**הטכניון – מכון טכנולוגי לישראל**

**הפקולטה למדעי המחשב**

תאריך הגשה: 17.5.2016 שעה 23:55

**תא להחזרה: 62**

**אילון גל, 302921762,** [**eilongal@gmail.com**](mailto:eilongal@gmail.com)

**ג'ורג' אפדילא, 312561913,** [**gavdellas@gmail.com**](mailto:gavdellas@gmail.com)

**שאלה 1 - זיכרון מטמון**

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

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

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

גודל הכתובת 32 ביט.

1. **מהו מבנה הכתובת של המטמון הנ"ל?**

מכיוון שגודל שורה הוא 32 בתים, דרושים 5 ביטים עבור הdisplacement.

כמות הבלוקים בL1:

Cache Size/Line Size = 2KB/32B = 2^6=64

כמות הסטים:

# Blocks/# Ways =2^6/4=2^4=16

לכן דרושים 4 ביטים ע"מ לייצג את מספר הסט.

סה"כ: 5 ביטים disp, 4 ביטים #set, 23 ביטים עבור הtag.

כמות הבלוקים בL2:

Cache Size/Line Size = 32KB/32B = 1K = 2^10 = 1024

כמות הסטים:

# Blocks/# Ways =2^10/2=2^9=512

לכן דרושים 9 ביטים ע"מ לייצג את מספר הסט.

סה"כ: 5 ביטים disp, 9 ביטים #set, 18 ביטים עבור הtag.

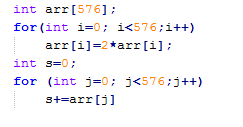
1. **מהו גודל ה- tag directory (הסיביות אשר צריך לשמור כדי לנהל את המטמון) עבור כל אחד מהמטמונים?**

עבור L1: כפי שראינו דרושים 23 ביטים עבור הtag עצמו. בנוסף, עבור מימוש LRU פשוט עבור 4-way דרושים 2 ביטים (לשמור את הסדר של השינויים), ועבור מדיניות write-back דרוש ביט נוסף. לכן יש סך הכל 23+2+1=26 ביטים בכל כניסה. נכפיל בכמות הבלוקים כדי לקבל את הגודל: # block\*bits=64\*26=1664bits=**208B**.

עבור L2: 23 ביטים עבור הtag. מכיוון שבמקרה זה יש 2-way מספיק ביט אחד שיציין מי האחרון שעשינו בו שימוש כדי לממש LRU. גם במקרה זה עקב מדיניות write-back דרוש ביט נוסף. סה"כ נקבל 23+1+1=25bit. הגודל:

# blocks\*bits=1024\*25=25600bit=**3200B**.

מריצים את התוכנית הבאה על המעבד:



הנחות:

\*המשתנים I,j,s וכן המצביע למערך שמורים ברגיסטרים.

\*המספרים בתוכנית הם בבסיס דצימלי.

\*המטמון ריק בתחילת התוכנית.

\*המערך arr מתחיל בכתובת 360H (בסיס הקסדצימלי).

\* גודל כל משתנה מסוג int הוא 4 בתים.

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

כתובת ההתחלה של האיבר ה18 היא 18\*4=72 בתים (דצימלי) אחרי תחילת המערך שהם 48 בתים (הקסדצימלי), נחבר מספרים בהקסדצימלי ונקבל שהכתובת היא 3A8, בפירוק לביטים נקבל 0011,1010,1000. בידוד 4 הביטים של הסט יתן לנו 1101, כלומר 13 בדצימלי.

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

עבור הגישה ל512 האיברים הראשונים, בכל פעם שנקבל miss נביא שורה של 32 בתים שהם 8 intים, ולכן מכיוון שאנו מבצעים גם קריאה וגם כתיבה לכל כתובת, נקבל miss כל 16 גישות. כל 512 האיברים הראשונים ממלאים בדיוק את הL1. כלומר הhit rate=15/16. עבור הגישה לאיברים הנותרים (64 איברים), נגמר המקום בL1 ולכן נתחיל לפנות לפי הLRU את האיברים הראשונים שהוכנסו. גם כאן בכל פעם נביא שורה שלמה שמכילה 8 intים ולכן גם כאן הhit rate=15/16 ובסך הכל 15/16 עבור הלולאה הראשונה.

