Computer Architecture, Spring 2024
Programming Assignment (1): Cache Hierarchy Simulator

בוצע ע״י:
גיל מנצור 313373151
עומר אלנאבלסי 322598558

תוכן עניינים

1. מבוא

פרויקט זה נועד בשביל להבין לעומק את תפקוד ה-"קאש" במקרים שונים. קבוצה שונה של דפוס גישה לזיכרון תיתן כמות שונה של פגיעה במטמון, כמובן זה תלוי לאיזה שכבה במטמון כאשר השכבה קרובה ביותר למעבד (L1) זמן הגישה הוא הקצר ביותר, והשכבה הרחוקה ביותר (DRAM) זמן הגישה ארוך משמעותית מכל גישה לכל שכבה אחרת בדרך וזו הבעיה שהמטמון בא לפתור בעצם והיא שאיפה לזמני גישה נמוכים בהרבה. לכן, אנו זקוקים לכלי כלשהו כדי ליצור עקבות גישה מרובות לזיכרון. האמצעי שבו השתמשנו לתיאור התנהגות זו נעשה ע"י קוד שמייצר סימולציה בשפת C. מטרתנו היא לבדוק זמני גישה לביצוע פקודות load ו store, איך השינוי של גדלי המטמון משפיעים על זמנים אלה, וכיצד ניתן לשפרם. מערכות מחשב מודרניות מסתמכות על היררכיות זיכרון מורכבות כדי לשפר את הביצועים. הסימולציה מנתחת את הגישות לזיכרון ומדגימה את ההשפעות של מבני זיכרון שונים על ביצועי המערכת.

2. תיאור הפתרון

- **המבנה הכללי של התוכנית מחולק לשני קבצים שנפרט על כל אחד מהם מיד:**

1. main.c

הקובץ הראשון הוא הקובץ הראשי שבו הקוד בנוי ממספר חלקים עיקריים:

1. קריאת קבצי .trc
2. ניתוח כתובות הזיכרון ופעולות הקריאה והכתיבה.
3. סימולציית גישה לזיכרון בכל אחת מרמות המטמון והזיכרון הראשי (DRAM).
4. חישוב מדדים סטטיסטיים והצגת תוצאות

- **קריאת קבצי .trc:**

קובצים אלה מכילים רצף של גישות לזיכרון בפורמט מסוים. המערכת פותחת את הקובץ וקוראת כל שורה כדי לפענח את הכתובת ואת הפקודה הרצויה.

- **ניהול כתובות הזיכרון:**

לכל רגיסטר זיכרון בקובץ מתבצע תהליך של פענוח הכתובת והבנת הפעולה המתבצעת. הכתובת מומרת לפורמט של tag, index ,offset המתאים כדי שניתן יהיה לעבוד איתה בשלב הסימולציה, פורמט זה משתנה בגדלי הפרמטרים בהתאם לגודל השכבה שהוא בה מבחינת אחסון.

בעת קריאת קבצי trc. המייצגים גישות לזיכרון, ה- Valid Bit וה- Dirty Bit משפיעים על אופן הטיפול בגישות אלו במטמון:

- **קריאת נתונים**

כאשר כתובת נקראת מקובץ trc. תחילה נבדק ה-Valid Bit אם ה-Valid Bit הוא 1, הנתונים במטמון תקפים וניתן להשתמש בהם. אם ה-Valid Bit הוא 0, הנתונים אינם תקפים ויש לטעון נתונים חדשים מהזיכרון הראשי למטמון.

