ת"ב 2 – זיכרון מטמון Cache

קצר ולעניין

בתרגיל זה תממשו סימולטור Cache משלכם, בדומה לנלמד בכיתה. תצורת ה- cache תהיה גמישה ותוגדר בתחילת הריצה באמצעות פרמטרים. בנוסף, הסימולטור שלכם יקרא קובץ קלט שבו יהיו מפורטות הגישות לזיכרון. הסימולטור שלכם יצטרך לחשב את ה- Hit/Miss rate ואת זמן הגישה הממוצע לזיכרון עבור התצורה שנקבעה ועבור ה-trace (קובץ הקלט).

מאפייני סימולטור ה-cache

- הסימולטור ידמה גישות data בלבד
- הסימולטור יכיל שתי דרגות (L1 ו- L2)
- שני ה-cache-ים יעבדו במדיניות Write Back ומדיניות Write Allocate או No Write Allocate
- ה- cache יעבוד על פי עקרון ההכלה (inclusive)
- מדיניות הפינוי הינה LRU
- בתחילת הריצה ה-cache יהיה ריק
- כל הגישות הן בגודל 4 בתים ומיושרות לפי גבול של ארבעה בתים (שתי הסיביות התחתונות של הכתובת תהיינה תמיד 00)
- הסימולטור יכול להכיל Victim Cache

המאפיינים שניתן לקבוע באמצעות פרמטרים לאיתחול:

- גודל ה- cache
- גודל הבלוק
- רמת האסוציאטיביות
- זמני הגישה (במחזורי שעון)
- מדיניות Write Allocate או No Write Allocate
- עם/ללא מנגנון Victim Cache

מבנה קובץ הקלט

השורות של קובץ הקלט מכילות עקבות (trace) של גישות לזיכרון מריצת תכנית כלשהי, כאשר כל שורה תכיל תיאור של גישה שכזאת במבנה של 2 שדות עם רווח ביניהם:

1. קריאה (r) או כתיבה (w)
2. הכתובת ממנה קוראים או כותבים (ב- hexa)

לדוגמא:

```
w 0x1ff91ca8
r 0x20000cdc
```

כמו כן, מסופקת לכם דוגמא לקובץ קלט עם חומרי התרגיל.

אז מה תכל'ס צריכים לעשות?

התכנית שלכם תקרא cacheSim וההרצה שלה תהיה כזו:

```
./cacheSim <input file> --mem-cyc <# of cycles> --bsize <block log2(size)>
     --wr-alloc <0: No Write Allocate; 1: Write Allocate>
     --l1-size <log2(size)> --l1-assoc <log2(# of ways)> --l1-cyc <# of cycles>
     --l2-size <log2(size)> --l2-assoc <log2(# of ways)> --l2-cyc <# of cycles
     --vic-cache <0: No Victim Cache;  1: there is Victim Cache>
```

כאשר גודל ה-cache, גודל הבלוק ורמת האסוציאטיביות הינם בחזקות של 2 (מספר cycles אינו חזקה של 2). כל המספרים הינם שלמים. גודל ה-cache מתייחס לחלק ה-data ולא מתייחס לגודל ה-tag directory (במילים אחרות, גודל ה-cache הנתון בארגומנטים מתייחס לכמות *המידע* שניתן לשמור ב-cache).

ניהול Victim Cache:

במקרה של Miss ב-L2 במקום להביא את הבלוק מהזיכרון צריך קודם לחפש ב-Victim Cache, אם יש hit אז צריך להעביר את המידע ל- L1 ו- L2 , אחרת ניגשים לזיכרון ומביאים את המידע רק ל- L1 ו- L2 (לא ל- Victim Cache). במקרה של פינוי שורה מ-L2 במקום להחזיר אותה לזיכרון, צריך לשמור אותה ב-Victim Cache.

המאפיינים של Victim Cache:

- גודל בלוק ב- Victim Cache זהה לגודל בלוק ב- L1 ו- L2.
- Victim Cache יכיל 4 בלוקים בשיטת ארגון Fully Associative (ללא תלות בגודל L1\L2).
- מדיניות פינוי FIFO.
- זמן הגישה הוא מחזור שעון אחד.

