמן (יתרון – miss rate נמוך יותר. חיסרון – נדרש מנגנון למעקב אחרי השימוש בבלוקים)

כדי להינות מ-2 העולמות מממשים בד"כ מדיניות שהיא קירוב של מדיניות ה-LRU.

4) מה קורה בכתיבה?

קיימות 2 מדיניויות כתיבה עיקריות לזיכרון המטמון :

1. **Write through** – כתיבת המידע המעודכן גם ל-cache וגם לזיכרון הראשי.

ב-WT המעבד חייב להמתין עד לסיום הכתיבה של הבלוק לזיכרון הראשי. ניתן להמנע מהמתנה זו ע"י הוספת Write Buffer שמאפשר למעבד להמשיך בביצוע פקודות, בזמן שהחוצץ כותב את הבלוק לזיכרון.

2. **Write Back** – כתיבת המידע המעודכן רק לבלוק ב-cache. הבלוק המתאים בזיכרון יעודכן רק כאשר הבלוק "יזרק" מה-cache.

בכדי לדעת אם בלוק ב-cache השתנה, מוסיפים לכל בלוק dirty bit, שערכו 1 כאשר הבלוק עודכן – ביט זה חוסך כתיבות לזיכרון של בלוקים שלא נעשה בהם כל שינוי.

יתרונות מדיניות ה-Write Back :

- זמן הכתיבה תלוי במהירות ה-cache בלבד.
- מספר כתיבות לאותו בלוק מצריכות כתיבה אחת של הבלוק לזיכרון הראשי.

יתרונות מדיניות ה-Write Through:

- miss בקריאה לא גורמים לכתיבה לזיכרון הראשי של הבלוק ש"נזרק" מה-cache.
- קל יותר למימוש.

במקרה שהבלוק שאותו רוצים לעדכן לא נמצא ב-cache (Write miss), משתמשים באחת מהמדיניויות הבאות :

1. **Write Allocate** – הבלוק הדרוש מועבר מהזיכרון הראשי ל-cache, ולאחר מכן פועלים לפי מדיניות הכתיבה שנבחרה (WB או WT).
2. **No Write Allocate** – הבלוק מעודכן בזיכרון הראשי, אך אינו מועבר ל-cache.

דוגמאות

בכל הדוגמאות נניח :

גודל בלוק - 8 Bytes

גודל זיכרון ראשי – 256 Bytes (= 32 בלוקים)

גודל cache – 64 Bytes (= 8 בלוקים)

מדיניות מיפוי 2 way set associative

מדיניות החלפה LRU (נתעלם ממנגנון המימוש).

דוגמא מס' 1 – Read miss ללא החלפה.

ה- cache ריק מנתונים (בשימוש ראשון). קריאה של נתון מכתובת (Byte) דצימלית 117 :

$(117)_{10} = 01110101_2$

מבנה כל כתובת הינו :

Byte Address		
Block Frame (= Block address)		**Offset**
Tag	**Index**	
8-2-3 = 3bits	$\log_2$(מספר הקבי' ב-cache) =log2(8/2) = 2 bits	Log_2 (block size) = 3 bits

לכן עבור כתובת 117 :

$\text{Offset} = 101_2 = 5_{10}$

$\text{Index} = 10_2 = 2_{10}$

$\text{Tag} = 011_2 = 3_{10}$

$\text{Block Address in main memory} = 01110_2 = 14_{10}$

המצב לפני הקריאה:

ה-cache :

2 way set associative

block No.	0	1	2	3	4	5	6	7
valid bit	0	0	0	0	0	0	0	0
Tag 3 bits	x	x	x	x	x	x	x	x
Block data	x	x	x	x	x	x	x	x
	set 0		set 1		set 2		set 3	

Block No.	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
															בלוק נדרש																	

