פרויקט ISA, קורס מבנה המחשב

מגישים:
עידו מסטאי
ירין קורן
יוחי שילה

רקע קצר – מבנה המעבד ושפת האסמבלי

בפרויקט אנו נדרשים לדמות את עבודתו של מעבד בשם SIMP הדומה למעבד MIPS ולהריץ תוכנות הכתובות בשפת האסמבלי הייחודית שלו. מעבד זה הוא בעל זיכרון (dmemin.txt) ומחובר לזיכרון חיצוני (disk drive).
בשפה זו ישנן 22 פקודות אסמבלי המוצגות להלן, כאשר בקובץ הזיכרון(imemin.txt) כל פקודה רשומה כשורה הקסאדצימלית ברוחב 12 תווים (כ- 48 ביטים). הפקודה בנויה ע"פ הצורה הבאה,מימין לשמאל:

47:40	39:36	35:32	31:28	27:24	23:12	11:0
opcode	rd	rs	rt	rm	immediate1	immediate2

תווים 1-3 – שדה ה-Immediate2 שיוכנס לתוך אוגר ה-Immediate2 בביצוע הפעולה
תווים 4-6 – שדה ה-Immediate1 שיוכנס לתוך אוגר ה-Immediate1 בביצוע הפעולה
תו 7 – אוגר rm
תו 8– אוגר rt
תו 9- אוגר rs
תו 10- אוגר rd
תווים 11-12–opcode (שם הפקודה)

להלן טבלה עם כל הפקודות שתוכנות האסמבלר והסימולטור תומכות בהן:

מספר opcode	שם בשפת אסמבלי	משמעות
0	Add	Rrd=R[rs]+R[rt]+ R[rm]
1	Sub	R[rd]=R[rs]-R[rt]- R[rm]
2	mac	R[rd]=R[rs]*R[rt]+ R[rm]
3	And	R[rd]=R[rs]&R[rt]& R[rm]
4	or	Rrd=R[rs]|R[rt]| R[rm]
5	xor	R[rd]=R[rs]^R[rt]^ R[rm]
6	Sll	Rrd=Rrs≪R[rt]
7	Sra	הזזה אריתמטית עם הארכת סימן R[rd]=R[rs]≫R[rt]
8	Srl	Rrd=Rrs≫R[rt] *הזזה לוגית*
9	beq	if (R[rs]==R[rt]) pc= R[rm] [11:0 ביטים אחרונים]
10	bne	if (R[rs]!=R[rt]) pc= R[rm] [11:0 ביטים אחרונים]
11	blt	if (R[rs]<R[rt]) pc= R[rm] [11:0 ביטים אחרונים]
12	bgt	if (R[rs]>R[rt]) pc= R[rm] [11:0 ביטים אחרונים]
13	Ble	if (R[rs]<=R[rt]) pc= R[rm] [11:0 ביטים אחרונים]
14	bge	if (R[rs]>=R[rt]) pc= R[rm] [11:0 ביטים אחרונים]
15	jal	R[rd]=pc+1, pc= R[rm] [11:0 ביטים אחרונים]

16	lw	R[rd]=MEM[R[rs]+R[rt]]+ R[rm]
17	sw	MEM[R[rs]+R[rt]= R[rm]+R[rd]
18	reti	pc=IORegister[7]
19	in	R[rd]=IORegister [R[rs]+R[rt]]
20	out	IORegister [R[rs]+R[rt]]= R[rm]
21	halt	פקודת יציאה מהסימולטור

כאשר בכל פקודות ה-branch ההתניה נעשית רק על 11 הביטים הימניים ביותר של האוגרים, 'PC' (עבור program counter) פירושו מספר השורה הנוכחית ו- 'mem' היא הכתובת של שורה בזיכרון.
פקודת אסמבלי אחרונה היא הפקודה ".word" פקודה זו מאחסנת מילה של 32 ביטים(8 תווים הקסאדצימלים) ישירות בזיכרון, שבנויה באופן הבא: word {address} {data}.
בכלל, הוראות בקבצי אסמבלי מתחילות בפקודה ע"ב רשימת הפקודות למעלה, לפעמים בתוספת הערות והסברים לפעולת הפקודה שיופיעו לאחר תו ייחודי – '#'.

