ת"ב 1 – חזאי סיעוף – Branch Predictor

קצר ולעניין

בתרגיל זה תממשו מדמה (Simulator) חזאי סיעוף בעל שתי רמות (2-Level Branch Predictor), בדומה לנלמד בכיתה. תצורת החזאי תהיה גמישה ותוגדר בתחילת הריצה של המדמה באמצעות פרמטרים. סביבת המדמה תספק trace (עקבות ריצה) מריצת תכנית כלשהי, שיתאר את אירועי הסיעוף בלבד – כתובת פקודה, הכרעת הקפיצה וכתובת יעד מחושבת. המדמה יידרש לתת חיזוי לכל אירוע סיעוף (על פי כתובת הפקודה בלבד) ולאחר מכן לעדכן את מצבו בהתאם לזיהוי הפקודה בפועל והכרעת הסיעוף (תוצאת שלב ה-EXE) בפועל, כפי שמפורט ב-trace. הפרמטרים וה-trace מסופקים בקובץ קלט שנקרא על ידי סביבת המדמה שמסופקת לכם.

מאפייני החזאי

תצורת החזאי, כלומר המאפיינים שניתן לקבוע באמצעות פרמטרים לאיתחול, כוללים:

- גודל טבלת ה-BTB: מספר כניסות בטבלה. ערכים אפשריים: 1,2,4,8,16,32
- גודל רגיסטר ההיסטוריה: מספר ביטים. ערכים אפשריים: 1 עד 8
- גודל שדה ה-tag ב-BTB: מספר ביטים. ערכים אפשריים: $0...(30 - \log_2[BTB\ Size])$
- מכונות המצבים בטבלאות הן Bimodal (2-ביטים) כפי שנלמדו בכתה. המצב ההתחלתי נקבע להיות אחד מארבעת המצבים {SNT = 0, WNT = 1, WT = 2, ST = 3}
- היסטוריה לוקלית או גלובלית
- טבלאות מכונות מצבים לוקליות או גלובליות
- שימוש ב-Lshare/Gshare – כן או לא – רלוונטי רק כשטבלת מכונות המצבים היא גלובלית

מאפיינים נוספים (קבועים):

- גודל כתובת במעבד הינו 32 ביט. כל פקודה מתחילה בכתובת מיושרת ל-4, כלומר שני הביטים התחתונים של כתובת פקודה תמיד 00. ה-tag מתחיל בביט ה-$\log_2[BTB\ Size] + 2$ (הספירה מתחילה מ-0) – ראה דוגמה בהמשך. גודל ה-tag עשוי להיות *קטן* ממספר הביטים הנדרשים לזיהוי חד-משמעי של כתובת הפקודה בטבלה. לפרטים נוספים על ארגון ה-BTB ראה בפרק "ארגון ה-BTB" בהמשך התרגיל.
- using_share_lsb: ה-XOR במקרה של share-L/G יהיה עם הביטים של כתובת הפקודה החל מביט 2 (השלישי).
- ללא קשר לגודל ה-tag. כלומר, גם אם ה-tag קטן מגודל רגיסטר ההיסטוריה, נשתמש במספר ביטים מה-PC
- של כתובת פקודת הסיעוף המתאים לגודל רגיסטר ההיסטוריה, החל מביט 2.
- using_share_mid: ה-XOR במקרה של share-L/G יהיה עם הביטים של כתובת הפקודה החל מביט 16, ללא
- קשר לגודל ה-tag. כלומר, גם אם ה-tag קטן מגודל רגיסטר ההיסטוריה, נשתמש במספר ביטים מה-PC של
- כתובת פקודת הסיעוף המתאים לגודל רגיסטר ההיסטוריה, החל מביט 16.

מבנה קובץ הקלט

תצורת החזאי וה-trace נקראים על ידי ה-main שמסופק לכם מתוך קובץ קלט ששמו ניתן בשורת הפקודה של bp_main. מסופקות לכם מספר דוגמאות לקבצי קלט עם חומרי התרגיל.