- **כתיבת נתונים**

כאשר כתובת נכתבת מקובץ trc. הנתונים נכתבים לכל שכבות המטמון וה-Dirty Bit של השורה מעודכן ל-1 כדי לציין שהנתונים שונו. אם יש צורך להחליף שורה במטמון (כדי לפנות מקום לנתונים חדשים), תחילה נבדוק את ה-Dirty Bit של השורה המוחלפת. אם ה- Dirty Bit הוא 1, המערכת משחזרת את הכתובת הישנה ושומרת אותה בצד, ואז שמים את הטאג של הכתובת החדשה במיקום האינדקס המתאים, לאחר מכן לוקחים את הכתובת הישנה ובודקים אם המיקום המתאים לה בשכבה אחת מעל לשכבה שהייתה בה, ומסיימים אם המקום שם פנוי ואם לא פנוי ולא מלוכלך אז דורסים. אחרת, עושים את הפעולה שוב עד שימצא מקום בשכבות ואם אין טוענים את הכתובת מהזיכרון הראשי ומוסיפים את זמן גישה זה לחישובים הסטטיסטיים.

- **<u>סימולציית גישה למטמון:</u>**

גישה למטמון L1

השלב הראשון בסימולציה הוא ניסיון גישה לכתובת בזיכרון מטמון L1 אם הכתובת קיימת במטמון, נרשום פגיעה (hit) ומעדכנים את הסטטיסטיקה המתאימה, במקרה של החטאה מעדכנים את ההחטאות, עוברים לשכבות הבאות לשם בדיקה שכל החטאה שם נכללת גם בחשבון.

גישה למטמון L2 ו-L3

אם הכתובת אינה נמצאת במטמון L1 נעשה ניסיון לגשת לכתובת במטמון L2 ולאחר מכן במטמון L3 , אך כמה פרמטרים עוברים שינוי בבדיקת הכתובת ברמות שונות במטמון וזה ע״י פירוק הכתובת לשלושה פרמטרים מרכזיים שהם טאג, אינדקס, וסטייה שבעזרתם מתבצעת בדיקה והשוואה לקביעה אם הכתובת נמצאת (פגיעה) או לא (החטאה). גם כאן, אם הכתובת נמצאת, נרשם פגיעה ומעודכנות הסטטיסטיקות, אם לא ״נשלוף״ את הכתובת מהזיכרון הראשי במקרה שהיא לא נמצאה בשכבה האחרונה.

גישה ל-DRAM

אם הכתובת אינה נמצאת באף אחד מהמטמונים, מתבצע חישוב עלות גישה לזיכרון DRAM, החישוב מתבצע על פי נתונים כמו (RAS ו CAS).

- **<u>חישוב מדדים סטטיסטיים:</u>**

שיעור החמצות (Miss Rate)

שיעור החמצות מחושב עבור כל רמת מטמון לפי הנוסחה :

L# missrate = (L#->miss) / (L#->miss + L#->hits)

זמן גישה ממוצע לזיכרון (AMAT)

זמן הגישה הממוצע לזיכרון מחושב על פי הנוסחה:

AMAT = L1->access + L1_missrate * (L2->access + L2_missrate * (L3->access + L3_missrate * (CAS + DRAM_missrate * (RAS + CAS))))

לאחר חישוב המדדים הסטטיסטיים, התוצאות מוצגות כדי לאפשר ניתוח והשוואה בין הקבצים השונים שקיבלנו.

<u>configuration.h</u> ***<u>.2</u>***

הקובץ השני הוא הקונפיגורציה של התוכנית בכלל ולקובץ הראשי בפרט, המכיל הגדרות ותצורות חיוניות בשימוש בכל הפרויקט. מרכיבי מפתח כוללים:

- **<u>מאקרו וקבועים:</u>** קבועים ופקודות מאקרו שונות מוגדרות לסטנדרטיזציה של ערכים והגדרות המשמשים לאורך הפרויקט. זה מבטיח עקביות ומקל על ניהול השינויים במקום מרכזי אחד.

- **<u>פרמטרי תצורה:</u>** פרמטרים עבור רכיבים שונים, כגון גדלי מאגר, ערכי זמן קצוב והגדרות אחרות כמו מבנים של cache, cacheline ומבנה לכתובת שמקנה לנו קלות רבה בגישה ישירה לחלקים כמו tag, index, offset. פרמטרים אלו מאפשרים כוונון עדין של התנהגות האפליקציה מבלי לשנות את בסיס הקוד הראשי.

