תיעוד – פרויקט מבנה המחשב

אסף וינברג 315398975 קארינה פלקסר 206586679 גיא פז בן יצחק 315328963

הקדמה

בפרויקט זה מימשנו מעבד RSIC בשם SIMP. הפרויקט חולק ל-3 חלקים מרכזיים:

- אסמבלר – נכתב בשפת C
- סימולטור – נכתב בשפת C
- תוכניות אסמבלי – שנכתבו כמובן בשפת אסמבלי בפורמט SIMP (הוראות ברוחב 48 ביטים)

להלן פירוט על אופן כתיבת התוכניות:

אסמבלר

במסמך האסמבלר שמרנו מס' משתנים כלליים, ובהם טבלת אופ-קודים בה שמורים כל שמות האופקודים (כלומר הפקודות במילים), טבלת רגיסטרים המכילה את שמות כל הרגיסטרים, משתנה השומר את מס' הלייבלים השונים ומשתנה השומר את הכתובת הגדולה ביותר אותה עלינו לשמור בזיכרון. בנוסף, שמרנו מערך השומר את המידע שעלינו להכניס לכל כתובת בזיכרון (דבר הניתן לנו בתוכנית הקלט באסמבלי).

כמו כן אתחלנו מבנה נתונים ששומר את הלייבלים של תוכנית הקלט באסמבלי. כל רשומה שומרת את הלייבל ואת הכתובת שלו (כלומר ה-PC).

לאחר וידוא נכונות ארגומנטים, מהלך תוכנית האסמבלר כולל שני מעברים על תוכנית האסמבלי:

1. מעבר ראשון (בפונקציה PassOne): בפונקציה זאת אנו קוראים שורה אחר שורה מתוכנית הקלט באסמבלי. כל שורה אנו שומרים במערך char. באמצעות פונקציות עזר אנו מורידים את ההערות, מתעלמים מפקודות הקשורות להכנסה זיכרון (נכנה אותן פקודות DATA, כלומר פקודות word.) ומוחקים רווחים מיותרים.
   על שורות ריקות אנו מדלגים.
   בעזרת פונקציית משנה אנו בודקים האם מדובר בלייבל בלבד, בפקודה בלבד או בפקודה עם לייבל. אם מדובר בפקודה אז נקדם את ה-PC באחד, ואם מדובר בלייבל נוסיף אותו לרשומות הלייבלים.

2. מעבר שני (בפונקציה PassTwo): בפונקציה זאת שוב אנו קוראים שורה אחר שורה מתוכנית הקלט באסמבלי. כל שורה אנו שומרים במערך char, ובאמצעות פונקציות עזר אנו מורידים את ההערות ומוחקים רווחים מיותרים.
   על שורות ריקות אנו מדלגים.
   לאחר מכן אנו בודקים באמצעות פונקציית משנה האם מדובר בפקודת Data (כפי שהזכרנו קודם, כוללת word.). אם כן אנו מטפלים בה ע"י שמירת הערך המתאים באינדקס המתאים שנכתב לנו במערך השומר את המידע שעלינו להכניס לכל כתובת בזיכרון אותו הזכרנו למעלה.

כמו כן אנו שומרים את הכתובת הגדולה ביותר אותה עלינו לשמור בזיכרון תחת המשתנה הרלוונטי. לבסוף אנו עוברים לקרוא את השורה הבאה.
אם לא מדובר בפקודה מסוג זה, בעזרת פונקציית משנה אנו בודקים האם מדובר בלייבל בלבד, בפקודה בלבד או בפקודה עם לייבל.
אם מדובר בפקודה, נקודד את הפקודה לביטוי בביטים בעזרת פונקציית משנה לפי סט ההוראות וקידודן. כל שורה כזאת נדפיס בקידוד לביטים בפורמט הקסאדצימלי לקובץ הפלט הראשון.
אם לא מדובר בפקודה לא נעשה דבר.
לאחר מכן נדפיס לקובץ הפלט השני את כל תוכן הזיכרון ששמרנו במערך השומר את המידע שעלינו להכניס לכל כתובת בזיכרון, בפורמט הקסאדצימלי. ניעזר במשתנה ששומר את הכתובת הגדולה ביותר אותה עלינו לשמור בזיכרון ששונה מאפס, וכך לא נדפיס ערכים שהם אפס לאחר הערך האחרון שאינו אפס.

בכך מסתיימת פעולת האסמבלר.

סימולטור

את הסימולטור חילקנו למספר מודולים, בכל אחד מהם ישנו מימוש של פונקציות קשורות:

- clockModule: מודול שאחראי על אתחול, תחזוק וכן פעולות קריאה וכתיבה של רגיסטרי הטיימר והשעון וכן שמירה של מספר מחזורי השעון הכולל שעוברים. כמו כן עוזר לתחזק את פסיקה irq0 ע"י משתנה.

