אוניברסיטת בר-אילן (ע"ר)
הפקולטה להנדסה
רמת-גן 52900

BAR-ILAN UNIVERSITY (RA)
Faculty of Engineering
Ramat-Gan 52900, Israel

Tel: 03-5317722

engbi@mail.biu.ac.il

מבנה מחשבים ספרתיים

תשפ"א סמסטר ב' מועד א'

83-301

מרצה: פרופ' שמואל וימר

מתרגל: מר בנימין פרנקל

- **יש לקרוא היטב את ההוראות.**
- **חובה לענות על כל השאלות.**
- **ציון מקסימלי בבחינה: 100 נקודות.**
- **יש לנמק את כל תשובותיכם.**
- יש להקפיד על כתב יד קריא!
- יש לרשום תשובות בתוך הטבלאות המצורפות במקום שנדרש.
- **חומר עזר מותר בשימוש**: כל חומר עזר מודפס ומחשבון.
- **משך הבחינה:** שלוש שעות.
- **יש לצרף את שאלוני הבחינה למחברת!**

בהצלחה!

שאלה מספר 1 – Multithreading (60 נקודות)

בשאלה זו נבחן ביצועים של מעבד multithreaded.

המעבד הינו וריאנט של in-order pipelined MIPS בעל 5 שלבים:

- Fetch – בכל מחזור שעון מתבצע fetch עבור פקודה בודדת.
- Decode – בכל מחזור שעון מתבצע decode ו- issue עבור פקודה בודדת. אם לא ניתן לנפק את הפקודה בגלל data dependencies המעבד יושהה (stall).
- Execution – של ה- EX לוקח מחזור שעון בודד לכל סוגי הפקודות. Branch Resolution מתבצע בשלב ה- EX.
- Memory – המעבד עובד ללא זיכרון מטמון. כל גישה לזיכרון לוקחת 3 מחזורי שעון. יחידת הזיכרון הינה non-blocking, כלומר, לאחר ניפוק פקודת memory המעבד יכול להמשיך בביצוע התוכנית עד שיגיע לפקודה שתלויה בערך המיוצר על-ידי פקודת הזיכרון.
- Write-Back – זהה ל- MIPS המקורי. כתיבה בשלב ה- WB לוקחת מחזור שלם (אין חלוקה של ID ו- WB לשני חצאי מחזורים).
- המעבד מממש forwarding מלא, כלומר, ניתן להשתמש בערכים כבר במחזור השעון הבא אחרי זה שבו הערכים נוצרו.

המעבד תומך ב- fine grain multithreading של עד N חוטים (threads). בכל מחזור שעון מתבצע fetch עבור thread שונה, בצורה מחזורית (round-robin). ה- threads השונים משתמשים בעותק של הרגיסטרים ובכללם רגיסטר ה- PC (program counter).

אנו נבחן את ביצועי המעבד בעזרת ה- benchmark הבא, אשר עובר על אברי רשימה מקושרת ובודק האם מספר האברים השליליים גדול ממספר האברים הלא-שליליים. כאשר יש מספר threads במערכת, ירוץ כל thread על **רשימה נפרדת**.

```
struct node {
        Int data;
        node* next_ptr;
}

Int AreMostElementsNegative (node *list_head);
```

הפונקציה AreMostElementsNegative מתורגמת לקוד האסמבלי הבא:

```
; R1 contains a pointer to the head of the list
; R2 and R6 are initialized to zero
1   Loop:    lw R4, 0(R1)        ; R4 := data
2            lw R3, 4(R1)        ; R3 := next_ptr
3            slt R5, R4, R0      ; check if (data<0)
4            bez R5, NonNeg
5            addi R2, R2, 1
6            jmp Next
7   NonNeg:  subi R2, R2, 1
8            addi R6, R6, 1
9   Next:    add R1, R0, R3      ; R1 := R3, i.e., set R1 to next node
10           bnez R1, Loop       ; continue to loop until R1 == NULL
```

א. (5 נקודות) על ערכו של איזה רגיסטר צריך להסתכל בסוף ריצת התוכנית כדי לדעת האם רוב האיברים ברשימה המקושרת הם שליליים? מה יהיה ערכו של רגיסטר זה אם אכן רוב האיברים ברשימה הם שליליים?