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

בתום הלולאה הראשונה נדרסו 8 בלוקים בWay 0 (בחירת הWay בה"כ). כעת כשננסה לגשת לכתובות המתאימות ל8 בלוקים הללו (64 intים ראשונים) נקבל miss מכיוון שהן נדרסו. לכן, נפנה לפי הLRU 8 בלוקים מתוך הWay הבא אחריו, Way 1. עבור הכתובות המתאימות ל8 הבלוקים הנותרים שלא נדרסו מתוך Way 0, נקבל Hit. כעת, כאשר נסיים לעבור על כל 128 הintים הראשונים שמכסים בדיוק Way שלם, התהליך הנ"ל יקרה בצורה זהה עבור Way 1 כי פינינו 8 בלוקים מתוכו, וכן הלאה במשך כל ריצת התוכנית. כדי לחשב את כמות ההחטאות, נסתכל על Way ספציפי כלשהו (הגענו למסקנה כי אותו תהליך חוזר בכל Way). מכיוון שנקבל miss עבור 8 גישות מתוך 16(# blocks in a way)\*8(# of ints in a block)=128 גישות בWay כלשהו, בסך הכל הmiss rate יהיה 32/512 עבור מעבר שלם על כל 4 הWayים, ועל 512 הintים הראשונים. לאחר שנסיים לעבור על כל 4 הWayים עדיין יישארו 64 intים לגשת אליהם, ומכיוון שהמידע עבורם לא נמצא בcache נקבל miss כל בלוק, כלומר miss כל 8 גישות, ובסך הכל 8/64 החטאות. בסך הכל נקבל 32+8/576 כיחס החטאות או **hit rate=0.9305**.

**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.**

נניח כי הכתובות X, Y, Z הן שלוש כתובות שביטים 30-31 שונים אצל כל אחת מהם (כך שלכל אחת יש tag שונה), ושעבורן מספר הset שווה בשני הcacheים. (כאשר ביטים 5-13 הם אפס בשלושת הכתובות נקבל כי הset יהיה אפס בשני הcacheים). ננצל את העובדה שcache אחד הוא 4-way וcache 2 הוא 2-way.

ננתח את סדרת הקריאות הבאה:

1. קריאה מX
2. קריאה מY
3. קריאה מZ
4. קריאה מX

עבור קריאה 1 נקבל החטאה בשני הקאשים והבלוק יובא לשניהם (טאג לא קיים).

עבור קריאה 2 נקבל החטאה בשני הקאשים והבלוק יובא לשניהם (טאג לא קיים).

עבור קריאה 3 נקבל החטאה בשני הקאשים (טאג לא קיים), בL1 הבלוק יובא לWay 3, אבל בL2 נגמרו הWays, ולכן לפי LRU נפנה את כתובת X מL2 ונביא במקומה את כתובת Z.

בקריאה הבאה מX נקבל פגיעה בL1 כי הוא עדיין שם מאז קריאה 1, אבל החטאה בL2 כי פינינו בעת ביצוע קריאה 3.

**שאלה 2 - מבנה המטמון**

א. נתון מטמון direct map עם מבנה הכתובת הבא:

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

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

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

השינוי הנ"ל יגרום לעלייה בmiss rate. מכיוון שהמטמון הוא מסוג direct map יש לנו למעשה רק way אחד. לכן, המטמון יכול לשמור רק כתובת אחת באותו set בכל רגע נתון. מכיוון שכעת הset הוא בMSB, יש טווח מקסימלי של כתובות שנכנסות באותו set: לפני השינוי יש 2^6 כתובות בלבד באותו סט, לאחר השינוי יש 2^30 כתובות באותו סט. לכן, כמעט כל הכתובות בזכרון נכנסות באותו סט ומכיוון שאנו יכולים לשמור רק כתובת אחת עבור כל הסט הזה נקבל miss rate גבוה מאוד ביחס לmiss rate שלפני השינוי.

