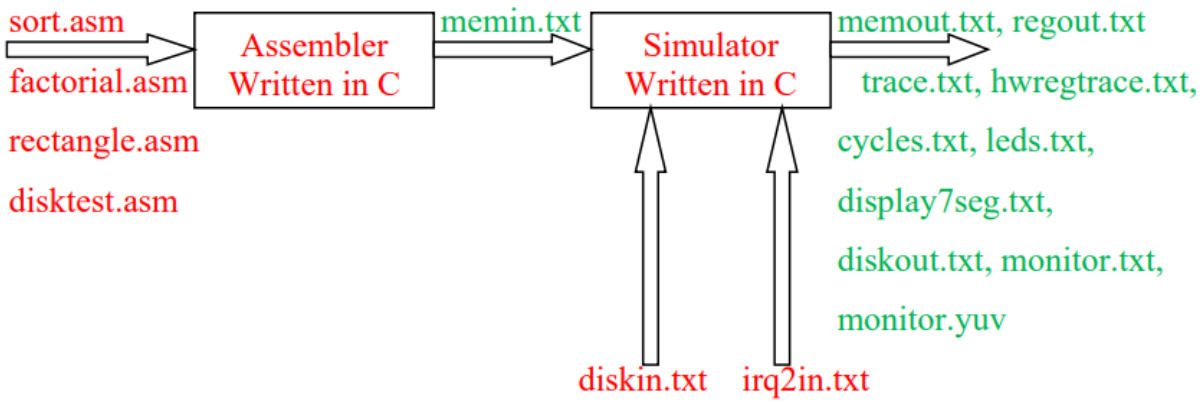
פרויקט SIMP Processor

עידו יוספסברג 322641135, אופק ברוך 208504266, נועה כתר 318875770



איור 1: דיאגרמת בלוקים של הפרויקט. באדום החלקים אותם מימשנו בעצמנו ובירוק קבצים שנוצרו אוטומטית ע"י תוכנות האסמבלר והסימולטור.

בפרויקט נדרשנו לממש אסמבלר, סימולטור וקבצי בדיקה הכתובים בשפת אסמבלי עבור מעבד RISC בשם SIMP.
להלן הדיאגרמה שניתנה בקובץ הוראות הפרויקט הממחישה את מבנה הפרויקט, כעת נרחיב:

סימולטור:

הסימולטור אחראי לבצע סימולציה של לולאת ה- fetch-decode-execute().
הוא מקבל 13 command line parameters לפי שורת ההרצה הבאה:

sim.exe memin.txt diskin.txt irq2in.txt memout.txt regout.txt trace.txt hwregtrace.txt cycles.txt leds.txt display7seg.txt diskout.txt monitor.txt monitor.yuv

בפועל, פעולת הסימולטור מורכבת מטעינת מידע המתקבל מהקבצים memin.txt, diskin.txt, irq2in.txt ומבצע את הinstructions בהתאם.

את הסימולטור מימשנו בעזרת מספר קבצים שיצרנו אשר מאגדים פונקציות הרלוונטיות לכל פונקציונליות הנדרשת מהסימולטור. להלן פירוט הקבצים הקיימים ואופי הפונקציות הנכללות בתוכם.

C files	Header files	פירוט
main.c	-	קורא את ערכי הcommand line parameters ואחראי על קריאה לפונקציות האחריות על fetch, decode והפונקציות שכותבות את המידע לקבצים הרלוונטים
files_handler.c	files_handler.h	קובץ המאגד פונקציות עזר שאחריות על פתיחה, קריאה, וסגירה של קבצים
write_helpers.c	write_helpers.h	פונקציות עזר שאחריות על כתיבה של מבני נתונים מסוגים שונים לתוך קבצי טקסט.
fetch_n_decode.c	fetch_n_decode.h	כולל 3 פונקציות מרכזיות בעבודת הסימולטור. פונקציית הfetch שאחראית על הבאת ההוראה הנדרשת לביצוע בסימולטור מהזיכרון, פונקציית הdecode שאחראית על פיענוח ההוראה

