ת"ב 4 – סימולטור Multithreading

בתרגיל בית זה תממשו סימולטור למעבד מרובה חוטים. הסימולטור יתמוך ב-(1) Blocked Multithreading (MT) ו-(2) Fine-grained MT.

המיקרו-ארכיטקטורה של המעבד

לשם פשטות, שתי המיקרו-ארכיטקטורות בבסיסן הן Single-Cycle, כלומר, פקודה לוקחת מחזור שעון יחיד בהינתן כי ה-opearndים שלה מוכנים וכי היא לא ניגשת לזיכרון. הארכיטקטורה תומכת בפקודות הבאות:

- פקודות גישה לזיכרון: LOAD, STORE
- פקודות אריתמטיות: ADD, ADDI, SUB, SUBI
- HALT - פקודה מיוחדת אשר תשמש לעצירת החוט שרץ כרגע (גם כן לוקחת מחזור יחיד).

בנוסף נתון:

- בארכיטקטורה הנתונה 8 רגיסטרים כלליים, R0 – R7.
- פקודות אריתמטיות יבוצעו בין רגיסטרים או בין רגיסטר למספר קבוע.
- פקודות גישה לזכרון יקחו מספר מחזורי שעון (מוגדר בקובץ הפרמרטרים).
- בכל מחזור שעון מתבצעת פקודה אחת ($IPC_{max} = 1$).
- מרחב הכתובות של כל החוטים זהה, כלומר, חוט 1 עשוי להשפיע על המידע של חוט 2. ניתן להניח שמרחב הכתובות הווירטואלי זהה לפיזי (איננו מתייחסים ל-Virtual Memory בתרגיל זה).
- עבור תצורה של Fine-grained MT:
  - החלפת ההקשר נעשת כל מחזור שעון אחד.
  - אין קנס (penalty) בהחלפת הקשר.
- עבור תצורה של Blocked MT
  - קיים קנס בהחלפת הקשר (מוגדר בקובץ הפרמרטרים).

הממשק לסימולטור

תיעוד הממשק לסימולטור נתון בקובץ core_api.h. עליכם לממש את הפונקציות המוגדרות בקובץ זה.

המימוש שלכם ייכתב בקובץ בשם core_api.c או core_api.cpp, למי שמעדיפים לממש ב-C++. שימו-לב שגם עבור מימוש ב-C++ עליכם לחשוף ממשק C, כפי שמוגדר בקובץ core_api.h.

גישה לזיכרון

אנו מספקים לכם סימולטור של מערכת הזיכרון. הממשק לסימולטור מוגדר בקובץ sim_api.h (הפונקציות ששמם מתחיל ב-SIM_Mem...) והמימוש שלו בקובץ sim_mem.c. הממשק לזיכרון מאפשר לסימולטור שלכם לקרוא פקודות ולקרוא/לכתוב נתונים. קריאת פקודה היא **מיידית ותיקח 0 מחזורי שעון**. אולם, קריאה וכתיבה של נתונים עשויה לקחת מספר מחזורי שעון. מספר מחזורי השעון עבור קריאה וכתיבה מצוין בקובץ ה-img המתאים. **ניתן להניח כי זמן הכתיבה קטן או שווה לזמן הקריאה.**

את תוכן הזיכרון ניתן לאתחל מקובץ מפת הזיכרון. קבצי דוגמה למפת הזיכרון לטעינה כלולים בחומרי התרגיל (הקבצים עם סיומת img) וכוללים גם תיעוד מבנה הקובץ בהערות. קובץ מפת הזיכרון מכיל הן פקודות לביצוע והן נתונים לקריאה/כתיבה. סימולטור הזיכרון מוגבל להכיל 100 פקודות עבור כל חוט ו-100 נתונים עוקבים.

הגישות לזיכרון נעשות בצורה סידרתית. לדוגמה, נאמר כי פקודת load לוקחת 100 מחזורי שעון, ופקודת store לוקחת מחזור שעון אחד. כמו כן, נאמר כי חוט 0 מבצע פקודת load במחזור שעון t, וחוט 1 מבצע פקודת store במחזור שעון $t+1$. במקרה זה, למרות שחוט 1 לכאורה מסיים את העבודה מול הזיכרון לפני חוט 0, חוט 0 יקרא את המידע מהזכרון כפי שהיה בזמן t.

חוטים

כל מעבד יריץ מספר חוטים כפי שנקבע בקובץ ה-img. ניתן לחלץ מספר זה מהפונקציה שמסופקת לכם ב sim_mem.c. אין מגבלה על כמות החוטים אשר המעבד יכול להריץ.

החלפת חוטים תהיה בשיטת round-robin. מתחילים מ 0 ועד החוט האחרון שרץ. אם חוט כלשהו סיים (הגיע ל halt) לפני אחרים אז מדלגים עליו. לדוגמה, אם חוט 1 סיים ונשארו במערכת החוטים 0 2 3, אז אחרי חוט 0 מגיע תורו של חוט 2.