3. אמצעי מחקר

כלי כתיבת הסימולציה בו השתמשנו הוא visual studio, שפת הכתיבה היא שפת c. התוצאות שהתקבלו אחרי הרצת הקוד היו תוצאות מספריות שהדפסנו על ה cmd, אותם מספרים נלקחו ונותחו ע״י גרפים שייצרנו ב-MATLAB שמדמים בבירור את ייחוס התוצאות לשאלות המחקר הנדרשים.

4. תיאור וניתוח התוצאות הניסוייות

Cache parameters:	**DRAM parameters:**
Size: L1 16KB, L2 512KB, L3 2MB. Associativity: All direct mapped Hit time (access time): L1 1cyc, L2 6cyc, L3 30cyc Write allocate Write Back	Bus width 4bytes 1 channel 1 DIMM 4 banks RAS time: 100 Cyc CAS time: 50 Cyc

- קיבלנו קבצי trace של תוכניות אמיתיות ועבור הסימולציה שבנינו עיבדנו את קבצים אלו כך שיכילו שורות עדכון לרגיסטרים בתוכנית ושורות המכילות פקודות store, load.

ראשית נריץ את הסימולציה, נבחן ונגזור מסקנות כלליות עבור נתוני הבסיס :

```
coremark_val_filtered.trc :
Total memory accesses: 1700981
L1 cache hits: 1700733
L1 cache misses: 248
L1 miss rate : 0.000146
L1 Avarage Memory Access Time is : 1.023474
L2 cache hits: 0
L2 cache misses: 248
L3 cache hits: 0
L3 cache misses: 248
DRAM time spend: 37200
```

```
dhrystone_val_filtered.trc :
Total memory accesses: 3903655
L1 cache hits: 3903580
L1 cache misses: 75
L1 miss rate : 0.000019
L1 Avarage Memory Access Time is : 1.003093
L2 cache hits: 0
L2 cache misses: 75
L3 cache hits: 0
L3 cache misses: 75
DRAM time spend: 11250
```

```
fibonacci_val_filtered.trc :
Total memory accesses: 4418606
L1 cache hits: 4418505
L1 cache misses: 101
L1 miss rate : 0.000023
L1 Avarage Memory Access Time is : 1.003680
L2 cache hits: 0
L2 cache misses: 101
L3 cache hits: 0
L3 cache misses: 101
DRAM time spend: 15150
```

```
linpack_val_filtered.trc :
Total memory accesses: 756810
L1 cache hits: 749227
L1 cache misses: 7583
L1 miss rate : 0.010020
L1 Avarage Memory Access Time is : 1.933415
L2 cache hits: 3319
L2 cache misses: 4264
L3 cache hits: 0
L3 cache misses: 4264
DRAM time spend: 639600
```

כדי ליצור שפה משותפת נמספר את הקבצים ונדבר עליהם כ"קובץ x" ונשתדל ככל שניתן לסדר משמאל לימין את הקבצים בסדר עולה.

קובץ 1 -Cormark, קובץ 2 – Dhrystone, קובץ 3 – Fibonacci, קובץ 4 – Linpack.

