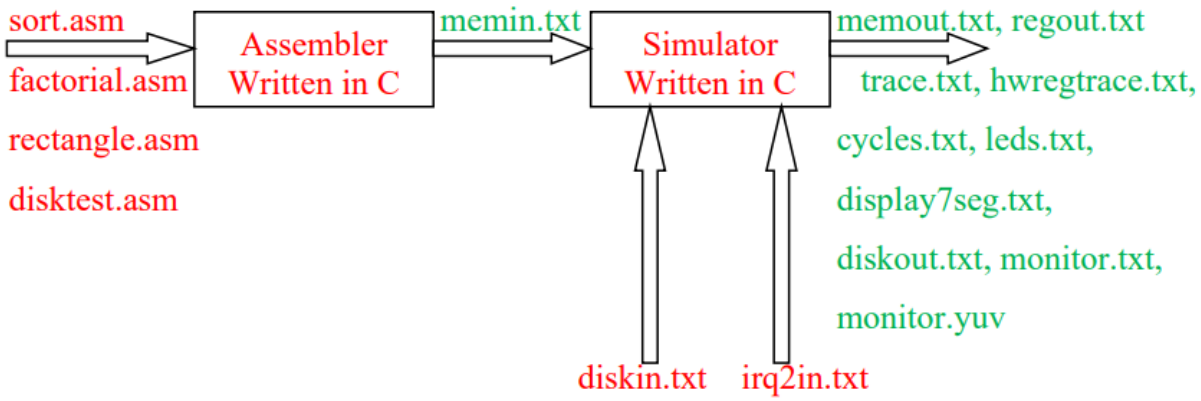
פרויקט SIMP Processor

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:

simulator.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 האחראיות על fetch, decode והפונקציות שכותבות את המידע לקבצים הרלוונטיים.
files_handler.c	files_handler.h	קובץ המאגד פונקציות עזר שאחראיות על פתיחה, קריאה וסגירה של קבצים.
write_helpers.c	write_helpers.h	פונקציות עזר שאחראיות על כתיבה של מבני נתונים מסוגים שונים לתוך קבצי טקסט.
simulator.c	simulator.h	קובץ שמאתחל את מבנה הנתונים Simulator המאגד את השדות הרלוונטיים והמידע הדרוש לריצת הסימולטור. בנוסף כולל פונקציות שמאתחלות את מבנה הנתונים בהתאם להוראות המתקבלות ומשחררות את תוכנו בסיום הריצה.

fetch_n_decode.c	fetch_n_decode.h	כולל את פונקציית הfetch שאחראית על הבאת ההוראה הנדרשת לביצוע בסימולטור מהזיכרון, פונקציית הdecode שאחראית על פיענוח ההוראה וביצועה ופונקציית הfetch_n_decode_loop שעוטפת את שתי הפונקציות הנ"ל ומאפשרת את ביצוען בסדר הנכון. כמו כן, הקובץ כולל את הטיפול בפסיקות והעברת ה PC לכתובת הנכונה במידת הצורך.
dynamic_mem.c	dynamic_str.h	מגדיר struct המכיל מצביע לזיכרון דינמי ומספר המייצג את גודל הזיכרון שהוקצה בבתים. בנוסף, כולל פונקציות עזר אשר תומכות בניהול הזיכרון שהוקצה.
isa_func.c	isa_func.h	קובץ הכולל מימושים של ההוראות בהן המעבד תומך.
disk.c	disk.h	כולל את הפונקציה שאחראית לכתוב את תוכן הדיסק בסיום הריצה לקובץ diskout.txt.
memory.c	memory.h	כולל בתוכו את הפונקציות הקשורות לזיכרון. קריאה וכתיבה של הזיכרון וכתיבתו בסיום הריצה לקובץ הפלט memout.txt.
register.c	register.h	כולל בתוכו את כל הפונקציות הקשורות לרגיסטרים. קריאה של הרגיסטרים, כתיבה אל הרגיסטרים וכתיבת 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 לקובץ ה - trace. אתחול ועדכון של התוכן אשר הקובץ צריך להכיל בהתאם להוראות וכתיבתו בסיום הריצה לקובץ הפלט trace.txt.
hwregtrace_handler.c	hwregtrace_handler.h	כולל בתוכו את כל הפונקציות הקשורות לקובץ ה - hwregtrace. אתחול ועדכון של התוכן אשר הקובץ צריך להכיל בהתאם להוראות וכתיבתו בסיום הריצה 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 לקובץ ה - leds. אתחול ועדכון של התוכן אשר הקובץ צריך להכיל בהתאם להוראות וכתיבתו בסיום הריצה לקובץ הפלט leds.txt.
seg7display_handler.c	seg7display_handler.h	כולל בתוכו את כל הפונקציות הקשורות לקובץ ה - display7seg. אתחול ועדכון של התוכן אשר הקובץ צריך להכיל בהתאם להוראות וכתיבתו בסיום הריצה 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. בפונקציה זו אנו ממשים את לולאת ה - fetch and decode. אנו ממשיכים בביצוע הלולאה כל עוד הסימולטור רץ (כל עוד לא התקבל פקודת halt). בתוך הלולאה אנו בודקים האם אנו במחזור השני של פקודה (מסוג bigimm). אם לא, אנו בודקים אם התקבלה פסיקה ואם אכן התקבלה שומרים את הPC הנוכחי ומעדכנים ל PC הנדרש. לאחר מכן, אנו מפענחים את הפקודה ששמורה ב PC הנוכחי. אם הפקודה שהתקבלה מקודדת בשורה אחת (bigimm=0) אנו מבצעים את הפקודה ומעדכנים את קבצי הפלט הרלוונטיים. אחרת, אנו ממשיכים למחזור הבא, בו אנו מחלצים את ערך הקבוע ולאחר מכן מבצעים את הפקודה ומעדכנים את הקבצים הרלוונטיים.
   בסוף כל איטרציה אנו מקדמים את ה PC במידת הצורך (אם לא התבצעה פקודת קפיצה), מעדכנים את מצב הפסיקות ומעדכנים את ספירת המחזורים.
4. בסיום ריצת ה fetch_n_decode 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:

את האסמבלר בפרויקט כתבנו בשפת C ותפקידו לתרגם את קוד האסמבלי של מעבד הSIMP לשפת מכונה. לטובת כתיבת האסמבלר נעזרנו בספריות 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:

assembler.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 שמימשנו הנדרשות לריצה תקינה של האסמבלר. להלן פירוט הקבצים הקיימים ואופי הפונקציות הנכללות בתוכם.

C files	Header files	פירוט
main.c	-	כולל את פונקציית הmain שאחראית 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 (בגודל 32 ביט) בבסיס הקסדצימלי, בהתאם לפורמט ההוראה במעבד SIMP המפורטת בהוראות הפרויקט.

אופן ריצת הקוד:

1. פתיחת קובץ program.asm לקריאה.
2. אתחול מערך מסוג line בגודל הזיכרון המקסימלי (4096 שורות). כל אלמנט במערך הינו מסוג מבנה הנתונים line הכולל את המידע הרלוונטי להוראה המבוצעת בשורה לטובת תרגום מהיר (מבנה הנתונים מסוג line מפורט בקובץ line_handler.h).
3. אתחול מערך מסוג Label בגודל הזיכרון המקסימלי. כל אלמנט במערך הינו מסוג מבנה נתונים Label, הכולל את שם התווית שנמצאה בקוד האסמבלי ואת הכתובת שלה בזיכרון, נרחיב על אופן השימוש בו בהמשך.
4. תרגום קוד האסמלי הכתוב בקובץ 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 הנתונים הבאים: הפעולה (opcode), הרגיסטרים (rd, rs, rt) ואת ערך ה-imm (בין אם מדובר בקריאה לLabel ובין אם מדובר בערך מספרי). את הנתונים שחילצנו מהשורה נכניס למבנה נתונים מסוג line שנמצא שמוחזק באלמנט המתאים של המערך מסוג line המוזכר לעיל. האינדקס של מבנה הline אותו מעדכנים במערך תואם למיקום ההוראה בקוד האסמבלי. נעיר כי פונקציה זו יודעת להתמודד עם מספרים דצימליים והקסדצימליים (באותיות גדולות או קטנות) הרלוונטיים לשדה הimm.

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. בסיום המעבר הראשון מערך בLabel יכיל את כל התוויות האפשריות וכתובותיהן לטובת עדכון הכתובת בשורות המתאימות במעבר השני על הקוד.
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 ה lineהמוקצה לה, בשדה word מסוג Word הכולל את הdata והכתובת המפורטים בהוראה בהתאם לפורמט המצוין בהוראות הפרויקט.

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

5. 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 מקוננות לטובת השוואה - כל מספר מושווה לכל אחד מהמספרים שעוד לא מוינו, 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 מכתובת 0x100 בזיכרון) בעזרת מימוש רקורסיבי. בכל קריאה לפונקציה, התכנית שומרת מצב התחלתי למחסנית: את n כפי ששמור בְּ- a0 ואת ra, בודקת אם תנאי עצירה n == 0 מתקיים וממשיכה בהתאם לאלגוריתם:

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

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

לאחר ההחזרה - 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 (כולל היקף המלבן). בעזרת הכתובות הנתונות, התכנית מחשבת את הערכים הנ"ל (ערכי X וְ-Y של הקודקוד A וערכי X וְ-Y של הקודקוד C) ולאחר מכן עוברת בצורה איטרטיבית על כל כתובות הפיקסלים במסך בלולאה מקוננת – עבור כל שורה היא עוברת על כל העמודות שנמצאות בטווח ערכי X שחולצו מהקודקודים וצובעת את הפיקסלים בלבן. את הצביעה התכנית מבצעת ע"י כתיבה לרגיסטרי ה IO בעזרת פקודת ה- out.

4. **disktest:**

תוכנית זו מבצעת סכימה של תוכן הסקטורים 0 עד 3 בדיסק הקשיח וכותבת את תוצאת הסכום לסקטור מספר 4. הסכימה מתבצעת עבור כל מילה בסקטור. כלומר, בסיום הריצה, כל מילה בסקטור מספר 4, תהיה שווה לסכום 4 המילים המתאימות מסקטורים 0 עד 3.