תיעוד לפרויקט scoreboard

מוגש על ידי:

כפיר גולדברג 206335036

מתן קוריאט 312278245

# design

בפרק הזה נסקור את התכן שביצענו למערכת משני כיוונים –כיוון החומרה וכיוון התוכנה.

נתחיל מסקירת רכיבי החומרה:

1. זיכרון (RAM) – אחסון פקודות התוכנית ומידע לפקודות ST/LD, 4096 שורות ברוחב 32 ביט
2. מנוע FP (FP Engine) – ביצוע פקודות FP אריתמטיות, בעל מספר משתנה של יחידות ביצוע לכל סוג לפי קובץ הקונפיגורציה
3. מנוע זיכרון (Memory Engine) – ביצוע הגישות לזיכרון, בעל מספר משתנה של יחידות ביצוע לכל סוג לפי קובץ הקונפיגורציה
4. קובץ רגיסטרים (Register File) – 16 הרגיסטרים של המעבד
5. תור הוראות (Instruction Queue) – תור עם מקום ל-16 פקודות הממתינות לביצוע Issue ליחידה פונקציונלית, כאשר הissue מתבצע בסדר FIFO ביחס לFetch שקורה סדרתית לפי תוכן הזיכרון עד לפקודת Halt.

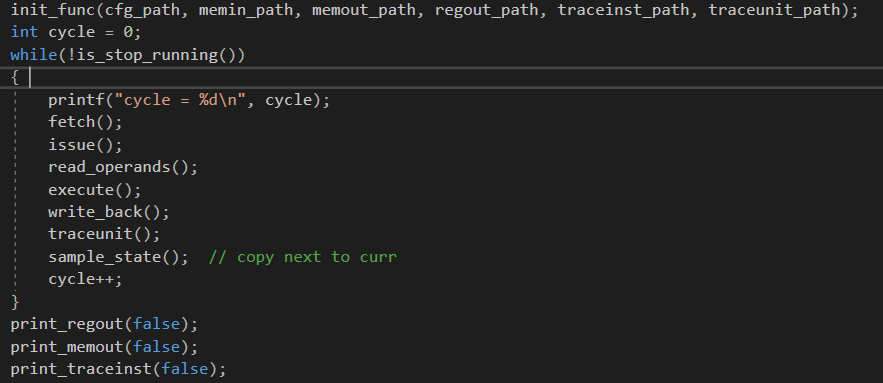
מכיוון שאנו מסמלצים חומרה, אנו צריכים לקחת בחשבון עניינים הנובעים מהאופי המקבילי של חומרה – כלומר כל יחידות הביצוע פועלות במקביל אחת לשנייה אע"פ שהתוכנה שלנו כתובה באופן סדרתי. דוגמה למצב כזה היא ביצוע fetch וissue. בחומרה, כשנבצע fetch לפקודה היא תופיע בinstruction queue רק במחזור השעון הבא, אך בתוכנה אם נבצע את הפקודות באופן סדרתי נוכל לקבל מצב בו בפונקציית הfetch נוסיף פקודה לתור ההוראות ולאחר מכן בפקודת הissue מיד נקרא אותה – וזו טעות.

על כן, רצינו לדמות את המצב בחומרה בו לכל רגיסטר יש כניסת D שקובעת מה יהיה ערכו במחזור הבא ויציאת Q שהיא המידע הזמין ממנו כרגע. לצורך כך שכפלנו כל יחידה פונקציונלית, רגיסטר מקובץ הרגיסטרים, ואת תור ההוראות, ככה שמההעתק עם הסיומת \_curr נקרא תמיד ומההעתק עם הסיומת \_next תמיד נכתוב. בסוף מחזור השעון, נחקה את פעולת הדגימה של רגיסטר – כלומר נעתיק את \_next לתוך \_curr, וכך נקבל סימלוץ פשוט של פעולת החומרה.

כעת נדון במימוש הקוד שלנו. נכתוב כאן פסאודו-קוד פשוט שמתאר את הלוגיקה העילית שרצינו למממש ולאחר מכן נדון בפרטים:

1. אתחל את כל החומרה הנדרשת בהתאם לקובץ הקונפיגורציה
2. כל עוד ישנן פקודות שעדיין לא סיימו את הביצוע:
   1. הבא את הפקודה הבאה
   2. בצע issue לפקודה הבאה
   3. קרא את הoperands לכל הFUs המוכנות
   4. קדם את הביצוע של כל הFUs המוכנות
   5. בצע writeback מכל הFUs המוכנות
   6. עדכן את מבני הנתונים הרלוונטיים להדפסת הפלט
   7. דגום את כל יחידות החומרה
3. הדפס את קבצי הפלט

כל אחד מהשלבים המצוינים כאן מופיע כפונקציה אחרת בmain שלנו:



כעת נדון בכמה בחירות נושאים ארכיטקטוניים ובדרך ההתמודדות שלנו איתם, ומעבר לכך ניתן למצוא תיעוד בקוד לנושאים הנוגעים במימוש.