צריך להסתכל על ערכו של R2: אם רוב האיברים ברשימה הם שליליים אזי ערכו של R2 יהיה גדול מאפס.

- בסעיפים ב' וג' נניח כי קיים branch predictor אידיאלי.

ב. (12 נקודות) כמה מחזורי שעון דרושים לשם ביצוע איטרציה אחת של הלולאה במצב יציב, כאשר קיים tread יחיד במערכת? השתמשו בטבלה המצורפת לתיאור איטרציה שלמה (מרגע כניסת פקודה מספר 1 ועד לסיום פקודה מספר 10). פרטו את כל הקידומים ואת הסיבות לכל stall (אם ישנם).

ראו טבלה מצורפת. פקודה מספר 3 תלויה בפקודה מספר 1 דרך רגיסטר R4, ולכן, יש צורך בשני מחזורי stall (משום שגישה לזיכרון אורכת 3 מחזורי שעון). בסה"כ, ביצוע איטרציה אחת של הלולאה דורשת 14 מחזורי שעון. הפתרון הניח שהאיבר הנוכחי ברשימה המקושרת הוא שלילי. כמובן, שמתקבלת גם ההנחה שהאיבר חיובי (מספר מחזורי השעון הדרושים לא תלוי בהנחה זו).

ג. (10 נקודות) אם נסמן את ה- latency של יחידת ה- Mem ב- k (לדוגמא, בסעיף הקודם עבדנו עם k=3), מהו המספר המינימלי של threads אשר יאפשר למעבד לעבוד בניצולת מלאה (כלומר, ללא מחזורי stall כלל)? בטאו את תשובתכם כפונקציה של k.

הערך של R4 מוכן בחזור ה- $4 + (k - 1)$, ולכן, פקודה מספר 3 יכולה להיכנס לשלב ה- EX במחזור שעון ה- $4 + k$. על מנת שלא יהיו מחזורי stall בכלל, פקודה מספר 3 צריכה להיכנס ל- pipe במחזור שעון ה- $4 + k - 2$ לכל המוקדם.

בהינתן N חוטים (threads) פקודה 3 של החוט הראשון תכנס ל- pipe במחזור שעון ה- $2N + 1$. לכן נדרוש: $2N + 1 \geq 4 + k - 2$, ולכן:

$$N \geq \left\lceil \frac{k+1}{2} \right\rceil$$

בהקשר של תוכנית כללית, המצב הבעייתי ביותר מבחינת thread מסוים הוא כאשר פקודת ALU מגיעה מיד לאחר פקודת lw וממתינה לערך הנטען מהזיכרון, לדוגמא:

```
lw R3, 0(R1)
add R5, R3, R3
```

במקרה שכזה, יש צורך ב- $N \geq k + 1$ חוטים שונים על מנת להגיע לניצולת מלאה.

- כעת, נניח כי במעבד קיים 1-bit branch predictor **יחיד** אליו ניגשים לפי כתובת ה- PC בשלב ה- IF. ה- BTB מתעדכן בסוף שלב ה- EX. הניחו כי אין penalty על פקודות jump (חיזוי אידיאלי עבור פקודות jump).

ה- benchmark מורכב משתי רשימות מקושרות בעלות M איברים כל אחת. כל אברי הרשימה listA הינם שליליים, וכל אברי הרשימה listB הינם חיוביים. על כל אחת מהרשימות מופעלת הפונקציה AreMostElementsNegative.

```
Res1 = AreMostElementsNegative (listA);
Res2 = AreMostElementsNegative (listB);
```

ד. (12 נקודות) מהו הזמן הדרוש לביצוע ה- benchmark כאשר thread בודד מבצע את התוכנית כולה (תחילה ה- thread מבצע את הפונקציה על listA ואח"כ הוא מבצע אותה על listB)? הניחו כי פקודות 4 ו- 10 אינן ממופות לאותו entry ב- BTB, וכי בתחילת הריצה ה- BTB מכיל חיזוי taken עבור שתי הפקודות הנ"ל.