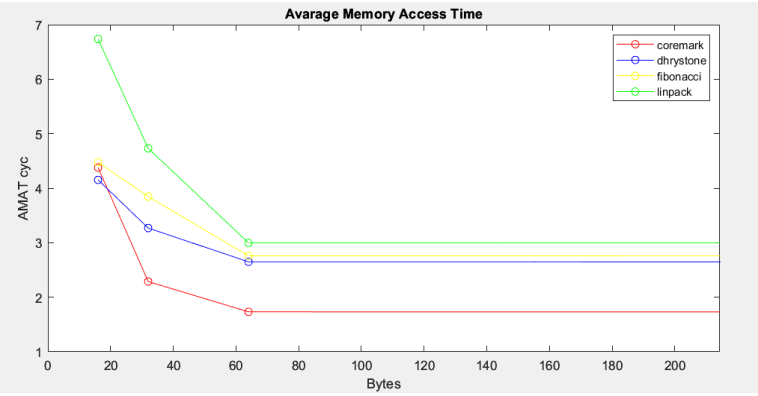
- ניתן לראות כי הקבצים 1-3 מכילים "פגיעות" רק ברמה L1 והמשמעות הנגזרת מכאן היא שכאשר קיבלנו כתובת שלא נמצאת ברמה L1 אזי שהיה לה מקום פנוי ברמה זאת ובשאר הפעמים בתוכנית שוב נתבקשו אותן כתובות הנמצאות כבר ברמה זאת. נוסיף ונציין כי הכתובות המדוברות הובאו מהזיכרון ונשמרו בכל הרמות ועל כן יש לנו את אותם "פספוסים" בכל הרמות.
- בנוסף ניתן להסיק כי מספר הכתובות השונות הנמצאות ברמה L1 הוא שווה למספר "הפספוסים" (רק עבור קבצים 1-3).
- לגבי זמן הגישה הממוצע לזיכרון, ניתן לראות ככל שאחוזי הפגיעה גבוהים זמן הגישה הממוצע יתקרב לזמן הגישה לרמה L1.
- אם נגדיל את L1 אז רק עבור קובץ 4 נראה שינוי מכיוון שיהיו יותר שורות ויותר כתובות שונות שיוכל להכיל, לכן גם בהמשך נתעסק בשינוי הגדלים ונראה את השפעתן על זמני הגישה.

<u>שאלת המחקר 1:</u> בהינתן נתוני הבסיס הציגו גרפים של ניתוח של זמן הגישה הממוצע לזיכרון ושיעור ההחטאות של ה-L1 cache כאשר גודל ה-cache הוא 16, 32, 64, 128KB. גודל יתר הפרמטרים על פי נתוני הבסיס.

כיצד באופן כללי אנו מחשבים את זמן הגישה הממוצע לזיכרון.
הנוסחה הכללית היא :

Average memory access time = Hit time + Miss rate * Miss penalty
Miss rate = Number of Misses / Total access to cache
L1 Miss Penalty = (L2 Hit time + L2 miss rate * (L3 Hit time + L3 miss rate * DRAM AMAT))

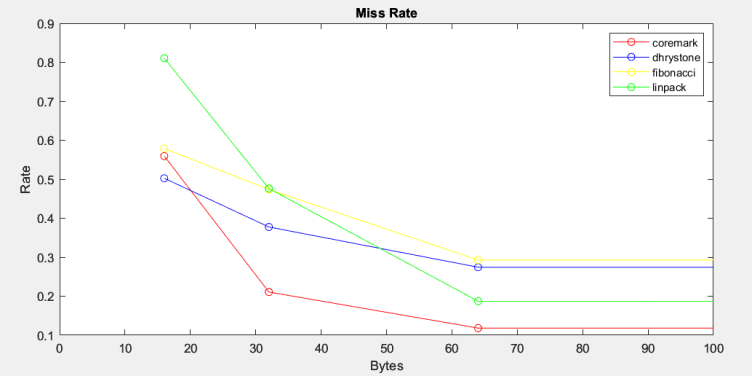
<u>גרף 1</u>



על מנת להציג את הגרף באופן שימחיש את השינויים בחרנו להראות את הגרף ללא הנקודה של 128KB כאשר שם השאיפה של קבצים 1-3 הוא ל-1 Cyc ובקובץ 4 ל-1.9 Cyc.

על פי הגרף שקיבלנו ניתן לראות שכאשר אנו מגדילים את גודל המטמון זמן הגישה הממוצע הוא קטן. בהתחלה הוא אכן גבוהה יותר מהסיבה שיש המון פגיעות ב-L2 ואז זמן הגישה מתקרב לזמן הממוצע של מטמון זה. כאשר המטמון L1 גדול מספיק זמן הגישה הממוצע מתקרב לזמן הגישה למטמון זה.