למעבד 16 אוגרים עם תפקידים שונים:

מספר הרגיסטר	שם	תפקיד
0	$zero	תמיד מכיל אפס
1	$imm1	הפניה לשדה ה-immediate1 של הפקודה
2	$imm2	הפניה לשדה ה-immediate2 של הפקודה
3	$v0	ערך מוחזר מפונקציה
4	$a0	ארגומנט של פונקציה
5	$a1	ארגומנט של פונקציה
6	$a2	משתנה זמני
7	$t0	משתנה זמני
8	$t1	משתנה זמני
9	$t2	משתנה זמני
10	$s0	משתנה זמני נשמר
11	$s1	משתנה זמני נשמר
12	$s2	משתנה זמני נשמר
13	$gp	Global pointer(static data)
14	$sp	מצביע למחסנית
15	$ra	כתובת חזרה לאחר שימוש בפונקציה

מכיוון שהסימולטור תומך בעבודה עם התקנים חיצוניים (דיסק, מוניטור וכו'), יש לנו את הפקודות הייעודיות (in, out, reti) לעבודה איתם. פקודות אלו פונות לסט רגיסטרים נוסף אשר בעזרתם אנו עובדים עם ההתקנים החיצוניים. רגיסטרים אלו הם בעלי גדלים משתנים ולא גודל קבוע כמו הרגיסטרים הרגילים של המעבד, והם מפורטים בטבלה הבאה:

IORegiter number	שם הרגיסטר	מספר הביטים	משמעות
0	irq0enable	1	IRQ0 הוא פעיל אם מוגדר ל-1, אחרת לא פעיל
1	irq1enable	1	IRQ1 הוא פעיל אם מוגדר ל-1, אחרת לא פעיל
2	irq2enable	1	IRQ2 הוא פעיל אם מוגדר ל-1, אחרת לא פעיל

3	irq0status	1	הסטטוס של - IRQ0 מאותחל ל-1 כאשר irq0 מעורר
4	irq1status	1	הסטטוס של – IRQ1 מאותחל ל-1 כאשר irq1 מעורר
5	irq2status	1	הסטטוס של – IRQ2 מאותחל ל-1 כאשר irq2 מעורר
6	irqhandler	12	מכיל את הכתובת עבור ה-handler
7	irqreturn	12	מכיל את הכתובת חזרה אחרי ההפרעה
8	clks	32	סופר מחזורי שעון. מתחיל מ-0 ועולה ב-1 כל מחזור שעון, כאשר מגיע לערך מירבי (0Xffffffff) הוא חוזר בחזרה ל-0.
9	leds	32	מחובר ל-32 לדים ולכל מספר לד i הלד דולק כאשר leds[i]==1 אחרת הוא כבוי.
10	display7seg	32	מחובר ל 7-segment display אשר מורכב מ-8 אותיות. כל ארבעה ביטים מייצגים ספרה אחת מ F-0 כאשר ארבעת הביטים הימניים (0:3) מיצגים את האות הימנית והשמאליים (28:31) מייצגים את השמאלית ביותר.
11	timerenable	1	כאשר-1 הטיימר פועל. כאשר -0 הטיימר כבוי.
12	timercurrent	32	הערך בהווה של הקאונטר.
13	timermax	32	הערך המקסימלי אליו הקאונטר יכול להגיע.
14	diskcmd	2	0= אין פקודה. 1= קריאת סקטור. 2= כתיבת סקטור.
15	disksector	7	מספר הסקטור בוא נבצע פעולה, מתחיל מ-0.
16	diskbuffer	12	מצביע לראש המערך (128 מילים) בזיכרון, שאליו כותבים את הסקטור/ שאותו מעתיקים לסקטור בדיסק.
17	diskstatus	1	0= פנוי לקבל פקודה חדשה. 1= עסוק בפעולת קריאה/ כתיבה.
18	reserved	-	שמור לשימוש עתידי
19	reserved	-	שמור לשימוש עתידי
20	monitoraddr	16	כתובת הפיקסל ב- frame buffer.
21	monitordata	8	ערך צבע הפיקסל(בגווני אפור) מ 0-255
22	monitorcmd	1	0= אין הוראה. 1= לכתוב פיקסל למוניטור.