		וביצועה ופונקציית fetch_n_decode_loop שעוטפת את שתי הפונקציות הנ"ל ומאפשרת את ביצוען בסדר הנכון.
dynamic_mem.c	dynamic_str.h	כולל פונקציות עזר שאחראיות לאתחל את המקום בזיכרון עבור על הפונקציות בסימולטור שכמות המידע בהן מתעדכנת. בנוסף כולל פונקציות שבודקות האם בכל אתחול של מידע יש להגדיל את נפח הזיכרון, אם ניתן הן מבצעות אלוקציה למידע.
isa_func.c	isa_func.h	קובץ הכולל מימושים של ההוראות בהן המעבד תומך
simulator.c	simulator.h	קובץ שמאתחל את מבנה הנתונים Simulator המאגד את השדות הרלוונטים בכל מאוחסן המידע הדרוש לריצת הסימולטור. בנוסף כולל פונקציות שמאתחלות את מבנה הנתונים בהתאם להוראות המתקבלות ומשחררות את תוכנו בסיום הריצה.
disk.c	disk.h	כולל את הפונקציה שאחראית לכתוב את תוכן הדיסק בסיום הריצה לקובץ diskout.txt.
memory.c	memory.h	כולל בתוכו את כל הפונקציות הקשורות לזיכרון. אתחול הזיכרון וקריאה של הקבצים הרלוונטיים לו, עדכונו בהתאם להוראות וכתיבתו בסיום הריצה לקובץ הפלט memout.txt.
register.c	register.h	כולל בתוכו את כל הפונקציות הקשורות לרגיסטרים. אתחול הרגיסטרים ועדכונם בהתאם להוראות וכתיבתם תוכנם בסיום הריצה לקובץ הפלט regout.txt.
io_registers.c	io_registers.h	כולל פונקציות התומכות ברגיסטרי הקלט/פלט. בדומה למימוש עבור הרגיסטרים הנ"ל.
trace_handler.c	trace_handler.h	כולל בתוכו את כל הפונקציות הקשורות לtrace. אתחול ועדכון הtrace בהתאם להוראות וכתיבת תוכנו בסיום הריצה כותב את תוכן הtrace לקובץ הפלט trace.txt.
hwregtrace_handler.c	hwregtrace_handler.h	כולל בתוכו את כל הפונקציות הקשורות לְ-hwregtrace . אתחול ועדכון בהתאם להוראות לבסוף כתיבת תוכן ה-hwregtrace בסיום לקובץ הפלט hwregtrace.txt.
cycles.c	cycles.h	כולל את הפונקציה שאחראית לכתוב את כמות מחזורי השעון שרצה התכנית לקובץ cycle.txt
leds_handler.c	leds_handler.h	כולל בתוכו את כל הפונקציות הקשורות לנורות הלד. אתחול המידע ועדכונו בהתאם להוראות לבסוף כתיבת המידע לקובץ הפלט leds.txt.
seg7display_handler.c	seg7display_handler.h	כולל בתוכו את כל הפונקציות הקשורות לְ-7-segment display. אתחול ועדכון בהתאם להוראות ובסיום כתיבת המידע לקובץ הפלט display7seg.txt.
monitor.c	monitor.h	כולל את הפונקציה שאחראית לכתוב את ערכי הפיקסלים שבמסך בסיום הריצה לקובץ monitor.txt וְ-monitor.yuv
-	macros.h	כולל את כל הגדלים בהם נעשה שימוש במימוש הסימולטור נלקחו מהוראות הפרויקט ומומשו בקוד באמצעות פקודת #define.

<u>אופן ריצת הקוד:</u>

1. טעינת ניתובי קבצי הפלט והקלט במבנה נתונים output_path וְ-input_path.
2. אתחול מבנה נתונים מסוג סימולטור ששדותיו מכילים את המידע הדרוש להרצת כל הפעולות הנדרשות לביצוע ע"י הסימולטור.
3. קריאה לפונקצית fetch_n_decode שאחראית על הדברים הבאים:
   i. בנוסף בודקת האם ההוראה מקודדת בתא בודד או בשני תאים. אם ההוראה מקודדת בשני תאים (bigimm==1). היא גם בודקת האם יש פסיקה פעילה ואם צריך לעדכן את ה– irqhandler.
   מעבר לטיפול בפסיקות, אם bigimm==1 מתקיים גם:
   - קוראת לפונקציית הfetch שאחראית על הבאת ההוראה הנדרשת לביצוע בסימולטור מהזיכרון
   - קוראת לפונקציית הdecode שאחראית על פיענוח ההוראה שיובאה וביצועה שלה בהתאם להגדרות הפרויקט.

   אם ההוראה מקודדת בתא בודד (bigimm==0), הפונקציה fetch_n_decode קוראת לפונקציה update_all_traces_n_execute שמאתחלת את התאים ברלוונטים בשדה הsimulator ומבצעת את הפעולה הנדרשת.
   ii. לבסוף מוודאת כי לא התקבלה פסיקה, מעדכנת את הטיימר ואת זמינות הדיסק.
4. בסיום ריצת ה_fetch_n המידע הרלוונטי נכתב לקובץ מתאים לו (בהתאם להגדרות הפרויקט).
5. שחרור הזיכרון המוחזק במבנה הנתונים Simulator.