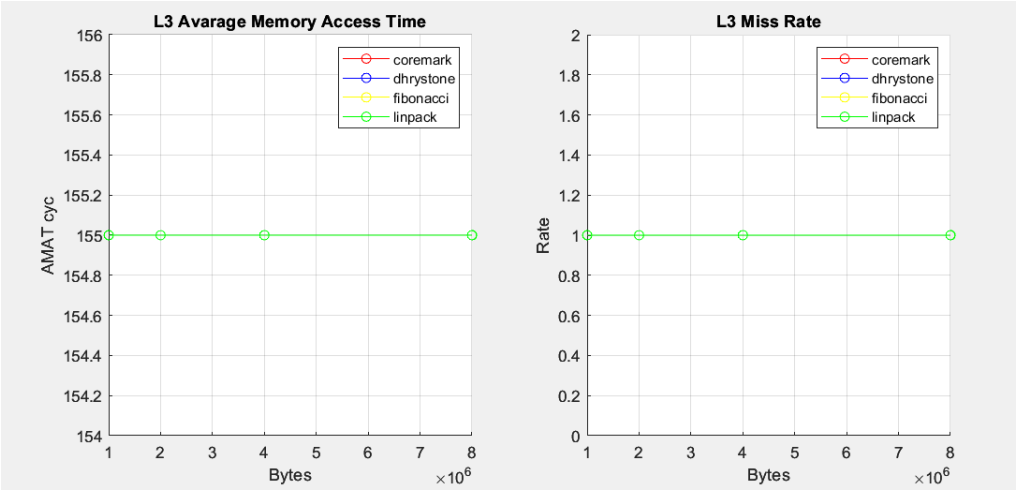
<u>גרף 2</u>



גם כאן נרמלנו את הגרף כדי לצפות בתוצאות. בדומה לגרף הקודם ניתן לראות שכאשר המטמון גדול יותר שיעור ההחטאות קטן, וזאת מכיוון שכאשר המטמון גדול יותר יש לנו יותר שורות להכיל כתובות ודבר זה יגרור שכתובות שננסה לשלוף יהיו במטמון.

שאלת מחקר 2: בהינתן נתוני הבסיס הציגו גרפים של ניתוח של זמן הגישה הממוצע לזיכרון ושיעור ההחטאות של ה-cache L3 כאשר גודל ה-cache הוא 1, 2, 4, 8 MB. גודל יתר הפרמטרים על פי נתוני הבסיס.

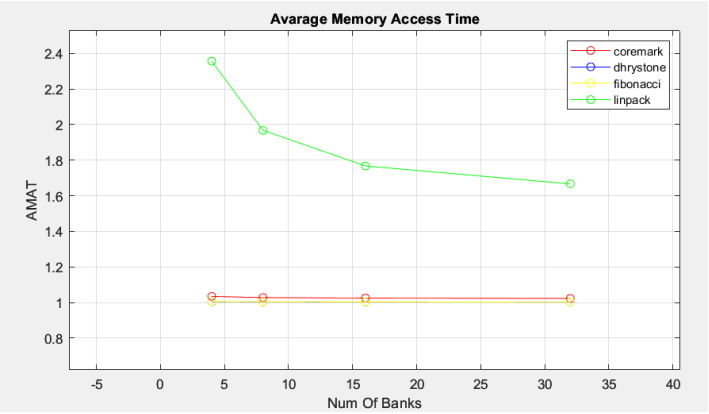
גרף 3



עבור המטמון L3 ניתן לראות שלא יתבצע כל שינוי מכיוון שלאחר שאנו מאתחלים את כל הכתובות השונות הCPU לא מגיע שוב ל-L3 מכיוון שכל המידע כבר נטען למטמונים L1 L2 אז הצורך בגישה לרמה L3 זניחה.

