הטכניון – מכון טכנולוגי לישראל

הפקולטה למדעי המחשב

תאריך הגשה: 17.5.2016 שעה 23:55

הוראות הגשה: ההגשה בזוגות . הוסיפו שמות, ת.ז., אי-מייל, תא אליו יש להחזיר את התרגיל ואת תשובותיכם לתרגיל, הדפיסו והגישו לתא הקורס בקומה 1. עבור הגשות באיחור יש להגיש לתא של יוני.

מגישים:

ליאור בן עמי – 201182839

עידן אטיאס – 201368958

תא החזרה - 39

שאלה 1 - זיכרון מטמון

נתון מעבד עם שתי רמות זיכרון מטמון. עיקרון ההכלה לא מתקיים.

L1 : גודל שורה 32 byte , 4 -way set associative, גודל 2KB , מדיניות כתיבה write back מדיניות write allocate, מדיניות פינוי LRU .

L2: גודל שורה 32 byte , 2 -way set associative, גודל 32KB , מדיניות כתיבה write back, מדיניות write allocate, מדיניות פינוי LRU .

גודל הכתובת 32 ביט.

א. מהו מבנה הכתובת של המטמון הנ"ל?

תשובה:

עבור L1:

31	tag	9	8	set	5	4	offset	0

הסבר:

Offset_bits = log (block size) == log (line size) = log 32 = 5

Set_bits = log ((cache size / line size) / total_ways)) = log (64 / 4) = 4

Tag_bits = 32 – set_bits – offset_bits = 32 – 4 – 5 = 23

באותו אופן, עבור L2:

31	tag	14	13	set	5	4	offset	0

ב. מהו גודל ה- tag directory (הסיביות אשר צריך לשמור כדי לנהל את המטמון) עבור כל אחד מהמטמונים?

תשובה:

<u>במטמון L1</u> נחזיק עבור כל tag ביט valid וגם ביט dirty (בגלל שמדיניות הכתיבה היא write back). בנוסף, כיוון שמדיניות הפינוי היא LRU וישנם 4-ways אזי מספר הביטים המינימלי שנצטרך לשמור הוא 5 (עבור כל סט) ע"מ לקודד 4! אפשרויות של סידור התור. לכן סה"כ גודל ה tag directory (בביטים) המתקבל יהיה:

(|tag| + valid + dirty) * total_ways * total_sets + LRU_bits* total_sets =
= (23 + 1 + 1) * 4 * 16 + 5*16 = 1680

<u>עבור L2</u> החישוב יתבצע באופן דומה למתואר עבור L1 ולכן סה"כ גודל ה tag directory (בביטים) המתקבל יהיה:

(|tag| + valid + dirty) * total_ways * total_sets + LRU_bits* total_sets =
= (18 + 1 + 1) * 2 * 512 + 1*512 = 20992

מריצים את התוכנית הבאה על המעבד:

```
int arr[576];
for(int i=0; i<576;i++)
    arr[i]=2*arr[i];
int s=0;
for (int j=0; j<576;j++)
    s+=arr[j]
```

הנחות:

*המשתנים I,j,s וכן המצביע למערך שמורים ברגיסטרים.
*המספרים בתוכנית הם בבסיס דצימלי.
*המטמון ריק בתחילת התוכנית.
*המערך arr מתחיל בכתובת H360 (בסיס הקסדצימלי).
* גודל כל משתנה מסוג int הוא 4 בתים.

ג. לאיזה set ב-L1 ייכנס האיבר ה- 18 (בבסיס דצימלי) של המערך?

תשובה:

עבור arr[16] שנמצא בכתובת H530 נקבל miss, נביא את הנתון מהזכרון הראשי ונשים אותו בשורה פנויה (בהכרח יש כזו כי לפניכן נתפסו רק 2 שורות עבור i=0,8) ב set שנגזר מכתובת האיבר arr[16] : H360 + H10*4 = H3A0 ובבינארי נקבל: 1110100000 ולכן הסט הוא 13 בדצימלי. כיוון שבכל miss אנחנו עושים fetch מהזכרון הראשי ב chunks של 32 בתים אזי בפרט arr[17] (=האיבר ה18) יימצא גם בסט מספר 13.