השורה הראשונה בקובץ הקלט מכילה רצף שדות של מאפייני החזאי עם רווח בין כל אחד מהם:

1. מספר הכניסות ב-BTB
2. מספר הביטים בהיסטוריה
3. מספר הביטים ב-tag
4. המצב ההתחלתי של מכונות המצבים
5. local_history או global_history
6. local_tables או global_tables
7. using_share_lsb או using_share_mid או not_using_share

לדוגמה:

```
16 5 20 1 global_history global_tables not_using_share
```

כל השורות הבאות בקובץ מכילות עקבות (trace) של פקודות סיעוף מריצת תוכנית כלשהי, כאשר כל שורה תכיל תיאור של עיבוד פקודת סיעוף אחת במבנה של 3 שדות עם רווח ביניהם:

1. כתובת פקודת הסיעוף.
2. הכרעת הסיעוף: T (Taken) או N (Not taken).
3. כתובת יעד קפיצה מחושב (גם אם Not taken).

לדוגמה:

```
0x1036 N 0x1050
```

אז מה עושים?

אנחנו מספקים לכם קובץ bp_main.c שכבר מבצע קריאה של קובץ הקלט כמפורט לעיל. ה-main קורא לפונקציית החזאי שלכם על מנת לאתחל אותו עם התצורה שנקראה ולאחר מכן קורא לפונקציות לחיזוי ועדכון מצב החזאי, שאותן עליכם לממש.

עליכם לממש את החזאי בקובץ bp.c או bp.cpp. על החזאי לממש את הפונקציות המוגדרות בקובץ bp_api.h:

1. פונקציית איתחול החזאי. מגדירה את תצורת החזאי לפני תחילת עבודה.

```
int BP_init(unsigned btbSize, unsigned historySize, unsigned tagSize,
unsigned fsmState, bool isGlobalHist, bool isGlobalTable, int isShare);
```

2. פונקציית חיזוי לשימוש בשלב כמו IF. מקבלת ערך PC של פקודה נוכחית. אם הפקודה מוכרת בחזאי כפקודת סיעוף, היא מחזירה חיזוי קפיצה – true עבור Taken ו-false עבור Not-taken. במקרה של חיזוי Taken, בפרמטר dst יוחזר ערך PC של יעד הקפיצה המחושב שנשמר בחזאי עבור אותה פקודה. עבור כתובת פקודה שלא מוכרת כפקודת סיעוף בחזאי או מוכרת שזוהתה כסיעוף אבל בחיזוי NT, יש להחזיר false ובפרמטר יעד הקפיצה את הערך pc+4.

```
bool BP_predict(uint32_t pc, uint32_t *dst);
```

3. פונקציית עדכון מצב חזאי עבור פקודת סיעוף. משמשת לאחר שלב ה-EXE לעדכון מצב החזאי בהתאם להכרעת פקודת הסיעוף. עדכון מצב החזאי כולל הכנסת רשומה חדשה ל-BTB. בתרגיל זה תמיד מתבצעת הכנסת רשומה חדשה ל-BTB גם אם פקודת הסיעוף היא Not-taken. שימו-לב כי בשונה מדיאגרמת הזמנים שניתנה כדוגמה בתירגול 3 שקף 16, במימוש בסימולטור בתרגיל זה אנו מניחים שכל העדכונים נעשים בשלב ה-EXE, כולל יצירת כניסה חדשה. ה-pc שניתן כפרמטר הראשון לפונקציה זו הינו בהכרח PC השייך לפקודה סיעוף.

```
void BP_update(uint32_t pc, uint32_t targetPc,
               bool taken, uint32_t pred_dst);
```

`targetPc` היא הכתובת המחושבת לקפיצה, גם אם היא not taken (מה שיישמר ב-BTB).
`pred_dst` היא הכתובת הספקולטיבית לקפיצה (שהוחזרה מ-BP_predict).