יש לזהות את מספר ה- miss predictions שיהיו במהלך התוכנית:

עבור פקודה 4 יהיו שני חיזויים שגויים: אחד באיטרציה הראשונה של listA (החיזוי יהיה taken), ואחד באיטרציה הראשונה של listB (החיזוי יהיה not-taken).

עבור פקודה 10 יהיו 3 חיזויים שגויים: אחד באיטרציה האחרונה של listA (החיזוי יהיה taken), אחד באיטרציה הראשונה של listB (החיזוי יהיה not-taken), והאחרון באיטרציה האחרונה של listB (החיזוי יהיה taken).

Miss prediction penalty במעבד הוא 2 מחזורי שעון.

כמו-כן, יש לשים לב כי איטרציה חדשה של הלולאה מתחילה לפני שהאיטרציה הקודמת מסתיימת (כלומר, לפני שפקודה 10 יוצאת מה- pipe), ולכן, זמן הביצוע של ה- benchmark כולו הינו:

$$(\#Iterations \cdot Cycles\ Per\ Iteration) + (\#misprediction \cdot misprediction\ penalty) +$$

$$+ (leftovers\ from\ last\ iteration) = (2 \cdot M \cdot 10) + (5 \cdot 2) + (4) = 20 \cdot M + 14$$

ה. (12 נקודות) כעת פוצלה התוכנית לשני threads: thread אחד מפעיל את הפונקציה על listA, וה- thread השני מפעיל את הפונקציה על listB. שני ה- threads מופעלים בו-זמנית, כך שמחזור השעון הראשון מוקצה ל- thread הראשון, מחזור השעון השני מוקצה ל- thread השני וחוזר חלילה. הניחו כי בתחילת הריצה ה- BTB מכיל חיזוי taken עבור פקודות 4 ו- 10.

a) תארו בטבלה המצורפת את האיטרציה הראשונה של התוכנית, מרגע כניסת פקודה 1 של threadA ועד לסיום פקודה 10 של threadB. ציינו את כל הקידומים ואת הסיבות כל stall (אם ישנם). ברישום בטבלה הבדילו בין הפקודות של threadA לבין הפקודות של threadB על-ידי סימון כוכבית ליד מספר הפקודה של threadB, לדוגמא:
פקודה מספר אחת של threadA תצוין בטבלה כך: **1**
פקודה מספר אחת של threadB תצוין בטבלה כך: ***1**

ראו בטבלה המצורפת.

b) מהו הזמן הדרוש לביצוע ה- benchmark כולו?

שימו-לב, שה- predictor חוזה באופן שגוי את כל הקפיצות בפקודה 4 (עבור שני ה- threads):

החיזוי הראשוני taken שגוי עבור threadA. ה- BTB מתעדכן בסוף שלב ה- EX, ולכן, בסוף מחזור שעון מספר 9 מעודכן החיזוי של פקודה 4 ל- not taken. הדבר מוביל לחיזוי שגוי גם עבור threadB. במחזור שעון מספר 10 יעודכן ה- BTB חזרה ל- taken, וכעת חזרנו למצב ההתחלתי. מכאן, שכל איטרציה תחזה באופן שגוי את הקפיצה עבור שני ה- threads.

בנוסף, יהיו לנו חיזוי שגוי אחד עבור פקודה מספר 10 באיטרציה האחרונה (עבור threadA בלבד). על חיזוי שגוי נאבד מחזור שעון אחד (ה- Miss prediction penalty של המעבד כאשר ישנם שני threads הוא מחזור שעון אחד), ולכן, יש להוסיף עוד מחזור שעון.

בדומה לסעיף הקודם, יש לשים לב שאיטרציה חדשה מתחילה לפני שהאיטרציה הקודמת נגמרת, לכן, זמן ביצוע ה- benchmark כולו הינו:

$$(\#Iterations \cdot Cycles\ Per\ Iteration) + (\#misprediction \cdot misprediction\ penalty) +$$