Index = 2, לכן נבדקים כל (2) הבלוקים שב-set 2, מאחר שה-valid bit=0 יש Read miss.
ה-cache שולח סיגנל stall שעוצר את ה-CPU (עד שתושלם הבאת הנתון מהזיכרון הראשי)
בלוק מסי 14 (בו נמצא 117 Byte) מובא מהזיכרון הראשי ומושם באחד מ-2 הבלוקים שב-set 2.
ה-valid bit של הבלוק ב-cache מתעדכן ל-1, וה-Tag מתעדכן עפ"י ה-Tag של הבלוק(Tag = 3)
מתבצעת חזרה על תהליך הקריאה שהפעם מצליחה והבלוק מועבר ל-CPU, ממנו נלקח הבית ה-5.

אחרי הקריאה:

ה-cache:

2 way set associative

block No.	0	1	2	3	4	5	6	7
valid bit	0	0	0	0	1	0	0	0
Tag 3 bits	x	x	x	x	3	x	x	x
Block data	x	x	x	x	בלוק נדרש	x	x	x
	set 0		set 1		set 2		set 3	

Block No.	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
															בלוק נדרש																	

דוגמא מס' 2 – Read miss עם החלפה וכתיבה

תוכן ה- cache :

block No.	0	1	2	3	4	5	6	7
dirty bit	0	0	1	0	1	1	1	0
valid bit	1	0	1	1	1	1	1	0
Tag 3 bits	0	x	1	2	5	7	4	x
Block data		x				E	F	x
	set 0		set 1		set 2		set 3	

מתבצעת פעולת קריאה של נתון מכתובת 232_{10}. הנח מדיניות כתיבה Write Back.

הנח שבלוק A היה לא שמיש במשך התקופה הכי ארוכה, אחריו B וכן הלאה(C,D,E,F)

$232_{10} = 111_01_000_2$

$\text{Offset} = 000_2 = 0_{10}$

$\text{Index} = 01_2 = 1_{10}$

$\text{Tag} = 111_2 = 7_{10}$

$\text{Block Address in main memory} = 11101_2 = 29_{10}$

הזיכרון הראשי :

Block No.	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
																														בלוק נדרש G		

Index =1 לכן נבדקים כל (2) הבלוקים שב-set 1 , ה-valid bit=1 אך אף אחד מה-Tag של הבלוקים שב-set 1 אינו 7,לכן יש Read miss.

ה-cache שולח סיגנל stall שעוצר את ה-CPU (עד שתושלם הבאת הנתון מהזיכרון הראשי)

בלוק מס' 2 ב- cache נבחר להיזרק (LRU), אך מאחר שה-dirty bit=1, יש לבצע כתיבה של בלוק מס' 2 לזיכרון הראשי, ולשם כך יש לחשב את הכתובת בזיכרון הראשי אליו שייך הבלוק :

$\text{Index(set)} = 1, \text{Tag} = 1 \rightarrow \text{Block address} = 001_01_2 = 5_{10}$

לכן מבוצעת כתיבה של בלוק 2 ב-cache לבלוק 5 בזיכרון הראשי.

לאחר מכן מובא בלוק מס' 29 מהזיכרון הראשי ומושם בבלוק 2 שב-cache.

ה-valid bit מתעדכן ל-1

ה- dirty bit מתעדכן ל-0.

שדה ה-Tag מתעדכן לשדה ה-Tag של בלוק 29, כלומר ל-7.

מתבצעת חזרה על תהליך הקריאה שהפעם מצליחה והבלוק מועבר ל-CPU, ממנו נלקח בית מס' 0.

אחרי הקריאה:

תוכן ה- cache :

block No.	0	1	2	3	4	5	6	7
dirty bit	0	0	0	0	1	1	1	0
valid bit	1	0	1	1	1	1	1	0
Tag 3 bits	0	x	7	2	5	7	4	x
Block data		x				E	F	x
	set 0		set 1		set 2		set 3	

הזיכרון הראשי:

lock No.	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
						בלוק מעודכן B																								בלוק נדרש G		

דוגמא מס' 3 – Write miss עם no write allocate.