שאלת מחקר 3: עבור מערכת עם נתוני הבסיס ניתחנו את השפעת מספר הבנקים על זמן הגישה הממוצע לזיכרון. הבדיקה התבצעה עבור מספר בנקים של 4, 8, 16 ו 32. -יתר הפרמטרים על פי נתוני הבסיס .

גרף 4



הגרף מראה בבירור שעבור linpack, הגדלת מספר הבנקים מובילה לירידה משמעותית בזמן הגישה הממוצע לזיכרון. עבור הקבצים האחרים, זמן הגישה הממוצע נשאר כמעט קבוע. מה שמראה שהם אינם מנצלים את היתרונות של גישה לבנקים מרובים ומהסיבה שיש לנו יותר מידע שהדף זיכרון פתוח עליו.

<u>שאלת מחקר 4:</u> עבור המערכת עם נתוני הבסיס ניתחנו את השפעת זמן ה RAS -על זמן הגישה הממוצע לזיכרון. הנחנו את טווח הערכים הבאים 50, 100, 200, 250 מחזורי שעון. יתר הפרמטרים על פי נתוני הבסיס .

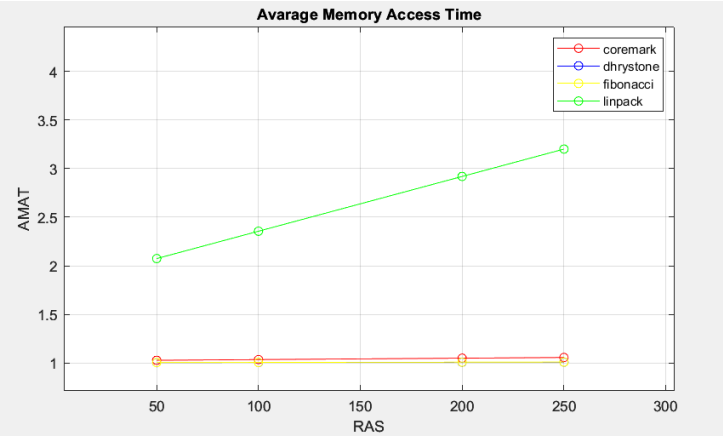
עומסים שאינם רגישים לזמן ה-RAS

- coremark, dhystone ו fibonacci-אינם מושפעים משינויים בזמן ה-RAS מכיוון שכנראה יש להם גישות רציפות או גישות מקומיות שאינן דורשות פתיחות שורה רבות.

עומסים רגישים לזמן ה-RAS

- linpackמושפע מאוד משינויים בזמן ה-RAS מכיוון שיש לו דפוס גישות שמחייב פתיחות שורה רבות. כאשר זמן ה- RAS עולה, כך גם זמן ההמתנה לפתיחת השורה וכתוצאה מכך זמן הגישה הממוצע לזיכרון.