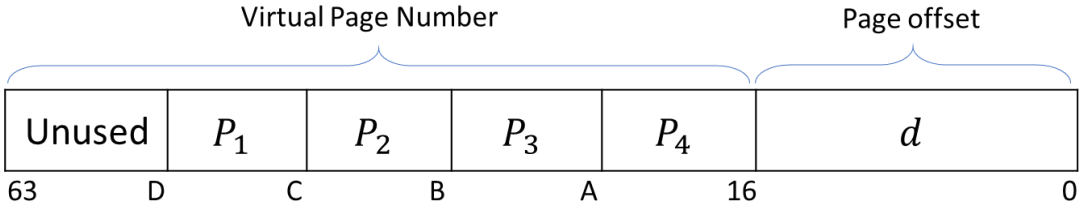
$$+(leftovers\ from\ last\ iteration) = (M \cdot 18) + (1 \cdot 1) + (4) = 18 \cdot M + 5$$

ו. (9 נקודות) הסעיף הקודם חושף בעיה במיקרו-ארכיטקטורה הנתונה. אפיינו בעיה זו והציעו שינוי מיקרו-ארכיטקטוני אשר יפתור את הבעיה. הסבירו בקצרה את המימוש. מהי עלותו ביחס למימוש הנוכחי כתלות במספר ה- threads בהם תומך המעבד (N)? נדרשת תשובה איכותית בלבד.

הבעיה נוגעת ל- branch predictor אשר מכיל חיזוי משותף לשני ה- threads. יש להפריד בין החיזויים של כל אחד מה- threads, כאשר הדרך הפשוטה ביותר לעשות זאת היא ע"י הוספה של $\lceil \log_2 N \rceil$ סיביות לזיהוי ה- thread עבור כל כניסה ב- BTB.

שאלה מספר 2 – Virtual Memory (60 נקודות)

נתון מעבד בעל מבנה דפדוף היררכי בעל ארבע היררכיות (4-level paging) התומך בדפים בגודל יחיד. מבנה הכתובת הווירטואלית הינו:



גודל כל כניסה (entry) בטבלת הדפים הוא 8 בתים, בכל אחת מהרמות.

גודל כל טבלה שונה מרמה לרמה, באופן הבא:

- גודל טבלת P_1 הוא כגודל דף
- גודל טבלת P_1 גדול פי 4 מגודל טבלת P_2
- גודל טבלת P_2 גדול פי 4 מגודל טבלת P_3
- גודל טבלת P_3 גדול פי 4 מגודל טבלת P_4

לא קיים TLB במערכת. כמו-כן, לא קיים data cache במערכת.

א. (8 נקודות) מהו ערכם של A, B, C, D ?

לפי הנתונים מתקיים:

$$|page| = |P_1| = 4 \cdot |P_2| = 4 \cdot (4 \cdot |P_3|) = 4 \cdot (4 \cdot (4 \cdot |P_4|))$$

$$|page| = |P_1| = 2^2 \cdot |P_2| = 2^4 \cdot |P_3| = 2^6 \cdot |P_4|$$

גודל ה- page offset הוא 16 סיביות, ומכאן:

$$|page| = 2^{16}B = 64KB$$

נתון שגודל הכניסות בכל טבלה הן $8B$, ולכן:

$$\#P_1\ entries = \frac{|P_1|}{|entry|} = \frac{2^{16}B}{8B} = 2^{13} = 8K$$

$$\#P_2\ entries = \frac{|P_2|}{|entry|} = \frac{2^{16}B/2^2}{8B} = 2^{11} = 2K$$

$$\#P_3\ entries = \frac{|P_3|}{|entry|} = \frac{2^{16}B/2^4}{8B} = 2^9 = 512$$

$$\#P_4\ entries = \frac{|P_4|}{|entry|} = \frac{2^{16}B/2^6}{8B} = 2^7 = 128$$

היות ו- $\#P_4 = 2^7$ לכן נדרשות 7 סיביות בשביל שדה ה- P_4, ולכן:

$$A = 16 + 7 = 23$$

היות ו- $\#P_3 = 2^9$ לכן נדרשות 9 סיביות בשביל שדה ה- P_3, ולכן:

$B = A + 9 = 32$

היות ו- $\#P_2 = 2^{11}$ לכן נדרשות 11 סיביות בשביל שדה ה- P_2, ולכן:

$C = B + 11 = 43$

היות ו- $\#P_1 = 2^{13}$ לכן נדרשות 13 סיביות בשביל שדה ה- P_1, ולכן:

$D = C + 13 = 56$

- המעבד מריץ את קטע הקוד הבא:

```
int s = 0 , i ;
for ( i = 0 ; i < 2^22 ; i++ )
        s+= arr[i*2^12] ;
```

המשתנה arr מצביע על מערך מסוג int שנמצא בזיכרון.