שיטת ה-Round-Robin (RR) הנתונה אינה מדלגת על חוטים אשר ממתינים לקריאה או לכתיבה מהזיכרון. במילים אחרות, מנגנון ה-RR יבחר חוט אשר מחכה לזיכרון ולא ידלג עליו, כך שבפועל יתבזבז מחזור שעון. לדוגמה, נאמר כי במחזור שעון t לחוט 1 נותר מחזור שעון אחד להמתנה לזיכרון, וכמו כן מנגנון ה-RR בחר בו. מכיוון שבזמן t חוט 1 עדיין ב-idle, הוא לא יבצע עבודה. בזמן $t+1$, יבחר חוט מספר 2 (בהינתן והוא לא הגיע ל-halt) ו"במקביל" המידע לזיכרון אשר חוט 1 מחכה לו יהיה זמין עבורו. כך שפעם הבאה שמנגנון ה-RR יגיע אליו הוא יוכל לבצע עבודה.

חוט שמגיע לפקודת halt הוא חוט שסיים את עבודתו. **פקודת halt לוקחת מחזור שעון.**

הפונקציות שעליכם לממש:

- Core_blocked_Multithreading: הפונקציה תממש פעולת מכונה בתצורה של Blocked MT (לדוגמה, עדכון הקשר לכל תהליך).
- Core_finegrained_Multithreading: הפונקציה תממש פעולת מכונה בתצורה של Finegrained MT.
- Core_finegrained_CPI: מחזירה את ה CPI (cycles per instruction), עבור סוג finegrained.
- Core_blocked_CPI: מחזירה את ה CPI (cycles per instruction), עבור סוג blocked.
- Core_blocked_context: מחזירה את ההקשר של תהליך ספציפי עבור מכונה בתצורה של Blocked MT.
- Core_finegrained_context: מחזירה את ההקשר של תהליך ספציפי עבור מכונה בתצורה של Finegrained MT.

סביבת בדיקה

על מנת לבדוק את הסימולטור שלכם אנו מספקים קובץ sim_main.c שמאתחל את הסביבה של הסימולטור ומתפעל את הסימולטור שלכם למשך מספר מחזורי שעון. בסיומם הוא קורא את מצב הסימולטור ומדווח אותו לפלט הסטנדרטי. כמו-כן, מסופק לכם makefile על מנת לבנות את סביבת הבדיקה בשילוב המימוש שלכם, באופן דומה לבניה שיבצע הבודק. לאחר הבניה תקבלו קובץ ריצה בשם `sim_main`

את הקובץ המצורף ניתן להפעיל באופן הבא:

```
sim_main <test_file>
```

לדוגמא:

```
sim_main example1.img
```

שימו-לב: ה- `main` שניתן נועד להקל עליכם בבדיקה, אולם אתם מחויבים למימוש **הממשק** לסימולטור כפי שמוגדר ב-core_api.h . כלומר, ייתכן והסימולטור ייבדק בדרכים **שונות** מהמודגם ה-`main` וייתכן שימוש בקובץ `main` אחר מהמסופק, אשר משתמש באותו הממשק. לכן הקפידו שהמימוש שלכם יעמוד בדרישות המוגדרות בתרגיל.

דרישות ההגשה

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

מועד ההגשה: עד ה-20.06.2017 בשעה 23:55.

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

עליכם להגיש קובץ tar.gz* בשם hw4_*ID1_ID2*.tar.gz כאשר ID1 ו-ID2 הם מספרי ת.ז. של המגישים. לדוגמה: hw4_012345678_987654321.tar.gz. ה-tar יכיל שני קבצים:

- קוד המקור של הסימולטור שלכם: `core_api.c` **או** `core_api.cpp`.
  קוד המקור חייב להכיל תיעוד פנימי במידה סבירה על מנת להבינו.

*דוגמה ליצירת קובץ tar.gz:

```
tar czvf hw4_12345678_87654321.tar.gz core_api.cpp
```

בדקו שהרצת פקודת הפתיחה של הקובץ אכן שולפת הקבצים שלכם:

```
tar xzvf hw4_12345678_87654321.tar.gz
```

דגשים להגשה:

1. המימוש שלכם **חייב** להתקמפל בהצלחה ולרוץ במכונה הוירטואלית שמסופקת לכם באתר המודל של הקורס. זוהי סביבת הבדיקה המחייבת לתרגילי הבית. **כל בעיה בהגשה המונעת את הרצת הקוד (כיווץ לא נכון של הקבצים, קוד לא מתקמפל, ...) יגרור ציון 0!**
2. מניסיונם של סטודנטים אחרים: הקפידו לוודא שהקובץ שהעלתם לאתר הקורס הוא אכן הגרסה שהתכוונתם להגיש. לא יתקבלו הגשות נוספות לאחר מועד ההגשה בטענות כגון "הקובץ במודל לא עדכני ויש לנו גרסה עדכנית יותר שלא נקלטה."

בהצלחה!