**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)?**

לפני השינוי כמות הבלוקים היא:

Cache Size/Line Size=4KB/32B=128

ולכן כמות הסטים היא:

# Blocks/# Ways = 128/4 =32

כלומר מבנה הכתובת הוא 5 ביטים לאופסט, 5 ביטים לסט, 22 ביטים לtag.

ולכן גודל ספריית הtag יהיה: # Blocks \* # Bits = 128\*22 = 352B

אחרי השינוי כמות הבלוקים היא:

Cache Size/Line Size=4KB/64B=64

ולכן כמות הסטים היא:

# Blocks/# Ways = 64/4 =16

כלומר מבנה הכתובת הוא 6 ביטים לאופסט, 4 ביטים לסט, 22 ביטים לtag.

ולכן גודל ספריית הtag יהיה: # Blocks \* # Bits = 64\*22 = 176B

כלומר גודל הספריה יקטן בחצי לאחר השינוי.

**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 ?**

אחרי השינוי כמות הבלוקים לא השתנתה, כלומר 64 בלוקים.

ולכן כמות הסטים היא:

# Blocks/# Ways = 64/8 = 8

כלומר מבנה הכתובת הוא 6 ביטים לאופסט, 3 ביטים לסט, 23 ביטים לtag.

ולכן גודל ספריית הtag יהיה: # Blocks \* # Bits = 64\*23 = 184B

**שאלה 3– חיזוי קפיצות**

נתונה תוכנית:

0 MOV R4,#3

4 L4 DEC R4

**…..**

לולאה של 5 איטרציות.

16 MOV R1,#5

20 L1 …

24 …

28 DEC R1

32 B1: BNE R1,R0,L1

לולאה של 3 איטרציות.

**…..**

56 MOV R2,#3

60 L2 DEC R2

לולאה של 3 איטרציות.

64 ...

68 B2: BNE R2,R0,L2

**…..**

84 MOV R3,#2

88 L3 …

לולאה של 2 איטרציות.

92 DEC R3

96 B3: BNE R3,R0,L3

**…..**

148 B4: BNE R4,R0,L4

(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:

NT

NT

NT

00 SNT

01 WNT

10 WT

11   
ST

Taken

Taken

Taken

Taken

Not Taken

Initial

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:

|  |  |  |
| --- | --- | --- |
| Tag | כתובת קפיצה | מצב חיזוי |

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.

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

**BTB2:**

מכיוון שיש רק 4 פקודות branch, והBTB הנ"ל הוא fully associative בגודל 4, אז אין התנגשות וכולם יכולים להכנס בו זמנית.

**אירטציה ראשונה של B4:**

בפעם הראשונה שמגיעים לB1 תתקבל החטאה מכיוון שהBTB ריק. אחרי ההחטאה הראשונה נקבל:

|  |  |  |
| --- | --- | --- |
| B1 | L1 | WT |

באיטרציה הבאה לא נקבל החטאה מכיוון שהקפיצה נלקחת, הBTB יתעדכן ל:

|  |  |  |
| --- | --- | --- |
| B1 | L1 | ST |

בשתי האיטרציות הבאות החיזוי יהיה נכון כי הקפיצה נלקחת, והBTB לא ישתנה.