4. פונקציית החזרה של סטטיסטיקה אודות הסימולטור.

```
void BP_GetStats(SIM_stats *curStats);
```

על הסימולטור לשמור את כמות פקודות הסיעוף הכוללת שטופלה על ידו (קריאות ל-update) וכמות החיזויים שגרמו ל-flush-ים , מכל סיבה. חלוקה של השני בראשון תיתן מדד ליעילות החזאי (כפונקציה של הפרמטרים של תצורת החזאי). בנוסף, יש להחזיר את גודל הזיכרון (התאורטי) הנדרש לחזאי בביטים (כולל ה-valid bits). מובטח כי פונקציה זו נקראת פעם אחת בסוף הסימולציה. ניתן להניח כי פקודה זו תיקרא פעם אחת בסוף הסימולציה כך שניתן לבצע בפקודה זו שחרור הקצאות זיכרון דינאמיות.

שימו-לב: ה-main שניתן נועד להקל עליכם בבדיקה, אולם אתם מחויבים למימוש הממשק לחזאי כפי שמוגדר ב-bp_api.h. כלומר, ייתכן והחזאי ייבדק בדרכים שונות מהמודגם ב-main.

ארגון ה-BTB

טבלת ה-BTB מקצה כניסה לכל פקודת סיעוף שנתקלים בה במהלך ריצה (עד כדי מגבלת מקום). על מנת לייעל חיפוש בטבלה, לעיתים קובעים את מספר הכניסה בטבלה עבור פקודה מסוימת באופן עקבי, על ידי שימוש במספר ביטים מכתובת הפקודה בזיכרון. **לדוגמה**, אם נגדיר את ה-BTB בגודל 16 כניסות, כלומר 2^4, אזי נוכל להשתמש בארבע הביטים [2:5] של ה-PC של פקודת סיעוף על מנת לבחור את הכניסה בטבלת ה-BTB בה נשים את אותה פקודה. באופן הזה, החזאי יצטרך לחפש פקודת סיעוף רק בכניסה מסוימת – פעולה יעילה יותר מאשר חיפוש בכל הטבלה. בכל כניסה יהיה TAG , כפי שנלמד בכתה, על מנת לזהות את הכתובת של הפקודה שנשמרה שם.

לשיטת הקצאה כזו של כניסות בטבלה נקרא direct mapping. מהגדרת אופן המיפוי של PC לכניסה בטבלה, ברור כי ייתכן ותהיה התנגשות בטבלה בין שתי פקודות סיעוף בעלות אותו ערך בביטים המשמשים למיפוי (ביטים 5-2 בדוגמה). במצב כזה, פעולת הכנסת פקודת סיעוף אחת תחליף את הרשומה של הפקודה השנייה שכרגע בטבלה באותה הכניסה. כלומר, בטבלה תיוותר רק הפקודה האחרונה שטופלה מבין השתיים שמתמפות לאותה הכניסה בטבלה.

שימו לב שבכל פעם שמוכנסת פקודת סיעוף חדשה בכניסה כלשהי ב-BTB - לא משנה אם לכניסה ריקה או במסגרת החלפה - נאפס את רגיסטר ההיסטוריה של אותה כניסה (במקרה של היסטוריה לוקלית) וכן נאתחל את טבלת מכונות המצבים המקושרת לכניסה למצב ברירת המחדל כפי שהופיע במאפייני החזאי בשורה הראשונה בקובץ הקלט (במקרה של טבלאות לוקליות, בלבד). מכיוון שבתרגיל זה הן איתחול והן עדכון הטבלאות נעשה בשלב ה-EXE, נשנה את ההיסטוריה ואת מכונת המצבים הרלוונטית בטבלה על בסיס ברירת המחדל מיד בארוע ה-update (כלומר לאחר update שמאתחל כניסה, המכונה הרלוונטית תהיה או במצב ברירת מחדל+1 או במצב ברירת מחדל -1).

בתרגיל זה אתם מתבקשים לממש את ההקצאות של הכניסות ב-BTB בשיטת direct mapping, בהתאם לגודל הטבלה בהגדרת התצורה, תוך שימוש עבור ה-TAG בביטים של ה-PC החל מהביט העוקב לזה שמוקצא עבור האינדקס ל-BTB. לדוגמה, נאמר כי גודל ה-BTB הוא 16 כניסות ול-TAG קיימים 10 ביט, אזי הכתובת תהיה מהצורה הבאה:

ZZZZZZZZZZ YYYY 00

כלומר, שני הביטים הראשונים הם 0 (הכתובת מיושרת ל-4), ארבעה ביטים לאינדקס של ה-BTB (ה-BTB בגודל 2^4) ועשרה ביטים עבור ה-TAG.