הרגיסטרים של irq0, irq1, irq2 מאפשרים לנו לעבוד בפסיקות אל מול ההתקנים החיצוניים כך ש- irq0 משויך לטיימר, irq1 משויך לדיסק הקשיח ו-irq2 מחובר אל קו חיצוני למעבד ולמעשה מופעל ע"י קובץ קלט אשר אנו מזינים וקובע מתי מתבצעת הפרעה.
במידה ומתבצעת הפרעה כך ש- irq status של אחד הרכיבים עולה ל-1 ובנוסף ה- irq enabale שלו פעיל על -1 אזי המעבד יקפוץ (בעזרת ה-irqhandler) אל שגרת הטיפול אשר הוגדרה לו בקוד ויישאר בה עד אשר יסיים את הטיפול או לחלופין יתחיל אותה מחדש במידה ויקבל הפרעה נוספת.

קבצי האסמבלי

1. mulmat.asm

בקובץ זה כתבנו פקודות לתוכנית אשר תבצע כפל בין שתי מטריצות אשר הגדרנו באופן אקראי בתחילת הקובץ ותוצאת הכפל תישמר אל הזיכרון גם.
את matrix1 שמרנו בכתובות 256-271 בזיכרון, את matrix2 שמרנו בכתובות 272-287 בזיכרון ולבסוף את מטריצת התוצאה אחסנו בכתובות 288-303.
המטריצות שהגדרנו הן:

$$\underset{\text{Matrix1}}{\begin{pmatrix} 7 & 5 & 8 & 2 \\ 1 & 9 & 6 & 10 \\ 3 & 4 & 8 & 9 \\ 10 & 5 & 6 & 7 \end{pmatrix}} * \underset{\text{matrix2}}{\begin{pmatrix} 4 & 2 & 1 & 3 \\ 7 & 6 & 1 & 9 \\ 4 & 2 & 10 & 3 \\ 6 & 8 & 5 & 4 \end{pmatrix}} = \underset{\text{solution}}{\begin{pmatrix} 107 & 69 & 109 & 98 \\ 151 & 147 & 121 & 142 \\ 126 & 115 & 135 & 105 \\ 141 & 108 & 120 & 121 \end{pmatrix}}$$

2. binom.txt

בקובץ זה כתבנו פקודות כך שהמערכת תבצע חישוב של הבינום- binom (n,k) לפי הקוד הבא(בשפת C):

```
int binom(n, k)
 {
if (k == 0 || n == k)
 return 1;
 return binom(n-1, k-1) + binom(n-1, k)
}
```

כאשר אנו בחרנו לחשב binom(5,3) אשר תוצאתו היא 10.
הקוד הוא רקורסיבי כפי שנתבקשנו, הערכים איתם הוא עובד (5 ו-3) נשמרו אל הזיכרון (dmemin.txt) בתחילת הריצה והתוצאה נשמרה גם היא אל הזיכרון (dmemout.txt) בסיום.

3. circle.asm

בקובץ זה כתבנו קוד אשר מקבל רדיוס מוגדר מראש בזיכרון (אנחנו הגדרנו בקוד את הרדיוס R=96) ומצייר מעגל במוניטור המחובר למעבד. המעגל יצוייר במרכז המוניטור כך שהפיקסלים בתוך המעגל ייצבעו בלבן וכל הפיקסלים מחוץ למעגל ייצבעו בשחור. המוניטור הוא בגודל של 256X256 והפיקסל השמאלי העליון ביותר מוגדר כ- (0,0) ולכן הגדרנו את מרכז המעגל לפיקסלים (127,127).
הצבעים של הפיקסלים במוניטור הם בגווני אפור (משחור ועד לבן) כך שהצבע השחור מוגדר כ-0 והלבן כ-255. ולכן על מנת לצבוע פיקסל בלבן עלינו להגדיר לרגיסטר mionitordata את הערך 255. מכיוון שבהנחת המוצא כל הפיקסלים במוניטור מוגדרים ל-0 בקוד שכתבנו ניגשנו רק לפיקסלים אשר נמצאים בתוך רדיוס המעגל ושינינו את ערכם ועל שאר הפיקסלים דילגנו.

disktest.asm .4

בקובץ זה כתבנו קוד אשר עובד מול הדיסק החיצוני ומבצע הזזה של סקטור אחד קדימה ל- 8 הסקטורים הראשונים (0-7) , כך שהמידע בהם יאוכסן בסקטורים 1-8 במקום. על מנת לבצע זאת עשינו הזזה מהסוף להתחלה, משמע ראשית הזזנו את סקטור -7 לסקטור-8, אח"כ את 6 ל-7 וכן הלאה. את המידע בסקטורים אתחלנו במידע שרירותי על מנת שלא נזיז קובץ ריק.