באיטרציה החמישית הקפיצה לא נלקחת, בניגוד לחיזוי, ולכן נקבל החטאה והBTB יעודכן ל:

|  |  |  |
| --- | --- | --- |
| B1 | L1 | WT |

**סה"כ: 2 החטאות בB1.**

בפעם הראשונה שמגיעים לB2 תתקבל החטאה מכיוון שהBTB לא מכיל את L2. אחרי ההחטאה הראשונה נקבל:

|  |  |  |
| --- | --- | --- |
| B2 | L2 | WT |

באיטרציה השנייה לא נקבל החטאה מכיוון שהקפיצה נלקחת, הBTB יתעדכן ל:

|  |  |  |
| --- | --- | --- |
| B2 | L2 | ST |

באיטרציה השלישית הקפיצה לא נלקחת, בניגוד לחיזוי, ולכן נקבל החטאה והBTB יעודכן ל:

|  |  |  |
| --- | --- | --- |
| B2 | L2 | WT |

**סה"כ: 2 החטאות בB2.**

בפעם הראשונה שמגיעים לB3 תתקבל החטאה מכיוון שהBTB לא מכיל את B3. אחרי ההחטאה הראשונה נקבל:

|  |  |  |
| --- | --- | --- |
| B3 | L3 | WT |

באיטרציה השנייה גם כן נקבל החטאה מכיוון שהקפיצה לא נלקחת בניגוד לחיזוי.

|  |  |  |
| --- | --- | --- |
| B3 | L3 | WNT |

**סה"כ: 2 החטאות בB3.**

בפעם הראשונה שמגיעים לB4 תתקבל החטאה מכיוון שהBTB לא מכיל את B4. אחרי ההחטאה הראשונה נקבל:

|  |  |  |
| --- | --- | --- |
| B4 | L4 | WT |

**סה"כ: החטאה אחת בB4.**

בסוף האיטרציה הראשונה, לאחר הקפיצה, מצב הBTB:

|  |  |  |
| --- | --- | --- |
| B1 | L1 | WT |
| B2 | L2 | WT |
| B3 | L3 | WNT |
| B4 | L4 | WT |

**איטרציה שנייה של B4:**

בשתי הbranchים הראשונים, B1 ו B2, ניתן לראות כי שניהם מופיעים בBTB והחיזוי עבור הקפיצה הראשונה בכל אחד מהם נכון. לכן, קיימת רק החטאה אחת ביציאה מכל לולאה ולכן בסך הכל שתי הלולאות הראשונות מחטיאות פעמיים.

**סה"כ: 2 החטאות בB1 וB2 יחד.**

בלולאה השלישית נקבל החטאה בחיזוי הראשון כי הקפיצה נלקחת אך החיזוי אומר WNT, לאחר מכן החיזוי יהפוך לWT ונקבל החטאה שנייה מכיוון שבאיטרציה השנייה הקפיצה לא נלקחת בניגוד לחיזוי.

כעת כשנגיע לB4 נקבל פגיעה כי החיזוי אומר שהקפיצה תלקח וכך גם קורה בפועל.

**סה"כ: 2 החטאות בB3.**

מצב הBTB בסוף האיטרציה השנייה לאחר הקפיצה השנייה של B4:

|  |  |  |
| --- | --- | --- |
| B1 | L1 | WT |
| B2 | L2 | WT |
| B3 | L3 | WNT |
| B4 | L4 | ST |

**בסך הכל בשתי האיטרציות קיבלנו 11 החטאות.**

**BTB1:**

מכיוון שהמטמון הוא direct mapping עם 8 כניסות, יש למעשה 8 סטים בתוך way אחד. לכן, דרושים 3 ביטים עבור הset, והם יהיו ביטים 2-4. לאחר כתיבת הכתובות בבינארי מצאנו כי הסטים עבור פקודות הקפיצה הם:

B1: 00100000

B2: 01000100

B3: 01100000

B4: 10010100

B1=0, B2=1, B3=0, B4=5

לכן, תהיה התנגשות בין B3 וB1, כל פעם שנגיע לאחד מהם לפי LRU נפנה את השני אם הוא כבר קיים בטבלה.

