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 תכנית 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
- פקודה מיוחדת HALT שתשמש לעצירת המעבד וסיום התכנית.

כמו ב-MIPS, יהיו 32 רגיסטרים כלליים - GPR, כאשר רגיסטר r0 תמיד מכיל 0 (כתיבה אליו לא משנה את ערכו).
בנוסף, קיים רגיסטר PC שמכיל את כתובת הפקודה שנקראת בשלב ה-IF . כל הרגיסטרים בני 32 ביט.

פעמים רבות בהערכת ביצועים של מעבדים, נרצה לבדוק עד כמה שיפור או שינוי מסוים משפיע על הביצועים. לשם כך נאפשר הפעלה או כיבוי של "מתגים" שיפעילו פונקציונליות נוספת של המיקרו-ארכיטקטורה.

המימוש שלכם יצטרך לאפשר להשוות בין pipeline מ-3 תצורות.

בבסיס, המימוש הנאיבי של טיפול ב data-hazards באמצעות STALL. כלומר, הזרקה של "בועות" (NOP) לצינור כנדרש על מנת לפתור את ה-hazard.

הפעלת מתג ה-*split-regfile* תאפשר את חציית מחזור השעון של ה-register file לשתי פאזות – כתיבה בחצי מחזור שעון הראשון, ואח"כ קריאה בחצי המחזור השני. כלומר, הערך העדכני משלב ה-WB ניתן לקריאה באותו מחזור שעון משלב ה-ID, כפי שהודגם בכתה.

הפעלת מתג ה-*forwarding* יישם את עקרון ה-forwarding הן משלב MEM והן משלב WB, על מנת להימנע מ-STALL כתוצאה מ-data-hazard היכן שאפשר. במצב זה יופעל גם מנגנון חציית מחזור השעון של ה-register-file לשתי פאזות, כנ"ל. שימו-לב, שלמרות מימוש forwarding ,ייתכנו מצבים בהם 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 (בזיכרון) 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 בשלב MEM תמיד תסתיימנה במחזור שעון אחד. כך גם קריאת פקודות בשלב IF. אולם גישות לקריאת **נתונים** בשלב MEM 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 וממשיכות להתקדם בצינור).

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 כל אחד מרצפי הנתונים הללו עשוי להיות ממוקם בכל כתובת במרחב הזיכרון 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.

פונקציית SIM_CoreGetState קוראת את מצב המכונה בכל שלב בצינור ב**סופו** של מחזור השעון (כלומר, המידע שיידגם ב-clock tick הבא ל-latch שחוצץ לשלב הבא).

לאחר ביצוע RESET באמצעות הפונקציה SIM_CoreReset הניחו שעובר מספיק זמן כך שבמוצא ה IF נמצאת הפקודה שמופיעה בכתובת 0 של הזיכרון. היעזרו בדוגמאות המצורפות על מנת לראות את מצב המכונה המצופה.

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> <max_num_of_cycles> [-s|-f]
```

לדוגמא:

```
sim_main example1.img 80
sim_main example1.img 80 -s
sim_main example1.img 80 -f
```

דגל `s-` יפעיל את אופציית *split-regfile* , ודגל `f-` יפעיל את אופציית ה-*forwarding* ביחד עם אופציית *split-regfile*.
אין צורך לתמוך ב *forwarding* ללא *split-regfile*.

הפעלת הדגלים תשמור ערך אמת בוליאני למשתנים גלובליים בשמות האופציות, כמוגדר בקובץ sim_api.h. תוכלו להשתמש בערכי משתנים אלו כדי להתאים את התנהגות המימוש שלכם במהלך הריצה בהתאם לתצורות השונות של המיקרו-ארכיטקטורה.

עצירת התכנית יכולה להתבצע בשני מקרים. או שמספר הפקודות מגיע ל-`max_num_of_cycles` המסופק בשורת הפקודה, או כאשר פקודת HALT מגיעה לשלב ה-WB - הראשון מבינהם (ראו דוגמאות).

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 ה-`main` וייתכן שימוש בקובץ `main` אחר מהמסופק, אשר משתמש באותו הממשק. לכן הקפידו שהמימוש שלכם יעמוד בדרישות המוגדרות בתיעוד הממשק מעבר לתפקודו עם ה-`main` הנתון.

דרישות ההגשה

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

מועד ההגשה: עד ה' 19.04.2017 בשעה 23:55.

אין לערוך שינוי באף אחד מקבצי העזר המסופקים לכם. ההגשה שלכם לא תכלול את אותם קבצים, מלבד המימוש שלכם ב-`sim_core.c/cpp` ותיבדק עם גרסה של סביבת הבדיקה של הבודק. עמידה בדרישות הממשק כפי שמתועדות בקובץ הממשק (sim_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 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.
  - סיכום קצר על השיפור בביצועים שזיהיתם בעקבות הפעלת המתגים השונים. בפרט, רישמו את זמני הביצוע עבור כל אחת מהדוגמאות שניתנה לכם ואת הממוצע שיפור (speedup) עבור כלל הטסטים שהרצתם (כולל כאלה שאתם כתבתם) בכל אחד מהאופציות (split-regfile, forwarding), לעומת המצב הבסיסי שעובד רק עם STALL.

*דוגמה ליצירת קובץ tar.gz:

```
tar czvf hw1_12345678_87654321.tar.gz sim_core.cpp hw1_summary.pdf
```

בדקו שהרצת פקודת הפתיחה של הקובץ אכן שולפת הקבצים שלכם:

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

דגשים להגשה:

1. המימוש שלכם – sim_core.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. 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!