האסמבלר

רקע

בפרויקט זה נדרשנו לכתוב תוכנת אסמבלר בשפת C. האסמבלר מקבל קבצי טקסט המכילים קוד בשפת אסמבלי (שהמבנה שלה מפורט למעלה) ומתרגם אותם לקבצים הקסאדצימליים בשפת מכונה, כלומר כאלה שהמעבד יכול לקרוא ולעבד או ולבצע את התוכן שלהם.

האסמבלר עובר פעמיים על הקוד. בפעם ראשונה הוא מאתר בקוד תוויות (Labels) ומיקומם, ובפעם השנייה הוא מתרגם את הקוד ומייצא שני קבצי טקסט: imemin.txt ו-dmemin.txt, עליהם נרחיב מיד. בקבצי טקסט אלו ישתמש הסימולטור להמשך עבודתו.

קבצי הoutput

imemin.txt – קובץ טקסט המכיל את התרגום של כל ההוראות שהתקבלו משפת האסמבלי לשפת מכונה בייצוג הקסאדצימלי, על פי הטבלאות בדפי הפרויקט ושמופיעות למעלה.

dmemin.txt – קובץ המכיל את המידע ההתחלתי של הזיכרון הראשי(ללא זיכרון הדיסק, שיצורף בנפרד) שיועבר כקלט למעבד, כפי שהתקבל מקוד האסמבלי באמצעות הפקודה 'word.'.

אופן פעולת קוד האסמבלר

במעבר הראשון האסמבלר סורק את הקוד שהתקבל, מאתר בו את התוויות ושומר במרוכז, באמצעות רשימה מקושרת, לכל תווית את השם שלה והכתובת בקוד בה היא נמצאת. מכיוון שהתוויות אינן חלק מהפקודות לביצוע אלא רק פסאודו-הוראה (כלומר שורת קוד שאינה מבוצעת ע"י המעבד כהוראה אלא רק נועדה רק להקל על כתיבת הקוד ובהמשך תוחלף בערך מתאים), האסמבלר לא מחשיב אותה כשורת כתובת בפני עצמה והכתובת של התווית תהיה הכתובת של שורת ההוראה שמתחילה אחריה(מסיבה דומה, גם שורות בקוד שמאחסנות מידע בזיכרון, פקודות word., אינן בעלות כתובת בקוד). בהמשך ייעשה שימוש ברשימת התוויות על מנת להחליף בקוד המכונה, שיתקבל במעבר השני, את ההפניות לתוויות השונות בכתובת המתאימה.

במעבר השני על קבצי האסמבלי הקוד מפרק כל הוראה בקוד האסמבלי לחלקים הבאים:
opcode, rd, rs, rt, rm, imm1, imm2 , וכל חלק מתורגם לקוד הקסאדצימלי על פי הטבלאות הנמצאות למעלה ובדפי הפרויקט. לאחר מכן מאוחדים כל החלקים לשורה הוראה אחת בשפת מכונה, בייצוג הקסאדצימלי ומודפסים לקובץ ה-imemin.txt. כאשר האסמבלר מאתר בקוד שורה שמורה על שמירה לזיכרון (כלומר ה-opcode שווה ל-word.), אנו כותבים לקובץ הזיכרון, ה-dmemin.txt, את הערך המתאים במיקום המבוקש בייצוג הקסאדצימלי.

הסימולטור

רקע

בפרויקט זה נתבקשנו לכתוב סימולטור בשפת C אשר ממדל את פעולת מעבד ה-simp, שמתואר בפתיחת המסמך. הסימולטור מקבל את הקבצים imemin.txt ו-dmemin.txt שנוצרו על ידי האסמבלר. בנוסף הוא מקבל שני קבצי קלט נוספים - irq2in.txt ו-diskin.txt עליהם נרחיב בהמשך. הסימולטור מבצע את ההוראות שהתקבלו, שמופיעות בייצוג הקסאדצימלי של הפקודות בשפת המכונה, על פי ההגדרות המפורטות למעלה ובדפי הפרויקט, ומייצא את

הקבצים הבאים כפלט: dmemout.txt, regout.txt, trace.txt, hwregtrace.txt cycles.txt, leds.txt , monitor.yuv, monitor.txt, diskout.txt, display7seg.txt

קבצי input

imemin.txt – קובץ זה נבנה ע"י האסמבלר(ראו פירוט למעלה).
dmemin.txt – קובץ זה נבנה ע"י האסמבלר(ראו פירוט למעלה).
Diskin.txt – מכיל את המצב ההתחלתי של הזיכרון המשני(זיכרון הדיסק, "הזיכרון החיצוני") ב-diskdrive
Irq2in.txt – מכיל את מחזורי השעון אשר בהם אנו מעוניינים לעורר interruption .