ד. מהו ה-hit rate ב-L1 עבור הלולאה הראשונה?

תשובה:

גודל cache line הוא 32 בתים וגודל int הוא 4 בתים, לכן בכל miss אנו מביאים עוד 7 integers נוספים מעבר לאיבר שבגללו חטפנו miss.
בנוסף, בכל איטרציה של הלולאה הראשונה יש גם נסיון קריאה של איבר i במערך מהזכרון וגם נסיון כתיבה אליו ולכן בכל איטרציה ישנן 2 גישות לזיכרון כאשר רק באחת מהן נקבל miss. לכן סה"כ הנ"ל שקול ל miss יחיד עבור 16 גישות לזיכרון. (כאמור כל miss גורר "הבאה" של עוד 7 intים נוספים ל cache) ולכן סה"כ ה hit rate ב L1 עבור הלולאה הראשונה יהיה 15/16 = (1/16)-1.

ה. מהו ה- hit rate ב- L1 עבור הלולאה השנייה ?

תשובה:

גודל L1 הוא 2KB ולכן נוכל להחזיק בו בכל רגע נתון לכל היותר 512 = 2KB/4B משתנים מסוג int. לכן ניתן לומר כי בלולאה השנייה 576-512=64 איברים יהיו "חסרים" ב cache. ע"פ ההסבר שלנו בסעיף קודם, הנ"ל יגרום ל-8 misses בכל way (ב way מסוים X נדרסו 8 כניסות [כיוון שיש מקום רק ל512 intים ב cache] ולכן נפנה ב way הבא 8 כניסות לפי מנגנון ה LRU ופינוי זה יגרור בתורו גם 8 misses כשנגיע אליו...) ולכן סה"כ נקבל 32 misses במעבר על 512 האיברים הראשונים בלולאה. עבור ה64 הנותרים "נחזור" שוב למצב ההתחלתי בו נצטרך לדרוס עוד 8 כניסות ב wayX, כלומר 8 misses נוספים.
סה"כ נקבל כי ה hit-rate ב L1 עבור הלולאה השנייה יהיה:
1-(32+8)/576 = 0.93

ו. ללא קשר לסעיפים הקודמים, הציעו סדרת פניות לזיכרון כך שתתרחש בסופה פגיעה ב- L1 אך הנתון לא נמצא ב- L2 . הניחו כי בתחילת סדרת הגישות הנ"ל המטמונים ריקים. הסבירו את תשובתכם.

תשובה:

ניזכר במבנה כתובת ב L1:

31	tag	9	8	set	5	4	offset	0

ועבור L2:

31	tag	14	13	set	5	4	offset	0

ונגדיר 3 כתובות האופן הבא: (הביטים המודגשים הם 5-13)

X = [0000 0000 0000 0000 0000 0000 0000 0000]
Y = [0000 0000 0000 0000 0100 0000 0000 0000]
Z = [0000 0000 0000 0000 1000 0000 0000 0000]

כעת נבצע את הפעולות הבאות:

fetch(X) – 1
fetch(Y) – 2
fetch(Z) – 3
fetch(X) – 4

נשים לב כי עבור 3 הכתובות מספר הסט שלהם זהה גם עבור L1 וגם עבור L2 אבל ה-tag שלהם שונה (מה שמבטיח שתוכן הכתובות יהיו ב ways שונים). בנוסף, L1 הוא 4 way ולכן תוכן הכתובות X,Y,Z יובא ל3 ways שונים (באותו סט) בL1.
L2 לעומת זאת הוא 2 way ולכן הסט המתאים מתמלא לאחר הבאת התוכן של כתובות X ו – Y, כלומר עבור התוכן שב Z נצטרך לפנות, לפי מדיניות LRU, את הכניסה המתאימה ב way שמכילה את תוכן X. כעת עבור פעולה מס' 4 נקבל כי הנתון נמצא בL1 אך לא בL2.