המשתנים s, i וכן המצביע arr מאוחסנים ברגיסטרים.

משתנה מסוג int תופס 4 בתים בזיכרון והוא מיושר (aligned).

המערך arr מתחיל בכתובת 0 וכולו יושב בצורה רציפה בזיכרון.

ב. (8 נקודות) בכמה טבלאות תרגום שונות (מכל רמות ההיררכיה) יש שימוש במהלך ריצת קטע הקוד?

התוכנית עוברת על מרחב כתובות בתחום:

$$[0: 2^{12} \cdot (2^{22} - 1) \cdot 4] = [0: 2^{36} - 2^{14}]$$

ומכאן שהסיבית הכי גבוהה שמשתנה היא סיבית מספר 35 (נמצאת בתוך שדה ה- PDP). היות והגישות לזיכרון נעשות בקפיצות של 2^{14}, הרי שהסיבית הכי נמוכה שמשתנה היא סיבית מספר 14 (נמצאת בתוך שדה ה- d). מכאן שרק השדות d, P_4, P_3 ו- P_2 משתנים.

טבלת P_1 הינה יחידה (בהגדרה).

שדה ה- P_1 איננו משתנה ← טבלת P_2 יחידה.

בשדה ה- $P_1 + P_2$ משתנות 4 סיביות [35:32] ← יש שימוש ב- 16 טבלאות P_3.

שדה ה- P_3 מקבל את כל הערכים (512 ערכים) ושדה ה- $P_1 + P_2$ מקבלים 16 ערכים ← יש שימוש ב- 8192 טבלאות P_4 שונות.

בסה"כ, במהלך ריצת קטע הקוד יש שימוש ב- 8210 טבלאות תרגום שונות.

ג. (6 נקודות) כמה גישות לזיכרון לצורך קריאת טבלאות התרגום בלבד (מכל רמות ההיררכיה) יש במהלך ריצת הקוד?

נזכור שאין TLB, וגם אין data cache. לכן, כל גישה לזיכרון בקוד דורשת 4 גישות **נוספות** בשביל התרגום. במהלך ריצת התוכנית ישנם $2^{22} = 4M$ גישות לזיכרון מהקוד, ולכן תהינה $16M$ גישות בשביל התרגום.

- על מנת לחסוך במספר הגישות לזיכרון, מוסיפים את ה- caches הבאים:

  P_3 entries cache – בעל 4 כניסות, fully associative

P_2 entries cache – בעל 4 כניסות, fully associative

P_1 entries cache – בעל 4 כניסות, fully associative

שלושת ה- caches הללו מכונים translation caches, ומטרתם לחסוך גישות לזיכרון לצורך הבאת תרגומים של ההיררכיות המתאימות. חשוב להבדיל בין ה- translation caches הללו לבין ה- TLB: בעוד שה- TLB נותן את התוצאה הסופית של מנגנון התרגום, דהיינו, את הכתובת הפיזית של הדף המבוקש, הרי שה- translation caches נותנים תוצאות ביניים בלבד (מצביעים לכתובת של תחילת הטבלה בהיררכיה הבאה).

שימו-לב כי עדיין אין TLB (עבור ה- P_4 entries). ניתן להניח כי ה- translation caches שנוספו הינם ריקים לפני תחילת ריצת הקוד.

ד. (7 נקודות) כמה גישות לזיכרון הראשי לצורך קריאת טבלאות התרגום בלבד (מכל רמות ההיררכיה) ישנן במהלך ריצת הקוד?

שדה ה- P_1 הינו קבוע ← גישה אחת בשביל שהרשומה תהיה ב- cache.

שדה ה- $P_1 + P_2$ מקבל 16 ערכים שונים, כאשר כל הבקשות לאותה רשומה ב- $P_1 + P_2$ הן ברצף (לוקאליות בזמן), ולכן די בגישה אחת לזיכרון לכל ערך בשדה ה- $P_1 + P_2$ ← 16 גישות.