1. התמודדות עם גישות רצופות לאותה כתובת בזיכרון – במקרה בו פקודה מסוימת ניגשת לכתובת בזיכרון בזמן שפקודה קודמת הניגשת לאותה הכתובת טרם סיימה את הביצוע, עלולה להיווצר בעיה. נבדיל בין שני מקרים:
   1. LD-LD / ST-ST – אם שתי הפקודות הן מאותו הסוג, אין בעיה – אנו יודעים שמשך ההשהיה של יחידות הביצוע זהה ולכן הן יסתיימו בסדר בו התחילו את הביצוע שהוא הסדר בתוכנית, ולכן אין כאן בעיה
   2. LD-ST / ST-LD – במצב כזה אם הפקודה השנייה שהוקצתה אמורה להסתיים לפני הראשונה, עלולה להיווצר בעיית RAW או WAR. אנו פתרנו את הבעיה על ידי חסימת הפקודה המאוחרת עד שהפקודה המוקדמת תסיים את הביצוע.
2. סיום התוכנית – כאשר נקראת פקודת halt מהזיכרון אנו יודעים כי אין פקודות נוספות לביצוע בתוכנית, אך עדיין צריך לסיים את ביצוע כל הפקודות הקודמות. הדרך שלנו לדעת מתי לסיים את התוכנית היא להדליק דגל כאשר נקראת פקודת halt. כאשר הדגל דולק, נבחן בתוך מבנה נתונים שהגדרנו האם יש פקודות שעדיין לא ביצעו write\_back, וכל עוד יש כאלה – נמשיך בביצוע. ברגע שכל הפקודות יסיימו את שלב הwrite\_back, נוכל לסיים את ריצת התוכנית.
3. ניהול תור ההוראות – כפי שהוגדר, ישנו תור הוראות באורך 16. אנו בחרנו למממש את התור באמצעות מערך עם שני מצביעים – אחד לכתיבה ואחד לקריאה, ובנוסף מונה של מספר המקומות הפנויים. בכל מחזור נכתוב למקום המוצבע על ידי מצביע הכתיבה ונקרא לפי מצביע הקריאה. במידה והמצביעים שווים, יש לבדוק האם מספר המקומות הפנויים הוא 16 – כלומר התור ריק, ואז ניתן לכתוב לתור אך לא לקרוא ממנו, או שמספר המוקמות הפנויים הוא 0 – כלומר התור מלא, ואז ניתן לקרוא מהתור אך לא לכתוב אליו.

# Testplan

הגדרנו מספר טסטים לתוכנית, אשר באו לבחון מצבים שאנו זיהינו בהם פוטנציאל לבעיות (ואכן מצאנו מספר לא מבוטל של באגים בעזרתם).

בטבלה הבאה נפרט את הטסטים שחשבנו עליהם, איזה מקרה הם באים לבחון, מה התרחיש שיקרה בהם בפועל ומה הקונפיגורציה הדרושה לכך. יש כאן יותר מ3 טסטים, מכיוון שבאמת השתמשנו בכל הטסטים האלו חשבנו שכדאי לציין אותם

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר | שם | תרחיש | קונפיגורציה | קריטריון מעבר |
| 1. | Store\_load\_store\_load | כתיבות וקריאות רצופות מאותה הכתובת | יחידת ST אחת עם השהייה ארוכה והרבה יחידות LD בעלות השהייה קצרה | 1. נכונות המידע בזיכרון וברגיסטרים 2. פקודות נכנסות לביצוע רק לאחר שהקודמת סיימה |
| 2. | Arithmetic | ביצוע פקודות אריתמטיות שונות | הרבה יחידות ביצוע | בחינת התוצאה של החישובים המתמטיים על מנת לוודא כי הייצוג שלנו לFP עובד כראוי |
| 3. | Structural\_hazard | ביצוע 4 פקודות ADD ברצף | 3 יחידות ADD בעלות השהייה ארוכה | פקודת הADD הרביעית נכנסת לביצוע רק לאחר שהראשונה סיימה |
| 4. | Wb\_stall | דוגמא מההרצאה – פקודת DIV משתמשת ברגיסטר F6 ומחכה לביצוע בעוד שפקודת ADD שאחריה אמורה לכתוב לF6 ומחכה בשלב הWB | כמו בהרצאה | הפקודה האחרונה מחכה בשלב הWB עד שפקודת הDIV שלפניה תסיים לקרוא את האופרנדים |
| 5. | Waw\_arithemtic | כתיבה רצופה של פקודות אריתמטיות לאותו הרגיסטר | מספר יחידות ביצוע, והשהיות כך שהפקודה השנייה תסתיים לפני הראשונה | הרגיסטר מכיל את הערך מהפקודה השנייה בתום הביצוע |
| 6. | Performance stress | המון פקודות מאותו סוג (ADD) | השהייה של מחזור שעון יחיד | וידוא כי אין השהיות מיותרות |