- diskModule: מודול שאחראי על הדיסק הקשיח. מודול זה כולל אתחול של זכרון הדיסק הקשיח ע"י קריאה מקובץ קלט, פעולות קריאה וכתיבה מסקטורים וכן מתחזק מספר רגיסטרי חומרה הקשורים לדיסק הקשיח. כמו כן מודול זה אחראי על הוצאת קובץ פלט המכיל את תוכן הדיסק הקשיח בסיום הריצה. מודול זה עוזר לתחזק את פסיקה irq1 ע"י משתנה.

- displayModule: מודול שאחראי על תצוגת ה-7-segment display. המודול אחראי על כתיבת קובץ הפלט המכיל 32 ביטים שמייצגים 8 אותיות. למודול פעולות קריאה וכתיבה של ערך ה-display. כאשר מתחילה ריצת התוכנית, המודול פותח את קובץ הפלט לכתיבה, וכאשר היא מסתיימת הוא סוגר אותו.

- instructionModule: מודול זה אחראי על ביצוע ההוראות של התוכנית. המודול שומר מערך של שמות רגיסטרי החומרה.
המודול מבצע הוראות באמצעות הפונקציה executeInstruction. פונקציה זאת משתמשת תכני הרגיסטרים השמורים, ומבצעת חלוקה של סוגי הוראות: הוראות אריתמטיות, הוראות Branch והוראות נוספות (כגון שמירה בזיכרון). כל הוראה מבוצעת בעזרת פונקציית משנה מתאימה, והיא נשלחת לפונקציה זאת ע"י מנגנון case-switch. כמו כן כל הוראה שומרת במקום המתאים ברגיסטרים/זיכרון/דיסק קשיח וכו' את המידע הרלוונטי. הפונקציה executeInstruction מקבלת גם מצביע לערך ה-pc כך שתוכל לעדכן אותו בהתאם לסוג ההוראה.
המודול תומך גם בביצוע פעולות קריאה וכתיבה לרגיסטרי החומרה, שנשלחות באמצעות פונקציות למודולים הרלוונטיים. הוא גם מבצע את פעולות הכתיבה לקובץ הפלט המתעד כל קריאה או כתיבה לרגיסטר חומרה.

- interuptsModule: מודול האחראי על תחזוק הפסיקות. תחילה מעודכנים כל משתני הפסיקות לערך 0. המודול שומר מערך של מחזורי שעון בהם יש להדליק את פסיקה irq2status שמתקבלים בקובץ קלט, וכן כולל פעולות קריאה וכתיבה לרגיסטרי החומרה של הפסיקות (כתיבה לכולם מלבד של רגיסטרי הסטטוס שמתוחזקים בעזרת מודול הדיסק הקשיח ומודול השעון ובאמצעות מודול זה).
מעבר לכך, המודול כולל פונקציה בשם checkInterruption, אשר נקראת ע"י הסימולטור בלולאה הראשית שלו (פירוט על כך בהמשך) ובודקת האם ישנה פסיקה (פסיקות 0,1 באמצעות המודולים האחרים, פסיקה 2 בטיפול פנימי של המודול הזה). כמו כן הפונקציה תומכת בטיפול בפסיקה שקורית בזמן טיפול בפסיקה אחרת.

- ledsModule: מודול שאחראי על תחזוק נורות הלד. למודול פעולות קריאה וכתיבה של ערך ה-LEDS שנכתב לרגיסטר החומרה הרלוונטי. כאשר מתחילה ריצת התוכנית, המודול פותח את קובץ הפלט לכתיבה, וכאשר היא מסתיימת הוא סוגר אותו.

- memoryModule: מודול האחראי על זיכרון הנתונים, באמצעות שמירת מערך בגודל הנחוץ. הזיכרון מאותחל לאחר קריאה מקובץ קלט והכנסתו למקומות הרלוונטיים במערך. כמו כן ישנן פונקציות המאפשרות קריאה וכתיבה לזיכרון (למערך זה), וכן לאחר פונקציה שמופעלת בסיום ריצת התוכנית וכותבת לקובץ פלט את זיכרון הנתונים, עד לשורה האחרונה בזיכרון ששונה מאפס (אנו יודעים זאת באמצעות פונקציית מעבר על אינדקסים).

- monitorModule: מודול האחראי על תחזוק המוניטור. במודול זה אנו שומרים ע"י משתנים את רגיסטרי החומרה הרלוונטיים ומאפשרים פעולות קריאה וכתיבה אליהם. אנו מאפשרים גם פעולת כתיבה של פיקסל, ואת הפיקסלים של המוניטור שומרים באמצעות מערך דו-מימדי. בסיום הריצה אנו כותבים את תוכן הפיקסלים לקבצי הפלט הרלוונטיים.