נקודות חשובות:

- כל הגדלים בהם נעשה שימוש במימוש הסימולטור נלקחו מהוראות הפרויקט ומומשו בקוד באמצעות פקודת #define. כל הגדלים שהוגדרו מפורטים בקובץ macros.h.
- נעשה שימוש במבנים המייצגים אלמנטים בהם נעשה שימוש במהלך הריצה. כגון Simulator (מוצהר בקובץ simulator.h) output_paths וְ- input_pathes (מוצהרים בקובץ files_handler.h).
- הפונקציה המרכזית היא פונקציית main שמופיעה בקובץ main.c ושאר הפונקציות בקוד הן פונקציות עזר שמחולקות לקבצים מתאימים על מנת לקבל קוד קריא ומודולרי.

אסמבלר:

האסמבלר בפרויקט כתבנו בשפת C ותפקידו לתרגם את קוד האסמבלי של מעבד הSIMP לשפת מכונה. לטובת כתיבת האסמבלר ביצענו בעזרת ספריות בסיסיות של C תוך הגדרת גדלים קבועים שהוגדרו בהוראות הפרויקט בעזרת #define (לדוגמה גדלים מקסימליים של הזיכרון, אורך Label וכד').

האסמבלר מקבל 3 command line parameters לפי שורת ההרצה הבאה:

asm.exe program.asm memin.txt

כאשר program.asm מייצג את כל אחת מתוכניות הבדיקה האפשריות הנכתבות בשפת אסמבלי, נרחיב עליהן בהמשך.

את האסמבלר מימשנו בעזרת מספר קבצים שיצרנו אשר מאגדים פונקציות הרלוונטיות הנדרשות לטובת אופן פעולת האסמבלר. להלן פירוט הקבצים הקיימים ואופי הפונקציות הנכללות בתוכם.

C files	Header files	פירוט
main.c	-	כולל את פונקציית הmain שהאחראית על אופן פעולת הריצה.
assembler.c	assembler.h	כולל את הפונקציות שאחריות על רצף תרגום קוד האסמבלי לשפת מכונה (נרחיב בהמשך באופן ריצת הקוד).
line_handler.c	line_handler.h	כולל פונקציות המסווגות את שורות האסמבלי בהתאם לפעולות הכתובות בהן ומתרגמות את קוד האסמבלי הכתוב בשורה למספר בבסיס הקסדצימלי (בגודל 32 ביט) בהתאם לפורמט ההוראה במעבד SIMP המפורטת בהוראות הפרויקט.

אופן ריצת הקוד:

1. פתיחת קובץ program.asm לקריאה.
2. תרגום קוד האסמלי הכתוב הקובץ לשפת מכונה. אופן התרגום מתחלק לשני שלבים עיקריים.

i. **first_pass – מעבר ראשון על קוד האסמבלי:**

בחלק זה מבצעים מעבר ראשון על הקוד במהלכו עוברים שורה שורה על קוד האסמבלי ומחלצים מהכתוב בה את הפעולה (opcode), הרגיסטרים ואת ערך ה-imm (בין אם מדובר בקריאה לLabel ובין אם מדובר בערך מספרי) ומעדכנים את סוג השורה. את הנתונים שחילצנו מהשורה נכניס למבנה נתונים מסוג line הכולל את המידע הרלוונטי להוראה המבוצעת בשורה לטובת תרגום מהיר. נעיר כי פונקציה זו יודעת להתמודד עם מספרים דצימליים, והקסדצימליים (באותיות גדולות או קטנות) הרלוונטים לשדה הimm.
בנוסף מוחזק פוינטר למערך מסוג line שמחזיק בכל אלמנט מבנה מסוג line כך שאינדקס המבנה בשורה תואם למיקום ההוראה בקוד האסמבלי.
פעולה נוספת שנעשית עבור כל שורה הינה חיפוש Labels (תוויות). כל תווית נשמרת במבנה ייעודי מסוג Label המכיל שדה של שם התווית ושדה של כתובת התווית, המחושבת לפי כתובת ההוראה הראשונה של אותה תווית, כלומר, השורה אליה עוברים בקריאה לתווית.

גם עבור האלמנטים מסוג Label מוחזק פוינטר למערך מסוג זה, שמתפקד כמילון ומחזיק את ערכי התוויות שנמצאו בקוד האסמבלי וכתובתן.

פונקציונליות נוספת שנתמכת הינה הוראת .word, אם שורת אסמבלי כוללת את הוראה זו בשלב זה בקוד מתבצע עיבוד של השורה ושמירה של השדות הרלוונטים לה במבנה הline המוקצה לה.

ii. **second_pass – מעבר שני על הקוד:**

בחלק זה מאתחלים מערך של memin מסוג uint32_t שכל ארגומנט בו מייצג שורה שנכתבת לקובץ הפלט memin.txt.

