**דוקומנטציה לסימולטור**

הסימולטור מסמלץ 4 ליבות העובדות במקביל. בכל ליבה ישנה צנרת בת 5 שלבים: fetch, decode, execute, memory ו-write back ופקודות אסמבלי עוברות דרכה. שלב ה-memory ממומש לפי פרוטוקול MSI. לכל ליבה ישנה מטמון עם מדיניות כתיבה allocate write, back write משלה וגישה לשאר הליבות ולזכרון הראשי דרך bus משותף.

טיפוסים עיקריים:

 **:Bus** זהו bus המאפשר לליבות לשתף ביניהן מידע ולהעביר ולקבל מידע גם מהזכרון הראשי. ה-bus מכיל בתוכו את השדות הבאים המדמים חוטים. הוגדרו שני bus-ים. האחד המתעדכן במהלך המחזור ואחד שמעודכן עם סיום המחזור. זאת על מנת לדמות פעולה תחת Flip Flop

origid- מי שיוצר את הטראנסקציה וכותב לקווי הבס במחזור השעון הנוכחי (ליבה או הזיכרון הראשי(.

cmd- הפעולה שעתה מתבצעת על ה-bus.

address- הכתובת שליבה רוצה לטעון או לכתוב לתוכה

data- המידע שליבה רוצה לטעון מהכתובת או לכתוב לתוך אותה כתובה.

 **:tsram** מערך בן 4 מקומות, לכל ליבה, המדמה את החלק שמחזיק את מרחב הכתובות ששמורות במטמון של כל ליבה. מכיל 256 שורות, כאשר כל שורה מכילה עבור כל בלוק במטמון את השדות:

.Modified – 2 ,Shared – 1 ,Invalid – 0 :MSI המתאר את מצב הבלוק בהתאם לפרוטוקול הקוהרנטיות MSI.

Tag: ה-tag של הכתובת שהמטמון מחזיק.

 **dsram**: מדמה זכרון SRAM שהינו ברוחב 32 ביטים ובעומק 256 מילים, ומכיל את הדאטא השמור במטמון. הוגדר מערך בגודל 4 מסוג טיפוס זה, בעבור כל ליבה.

 **mem\_stage**: על מנת לממש את פרוטוקול הקוהרנטיות MSI הוגדר מערך בגודל 4, לכל ליבה, כך שבכל ליבה מצוין במערך זה האם היא ניגשת במחזור הנוכחי למטמון, ממתינה לגישה ל-bus או שמחזיקה באופן בלעדי את הגישה ל-bus.

 **watch**: טיפוס זה נועד למימוש פקודות store conditional(SC) ו-load\_conditional(LL). הוגדרו 4 כאלה, אחד בעבור כל ליבה. שדה אחד במבנה זה הוא sc\_dirty . כאשר ליבה מצליחה לבצע את פעולת SC אזי שדה זה מקבל את הערך 1 בשביל לאותת לשאר הליבות שמתכוונות לבצע SC לכתובת זהה שהן יכשלו לעשות זאת. שדה שני בטיפוס זה הוא address והוא מחזיק את הכתובת ממנה הליבה מבצעת LL ולהיכן היא מעוניינת לבצע SC.

**פונקציות**:

  **:** **mem** מקבלת את מספר הליבה שמגיעה במחזור הנוכחי לשלב ה-memory בצנרת. פונקציה זו אחראית לממש את פרוטוקול הקוהרנטיות MSI ולבצע את כל פקודות הטעינה והכתיבה לזכרון שיש בקוד ה-assembly.

 **search\_address\_cache:** מקבלת את הטיפוס bus ובודקת האם ישנו בלוק בליבה אחרת במצב modified המכיל את הכתובת המבוקשת. אם ישנו בלוק כזה הפונקציה תחזיר את מספר הליבה שבו קיים הבלוק.

:**free\_access\_bus ** מקבלת את מספר הליבה שמגיעה במחזור הנוכחי לשלב ה-memory בצנרת. הפונק' בודקת האם אין ליבה אחרת שמחזיקה את הגישה ל-bus. אם ליבה כזאת הפונקציה תחזיר אמת ואם יש אז יוחזר שקר.

** :sc\_func** מקבלת כתובת אליה הליבה שנמצאת בשלב memory ופקודת store conditional, ומספר הליבה. הפונקציה מחזירה האם פקודת SC תצליח או תכשל.

** write\_output**: מקבלת כקלט את שם קובץ המוצא. עוברת על המערכים הגלובליים שעדכנו

בפונקציות הקודמות ומסדרת אותם במערך עזר output לפי הסדר והפורמט בו הם צריכים

** advance\_bus :** מסמלצת את התפקוד של המעבד תחת flip flop. בסיום כל מחזור updated bus מעביר את כל המידע בשדות שלו ל-current bus.

**דוקומנטציה לטסטים באסמבלי**

**mulserial:** בטסט זה מימשנו כפל מטריצות תוך שימוש בליבה אחת בלבד. במטריצה אחת רצנו על האיברים שורה אחרי שורה כך שרק אחרי עדכון שורה שלמה במטריצת התוצאה עברנו לשורה הבאה. במטריצה השניה בצענו ריצה עמודה אחרי עמודה.

**mulparallel:** בטסט זה מימשנו כפל מטריצות תוך שימוש בארבעת הליבות תוך בדיקה של מימוש פרוטוקול הקוהרנטיות MSI. בכל ליבה בצענו ריצה על שורות אחרות עוקבות במטריצה אחת עליה אנחנו עוברים שורה שורה ומעדכנים בלוקים שונים במטריצת התוצאה בכל ליבה (ארבע שורות עוקבות). הריצה על המטריצה השניה עליה רצים עמודה עמודה, נשארת זהה לטסט הקודם (ריצה על כל העמודות של המטריצה סה"כ בעדכון של כל שורה במטריצת התוצאה). בליבה אחת מתבצעת ריצה על 4 השורות הראשונות במטריצה אחת ועדכון של ארבע השורות הראשונות במטריצת התוצאה. באותו האופן עוברים לקרוא ולכתוב מארבעת השורות הבאות במטריצה אחת ובמטריצת התוצאה בהתאמה. לבסוף, בליבה 3 נעבור על 4 השורות האחרונות במטריצה אחת ונעדכן את 4 השורות האחרונות במטריצת התוצאה.