<u>גרף 5</u>

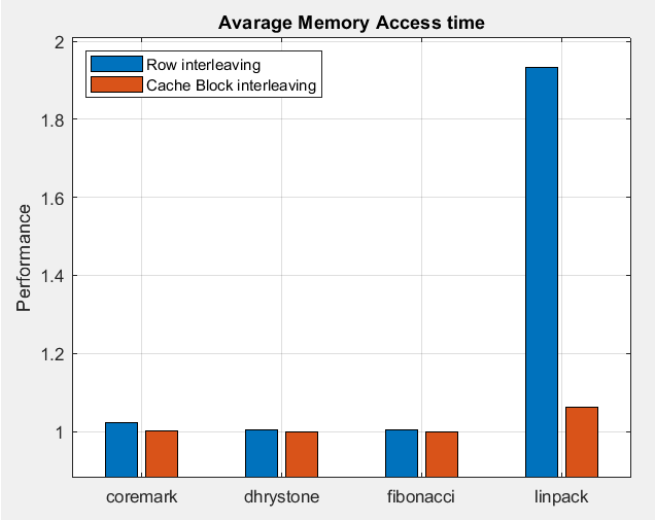


הגרף מראה בבירור שעבור עומס linpack, עלייה בזמן ה RAS-גורמת לעלייה משמעותית בזמן הגישה הממוצע לזיכרון. עבור העומסים האחרים, זמן הגישה הממוצע נשאר כמעט קבוע, מה שמראה שהם אינם מושפעים באותה מידה מזמן ה .RAS-הבנת השפעת זמן ה-RAS חשובה במיוחד עבור אופטימיזציה של ביצועי הזיכרון בעומסים עם דפוסי גישות שונים.

שאלת מחקר 5: השוונו את ביצועי המערכת (זמן גישה ממוצע לזיכרון) עבור מערכת נתוני הבסיס כאשר שיטת המיפוי היא Row interleaving לעומת Cache block interleaving .

- **Row Interleaving** בשיטה זו, שורות זיכרון עוקבות מאוחסנות בבנקים שונים של זיכרון, כך שקריאות וכתיבות רציפות יכולות להתבצע במקביל. שיטה זו מפחיתה את התנגשותי הזיכרון ומאפשרת גישה מהירה יותר, מתאים יותר לעומסי עבודה שדורשים גישה רציפה ומקבילה לשורות זיכרון שונות. הוא מפחית עיכובים על ידי פיזור גישות הזיכרון על פני בנקים שונים.
- **Cache Block Interleaving** בשיטה זו, בלוקים שלמים של מטמון מאוחסנים בבנקים שונים של זיכרון. השיטה הזו מאפשרת גישה מהירה יותר לבלוקים גדולים של נתונים אך יכולה ליצור עיכובים במקרה של גישות אקראיות. מתאים לעומסי עבודה שדורשים גישה לבלוקים גדולים של נתונים. אם העומס אינו מנצל את המבנה הזה, ביצועיו יהיו פחות טובים לעומת Row Interleaving.

גרף 6



הגרף מראה בבירור שלעומסי עבודה מסוימים כמו Linpack, Row Interleaving מציג ביצועים טובים בהרבה מ Cache Block-Interleaving בעומסים אחרים, שתי השיטות מציגות ביצועים דומים. הבחירה בשיטת מיפוי צריכה להיות תלויה באופי העומס ובסוג הגישות לזיכרון שמערכת העבודה דורשת.

5. סיכום

הפרויקט היווה הזדמנות מצוינת להעמיק את הידע וההבנה של הארכיטקטורות המתקדמות והשפעתן על ביצועי המערכת בפרמטרים שונים כמו גודל המטמון, השפעת גודל זמני הגישה, מספר הבנקים ושיטות המיפוי על תזמוני מערכת הזיכרון. בניית הסימולטור והתנסות בהרצות ובניתוח התוצאות העניקו לנו כלים חשובים להבנה מעמיקה של היררכיית הזיכרון והשפעתה על ביצועי המערכת, כאשר הסימולטור שייצרנו מדמה שיטת מיפוי מסוג ישיר (Direct Map), התמודדות עם write back ו write allocate . התרומה המרכזית של העבודה שביצענו היא פיתוח כלי שמאפשר למדוד ולנתח את ביצועי הזיכרון במערכות מחשב, מה שיכול לסייע בעתיד בתכנון ושיפור מערכות מחשב מתקדמות. בנוסף לכך, חשיבה ביקורתית על הפרויקט מעלה כי המימוש היה יעיל מבחינת ביצוע הקוד ומהירות קבלת התוצאות והסימולטור עמד במטרות שהצבנו לו. יחד עם זאת, יש מקום לשיפור בממשק המשתמש ובאפשרות להפוך את הסימולטור למערכת שיכולה לזהות כתובות שהשימוש שלהם מינימלי יחסית לכתובות אחרות (LRU) וע״י כך יוחלפו במטמון.