קבצי הoutput

dmemout.txt – מכיל את מצב הזיכרון המשני בסוף ריצת הסימולטור.
regout.txt - מכיל את מצב הרגיסטרים 3-15 בסוף ריצת הסימולטור.
trace.txt – מכיל בכל שורה את ערך ה-PC והערכים המספריים שמועברים לשורת הפקודה(באמצעות רגיסטרי ה-immediate), וכן את מצב הרגיסטרים לפני ביצוע ההוראה, עבור כל הוראה שמבוצעת ע"י הסימולטור.
hwregtrace.txt – מכיל שורת טקסט עבור כל כתיבה או קריאה לרגיסטרי החומרה (i/o registers), אשר נעשית בעזרת ההוראות in ו-out. כל שורה מכילה את שם הרגיסטר, איזו פעולה בוצעה (קריאה או כתיבה), והתוכן אשר נקרא או נכתב(בהתאם).
cycles.txt – מכיל את סך מחזורי השעון אשר לקח לסימולטור לרוץ.
leds.txt – מכיל את המצב של כל אחד מ 32 נורות הלד בכל מחזור שעון בו בוצע שינוי של המצב של אחד מהן.
display7seg - מכיל את המצב של ה-display בכל מחזור שעון בו היה שינוי במצבו.
diskout.txt - מכיל את מצב הזיכרון ב-diskdrive בסוף ריצת הסימולטור.
monitor.txt – קובץ טקסט בו כל שורה מכילה את מצבו של פיקסל מסוים, בסוף ריצת הסימולטור.
monitor.yuv – קובץ בינארי המכיל את אותו המידע שמכיל הקובץ monitor.txt. קובץ זה ניתן להציג בצורה גרפית על מסף המחשב בעזרת התוכנה yuvplayer.

אופן פעולת הקוד של הסימולטור

בשלב ראשון הסימולטור קולט את קבצי הקלט השונים, קורא את התוכן שלהם ומאחסן אותו במרוכז במערכים מתאימים(באמצעות מערכי מידע מטיפוסים שונים, או רשימה מקושרת במקרה של 'irq2in'), עבור כל אחד מהקבצים. לאחר מכן הסימולטור עובר לחלק המרזי של פעולתו – לולאת ביצוע ההוראות בשפת המכונה, כפי שתקבלו, עד לסיום ריצת הקוד(כאשר יגיע תור הפקודה halt להתבצע). לאחר כתיבה לקובץ ה-trace את התוכן הדרוש(ואגב כך ווידוא איפוס הרגיסטר השמור zero), הסימולטור מאתר סוג ההוראה לביצוע(ה-opcode), מבצע אותה בהתאם למאפיינים שלה(שמוגדרים בדפי הפרויקט ומפורטים למעלה) ובהתאם לערכים שהועברו לה בקוד המכונה, ואם נדרש מדמה ייצוא להתקני הקלט השונים של המידע הרלוונטי – הדלקת נורות לד או הצגה בד7segment display וכ', ומתעד בקבצי הפלט את הפעולה שהתבצעה. כמו כן, אם נדרש הסימולטור מבצע קריאה או כתיבה לזיכרון או למוניטור, וממשיך להוראה הבאה, לאחר שווידא שאין interruption שדורש המתנה מביצוע פעולות שונות או התקדמות לפקודה הבאה.
לאחר סיום המעבר על הקוד, הסימולטור מתעד את המידע הסופי בקבצי הפלט המתאימים – את הזיכרון הראשי(ב-dmemout), הזיכרון החיצוני(ב-diskout), מספר מחזורי השעון שעברו(ב-cycles), מצב הרגיסטרים(ב-regout) ומצב המוניטור(בני קבצי המוניטור), וסוגר אותם ואת שאר קבצי הפלט.