ת"ב 1 – סימולטור לצינור מכונת in-order

בתרגיל בית זה תממשו סימולטור לצינור (pipeline) של מעבד in-order הדומה למעבד MIPS שהכרתם בתרגולים. הסימולטור שלכם יריץ קוד של תוכנית ויממש את כל שלבי הצינור עבור הפקודות הנקראות.

המיקרו-ארכיטקטורה של המעבד

המעבד שתממשו בסימולטור יישם מיקרו-ארכיטקטורה דומה למעבד ה-MIPS, כלומר, צינור 5 שלבים, כפי שראיתם בתרגול 2. לשם הפשטות, תתמכו רק בפקודות הבאות:

- פקודות גישה לזכרון: LOAD + STORE
- פקודות אריתמטיות (בין רגיסטרים): ADD + SUB
- פקודות סיעוף/קפיצה: BR + BREQ + BRNEQ

כמו ב-MIPS יהיו 32 רגיסטרים כלליים (GPR), כאשר GPR[0] תמיד מכיל 0. בנוסף, קיים רגיסטר PC שמכיל את כתובת הפקודה שנקראת בשלב ה-IF. כל הרגיסטרים בני 32 ביט.

המימוש יישם את עקרון ה-forwarding על מנת להימנע מ-stall כתוצאה מ-data hazards (מתי שאפשר, במגבלות רצף הפקודות והמיקרו-ארכיטקטורה). ה-forwarding יתאפשר הן משלב MEM והן משלב WB. כמו-כן, ה- Register File מתעדכן בחצי הראשון של מחזור השעון ונקרא בחצי השני – כלומר, הערך העדכני משלב ה-WB ניתן לקריאה באותו מחזור שעון משלב ה-ID, כפי שהודגם בכתה. שימו-לב, שלמרות מימוש forwarding, ייתכנו מצבים בהן לא ניתן להימנע מ-stall. עליכם לממש stall כאשר רצף הפקודות דורש זאת, בהתאם לזיהוי המצבים באמצעות יחידת ה-HDU שלכם.

פקודות קפיצה לא יגרמו ל-stall בגלל control-hazard על ידי שימוש בחיזוי קבוע - Always **not**-taken - כלומר, כל זמן שלא התקבלה תוצאת התנאי לקפיצה ייכנסו הפקודות העוקבות בתכנית (בזכרון) לצינור. במקרה של ביצוע קפיצה בפועל, הצינור ינוקה (flush) מכל הפקודות שלאחר פקודת הקפיצה (אך לא מאלה שלפניו בסדר הביצוע) ויחל לטעון פקודות מיעד הקפיצה. Branch resolution מתבצע בשלב ה-MEM, כך שהפקודה הנכונה נטענת בשלב IF במחזור **שלאחר** הגעת פקודת הקפיצה לשלב ה-MEM.

הגישות לזיכרון הראשי משלב ה-MEM תשתמשנה בסימולטור של הזיכרון שמסופק לכם. גישות לכתיבה תמיד תסתיימנה במחזור שעון אחד. כך גם קריאת פקודות. אולם גישות לקריאת **נתונים** עשויות להימשך מספר מחזורי שעון לא ידוע. בהתאמה, שלב ה-MEM צריך להתמודד עם עיכוב פקודה הקוראת מהזיכרון למשך יותר ממחזור אחד (ועיכוב הפקודות שאחריה בהתאם – אולם לא את הפקודות שלפניה, שכבר עברו את שלב ה-MEM).

הממשק לסימולטור

על מנת לתפעל את הסימולטור שלכם אתם תממשו עבור הסימולטור שלכם את הפונקציות המוגדרות בקובץ sim_api.h המתחילות ב-`SIM_Core...` . ראו תיעוד הממשק באותו קובץ.