בחלק זה עוברים בשנית על הקוד שורה אחר שורה ע"י מעבר על מערך הline המחזיק את השורות המתורגמות של קוד האסמבלי. בשלב זה עבור כל השורות בהן מתבצעת קריאה לLabel מעדכנים בשדה הimm32 (שיחזיק את הכתובת בשורה עוקבת בזכרון, לפי הגדרת הפרויקט) את הכתובת של התווית.

בנוסף, לאחר עדכון השדות הרלוונטים במבנה בline של השורה מתרגמים את השורה למספר בבסיס הקסדצימלי (בגודל 32 ביט) בהתאם לפורמט הוראה במעבד SIMP המפורטת בהוראות הפרויקט.

3. לבסוף לאחר שמסיימים את המעבר השני על הקוד, כותבים את תוכנו לקובץ טקסט memin.txt שהוא הפלט המתקבל מהאסמבלר.

נקודות חשובות:

- כל הגדלים בהם נעשה שימוש במימוש הסימולטור נלקחו מהוראות הפרויקט ומומשו בקוד באמצעות פקודת #define בקבצי הheader.
- נעשה שימוש במבנים המייצגים אלמנטים בהם נעשה שימוש במהלך הריצה. כגון line, Label וWord (מוצהר בקובץ line_handler.h).
- הפונקציה המרכזית היא פונקציית main שמופיעה בקובץ main.c ושאר הפונקציות בקוד הן פונקציות עזר שמחולקות לקבצים מתאימים על מנת לקבל קוד קריא ומודולרי.

<u>תכניות הבדיקה:</u>

כתבנו ארבע תוכניות אסמבלי לבדיקה:

1. **sort:**

תוכנית זו מממשת את אלגוריתם bubble-sort ומבצעת מיון in-place של 16 מספרים הנתונים בכתובות מוגדרות בזיכרון (0x100 to 0x10f). התכנית כתובה בהתאם לאלגוריתם: מבצעת 2 לולאות מכוננות לטובת השוואה - כל מספר מושווה לכל אחד מהמספרים שעוד לא מויינו, ואם נמצא שהם בסדר הפוך - מבוצעת החלפה בינהם. התכנית מממשת "אתחול" ושחזור" עבור רגיסטרים בהם נעשה שימוש.

2. **factorial:**

תוכנית זו מחשבת עצרת של ארגומנט קלט n (אשר נקרא מתא 0x100 בזיכרון) בעזרת מימוש רקורסיבי. בכל קריאה לפונ' התכנית שומרת מצב התחלתי למחסנית: את n כפי ששמור ב a0 ואת ra, כעת בודקת תנאי עצירה n==0 וממשיכה בהתאם לאלגוריתם:

*אם n!=0: מעדכנת n-=1, וקוראת שוב ל factorial.

*אם n=0: מחזירה 1 (כותבת ל ra), משחזרת את הזיכרון מהמחסנית, משחררת אותו וחוזרת לקורא.

לאחר החזרה - התכנית כופלת את הערך שחזר עם n הנוכחי.

3. **rectangle:**

תוכנית זו מממשת צביעה למסך של מלבן מלא בצבע לבן. כאשר הכתובות של קודקודים A,B,C,D נתונים בזיכרון. A הוא שמאלי עליון ו C הוא ימני תחתון. הבחנה: פיקסל P נמצא על/בתוך המלבן אמ"מ הוא מימין ל A וגם משמאל ל C וגם מעל C וגם מתחת ל A. בעזרת הכתובות הנתונות התכנית מחשבת את הערכים הנ"ל Y(C, (X(C, (Y(A ,(X(A) ולאחר מכן התכנית עוברת בצורה איטרטיבית על כל כתובות הפיקסלים במסך - שורה אחר שורה ומחלצת מכל כתובת את הערכים Y(P,(X(P) שלו. כעת בודקת את כל התנאים הנ"ל ברצף - אם אחד התנאים לא מתקיים אזי הפיקסל לא במסך והוא נצבע בשחור. אחרת כל התנאים מתקיימים ונצבע אותו בלבן. את הצביעה התכנית מבצעת ע"י כתיבה לרגיסטרי ה IO בעזרת .out

4. **disktest:**

תוכנית זו מבצעת סכימה של תוכן הסקטורים 0 עד 3 בדיסק בקשיח וכותבת את תוצאת הסכום לסקטור מספר 4. הסכימה מתבצעת עבור כל מילה בסקטור. כלומר בסיום הריצה כל מילה בסקטור מספר 4, קרי המילה ה-i, תהיה שווה לסכום 4 המילים המתאימות (המילה ה-i בכל סקטור) מסקטורים 0 עד 3.

בנוסף קוד אסמבלי זה תומך בפסיקה מסוג 1.