שדה ה- $P_1 + P_2 + P_3$ מקבל $2^{13} = 8K$ ערכים שונים, כאשר כל הבקשות לאותה רשומה ב- $P_1 + P_2 + P_3$ הן ברצף (לוקאליות בזמן), ולכן די בגישה אחת לזיכרון לכל ערך בשדה ה- $P_1 + P_2 + P_3$ ← $8K$ גישות.

אין cache עבור ה- entries של P_4, ולכן יש צורך בגישה לזיכרון עבור כל גישה בקוד ← $2^{22} = 4M$ גישות.

בסה"כ במהלך ריצת הקוד תהינה $1 + 16 + 8K + 4M$ גישות לזיכרון בשביל התרגום.

- מעוניינים לשנות את מבנה הדפדוף ההיררכי, ולהפוך את גדלי הטבלאות כך ש:

גודל כל כניסה בטבלת הדפים הוא 8 בתים, בכל אחת מהרמות (כמו קודם).

גודל כל טבלה שונה מרמה לרמה, באופן הבא:

- גודל טבלת P_4 הוא כגודל דף (גודל הדף לא השתנה)
- גודל טבלת P_4 גדול פי 4 מגודל טבלת P_3
- גודל טבלת P_3 גדול פי 4 מגודל טבלת P_2
- גודל טבלת P_2 גדול פי 4 מגודל טבלת P_1

ה. (8 נקודות) עבור תהליך שניגש למרחב זיכרון וירטואלי רציף של 2^{40} בתים, איזו מבין האפשרויות עדיפה מבחינת גודל הזיכרון שדורשות טבלאות התרגום? הסבירו את תשובתכם.

בשתי האפשרויות הדפים הם באותו גודל, ולכן יש צורך בתרגום עבור אותו מספר של דפים. מכאן שמספר ה- entries הכולל של כל ה- P_4s זהה בשתי האפשרויות. היות ובאפשרות השנייה גודל טבלת P_4 הוא גדול יותר, הרי שישנן פחות טבלאות P_4 באפשרות השנייה, ומכאן שצריך פחות טבלאות בשאר ההיררכיות. נעשה את החישוב:

באפשרות החדשה עדיין מתקיים:

$$|page| = 2^{16}B = 64KB$$

אבל עכשיו:

$$|page| = |P_4| = 4 \cdot |P_3| = 4 \cdot (4 \cdot |P_2|) = 4 \cdot (4 \cdot (4 \cdot |P_1|))$$

$|P_4| = 64KB \quad \Rightarrow \#P_4\ entries = 8K \quad \Rightarrow 13b$

$|P_3| = 16KB \quad \Rightarrow \#P_3\ entries = 2K \quad \Rightarrow 11b$

$|P_2| = 4KB \quad \Rightarrow \#P_2\ entries = 512 \quad \Rightarrow 9b$

$|P_1| = 1KB \quad \Rightarrow \#P_1\ entries = 128 \quad \Rightarrow 7b$

<u>עבור האפשרות הראשונה:</u>

מספר טבלאות ה- P_4 שניגשים אליהם הוא: $2^{40-23} = 2^{17}\ P_4s$

מספר טבלאות ה- P_3 שניגשים אליהם הוא: $2^{40-32} = 2^{8}\ P_3s$

בנוסף, ניגשים לטבלת P_2 אחת וכן לטבלת P_1 אחת

סה"כ זיכרון:

$$|P_1| + |P_2| + 256 \cdot |P_3| + 128K \cdot |P_4| =$$

$$= 64KB + 16KB + 256 \cdot 4KB + 128K \cdot 1KB = 1104KB + 128MB$$

<u>עבור האפשרות השנייה:</u>

מספר טבלאות ה- P_4 שניגשים אליהם הוא: $2^{40-29} = 2^{11}\ P_4s$

בנוסף, ניגשים לטבלת P_3 אחת, לטבלת P_2 אחת וכן לטבלת P_1 אחת

סה"כ זיכרון:

$$|P_1| + |P_2| + |P_3| + 2K \cdot |P_4| =$$

$$= 1KB + 4KB + 16KB + 2K \cdot 64KB = 21KB + 128MB$$

ההפרש בין הקצאת הזיכרון לטבלאות הוא: $1083KB$

כל זה בהנחה שהזיכרון מיושר לתחילת כל טבלה. מה קורה אם הזיכרון לא מיושר? לכל היותר נקבל עוד טבלה מכל היררכיה:

$$1KB + 4KB + 16KB + 64KB = 85KB < 1083KB$$

ולכן, עדיין עדיפה האפשרות השנייה.