<u>שאלה 2 - מבנה המטמון</u>

א. נתון מטמון direct map עם מבנה הכתובת הבא:

tag (24-bits)	set (4-bits)	offset (6 bits)

משנים את המטמון כך שהמיפוי של שורת מטמון ל- set במטמון יתבצע לפי 4 הסיביות ה- MSB של הכתובת, כך שכתובת המטמון לאחר השינוי נראית כך:

set (4-bits)	tag (24-bits)	offset (6 bits)

כיצד צפוי השינוי הבא להשפיע על ה-miss rate של המטמון ?

תשובה:
ה- miss rate יגדל משמעותית. הסבר: הפיכת המיפוי גורם לעד פי $2^{32-6} = 2^{26}$ כתובת להיות ממופות לאותה שורה. לפני השינוי, מתוך מרחב זיכרון של $2^{32} = 4GB$ כתובות אפשריות, עד כ $2^6 = 64$ כתובות בלבד ממופות לאותו סט. אחרי השינוי, עד כ $2^{30} = 1GB$ כתובות ממופות לאותו set. מה שעלול לגרום לשיעור שגיאות conflict הרבה יותר גבוה.

ב. בחברת double flops מייצרים מעבד בעל זיכרון מטמון :
גודל שורה 32 byte, 4-way set associative, 4kb גודל המידע.
גודל הכתובת הוא 32 ביט.

משנים את המטמון כך שגודל השורה יהיה 64 byte. גודל המטמון יישאר 4kb.
מה השינוי הצפוי בגודל ה tag directory (הזכרון הנוסף מעבר ל DATA שדרוש לניהול המטמון)

תשובה:
גודל ה tag directory <u>לפני</u> השינוי:

- **מספר השורות הכולל:** $\#blocks = \frac{4\ kb}{32\ byte} = 128$
- **מספר השורות בכל way:** $\#sets = \frac{128}{4} = 32$
- **גודל כל שורת (tag) –**

$offset = \log_2 32 = 5\ bit$
$set = \log_2 32 = 5\ bit$
$tag = 32 - 10 = 22\ bit$

לכן סה"כ גודל ה tag directory לפני השינוי - $22\ bit \cdot 32 \cdot 4 = 352\ byte$.

- **גודל ה tag directory <u>אחרי</u> השינוי:**
  - **מספר השורות הכולל:** $\frac{4\ kb}{64\ byte} = 64$

- **מספר השורות בכל way:** $\frac{64}{4} = 16$
- **גודל כל שורת (tag) –**

$$offset = \log_2 64 = 6\ bit$$
$$set = \log_2 16 = 4\ bit$$
$$tag = 32 - 10 = 22\ bit$$

לכן סה"כ גודל ה tag directory אחרי השינוי - $22\ bit \cdot 16 \cdot 4 = 176\ byte$.

- **לכן גודל השינוי הוא:**

$$\frac{\#(\text{tag directory})_{after}}{\#(\text{tag directory})_{before}} = \frac{352}{176} = 0.5$$

כלומר, יקטן בחצי.

ג. בהמשך לסעיף ב , לאחר הגדלת השורה:

משנים את המטמון כך שיהיה 8-way set associative . גודל המטמון יישאר 4kb.
מה השינוי הצפוי בגודל ה tag directory ?

תשובה:

- **גודל ה tag directory <u>לפני</u> השינוי:**
  חישבנו בסעיף קודם: $176\ byte$
- **גודל ה tag directory <u>אחרי</u> השינוי:**
  - **מספר השורות הכולל:** $\#blocks = \frac{4\ kb}{64\ byte} = 64$
  - **מספר השורות בכל way:** $\#sets = \frac{64}{8} = 8$
  - **גודל כל שורת (tag) –**

$$offset = \log_2 64 = 6\ bit$$
$$set = \log_2 8 = 3\ bit$$
$$tag = 32 - 9 = 23\ bit$$

לכן סה"כ גודל ה tag directory אחרי השינוי - $23\ bit \cdot 8 \cdot 8 = 184\ byte$.