כתיבה לכתובת 232_{10}. הנח מדיניות no write allocate

$232_{10} = 111_01_000_2$
$Offset = 000_2 = 0_{10}$
$Index = 01_2 = 1_{10}$
$Tag = 111_2 = 7_{10}$
Block Address in main memory $= 11101_2 = 29_{10}$

תוכן ה- cache :

block No.	0	1	2	3	4	5	6	7
dirty bit	0	0	1	0	1	1	1	0
valid bit	1	0	1	1	1	1	1	0
Tag 3 bits	0	x	1	2	5	7	4	x
Block data		x				E	F	x
	set 0		set 1		set 2		set 3	

הזיכרון הראשי:

Block No.	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
																														בלוק לעדכון G		

Index =1 לכן נבדקים כל (2) הבלוקים שב-set 1 , ה-valid bit=1 אך אף אחד מה-Tag של הבלוקים שב-set 1 אינו 7,לכן יש Write miss.
בגלל מדיניות ה-No write allocate, פעולת הכתיבה תבוצע בזיכרון הראשי והבלוק לא יובא ל-cache, לכן תוכן ה-cache לא ישתנה, ותוכן בלוק 29 בזיכרון הראשי יעודכן בהתאם לנתון לכתיבה.

דוגמא מס' 4 – Write miss עם Write back ו-Write allocate

כתיבה לכתובת 232_{10}. הנח מדיניות write allocate ו-Write back .

$232_{10} = 111_01_000_2$
Offset = $000_2 = 0_{10}$
Index = $01_2 = 1_{10}$
Tag = $111_2 = 7_{10}$
Block Address in main memory = $11101_2 = 29_{10}$

תוכן ה- cache :

block No.	0	1	2	3	4	5	6	7
dirty bit	0	0	1	0	1	1	1	0
valid bit	1	0	1	1	1	1	1	0
Tag 3 bits	0	x	1	2	5	7	4	x
Block data		x				E	F	x
	Set 0		Set 1		Set 2		Set 3	

הזיכרון הראשי:

Block No.	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
																														בלוק לעדכון G		

Index =1 לכן נבדקים כל (2) הבלוקים שב-set 1 , ה-valid bit=1 אך אף אחד מה-Tag של הבלוקים שב-set 1 אינו 7,לכן יש Write miss.
בגלל מדיניות ה-write allocate , מובא בלוק 29 מהזיכרון הראשי ל-set 1 ב-cache.
בלוק מס' 2 ב- cache נבחר להיזרק (LRU), אך מאחר שה-dirty bit=1, יש לבצע כתיבה של בלוק מס' 2 לזיכרון הראשי, ולשם כך יש לחשב את הכתובת בזיכרון הראשי אליו שייך הבלוק :
Index = 1,Tag =1 ➔Block address = $001_01_2 = 5_{10}$
לכן מבוצעת כתיבה של בלוק 2 ב-cache לבלוק 5 בזיכרון הראשי.
לאחר מכן מובא בלוק מס' 29 מהזיכרון הראשי ומושם בבלוק 2 שב-cache.
כעת מעודכן הבלוק ב-cache, בהתאם לנתון שרצינו לכתוב לתוכו.
ה-valid bit מתעדכן ל-1
ה- dirty bit מתעדכן ל-1 (כי תוכן הבלוק ב-cache מעודכן יותר מבזיכרון הראשי)
שדה ה-Tag מתעדכן לשדה ה-Tag של בלוק 29, כלומר ל-7.

אחרי הכתיבה:

תוכן ה- cache :

block No.	0	1	2	3	4	5	6	7
dirty bit	0	0	1	0	1	1	1	0
valid bit	1	0	1	1	1	1	1	0
Tag 3 bits	0	x	7	2	5	7	4	x
Block data		x	`			E	F	x
	set 0		set 1		set 2		set 3	

הזיכרון הראשי:

lock No.	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
						בלוק B																								בלוק G		