הערה: שימו-לב כי ההפרש בין גודל הטבלאות של שתי האפשרויות זניח לעומת גודל הטבלאות $(1083KB \ll 128MB)$, ועל אחת כמה וכמה שהוא זניח לעומת גודל הזיכרון $(1083KB \ll 1TB)$

- נתון מעבד אחר, בעל מרחב כתובות וירטואלי של 36 סיביות, רוחב מילה $64-bit$ וזיכרון פיזי 4GB, וכן נתון כי רזולוציית המיעון היא per-byte.
  במעבד קיים $4-way$ set associative cache בגודל 64KB. גודלו של בלוק הוא 16B וזיכרון המטמון הינו physically tagged. נתון שגודל frame הינו 4KB.

ו. (7 נקודות) כמה סיביות דרושות כדי לגשת אל ה- cache הנ"ל (index + offset)?

גודל בלוק הוא 16B, ולכן יש צורך ב- 4 סיביות בשביל offset [3:0].

מספר הבלוקים ב- cache הינו:

$$\#blocks = \frac{|cache|}{|block|} = \frac{64KB}{16B} = 2^{16-4} = 2^{12}$$

היות ומדובר ב- $4-way$ set associative cache הרי שמספר ה- sets ב- cache הינו:

$$\#sets = \frac{|cache|}{|block| \cdot 4\_way} = \frac{64KB}{16B \cdot 4} = 2^{10}$$

לכן, יש צורך ב- 10 סיביות בשביל שדה ה- index [13:4].

בסה"כ יש צורך ב- $10 + 4 = 14$ סיביות כדי לגשת אל ה- cache המתואר בשאלה.

- במעבד זה, קיים גם TLB. רוצים לגשת ל- TLB במקביל לגישה אל ה- cache, ולבצע השוואה של ה- tags בסוף הגישה המקבילית. היות ואין מספיק סיביות ב- page-offset על מנת לגשת את ה- cache, הוחלט להשתמש ב- LSBs של ה- Virtual Page Number (VPN) כדי להשלים את הסיביות החסרות.

ז. (8 נקודות) מהו גודל ה- tag שיש לשמור ב- cache על מנת לוודא שהבלוק שמצאנו ב- cache הוא אכן הבלוק המבוקש?

גודל הזיכרון הפיזי הוא $4GB$, ולכן גודל הכתובת הפיזית הוא 32 סיביות. כמו-כן, גודל frame הינו $4KB$, ולכן, גודל ה- page-offset הינו 12 סיביות.

לכן, גודל שדה ה- PFN הינו: $32 - 12 = 20$.

על מנת לגשת אל ה- cache יש צורך ב- 14 סיביות, ואילו גודל ה- page-offset הוא 12 סיביות בלבד. לכן, השתמשנו בשתי סיביות ה- LSBs מתוך ה- VPN (סיביות מספר 12 ו- 13 של הכתובת הווירטואלית). שימו-לב, כי למרות שהשתמשנו כבר בסיביות 12 ו- 13 מהכתובת הווירטואלית כדי לגשת אל ה- cache, סיביות 12 ו- 13 של הכתובת הפיזית יכולות להיות שונות (לאחר התרגום), ולכן יש צורך להשתמש בכל ה- PFN בשביל לבצע זיהוי וודאי של הבלוק המבוקש. לכן, גודל ה- tag שיש לשמור ב- cache הוא 20 סיביות, שהן סיביות [31:12] של הכתובת הפיזית.

ח. (8 נקודות) למרות השוואת ה- tags כפי שנקבע בסעיף הקודם, עדיין יכולה להיווצר בעיה במערכת הזיכרון עקב השימוש בסיביות מה- VPN לצורך גישה ל- cache. תארו את הבעיה והציעו פתרון.