נציין כי מלבד קבצים אלה, הוספנו גם קבצי header לקישור בין כל הקבצים.

הקובץ האחרון אותו כתבנו הוא קובץ הסימולטור המתכלל בין כל המודולים השונים ומנהל את הריצה.

ראשית אנו מריצים אתחולים של כל המודולים השונים, ובפרט גם אתחול של הוראות אותן אנו שומרים ברשימה מקושרת וקוראים מקובץ הקלט של הסימולטור.

לאחר האתחול, אנו מנהלים את הריצה ע"י לולאה שרצה ללא הגבלה, עד שתנאי סיום מגדיר אותה, והוא שקראנו פקודת halt (החזרה של ערך false באחת מפונקציות המודולים של קריאת ההוראות).

בכל איטרציה של הלולאה אנו מעדכנים את הדיסק הקשיח באמצעות פונקציה במודול שלו האם לבצע פקודת קריאה/כתיבה, וזאת באמצעות בדיקת השעון הנוכחי.

לאחר מכן אנו בודקים האם הייתה פסיקה באמצעות המודול המתאים עם הפונקציה checkInterruption. אם כן נשמור את ערך ה-PC המתאים ונסמן באמצעות משתנה שאנו מטפלים כרגע בפסיקה. לאחר מכן אנו קוראים לפונקציה במודול ההוראות שמסדרת את רגיסטרי ה-imm וכותבים את תוכן הרגיסטרים הנוכחי בתוספת דברים נוספים לקובץ הקלט הרלוונטי.

לאחר מכן אנו מבצעים את ההוראה הנוכחית ברשימת ההוראות ע"י קריאה לפונקציה במודול ההוראות. אם הפונקציה מחזירה ערך true נמשיך בריצה, נעדכן את השעון הנוכחי ונעבור לאיטרציה הבאה. אם הפונקציה מחזירה ערך false נעצור את הריצה, נפעיל את הפונקציה exitSimulator שפונה לכל הפונקציות הרלוונטיות במודולים. פונקציות אלה ישחררו זיכרון, יכתבו לקבצי פלט. בנוסף נסגור קבצים שנשארו פתוחים.

קבצי האסמבלי

- mulmat: תוכנית המחשבת את מטריצת התוצאה של כפל 2 מטריצות בגודל 4X4.
בתוכנית זאת השתמשנו במחסנית ע"מ לשמור 3 אינדקסים: i (אינדקס שורה), j (אינדקס עמודה) ו-k (אינדקס רץ). כמו כן שמרנו במחסנית את הערך של תוצאת הכפל המצטברת עבור כל איבר במטריצת התוצאה. ברגיסטרים s0,s1,s2 השתמשנו עבור מטריצות הקלט והתוצאה (בהתאמה). ברגיסטרים t0,t1,t2 השתמשנו עבור חישובים זמניים.
את הכפל ביצענו לפי הנוסחה: $c[i][j] = \sum_{k=0}^{3} a[i][k] \cdot b[k][j]$ לחישוב כפל מטריצות, ונענו בין חישובים לפי הזזת האינדקסים בהתאם למקום בו אנו נמצאים בתוכנית.
- binom: תוכנית המחשבת את מקדם הבינום של ניוטון באופן רקורסיבי. בתוכנית זאת אנו משתמשים במחסנית, כאשר בכל קריאה רקורסיבית אנו שומרים את רגיסטר ההחזרה, ואת רגיסטרי s0, s1. לפני כל קריאה רקורסיבית נבדוק האם תנאי העצירה מתקיים, ואם הוא מתקיים נוסיף לרגיסטר ה"תוצאה" v0 את הערך 1. כך יחושב אט אט ערך הבינום. הקריאות מתבצעות לפי זהות פסקל - $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$.
בסיום הריצה אנו טוענים חזרה מהמחסנית את הרגיסטרים ששמרנו ומחזירים את ערך ההחזרה.
- circle: תוכנית המציירת על המסך עיגול מלא בצבע לבן במרכז המסך. התוכנית עוברת על כל הפיקסלים ומחליטה לפי משוואת העיגול האם לצבוע את הפיקסל. אם הפיקסל מקיים את המשוואה (לפי הרדיוס הנתון) אז אנו צובעים את הפיקסל, ואם לא, לא נעשה דבר.
- disktest: תוכנית שמבצעת הזזה של תוכן 8 הסקטורים הראשונים בדיסק סקטור אחד קדימה. התוכנית עובדת באמצעות שמירת אינדקסים ופעולות כתיבה וקריאה לדיסק הקשיח שעוברות דרך הזיכרון, שמובילות לשינוי תכני הסקטורים לערכים הרצויים. הפעולות נעשות מהאינדקס הגבוה ביותר (8) לנמוך.