- **לכן גודל השינוי הוא:**

$$\frac{\#(\text{tag directory})_{after}}{\#(\text{tag directory})_{before}} = \frac{184}{176} = 1.05$$

כלומר, גדל.

שאלה 3– חיזוי קפיצות

נתונה תוכנית:

```
0          MOV R4,#3
4   L4       DEC R4
.....
16         MOV R1,#5
20  L1       ...
24           ...
28           DEC R1
32  B1:      BNE R1,R0,L1
.....
56           MOV R2,#3
60 L2        DEC R2
64  ...
68 B2:       BNE R2,R0,L2
.....
84           MOV R3,#2
88  L3       ...
92           DEC R3
96  B3:      BNE R3,R0,L3
.....
148 B4:    BNE R4,R0,L4
```

לולאה של 5 איטרציות.

לולאה של 3 איטרציות.

לולאה של 2 איטרציות.

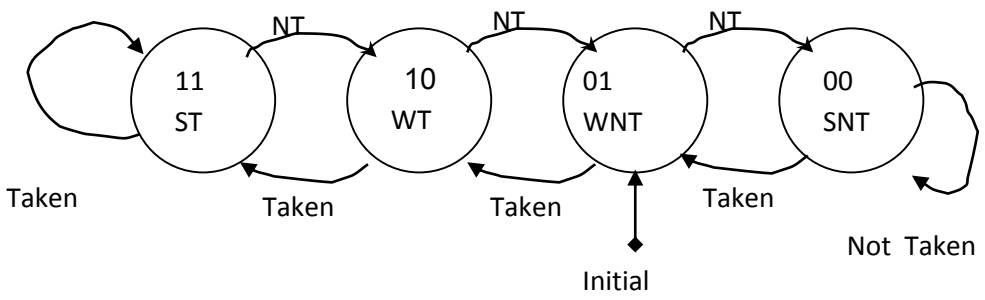
לולאה של 3 איטרציות.

(המספרים בעמודה הראשונה הם כתובות של הוראות והם עשרוניים).
שים לב: ההוראות הן aligned, כלומר כתובת הוראה היא כפולה של 4.
ההוראות שאינן מפורטות לא פונות לרגיסטרים R1,R2,R3,R4 ולא מבצעות Branch. להזכירכם, ערך של R0 תמיד 0. אורך כתובת של הוראה הוא 4 בתים.

התוכנית רצה על מעבד דוגמת MIPS שנלמד בכיתה, למעבד 5 שלבים (IF,ID,EX,MEM,WB), הוא עובד בשיטת in-order ופקודות branch מוכרעות בשלב MEM.

במערכת קיים מנגנון חיזוי המשתמש ב-BTB.

משתמשים במכונת חיזוי הבאה לכל אחד מה- Branch-ים:



לפני ריצת התכנית BTB ריק. אם הוראת Branch לא נמצאת ב-BTB, מצב החיזוי מאותחל ל-Weakly Not Taken, ומעודכן לאחר מכן בהתאם לתוצאת אותה הוראת branch.

מבנה כניסת BTB:

מצב חיזוי	כתובת קפיצה	Tag

ישנן 2 קונפיגורציות של BTB:

BTB1: direct mapped עם 8 כניסות

BTB2: fully associative עם 4 כניסות.

שדה ה-set (אם יש כזה) מתחיל בסיבית 2 (השלישית מימין) והוא רצוף.
מדיניות הפינוי היא LRU.

א) עבור כל BTB הסבר כמה החטאות ייווצרו במהלך האיטרציה הראשונה של B4 וכמה במהלך השנייה. – **(שים לב שיש 3 אטרציות ל B4)**.