הבעיה: אם יש מצב שבו שני דפים וירטואליים שונים ממופים לאותו דף פיזי (הדבר אפשרי), ושני הדפים הווירטואליים שונים בשתי הסיביות ה- LSBs של ה- VPN שלהם, אזי כאשר רוצים לגשת לכתובת מסוימת, וניגשים אל ה- cache כדי לחפש את הבלוק המבוקש בעזרת סיביות וירטואליות, הרי שאנחנו עלולים לקבל שני עותקים של אותו בלוק בשני סטים שונים ב- cache. בעיה זו נקראת synonym problem. מלבד הפגיעה בנצילות של ה- cache, ישנה בעיה חמורה יותר, שהרי אם תוכנית אחת תפנה לבלוק כדי לכתוב בו ערך אחד ותוכנית שנייה תפנה לבלוק כדי לכתוב בו ערך אחר, הרי שיש לנו באותו cache שני עותקים של אותו הבלוק, אך העותקים הם בעלי ערכים שונים!

קיימים כמה פתרונות לבעיה זו. אחד הפתרונות הפשוטים הוא שכאשר רוצים להביא בלוק אל ה- cache מבצעים קודם בדיקה ב- 4 סטים במקביל כדי לוודא שאין שם עותק נוסף של אותו הבלוק (בדיקה נעשית ע"י השוואת tags), ואם ישנו עותק כזה, יש למחוק אותו מה- cache לפני שמביאים את העותק אל הסט המתאים. באופן זה, מונעים מציאות של שני עותקים של אותו בלוק בסטים שונים ב- cache.

בעיה דומה קיימת גם כאשר מבצעים snooping, עושים זאת בעזרת כתובת פיזית. היות ושתי הסיביות ה- MSBs של האינדקס הן וירטואליות, הרי שכל כתובת פיזית יכולה להיות ממופה ל- 4 סטים שונים ב- cache, כתלות בדף הווירטואלי שממפה לאותו דף פיזי. לכן, צריך לבצע snooping במקביל לארבעה סטים, ובכל סט לארבעה בלוקים.

טבלת עזר לשאלה מספר 1 סעיף ב'

#CC	IF	ID	EX	MEM	WB	Remarks
1	1					
2	2	1				
3	3	2	1			
4	4	3	2	1		
5	4	3	-	2	1	
6	4	3	-	-	2	R4 is ready (from instr 1)
7	5	4	3	-	-	instruction 3 progress to EX
8	6	5	4	3	-	FW of R5 from inst 3 to inst 4
9	9	6	5	4	3	
10	10	9	6	5	4	
11	1	10	9	6	5	
12	2	1	10	9	6	FW of R1 from inst 9 to inst 10
13	3	2	1	10	9	FW of R1 from inst 9 to inst 1
14	4	3	2	1	10	
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						
29						
30						

טבלת עזר לשאלה מספר 1 סעיף ה'

#CC	IF	ID	EX	MEM	WB	Remarks
1	1					
2	1*	1				
3	2	1*	1			
4	2*	2	1*	1		
5	3	2*	2	1*	1	
6	3*	3	2*	2	1*	R4 for threadA is ready
7	4	3*	3	2*	2	R4 for threadB is ready
8	4*	4	3*	3	2*	
9	7	4*	4	3*	3	Predict taken A ; FW of R5 from inst 3 to inst 4
10	5*	-	4*	4	3*	Predict not-taken B ; Miss prediction for threadA, flush 7 FW of R5 from inst 3* to inst 4*
11	5	-	-	4*	4	Miss prediction for threadB, flush 5*
12	7*	5	-	-	4*	
13	6	7*	5	-	-	
14	8*	6	7*	5	-	
15	9	8*	6	7*	5	
16	9*	9	8*	6	7*	
17	10	9*	9	8*	6	
18	10*	10	9*	9	8*	
19	1	10*	10	9*	9	FW of R1 from inst 9 to inst 10
20	1*	1	10*	10	9*	FW of R1 from inst 9* to inst 10*
21	2	1*	1	10*	10	
22	2*	2	1*	1	10*	
23						
24						
25						
26						
27						
28						
29						
30						