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

על מנת לגשת לזיכרון, אנו מספקים לכם סימולטור של מערכת הזיכרון. הממשק לסימולטור מוגדר, גם כן, בקובץ sim_api.h (הפונקציות ששמם מתחיל ב-`SIM_Mem...`) והמימוש שלו בקובץ sim_mem.c. הממשק לזיכרון מאפשר למימוש הסימולטור שלכם לקרוא פקודות ולקרוא/לכתוב נתונים משלבי הצינור הרלוונטיים. קריאת פקודה תמיד מסתיימת באותו מחזור שעון, אולם קריאת נתון עשויה לקחת מספר מחזורי שעון. במקרה כזה הפונקציה לקריאה `SIM_MemDataRead` תחזיר קוד שגיאה במחזורי השעון שבהם המידע עדיין לא זמין, ויש לנסות לקרוא שוב במחזור השעון הבא.

את תוכן הזיכרון ניתן לאתחל מקובץ מפת זיכרון. קבצי דוגמה למפת זיכרון 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 עוקבות ו-100 נתונים עוקבים, לצורך פשטות המימוש,

אולם הם עשויים להיות בכל מקום במרחב הזכרון של 32 ביט. **שימו-לב**, הקריאה מהזיכרון תמיד תהיה בכתובות מיושרות ל-4 (מילים שלמות), כלומר, הפרש הכתובות בין כל מילת 32 ביט בזיכרון הוא 4.

סביבת בדיקה

על מנת לבדוק את הסימולטור שלכם אנו מספקים קובץ main (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[i] על מנת לבנות את סביבת הבדיקה בשילוב המימוש שלכם, באופן דומה לבניה שיבצע הבודק. לאחר הבניה תקבלו קובץ ריצה בשם sim_main. הריצו אותו ללא פרמטרים לקבלת הודעה עם אופן השימוש.

שימו-לב: ה-main שניתן נועד להקל עליכם בבדיקה, אולם אתם מחויבים למימוש הממשק לסימולטור כפי שמוגדר ב-sim_api.h. כלומר, ייתכן והסימולטור ייבדק בדרכים שונות מהמודגם ב-main. למשל, ניתן לקרוא את מצב הסימולטור ולאחר מכן להמשיך את ריצתו ולקרוא שוב לאחר מספר מחזורי שעון נוספים. כמו-כן, ייתכן והבדיקה תבצע `SIM_CoreReset()` יותר מפעם אחת במהלך בדיקה. לכן, הקפידו לעמוד בדרישות הממשק כפי שמתועדות בקובץ sim_api.h.

אין לערוך שינוי באף אחד מהקבצים המסופקים לכם. ההגשה שלכם לא תכלול את אותם קבצים (מלבד המימוש שלכם ב-sim_core.c/cpp) ולכן תיבדק עם גרסה של סביבת הבדיקה של הבודק. עמידה בדרישות הממשק כפי שמתועדות בקובץ הממשק (sim_api.h) היא המחייבת.

דרישות ההגשה

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

מועד ההגשה: עד יום ה' 14.04.2016 בשעה 23:55.

עליכם להגיש קובץ ZIP בשם hw1_*ID1_ID2*.zip כאשר ID1 ו-ID2 הם מספרי ת.ז. של המגישים. לדוגמה: hw1_012345678_987654321.zip . ה-ZIP יכיל שני קבצים:

- קוד המקור של הסימולטור שלכם: sim_core.c או sim_core.cpp .
  קוד המקור חייב להכיל תיעוד פנימי במידה סבירה על מנת להבינו.
- קובץ PDF הכולל תיעוד חיצוני של המימוש שלכם, כולל פרטים כגון מבני נתונים שנבחרו, אופן המימוש של שלבי הצינור, והטיפול באירועי בקרה, כגון, forwarding , עיכוב זיכרון וכו'.

דגשים להגשה:

1. המימוש שלכם – sim_core.c/cpp – **חייב** להתקמפל בהצלחה ולרוץ בסביבת השרת הטכניוני T2 (או TX).
2. שימו-לב לסנקציות במקרה איחור כמפורט באתר הקורס. מאוד לא מומלץ לאחר את מועד ההגשה שהוגדר לתרגיל. בנסיבות מיוחדות יש לקבל אישור **מראש** מהמתרגל האחראי לאיחור.
3. 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".

בהצלחה!

[i] במידה ואינכם מכירים makefile תוכלו בקלות להבין את השימוש בו באמצעות מדריכים באתרים כמו:

http://mrbook.org/blog/tutorials/make/