תשובה:

<u>עבור BTB2 :</u>

עבור 3 הלולאות הפנימיות נקבל החטאה בכניסה וביציאה מהלולאה. כלומר פעמיים לכל לולאה. זאת בגלל שבהתחלה הbranch לא מופיע בBTB ועל כן יש החטאה בכניסה ללולאה ומכיוון שביציאה מהלולאה החיזוי הוא Taken אך אנו לא מבצעים עוד איטרציה (כפי שנלמד בתרגול).
עבור הלולאה החיצונית נקבל החטאה יחידה בכניסה ללולאה מאותה סיבה שכבר תיארנו.
לכן, סה"כ עבור איטרציה ראשונה של B4 נקבל 7 החטאות.
עבור <u>איטרציה שנייה</u> נקבל כי B1 יחטיא רק ביציאה מהלולאה ולא בכניסה כיוון שבאיטרציה הראשונה הערך האחרון של החיזוי עבורו היה WT.
B2 כנ"ל.
B3 נקבל החטאה בכניסה וביציאה מכיוון שסיים איטרציה ראשונה עם WNT.
B4 לא נקבל החטאה כיוון שהחיזוי הוא WT והקפיצה אכן מתבצעת.
לכן סה"כ נקבל 4 החטאות עבור האיטרציה השניה.

<u>עבור BTB1:</u> נמצא את ה set עבור כל אחת מהכניסות לפי כתובת המטרה. נשים לב שמכיוון שיש 8 כניסות, ה set באורך 3 ביטים.

<u>B1</u>: 32 (Dec) = 0010 0000 (BIN) -> <u>set = 000</u>

B2: 68 (Dec) = 0100 0100 (BIN) -> set = 001

<u>B3</u>: 96 (Dec) = 0110 0000 (BIN) -> <u>set = 000</u>

B4: 148 (Dec) = 1001 0100 (BIN) -> set = 101

כלומר, יש התנגשויות בין B1 ל B3 בטבלה.

איטרציה ראשונה:

עבור B1,B2,B3 - 2 החטאות לכל אחד – כניסה ויציאה, כמו ב BTB2. סה"כ 6.

עבור B4 – החטאה אחת בכניסה – כמו ב BTB2.

סה"כ איטרציה ראשונה – 7 החטאות.

איטרציה שנייה:

עבור B1, B3 – 2 החטאות לכל אחד – כניסה ויציאה. מכיוון שהם יושבים באותה שורה, בכניסה ל B1, B3 היה שמור מהאיטרציה הקודמת של B4, ובכניסה ל B3, B1 היה שמור מהאיטרציה הנוכחית של B4.

עבור B2 – החטאה אחת ביציאה. כמו ב BTB2.

עבור B4 – כמו ב BTB2.

סה"כ באיטרציה השנייה – 5 החטאות.

ב) נתון ש-BTB1 מספק את ערך החיזוי שלו בשלב ה-ID ואילו BTB2 בשלב ה-IF, אם מתברר שהחיזוי הוא taken אזי עושים flush לפקודה שנכנסה. הנח שהקוד והנתונים נמצאים כולם במטמון וכן אין context switch או קריאות לפונקציות במהלך הריצה. **איזה BTB ייתן את הביצועים הטובים ביותר (מבחינת זמן ביצוע)?**

תשובה:
נשים לב למקרים הבאים:

- **BTB1: עבור חיזוי taken שנלקח בפועל נצטרך לחכות מחזור אחד (מההגדרה בשאלה)**
- **BTB2: עבור חיזוי taken שנלקח בפועל לא נצטרך לחכות**
- **BTB1 וגם BTB2: עבור חיזוי not taken שלא נלקח בפועל לא נצטרך לחכות**
- **BTB1 וגם BTB2: עבור חיזוי שגוי ביחס למה שקורה בפועל נצטרך לחכות 2 מחזורים**

אחרי חישוב באופן דומה לסעיף א' נקבל כי סה"כ ההחטאות (ב-3 האיטרציות של הלולאה החיצונית) עבור BTB1 הוא 18 ועבור BTB2 16.
בנוסף נשים לב כי עבור BTB1 נצטרך לחכות מחזור שעון (להתעכב) גם עבור הצלחה בחיזוי בעוד שב BTB2 לא נצטרך לחכות.
על כן, ניתן לומר כי BTB2 ייתן ביצועים טובים יותר.