**איטרציה ראשונה של B4:**

באיטרציה הראשונה לא יהיה שינוי מהניתוח שעשינו עבור BTB1 מכיוון שבכל פעם שמגיעים לbranch חדש, הוא לא נמצא בתוך הBTB, ולכן הוא מוכנס בפעם הראשונה ומשם מתעדכן באותה צורה כמו שבBTB1. לכן, בסוף האיטרציה הראשונה מצב הBTB יהיה:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | B3 | L3 | WNT |
| 1 | B2 | L2 | WT |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 | B4 | L4 | WT |
| 6 |  |  |  |
| 7 |  |  |  |

**בסך הכל בלולאה הראשונה היו 7 החטאות כמו במקרה של BTB1.**

**איטרציה שנייה של B4:**

בהגעה הראשונה לB1 נגלה שהוא לא בBTB מכיוון שהוא פונה כאשר B3 הוכנס. לכן, בדומה לניתוח שעשינו עבור המקרה בו הקפיצה לא מופיעה בטבלה, נקבל החטאה אחת בפעם הראשונה כאשר היא מוכנסת, והחטאה נוספת ביציאה מהלולאה.

בהגעה לB2 נגלה שהוא כבר בטבלה והחיזוי נכון כי הקפיצה נלקחת. גם כאן באופן דומה לניתוח כאשר B2 כבר בטבלה, נקבל החטאה אחת רק ביציאה מהלולאה.

בהגעה לB3 שוב הוא איננו נמצא בטבלה מכיוון שהוא נדרס כאשר הכנסנו קודם לכן את B1, ולפי ניתוח זהה נקבל 2 החטאות, אחת עבור ההכנסה לטבלה, והשנייה כאשר הקפיצה השנייה לא נקלחת בניגוד לחיזוי.

בהגעה לB4 נגלה שהוא בטבלה והחיזוי נכון כי הקפיצה נלקחת, ולכן לא תהיה החטאה כאן. טבלת הBTB בסוף האיטרציה השנייה לאחר לקיחת הקפיצה:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | B3 | L3 | WNT |
| 1 | B2 | L2 | WT |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 | B4 | L4 | ST |
| 6 |  |  |  |
| 7 |  |  |  |

**בסך הכל נקבל 5 החטאות באיטרציה השנייה, ולכן באופן כללי נקבל 12 החטאות עבור BTB1.**

1. 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)?**

נפריד למקרים ע"פ התוצאות של החיזוי ומה שקרה בפועל, ונגדיר "קנס" במחזורי שעון על כל אחד מהמקרים:

BTB1:

1. חיזוי: נלקח בפועל: נלקח: קנס: מחזור אחד.
2. חיזוי: לא נקלח בפועל: לא נלקח קנס: אין.
3. חיזוי שגוי ביחס למהש קרה בפועל. קנס: 3 מחזורים.

BTB2:

1. חיזוי: נלקח בפועל: נלקח: קנס: אין.
2. חיזוי: לא נקלח בפועל: לא נלקח קנס: אין.
3. חיזוי שגוי ביחס למהש קרה בפועל. קנס: 3 מחזורים.

בסוף הסעיף הקודם הגענו למסקנה שBTB1 מחטיא 12 פעמים, ואילו BTB2 מחטיא 11 פעמים. בתוספת של האיטרציה השלישית והאחרונה של B4, אנו מקבלים כי בסך הכל בכל ריצת התוכנית BTB1 מחטיא 18 פעמים וBTB2 מחטיא 16 פעמים.

מכיוון שלBTB1 יש יותר החטאות, (שעבורם יש קנס של 3 מחזורי שעון) וגם יש קנס נוסף על הצלחות שלא קיים אצל BTB2, כבר בשלב זה ניתן לראות שBTB2 יספק ביצועים יותר טובים. זאת ממכיוון שחיזויים של taken שנלקחו בפועל רק יוסיפו קנס נוסף עבור BTB1.

מכיוון שזמן הביצוע נמצא ביחס ישר לכמות מחזורי השעון, אז BTB2 יספק ביצועים טובים יותר מבחינת זמן ריצה, כלומר פחות בזבוז של מחזורי שעון.