למשל, אם נרצה לדמות זמן גישה של 100 cycles לזיכרון הראשי, גודל בלוק של 32B, L1 cache בגודל 64KB ו- 8 ways עם זמן גישה של cycle בודד ובנוסף L2 cache בגודל 1M ו- 16 ways עם זמן גישה של 5 cycles, וכן שה-cache יהיה במדיניות Write Allocate והסימולטור יכיל Victim Cache. עבור trace ששמו example.in, נשתמש בפקודה הבאה:

```
./cacheSim example.in --mem-cyc 100 --bsize 5 --wr-alloc 1
--l1-size 16 --l1-assoc 3 --l1-cyc 1 --l2-size 20 --l2-assoc 4 --l2-cyc 5
--vic-cache 1
```

זמני הגישה לרמות השונות אינם כוללים את זמן הגישה לרמות הקודמות. לדוגמה, זמן הגישה במקרה של פספוס ב-L1 ופגיעה ב-L2 הוא:

$$t_{access} = t_{L1} + t_{L2}$$

במקרה של פספוס ב-L1 וב-L2, זמן הגישה יהיה:

$$t_{access} = t_{L1} + t_{L2} + t_{mem}$$

באותו אופן אם יש Victim Cache במערכת ויש פספוס ב- L1 ו- L2, ופגיעה ב- Victim Cache, זמן הגישה יהיה:

$$t_{access} = t_{L1} + t_{L2} + t_{victim}$$

אין צורך להתחשב בתקורות של writeback (שקורות ברקע בדר"כ, ולכן בסופו של דבר לא משפיעות ישירות על זמן הגישה). כלומר, לצורך התרגיל הזה זמן הגישה לא משתנה אם שורה הייתה dirty או לא.

גישה מ-L1 ל-L2 כתוצאה מ-writeback אינה תשפיע על חישוב ה-Miss Rate של L2.

הפלט של התכנית יהיה:

```
L1miss=<L1 miss rate> L2miss=<L2 miss rate> AccTimeAvg=<avg. acc. time>
```

השדות <L1/L2 miss rate> יהיו שברים עשרוניים בין 0 ל-1 (כלומר, לא באחוזים) בדיוק של 3 ספרות אחרי הנקודה בדיוק. השדה <avg. acc. time> יהיה ממוצע זמן גישה על פני כל הגישות במחזורי שעון בדיוק של 3 ספרות אחרי הנקודה בדיוק. יש לעגל את המספרים לרמת הדיוק המבוקשת על פי כללי עיגול רגילים (לקרוב ביותר).

יש להוסיף סימן שורה חדשה (\n) בסוף השורה הנ"ל (ראו קובץ פלט לדוגמא המצורף לתרגיל).

שימו לב !!!

- בדיקת התכניות שלכם תתבצע באופן אוטומטי. באחריותכם לספק פלט זהה לזה שהוגדר.
- עליכם לכתוב את התכנית ב- C או ב- ++C בלבד ולספק קובץ makefile בשביל הקומפילציה.

דרישות ההגשה

הגשה אלקטרונית בלבד באתר הקורס ("מודל") מחשבונו של אחד הסטודנטים.

מועד ההגשה: עד 12.05.2019 בשעה 23:55.

עליכם להגיש קובץ tar.gz בשם hw2_*ID1_ID2*.tar.gz כאשר ID1 ו-ID2 הם מספרי ת.ז. של המגישים. לדוגמה: hw2_012345678_987654321.tar.gz . ה-tar יכיל:

- קוד המקור של סימולטור ה-cache שלכם
  קוד המקור חייב להכיל תיעוד פנימי במידה סבירה על מנת להבינו
- `makefile` בשביל הקומפילציה. שימו לב כי התכנית (קובץ הריצה) שתייצרו תקרא `cacheSim`

דגשים להגשה:

1. המימוש שלכם **חייב** להתקמפל בהצלחה ולרוץ במכונה הוירטואלית שמסופקת לכם באתר הקורס. זוהי סביבת הבדיקה המחייבת לתרגילי הבית. יש לוודא בניה מוצלחת במכונה הוירטואלית באמצעות קובץ ה-makefile שמסופק לכם וקבצי העזר המקוריים. **קוד שלא יתקמפל יגרור ציון 0!**
2. מניסיונם של סטודנטים אחרים: הקפידו לוודא שהקובץ שהעלתם ל"מודל" הוא אכן הגרסה שהתכוונתם להגיש. לא יתקבלו הגשות נוספות לאחר מועד ההגשה שנקבע בטענות כמו "משום מה הקובץ במודל לא עדכני ויש לנו גרסה עדכנית יותר שלא נקלטה".

בהצלחה!