שימו-לב כי במקרה שה-TAG קטן מכמות הביטים הנדרשת לזיהוי מלא של כתובת הקפיצה (כמו בדוגמה לעיל), אזי ייתכן ושתי פקודות סיעוף *שונות* שתתמפנה לאותה הכניסה תהיינה זהות בערך ה-TAG שלהן. במקרה כזה, החזאי לא יוכל לדעת שמדובר בפקודות שונות ולא תתבצע החלפה ואיתחול הכניסה כנ"ל – כלומר, כל המרכיבים הלוקליים של הכניסה יישמרו ולא יאותחלו. כך שבמצב כזה, שתי פקודות סיעוף (או יותר) יכולות "להפריע" אחת לשניה, וההכרעות של כל אחת תשפענה על החיזוי של האחרות, שנמצאות באותה כניסה עם אותו ה-TAG (החלקי).

דוגמה לטבלת BTB בגודל 4 עם TAG בגודל 16 ביט:

BTB (size 4)		
tag	target	Local history (size 4)
0x0500	0xAA230	0000
0x0101	0xB340	1000
0x0010	0xC450	0110
0x020F	0xDD670	0101

למשל, פקודת ה-Branch בכתובת 0x108 שביטים [3:2] של הכתובת שלה הם 10 נכנסת למקום השלישי ב-BTB המתאים לביטים 10 עם הטאג 0x0010. במידה וכעת מגיעה פקודת Branch שכתובתה 0x128 היא נכנסת לאותו המקום ב-BTB (במקום ה-Branch בכתובת 0x108) עם הטאג 0x0012, ומאתחלת את ההיסטוריה הלוקלית (ואת טבלת המכונות המתאימה, אם לוקלית). אולם, אם ה-tag בגודל 1 ביט בלבד, לא נזהה שמדובר בפקודות שונות, ושתי הפקודות הללו יחלקו בו-זמנית את אותה הכניסה עם הטאג 0x0.

דרישות ההגשה

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

מועד ההגשה: עד ה-28/02/2024 בשעה 23:55.

אין לערוך שינוי באף אחד מקבצי העזר המסופקים לכם. ההגשה שלכם לא תכלול את אותם קבצים, מלבד המימוש שלכם ב-bp.c/cpp ותיבדק עם גרסה של סביבת הבדיקה של הבודק. עמידה בדרישות הממשק כפי שמתועדות בקובץ הממשק (bp_api.h) היא המחייבת.

עליכם להגיש קובץ tar.gz בשם hw1_*ID1_ID2*.tar.gz כאשר ID1 ו-ID2 הם מספרי ת.ז. של המגישים. לדוגמה: hw1_012345678_987654321.tar.gz. ה-tar יכיל קובץ **אחד**:

- קוד המקור של החזאי שלכם: bp.c או bp.cpp .
  קוד המקור חייב להכיל תיעוד פנימי במידה סבירה על מנת להבינו.

דוגמה ליצירת קובץ tar.gz:

```
tar czvf hw1_12345678_87654321.tar.gz bp.cpp
```

בדקו שהרצת פקודת הפתיחה של הקובץ אכן שולפת הקבצים שלכם:

```
tar xzvf hw1_12345678_87654321.tar.gz
```

דגשים להגשה:

1. המימוש שלכם – bp.c/cpp – **חייב** להתקמפל בהצלחה ולרוץ במכונה הוירטואלית שמסופקת לכם באתר המודל של הקורס. זוהי סביבת הבדיקה המחייבת לתרגילי הבית. **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 ב*מודל* לא עדכני ויש לנו גרסה עדכנית יותר שלא נקלטה."
3. יש להעלות את קובץ ההגשה דרך החשבון של אחד השותפים בלבד.

העתקות יטופלו בחומרה